1

INTRODUCTION TO SYSTEM ANALYSIS AND DESIGN

1.1 INTRODUCTION

Systems are created to solve problems. One can think of the systems approach as an organized way of dealing with a problem. In this dynamic world, the subject System Analysis and Design (SAD), mainly deals with the software development activities.

1.2 OBJECTIVES

After going through this lesson, you should be able to

- define a system
- explain the different phases of system development life cycle
- enumerate the components of system analysis
- explain the components of system designing

1.3 DEFINING A SYSTEM

A collection of components that work together to realize some objectives forms a system. Basically there are three major components in every system, namely input, processing and output.

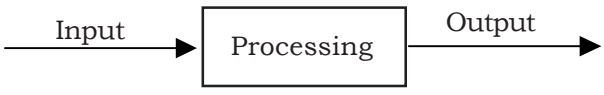


Fig. 1.1: Basic System Components

In a system the different components are connected with each other and they are interdependent. For example, human body represents a complete natural system. We are also bound by many national systems such as political system, economic system, educational system and so forth. The objective of the system demands that some output is produced as a result of processing the suitable inputs. A well-designed system also includes an additional element referred to as 'control' that provides a feedback to achieve desired objectives of the system.

1.4 SYSTEM LIFE CYCLE

System life cycle is an organizational process of developing and maintaining systems. It helps in establishing a system project plan, because it gives overall list of processes and sub-processes required for developing a system.

System development life cycle means combination of various activities. In other words we can say that various activities put together are referred as system development life cycle. In the System Analysis and Design terminology, the system development life cycle also means software development life cycle.

Following are the different phases of system development life cycle:

- Preliminary study
- Feasibility study
- Detailed system study
- System analysis
- System design
- Coding
- Testing
- Implementation
- Maintenance

The different phases of system development life cycle is shown in Fig. 1.2 below.

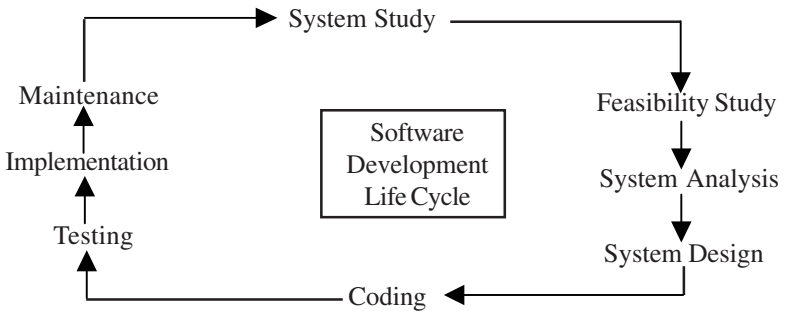


Fig. 1.2: Phases of System Development Life Cycle

INTEXT QUESTIONS

1. Write True or False for the following statements.
   (a) A collection of components that work together to realize some objectives forms a system.
   (b) System life cycle is not an organizational process of developing and maintaining a system.
   (c) In the system analysis and design terminology the system development life cycle means software development life cycle.
   (d) Coding is not a step in system development life cycle.
   (e) System analysis and system design are the same phase of system development life cycle.

1.5 PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE

Let us now describe the different phases and related activities of system development life cycle.

(a) Preliminary System Study

Preliminary system study is the first stage of system development life cycle. This is a brief investigation of the system under consideration and gives a clear picture of what actually the physical system is? In practice, the initial system study involves the preparation of a 'System Proposal' which lists the Problem Definition, Objectives of the Study, Terms of reference for Study, Constraints, Expected benefits of the new system, etc. in the light of the user requirements. The system proposal is prepared by the System Analyst (who studies the system) and places it before the user management. The management may accept the proposal and the cycle proceeds to the next stage. The management may also reject the proposal or request some modifications in the proposal. In summary, we would say that system study phase passes through the following steps:

- problem identification and project initiation
- background analysis
- inference or findings (system proposal)

(b) Feasibility Study

In case the system proposal is acceptable to the management, the

next phase is to examine the feasibility of the system. The feasibility study is basically the test of the proposed system in the light of its workability, meeting user's requirements, effective use of resources and of course, the cost effectiveness. These are categorized as technical, operational, economic and schedule feasibility. The main goal of feasibility study is not to solve the problem but to achieve the scope. In the process of feasibility study, the cost and benefits are estimated with greater accuracy to find the Return on Investment (ROI). This also defines the resources needed to complete the detailed investigation. The result is a feasibility report submitted to the management. This may be accepted or accepted with modifications or rejected. The system cycle proceeds only if the management accepts it.

(c) Detailed System Study

The detailed investigation of the system is carried out in accordance with the objectives of the proposed system. This involves detailed study of various operations performed by a system and their relationships within and outside the system. During this process, data are collected on the available files, decision points and transactions handled by the present system. Interviews, on-site observation and questionnaire are the tools used for detailed system study. Using the following steps it becomes easy to draw the exact boundary of the new system under consideration:

- Keeping in view the problems and new requirements
- Workout the pros and cons including new areas of the system

All the data and the findings must be documented in the form of detailed data flow diagrams (DFDs), data dictionary, logical data structures and miniature specification. The main points to be discussed in this stage are:

- Specification of what the new system is to accomplish based on the user requirements.
- Functional hierarchy showing the functions to be performed by the new system and their relationship with each other.
- Functional network, which are similar to function hierarchy but they highlight the functions which are common to more than one procedure.
- List of attributes of the entities – these are the data items which need to be held about each entity (record)

(d) System Analysis

Systems analysis is a process of collecting factual data, understand the processes involved, identifying problems and recommending feasible suggestions for improving the system functioning. This involves studying the business processes, gathering operational data, understand the information flow, finding out bottlenecks and evolving solutions for overcoming the weaknesses of the system so as to achieve the organizational goals. System Analysis also includes subdividing of complex process involving the entire system, identification of data store and manual processes.

The major objectives of systems analysis are to find answers for each business process: What is being done, How is it being done, Who is doing it, When is he doing it, Why is it being done and How can it be improved? It is more of a thinking process and involves the creative skills of the System Analyst. It attempts to give birth to a new efficient system that satisfies the current needs of the user and has scope for future growth within the organizational constraints. The result of this process is a logical system design. Systems analysis is an iterative process that continues until a preferred and acceptable solution emerges.

(e) System Design

Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. It is the most crucial phase in the developments of a system. The logical system design arrived at as a result of systems analysis is converted into physical system design. Normally, the design proceeds in two stages:

- Preliminary or General Design
- Structured or Detailed Design

Preliminary or General Design: In the preliminary or general design, the features of the new system are specified. The costs of implementing these features and the benefits to be derived are estimated. If the project is still considered to be feasible, we move to the detailed design stage.

Structured or Detailed Design: In the detailed design stage, computer oriented work begins in earnest. At this stage, the design of the system becomes more structured. Structure design is a blue print of a computer system solution to a given problem having the

same components and inter-relationships among the same components as the original problem. Input, output, databases, forms, codification schemes and processing specifications are drawn up in detail. In the design stage, the programming language and the hardware and software platform in which the new system will run are also decided.

There are several tools and techniques used for describing the system design of the system. These tools and techniques are:

- Flowchart
- Data flow diagram (DFD)
- Data dictionary
- Structured English
- Decision table
- Decision tree

Each of the above tools for designing will be discussed in detailed in the next lesson.

The system design involves:

i. Defining precisely the required system output

ii. Determining the data requirement for producing the output

iii. Determining the medium and format of files and databases

iv. Devising processing methods and use of software to produce output

v. Determine the methods of data capture and data input

vi. Designing Input forms

vii. Designing Codification Schemes

viii. Detailed manual procedures

ix. Documenting the Design

(f) Coding

The system design needs to be implemented to make it a workable system. This demands the coding of design into computer understandable language, i.e., programming language. This is also called the programming phase in which the programmer converts the pro-

gram specifications into computer instructions, which we refer to as programs. It is an important stage where the defined procedures are transformed into control specifications by the help of a computer language. The programs coordinate the data movements and control the entire process in a system.

It is generally felt that the programs must be modular in nature. This helps in fast development, maintenance and future changes, if required.

(g) Testing

Before actually implementing the new system into operation, a test run of the system is done for removing the bugs, if any. It is an important phase of a successful system. After codifying the whole programs of the system, a test plan should be developed and run on a given set of test data. The output of the test run should match the expected results. Sometimes, system testing is considered a part of implementation process.

Using the test data following test run are carried out:

- Program test
- System test

Program test: When the programs have been coded, compiled and brought to working conditions, they must be individually tested with the prepared test data. Any undesirable happening must be noted and debugged (error corrections)

System Test: After carrying out the program test for each of the programs of the system and errors removed, then system test is done. At this stage the test is done on actual data. The complete system is executed on the actual data. At each stage of the execution, the results or output of the system is analysed. During the result analysis, it may be found that the outputs are not matching the expected output of the system. In such case, the errors in the particular programs are identified and are fixed and further tested for the expected output.

When it is ensured that the system is running error-free, the users are called with their own actual data so that the system could be shown running as per their requirements.

(h) Implementation

After having the user acceptance of the new system developed, the

implementation phase begins. Implementation is the stage of a project during which theory is turned into practice. The major steps involved in this phase are:

- Acquisition and Installation of Hardware and Software
- Conversion
- User Training
- Documentation

The hardware and the relevant software required for running the system must be made fully operational before implementation. The conversion is also one of the most critical and expensive activities in the system development life cycle. The data from the old system needs to be converted to operate in the new format of the new system. The database needs to be setup with security and recovery procedures fully defined.

During this phase, all the programs of the system are loaded onto the user's computer. After loading the system, training of the user starts. Main topics of such type of training are:

- How to execute the package
- How to enter the data
- How to process the data (processing details)
- How to take out the reports

After the users are trained about the computerized system, working has to shift from manual to computerized working. The process is called 'Changeover'. The following strategies are followed for changeover of the system.

(i) **Direct Changeover:** This is the complete replacement of the old system by the new system. It is a risky approach and requires comprehensive system testing and training.

(ii) **Parallel run:** In parallel run both the systems, i.e., computerized and manual, are executed simultaneously for certain defined period. The same data is processed by both the systems. This strategy is less risky but more expensive because of the following:

- Manual results can be compared with the results of the computerized system.

- The operational work is doubled.
- Failure of the computerized system at the early stage does not affect the working of the organization, because the manual system continues to work, as it used to do.

(iii) **Pilot run:** In this type of run, the new system is run with the data from one or more of the previous periods for the whole or part of the system. The results are compared with the old system results. It is less expensive and risky than parallel run approach. This strategy builds the confidence and the errors are traced easily without affecting the operations.

The documentation of the system is also one of the most important activity in the system development life cycle. This ensures the continuity of the system. There are generally two types of documentation prepared for any system. These are:

- User or Operator Documentation
- System Documentation

The user documentation is a complete description of the system from the users point of view detailing how to use or operate the system. It also includes the major error messages likely to be encountered by the users. The system documentation contains the details of system design, programs, their coding, system flow, data dictionary, process description, etc. This helps to understand the system and permit changes to be made in the existing system to satisfy new user needs.

(i) Maintenance

Maintenance is necessary to eliminate errors in the system during its working life and to tune the system to any variations in its working environments. It has been seen that there are always some errors found in the systems that must be noted and corrected. It also means the review of the system from time to time. The review of the system is done for:

- knowing the full capabilities of the system
- knowing the required changes or the additional requirements
- studying the performance.

If a major change to a system is needed, a new project may have to be set up to carry out the change. The new project will then proceed through all the above life cycle phases.

INTEXT QUESTIONS

2. Fill in the blanks.

   (a) System study is the _____ stage of system development life cycle.

   (b) Analysis involves a _____ study of the current system.

   (c) All procedures requirements must be analysed and documented in the form of data flow diagrams, data dictionary, _____ and miniature specifications.

   (d) _____ is a blue print of a computer system.

   (e) In _____ run the new system installed in parts.

   (f) In parallel run computerized and _____ systems are executed in parallel.

1.6 WHAT YOU HAVE LEARNT

In this lesson a systematic approach to solve any given problem is explained. Phases of system such as preliminary system study, detailed system study, system analysis, design, coding, testing, implementation and maintenance are explained. Computer based systems are defined. System development life cycle is discussed in detail. The different phases of the development of system are explained in detail.

1.7 TERMINAL QUESTIONS

1. Define a system. Explain the components of a system.

2. How do you explain system development life cycle?

3. Discuss the importance of system analysis and design in the development of a system.

1.8 KEY TO INTEXT QUESTIONS

1. (A) True (b) False (c) True (d) False (e) False

2. (a) first (b) detailed (c) logical data structure

   (d) structure design (e) pilot (f) manual

2

SYSTEM DESCRIPTION TECHNIQUES

2.1 INTRODUCTION

Graphical representation of any process is always better and more meaningful than its representation in words. Moreover, it is very difficult to arrange and organize the large amount of data into meaningful interpretation of the whole. System Analysis and Design makes use of the various tools for representing and facilitating comprehension of the complex processes and procedure involved. In this lesson, we present some details about Flowchart, Data flow diagrams (DFD), Decision Tables and Decision Trees.

2.2 OBJECTIVES

After going through this lesson you would be able to

- draw flowchart
- represent any physical system through DFD
- prepare decision table
- display decision tree

2.3 FLOWCHART

The pictorial-representation of the programs or the algorithm is known as flowchart. It is nothing but a diagrammatic representation of the various steps involved in designing a system. Some of the symbols which are used in flowchart are:

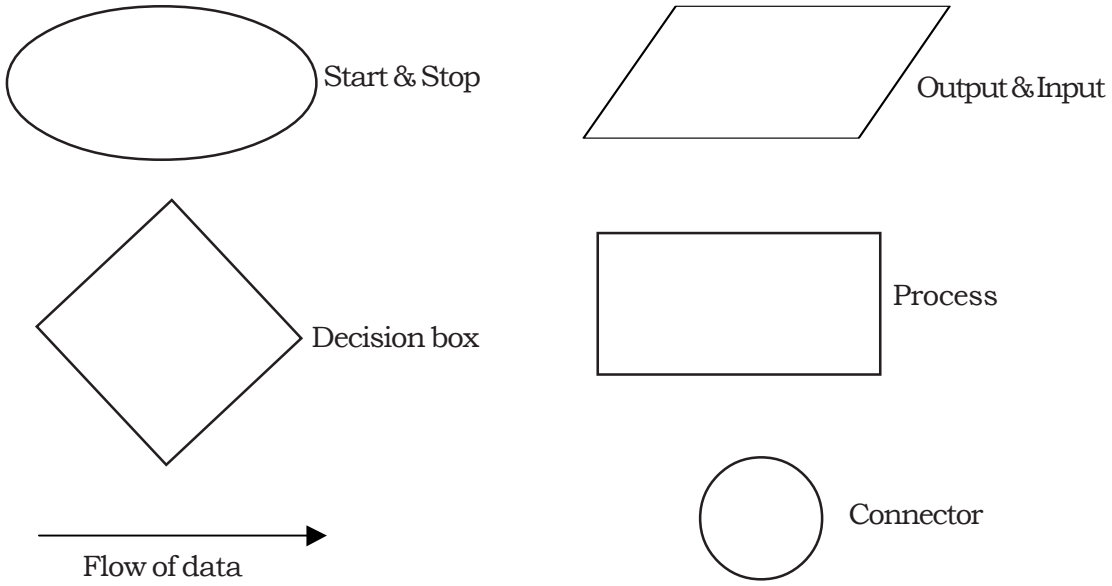


Fig. 2.1: Flowchart Symbols

A flowchart consists of a set of **'flowchart symbols'** connected by arrows. Each symbol contains information about what must be done at that point & the arrow shows the 'flow of execution' of the algorithm, i.e., they show the order in which the instructions must be executed. The purpose of using flowcharts is to graphically present the logical flow of data in the system and defining major phases of processing along with the various media to be used.

Flowcharts are of three types:

- System flowchart
- Run flowchart
- Program flowchart

(a) System Flowcharts

System flowchart describes the data flow for a data processing system. It provides a logical diagram of how the system operates. It represents the flow of documents and the operations performed in data processing system. It also reflects the relationship between inputs, processing and outputs. Following are the features of system flowcharts:

- the sources from which data is generated and device used for this purpose
- various processing steps involved
- the intermediate and final output prepared and the devices used for their storage.

Figure 2.2 is a sample system flowchart for the following algorithm:

1. Prompt the user for the centigrade temperature.
2. Store the value in C
3. If temperature = 0 Stop
4. Set F to 32 + (9*C/5)
5. Print the value of C, F
6. Stop

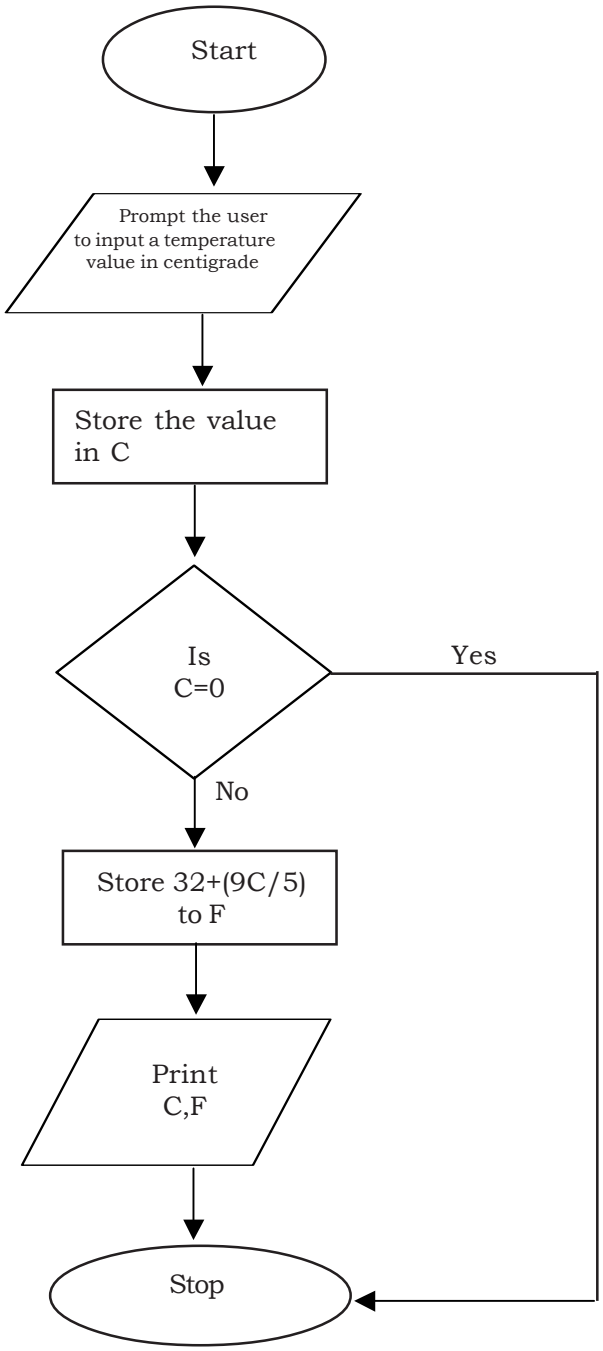


Fig. 2.2: System Flowchart

(b) Run Flowchart

Run flowchart is used to represent the logical relationship of computer routines along with inputs, master files transaction files, and outputs. Figure 2.3 illustrates a run flowchart.

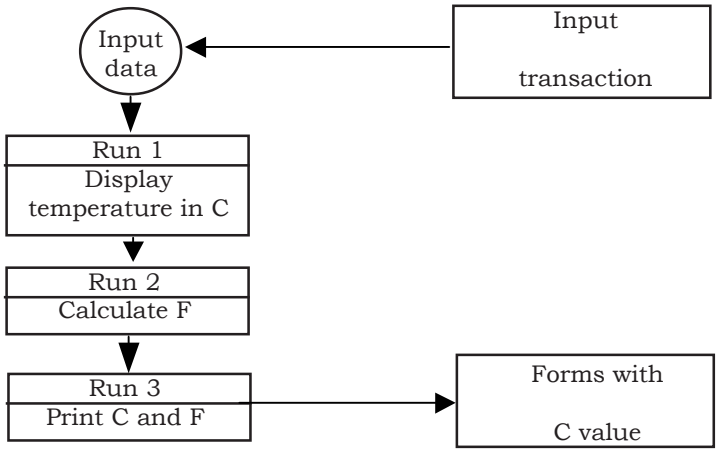


Fig. 2.3: Run Flowchart

(c) Program Flowcharts

A program flowchart represents, in detail, the various steps to be performed within the system for transforming the input into output. The various steps are logical/arithmetic operation, algorithms, etc. It serves as the basis for discussion and communication between the system analyst and the programmer. Program flowcharts are quite helpful to programmers in organizing their programming efforts. These flowcharts constitute an important components of documentation for an application.

Figure 2.4 represents a program flowchart for finding the sum of first five natural numbers (i.e., 1,2,3,4,5).

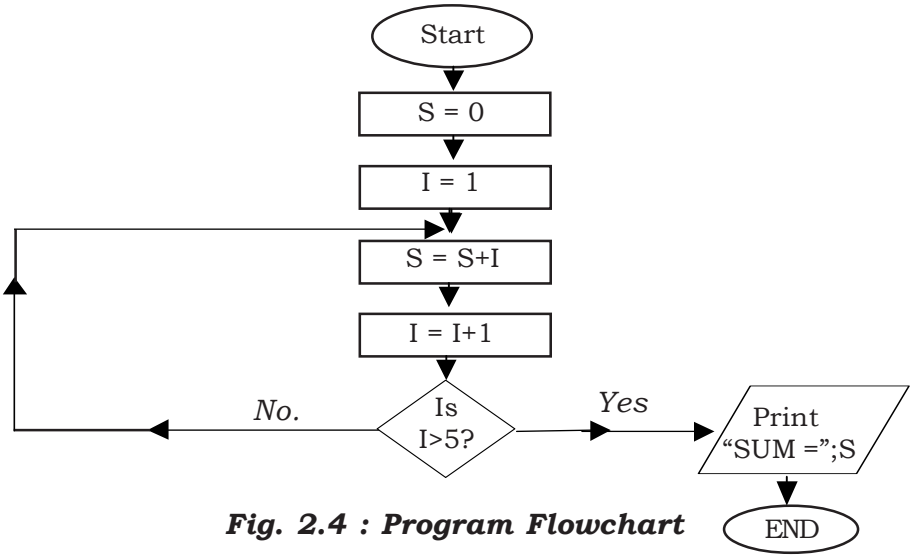


Fig. 2.4 : Program Flowchart

2.4 DATA FLOW DIAGRAM

Data flow diagrams are the most commonly used way of documenting the process of current and required systems. As their name suggest, they are a pictorial way of showing the flow of data into, around and out of a system.

(a) Defining DFD

Graphical representation of a system's data and how the process transform the data is known as Data Flow Diagram (DFD). Unlike flowcharts, DFDs do not give detailed descriptions of modules but graphically describe a system's data and how the data interact with the component of a system.

(b) Components of DFD

DFDs are constructed using four major components

- external entities
- data stores
- processes
- data flows

(i) External Entities

External entities represent the sources of data as input to the system. They are also the destination of system data. External entities can be called data stores out side the system. These are represented by squares.

(ii) Data Stores

Data stores represent stores of data within the system. Examples, computer files or databases. An open-ended box represents a data, which implies store data at rest or a temporary repository of data.

(iii) Processes

Processes represents activities in which data is manipulated by being stored or retrieved or transferred in some way. In other words, we can say that process transforms the input data into output data. Circles stand for a process that converts data into information.

(iv) Data Flows

Data flow represents the movements of data from one components to the other. An arrow identifies data flow – data in motion. It is a pipeline through which information flows. Data flows are generally shown as one-way only. Data flows between external entities are shown as dotted lines.

(c) Physical & Logical DFD

Consider the Fig. 2.5. It is clear from the figure that orders are placed, orders are received, the location of ordered parts is determined and delivery notes are dispatched along with the order.

Fig. 2.5

It does not, however, tell us how these things are done or who does them. Are they done by computers or manually and if manually who does them? A logical DFD of any information system is one that models what occurs without showing how it occurs.

A physical DFD shows, how the various functions are performed and who does them. Consider the following figure:

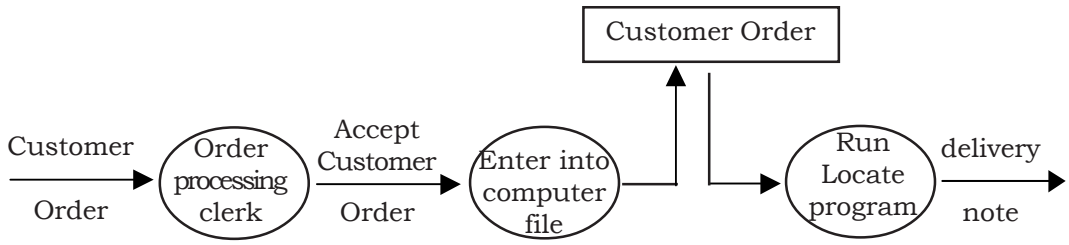


Fig. 2.6

Fig. *2.6 is opposite to the Fig. 2.5.* It shows the actual devices that perform the functions. Thus there is an "order processing clerk", an "entry into computer file" process and a "run locate program" process to locate the parts ordered. DFD that shows how things happen or the physical components, are called physical DFDs.

Typical processes that appear in physical DFDs are methods of data entry, specific data transfer or processing methods.

(d) Difference between Flowcharts and DFDs

The program flowchart describes boxes that describe computations, decisions, interactions & loops. It is important to keep in mind that data flow diagrams are not program flowcharts and should not include control elements. A good DFD should

- have no data flows that split up into a number of other data flows
- have no crossing lines
- not include flowchart loops of control elements
- not include data flows that act as signals to activate processes

INTEXT QUESTIONS

1. Write True or False for the following statements.
   (a) The pictorial representation of the programs or the algorithm is know as flowchart.
   (b) Flowcharts are of three types: system flowchart, run flowchart and program flowchart.
   (c) Run flowchart describes the data flow for a data processing system.
   (d) System flowchart represents the various steps to be performed within the system for transforming the input into output.
   (e) Graphical representation of a systems data and how the processes transform the data is known as Data Flow Diagram.
   (f) A good DFD should have crossing lines.
   (g) External entities represent the source of data as input to the system.

2.5 DECISION TABLES AND DECISION TREES

Decision tables and trees were developed long before the widespread use of computers. They not only isolate many conditions and possible actions but they help ensure that nothing has been overlooked.

(a) Decision Tables

The decision table is a chart with four sections listing all the logical conditions and actions. In addition, the top section allows space for title, date, author, system, and comment as shown in the Fig.2.7.

Following are the four sections of a decision table

TITLE:	DATE;
Author:	System;
Comments:	
Condition Stub	Condition entry
Action Stub	Action Entry

Fig. 2.7: Decision Table

The **Condition Stub** contains a list of all the necessary tests in a decision table. In the lower left-hand corner of the decision table we find the **Action Stub** where one may write all the processes desired in a given module. Thus Action Stub contains a list of all the processes involved in a decision table.

The upper right corner provides the space for the **Condition Entry** – all possible permutations of 'yes' and 'no' responses related to the condition stub. The 'yes' and 'no' possibilities are arranged as a vertical column called rules. Rules are numbered 1,2,3 and so on. We can determine the rules in a decision table by the formula:

Number of rules = 2^N = 2^N where N represents the number of condition and ^ means exponentiate. Thus a decision table with four conditions has 16=(2^4 =2 X 2X 2X 2 = 16) rules. One with six conditions has 64 rules and eight conditions yield 256 rules.

The **Condition entry** contains a list of all the yes/no permutations in a decision table. The lower right corner holds the **action entry**.

X's or dots/dash indicate whether an action should occur as a consequence of the yes/no entries under condition entry. X's indicate an action should occur as a consequence of the yes/no entries under condition entry, while dots indicate no action.

Thus Action entry indicates via dot or X whether something should happen in a decision table or not. Let us consider the following example of book order illustrated by Fig 2.8

If order is from book store

And if order is for 6 copies

Then discount is 25%

Else (if order is for less then 6 copies)

No discount is allowed

else (if order is from libraries)

If order is for 50 copies or more

Then discount is 15%

Else if order is for 20 to 49 copies

Then discount is 10%

Else if order is for 6 to 19 copies

Then discount is 5%

Else (order is for less then 6 copies)

No discount is allowed

A decision table for the above process is illustrated below

TITLE:		DATE:					
Author: Comments:		System:					
Condition Stub		Condition entry					
		1	2	3	4	5	6
IF	Customer is bookstore	Y	Y	N	N	N	N
	Order size is 6 or more	Y	N	N	N	N	N
	Customer is library	N	N	Y	Y	Y	Y
	Order size is 50 or more	N	N	Y	N	N	N
	Order size is 20-49	N	N	N	Y	N	N
	Order size is 6-19	N	N	N	N	Y	N
	Allow 25% discount	X	-	-	-	-	-
	Allow 15% discount	-	-	X	-	-	-
Then	Allow 10% discount	-	-	-	X	-	-
	Allow 5% discount	-	-	-	-	X	-
	No discount	-	X	-	-	-	X
	Action Stub	Action Entry					

Fig. 2.8: Decision Table

(b) Decision Tree

The decision tree defines the conditions as a sequence of left to right tests. A decision tree helps to show the paths that are possible in a design following an action or decision by the user. Fig. 2.9 illustrates the concept of decision tree.

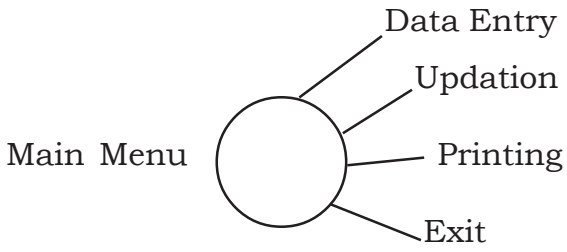


Fig. 2.9: Decision Tree

Decision tree turns a decision table into a diagram. This tool is read from left to right, decision results in a fork, and all branches end with an outcome. Fig. 2.10 illustrates the decision tree for the book order decision table we saw earlier.

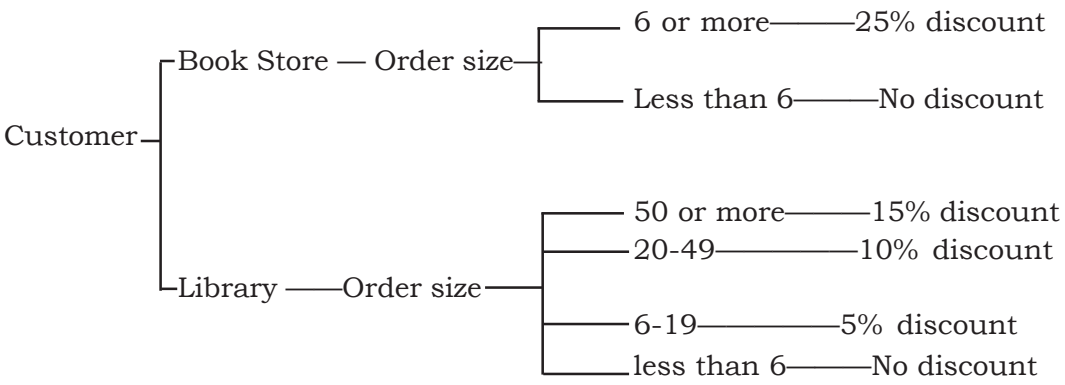


Fig. 2.10 Decision Tree for Book Order

INTEXT QUESTIONS

2. Write True or False for the following statements.

   (a) The condition stub contains a list of all necessary tests in a decision table.

   (b) The condition entry contains a list of all the yes/no permutations in a decision table.

   (c) The decision tree defines the conditions as a sequence of right to left tests.

   (d) A decision tree does not help to show the path that are possible in a design action or decision by the user.

   (e) The action stub contains a list of the processes involved in a decision table.

2.6 WHAT YOU HAVE LEARNT

Various specification tools and techniques of system analysis and designing were discussed in detail. Various tools such as flowcharts, data flow diagrams, decision tables & decision trees were explained. These tools and techniques are used when the system under study involves the development of computer based information system.

2.7 TERMINAL QUESTIONS

1. What is flow chart?
2. Explain different types of flow charts.
3. Define DFD and explain different components of DFD.
4. Explain decision table and decision tree.

2.8 KEY TO INTEXT QUESTIONS

1. (a) True (b) True (c) False (d) False (e) True (f) False (g) True

2. (a) True (b) True (c) False (d) False (e) True

3

SYSTEM DEVELOPMENT METHODOLOGIES

3.1 INTRODUCTION

Different types of system development methodologies are used in designing information system. Depending upon the actual requirement of the system, different approaches for data processing are adopted. However, some system groups recommend a centralized data processing system while others may go in for a distributed data processing system. In a centralized data processing, one or more centralized computers are used for processing and the retrieval of information is done from them. The distributed processing system involves a number of computers located remotely in the branches/ departments of the organization. The client/server technology is also gaining popularity these days.

3.2 OBJECTIVES

After going through this lesson you would be able to

- differentiate between the advantages and disadvantages of centralized/distributed data processing system
- distinguish between various approaches to the information system
- explain networking environment
- explain the meaning of client/server technology

3.3 DATA PROCESSING SYSTEM

Data processing techniques are very much dependent on the kind

of applications and the working environment. The activities involved in data processing are along departmental lines and application based such as Store Management, Production Planning & Control, Sales Accounting, Financial Accounting, Student Information System, etc. The basic input data are the real resource of data processing. With the progress in technology the concept of integrated data processing has also come into being. In integrated data processing the output data of one application can be used as the input of another application. Depending upon the application area, working environment and the needs of the management, there are basically two approaches to data processing:

- Centralized data processing
- Decentralized data processing

3.4 CENTRALISED DATA PROCESSING SYSTEM

With the increasing use of computer based data processing, there has been a growing tendency in the minds of management to centralize the data processing activities. A separate department EDP (Electronic Data Processing) department is established to carry out the data processing work of different departments in the organization. Many times the data processing is also done by hiring the services of the outside agencies and with the passage of time and experience in-house set is developed for data-processing.

The centralized data processing system provides the following benefits:

- The emergence of data takes place only at one place.
- The loss of data is minimized.
- The methods and machines can be standardized.
- Services of more competent and technical personnel can be taken.
- It is also very cost-effective particularly in the case of large operations.
- Duplication of work can be avoided.

The disadvantages, however, are:

- Lack of cooperation from managers, who do not like to be under control of centralized Data Processing department.

- Resistance from managers for mechanization of the data processing activities relating to their various functions.
- It is difficult to provide equitable services to various departments.
- The data security is also questioned.

3.5 DECENTRALISED DATA PROCESSING SYSTEM

In the decentralized data processing system, there is really a divisional breakdown of computing services. Each division, unit or department handles its own computer needs and does not like to interact with any other division, unit or department. It is well suited to a decentralized mangement scheme in which organizational autonomy is important. For example, research divisions of a large organization may adopt the decentralized data processing approach to provide data security of their work.

Arguments in support of decentralized data processing include the following:

- Familiarity with local problems
- Rapid response to local processing needs
- Profit-and-loss responsibility can be easily fixed

The drawbacks of the decentralized data processing system are:

- There is duplication of activities and redundancy in the maintenance of files.
- It is difficult to maintain uniformity in the procedures throughout the organization.
- The overall cost of the data processing for the organization is more.

3.6 INFORMATION SYSTEM

The information system aims at providing detailed information on a timely basis throughout the organization so that the top management can take proper and effective decisions. The information system cuts across departmental lines and help achieving overall optimization for the organization.

The organization is viewed as a network of inter-related sub-systems rather than as a hierarchy of manager-subordinate relationship. The information system can be of two types:

- Integrated information system
- Distributed information system

(a) Integrated Information System

The integrated information system is based on the presumption that the data and information are used by more than one system in the organization. Accordingly, data and information are channeled into a reservoir or database. All the data processing and provision of information is derived and taken from this common database. The development of an integrated information system requires a long-term overall plan, commitment from management at all levels, highly technical personnel, availability of sufficient fund, and sophisticated technology. It also requires adequate standby facilities, without which the system is doomed to failure. Because of its integrated component, the modification to the system is quite difficult and the system development takes a fairly long time.

(b) Distributed Information System

There are opinion that development of an integrated information system is embodied with several practical problems and therefore, not feasible. This view has been reinforced by the failure of integrated systems in various large organizations. The concept of a distributed information system has emerged as an alternative to the integrated information system. In the distributed information system, there are information sub-systems that form islands of information systems. The distributed information system aims at establishing relatively independent sub-systems, which are, however, connected through communication interfaces.

Following are the advantages of the distributed information system:

- The processing equipment as well as database are dispersed, bringing them closer to the users.
- It does not involve huge initial investment as is required in an integrated system.
- It is more flexible and changes can be easily taken care of as per user's requirements.
- The problem of data security and control can be handled more easily than in an integrated system.
- There is no need of standby facilities because equipment breakdown are not as severe as in an integrated system.

The drawbacks of the distributed system are:

- It does not eliminate duplication of activities and redundancy in maintaining files.
- Coordination of activities becomes a problem.
- It needs more channels of communications than in an integrated system.

It is possible to consider several alternative approaches, which fall between the two extremes – a completely integrated information system and a totally independent sub-system. It is to be studied carefully what degree of integration is required for developing an information system. It depends on how the management wants to manage the organization, and the level of diversity within the organization.

3.7 MULTI-USER ENVIRONMENT

The necessity of sharing of data and information gave rise to multi-user environment. In a multi-user environment, there is a concept of file server and user nodes or user terminals connected to the file server. There are various ways of developing a multi-user environment depending upon the connectivity. There is local area network (LAN) where nodes are connected with the file server with cables through which the data and information are transferred from file server to the different nodes connected to the file server and vice-versa. In a Wide Area Network (WAN), the nodes are connected through MODEM or through satellite.

3.8 NETWORKING/FILESERVER SYSTEM

In a Local Area Network, all the data and programme files are stored in a file server. A file server is the central node in the network. All the users connected to the file server through different nodes can access the data and information stored in the fileserver simultaneously. The file server in a LAN acts as a central hub for sharing peripherals like, printers, modems, etc. In a LAN an application running on a workstation reads and writes files on the file server. In many cases the entire files are pumped across the network on behalf of the operations taking place on LAN PCs. A file server is not involved in processing of an application. It simply stores files for applications that run on LAN PCs. For example, you might have a personal database manager and then information in a file on the file server. The file server sends all or part of the data file across the network to

your workstation. As you work with your personal database manager and the database on your workstation, the file server does not take part at all. When you save the file it goes back to the file server across the network.

The inherent design of the LAN/file server computing model is shown in Fig.3.1.

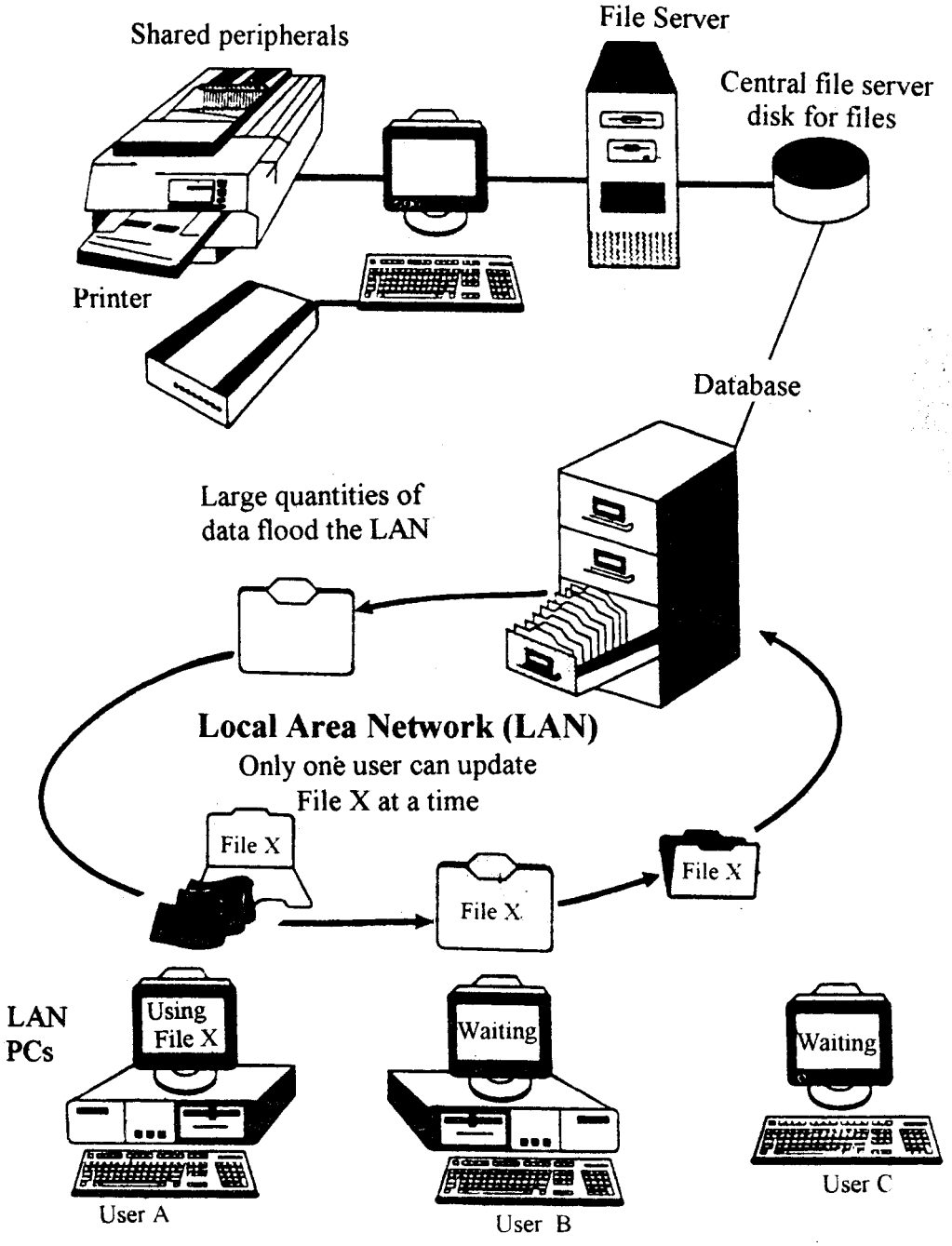


Fig. 3.1: Network/File server system

Two flaws limit a file server system for multi-user applications. First, the file server model does not deliver the data concurrency (simultaneous access to a single data set by more than one user), that is required frequently by multi-user applications. The reason behind it is that the file server operates in files, which are set of large number of data records and prevents a user from sharing a file when another user has locked it out. Second, if many workstations request and send many files in a LAN, the network can quickly become saturated with traffic, creating a bottleneck that degrades overall system performance.

INTEXT QUESTIONS

1. Fill in the blanks.

   (a) There are basically two approaches of data processing centralized data processing and _____.

   (b) In _____ data processing the loss of data is minimized.

   (c) The _____ information system is based on the presumption that data and information are used by more than one system in the organization.

   (d) The drawback of distributed system is that, it does not eliminate_____ of activities and redundancy in maintaining files.

   (e) In a Local Area Network all the data and program files are stored in a _____.

3.9 CLIENT\SERVER SYSTEM

The limitations of the network\file server system have led to the development of the client\server system. It delivers the benefits of the network-computing model along with the stored data access. Any local area network could be considered as client\server system, since work-stations (clients) request services such as data, program file or printing from server.

A client\server has three distinct components, each focusing on a specific job: a data-base server, a client application and a network. Fig.3.2 represents an idea about the components of the client\server-computing model.

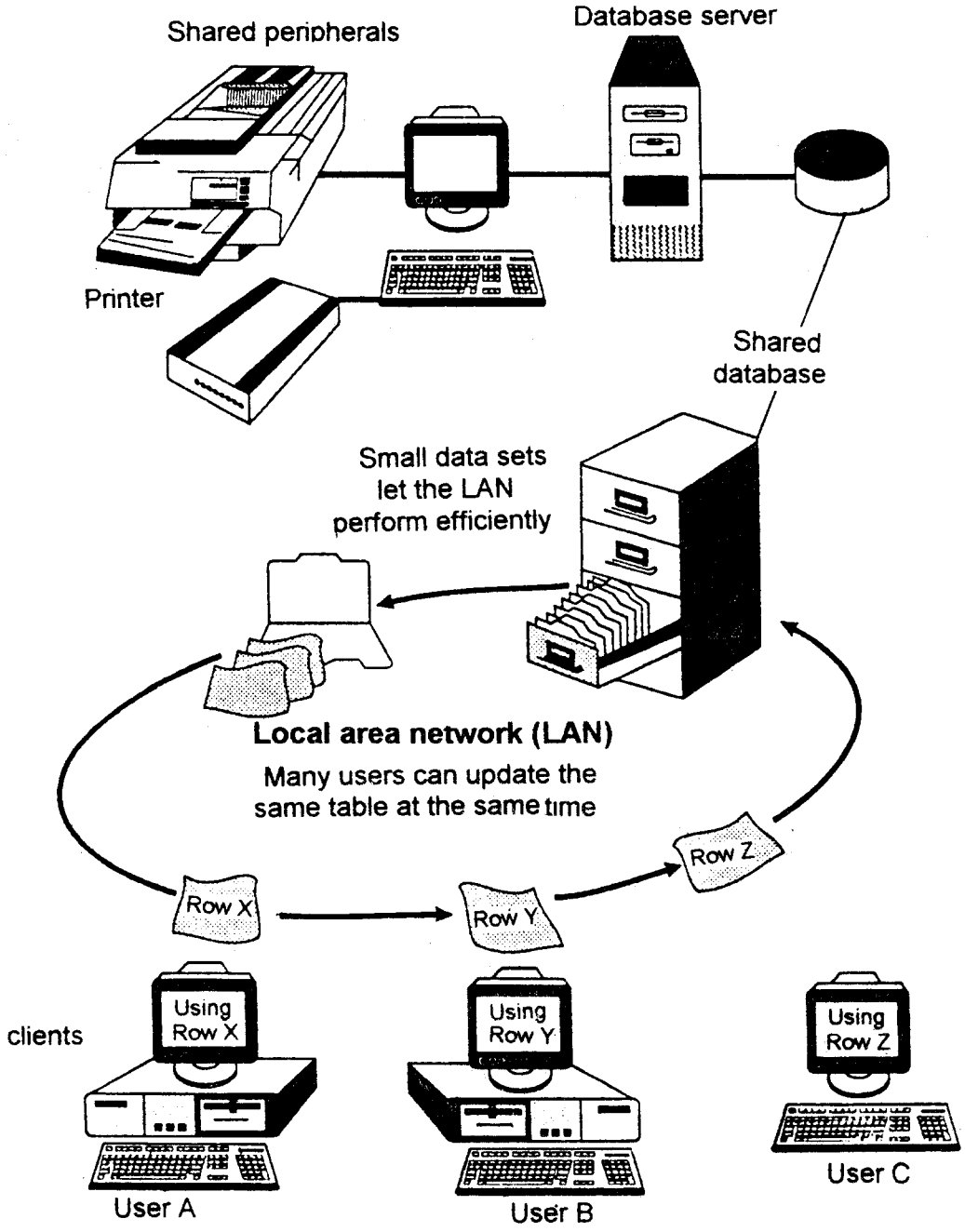


Fig. 3.2: Components of client/ server computing module

3.10 DATABASE SERVER

A server (or "back end") manages the resources such as database, efficiently and optimally among various clients that simultaneously request the server for the same resources. Database server mainly concertrates on the following tasks.

- Managing a single database of information among many concurrent users.
- Controlling database access and other security requirements.
- Protecting database of information with backup and recovery features.
- Centrally enforcing global data integrity rules across all client applications.

3.11 CLIENT APPLICATION

A client application (the "front end") is the part of the system that users apply to interact with data. The client application in a client/server model focus on the following job:

- Presenting an interface between the user and the resource to complete the job
- Managing presentation logic
- Performing application logic
- Validating data entry
- Managing the request traffic of receiving and sending information from database server

3.12 NETWORK

The third component of client/server system is network. The communication software are the vehicles that transmit data between the clients and the server in client server system. Both the client and the server run communication software that allows them to talk across the network.

INTEXT QUESTIONS

2. State whether the following statements are True and False.

   (a) Any Local Area Network could be considered as client/server system.

(b) A server application is the part of the system that users interact with.

(c) A client/server has three distinct components: a data base server, a client application and a network.

(d) The third component of a client/server system is network.

(e) Database server mainly concentrates on validating data entry.

3.13 WHAT YOU HAVE LEARNT

In this lesson different data processing methodologies such as centralized and decentralized have been explained. The development methodologies of various information system are also explained. A brief description of network/file server and client/server technology has also been made.

3.14 TERMINAL QUESTIONS

1. Explain various data processing techniques.
2. What is the difference between integrated information system and distributed information system? Explain in brief.
3. Define client/server system. Explain the different components of a client/server system.

3.15 KEY TO INTEXT QUESTIONS

1 (a) decentralized data processing (b) centralize (c) integrated (d) duplication (e) file server

2. (a) True (b) False (c) True (d) True (e) False

4

VISUAL FOXPRO - AN INTRODUCTION

4.1 INTRODUCTION

Visual FoxPro is a Relational Database Management System (RDBMS), which allows you to work with several logically related tables of data simultaneously. A Table in a database contains a number of Rows and Columns. One row in the table is equivalent to one record and one column is equivalent to one field.

4.2 OBJECTIVES

After going through this lesson, you would be able to

- start Visual Foxpro menus
- use shortcut keys
- use Visual Foxpro menus.
- explain the concept of DBMS

4.3 CONCEPT OF DATABASE MANAGEMENT SYSTEM (DBMS)

Before you understand the concept of Database Management System. (DBMS), it is necessary that you understand few basic terms related to DBMS. Let us begin with *data.* Anything can be data, e.g., a number, name of a person or place, address, etc. When a data is meaningful, it is called ***Information*** and a ***Database*** is an organized collection of related information. You can use database for

- retrieving desired information
- taking meaningful decision
- reorganizing information
- processing information

Database Management System (DBMS) is a generalized software package used to build and manage the database, i.e., add, modify (edit), update, delete and sort (arrange in a particular order) information in the database. DBMS also helps to retrieve the desired information in the required format from the database. Visual FoxPro is a leading database Management System. It is a member of Microsoft Visual Studio. Visual FoxPro is one of the Relational Database Management System (RDBMS). It is windows based Graphical User Interface (GUI) RDBMS. The database is a broader concept in Visual FoxPro in which the information is stored in related tables. The table is equivalent to database of the earlier version of FoxPro. A database is a collection of various records comprising rows and columns. One row is one record and one column is one field. A *record* is collection of logically related fields and a *field* is one column information. In Visual FoxPro, there can be more than one table storing different stream of information. A table in a database contains a number of Rows and Columns. One row in the table is equivalent to one record and one column is equivalent to one field.

4.4 STARTING VISUAL FOXPRO

The minimum hardware/software configuration of the machine required for installing Visual FoxPro is that you should have a Pentium Series of Computer with 32 MB RAM and 10GB of Hard Disk Drive with Windows 95 or higher operating system. Here we have discussed 6.0 version of Visual Foxpro

In order to start with Visual FoxPro, you must ensure that Visual FoxPro system is already installed on your computer. To invoke Visual FoxPro, Double click the *My Computer* Icon on the Desktop Window. Then double click *C:* drive. Now, click on *Program Folder* and then *Microsoft Visual Studio*. Now double click the *Microsoft Visual FoxPro* icon as shown in Fig. 4.1

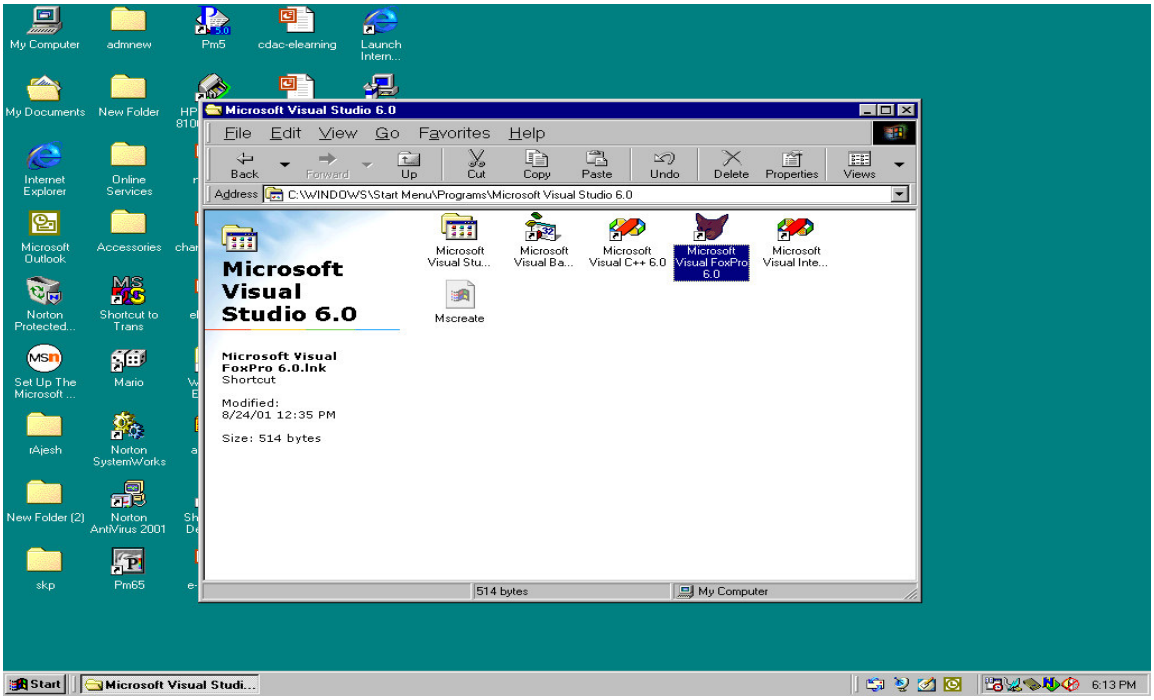


Fig. 4.1: Visual FoxPro icon within Program Folder of Windows

You can also make a shortcut of Visual FoxPro icon on your desktop for directly loading the system as shown in Fig. 4.2.

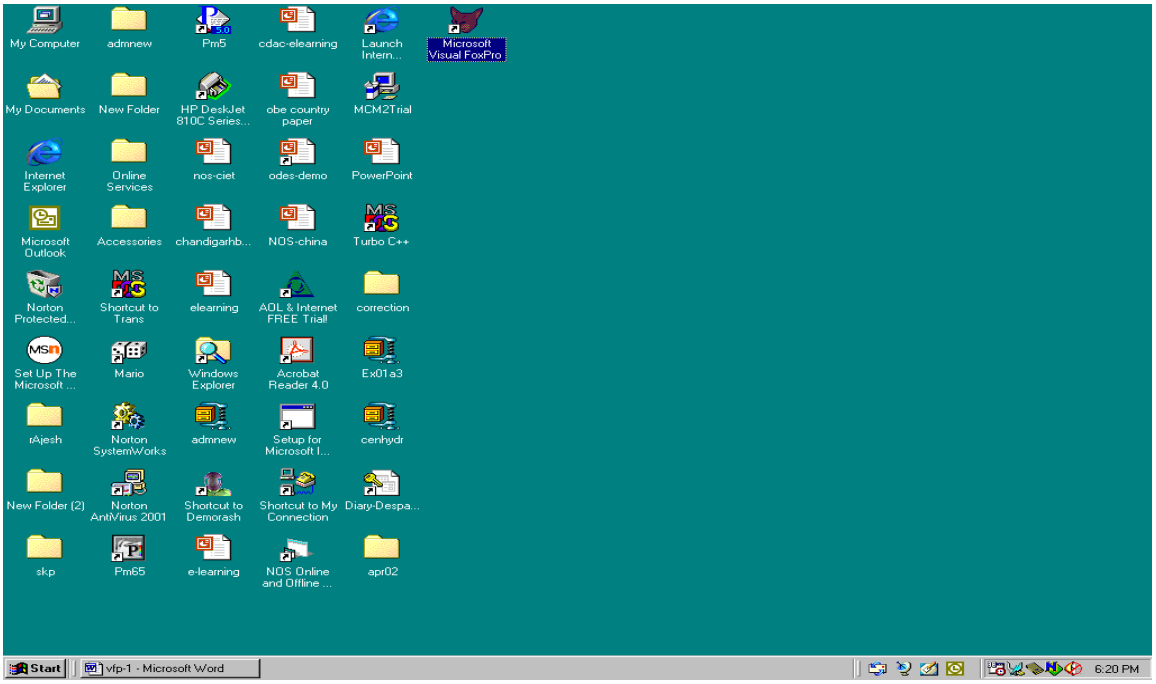


Fig. 4.2: Highlighted Visual FoxPro icon on the Desktop

In this case double click the Visual FoxPro icon from the desktop itself and then Visual FoxPro window will appear on your screen as shown in Fig. 4.3

Fig. 4.3 : Visual FoxPro with Command Window

Alternatively, you ca
Start menu then cli
Visual Studio 6.0.
on ***Microsoft Visua***
click the left mouse

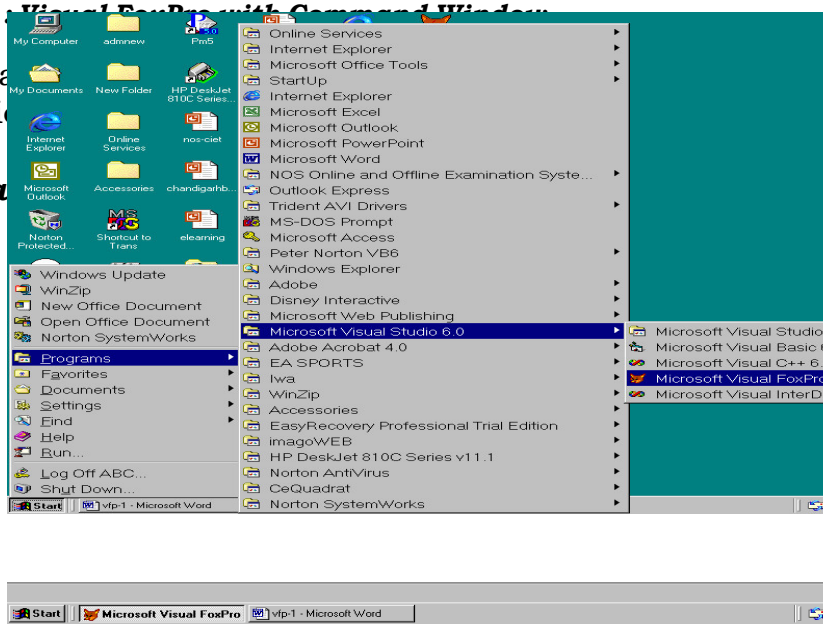


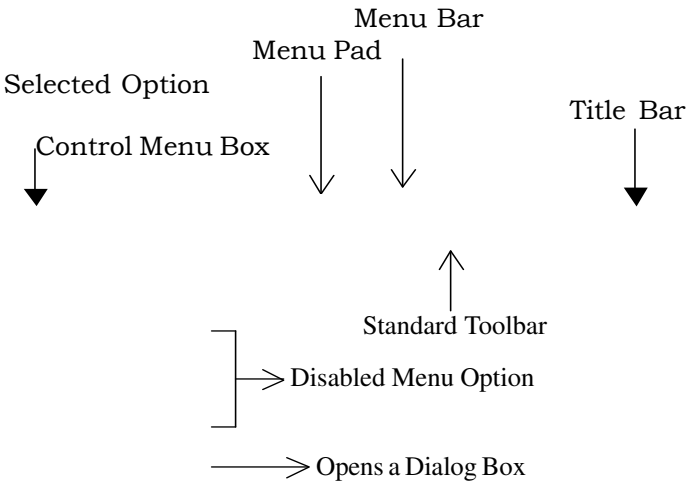
Fig. 4.4: Visual FoxPro through Start Menu

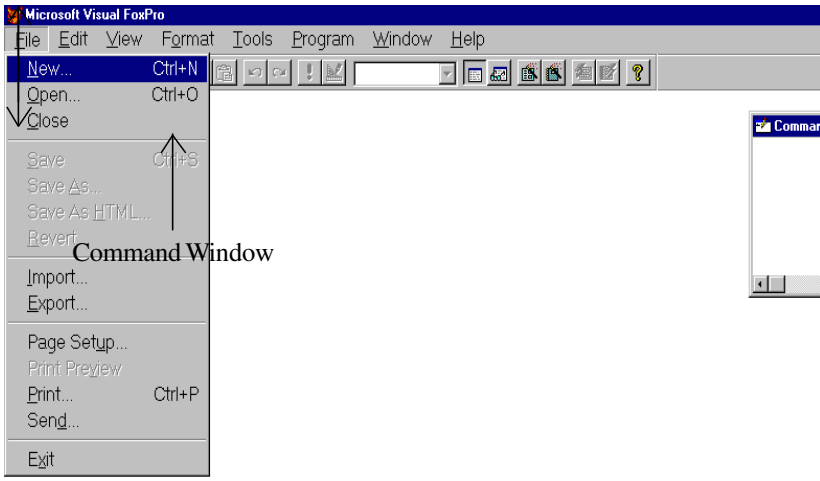
The Visual FoxPro window will appear on your screen as shown in the above fig. 4.3.

4.5 USING THE VISUAL FOXPRO MENUS

The Visual FoxPro Menu system can be classified into the following components as:

- Menu Bar
- Menu Pad (Item)
- Menus (Pull Down Menu)
- Menu Options
- Toolbar





Pull down Menu (Menu Popup)

Fig. 4.5: Visual FoxPro Menu Systems

4.5.1 Menu Bar

A **Menu Bar** is located at the top of the screen and displays all the available titles for menu as shown in the figure 1.5. The content of the menu bar changes when you move from one title to another title of the menu bar.

4.5.2 Menu Pads (Items)

The titles displayed on the menu bar are called **Menu Pads (Items)**. You can use either the mouse or the keyboard to display the menu associated with each menu pad. Using the mouse, click on the title of the menu you wish to use. To access the menu bar from keyboard, press <ALT> or <F10> key and then type the underlined letter for the menu you wish to use. For example <ALT>+F for invoking File Menu. You can also use the Left and Right arrow keys to move from one menu pad to another menu pad, and press <RETURN> key.

You will also find that some of the menu pads appear dimmed and cannot be highlighted or selected. These menu pads are disabled. You cannot display the menu if the menu pad is disabled.

4.5.3 Menus

When you choose a menu pad from the menu bar by clicking the menu pad, Visual FoxPro displays a **pull-down menu**. A **Menu** is a list of related options displayed in the pull-down menu. When you choose an option from a menu, Visual FoxPro executes it. Choosing an option means activating a highlighted option by clicking with mouse or pressing the <Spacebar> key or <Return> key.

4.5.4 Menu Options

When you choose a menu pad from the menu bar, Visual FoxPro displays a pull-down menu. A pull-down menu contains options as shown in the fig. 4.3. The options on each menu are logically related to menu pad.

To choose a desired menu option you want, use one of the following methods:

- Move to the menu pad and click. The pull-down menu appears. Click on the desired option you want.
- Press <Alt> key or <F10> key and then press the underlined letter (hot key) in the menu pad name. Type the underlined key for the menu option you want.

- Use the Left and Right Arrow keys to the menu pad you want to use and then press <Return> key. Use the Up and Down arrow keys to select the desired option and then press the <Return> key or <Spacebar> key.

You will find that some of the menu options have an ellipsis (...) after the option. This indicates that the option will further open a dialog box. A dialog box appears to request the additional information.

4.5.5 Toolbar

The **Toolbar** appears below the menu bar and displays the icons as shown in Fig. 4.5. To access the toolbar using the mouse, click on the icon you want to use.

INTEXT QUESTIONS

1. What is the difference between data and information?
2. Give the full form of RDBMS.
3. Choose the Correct Answer:
   (a) A database is a collection of
   (i) Fields (ii) Table (iii) Records (iv) Column
   (b) In Visual FoxPro, a Table is equivalent to
   (i) Row (ii) Column (iii) Field (iv) Database

4.6 VISUAL FOXPRO TEXT EDITOR

Modify Command is the standard **Text Editor** of the Visual FoxPro. In short it is also called **modi comm**. It allows you to create and modify text. Using this text editor you can code or modify a Visual FoxPro program. The following keys can be used for selecting and editing text.

Key(s)	Action
Cursor Movement	
Right Arrow	To move the cursor one character to the right.
Left Arrow	To move the cursor one character to the left.

Up Arrow	To move the cursor up one line.
Down Arrow	To move the cursor down one line.
Home	To move the cursor to the beginning of the current line.
End	To move the cursor to the end of the current line.
Page up	To move the cursor up the height of the current line.
Page down	To move the cursor down the height of the current line.
Ctrl+Home	To move the cursor to the beginning of the current text.
Ctrl+End	To move the cursor to the end of the current text.
Ctrl+Right Arrow	To move the cursor one word to the right.
Ctrl+Left Arrow	To move the cursor one word to the left.

Selecting Text

Shift+Right Arrow	To select one character to the right of the cursor.
Shift+Left Arrow	To select one character to the left of the cursor.
Shift+Up Arrow	To select one line up to the cursor.
Shift+Down Arrow	To select one line down to the cursor.
Ctrl+A	To select all text.

Deleting Text

Backspace	To delete the selected text or the character to the left of the cursor if no text is selected.
Del	To delete the selected text or the character to the right of the cursor if no text is selected.

Inserting Text

Ins	To switch between insert and overwrite mode.

4.7 SHORTCUT KEYS FOR MENU OPTIONS

Visual FoxPro provides various shortcut Keys to choose the menu options. Some of them are listed below.

Key(s)	Action
Ctrl+A	To select all the current objects.
Ctrl+C	To copy the current selection to the clipboard.
Ctrl+D	To start a Visual FoxPro Program
Ctrl+G	To move the cursor to the designated line in a text or in the command window.
Ctrl+M	To resume a program.
Ctrl+P	To Print.
Ctrl+R	To redo.
Ctrl+S	To save the file.
Ctrl+V	To paste the content of the clipboard at the current cursor position.
Ctrl+W	To leave the current operation and to save any modification.
Ctrl+X	To cut the selected object from its current location and to place it on the clipboard.
Ctrl+Y	To append new record.
Ctrl+Z	To undo.
Ctrl+F1	To activate the next window open within Visual FoxPro.
Ctrl+F2	To display the command window.
Ctrl+F4	To close the current window.
Ctrl+F5	To restore the current window to its original size.
Ctrl+F7	To move the current window with the arrow key.

Ctrl+F8	To increase/decrease the size of the current window.
Ctrl+F9	To minimize the current window to an icon.
Ctrl+F10	To maximize the current window.
Esc	To exit current editing without saving the modification.
F1	To activate on-line help.
F2	To open the view window.
F3	To list the content of the current table.
F4	To list the files in the current directory.
F5	To display the structure of the current table.
F6	To display the status.
F7	To display the current memory variables.
F8	To display the field names and the current record of the current table.
F9	To open the Browse window in append mode.

4.8 EXITING VISUAL FOXPRO

You can exit Visual FoxPro by using any one of the following methods:

- Type **quit** in the command window and press <Return> key.
- Open the **File** menu and choose the **Exit** option.
- Click mouse on the Control Menu Box from the Title bar of the main Visual FoxPro window as shown in the figure 4.3.
- Press <Alt+F4> key.

4.9 WHAT YOU HAVE LEARNT

In this lesson you have learned about the concept of database and how to invoke Visual FoxPro, Visual FoxPro menus, shortcut keys for menu options and how to exit Visual FoxPro. In the next lesson you will learn about the complete Visual FoxPro Menu System and Toolbars.

4.10 TERMINAL QUESTIONS

1. Discuss the concept of DBMS.
2. What are the various ways of starting Visual FoxPro?
3. Define the various Menu Pads available on the Menu Bar of Visual FoxPro.
4. Name the Text Editor of Visual FoxPro.
5. What is the shortcut for displaying the structure of the current table?

4.11 KEY TO INTEXT QUESTIONS

1 Data: Anything can be data, e.g. a number, name of a person or place, address, Information: When a data is meaningful, it is called *information.*

2 Relational Database Management Systems

3 (a) (iii) (b) (iv)

5

VISUAL FOXPRO - MENU SYSTEM

5.1 INTRODUCTION

The Visual FoxPro is a menu driven Relational Database Management System. It is easier to communicate with the Visual FoxPro menu system without programming. The in-built menu system of the Visual FoxPro helps the user to work without actually remembering the command. Visual FoxPro is suitable for multi user environment. It provides the facility for locking of Records when many users are simultaneously working on same database.

5.2 OBJECTIVES

After going through this lesson you would be in a position to

- explain the Visual FoxPro Menu System
- work with the various Menu Items of the Visual FoxPro
- explain the functionality of various options of the Toolbar

5.3 FILE MENU

The **File** menu contains options that permit you to create and save new files, open and close the existing files, settings for printing information on the paper, print files, enter printer information and to exit Visual FoxPro, as shown in Fig. 5.1

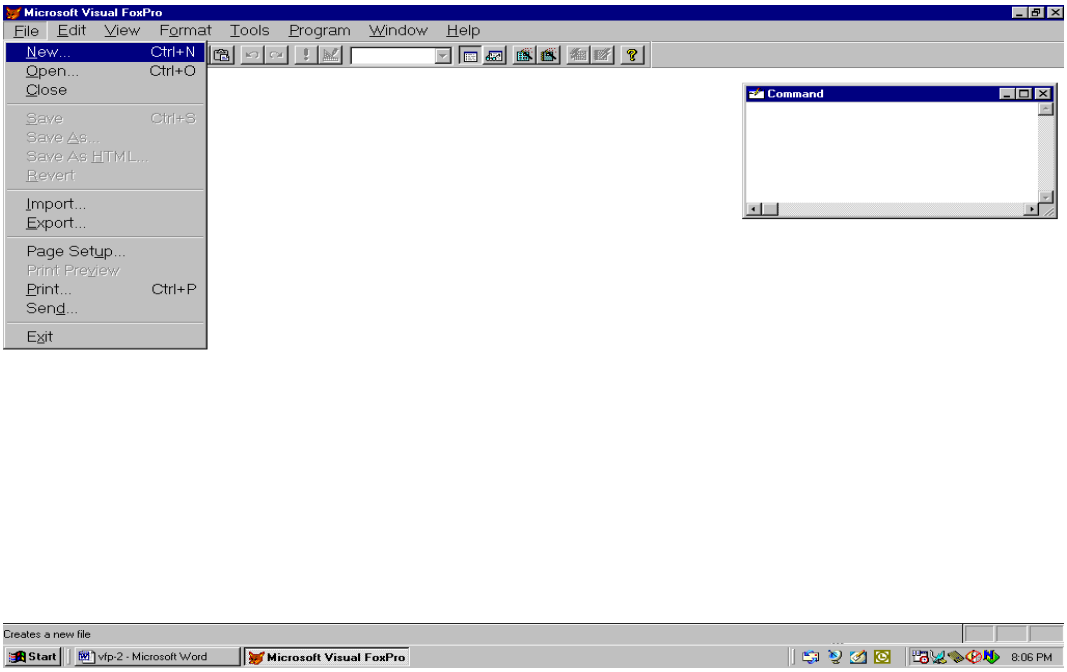


Fig. 5.1: File Menu

There are various options available in the **File** Menu option. The description of each option is given below:

Option	Description
New...	Allows you to create any type of File (Table/DBF, Program, Index, Report, Label, Form, Menu, Query Projects, Text File.)
Open...	Allows you to open any type of existing files.
Close	Closes the current window. If the window's contents have changed, you will be asked to save the changes.
Save	Allows you to store any changes you have made to the current text, program, label, report, screen, query, menu or project file without closing the file.
Save As...	Allows you to name and save a newly created file, or save a copy of a current file with a new name.

Save As Class... Allows you to save a form, or selected controls on a form, as a class definition. This option is available when you run the Form Designer.

Save as View... Allows you to save a query into a database (DBC) file. This option is available when you run the Query Designer.

Revert Allows you to return to last saved version of file. Note that the **Revert** option is only enabled when you have made changes to a text file, program file, label file, form file, report file, query file, menu file, project file or memo field since the file was last saved.

Import... Allows you to use file from other application

Export... Allows you to use Visual FoxPro file in other applications.

Print Setup... Allows you to select the printer you want to use and other several Options about the printer.

Print... Allows you to print only the sources listed below:

- The contents of the Command window.
- The contents of a file that is not currently open.
- The contents of ASCII file or clipboard.
- The contents of any open editing window.

Page Setup... Allows you to give settings for the page.

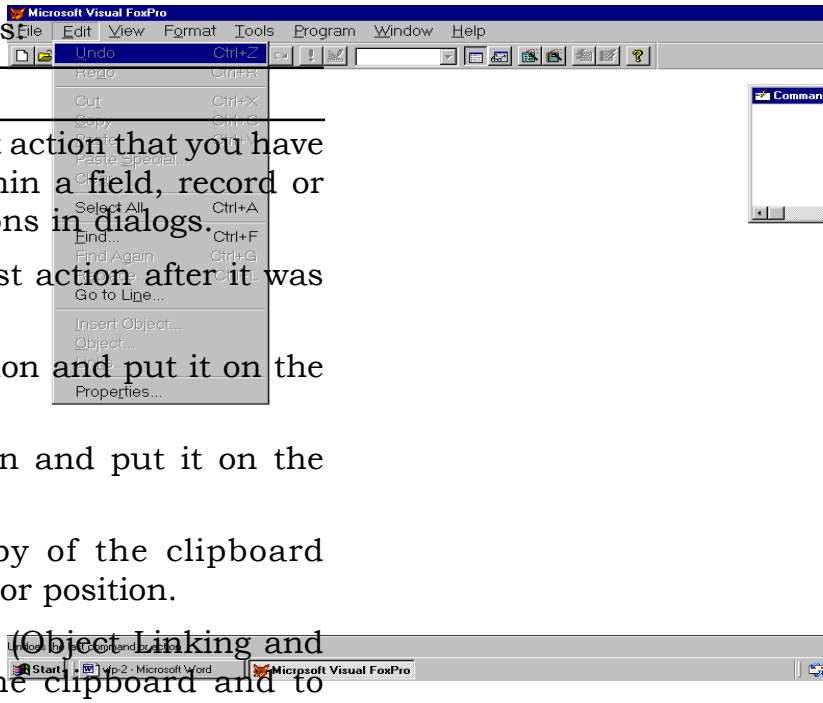
Page Preview Allows you to display, the printed image of the page on the screen.

Exit Allows you to leave Visual FoxPro and returns you to the windows desktop. **Exit** option of File menu is same as typing **Quit** in the Command Window or pressing <Alt+F4> key.

5.4 EDIT MENU

The **Edit** menu contains options that allow you to edit text in editing windows as shown in Fig. 5.2. You can also manipulate **Object Linking and Embedding (OLE)** objects in Visual FoxPro files.

Fig. 5.2: Edit Menu



The **Edit** menu options are described as follows:

Option	Description
Undo	Allows you to reverse the last action that you have performed on any text within a field, record or file, and in text editing regions in dialogs.
Redo	Allows you to restore the last action after it was undo.
Cut	Allows you to remove selection and put it on the clipboard.
Copy	Allows you to copy selection and put it on the clipboard.
Paste	Allows you to place a copy of the clipboard contents at the current cursor position.
Paste Special...	Allows you to place an OLE (Object Linking and Embedding) object from the clipboard and to establish a link to its source.
Clear	Allows you to remove the selected text, data, or object without placing it on the clipboard.

Select All	Allows you to select all objects or all lines of text, or all fields in current window.
Find...	Allows you to search for specified text.
Replace...	Allows you to find and replace specified text.
Goto Line...	Allows you to move the cursor to a designated line in a text, Program or memo editing window, or in the Command window.
Insert Object...	Allows you to place an OLE object on the clipboard and to establish a link to its source.
Object...	Allows you to activate OLE object, to edit the object or listen to it if it is a sound.
Links...	Allows you to edit the link from a source application to an OLE object in a file.

5.5 VIEW MENU

The **View** menu displays Reports, Label, and Form designers and toolbars and allows you to customize the way you work in a table.

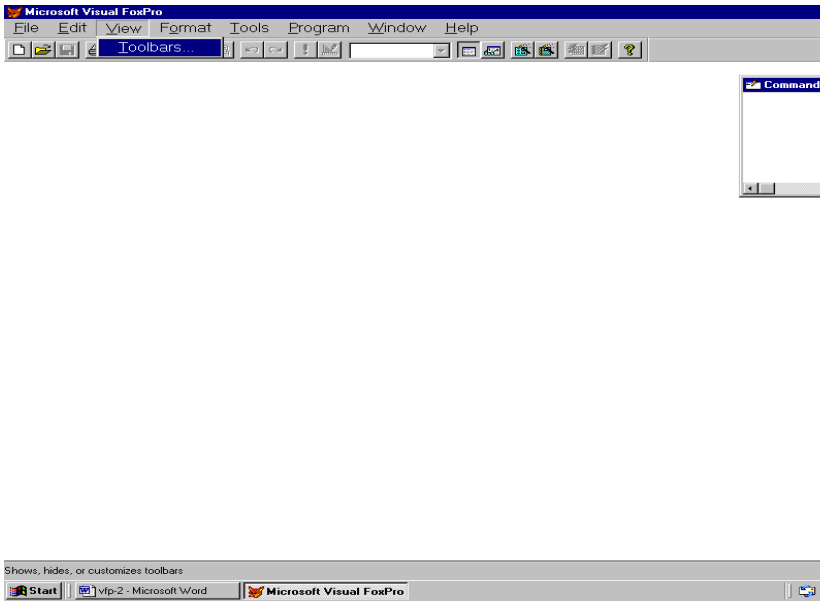


Fig. 5.3: View Menu

The **View** menu options are described as follows:

Option	Description
Toolbars...	Allows you to create, edit, hide and customize toolbars.
Properties...	Allows you to display the Properties window so you can set or change the properties of forms and controls. This option is available when you open the Form or Class Designer.
Browse <table>...	Allows you to display the contents of the current table horizontally in a Browse window. This option is available when you use a table.
Edit...	Allows you to display the contents of the selected table vertically. This option is available when you use a table.
Append Mode...	Allows you to add new records automatically to the end of the current table. This option is available when you use a table.
Database Designer...	Allows you to display the Database Designer. The option is available when you use a table that is associated with a database.
Table Designer...	Allows you to display the **Table Designer** for modifying a table's structure. This option is available when you use a table that is associated with a database.
Grid Lines	Allows you to remove or add lines in various windows. This option is available when you use a table or open the Label, Report, or Form or Class Designer.
Design...	Allows you to put a new or existing label, report, or form in design mode. This option is available when you open the Label, Report, or Form or Class Designer.
Tab Order...	Allows you to set the tab order for objects on a form. This option is available when you run the Form or Class Designer.
Code...	Allows you to display the code window to write, edit and display event code. This Option is

	available when you open the Form or Class Designer.
Form Controls	Allows you to display the Form Controls toolbar, which you
Toolbar...	use to create controls on a form. This option is only available when you open the Form or Class Designer.
Layout Toolbar...	Allows you to display the Layout toolbar, which you use to align, size, and position controls. This option is available when you open a Label, Report, or Form or Class Designer.
Color Palette Toolbar...	Allows you to display the Color Palette toolbar, which you use to specify foreground and background colors for a control. This option is available when you open the Label, Report, or Form or Class Designer.
Show Position...	Show in the status bar the position, height, and width of the selected object in a form, report, or label. This option is available when you open the Label, Report, Form or class Designer.
Preview...	Allows you to display a report or label in a print preview. This option is available when you open the Label or Report Designer.
Report Controls Toolbar...	Allows you to display the Report Controls toolbar which you use to create controls on a report. This option is available when you Open the Label or report Designer.
General Options...	Allows you to specify code for an entire menu system. This option is available when you open the Menu Designer.
Menu Options...	Allows you to specify code for a specific menu. The option is available when you open the Menu Designer.
Maximize Top Pane...	To enlarge the top pane of the Query and View Designers. This option is available when you

	open the Query Designer or View Designer.
Minimize Top Pane...	To reduce the top pane of the Query and View Designers. This option is available when you open the Query Designer or View Designer.
Data Environment...	Allows you to display the Data Environment Designer. This optionis available when you open the Form Designer, Report Designer, or Label Designer.

5.6 FORMAT MENU

The **Format** Menu contains option for fonts, spacing, alignment, and object positioning, as shown in figure 5.4.

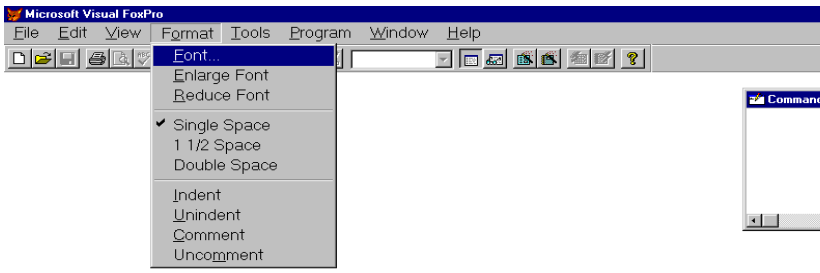


Fig. 5.4: Format Menu

The **Format** menu options are described as follows:

Option	Description
Text Alignment...	Allows you to adjust alignment and spacing of text within a field or label control. This option is available when you work with a report or label.

Fill...	Allows you to fill selected control with a pattern. This option is available when you work with a report or label.
Pen...	Allows you to set point size and the design pattern of lines and outlines for rectangles and rounded rectangles on your label or report. This option is available when you work with a report or label.
Mode	Determines whether the selected control is opaque or transparent. This option is available when you work with a report or label.
Font...	Allows you to select the font type, style and size. This option is available when you work with a text file, field or label control in a report or label, or the Command Window.
Enlarge Font	Allows you to increase the font size by the next-larger size available. This option is available when you work with a text or program file or the command window.
Reduce Font	Allows you to decrease the font size by the next-smaller size available. This option is available when you work with a text or program file.
Single Space	Allows you to display the text with no blank lines between lines of text. This option is available when you work with a text or program file.
1 ½ Space	Allows you to display the text with one-and-a-half blank lines between lines of text. This option is available when you work with a text or program file.
Double Space	Allows you to display the text with two blank lines between lines of text. This option is available when you work with a text or program file.
Indent	Allows you to indent the selected line or lines by one tab. This option is available when you work with a text or program file.

Remove Indent Allows you to remove previously inserted indents one at a time. This option is available when you work with a text or program file.

Align Aligns controls on a form or in a report, label or class. This option is available when you work with a report, label, form, or class.

Size Allows you to change control size. This option is available when you work with a report, label, form, or class, and create and select a control.

Horizontal Spacing Allows you to set the horizontal spacing of selected objects. This option is available when you work with a report, label, form, or class.

Vertical Spacing Allows you to set the vertical spacing between selected controls. This option is available when you work with a report, label, form, or class.

Bring To Front Allows you to send selected controls to the back layer so that any overlapping controls show up on top of them. This option is available when you work with a report, label, form, or class, add an object or objects, and select one or more of them.

Send To Back Allows you to send selected controls to the back layer so that any overlapping controls show up on top of them. This option is available when you work with a report, label, form, or class, add an object or objects, and select one or more of them.

Group Allows you to join selected controls so that they can be manipulated as a single control. It is equivalent to dragging the mouse to select all controls at once. This option is available when you work with a report or label and select more than one control.

Ungroup Allows you to separate previously grouped controls so that they can be manipulated individually. This option is available when

you work with a report, or label, and have grouped controls.

Snap to Grid — Allows you to move controls in grid increments when you select and drag them. The option is available when you work with a report, label, form, or class, add an object or objects, and select one or more of them.

Set Grid Scale — Allows you to define horizontal and vertical increments of the grid in pixels, and specifies whether the ruler displays inches or pixels. This option is available when you work with a report, label, form, or class.

5.7 TOOLS MENU

The **Tools** menu contains options that set system options, run wizards, create macros, and trace and debug source code, as shown in Fig. 5.5.

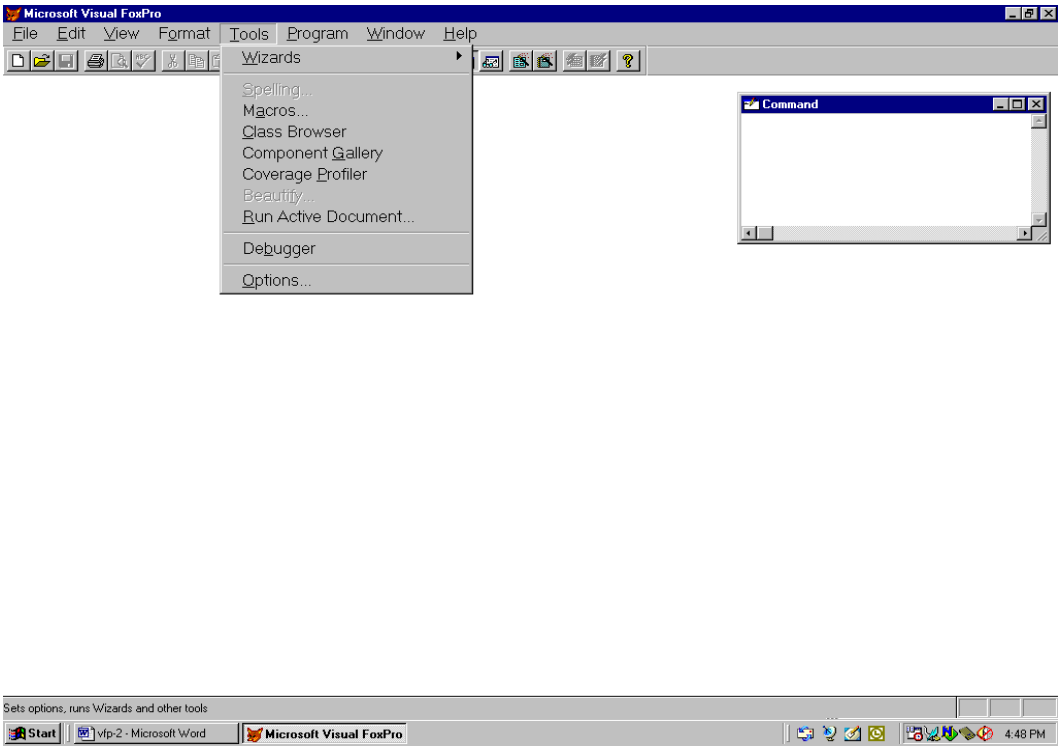


Fig. 5.5: Tools Menu

The **Tools** menu options are described as follows:

Option	Description
Wizards	Allows you to display a submenu of Visual FoxPro wizards.
Spelling	Allows you to run the spell checker.
Macros..	Allows you to define Key combinations to perform a series of keystrokes.
Trace Window	Allows you to open the Trace Window to watch program execution.
Debug Window	Allows you to open the Debug Window to monitor the values in programs.
Options...	Allows you to set many categories of system options.

5.8 PROGRAM MENU

The **Program** menu contains options that are used while programming in Visual FoxPro, as shown in Figure 5.6

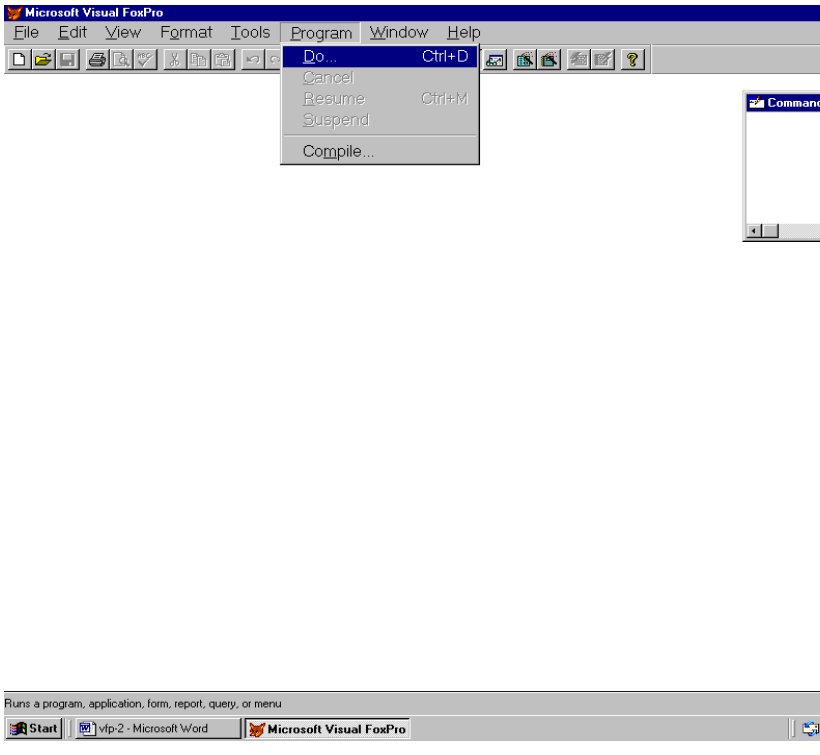


Fig. 5.6: Program Menu

The **Program** menu options are described as follows:

Option	Description
Do...	Allows you to execute a specified program file.
Cancel	Allows you to stop a currently running program.
Resume	Allows you to restart execution of a suspended (not cancelled) program at the line where execution paused when you choose **Trace** option.
Suspend	Allows, you to stop running the program, but keeps it open to resume program execution. This option is available when you run a program.
Compile...	Allows you to compile the specified program.

5.9 WINDOW MENU

The **Window** menu contains options that allow you to control windows, as shown in Fig. 5.7.

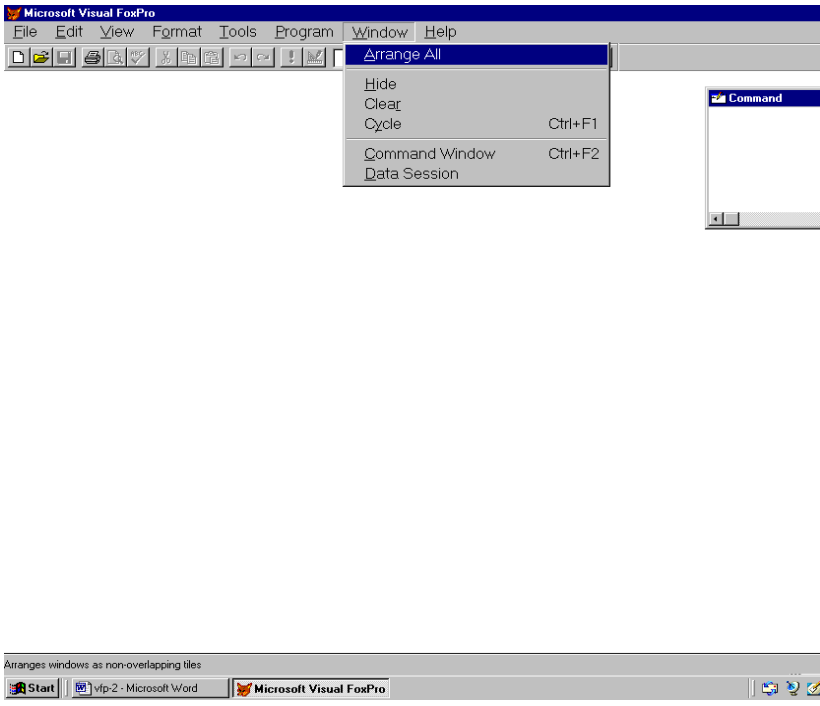


Fig. 5.7: Window Menu

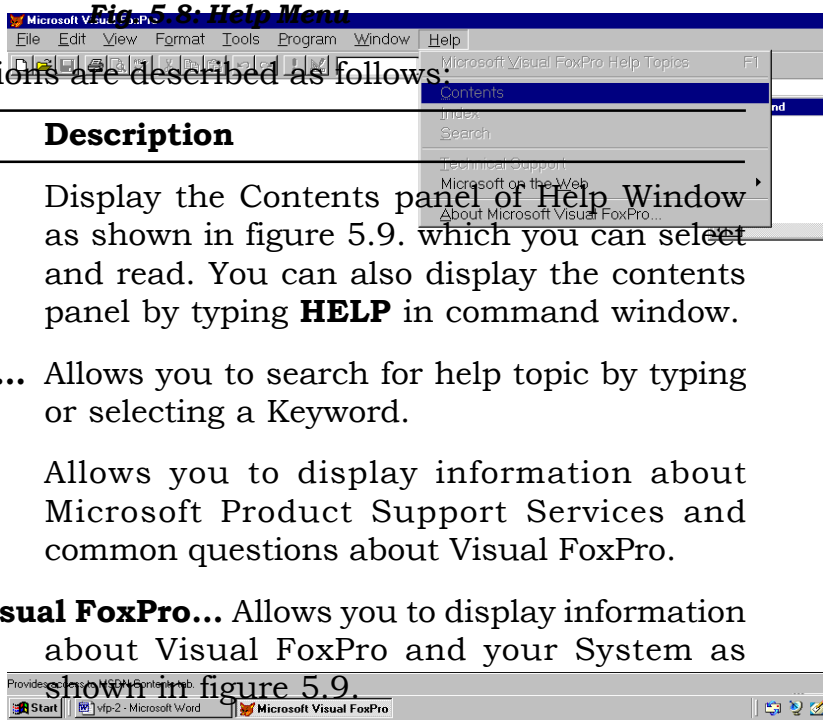
The **Window** menu options are described below:-

Option	Description
Arrange All	Allows you to display all open windows in an arranged fashion so that they don't overlap.
Hide	Allows you to remove the current window from view, but does not close it.
Hide All	Allows you to remove all windows from your view, but keep them open. This command is available when you hold down <Shift> Key while choosing the Window menu.
Show All	Allows you to display all open windows. This option is available When you hold down <Shift> Key choosing the window menu.
Clear	Allows you to remove the current output window. The command equivalent for the **Clear** option is to type **clear** in the command window.
Cycle	Allows you to rearrange open windows to bring successive ones to the front. The front most window moves to the back and the Next window becomes the front most window.
Command Window	Allows you to display the command window.
View Window	Allows you to open table files, establish relation and set work area properties.
1,2,3,...9	Displays the names of all open windows and allows you to choose from them.
More Windows	Allows you to display the More Window dialog box, in which you select a window to activate. This option is available when you have more than nine (9) windows open.

5.10 HELP MENU

The **Help** menu contains options that open Visual FoxPro online Help, direct you to technical assistance, and display information about your computer's configuration, as shown in figure 5.8.

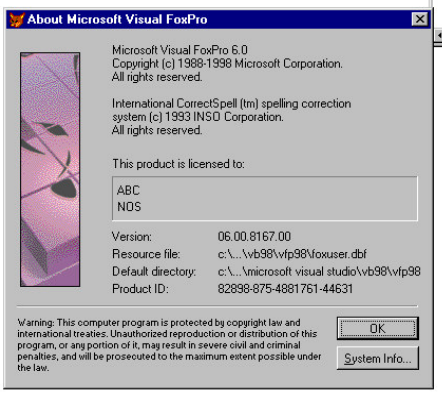
Fig. 5.8: Help Menu



The **Help** menu options are described as follows:

Option	Description
Contents	Display the Contents panel of Help Window as shown in figure 5.9. which you can select and read. You can also display the contents panel by typing **HELP** in command window.
Search for Help on...	Allows you to search for help topic by typing or selecting a Keyword.
Technical Support	Allows you to display information about Microsoft Product Support Services and common questions about Visual FoxPro.
About Microsoft Visual FoxPro...	Allows you to display information about Visual FoxPro and your System as shown in figure 5.9.

Fig. 5.9: About Visual FoxPro



INTEXT QUESTIONS

1. In order to open an existing table, which menu option will you choose?

   (a) Edit Menu (b) File Menu (c) Tools Menu (d) Help Menu

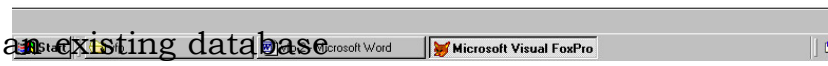
2. Which command physically removes the records from a table marked for deletion

   (a) Delete (b) Append (c) Pack (d) Erase

3. Which option of the Edit Menu allows you to insert an OLE object?

   (a) Paste (b) Copy (c) Link (d) Insert object

5.11 DATABASE MENU



The Database menu is added when you open an existing database or when you create a database schema to define persistent

relationships among the tables of a relational database (as shown in figure 5.10). The database schema contains the logically related tables, forms, reports, query etc.

The Database menu option are described as follows.

Option	Description
New Table	Allows you to create a new table using a wizard or a designer
Add Table	Allows you to add an existing table to a database
New Remote View	Allows you to create a new remote view using a wizard or a designer
New Local View	Allows you to create a new local view using a wizard or a designer
Remove	Allows you to remove the selected table from the database or delete it from the disk.
Modify	Allows you to open the selected table in the table designer.
Browse	Allows you to display the selected table in the Browse window for editing
Rebuild Table Indexes	Allows you to generate keys and rebuild indexes for the selected table.
Remove Deleted Records	Permanently removes all records from the active table that are marked for deletion.
Edit Relationship	Allows you to modify the relationship between tables.
Referential Integrity	Allows you to display the Referential Integrity Builder, in which you set up rules to control how records are inserted, updated or deleted in related tables.
Editing stored Procedures	Allows you to display a Visual FoxPro procedure in an editing window.
Clean Up Database	Allows you to run the PACK command to decrease the size of the database by removing rows that have been marked for deletion. The PACK command physically removes the record from the table marked for deletion.

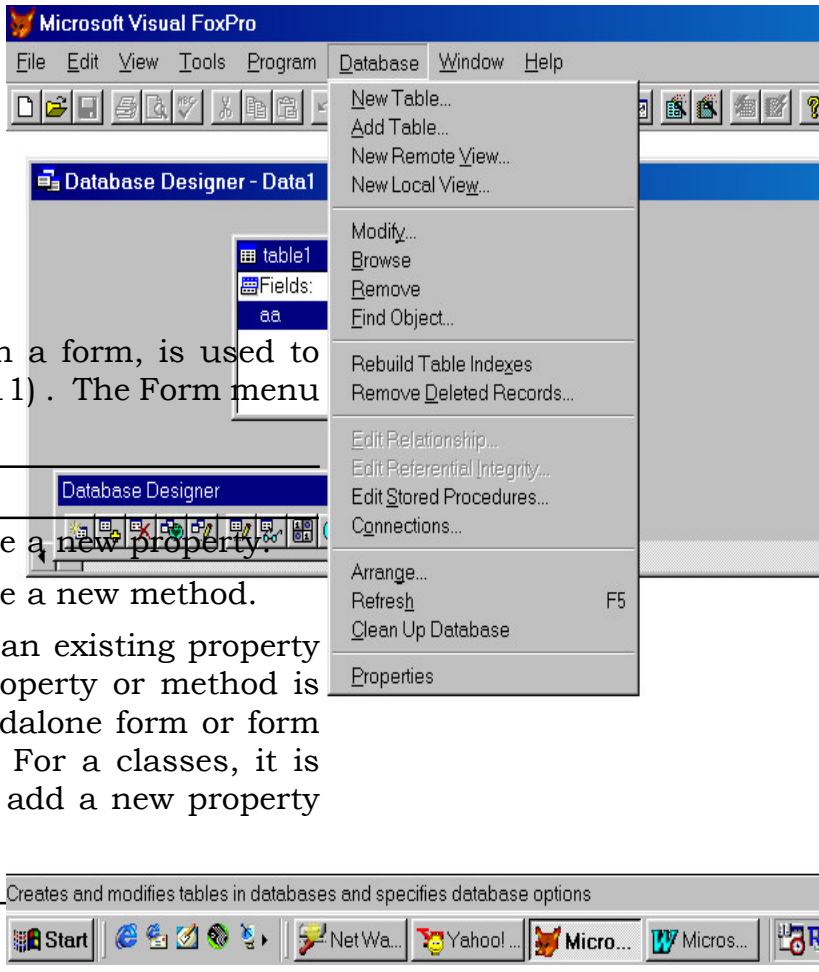


Fig. 5.10: Database Menu

5.12 FORM MENU

The Form menu that is added when you open a form, is used to create and modify forms (as shown in figure 5.11) . The Form menu options are described as follows:

Option	Description
New Property	Allows you to create a new property.
New Method	Allows you to create a new method.
Edit Property / Method	Allows you to edit an existing property or method. The property or method is scoped to the standalone form or form set, as applicable. For a classes, it is enabled when you add a new property or method.

Include File Allow you to specify a reader file of predefined compile-time constants for a user-defined class, form, or form set.

Quick Form Allows you to display the Form Builder dialog box, which helps you to create a simple form that you can customize by adding your own controls.

Create Form Allows you to display a new form set, which is a parent container for one or more forms.

Remove Form Set Allows you to remove an existing form set. This is applicable only if you have created a form set.

Run Form Allows you to run a form after you have designed and saved it.

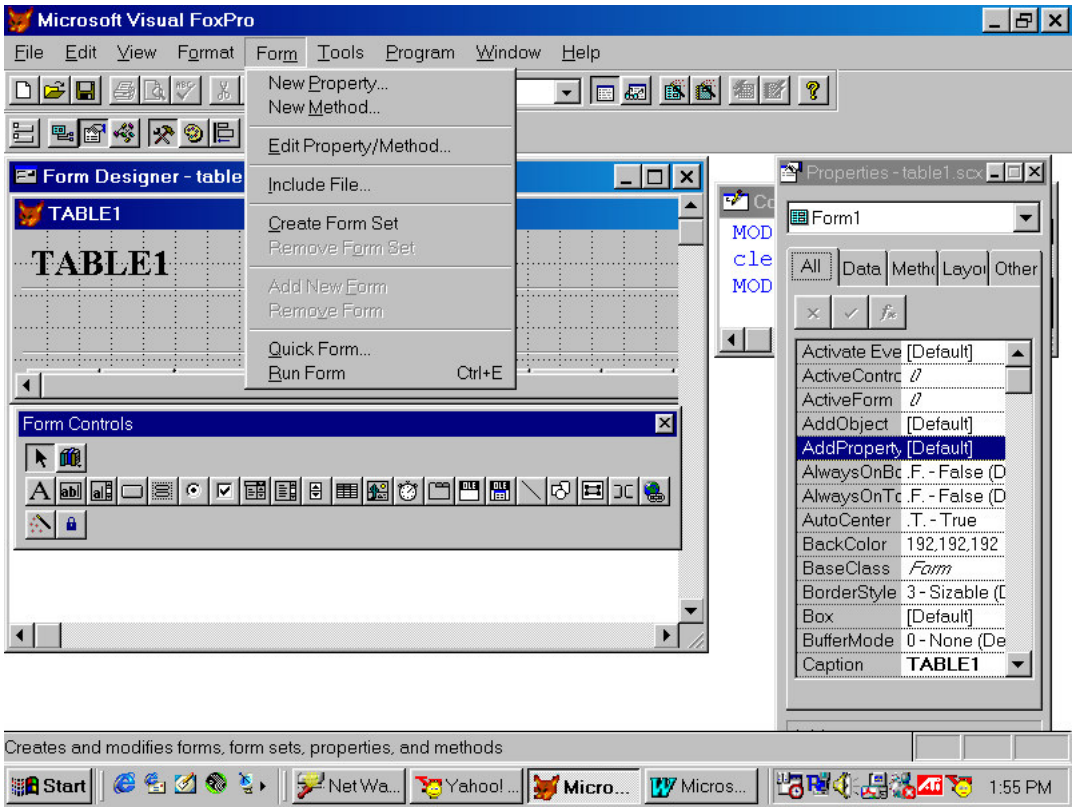


Fig. 5.11: Form Menu

5.13 QUERY MENU

The Query menu is added when you work with query. The Query menu is used to display selected records and fields of a table in the desired order. A Query provides you to retrieves selected records based on the selection criteria out of a table in a form which is convenient to use. The Query menu contains options that allow you to create, modify, and run queries (as shown in figure 5.12).

The Query menu contains options are described as follows:

Option	Description
Add Table	Allows you to display the Add Table or view dialog box to add a table or view to the designer window.
Remove Table	Allows you to removes the selected table from the top pane of the designer window.
Remove Join	Allows you to remove the selected join line from between tables.
Condition	
Selection Criteria	Allows you to specify the Selection Criteria tab on top in the Query or View Designer window.
Output Fields	Allows you to specify the Fields tab on top in the Query or View Designer window.
Order By	Allows you to specify the Order By tab on top in the Query or view Designer window.
Group By	Allows you to put the Group By tab on the top in the Query or View Designer window.
Update Criteria	Allows you to put the Update Criteria tab on top in the View Designer window.
Query Destination	Allows you to display the Query Destination dialog box. The dialog box allows you to send the results of your query to eight different outputs.
View SQL	Allows you to display the SQL statement your query is building

Advanced Options Allows you to display the Advanced options dialog box, in which you can fine-tune how records are retrieved in a view or how updates are made to the server or source tables.

View Parameters Allows you to set up views that prompt for a value entry to complete the query.

Comments Allows you to display notes or comments that you have written to identify the query or view its purpose.

Run Query To execute the SQL select statement you built, and send the results to the output destination you specified.

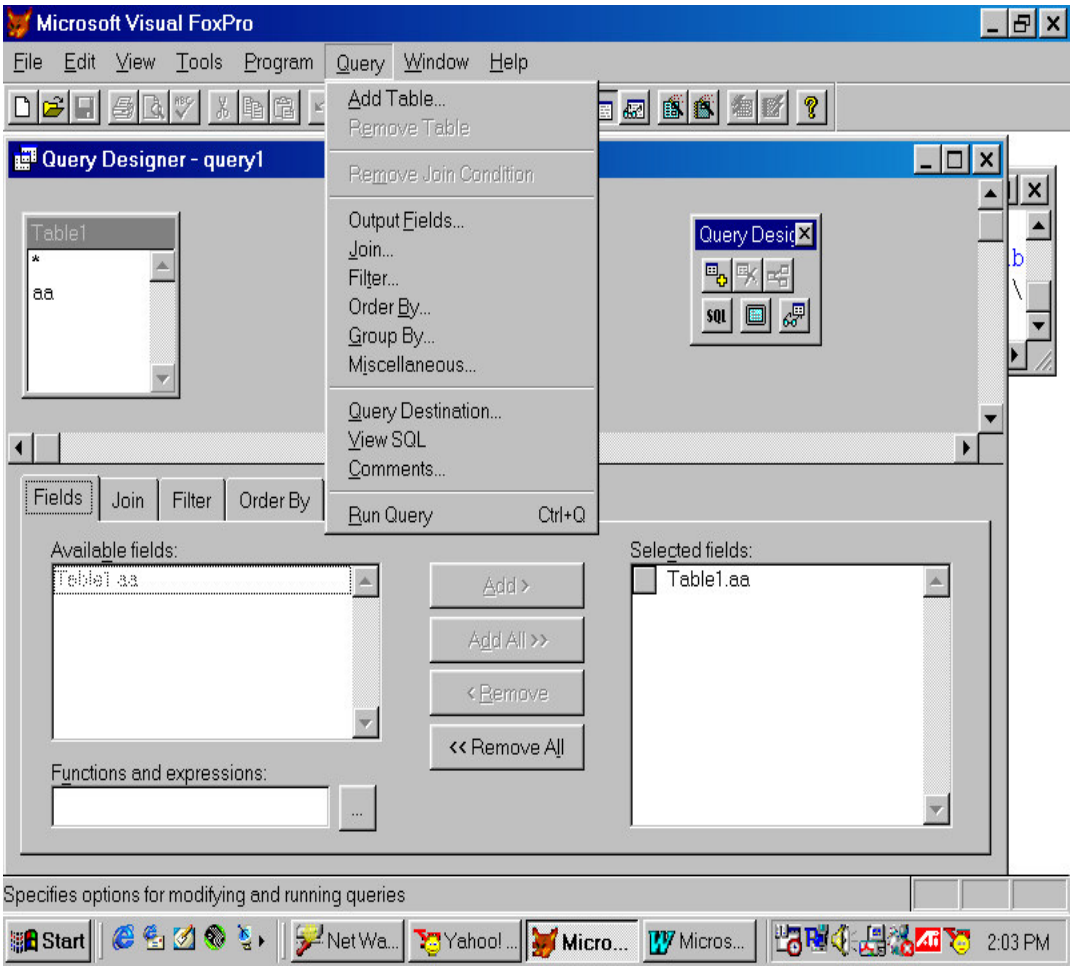


Fig. 5.12: Query Menu

5.14 REPORT MENU

The **Report** menu is added when you are working with the designing of a printed report. This option is used to create, modify and customise reports (as shown in figure 5.13).

The Report Menu contains options are described as follows:

Option	Description
Title/Summary	Allows you to display the Title/Summary dialog box, in which you specify the Title and/ or Summary for the report.
Data Grouping	Allows you to display the Data Grouping dialog box to create data groups and Specify their properties
Variables	Allows you to create memory variables within a report.
Default Font	Allows you to specify the persistent fonts. Font styles, and font sizes for report, Label and field controls.
Private Data Session	Allows you to set the report to have a data session that does not change if you Open or use tables in other designers.
Quick Report	Allows you to place selected fields automatically in an empty Report Designer Window
Run Report	Allows you to display the Print dialog box, to send the report to a printer.

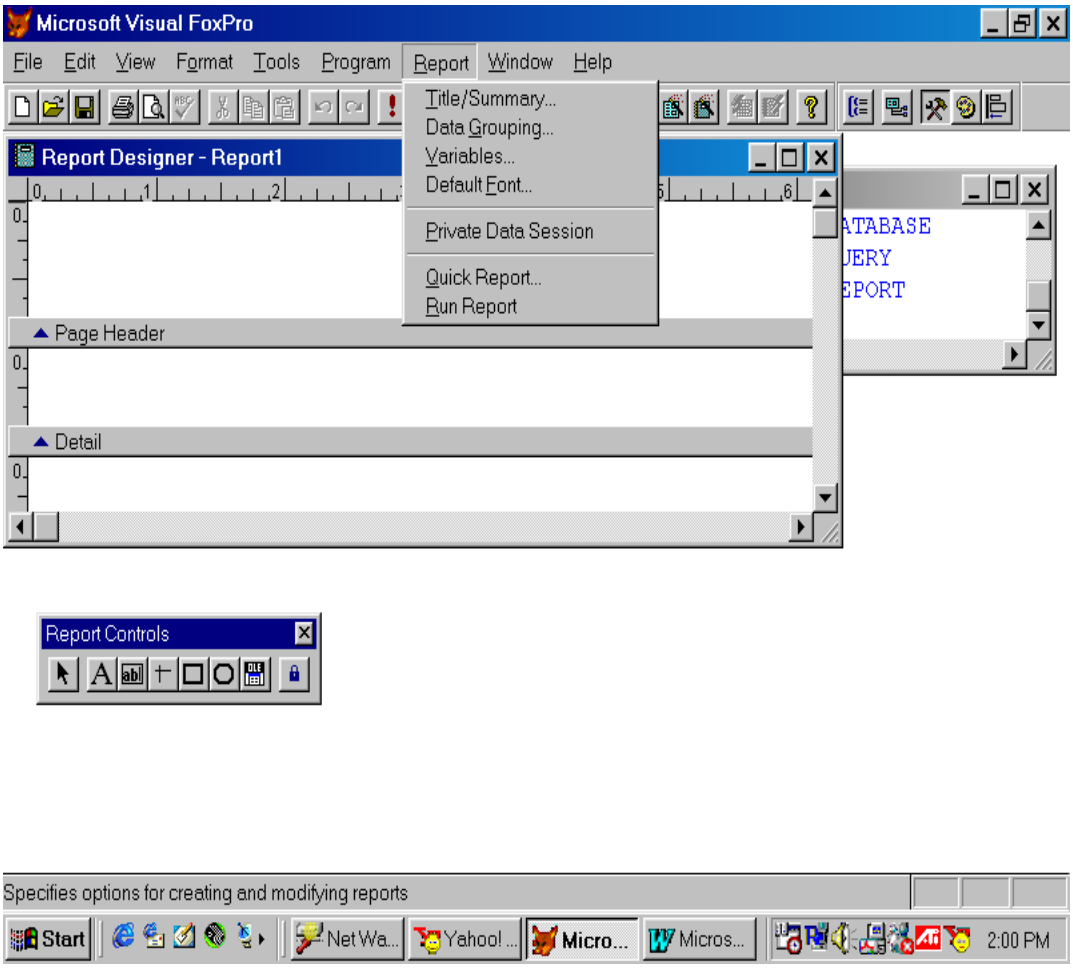


Fig. 5.13: Report Menu

5.15 TABLE MENU

The **Table** menu is added when you open a table and is used for working with data. The **Table** Menu contains options for examining and editing the active (selected) table.

The **Table** menu contains options are described as follows:

Option	Description
Properties	Allows you to open the Work Area Properties dialog box, in which you can modify the structure of a table, select index files and field, and define data filters.
Font	Allows you to open the Font dialog box, in which you can specify the font, font style, and font size that controls the type in a table or view in the Browse or Edit window.
Go To Record	Allows you to positions the record pointer on the record you specify.
Append New Records	Allows you to add record to the end of the current table.
Toggle Deletion Mark	Allows you to place or remove a deletion mark at the beginning of a selected record to mark or unmark it for deletion.
Append Records	Allows you to open the Append From dialog box, to add records to the active table from the another table.
Delete Records	Allows you to open the Delete dialog box, in which you mark records for deletion.
Recall Records	Allows you to open the Recall dialog box, in which you unmark records that are marked for deletion.
Remove Deleted Records	Allows you to permanently remove records that are marked for deletion.
Replace Field	Allows you to change field information in one record or in a range of records.
Size Field	Allows you to change the width of the selected field using the Keyboard.
Move Field	Allows you to move the selected field using the Keyboard.
Resize Partitions	To activate the window splitter so that you can partition the window or change the size of the existing partitions using the keyboard.
Link Partitions	When the Browse window is split, the two partitions are linked and they scroll vertically together as you move to different records and fields. To scroll each partition independently,

	choose the **Link Partitions** from the **Table** menu to remove the check mark(P) to the left of the option. This option is enabled only when Browse window is split.
Change Partitions	Allows you to move from one partition to another. This **Change Partitions** option is enabled only when the Browse window is split.
Rebuild Indexes	Allows you to rebuild any open index files associated with the active table so that they accurately reflect the current status of the table.

5.16 STANDARD TOOLBAR

The **Standard toolbar** contains buttons for performing the most common actions in Visual FoxPro. This toolbar is displayed by default as shown in figure 4.5.

The Standard toolbar contains buttons are described as follows:

Button	Description
New	Allows you to create new files using designers and wizards.
Open	Allows you to open an existing file or create a new file.
Save	Allows you to save changes to the active file.
Print	Allows you to save changes to the active file.
Print Preview	Allows you to show you the results of your work without printing it in WYSIWYG format.
Spelling	Allows you to check spelling. Available when editing text or memo field.
Cut	Allows you to remove selected text, controls, or anything else that is selectable, to the clipboard.
Copy	Allows you to duplicate selected text, controls, or anything else that is selectable.
Paste	Allows you to place cut or copied text, controls, or anything else that is selectable at the insertion point position.

Undo	Allows you to reverse the most recent action.
Redo	Allows you to reverse the most recent Undo command.
Run	Allows you to run a query, form, or report after you have designed and saved it.
Database	Allows you to specify the current database.
Command Window	Allows you to show commands as they are executed, and provides space for typing commands.
View Window	Allows you to provide an easy way to open tables, establish relations, and set work area properties.
Form Wizard	Allows you to run a Visual FoxPro form wizard.
Report Wizard	Allows you to run a Visual FoxPro report wizard.
AutoForm Wizard	Allows you to create a form without using a wizard.
AutoReport Wizard	Allows you to create a report without using a wizard.
Help	Allows you to display online Help.

5.17 WHAT HAVE YOU LEARNT

In this lesson you have learned about the various options of the Visual FoxPro Menu Systems. Now you are in a position to the functioning of additional Menu System such as Database Menu, Form Menu, Query Menu, Report Menu and Table Menu.

5.18 TERMINAL QUESTIONS

1. Explain the concept of OLE.
2. Define Database Schema.
3. What is a Query ?
4. Explain Query Menu.

5.19 KEY TO INTEXT QUESTIONS

1. (b) 2. (c) 3. (d)

6

CREATING TABLES AND VIEWING DATA

6.1 INTRODUCTION

The database is a concept of file organization. It consists of related files stored together so that groups of data items can be easily retrieved. Visual FoxPro also consists of related files organized in such a way that the user accesses groups of data easily. In Visual FoxPro the data and information is stored in the form of Table. More than one table can be created in a single Visual FoxPro database with different group of data items. Visual FoxPro allows the data to be stored, maintained, manipulated and retrieved.

6.2 OBJECTIVES

After going through this lesson you will be able to

- create a table with table Wizard
- create a table with table designer
- open an existing table
- append, edit and delete records in the table
- index and sort data

6.3 CREATING TABLE WITH TABLE WIZARD

There are two ways of creating a Table using:

(i) Table Wizard

(ii) Table Designer

The Table Wizard allows you to create new tables based on any of the group of typical standard tables that are in-built with Visual FoxPro. The Table Wizard is a useful shortcut, which helps you in creating your own table. For many applications, you need to create a new table from the scratch, which becomes more easier using Wizard. In this section, we will concentrate on creating table with Table Wizard.

To create a new table by using Table Wizard, do the following:

Choose **New...** option from the **File** menu. Cerate a new database for a specific application so that all the related components pertaining to the application can be stored in that database. The **New** dialog box for creating a fresh Table is shown in Fig. 6.1.

Fig. 6.1: Dialog Box for creating new components

Select the **Table** radio button, and then choose **Wizard.** The Table Wizard dialog box appears as shown in Fig. 6.2.

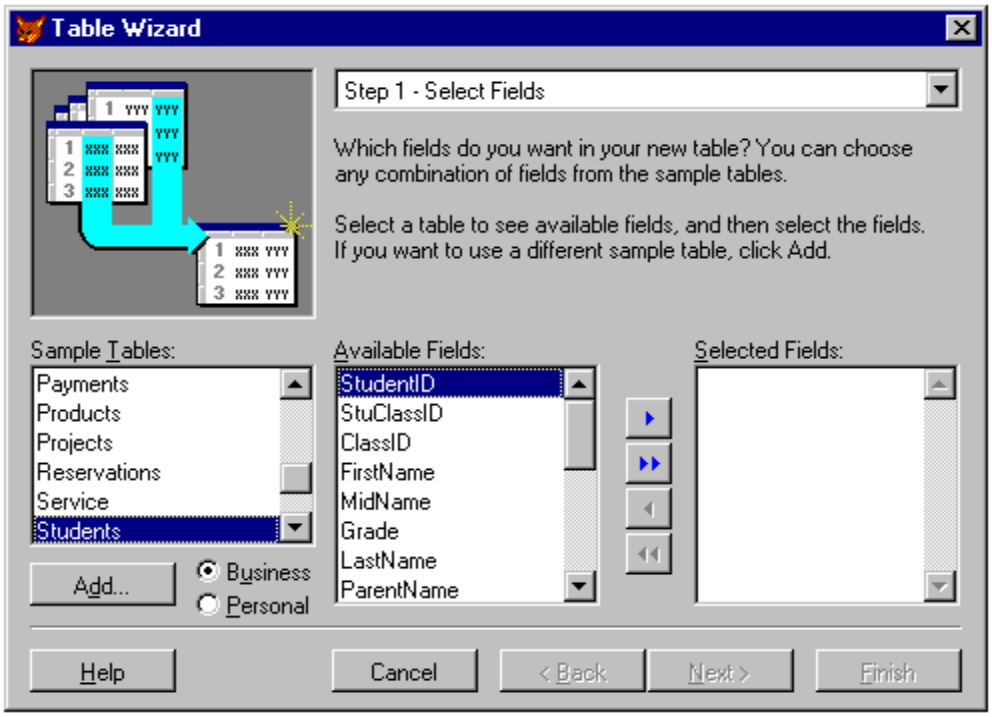


Fig. 6.2: step 1 of the Table Wizard dialog box

The Table Wizard dialog box displays a list of in-built available standard tables designed to store related data Payments, Products, projects, Reservations, Students and many other types of tables as shown in the figure 6.2. You can select any one of the table according to your requirement from the sample Table lists. For example ***Students*** Table is considered as example in rest of the lessons.

6.3.1 Selecting Table Fields

The fields available with the selected table will be displayed in the **Available Fields** list. You can select the fields to include in your new table by using the Field Picker to move them from the Available Fields list to the Selected Fields list as shown in the Fig. 6.3

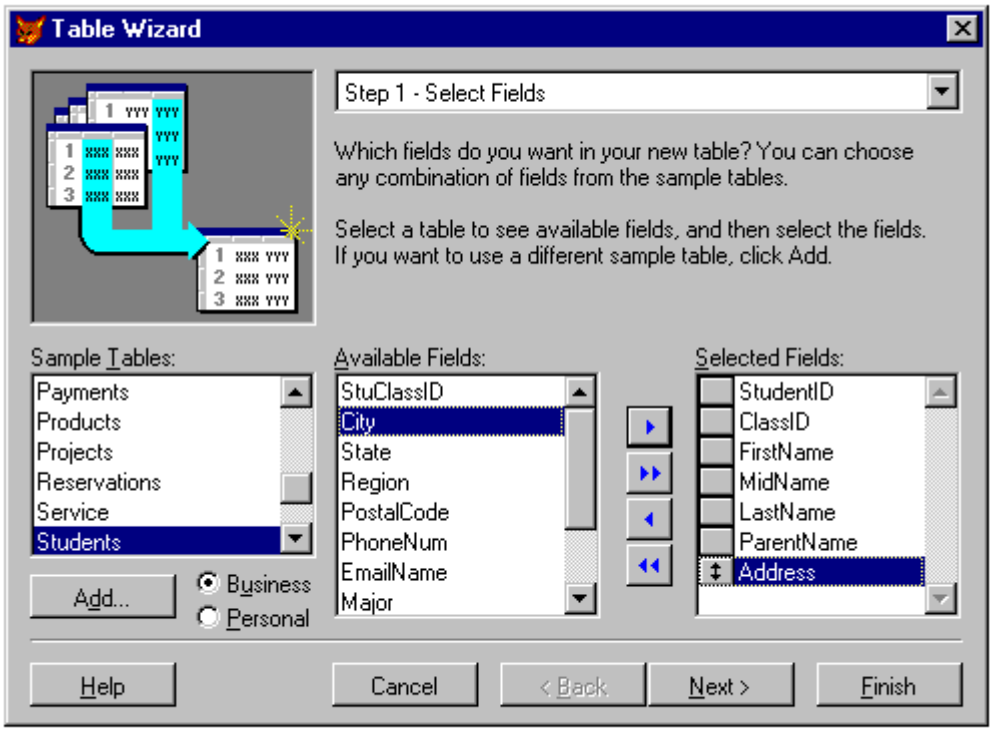


Fig. 6.3: Step 1 of the Table Wizard dialog box

- Select the required field from the Available Fields list and then click the [▶] push button to add the selected field into the Selected Fields list. You can also move the required field from the Available Fields list to the Selected Fields list by simply double clicking in it.
- To add all the fields from the Available Fields to Selected Fields list, click [▶▶] Push button.
- Select the field from the Selected Fields list and then click [◀] push button to remove it back from the Selected Fields list to the Available Fields list.
- To remove all the fields from the Selected Fields list back to Available Fields list, click [◀◀] push button.

To change the order of an existing field to a different location in the table, click the double-headed arrow  to the left of the field you want to move and drag the field to a new location.

You can also move an existing field by using the Keyboard. Press **<Tab>** Key until the desired field is selected, then hold down the **<Ctrl>** key and then press the **Up** and **Down** arrow key to drag the field to its new location.

Click **Next** button, once you are through with the selecting table and table fields. In this case we selected the **Student Table** and its available fields

6.3.2 Selecting a Database

The step 1a of the Table Wizard dialog box as shown in Fig. 6.4 allows you to select a database in which you wish to add your new table. You can create the Database as a stand alone free table. You can also change the name of the table from what was available in the Table Wizard from **Student** to **School.**

If a database is open you may add your new table in the existing database.

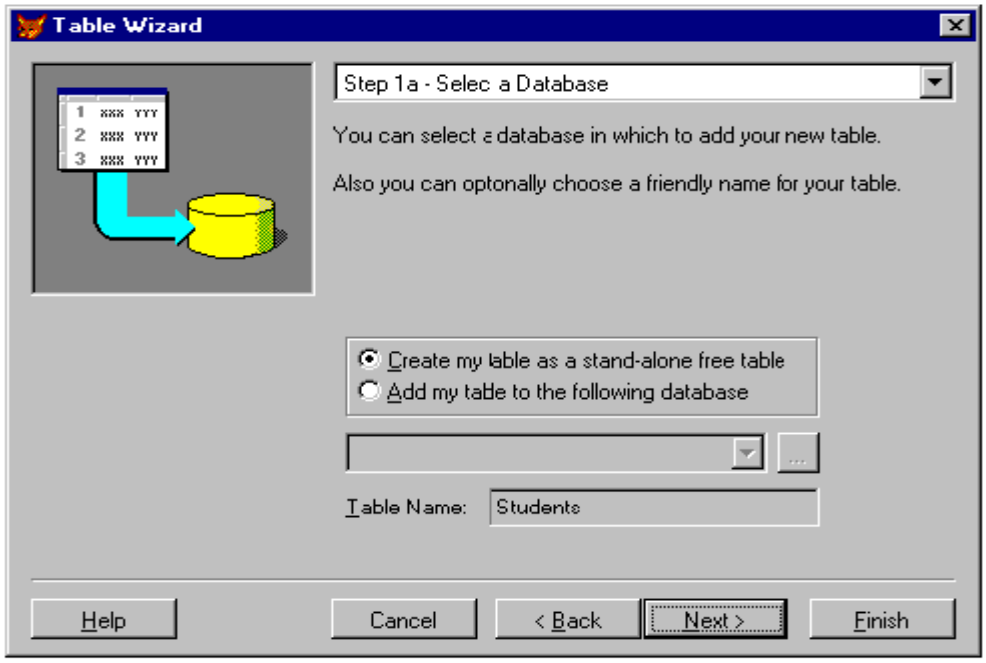


Fig. 6.4: Step 1a of the Table Wizard dialog box

6.3.3 Modifying Table Fields Settings

After selecting the required table and the table fields from the Table Wizard step 1 dialog box, you click **Next** push button and then the step 2 of Table Wizard dialog box appears as shown in Fig. 6.5, which allows you to modify the field settings.

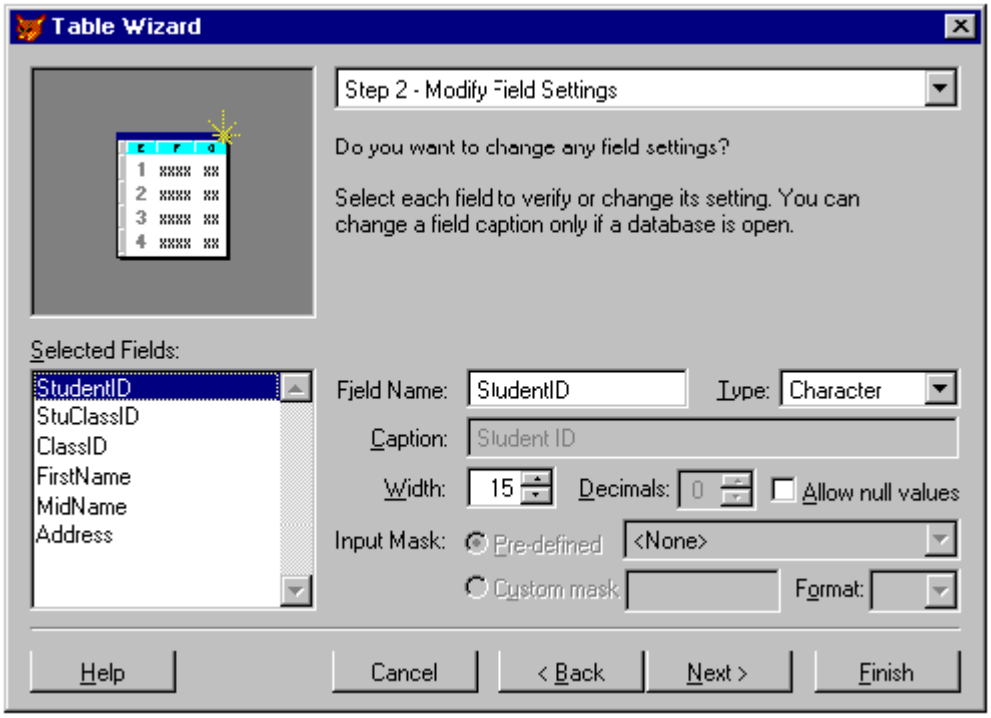


Fig. 6.5: step 2 of the Table Wizard Dialog box

This dialog box allows you to change the Field Name, Caption, Type and other attributes of the field, according to your requirement. To modify a field, select it and specify the settings you want.

6.3.4 Indexing Table Field

An Index is an arrangement of displaying records in a certain order of the datafield so that records can be accessed very fast. The step 3 of the Table Wizard dialog box as shown in figure 6.6 allows you to select fields that you have included to be used as the basis of an index, which can be used to determine the order in which the table's records are displayed.

If a database is open, you may select one field on the primary index

key. A primary index key contains a value that is used to uniquely identify each record.

You can also select the check boxes to create other indexes.

- To create an index tag for the table based on that field, which lets you control the order in which the records are displayed, click the check box to the left of the field name. An up arrow occurs, indicating an index in ascending order.
- Click the check box button again, if you want the index in descending order.
- To remove an index tag for a selected field, click the check box to the left of the field until the button is blank.
- To create a primary key with multiple fields select more than one field.

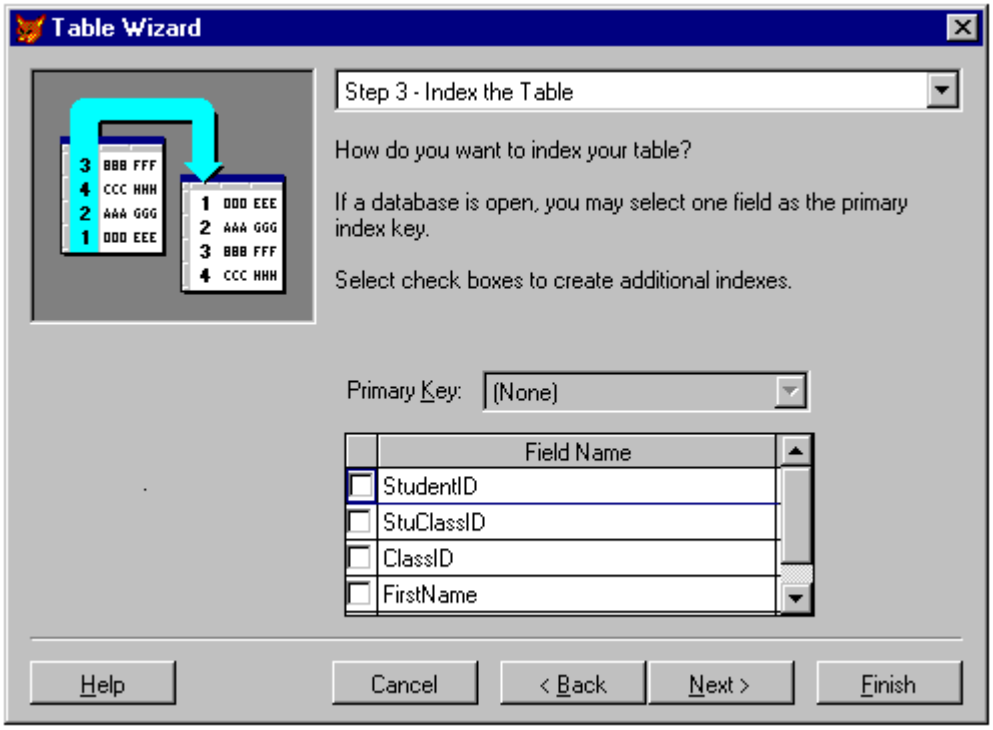


Fig. 6.6: Step 3 of the Table Wizard dialog box

6.4 SAVING TABLE

Click **Next** push button, after creating an index tag for your table.

The step 4 of the Table Wizard dialog box as shown in the Fig. 6.7 allows you to save the table for later use; save table and browse it; save table and modify it in the Table Designer.

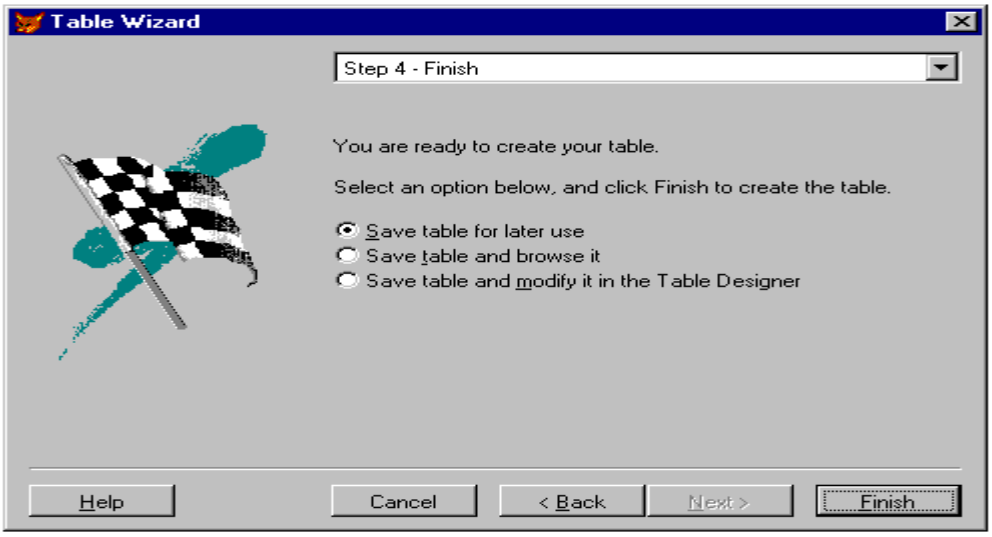


Fig. 6.7: step 4 of the Table Wizard dialog box

Choose **Finish** push button. The **Save As** dialog box appears as shown in the figure 6.8. You need to specify the directory where the new file will be stored and give a file name by typing a file name in the Name Text box and choose **Save** push button. Be sure that Save As type should be same as shown in figure 6.8.

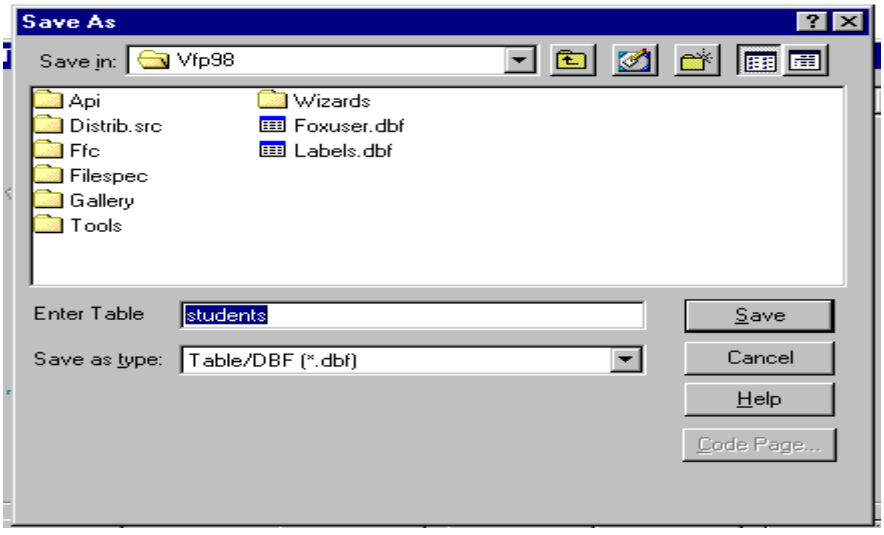


Fig. 6.8: Save As Dialog box

INTEXT QUESTIONS

1. What do you understand by Indexing?

2. In the Step 1 of creating a table with Table Wizard, you select

   (a) Table and Fields (b) Table and Records

   (c) Fields and Records (d) All the above

3. Primary Index Key in a table is always

   (a) many (b) unique (c) double (d) null

4. True or False

   (a) You can create a table only with Table Wizard.

   (b) You can change the field settings in a table created with Table Wizard.

6.5 CREATING A TABLE WITH TABLE DESIGNER

To create a new table from scratch, by using Table Designer. Choose **New...** option from the **File** menu. The **New** dialog box appears as shown in the figure 6.9. Select the **Table** button and then choose **New File**. The **Create** dialog box appears as shown in the figure 6.9. This table is identical to the **Save As** dialog box and is used in the same way to specify the directory and name of the table.

You can also invoke the Create dialog box by typing the command **CREATE** in the command window and then press **<Enter>** key.

For example, in this lesson, we will create a sample student details Table and name it STUDENT and it will be stored automatically with the extension DBF i.e. STUDENT.DBF (Figure 6.9)

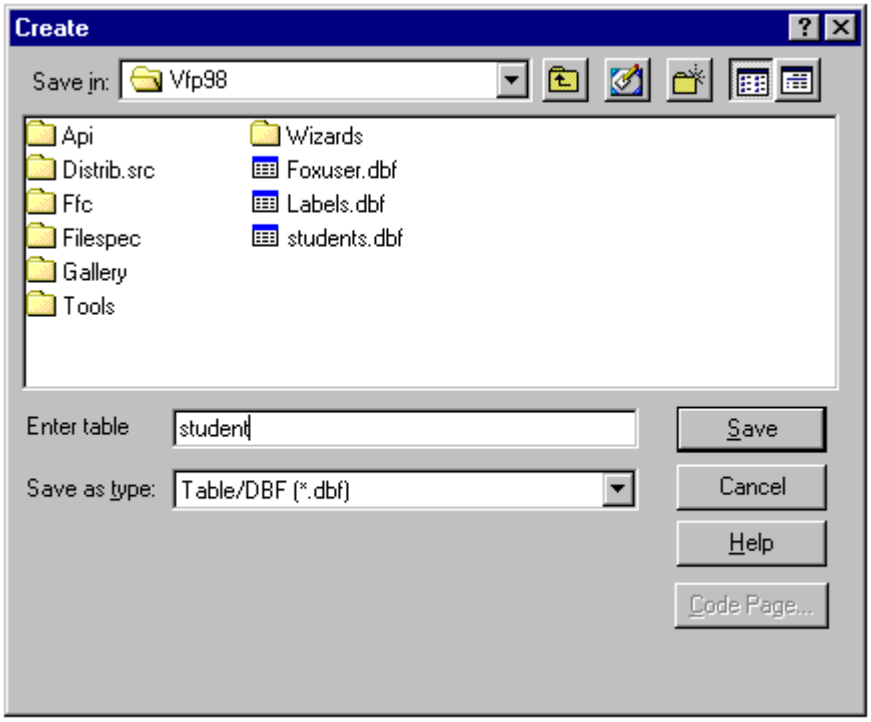


Fig. 6.9: Create Dialog box

Click the Save button, an empty Table Designer dialog box appears as shown in the figure 6.10.

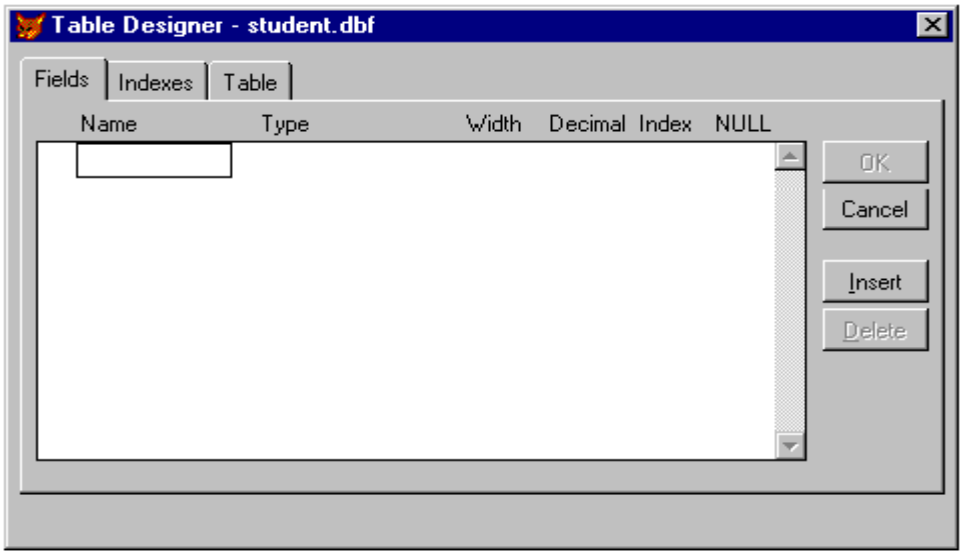


Fig. 6.10: An empty Table Designer Dialog box

You can also invoke the empty Table Designer dialog box directly by typing the command **CREATE** <table name> in the command window and then press **<Enter>** key as shown in the figure 6.11

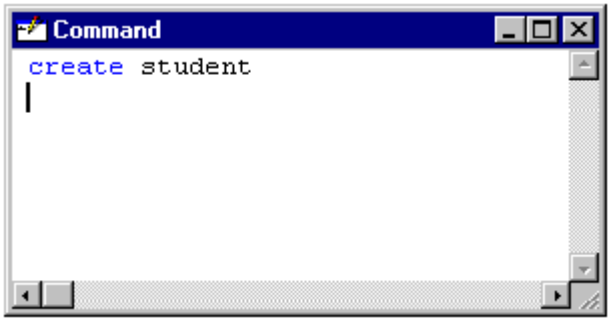


Fig. 6.11: Invoking Table Designer

6.5.1 Defining Table Fields

To define each field in the table using the **Table Designer** dialog box, you should follow these steps:

1. Type a field name (which can be up to 10 characters long in a text) below the name heading. You must note that spaces are not allowed in the field name. The only punctuation allowed is the underscores.

2. Open the **Type** popup and choose one of the data type from the list by clicking on it. The following data types are available on the Type popup.

Data Type	Description
Character	The character fields can be used to store letters, numbers, special symbols, and blank spaces. The maximum field width of a character field is 254. The default width is 10.
Currency	The currency fields can be used to store money values. It holds amount from over 900 trillion to less than -900 trillion. The default width of currency data is 8.
Numeric	The numeric fields can be used to store numbers on which calculation can be performed. The maximum field width of a numeric field is 20 including an optional +/- sign and decimal place. The default width of numeric data is 8.

Float The float field is specifically designed for scientific data. There is no difference between the numeric and float data types. The maximum field width of a float field is 20 including an optional +/- sign and decimal place. The default width of float data is 8.

Date The date fields are used to store dates. The field width of a date type field is always 8. The default date entry and display format is mm/dd/yy. This can be modified to other country formats by using the SET command.

DateTime The DateTime data type can store either a date such as 09/23/02, a time such as 06:35:45 AM, or a date and time such as 09/23/02 06:35:45 AM. The field width of a DateTime type field is always 14.

Double The double data type stores numeric data, but it does calculations with more accuracy than the Numeric data type.

Logical The logical fields accept only single character representing True/False values. Logical True (.T.) can be entered as T, t, Y, or y (yes) and Logical False can be entered as F, f, N, n (no).

Memo The memo fields may be used to store any printable ASCII character (letters, number, blank spaces or special symbols). All memo fields in a database file are stored in a database memo file with extension .dbt which is created while setting up the database file. Memo fields are indicated in the database (.dbf) file by the word memo. Memo field's sizes are limited by the amount of memory available. Each memo field occupies only 10 spaces in the table.

General The general fields are used to store OLE (Object Linking and Embedding) objects. A width of 10 characters is automatically assigned. The general field's sizes are also limited by the amount of memory available.

Character (Binary) The Visual FoxPro's Binary Character data may

include any 8 bit value, including the null character. The maximum field width of a binary character field is 254.

Memo (Binary) The Visual FoxPro's Binary Memo fields can be used to hold types of data that could never before be manipulated with database management programs, such as scanned images or digitized sound.

3. If the data type you have chosen has a variable field width, a spinner occurs under the heading **Width**. You can change the default value by entering a width in the **Width** text box or by clicking the spinner control.

4. For numeric, double or float data type, you have to specify a value under the heading **Dec** for the number of decimal places the field should have. You can enter the number by typing in the text box or by using the spinner control.

5. The **Null** column determines whether you can enter a **NULL** value in a field.

Based on these details you can create your own table structure named "SCHOOL" by typing the following fields in the Table Designer dialog box as shown in the figure 6.12.

Field	Name	Type	Width	Dec	Nulls
1	ENROL	Character	9		
2	NAME	Character	25		
3	FNAME	Character	25		
4	COURSE	Character	1		
5	ADD1	Character	25		
6	ADD2	Character	25		
7	ADD3	Character	25		
8	PIN	Numeric	6		

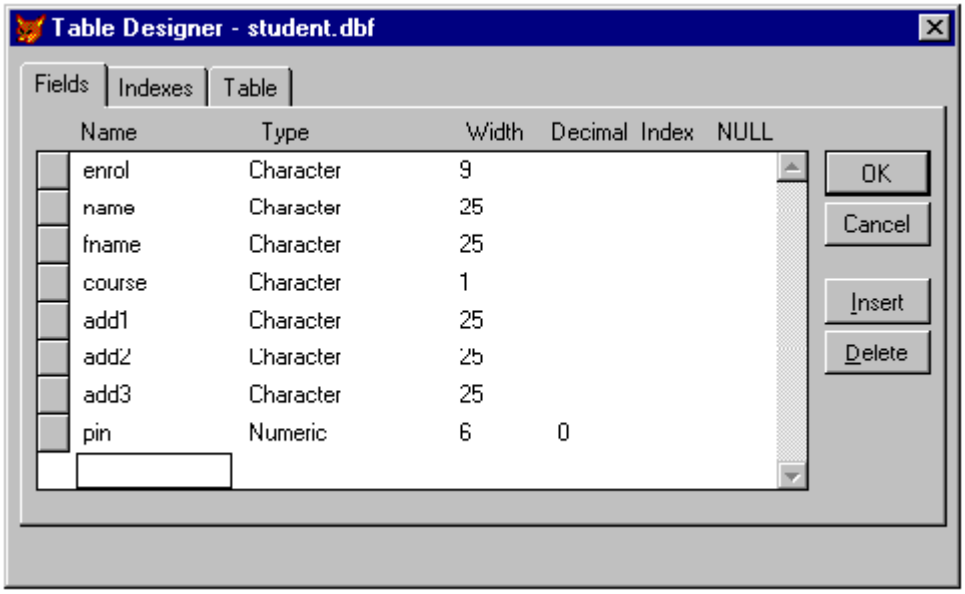


Fig. 6.12: Table designer dialog box with fields

After defining all the fields in a table structure dialog box, choose OK button. Visual FoxPro displays a dialog box asking whether you want to "Input data records now?" as shown in the figure 6.12.

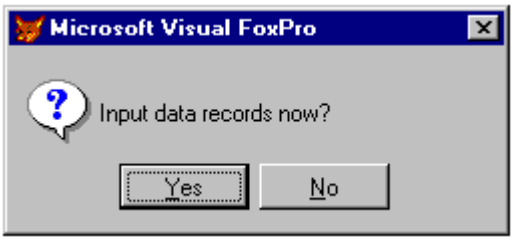


Fig. 6.13: Input data records now?

Choose **No**, if you don't want to input data now. Choose **Yes** to open a Browse window for the table in Append mode, so you can right away begin entering data into the table.

6.5.2 Inserting Table Field

To insert a new field in an existing table in the **Table Designer** dialog box, position the cursor in an existing field and choose the **Insert** push button. A new field appears with the name NEWFIELD with a character type and a default width of 10 characters is added before the selected field. Modify these as per your requirement.

6.5.3 Deleting Table Field

To remove an existing field from the **Table Designer** dialog box, position the cursor to that field and choose the **Delete** push button. The selected field is deleted and the fields below the deleted field move up one position in the Table Designer dialog box.

6.6 OPENING A TABLE

Once you have created a table, you can open it at any time by using any one of the following techniques:

- Type **Use <table name>** in the command window and press **<Return>** key. The name of the table, which you have specified with the USE command, is displayed at the left of the status bar. The following command will open the table STUDENT already created.

  Use STUDENT↵

- Choose **Open** option from the **File** menu. Select the directory and name of the table and choose OK. The name of the table is displayed at the left of the status bar. The status bar also indicates that it has no records, and that it is opened exclusively, so it is not accessible to other user on a network.

6.7 APPENDING RECORDS / DATA

After opening the existing table named "STUDENT", you can append records in the table by using any one of the following techniques:

- Choose **Append Mode** option from View menu. A new blank record is added to end of the table to enter the data in the table named "STUDENT".

- Type **append** in the command window and press <Return> key. A new blank record is added to end of the table to enter the data in the table named "STUDENT". The figure 6.14 shows the example of how to enter data in the Append mode.

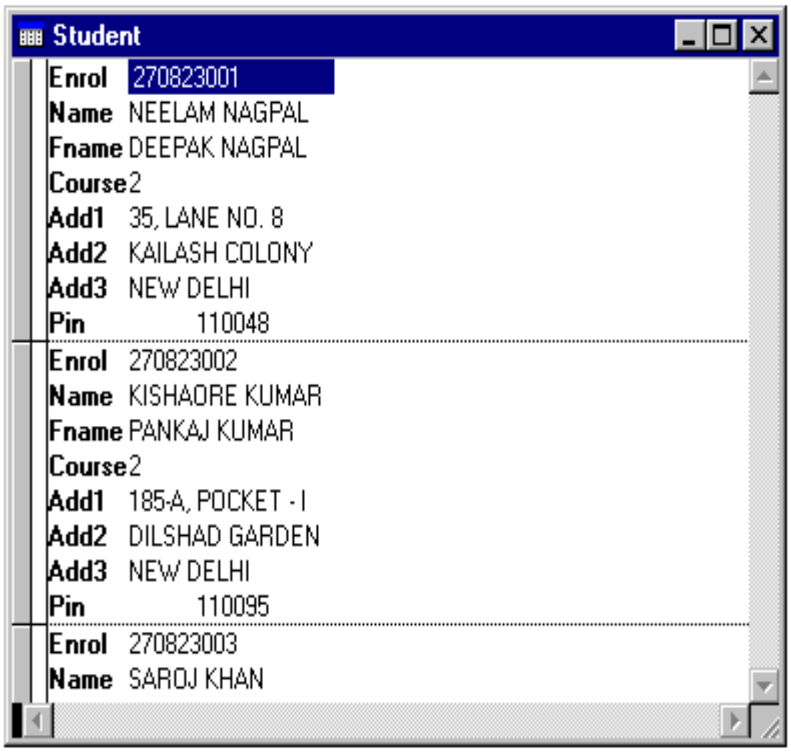


Fig. 6.14: Entering data in Append Mode

Now you can enter the following records, by typing it into each field and pressing <Return> to move to the next field:

Field	Values	Field	Values
ENROL	270823001	ENROL	270823002
NAME	NEELAM NAGPAL	NAME	KISHORE KUMAR
FNAME	DEEPAK NAGPAL	FNAME	PANKAJ KUMAR
COURSE	2	COURSE	2
ADD1	35,LANE NO.8	ADD1	185-A, POCKET -I
ADD2	KAILASH COLONY	ADD2	DILSHAD GARDEN
ADD3	NEW DELHI	ADD3	NEW DELHI
PIN	110048	PIN	110095
ENROL	270823003	ENROL	270823004
NAME	SAROJ KHAN	NAME	MUKESH KUMAR
FNAME	MOHAMMAD KHAN	FNAME	PANKAJ KUMAR
COURSE	2	COURSE	2
ADD1	35,LANE NO.10	ADD1	18-A, POCKET -D
ADD2	SHAHADARA	ADD2	MAYUR VIHAR
ADD3	NEW DELHI	ADD3	NEW DELHI
PIN	110032	PIN	110093

ENROL	270823005	ENROL	270823006
NAME	POONAM BANSAL	NAME	NARENDRA GOYAL
FNAME	RAKESH BANSAL	FNAME	PRAKASH GOYAL
COURSE	3	COURSE	3
ADD1	123-B, POCKET - P	ADD1	416/7 SECTOR - 19
ADD2	MALVIYA NAGAR	ADD2	VASUNDHARA ENCLAVE
ADD3	NEW DELHI	ADD3	NEW DELHI
PIN	110045	PIN	110096

After you enter the records, you can save the table with the entered data by pressing **<Ctrl> + W** or **<Ctrl> + <End>** key.

INTEXT QUESTIONS

5. Write True or False for the following statements.
   (a) The general fields are used to store objects
   (b) The field name in a Table can have space also.
   (c) You can not Insert or delete a field in a Table.
   (d) The Logical fields accept only single character.
6. The maximum length of a field name can be of
   (a) 8chr (b) 10chr (c) 15chr (d) 20chr
7. The maximum field width of a Character field is
   (a) 32 (b) 64 (c) 128 (d) 254
8. Write steps for inserting a column 'Date-of-Birthe' in Students Table

6.8 EDITING RECORDS / DATA

To change or edit data in the table, you need to first open the table by using any one of the techniques mentioned above at section 6.6. After that, choose **Browse STUDENT** from the view menu to display the data. You can also invoke the edit mode by typing **edit** in the command window. A Browse window in Edit mode occurs to edit the data, as shown in the figure 6.15. Remember that the Browse window in the edit mode allows you only to edit data.

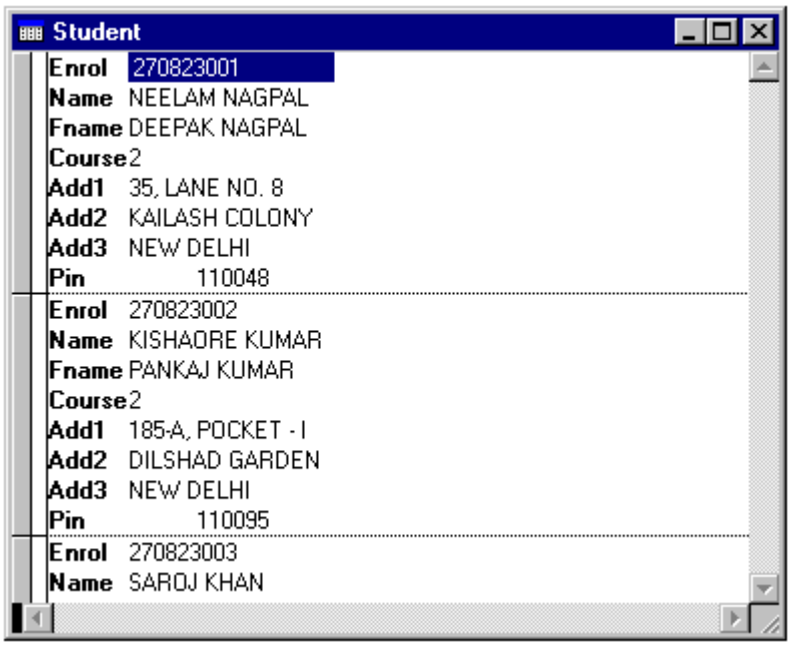


Fig. 6.15: Browse window in Edit mode

6.8.1 Moving to the specific record

If you are having a large number of records in a table, then it will take a long time to scroll. Visual FoxPro provides you a couple of shortcuts to move through the table more quickly.

Choose **Go to Record** option from the **Table** menu to display the submenu as shown in the figure 6.16. The option available with the **Go to Records** options are described as follows:

Option	Description
Top	Allows you to move to the first record of the table
Bottom	Allows you to move to the last record of the table
Next	Allows you to move to the record next to the current record. It is affected by the active index.
Previous	Allows you to move to the record preceding the current record. It is affected by the active index.
Record #	Allows you to bring up the **Go to Record** box, as shown in the figure 6.16(a). You can either type

the record number or use the spin controller and then choose **Go to** push button.

Locate... Allows you to bring up the Expression builder dialog box to define the Locate Record for expression.

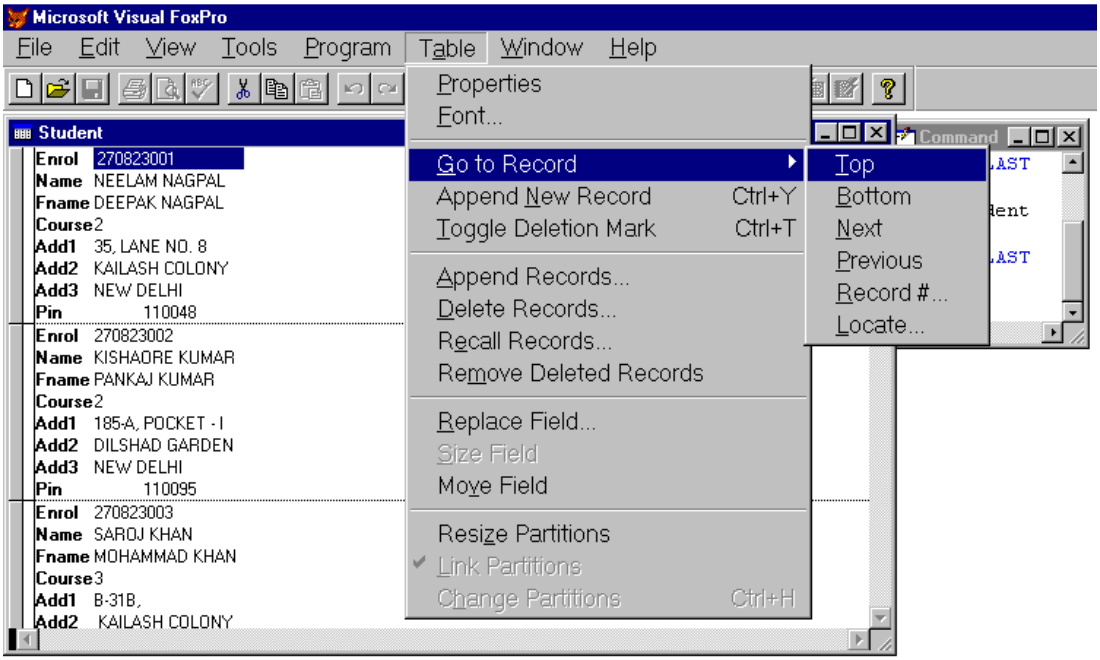


Fig. 6.16: Go to Submenu

Fig. 6.16(a) : Go to Record dialog box

6.8.2 Finding a Specific Text String

It is very difficult to find a specific text string, if you are having a large number of records in a table. Visual FoxPro provides you a shortcut to find a specific text string more quickly in your table.

Choose **Find...** option from the **Edit** menu, Visual FoxPro displays

the **Find** dialog box as shown in Fig. 6.17. For example, enter the data "BANSAL" you are searching for in the **Look For** text box. Click the **Find Next** push button from the find dialog

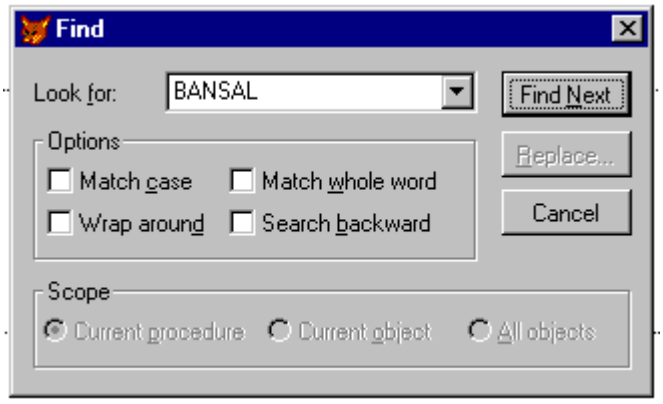


Fig. 6.17: Find dialog box

box. If the first value you find is not the one you required, continue this process until you get the required one. Then click **Cancel** push button to close the **Find** dialog box.

Click the **Replace** push button from the **Find** dialog box to enter a value that will replace the found value.

6.8.3 Deleting Record

To delete a record just click the deletion mark box as shown in figure 6.17 or choose **Toggle Deletion Mark** option from the **Table** menu. Alternatively, you can press <Ctrl> + T to put the mark of deletion at the current record.

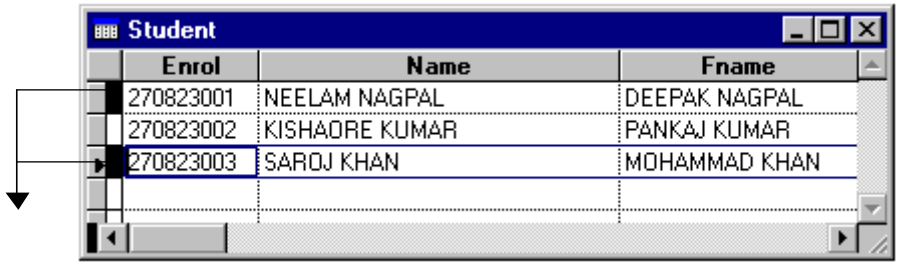


Fig. 6.18: Deletion mark put at the 1st and 3rd record

A filled in deletion mark indicates that the record will be purged during the next **PACK** command or if you choose **Remove Deleted Records** option from the **Table** menu. Repeat the toggle command to undelete a record.

Remember that once you use **PACK** command on a table, those records marked for deletion cannot be retrieved and is physically removed from the hard disk.

6.9 INDEXING DATA

A separate index file specifies the logical order of the records in a table. Records are displayed and accessed in the order specified by the controlling index expression in the index file. There are two types of index files in Visual FoxPro:

1. Compound Index File
2. Single-Entry Index File

6.9.1 Compound Index File

The compound index file contains multiple index entries or tags. You can have as many indexes or tags as you require depending upon the available disk space. When the file is opened, all indexes in a compound index file are available. The name of the compound index file is same as the table name with the extension **.CDX**

6.9.2 Single -Entry Index File

The single entry index file contains a single entry. You must open this index file and make it active for it to be updated with a table.

For example you can type the following in the command window:

```
Use student
Index on enrol to student
Use student index student
```

A table can have a number of single-entry index files. The name of the compound index file is same as the table name with the extension **.IDX**

6.9.3 Creating Indexes

To create index for STUDENT.DBF, go back to the Table Designer dialog box and click the index tab to switch to the index definition page as shown in the figure 6.19.

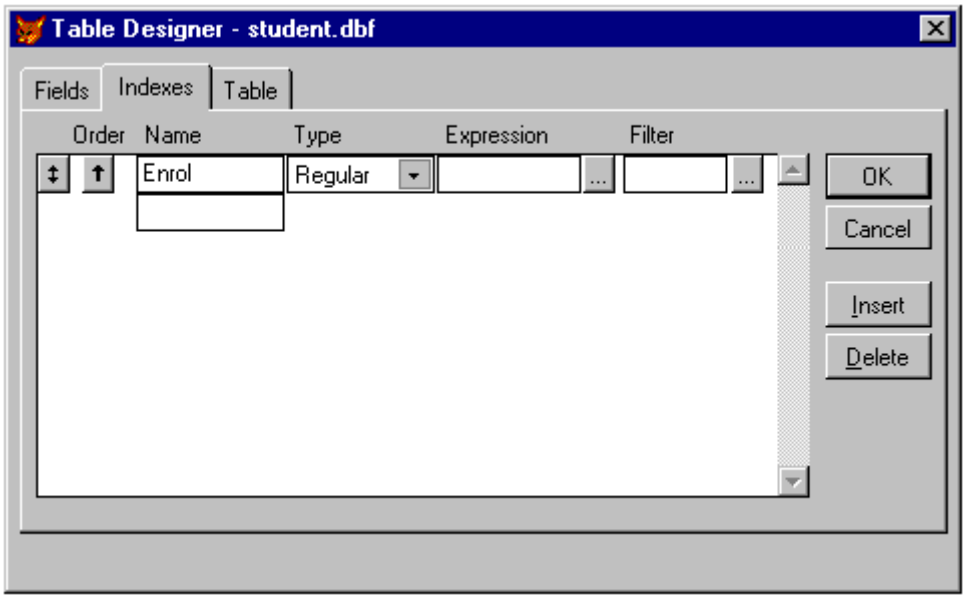


Fig. 6.19: Index definition page

The index definition began with the tag **Name** on the left, followed by the index **Type**, the tag **Expression** and a **Filter.** To enter a new tag, enter the name for the tag in the Name column. The name should describe the index. If your index is based on a single field, it is generally easiest to give it a name that is same as the field name. The arrows to the left of the names indicate whether the index is in ascending (up arrow) or descending (down arrow) order. To change the direction select the row and click the button that appears beneath the arrow. In the beginning, you accept the default index type, which is Regular. You can also make Expressions and use Filters while defining indexes.

6.10 DEFINING ORDER IN A TABLE

The records cannot be entered in a sorted order as you do not receive the data in sorted order. So you have to enter the records in a random order, but you want to view them sorted by one or more fields.

You can use the **SORT** command to reorder the records in a table. **SORT** takes an existing table and creates a new table sorted by a field or a combination of fields. For example,

```
USE STUDENT
SORT ON NAME TO STUNAME <Ascending/Descending>
```

Creates a new table named "STUNAME" sorted by the student name. Normally you do not use the SORT command as it takes more disk space by creating new table. Indexes provides you a more effective way to view and retrieve data from a table in an orderly manner. Because a table can have more than one index, you can define different indexes for different views or reports.

6.11 WHAT YOU HAVE LEARNT

In this lesson you have learned about how to create a table by using Table Wizard and Table Designer. We also discussed about modifying the structure of an existing table in terms of field's name, its type and width. You also learned how to create an Index in a table and sort a data table.

6.12 TERMINAL QUESTIONS

1. Differenciate between Sort and Index?
2. Write the Syntax for Indexing a Table.
3. What do you understand by compound indexing?
4. Explain Null Values.
5. How to add Primary Index to a table field?
6. Write steps for Inserting a new record, editing an existing record and moving to a particular record using the table Students.

6.13 KEY TO INTEXT QUESTIONS

1. An Indexing is an arrangement of the value to search a data value fast.

2 (a)

3. (b)
4. (a) F
   (b) F
5. (a) T (b) F (c) F (d) T
6. (b)
7. (d)
8. Open the Table 'Students'. In View menu, choose Table Designer. Type filed name as 'Date-of-Birth' and Type as Character and Width 8.

7

CREATING QUERY WITH QUERY WIZARD AND QUERY DESIGNER

7.1 INTRODUCTION

Query is the most powerful feature of any database. A Query is a set of instructions in a logical order, required to produce the output based on Table(s) in a particular fashion. In FoxPro, as everything is a menu driven, queries can be designed just by clicking some options. Visual FoxPro also provides this feature to the users. It provides several query type depending upon whether you intend to just view data or update it as well. Query can be created based on complex conditions. In visual FoxPro, you can also define queries on local as well as remote data.

7.2 OBJECTIVES

After going through this lesson you would be able to

- create query with Query Wizard
- create query using two Tables
- create query with Query Designer
- view query outputs

7.3 CREATING A QUERY WITH QUERY WIZARD

There are two ways of creating a Query using:

(i) Query Wizard

(ii) Query Designer

To create a query with Query Wizard, choose **New** option from **File** menu or click the **New** tool. The **New** dialog box appears as shown in figure 7.1. Choose **Query** radio button and then click **Wizard** push button.

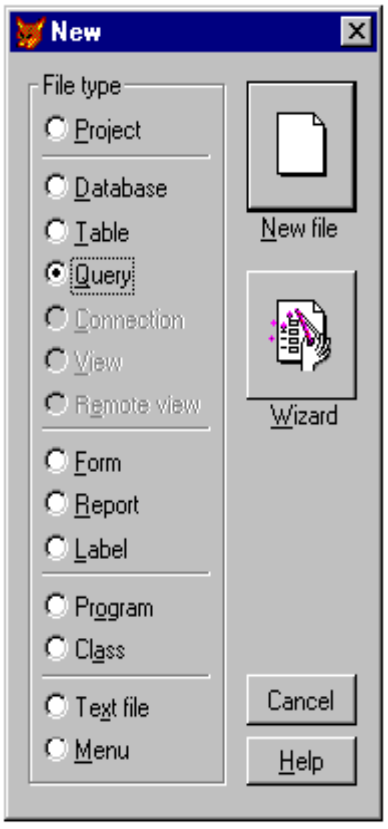


Fig. 7.1: New Dialog box

Visual FoxPro displays the **Wizard Selection** dialog box as shown in figure 7.2. The **Wizard Selection** dialog box provides options for creating several special purpose queries as well as ordinary visual FoxPro queries.

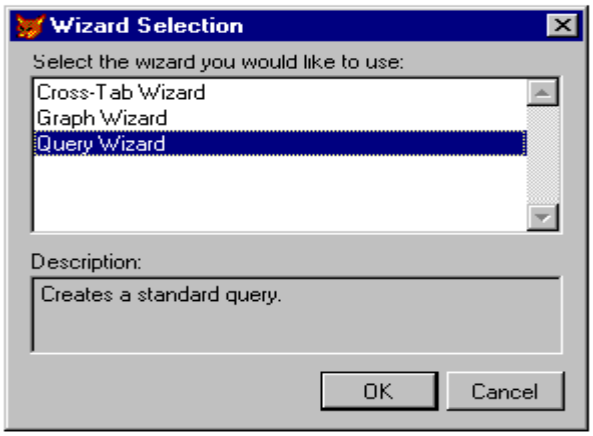


Fig. 7.2: Wizard Selection Dialog Box

The Cross-Tab wizard displays the data in Spreadsheet format. The Graph wizard displays a graph based on data from the FoxPro Tables. Select the **Query Wizard** in the **Wizard Selection** dialog box to display the "**Step-1 of the Query Wizard** dialog box as shown in the figure 7.3.

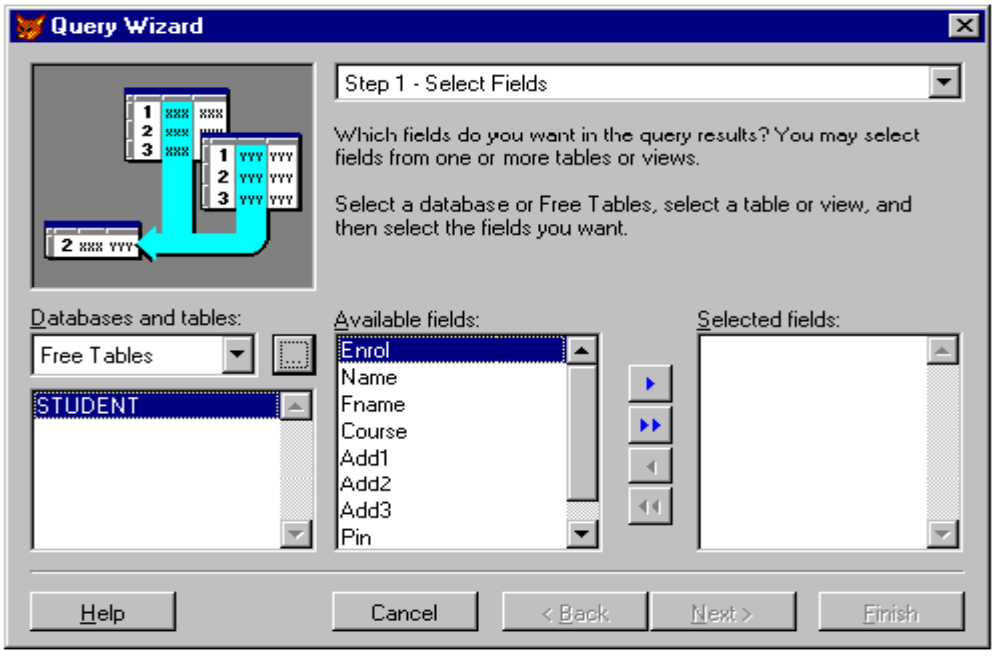


Fig. 7.3: Query Wizard Dialog Box

7.3.1 Selecting the Table

The first step in creating a Query is to select the database table and fields on which your Query is based. In fact, the first dialog box of Query Wizard allows you to select one or more tables and select the fields of that table to include in the Query result.

The **Database and tables** option available with the first step of **Query Wizard** dialog box displays the current database, if the database is open, otherwise, it displays the words **Free Tables** as shown in the figure 7.3. If a database is open, the names of the tables it contains appears in the scrollable list. If the database is not open, or if you want to use a Free Table, click the ellipsis button (...), which opens a dialog box that allows you to select the required table. The table name occurs alone in the table list, if you select a Free Table.

7.3.2 Selecting the Fields

After you select the **Table** from the table list you will notice that all the fields of the selected table are displayed in the **Available Fields** list. The Query Wizard dialog box includes a Field Picker buttons, which are found between the **Available fields** list and **Selected Fields** list. The arrow directions indicate the way they move fields. A single arrow [▶] moves a single field. Double arrow [▶▶] buttons move all the fields. To move a field from the available list, first select the field you want to move by clicking it or by using the arrow keys and then click the top buttons with right arrow. This moves the highlighted field to the selected fields list. Repeat this process to move more fields from **Available Fields** list to **Selected Fields** list as shown in figure 7.4.

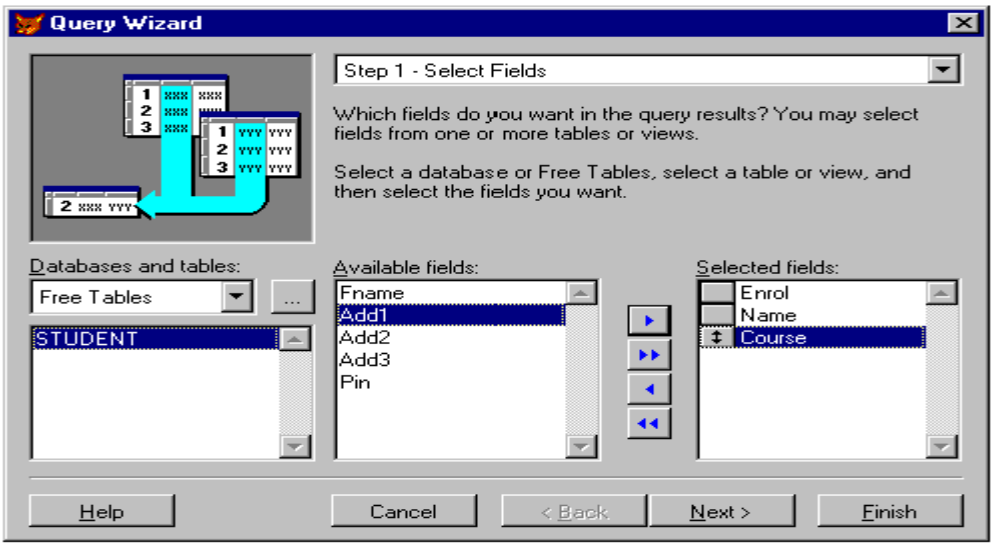


Fig. 7.4: Query Wizard with Select Fields

If you want to move all the fields from Available Fields list to Selected Fields list, click the double right arrow button. If you have selected the wrong fields or if you want to remove any field from the Selected Fields lists, then you can select the field from the Selected fields list and click the left arrow button. Similarly, you can choose the double left arrow key to move all the fields from the Selected Fields list to Available Fields list.

Remember that the order in which you select the fields for Query, the Query output will be shown in that order only. Choose **Next** push button to proceed to the next step of the Query Wizard.

7.3.3 Joining (Relating) the tables

If you want to extract data from more then one table, you can go back to the Database/Table column to select a different table, and then click additional fields to include in the query. Remember that in such a query, you must define a relation for each additional table used in the query.

The figure 7.4 shows the fields selected from the table "STUDENT". If you specify fields from more than one table, then the Query Wizard move to step-2 of the Query Wizard. Here we have selected fields from one table, so step 2 is not required. The Wizard automatically skip the steps which are not required. In step 2 Relationship of Query Wizard dialog box allows you to define a relation between each table from which you selected fields. This step will be discussed in the next topic when we will discuss about creating query with two tables. Choose **Next** push button to move to the step 3 of the query wizard.

7.3.4 Filtering the data

The Step 3 filtering of query wizard allows you to define filters. Filters mean defining conditional query. It limits the records included in the query by creating selection criteria or records in the source tables. The fields used for this selection may or may not be included in the query fields depending upon the fields you have selected for your query during the step-1 of the query wizard. For example, you might want the list of students from Jaipur;

STUNDENT.ADD3 = "JAIPUR"

To define the above expression in the "Step 3 Filtering" of query wizard as shown in figure 7.5.

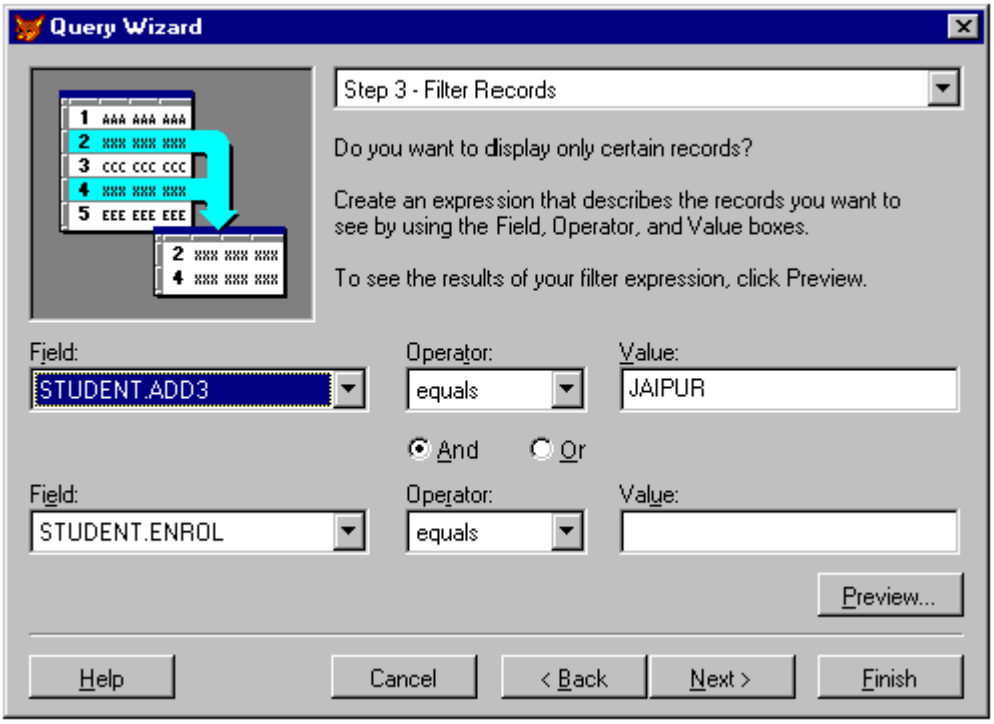


Fig. 7.5: Step-3 of Query Wizard with Filtering

Select STUDENT.ADD3 from the drop down list, select **equals** from the **Operator** list (default) and type JAIPUR in the **Value** text box.

The following operators are available with the filtering:

- Equals
- Not equals
- More than
- Is Blank
- Contains
- In
- Between

The quotes around the strings are optional. However, case is important, if the selected operator is not **Contains**. Contains performs a case-insensitive test. **Is Blank** requires no strings in the value text box. To use the operator **In**, separate the values in the list with commas, but do add extra blank. For the operator **Between** type the first and last value separated with a comma, and not space.

You can also define second filter criteria to use complete logical expression. Remember, you need to join the second filter to the prior one with either an **AND** or **OR** connector. The Wizard, however, limits you to just two filter conditions. To add more than two filter conditions you need to save the query and immediately modify it in the query Designer.

7.3.5 Specifying Sort Order

The Step 4 **Sort Order** of query wizard allows you to create a sort order for the query result output. You can select any field from the available fields list. You can select more then one field for specifying the sorting order. Remember that the order in which you select these fields determines their sort priorities. You can also change the sort priorities by moving the selected field using the double-headed buttons to the left of the field names. Just click a double-headed button and drag it up or down to a new position. You can also send the data in ascending or descending order as per your selection of the Ascending or Descending radio buttons in the Step-4 of the Query Wizard dialog box.

Figure 7.6 shows the field selected i.e. **Enrol** and **Name** for the sort order. Choose **Next** push button to move further to the step-4 of the query wizard.

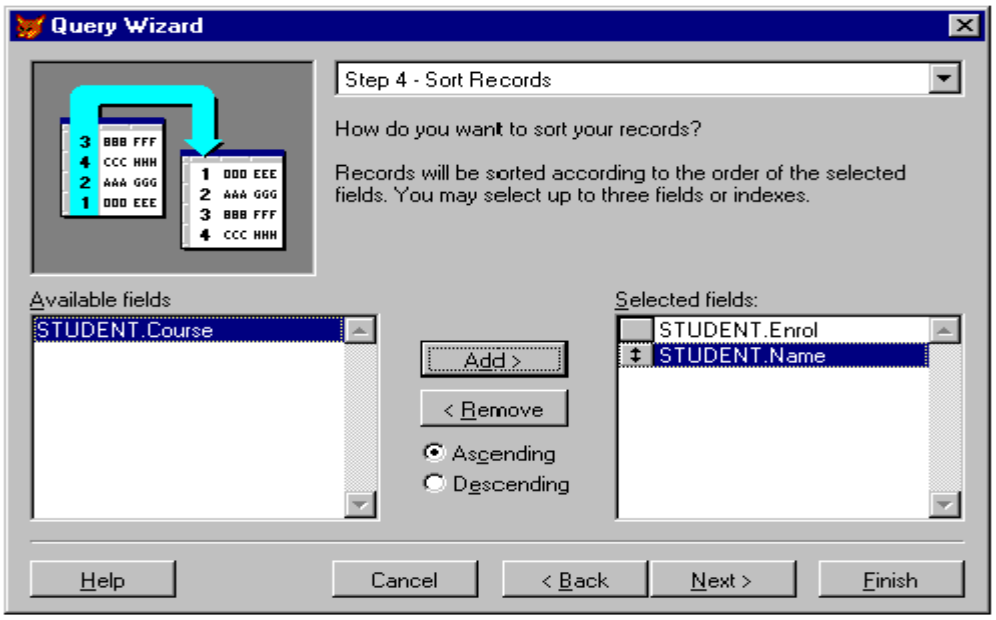


Fig. 7.6: Step 4 of Query Wizard with Sort Order

7.3.6 Previewing the Query Result

You can also click the **Preview...** push button form the step 3, to run the Filtering of query wizard while still in the Wizard and view the results. The Browse window displays the records that match the expression as shown the figure 7.7

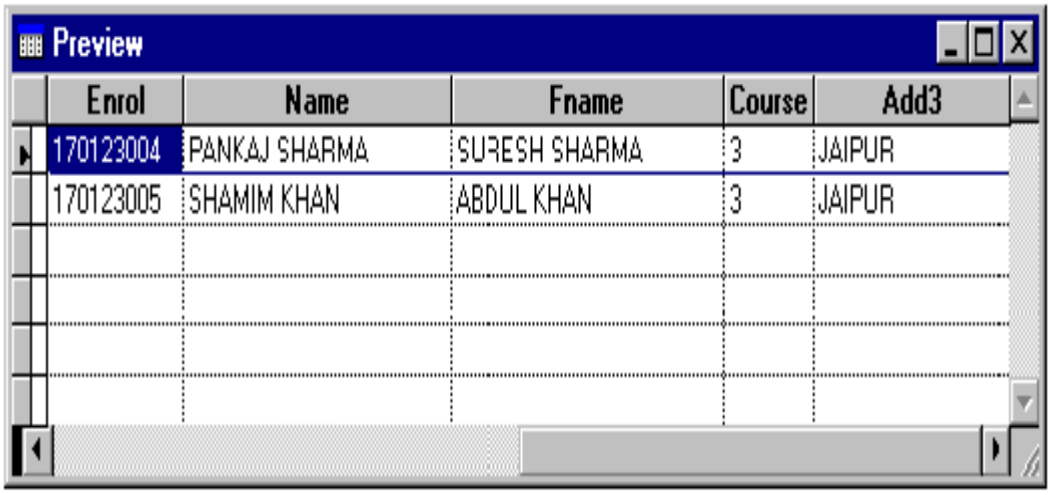


Fig. 7.7: Previewing the Query Result

You can scroll through the window to make sure the conditional expression is finding the records that you actually want. If you do not get the expected result records, you can use the back button to return to any prior step and change the query definition. You can move between the Preview and any wizard step as many times as you like. Choose **Next** push button to proceed further in the query wizard after closing the Browse window of the Query Results.

7.3.7 Saving a query

The last step of Query Wizard **Step 5 Finish** dialog box appears when you click **Next** push button from the **Sort Field** step-4 of **Query Wizard** as shown in figure 7.8. This last query wizard dialog box allows you to save the query with any one of the following three options:

- Save query
- Save query and run it
- Save query and modify it in the Query Designer.

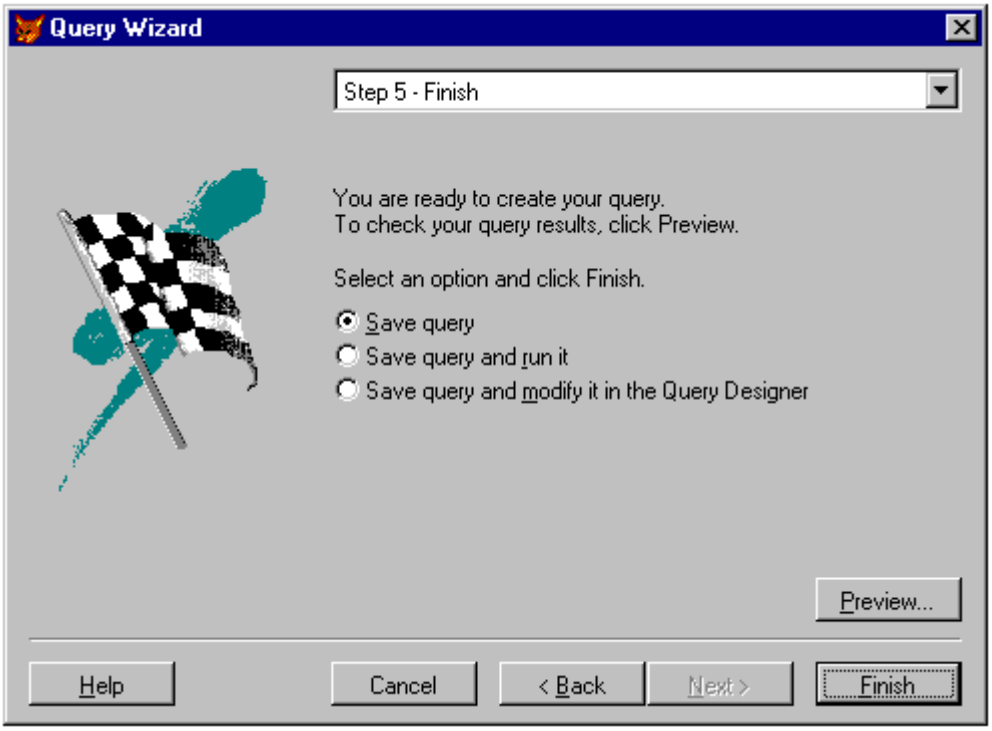


Fig. 7.8: Saving a Query

When you click **Finish** push button, Visual FoxPro Displays the **Save As** dialog box to type a name for the query. If you type the name as QRY-1 of your query then Visual FoxPro saves the query with this name and with extension name QPR, which stands for Query Program. Once you save the query, you can only modify it further by using Query Designer.

INTEXT QUESTIONS

1. Write True or False for the following statements

   (a) Query can only be viewing data and not updating data.

   (b) The first step in creating a Query is to select the database table and fields.

   (c) Query can be created using two or more tables.

   (d) You can not preview the query result.

2. Fill in the blanks:

(a) In selecting fields, the fields are moved from———————— field list to ———————— field list.

(b) You must define a ———————— for each additional table used in the query.

(c) Filters mean defining ———————————— query.

3. Write steps for finding Students from Jaipur and Delhi in Table Students.

7.4 CREATING A QUERY WITH TWO TABLES USING QUERY WIZARD

In the previous section you have learnt the queries using one table. Most databases contain several related tables.

To learn how to create a query with two tables, let us create one more table named "RESULT', with following structure and having one common field ENROL. To relate between two tables, both the data tables should have at least one common field.

Structure of table: RESULT.DBF

Field	Field Name	Type	Width
1.	ENROL	C	9
2.	SUBJECT	C	15
3.	MARKS	N	3

Now, add the following records to the table RESULT.DBF

Record	ENROL	SUBJECT	MARKS
1.	270813001	HINDI	70
2.	270813002	ENGLISH	65
3.	270813003	MATHS	72
4.	270813004	HINDI	64
5	270813005	ENGLISH	62
6.	270813006	HINDI	34

Now, do the following steps to create a query with two tables **STUDENT.DBF** and **RESULT.DBF** using query wizard.

1. Choose **New...** option from the **File** menu, select the Query radio button and click the Wizards push button. The **Wizard**

Selection dialog box appears, click the Query Wizard button. This is similar to what we discussed in the previous section.

2. The Step-1 Field Selection of **Query Wizard** dialog box appear, when you click **Query Wizard** from the Wizard Selection dialog box. Click the right button of the Database/Tables control, to display the **Open** dialog box to add the "STUDENT" table to the list and then click this button again to add one more table "RESULT" which we have created.

3. Select the table "STUDENT" to display its fields in the **Available Fields** list. Now select the following fields from the Available Fields and move them to the **Selected Fields** list by double clicking them or by using the right arrow button.

   - ENROL
   - NAME

   Then select the table "RESULT" from the Database/Tables list to display the fields of "RESULT" table in the Available Fields list. Select the fields SUBJECT and MARKS to add them in the selected fields list as shown in the figure 7.9

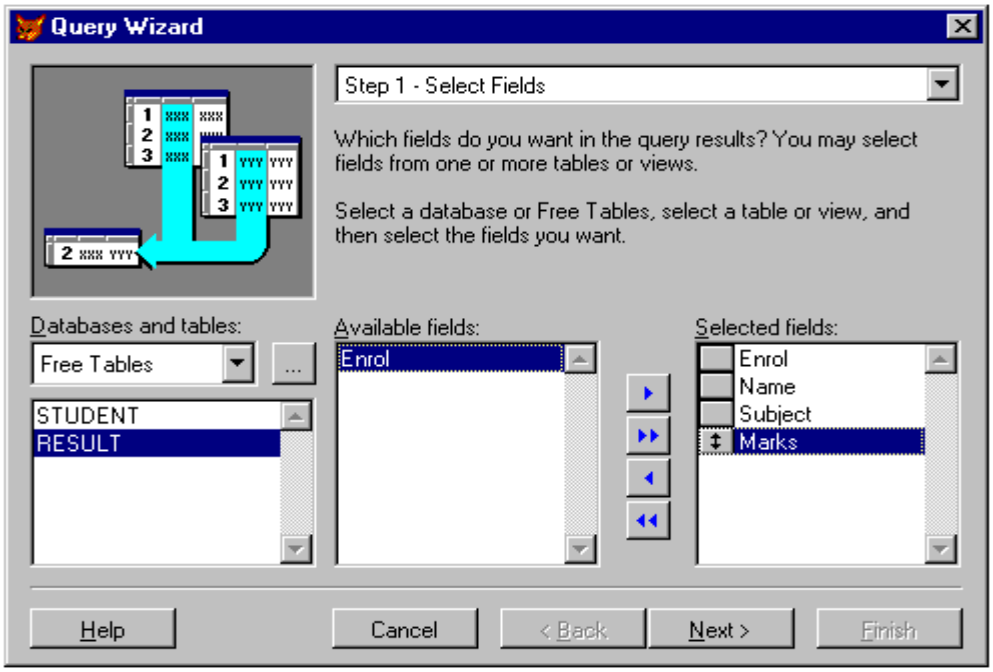


Fig. 7.9: Query Wizard with Two Tables

4. Click **Next** push button, Visual FoxPro displays the Step-2 Relationship dialog box of the Query Wizard. In this box you have to specify how the tables are related, as shown in figure 7.10.

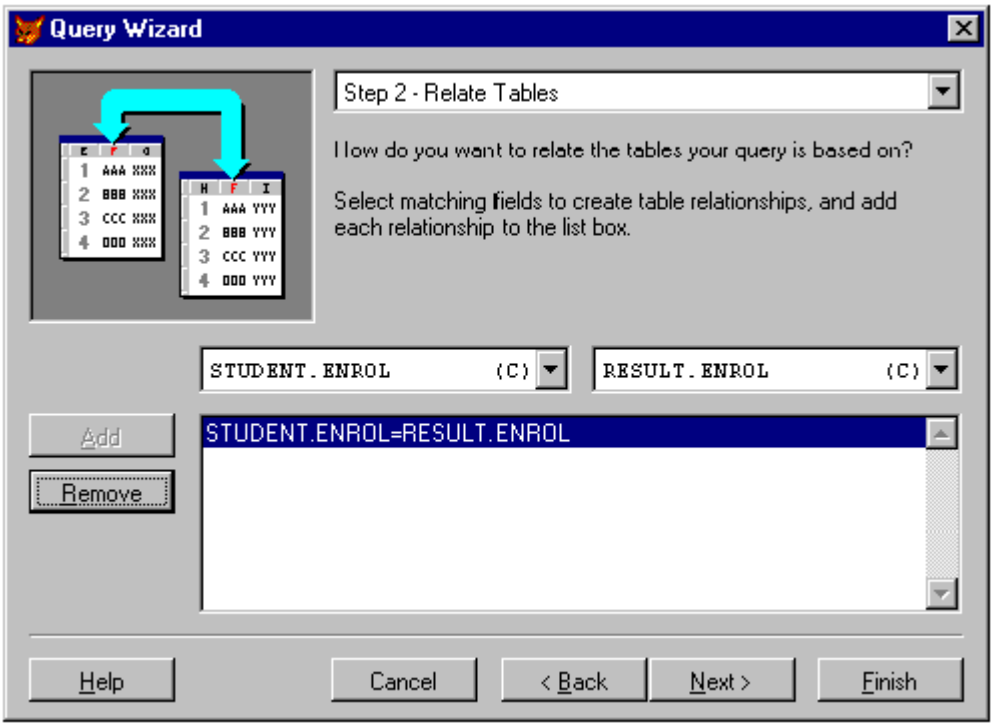


Fig. 7.10: ***Relate Table of the Query Wizard***

It is suggested that the field STUDENT ENROL and RESULT ENROL has a common field ENROL in both the tables. Click the **Add...** push button to display the relationship as shown in the figure 7.10.

5. Click the **Next** push button and the **Step-2a Include Records** of the Query Wizard dialog box appears. This dialog box allows you to include the records from the following options as shown in the figure 7.11.

   a. Only matching rows

   b. All rows from the Table STUDENT

   c. All rows from the Table RESULT

   d. All rows from both tables

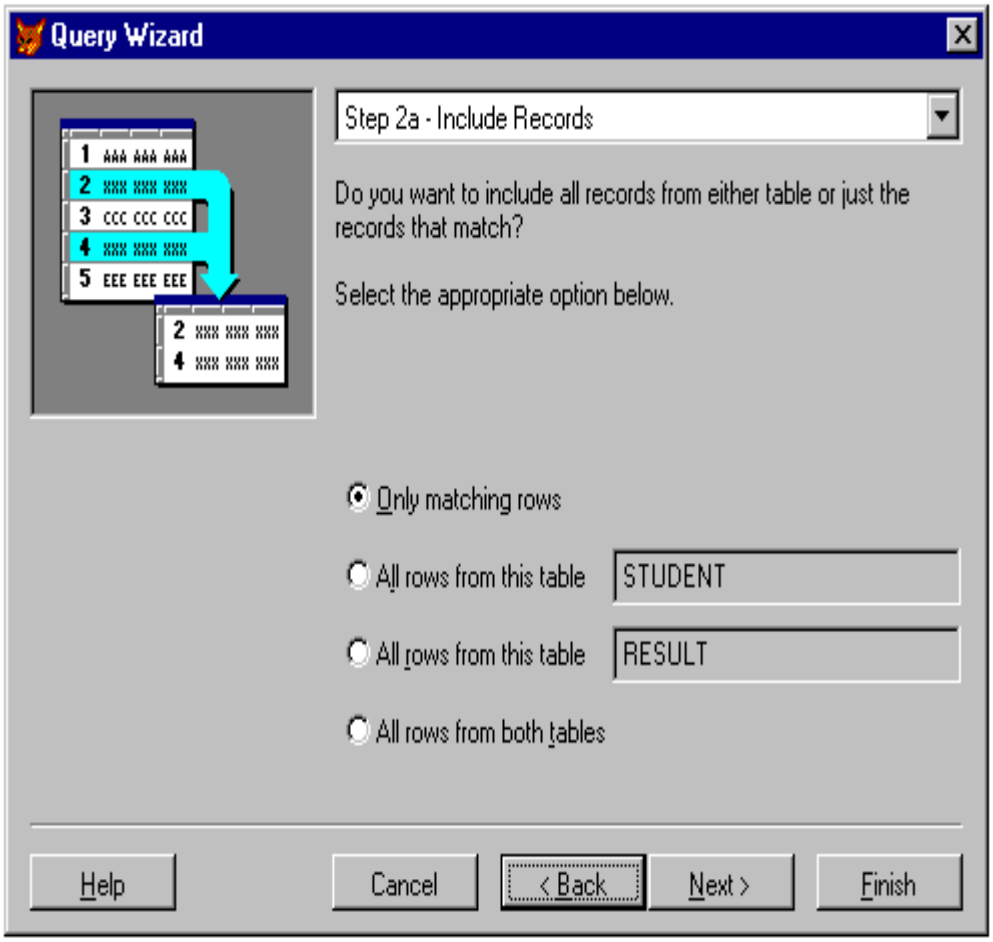


Fig. 7.11: Query Wizard Include Records

The default option is to include all records with matching records.

6. Click the **Next** push button and the Step-3 Filtering of the Query Wizard dialog box appears. If you want to filter the records of your query you can repeat the steps of Filtering as discussed in the previous section otherwise, you can skip this step by clicking **Next** push button again.

   Click the **Next** push button and Visual FoxPro displays the Step-4 Sort Order dialog box of the Query Wizard. Double click the STUDENT ENROL field to add it in the Selected Fields list as shown in the figure 7.12.

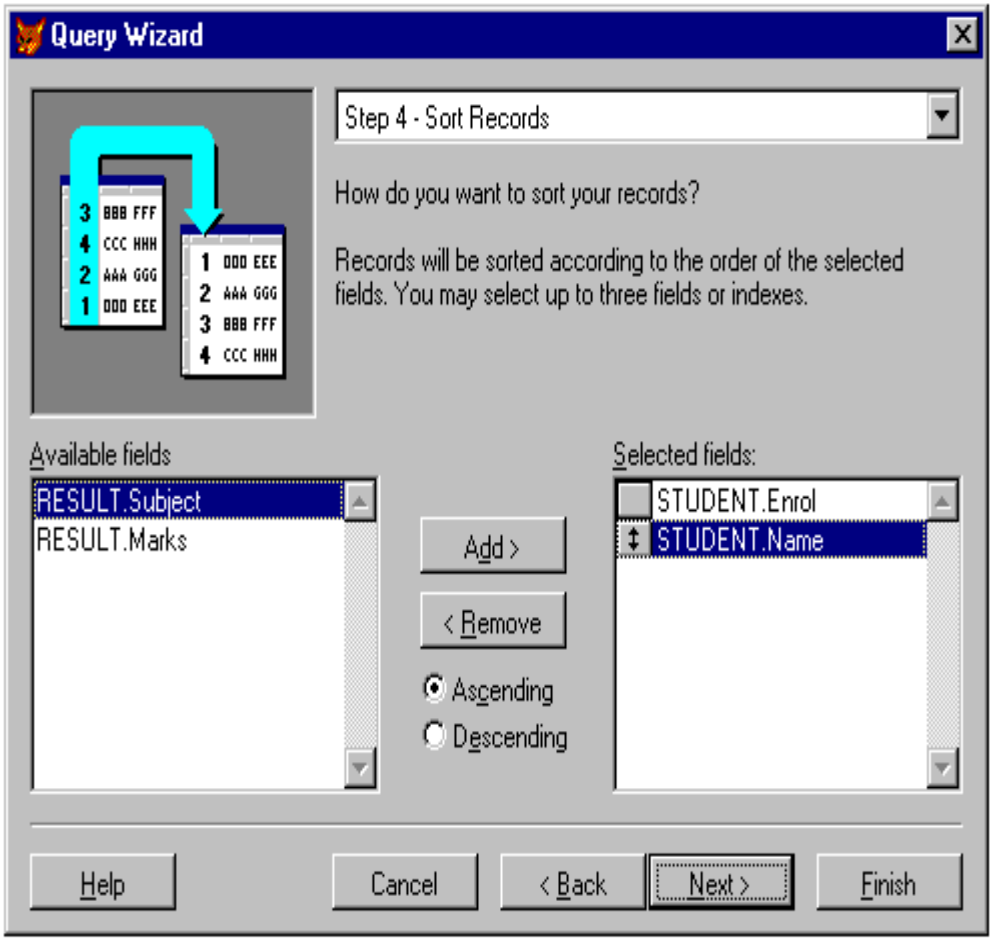


Fig. 7.12: Query Wizard with Sort Records

7. Click the **Preview** push button, to view the result of the Query, which includes the data form the two tables.

   Now click **OK** push button to close the Preview window. Lastly click the **Finish** push button from the step Finish and have the Query as QRY-2 in the **Save As** dialog box. Click **Save** push button from the **Save As** dialog box to save the Query.

7.5 CREATING QUERY WITH QUERY DESIGNER

To create a query from scratch with Query Designer, choose **New** option from **File** menu. The **New** dialog box appears as shown in the figure 7.13. Choose the **Query** radio button from the **New** dialog box and click the **New File** push button.

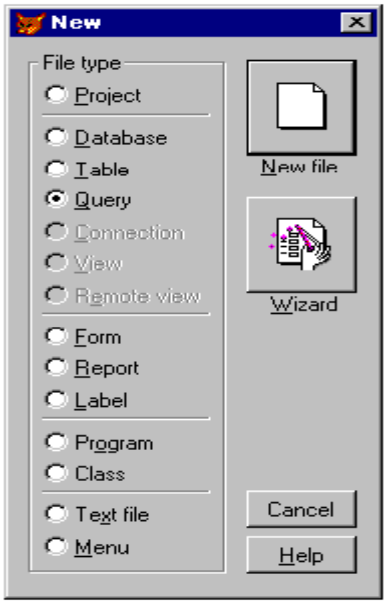


Fig. 7.13: New dialog box to create a Query

You can also use the query designer to modify a query, which you have created using **Query Wizard.** Choose **Open** option from the **File** menu and select the query file from the **Open** dialog box.

7.5.1 Adding a Table or View

A query is always created on a table. If no table is in use, Visual FoxPro displays the Add Table or View dialog box, which lists all the tables of the open database.

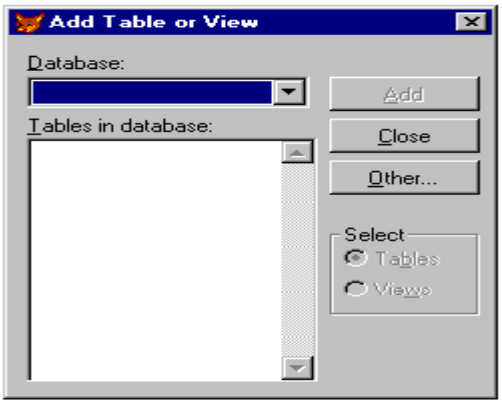


Fig. 7.14: Add Table or View

Figure 7.14 shows the **Add Table View dialog box**, where you can select a free table by choosing **Other...** push button, which display the Open dialog box. It allows you to select a table from any directory or drive, including network drive.

Figure 7.15 shows the table STUDENT.DBF that you have selected, is included in the panel at top of the Query Designer. The name of the table always appears in the window header.

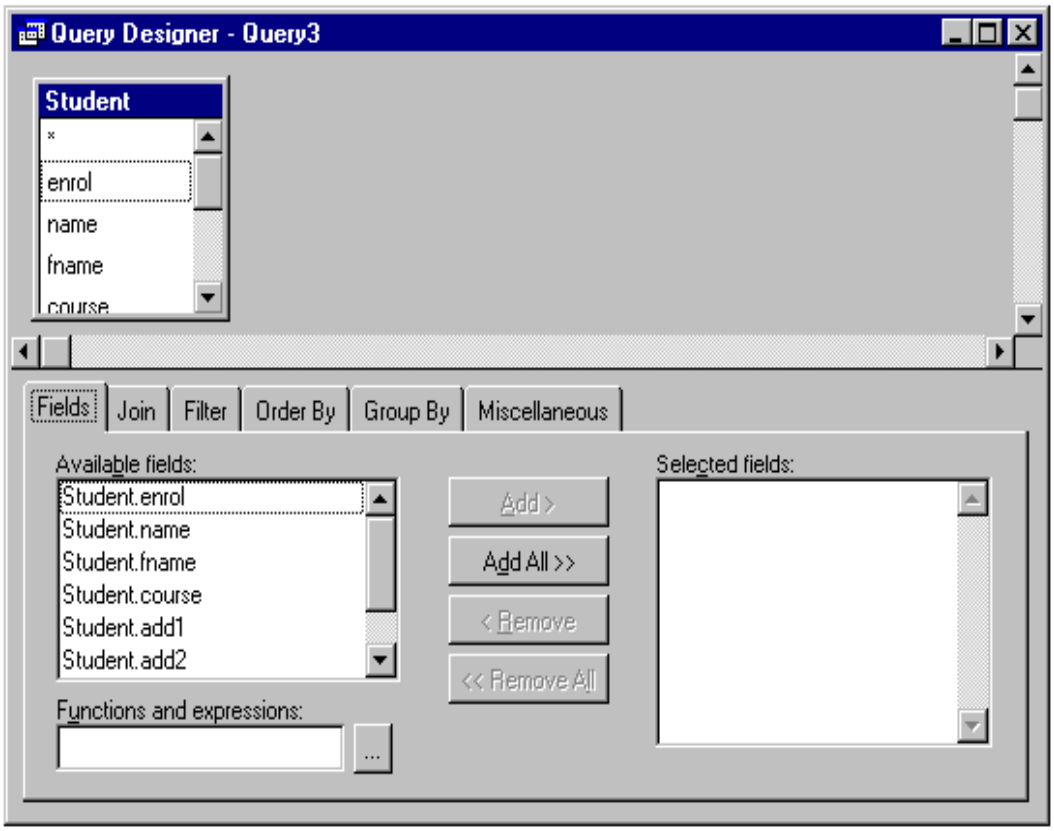


Fig. 7.15: Query Designer with One table

Remember that you can also add table at any time, by choosing **Add Table...** option from the **Query** menu or by clicking the **Add Table** tool from the **Query** toolbar to display the **Add Table or View** dialog box, which allows you to select a table. Similarly, you can also remove table at any time, by choosing **Remove Table** option from the **Query** menu or by clicking the **Remove Table** tool from the **Query** toolbar as shown in the figure 7.16.

Fig. 7.16: Query Toolbar

7.5.2 Field Panel

After selecting the table STUDENT.DBF in the Query designer, you need to specify which fields you want to include in the query result. The **Field** panel of the Query Designer allows you to specify the fields you want to include in your query result. To specify fields in the **Field** panel, choose **Fields** tab from the Query Designer. The Fields Panel dialog box appears as shown in the figure 7.15.

To move the fields you want to include from the **Available Fields** list to the **Selected Fields** list, do one of the following:

- Select one or more fields in the **Available Fields** list that you want to use and choose **Add** > push button to copy them to the **Selected Fields** list.
- Double click on a field in the **Available Fields** list to copy it to the **Selected Fields** list.
- If you have to select all the fields, you can click **Add All** button. Similarly, if you have to remove any field, you can do by clicking **Remove** key or **Remove All** key.

For example, let us choose the following fields from the Available Fields list by double clicking them.

STUDENT.ENROL

STUDENT.NAME

STUDENT.FNAME

STUDENT.COURSE

STUDENT.ADD3

The selected fields are displayed in the Selected Fields list as shown in the figure 7.17 You can also include functions or expressions in the result, if you are creating a grouped query, by selecting them from the Functions/Expressions drop-down list and clicking **Add** push button.

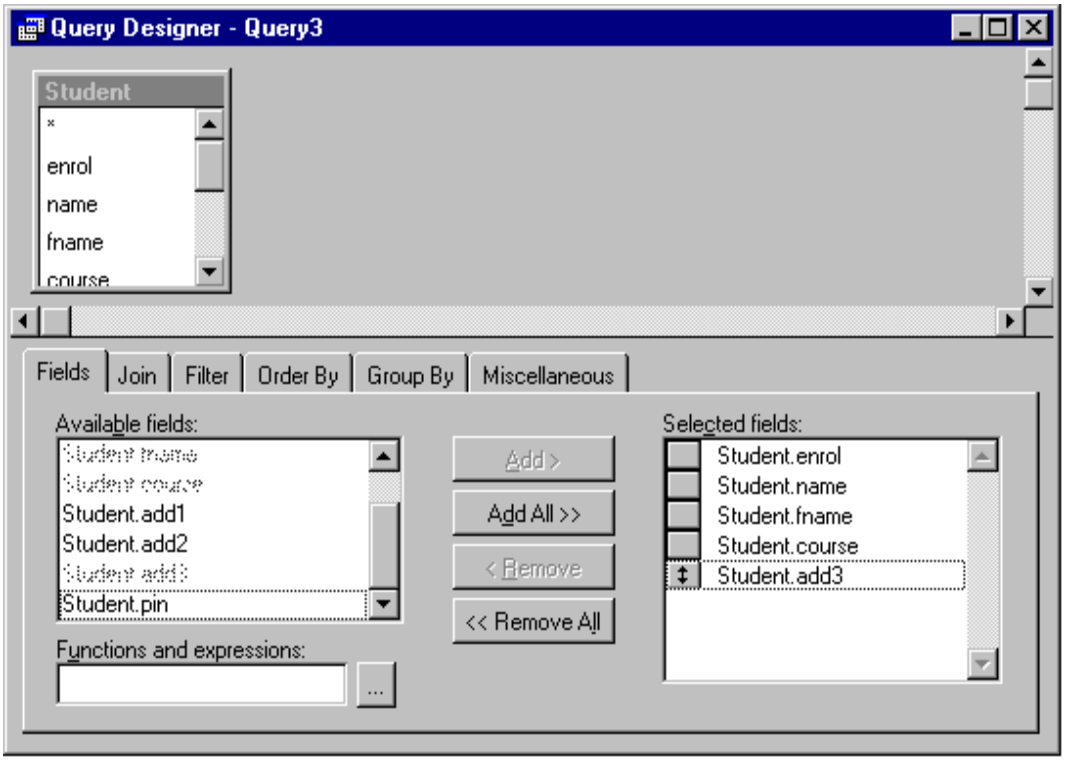


Fig. 7.17: Query Designer Fields selection

7.5.3 Join Panel

This option is required when you are creating a Query with two or more tables. This option will be explained when we will discuss "Creating Query with Query Designer using two Tables".

7.5.4 Filter Panel

The Filter option in the Query Designer determines which records are extracted from the Table. If this option is empty, the query will extract all the records from the open table named "STUDENT.DBF". To specify the Filter option to determine which records will be matched, select the **Field Name** drop down list to choose a field and **Criteria** list to choose the operator, and then enter the value you want to match the record in the data table as shown in the figure 5.18. Choose STUDENT.ADD3 field name from the **Field Name** drop-down list and enter the text JAIPUR in the **Example** text box, to find all the records having Address3 as JAIPUR.

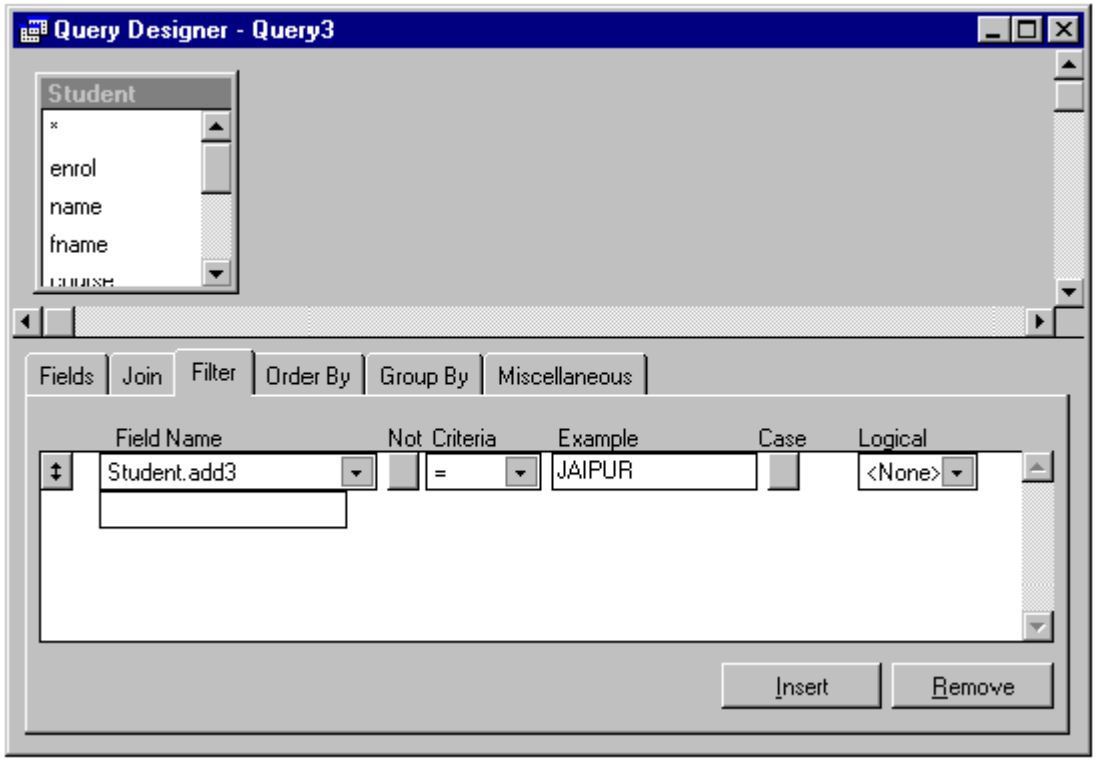


Fig. 7.18: Query Designer with Filter Option

Choose **Run Query** option from the **Query** menu, to run the query. The Browse window appears with all the records having Address3 equal to JAIPUR, as shown in the figure 7.19.

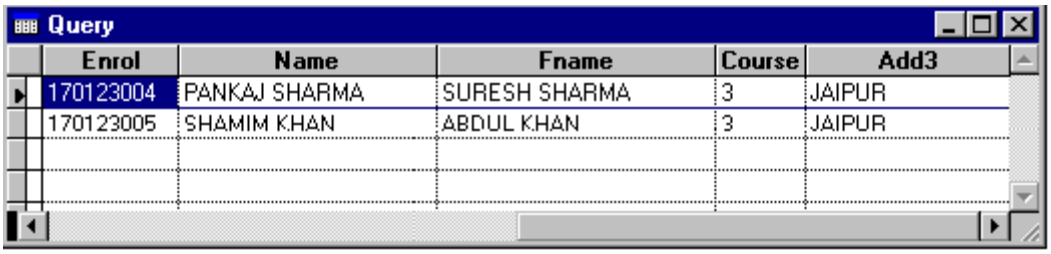


Fig. 7.19: Query Result of Filtering

7.5.5 Order By Panel

To specify the order in which records will appear in the query results, choose **Order By** panel from the Query Designer. The Order **By panel** occurs with all the fields in the **Selected Fields** list.

To specify fields that will determine the order of the query results, select a field from the **Selected Fields** list of **Order By** panel to determine the order of the query result as shown in the figure 4.20.

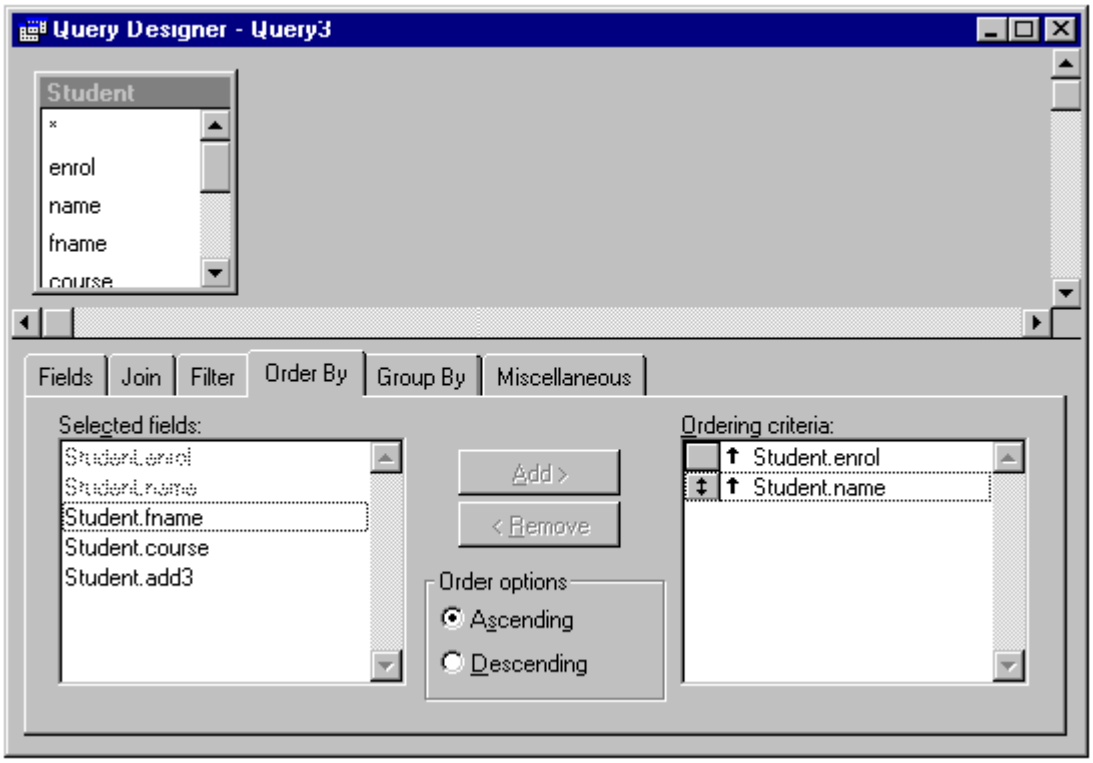


Fig. 7.20: Query Designer with Order By Panel

You can also select more than one field to determine the order of the query results. Choose the **Add** push button to copy it to the **Ordering Criteria** list. Once you add Selected fields to the Ordering Criteria list, you can determine the order by choosing **Ascending** or **Descending** radio button. You can also adopt a shortcut method by double-clicking on a field from **Selected Fields** list to move directly to the **Ordering Criteria** list.

To remove fields from the Ordering Criteria, select one or more fields from the **Ordering Criteria** list of **Order By** panel and choose the Remove push button, the selected fields no longer appear in the **Ordering Criteria** list or double-click on a field in the Ordering Criteria list to remove it directly from the **Ordering Criteria** list. The order in which fields appear in the Ordering Criteria list determines the order of priority of the query results.

For example, double-click on the fields **Student.Enrol**, then **Student.Name**. This moves both the fields to the **Ordering Criteria** list and will affect the output to be sorted by Enrolment, plus Name as shown in the figure 7.20.

Choose **Run Query** option from the **Query** menu, to do the query. The Browse window appears in the ascending order, as shown in the figure 7.21.

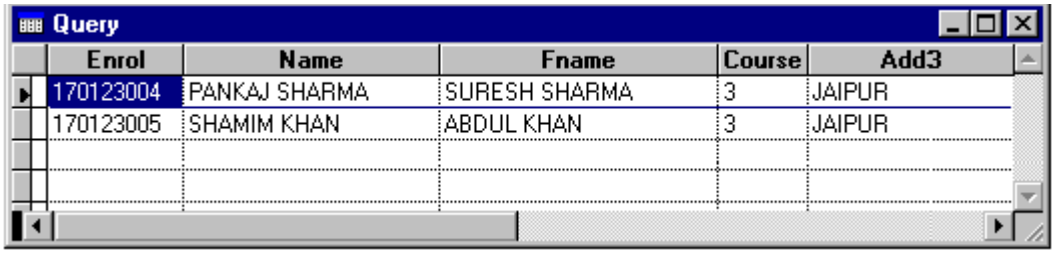


Fig. 7.21: Browse window with Order By

To save a query, choose **Save As** option from the File menu or click the Save tool. This opens the **Save As** dialog box to specify the name of the Query. Enter the name QRY3 in the text box and choose the **Save** push button as shown in the figure 7.22.

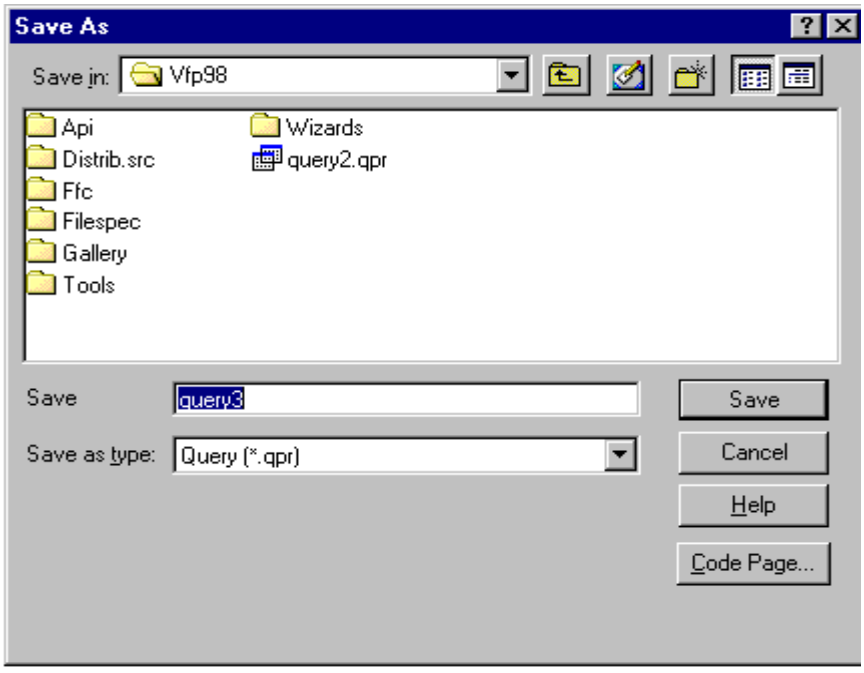


Fig. 7.22: The Save As dialog box

7.6 CREATING QUERY WITH QUERY DESIGNER USING TWO TABLES

To create a query with Query Designer using two tables, let us consider the two tables, which we have already created STUDENT.DBF and RESULT.DBF. Choose **New...** option from the **File** menu. Visual FoxPro displays the **New** dialog box. Choose **Query** radio button and then choose **New File** push button. Visual FoxPro displays the **Add Table or View** dialog box to select the table for query. From the **Open** dialog box select the table named "STUDENT.DBF". The **Query Designer** appears with the included table as shown in the figure 7.23

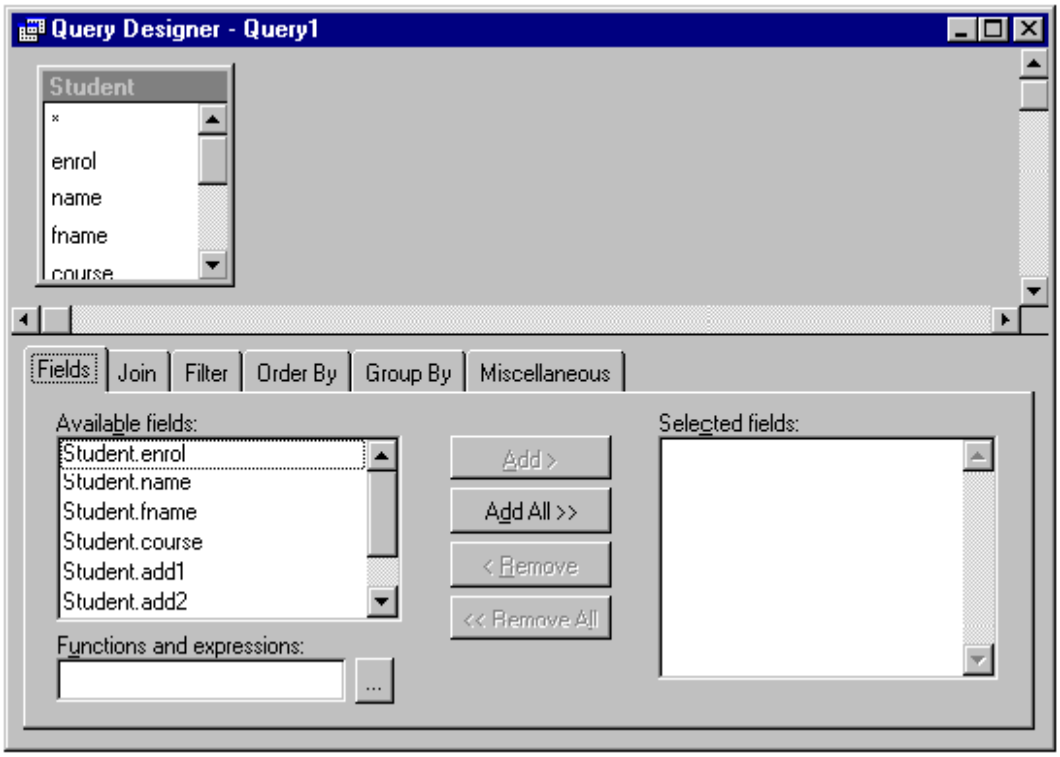


Fig. 7.23: Query Designer with Added Table at the top

Choose **Add Table...** option from the **Query** Menu. The **Add Table View** dialog box appears to select the second table. Choose **Other...** option to display the **Open** dialog box, and select the table named "RESULT.DBF" You can either double-click the file name after highlighting it or choose **OK**.

Visual FoxPro displays the **Join Condition** dialog box. By default,

the first field STUDENT.ENROL will be displayed on the left and the second table RESULT.ENROL will be displayed on the right with a joining line as shown in the figure 7.24. The **Join Condition** dialog box is displayed with the following option for **Type of Join** with its description given in the **Description** box:

- Inner Join — Create a result set that includes only the Student records that match Result records.
- Left Join — Create a result set that includes Student records and matching Result records.
- Right Join — Create a result set that includes Result records and matching Student records.
- Full Join — Create a result set that includes all Student records and all Result records.

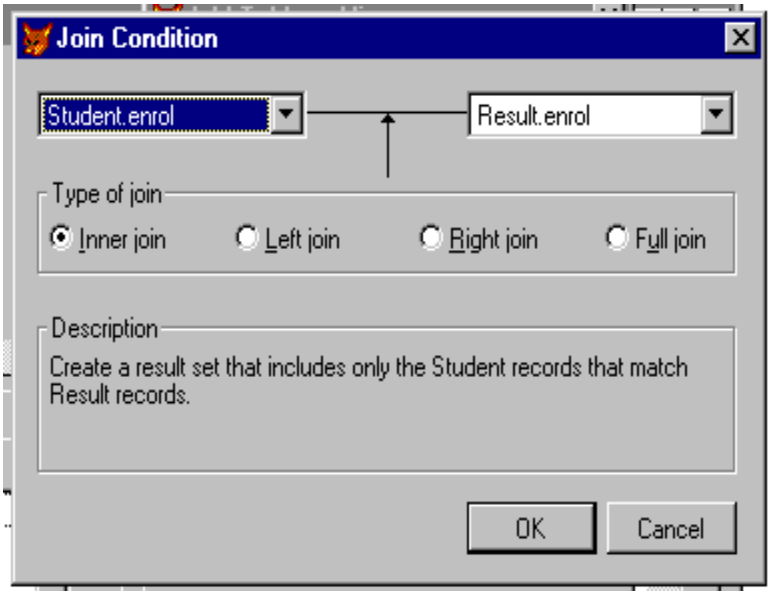


Fig. 7.24: Join Condition Dialog Box

Select the **Type of Join** as per the requirement of the query and choose **OK** push button and close the **Add Table or View** dialog box. Please note that the expression you have created with Join Condition is displayed in the Join Panel of the Query Designer, and the two tables are connected with the to indicate the join, as shown in figure 7.25.

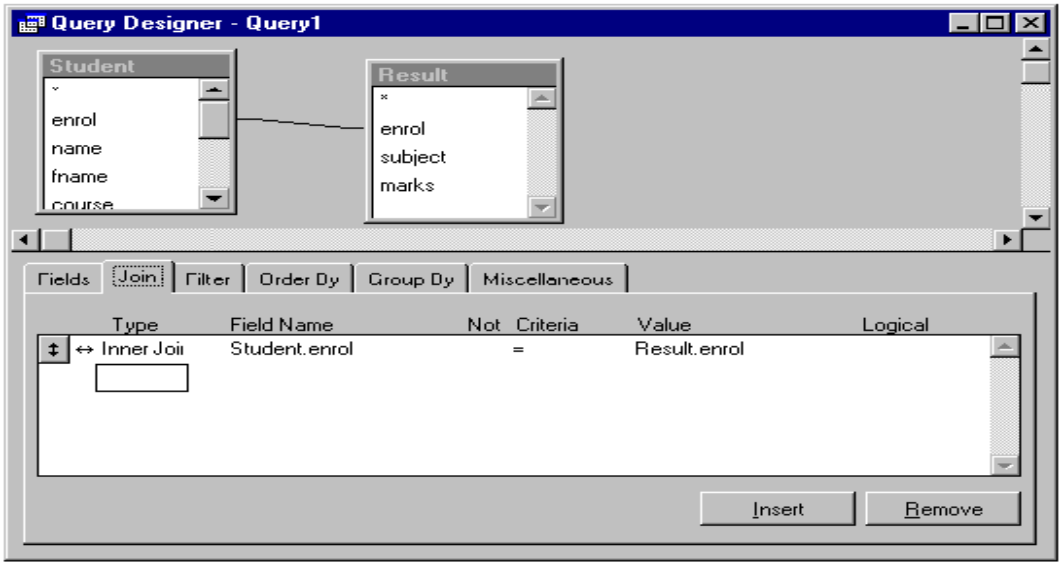


Fig. 7.25: Displaying the Join in the Query Designer

Choose **Fields** panel of the **Query Designer** to select the fields for the query results. Select the following fields from the Available Fields list and move them to the Selected Fields list as shown in the figure 7.26.

Student.enrol
Student.name
Student.course
Result.subject
Result.marks

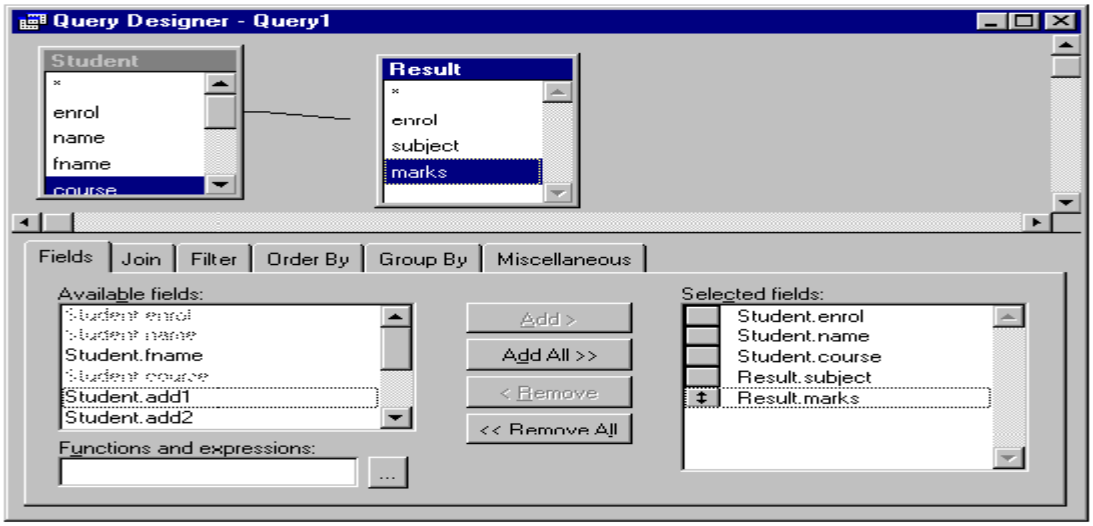


Fig. 7.26: Query Designer with two tables

Choose **Run Query** option from the Query menu to execute the query. The query result appears in the Browse window, as shown in the figure 7.27.

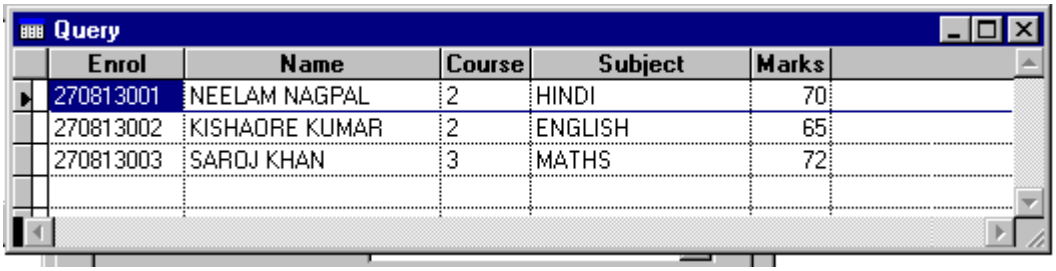


Fig. 7.27 Query Results using two tables

To save the query you have created using two tables, open the **File** menu and choose **Save** option. The **Save As** dialog box appears and give the query name as QUERY-4 and click the **Save** push button. Similarly, you can create a query using more than two tables.

7.7 INTEXT QUESTIONS

4. Write True or False for the following statements

   (a) To relate between two tables both the data table would have at least one common field.

   (b) Include Records of the query wizard allows you to include only matching rows.

   (c) You can add or remove table through Query Toolbar.

   (d) You can not select more than one field to determine the order of the query result.

5. Fill in the blanks:

   (a) In order to create a query with query designer, choose ——— radio button and click ———————— push button.

   (b) The name of the table always appears in the ————————.

   (c) The extension name given to the query file is ———————— ——.

6. What are the four different types of Joins?

7.8 QUERY DESTINATION

So far you have seen the query results in a Browse or Edit windows

on-screen. However, you may want to keep some query results for a longer time, perhaps saving them for future use or using them in a report. For this Visual FoxPro provides you with a feature **Query Destination**. To change the query destination from on-screen Browse window to a file or printer, choose the **Query Destination** option from the **Query** menu or by choosing **Query Destination** from the **Query Designer** toolbar. The **Query Destination** Dialog box appears as shown in the figure 7.28.

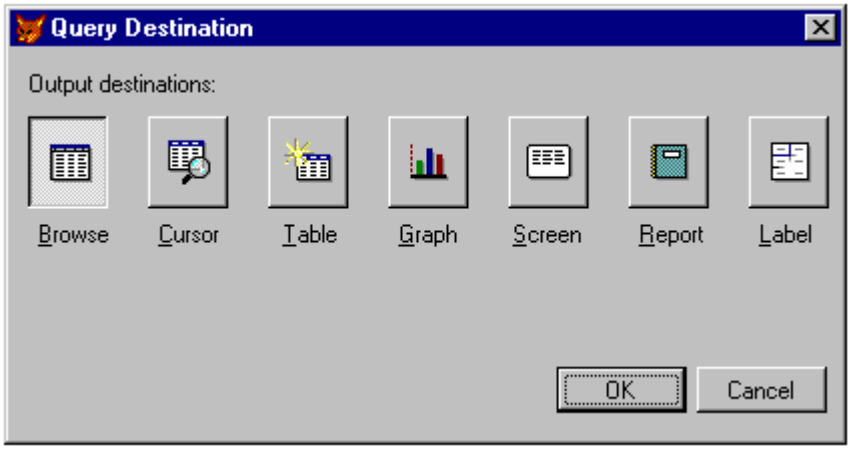


Fig. 7.28: Query Destination dialog box

Visual FoxPro displays the following options from the Query Destination dialog box:

- Browse
- Cursor
- Table
- Graph
- Screen
- Report
- Label

7.9 WHAT YOU HAVE LEARNT

In this lesson you have learned about creating a query with query wizard and query designer. You have also learned how to create a query using two or more tables and how to join two tables. The principles of creating relationship between two tables are discussed in this lesson. You have also learned about viewing the query results and saving the query results using query destination.

7.10 TERMINAL QUESTIONS

1. What is the different steps involved in creating a query using two tables?
2. Explain the principles of relationship between the two tables.
3. What do you understand by query destination?

7.11 KEY TO INTEXT QUESTIONS

1. (a) F, (b) T, (c) T, (d) F
2. (a) available, selected, (b) relation, (c) conditional
3. as done in section 4.3.4, student.add3='JAIPUR' and student.add3='DELHI'
4. (a) Query, New File
   (b) window header
   (c) qpr
5. (a) Query, New File
   (b) Database
   (c) qpr
6. Inner, Left, Right and Full

8

CREATING FORM WITH FORM WIZARD AND FORM DESIGNER

8.1 INTRODUCTION

Forms are very powerful tool embedded in almost all the Database Management System. It provides the basic means for inputting data for the application. Almost every application has at least one form. Some forms only display information to the users. However, most forms require some degree of user interaction. Forms are part of object-oriented programming whereby, users expect to be in control of the application.

When you create a form either with form Wizard or by using Form Designer, you are actually writing Visual FoxPro source code for your application. When you save a form with a file name, Visual FoxPro stores the information about the form designs in a special data file with the same name as of the form and an extension of **.SCX**

8.2 OBJECTIVES

After going through this lesson, you would be able to

- create a Form with Form Wizard
- run a Form
- create One-to-Many Form Wizard

- create a form using Form Designer
- use List Boxes

8.3 CREATING A FORM WITH FORM WIZARD

To create a form by using From Wizard, choose **New** option from **File** menu or click the **New** tool box. The **New** dialog box appears. Select the **Form** radio button and then click the **Wizard** push button. Visual FoxPro displays the **Wizard Selection** dialog box as shown in fig 8.1. This dialog box allows you to select one of the two form styles available in Form Wizards.

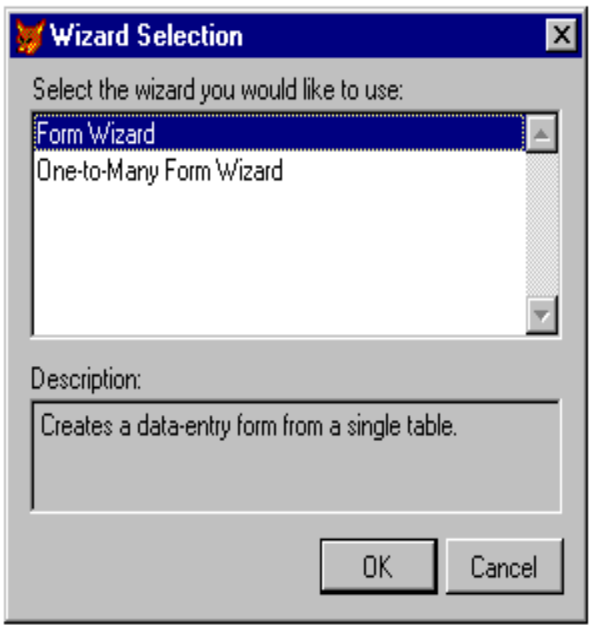


Fig. 8.1: Selecting a Form Wizard

The first one, Form Wizard is a simple form based on a single table. The second is a One-to-Many Form Wizard based on relational databases i.e. using two or more related tables. The description of each style is given in the Description Box.

Select the **Form Wizard** to display the first page-step1 of the Form Wizard dialog box as shown is figure 8.2.

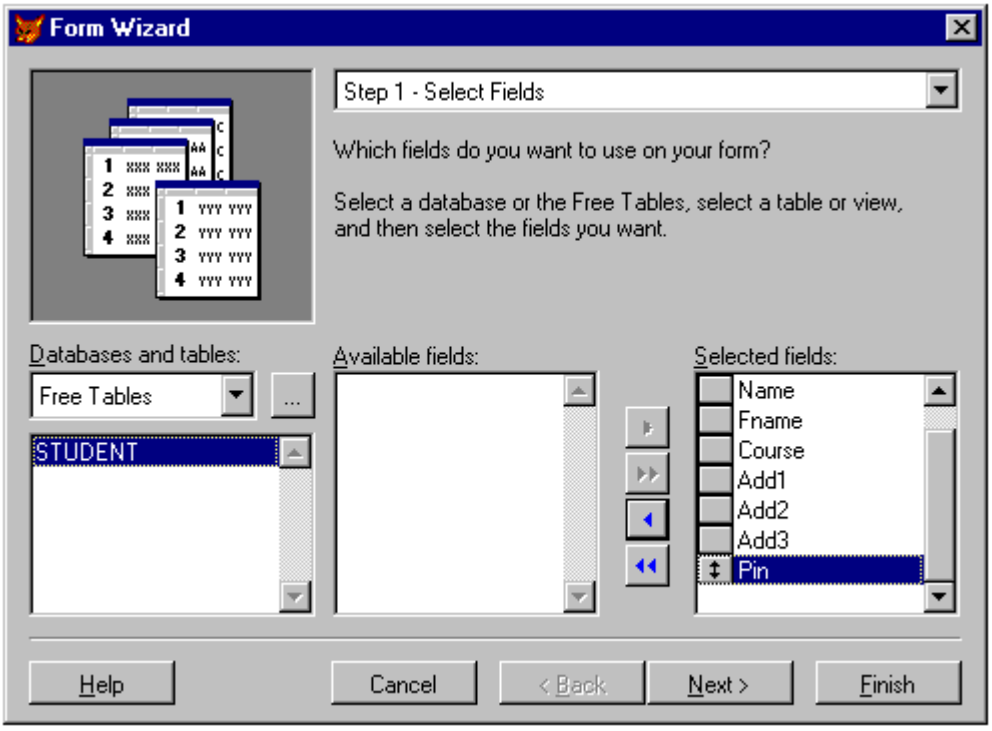


Fig. 8.2: First step of the Form Wizard Dialog box

8.3.1 Selecting the Table

Every form is based on a table. The first dialog box of Form Wizard allows you to select one or more tables on which your form is based and select the fields of that table to be included in the form.

The Databases/Tables option available with the first step of form Wizard dialog box, displays the current database, if it is open. Otherwise, it display the words **Free Tables** as shown in figure 8.2. If a database is open, the name of the tables it contains appears in the scrollable list. If the database is not open, or if you want to use a free table, click the ellipsis button (...), which opens a dialog box that allows you to select the required table. The **Open** dialog box displays all the existing tables and allows you to select the required table. Choose the table named "STUDENT".

8.3.2 Selecting Fields

After you choose the required table, you need to select the fields of the selected table to be included in the form. The dialog box also

includes Fields Picker buttons or movement buttons, which are found between the **Available Fields** list and **Selected Fields** list. The arrow directions indicate the way they move fields. A single arrow moves a single field and the Double arrow buttons move all the fields. To more a field from the Available Fields list, first select the field, you want to move by clicking it or by using the arrow keys and then click the top buttons with night arrow. This moves the highlighted field to the **Selected Fields** list. Repeat this process to move more fields from **Available Fields** list to **Selected Fields** list as shown in figure 8.3.

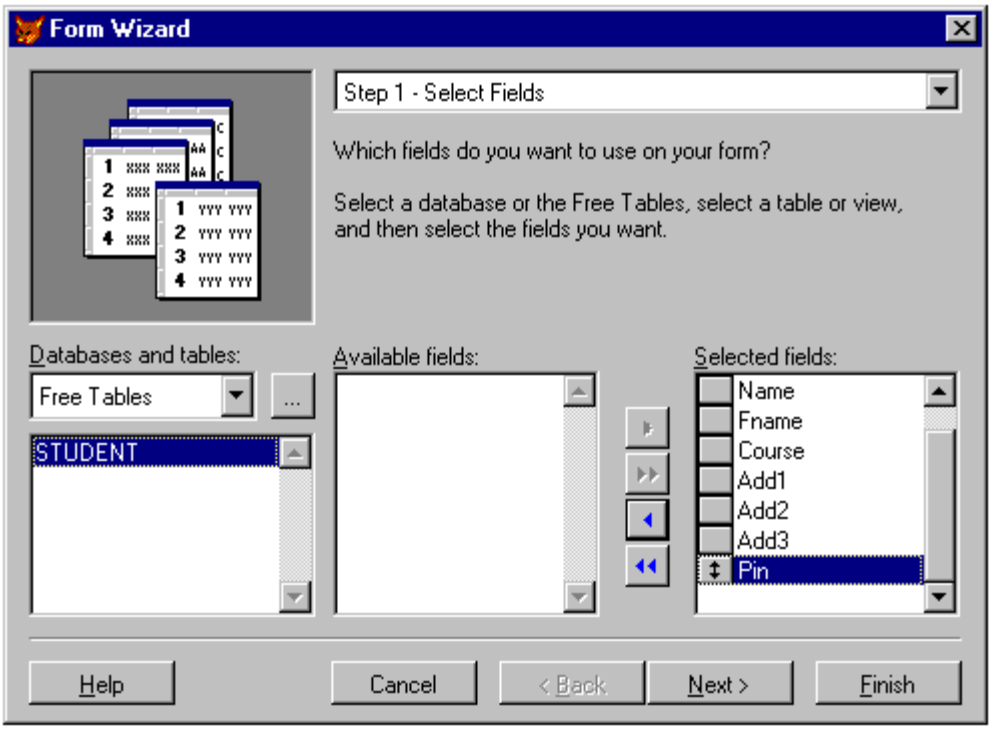


Fig. 8.3: Form Wizard Dialog box

If you want to move all the fields from the **Available Fields** list to **Selected Fields** list, click the double right arrow. If you have selected the wrong fields or want to remove any field from the Selected Fields list, then select the field from the Selected Fields list and click the left arrow. Similarly, you can choose the double-left arrow to move all the fields from the Selected Fields list to Available Fields list.

Let us click the double right arrow, to move all the fields of "STUDENT.DBF" from Available Fields list to Selected Fields list.

Remember that the order of the selected fields determines the order

in which the Wizard places the fields on the form. It also determines the default tab order for the form. However, you can customize your form generated by the Wizard by using Form Design.

Choose **Next** push button to proceed further to the Step-2 of the Form Wizard.

8.3.3 Selecting Style

The Step-2 **Choose Form Style** of Form Wizard allows you to define the overall form style. Visual FoxPro provides the five basic types of style.

- Standard
- Chiseled
- Shadowed
- Boxed
- Embossed

In the upper-left corner of the dialog box, you can preview an illustration of each Style as you select it as shown in figure 8.4.

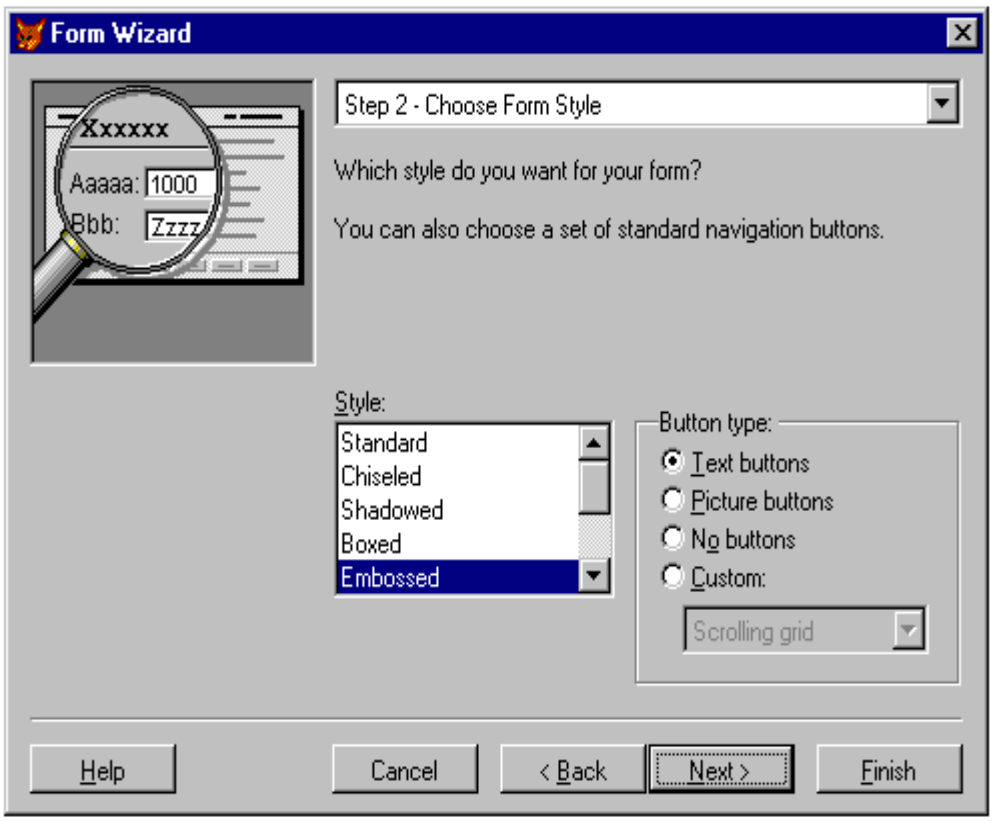


Fig. 8.4: Step2 of the Form Wizard Dialog box

The Styles box also includes four button type selections. This option refers to the buttons added automatically to the bottom of the form for navigation and editing. You can display these buttons with either text or picture icons to identify them. You can also choose **No buttons** radio button to not include buttons on the form. You can also choose the **Custom** radio button to customize your form in your own way, once you are finished with the Form Wizard.

Let us choose **Standard** from the **Style** list and **Text buttons** radio button from the **Button type**. Choose **Next** push button to proceed further to the Step3 of the Form Wizard.

The following controls buttons are added to Wizard generated form:

Button	Description
Top	Allows you to move to the first record.
Prev	Allows you to move to the previous record.
Next	Allows you to move to the next record.
Bottom	Allows you to move to the last record.
Find	Allows you to locate a record.
Print	Allows you to print the records in the table.
Add	Allows you to add a new record.
Edit	Allows you to edit the current record.
Delete	Allows you to delete the current record.
Save	Allows you to save the changes to the current record.
Revert	Allows you to throw out changes to current record and reread original field contents.
Exit	Allows you to come out of the form.

8.3.4 Specifying Sort Order

The next step of the Form Wizard is the Step3-Sort Record of Form Wizard. It allows you to create a sort order for the records. Since the forms save the data environment, it can define a sorted view of the selected table fields. If you do not select a sort order, the form displays the records from the table in their natural order in which the record are added in the table. You can select any field from the Available Fields list and move it to the Selected Fields list by clicking the **Add >** key. You can select even more than one field for the

sorting order. However, the order in which you select these fields determines their sort priority. You can also change the sort priority by moving the selected field using the double-headed buttons to the left of the field names. Just click a double-headed button and drag it up or down to a new position. If you have selected a wrong field for the sorting order or if you want to remove any field from the Selected Fields list to the Available Fields list, then select the field you want to remove and then click the **<Remove** push button. Let us select **Enrol** field from the Selected Fields list and move it to the Selected Fields list for specifying the sorting order as shown in the figure 8.4.

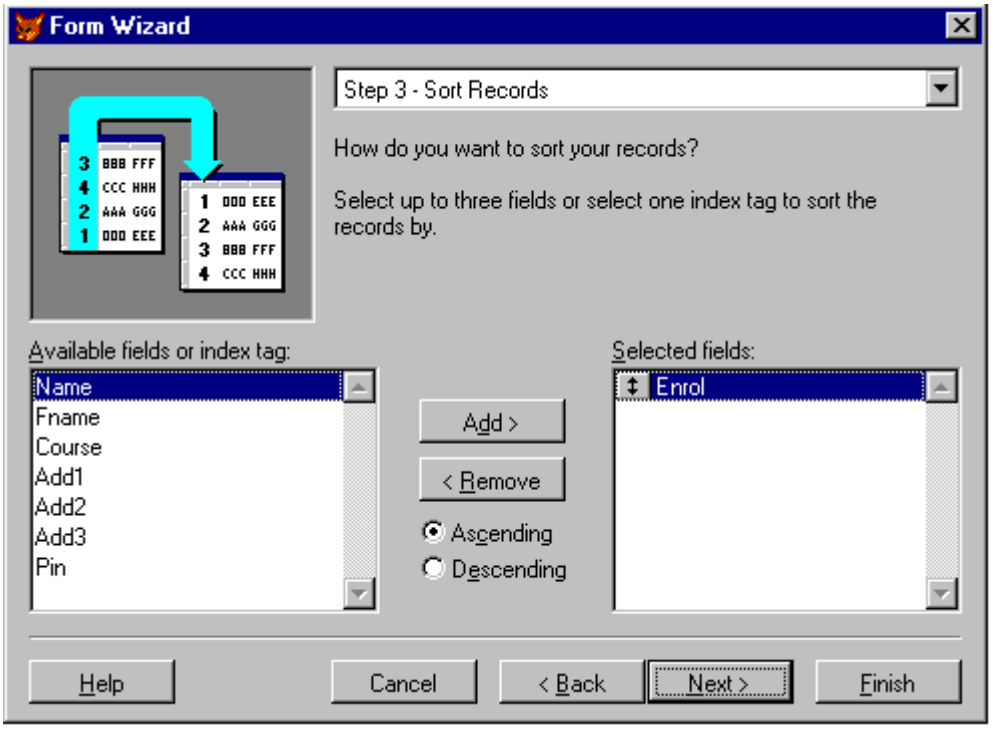


Fig. 8.5: Sort Records of Form Wizard Dialog box

You can also sort the data in ascending or descending order by clicking the required radio button available in this Form Wizard dialog box. Choose **Next** push button to proceed further to the **Finish** step of the Form Wizard.

8.3.5 Generating a Form

The last step of the Form Wizard is the Step4-Finish Form Wizard as

shown in the figure 8.6. Here you need to give the title of the form. The text appears at the title bar and it displays the name of the table as the default name of the form. You can also have three options:

- Save the form for later use
- Save and run the form
- Save the form and modify it in the Form Designer.

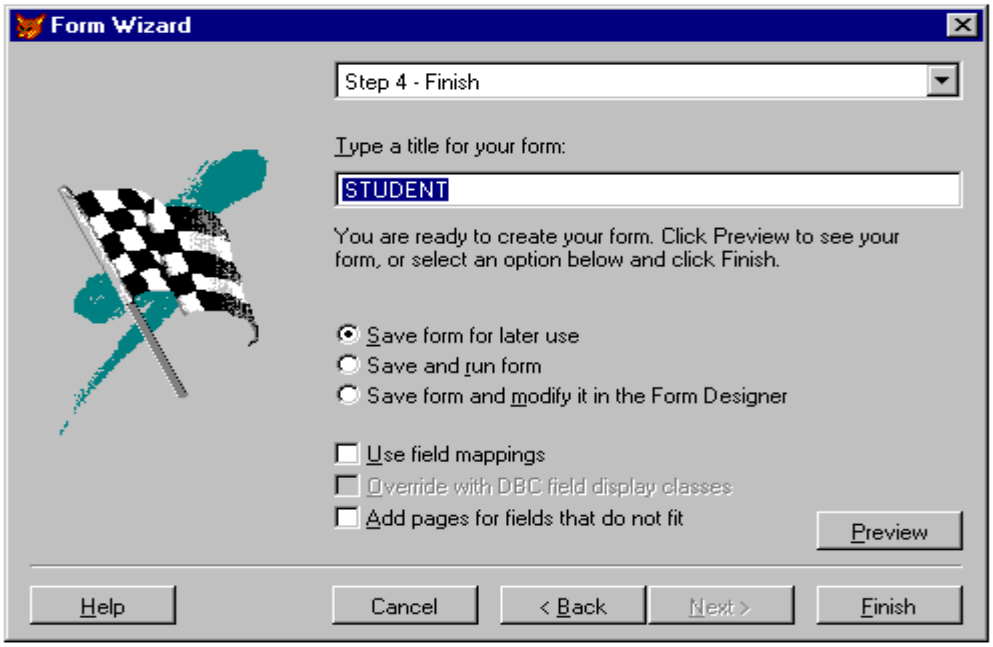


Fig. 8.6: The Finish Step of the Form Wizard

The first option lets you save the form for later use and returns you to wherever you were before you standardized the Form Wizard. The second option saves the form and immediately runs it. The third and last option saves the form and immediately opens the Form Designer with the new form allowing further customization.

You can also click the Preview push button from the Step 4 - Finish of Form Wizard to create the form based on the definitions provided. This is only to see the whether the form created is as per the requirement or not. The form is partially functional in this mode as you can see from the dimmed buttons at the bottom in the fig. 8.7 (Note that PIN field contains a comma after 3 digits as its type has changed to character). However, you can move through the records in the table to verify the order. But the other push buttons are not available until you generate the form.

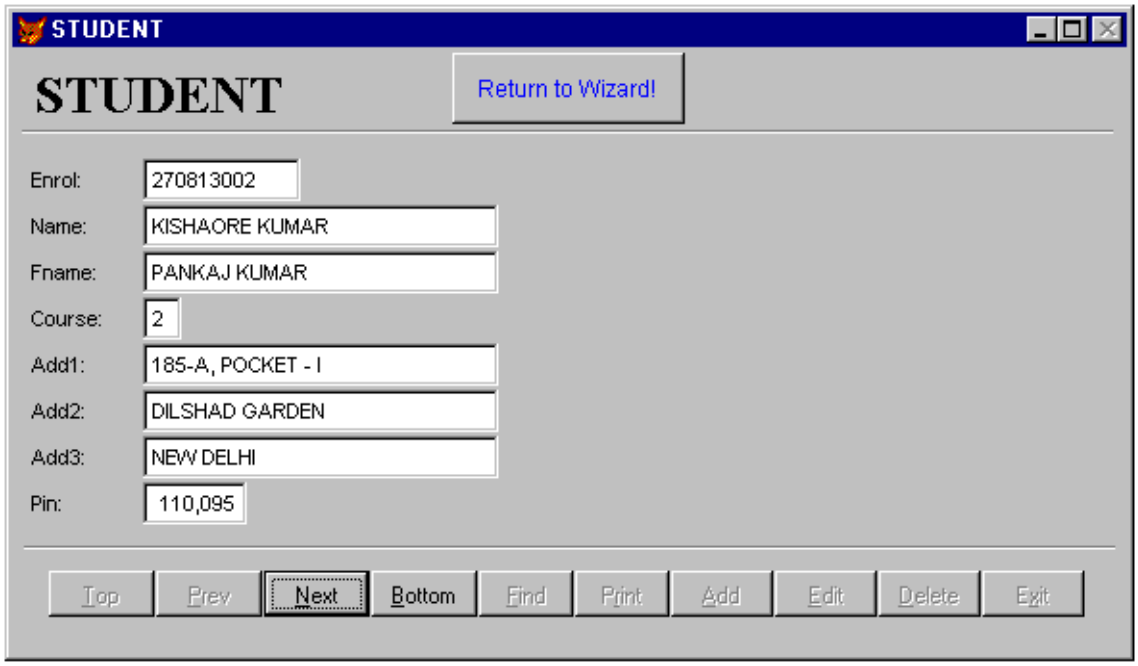


Fig. 8.7: Previewing the Form

Click the **Return to Wizard!** Push button to return to the last step of the Form Wizard. You can do the required changes in the Form at the appropriate step by using the **Back** push button. You can move between the steps and the preview mode as many times as you wish to finalize the Form.

After you click **Finish** push button, Visual FoxPro displays the Save As dialog box to let you name the form. Later you can make changes in the form through Form Designer only. Remember, you cannot re-enter the Wizard for an existing form. Let us save the form with the name STUDENT.

8.4 RUNNING A FORM

Like a Query, Visual FoxPro Form is a program, and you can run it the same way. Choose **Do** option from the **Program** menu. **Do** dialog box appears to select the form or choose **Run** button from the toolbar. You can also run a form by simply typing the following in the command window

```
Do Form Student
```

Fig. 8.8 shows an ordinary control panel with text buttons of a form.

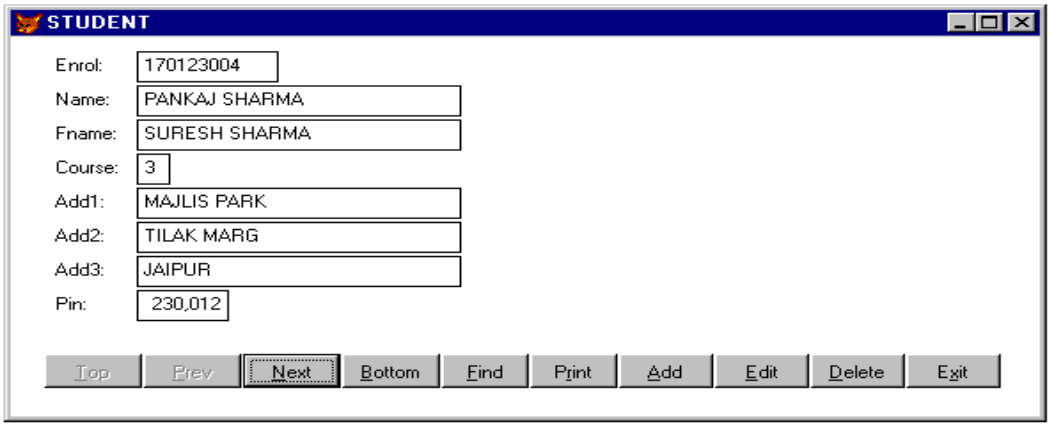


Fig. 8.8 : Visual FoxPro Form with Text Buttons on the Control panel

8.5 CREATING A ONE-TO-MANY FORM WIZARD

The One-to-Many Form Wizard is similar to the form Wizard that was covered in the previous section of this lesson. The primary difference is that it creates forms using two tables. More specifically it creates a form to display individual parent records and all of their children. The parent record fields typically appear on the top half of the form using individual text box fields. The child records generally appear in a scrollable browse like grid half of the form.

To create a One-to-Many Wizard, choose **New** option from **File** menu or click the **New** tool box. The **New** dialog box appears. Select the **Form** radio button and then click the **Wizard** push button. Visual FoxPro displays the **Wizard Selection** dialog box as shown in fig. 8.9.

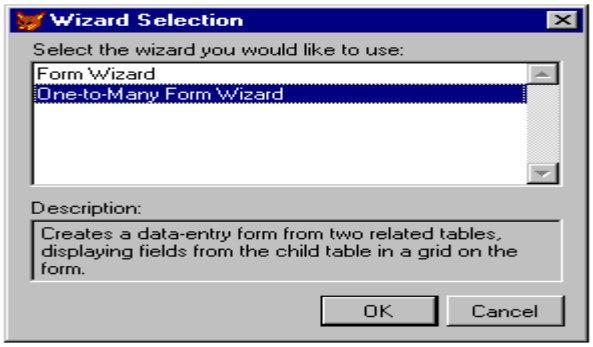


Fig. 8.9: Form Wizard Selection dialog box

Choose the **One-to-Many Form Wizard** form **Wizard Selection** dialog box. The "Step 1- Select Parent Table Fields" dialog box appears to select the fields from the main or parent table. It is similar to the first step of the regular form Wizard shown in fig. 8.10.

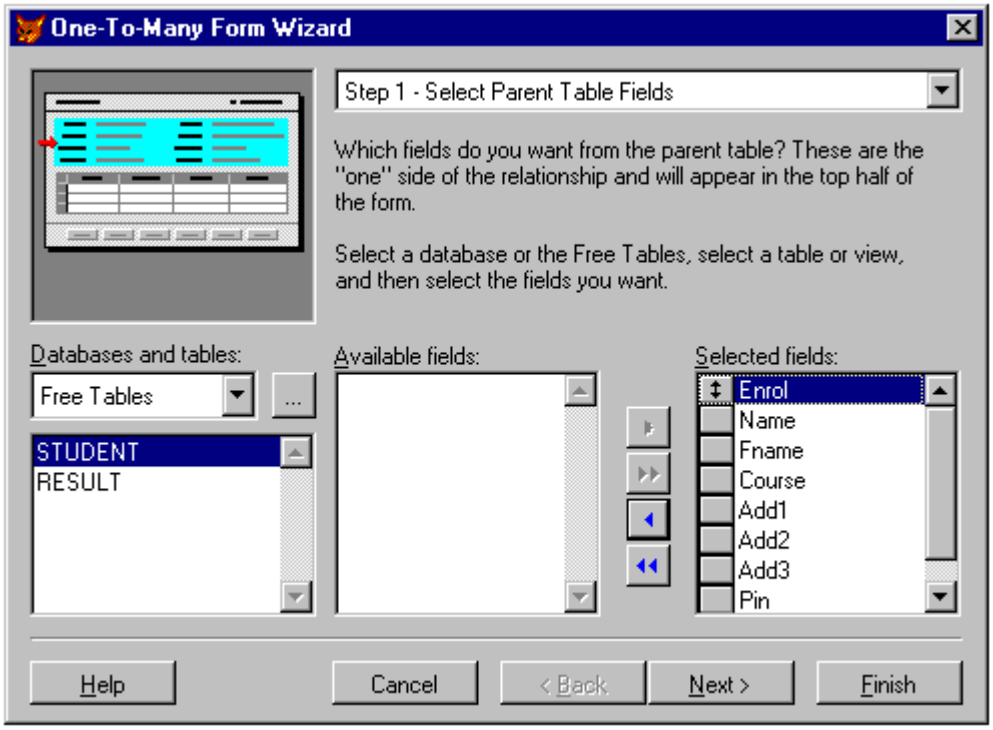


Fig. 8.10: Step1 Select Parent Table Fields

As with a normal Form Wizard, you start by selecting a database or table to use as the main or parent table. From the **Available Fields list,** select fields which are to be included on this form.

For example, click the ellipsis (...) button to the right of Database/ Table option and use the **Open** dialog box to choose "STUDENT" as the main table. Click the double-right arrow to select all the fields from the **Available fields** list to **Selected Fields** list, as shown in Fig. 8.10.

Please note the small, red arrow pointing to the top half of the form in the form image area in the upper left corner. This visually reminds you where the main data appears in the form.

Click the **Next** push button to go to "Step 2: Select Child Table Fields" as shown in Fig. 8.11.

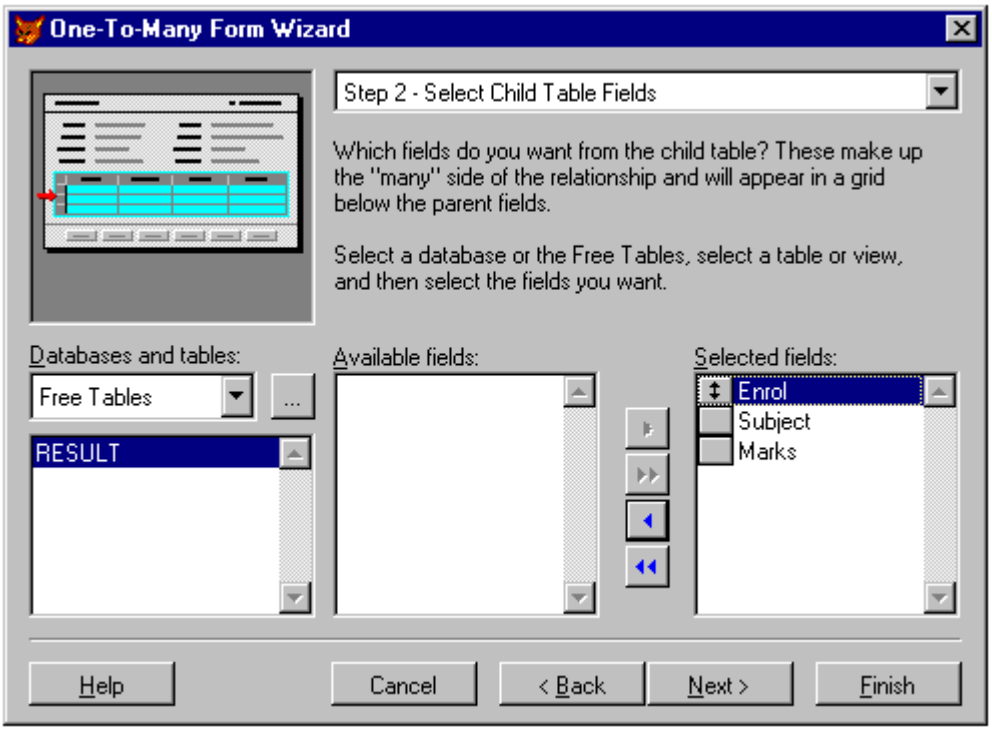


Fig. 8.11: Step 2 Select Child Table Fields

The second step looks very much similar to the first step, except that in this step prompts for the fields from the related or child table is displayed in the **Available Fields** list. Note that the red arrow in form image has shifted down to the lower half of the screen. All child records appear in a browse like control. Therefore, all fields must fit horizontally across the form. If you select more fields than those which can fit horizontally, the wizard adds a horizontal scroll bar to view them.

Click again the ellipsis (….) button to the right of the Databases/ Table option and choose “RESULT” as the related or the child table, click the double- right arrow to add all the fields from the **Available Fields** list to **Selected Fields** list, as shown in Fig. 8.11.

Note that the most important fields when creating a one-to-Many form are the fields that define the relation between the parent and child. In this case, ENROL appears in both STUDENT and RESULT tables. Choose **Next** push button to proceed further to the Step 3 - Relate Tables of the One-to-Many Form Wizard.

The Wizard sets this relation between the Parent table and the Child table in step 3 automatically when a persistent relation exists between the two files in their database. Otherwise, you can easily establish the relationship using the drop down field lists from both the tables as shown in the Fig 8.12. The relation formed between parent and child table is shown by the line joining the two Tables.

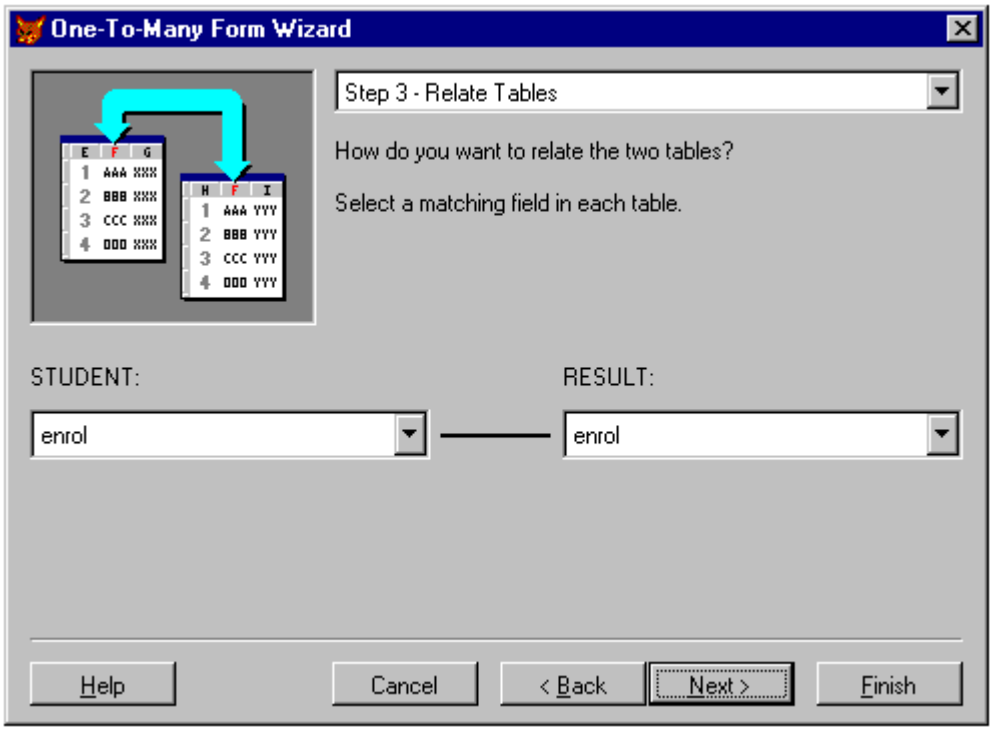


Fig. 8.12: Step 3 - Relate Tables of the One-to-Many Form Wizard

Click **Next** push button to go to " Step 4- Choose from Style" and select **Standard** as a style and **Text** radio button as control panel button type. The style options are the same as the Form Wizards styles listed in the previous section and shown in figure 8.3.

Click **Next** push button to go "Step 5- Sort Records" and double click on Enrol to use it as the basis of the sort order. This step is also the same as the one we discussed in the Form Wizard and shown in Fig. 8.5. Click **Next** push button to go to the "Step 6- Finish" dialog box. In this last step as in the case of Form Wizard, you need to give the title of the Form. Enter "**Data Entry Module**" as the title of the form. Select the "**Save and Run Form**" radio button and click **Finish**

push button. In the **Save As** dialog box, enter the name of the form (STUDENT3.SCX), and click **Save** push button. Visual FoxPro generates the forms as shown in Fig. 8.13.

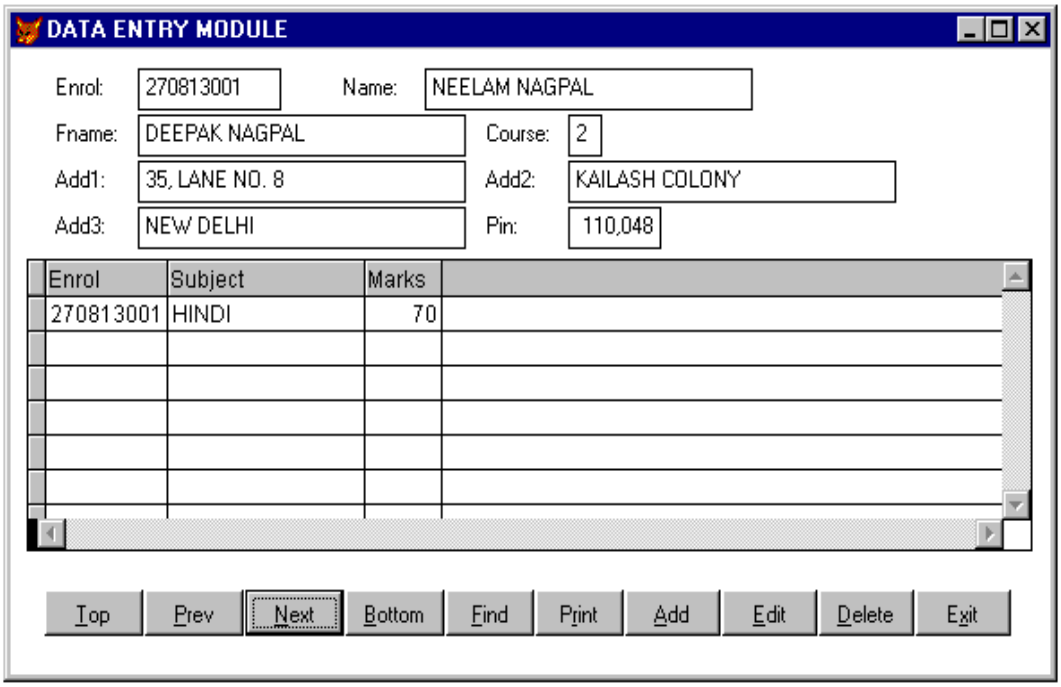


Fig. 8.13: Form Wizard with two tables

Now, try using your "Data Entry Module" form with all the available features at the control panel and choose **Exit** push button, when you have finished. Remember that once you save the form and exit the wizard, you cannot return to wizard to make changes. You must make all subsequent changes using the Form Designer, which we are going to discuss in the next lesson.

INTEXT QUESTIONS

1. Write True or False for the following statements

   (a) Forms provide the basic means for inputting data for the application.

   (b) Form wizard helps you to create a form step by step.

   (c) There is only one type of form style available.

   (d) You can have only text buttons in your form.

e. Can you create a blank form i.e. without using the tables.

2. Fill in the blanks:

   (a) The extension of a form file is given as ————————

   (b) One-to-many form is created using ————— tables.

   (c) In the last step of form wizard the ————————— of the form is given.

3. Where do you require a Form?

8.6 CREATING A FORM USING FORM DESIGNER

The Form Wizard has its own limitations for developing a form. These limitations are done away with the help of Form Designer. The Form Designer is the powerful tool available with Visual FoxPro to design your form as per your requirement. You also use Form Designer after you create a Form using Form Wizard for all subsequent changes you wish to do with the Form.

Visual FoxPro object oriented programs support Form as a Visual object. A Form is such an object in which you can place other objects i.e. you can create new forms from existing one. With objects inheritance you can create a basic form definition and then use it to create all subsequent forms for an application. For example, you may wish to define the overall appearance, color, fonts, etc. Then simply save this as a visual class and use it as a template for all new forms.

There are several ways to create a new form:

- Select **New** option from **File** menu. Select **Form** radio button and click the **New File** button. A blank form occurs as shown in figure 8.14. It shows the open Form Designer work area, in which you can build a visual representation of the form.
- You can also create a Form by typing the command **CREATE FORM** in the Command Window.
- You can also use the Form Wizard
- Enter the Command

```
Newform = (CREATEOBJECT("FORM)
New Form.Open
```

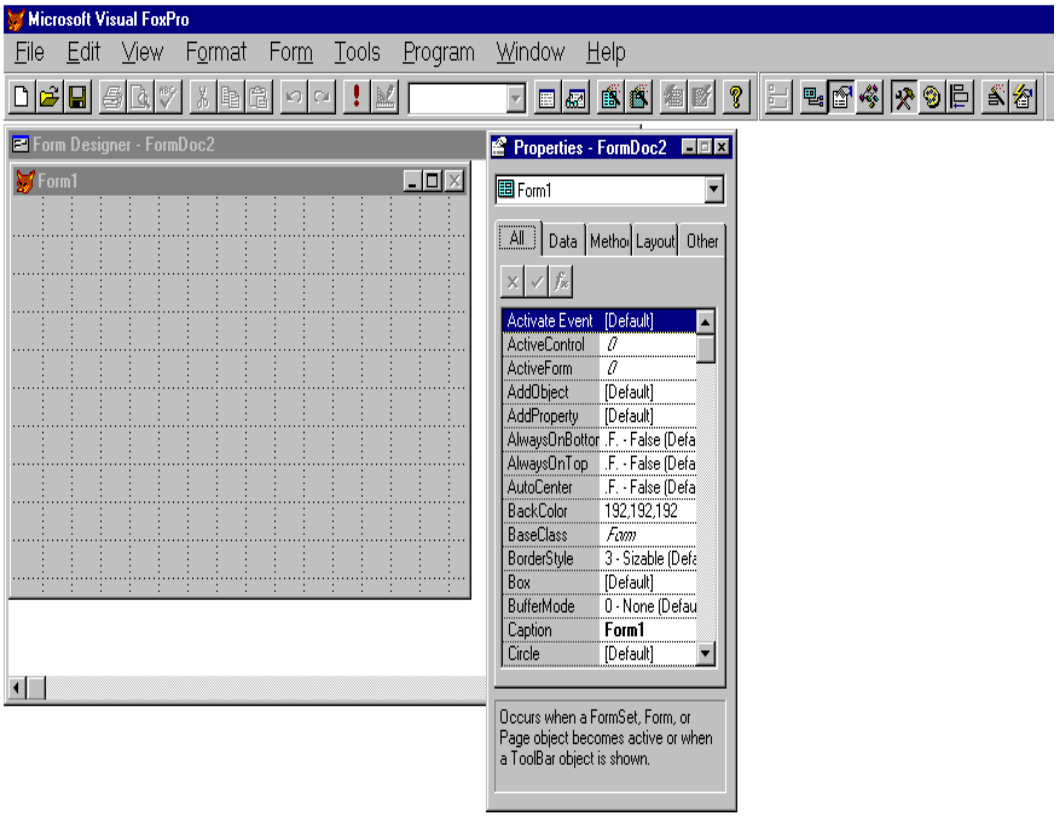


Fig. 8.14: Creating a Form with Form Designer

Similarly, there are several ways to modify an existing form:

- Select **Open** option from the **File** menu. **Open** file box appear on the screen. Choose **Form** from the **File Type** and open the form named STUDENT3.SCX or simply type the name of the form in the Name of the File text box. The form STUDENT3.SCX is displayed in the Form Designer as shown in the Fig. 8.15.
- As discussed earlier, you can also modify a form you have created using the Form Wizard. Enter the following Command in the Command Window and press Enter.

MODIFY FORM <Filename>

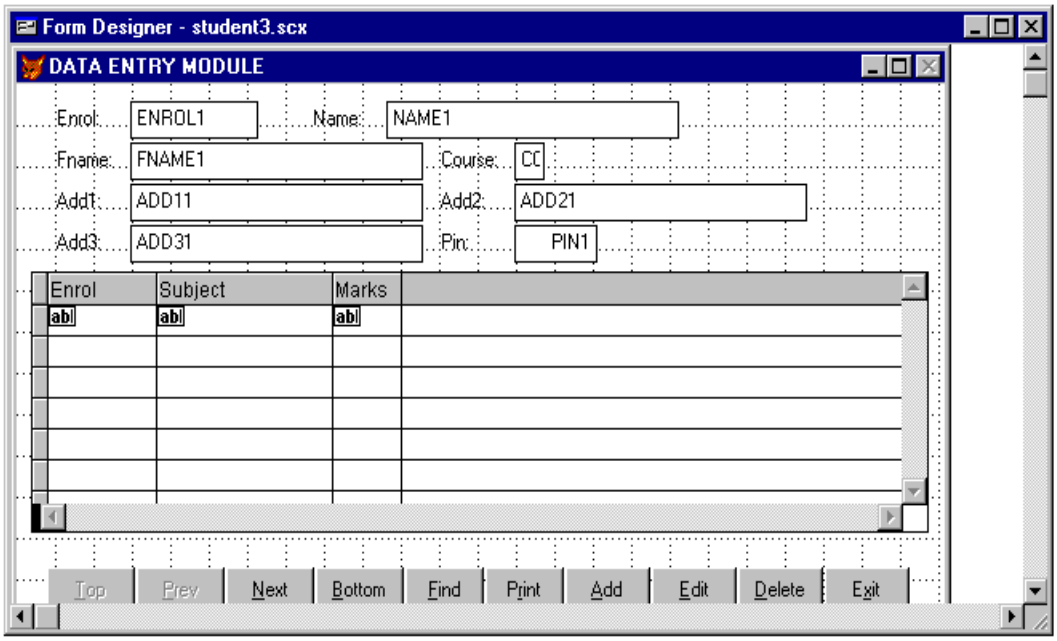


Fig. 8.15 : Modifying an existing Form using Form Designer

You can add various objects into the Form to make it more interactive or as per the requirement of the form using controls toolbar.

8.7 FORM PROPERTIES

Form properties are the features and attributes by which a form is defined. The basic visual appearance of a form is that of a window, its use of properties makes it much more than a simple window. Forms have over 60 properties. When you create a form, all properties have default values. In order to make the form more and more interactive, you need to specify the properties, which you want to change. When you specify or change the properties of a form, you actually do the coding for the Visual FoxPro. Fortunately, most of the time, only a few properties need to be changed. The most obvious of these include:

- Size
- Position on-screen
- Title
- Foreground and background colours

Open the properties window to see the current settings. Visual

FoxPro divides object properties into four frames: Data, Methods, Layout and Other. Of course, you will find one more frame, **All**, which includes all properties.

To change one, click it to select or highlight it with the arrow keys. Visual FoxPro places the current value in the edit box at the top of the page. Fig. 8.16 highlights the Caption property and has already changed the default value of Form1 to Data Entry Form. You will notice that the change is immediately reflected in the actual form's title bar.

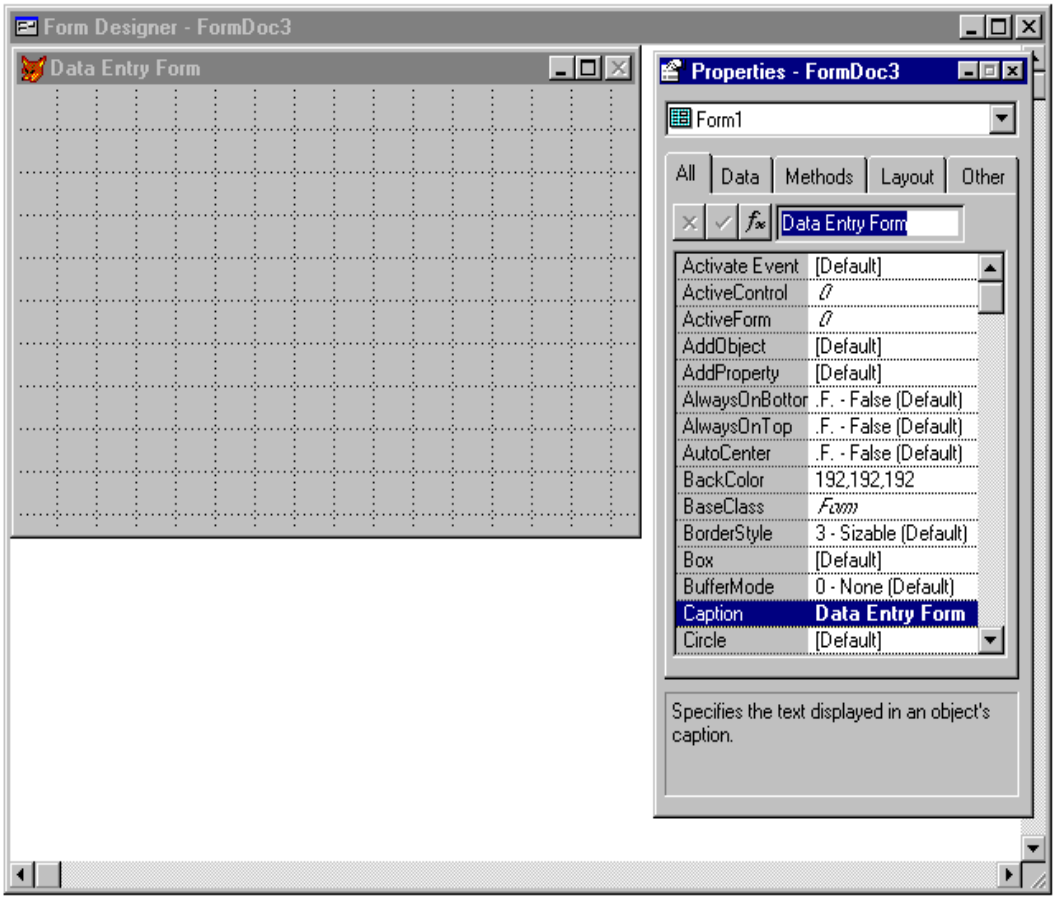


Fig. 8.16: The Form Designer with change Form Title

A few properties such as **ForeColor** and **BackColor** display a button with three dots or an ellipsis. This button indicates that another dialog box exists with options for this property. In this case, clicking the button opens the **Colour** dialog box, which displays a grid of 48 pre-defined colours. It also allows you to define, custom colours.

8.7.1 Data Environment

Almost every form needs to refer data from one or more tables. To include the required tables in the definition of the form, choose **Data Environment** option from the **View** menu. This opens a window titled Data Environment. Here you can add the required table for your Form.

8.7.2 Adding Controls to a Form

After defining the data environment, the next step is to add controls to the form. Visual FoxPro supports several types of controls. You can add controls to a form using Form Controls Toolbar. Here you add all the fields of the table required for your form.

Label Controls: Labels are simple text strings added to a form to identify fields or to display fixed character information to the user. A label can be a single line or it can span several lines. The most common label properties include the capability to change font, size and style. You can also define a background colour to a label. A simple box border can also surround the text of your label. Labels do not have a data source. You enter the text directly into its caption property. Remember, when the program is running, the user cannot edit or tab into a label.

Text Box Controls: The primary difference between a label and text box is their respective data source. For a label, the text is fixed as the Caption property for the object. On the other hand, the source for a text box is usually a table field, but it could also be a memory variable. The most common properties modified for a text box include those related to colour, font, size, position of the text box and the data source.

8.7.3 Specifying the Tab Order

To move from one field to the next by using mouse, is not always the most convenient way to do so. User can press **Enter** or **Tab**, once they are through with the current field and are ready to move to the next.

By default Visual FoxPro defines the tab order by the way you add the fields to the form. However, you may add fields later that did not exist in the original form, then by the default order the cursor will move to such field only at last. You can change the tab sequence by using the **Tab** Ordering pull-down list.

8.7.4 Using List Boxes

A list box is a control box that allows you to create a scrollable list from which the user can make a selection. The List Box control on the Forms Control toolbar allows you to create a list box on your form.

8.7.5 Using Combo Boxes

A combo box allows you to enter a value or select an item from a list. It is a combination of a text box and a list box. The Combo Box control on the Forms Control toolbar allows you to create a combo box on your form.

8.8 WHAT YOU HAVE LEARNT

In this lesson you have learned about the Form Wizard, which supports two form styles. The first is the simple form based on a single table. The second creates a on-to-many form. You have also seen the limitations of the Form Wizard and to break beyond those limits, you should use the Form Designer.

In the Form Designer, you have learned about the basics of creating a form and the usage of advanced form design controls such as labels, text boxes, list boxes and combo boxes to create more user-friendly applications.

8.9 TERMINAL QUESTIONS

1. Explain the different steps involved in creating a form using form wizard.
2. What do you understand by parent table and child table and when are they used?
3. Write steps for creating a form-based on a table.
4. Write command for running a form from command window.
5. Write steps for creating a control button 'Revert' to the existing form as in Fig. 8.7.
6. What is the difference between creating a form through Form Designer and Form Wizard?

8.10 KEY TO INTEXT QUESTIONS

1. (a) T (b) T (c) F (d) F (e) T

2. (a) .scx

   (b) two

   (c) title

3. A Form is used to present the data/information on screen/ paper from the stored huge database/tables. It can also contain text without data from Tables.

9

CREATING REPORTS WITH REPORT WIZARD AND REPORT DESIGNER

9.1 INTRODUCTION

Till now you have learned about creating Table, Query and Form using the respective Wizard and Designer mode. Every application has various types of reports. Report wizard and Report Designer are the main tools available in Visual FoxPro for taking outputs. These report tools are designed to meet a range of needs and requirements from very simple and limited to more complex and versatile. Visual FoxPro provides two types of report wizard: Report Wizard and One-to-Many Report Wizard.

The Report Wizard formats a single table report. The One-to-Many Report Wizard is like the One-to-Many Form Wizard, which uses a parent-child relation between two tables to create a report.

9.2 OBJECTIVES

After going through this lesson, you would be able to

- create a Report with Report Wizard
- create a Report with One-to-Many Report Wizard

- create a Report with Report Designer
- group data in Report Designer
- add Title and Summary Bands in Report
- print Report

9.3 CREATING REPORT WITH REPORT WIZARD

Reports can be created using:

(i) Report Wizard

(ii) Report Designer

To create a report by using Report Wizard, choose **New** option from **File** menu or directly choose **New** from the toolbar. The **New** dialog box appears with which you are quite familiar, as you have seen during creating Table, Query and Form using the respective Wizard. Choose **Report** radio button and then click **Wizard** push button. Visual FoxPro displays the **Wizard Selection** dialog box as shown in figure 9.1. There are two types of report wizard available in this dialog box **One-to-Many Report Wizard** and **Report Wizard**. You have to choose one of them. Let us select the Report Wizard to display the "Step 1 - Select Field " page of the **Report Wizard** dialog box.

Fig. 9.1: Selecting a Report Wizard

9.3.1 Selecting the Table

The first step in the Report Wizard, like the first step in the Form Wizard, is to select the table and the fields you want in your report. This dialog box is similar to the one you have seen in Query and Form wizards. You first need to select the database, followed by the table and then the individual fields you want to include in the report.

The **Database/Tables** option available with the first step of Report Wizard dialog box, displays the current database, if one is open. Otherwise it displays the words Free Tables. If a database is open, the names of the tables it contains appears in the scrollable list and you can choose the required table. If the database is not open, or if you want to use a free table, click the ellipsis button **(...)**, which opens a dialog box that allows you to select the required table as shown in figure 9.2. Let us select the STUDENT.DBF table from the open dialog box for creating a report.

9.3.2 Selecting Fields

The "Step 1-Select Fields" dialog box also includes Field picker buttons or movement buttons, which are found between the **Available fields** list and **Selected fields** list. The arrow directions indicate the way they move fields. A single arrow moves a single field. Double arrow buttons move all the fields. To move a field from the Available fields list, first select the field you want to move by clicking or by using the up and down arrow keys and then click the top buttons with the right arrow. This moves the highlighted field to the Selected fields list. Repeat this process to move more fields from the Available fields list to Selected fields list as shown in the figure 9.2.

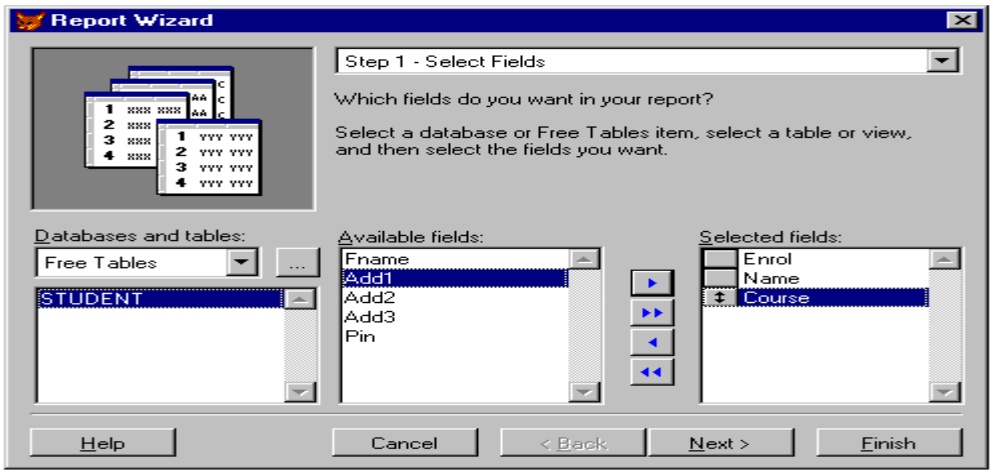


Fig. 9.2: Step 1 - Selecting Fields for the Report Wizard

If you want to move all the fields from **Available fields** list to **Selected fields** list, just click the double-right arrow. If you have selected the wrong fields or change your mind, then you can deselect a field from the Selected fields list. Select that field and click the left arrow. Similarly, you can choose the double-left arrow to move all the fields from the Selected fields list to Available fields list. Choose **Next** push button to proceed further.

Let us select *Enrol*, *Name* and *Course* fields from the Available fields and move them to Selected fields list as shown in the figure 6.2. Choose **Next** to move to next step of the Report Wizard.

9.3.3 Group Records

The Step 2 - Group Records of Report Wizard allows you to define grouping of records. You can select up to three levels of grouping as shown in the figure 9.3. In the three different levels of grouping of records, you have to select the field on which the grouping is done or select **<none>** if no grouping of records is required. Choose **Next** to proceed further.

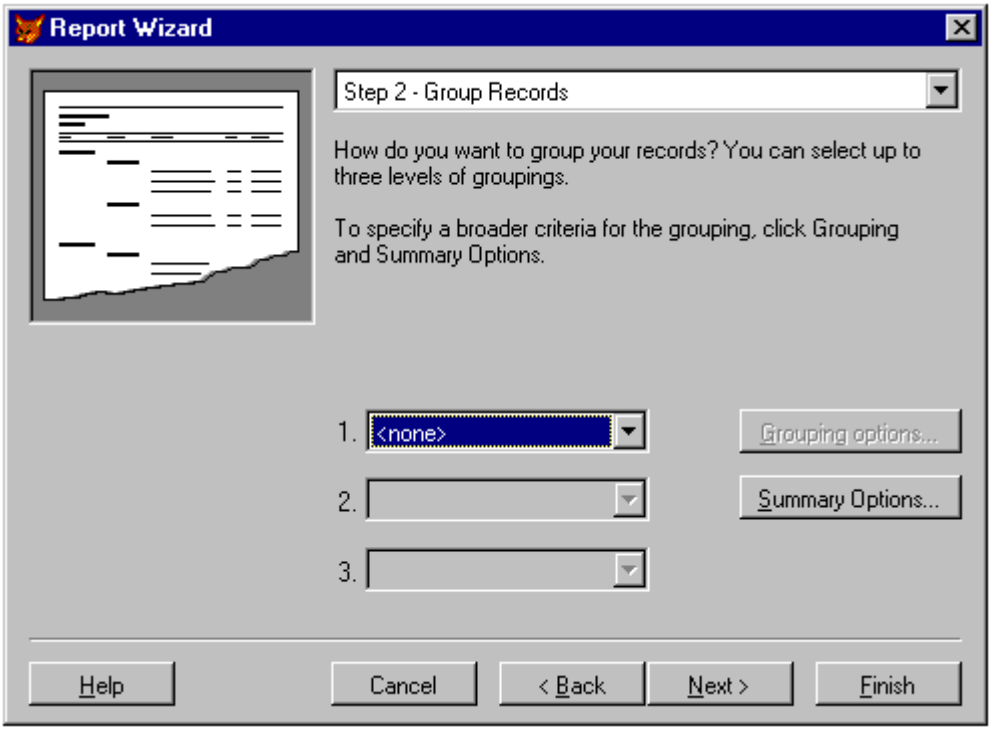


Fig. 9.3: Group Records of Report Wizard

9.3.4 Selecting Style

The Step 3 - **Choose Report Style** of **Report Wizard** allows you to define the over all report style. Visual FoxPro provides five standard report styles

- Executive
- Ledger
- Presentation
- Banded
- Dff

In the upper left corner of the dialog box, you can preview an illustration of each style as you select it, see figure 9.4.

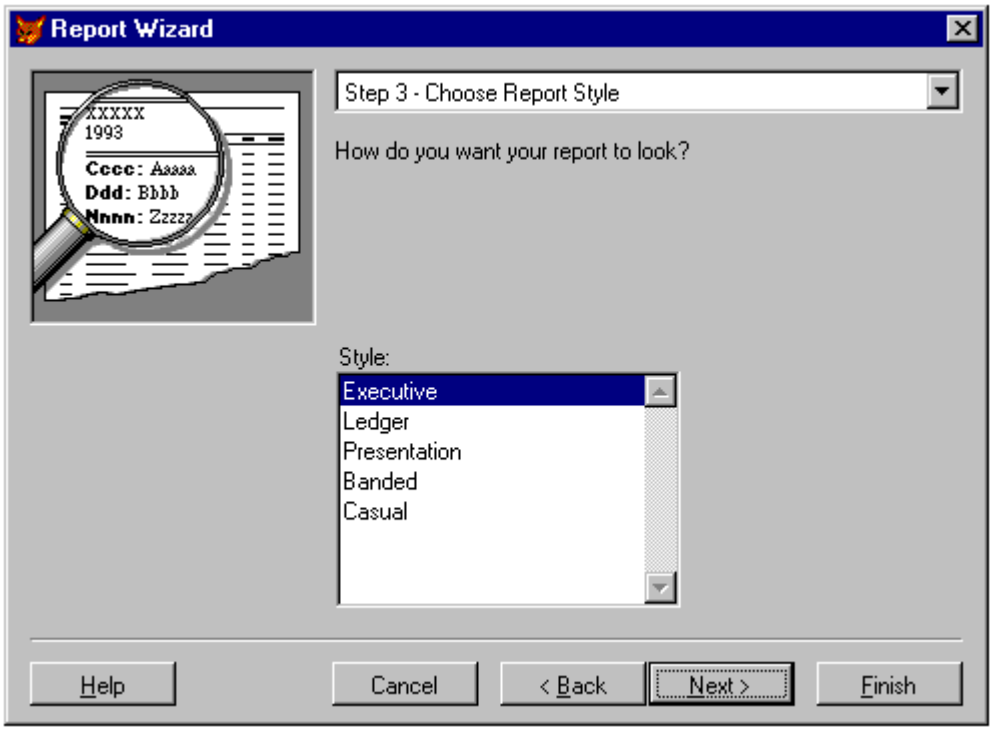


Fig. 9.4 : selection of Style of the Report Wizard

9.3.5 Specifying Layout

The step 4 - Define Report Layout of Report Wizard allows you to define the over all layout of the report as shown in the figure 9.5. The first option defines the field layout. The two options are either

Columns or Rows. If you have few fields which can fit across the page horizontally, consider the Column style report because it fits more records on a page. However, if you have many fields in each record to display, you may have to consider a Row style layout.

Use the **Orientation** buttons to identify the paper orientation, either portrait or landscape. The **number of columns** determines the number of columns or repeated data that can appear across a page. For example, if you use a row style field layout, the data may not extend beyond the middle of the page. In this case, you can consider having two columns on the page to double the amount of information printed on each page.

Choose **Next** to proceed further.

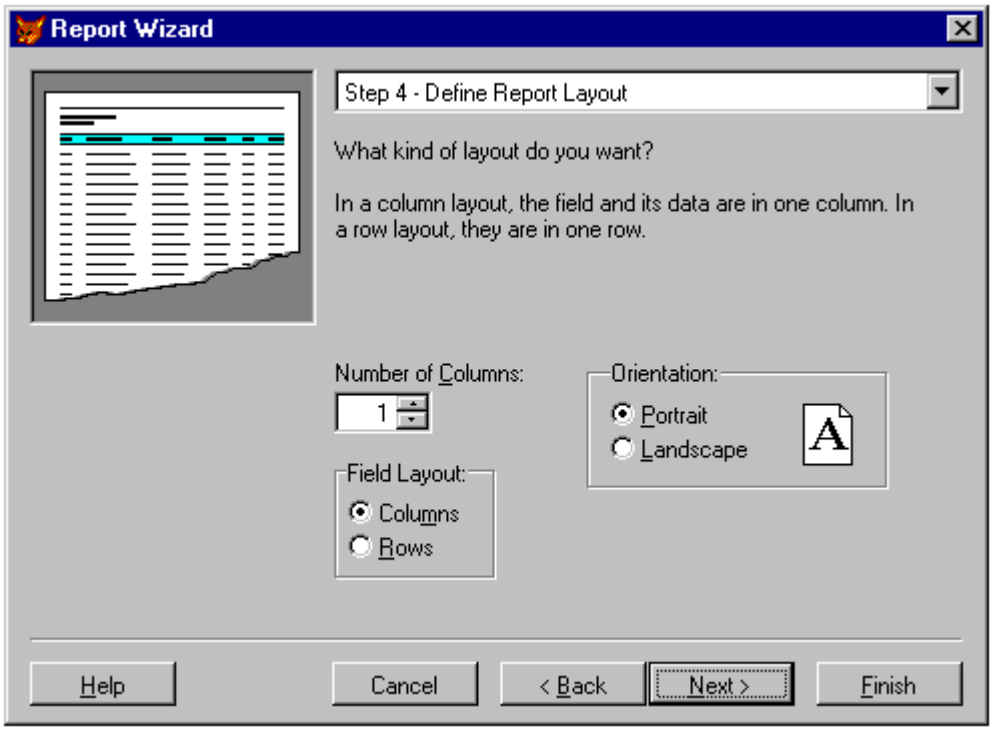


Fig. 9.5: Define Report Layout

9.3.6 Specifying Sort Order

Choosing **Next** option brings you to the "Step 5 - Sort Records" of the Report Wizard. It allows you to specify the sort order for the records, as you do in other wizards. You can select any field from the Available Fields list and move it to the Selected Fields list by

clicking the **Add >** key. You can select even more that one field for the sorting order. However, the order in which you select these fields determines their sort priority. You can change the sort priority by moving the selected field using the double-headed buttons to the left of the field names. Just click a double-headed button and drag it up or down to a new position. If you have selected a wrong field for the sorting order or if you want to remove any field from the Selected Fields list to the Available Fields list, then select the field you want to remove and then click the **<Remove>** push button. Let us select **Enrol** field from the Available Fields list and move it to the Selected Fields list for specifying the sorting order as shown in the figure 9.6.

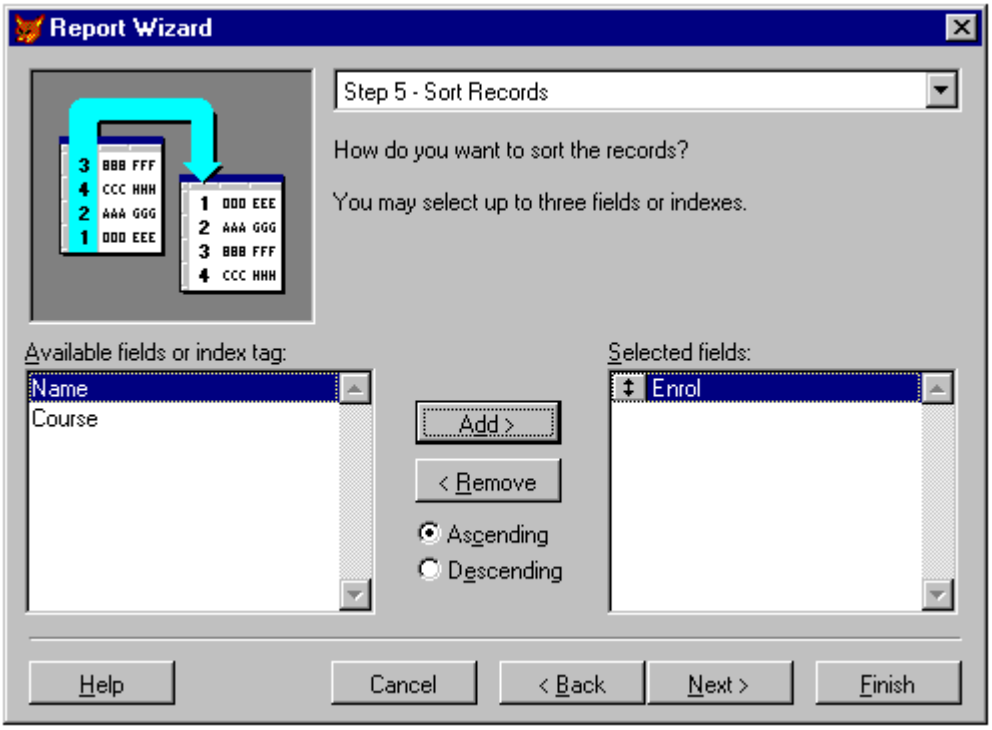


Fig. 9.6: Sort Records of Report Wizard

You can also sort the data in ascending or descending order. Choose **Next** push button to proceed further.

9.3.7 Generating the Report

The final step of the Report Wizard, as shown in the figure 9.7, appears when you click **Next** push button from the Sort Record

Step. Here you only need to supply the report title. The text appears at the top of the report. You can have three options:

- Save report for later use
- Save report and modify it in the Report Designer
- Save and Print report

The first option lets you save the report for later use and then returns you to whatever you were before you started the Wizard. The second option saves the report and immediately opens the Report Designer with the new report allowing further customization. The third option saves the report and immediately prints it.

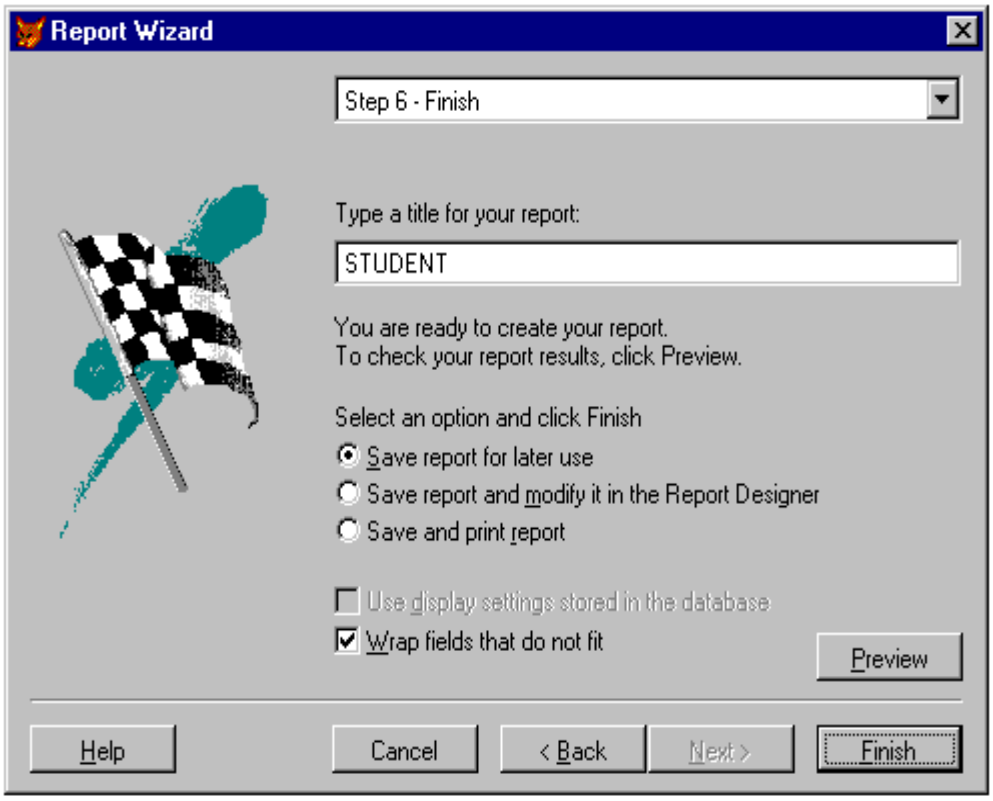


Fig. 9.7: Finish of the Report Wizard

You also have an option to wrap fields onto continuation lines, if they do not fit the column widths defined. You can use the **Preview** push button to preview the reports before actually saving it. Figure 9.8 shows the Preview window of the report.

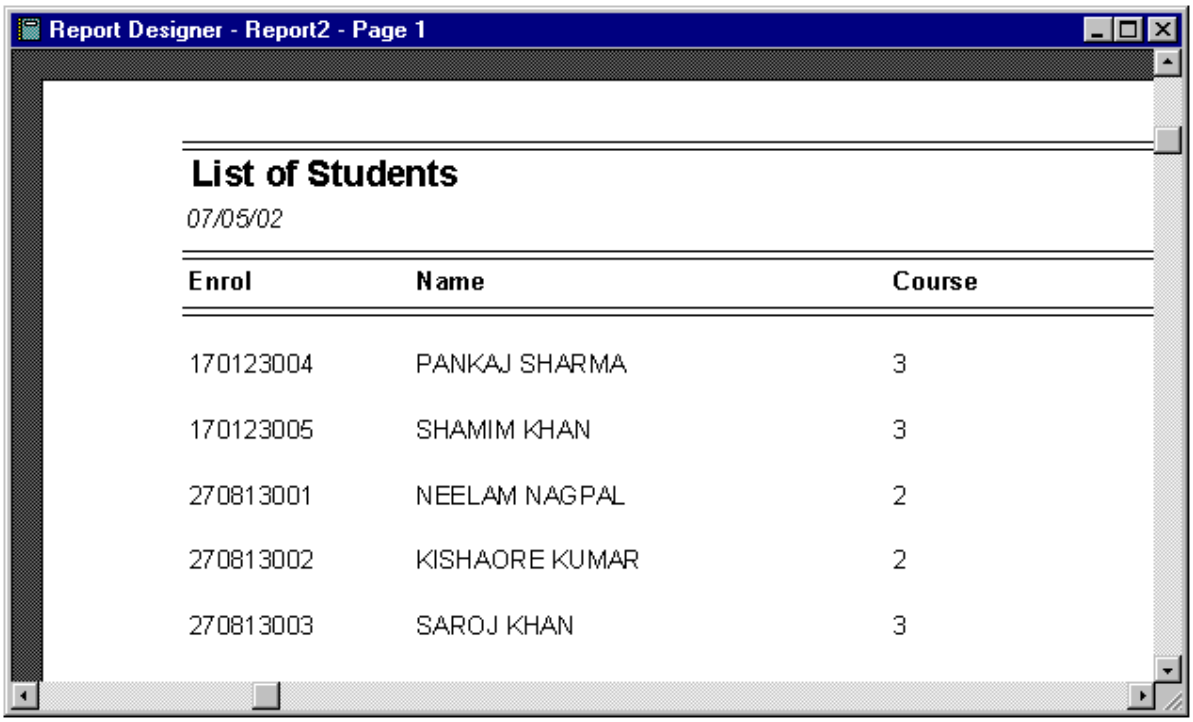


Fig. 9.8: The print Preview Window

You can still return to the previous steps in the wizard by clicking the **Back** button. However, if you save the report any subsequent modifications in the report must be made through the Report Designer. Note that you can not re-enter the Wizard for an existing report. Once you are satisfied with your report, click the **Finish** push button to save your report. The **Save As** dialog box appears on the screen, where you need to mention the name of your report in the text box. Visual FoxPro save the report file with an extension name **.frx** You can save the report as STUDENT.FRX.

9.4 CREATING A REPORT WITH ONE-TO-MANY REPORT WIZARD

The One-to-Many Report Wizard is similar to the One-to-Many Form Wizard. The various steps involved in creating a One-to-Many Report Wizard are similar to the Report Wizard that was covered in the previous section of this lesson. The primary difference is that it creates reports using two or more tables. More specifically it creates a report to display individual parent records and all of their children.

To create a One-to-Many Report, choose **New** option from **File** menu or click the **New** tool box. The **New** dialog box appears. Select the **Form** radio button and then click the **Wizard** push button. Visual FoxPro displays the **Wizard Selection** dialog box as shown in figure 9.9.

Fig. 9.9: Wizard Selection Box

Choose the **One-to-Many Report Wizard** form **Wizard Selection** dialog box. The "Step 1- Select Parent Table Fields" dialog box appears to select the fields from the main or parent table. It is similar to the first step of the regular Report Wizard shown in figure 9.10.

Fig. 9.10: Fields from Parent Table of One-to-Many Report Wizard

As with a normal Report Wizard, you start by selecting a database or table to use as the main or parent table. From the **Available Fields list,** select fields which are to be included on this report.

For example, click the ellipsis (...) button to the right of **Database and Table** option and use the **Open** dialog box to choose "STUDENT" as the main table. Click the double-right arrow to select all the fields from the **Available fields** list to **Selected Fields** list, as shown in Figure 9.10.

Please note the small, red arrow pointing to the top half of the report in the report image area in the upper left corner. This visually reminds you where the main data appears in the report.

Click the **Next** push button to go to "Step 2: Select Child Table Fields" as shown in figure 9.11. The second step looks very much similar to the first step, except that in this step prompts for the fields from the related or child table are displayed in the Available Fields list. Note that the red arrow in report image has shifted down to the lower half of the screen. All child records appear in a browse like control. Therefore, all fields must fit horizontally across the report. If you select more fields than those which can fit horizontally, the wizard adds a horizontal scroll bar to view them.

Fig. 9.11: Fields from Child Table of On-to-Many Report Wizard

Click again the ellipsis (….) button to the right of the Databases and Table option and choose “RESULT” as the related or the child table, click the double- right arrow to add all the fields from the **Available Fields** list to **Selected Fields** list, as shown in figure 9.11.

Note that the most important fields when creating a one-to-Many Report are the fields that define the relation between the parent and child. In this case, ENROL appears in both STUDENT and RESULT tables. Choose **Next** push button to proceed further to the Step 3 - Relate Tables of the One-to-Many Report Wizard.

The Wizard sets this relation between the Parent table and the Child table in step 3 automatically when a persistent relation exists between the two files in their database. Otherwise, you can easily establish the relationship using the drop down field lists from both the tables as shown in the figure 9.12. The line joining the two Tables shows the relation formed between parent and child table.

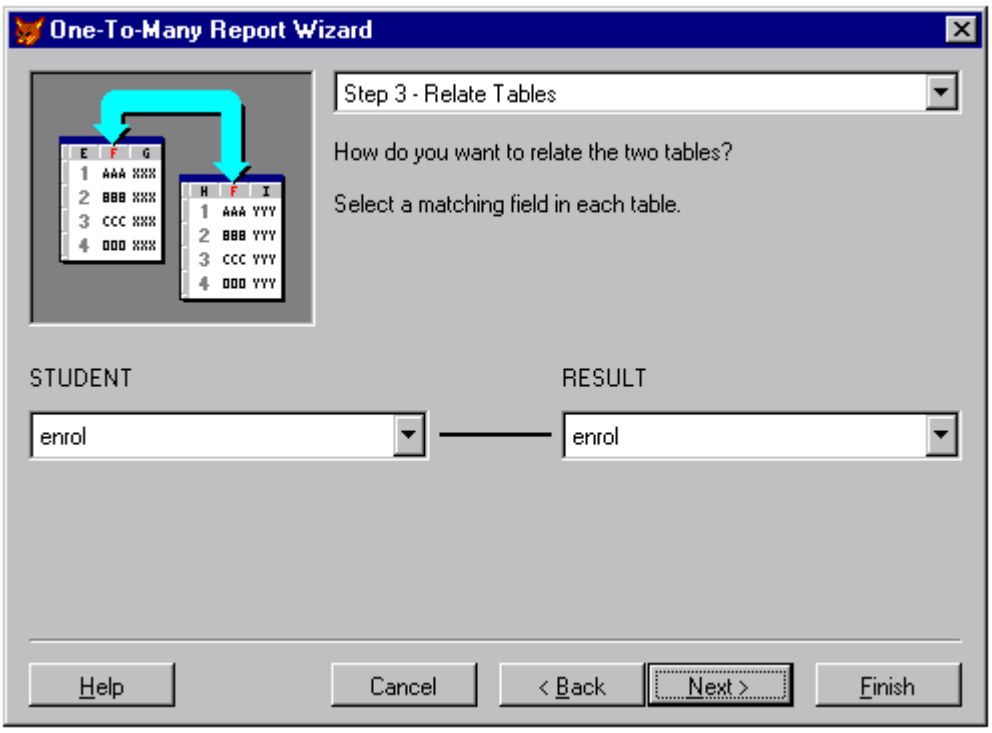


Fig. 9.12: Relation between Parent and Child Table

Click **Next** push button to go to “ Step 4- Sort Records”. Note that the sort is only on fields in the main or the parent table. Double-click on ***Enrol*** as the basic sort order, as shown in figure 9.13.

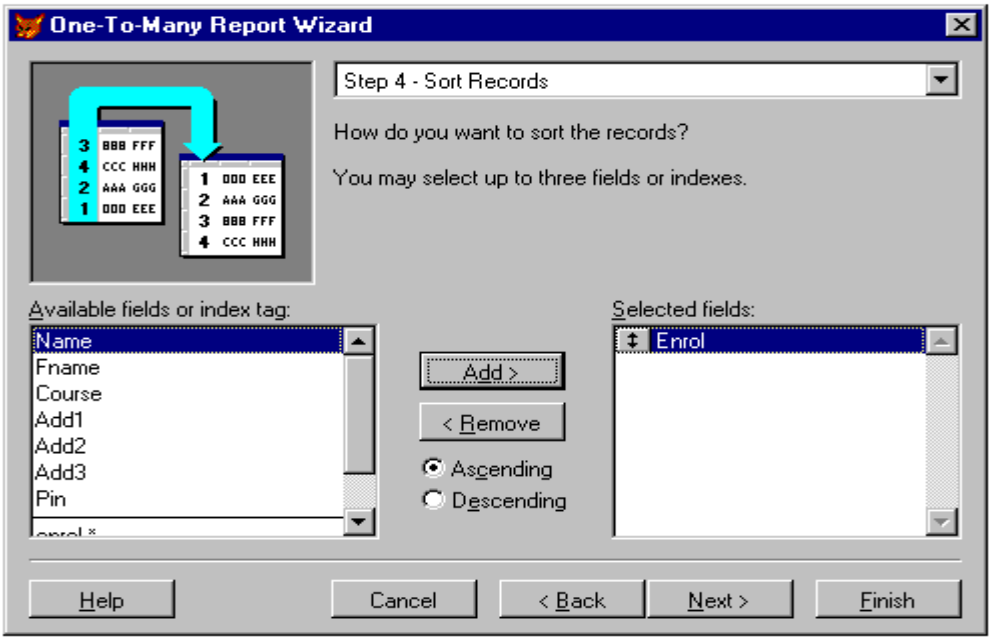


Fig. 9.13: Sort Records of One-to-Many Report Wizard

Click **Next** push button to go "Step 5 - Choose Report Style" dialog box. The fifth step defines the report layout option. This is similar to the normal Report Wizard. Select **Executive** Style and **Portrait** radio button as paper orientation as shown in the figure 9.14.

Fig. 9.14: Report Style of One-to-Many Report Wizard

Click **Next** push button to go to the "Step 6- Finish" dialog box.

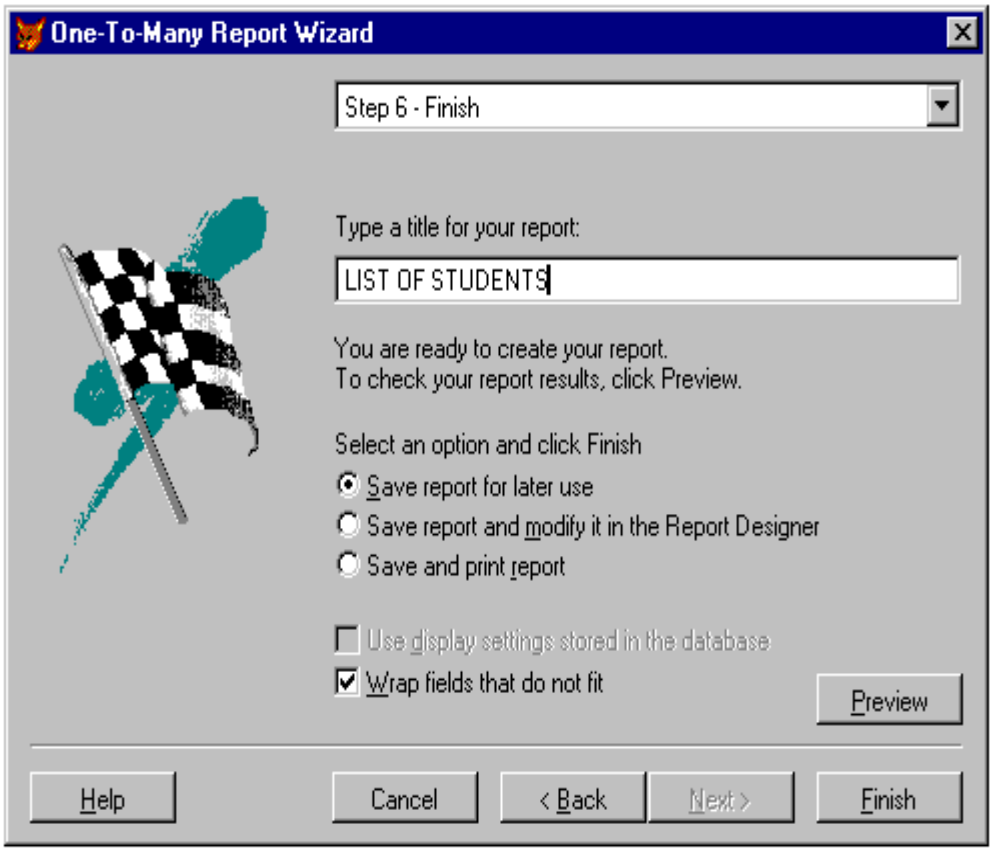


Fig. 9.15: The final step of One-to-Many Report Wizard

In this last step as in the case of Report Wizard, you need to give the title of the Report. Enter "**List of the Students**" as the title of the Report as shown in the figure 9.15. Click the **Preview** push button to see the reports based on the definitions provided. Figure 6.16 shows as the preview window of the report. Click the **Close** Preview button to return to the Wizard. If you want to make some changes in the report, return to the specific step used in its design by clicking **Back** push button and make the required change. You can move between the steps and the preview mode as long as you do not leave the Report Wizard.

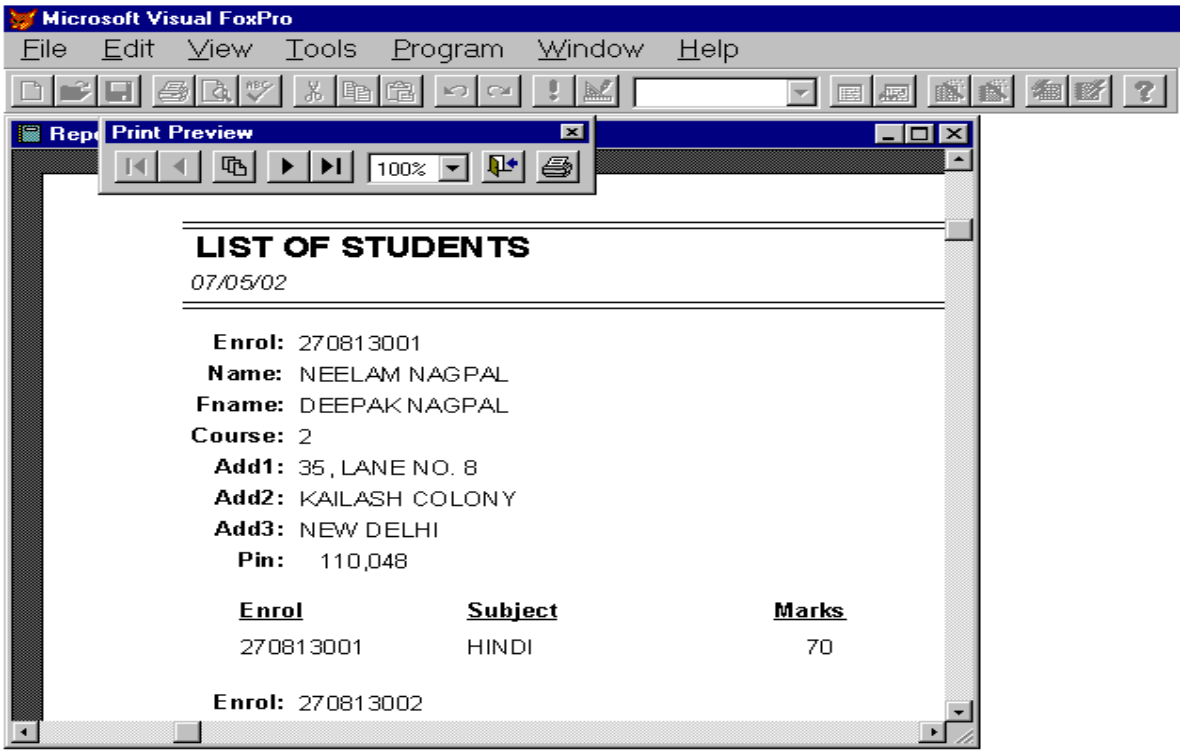


Fig. 9.16 The Print Preview Window

Once you are satisfied with your report, click the **Finish** push button to save your report. The **Save As** dialog box appears on the screen, where you need to mention the name of your report in the text box. Visual FoxPro save the report file with an extension name **.frx** You can save the report as STUDENT1.FRX. Any subsequent changes in your report can be done only through the Report Designer. You cannot re-enter the Wizard for an existing report.

INTEXT QUESTIONS

1. Write True or False for the following statements
   (a) You can select up to three levels of grouping of records.
   (b) You can not select more than one field for the sorting order.
   (c) You can customise the report created through Report Wizard.
   (d) The extension name given to report file is .FRX
2. How do you relate any two tables used in a report?
3. What is the difference between Print and Print Preview commands?

9.5 CREATING A REPORT WITH REPORT DESIGNER

Visual FoxPro provides a powerful tool to create new report through Report Designer. However, it is always convenient and good to create a report through Report Wizard and then open it in Report Designer for making any subsequent changes as per the requirement.

As you know that each report is based on a Database or a Table. Therefore, in order to create a report, first you need to open a table on which your report is based. Choose **Open...** option from the **File** menu and choose **Table** from the **List Files of Type** popup of **Open** dialog box. Select the existing table named STUDENT.DBF from the **File Name** list and choose **OK** push button.

Now to create a new report by using the Report Designer, do the following steps:

- Choose **New...** option from the **File** menu.
- In the **New** dialog box, choose **Report** radio button and then choose **New File** push button. A new report form appears in the Report Layout window as shown in the figure 9.17

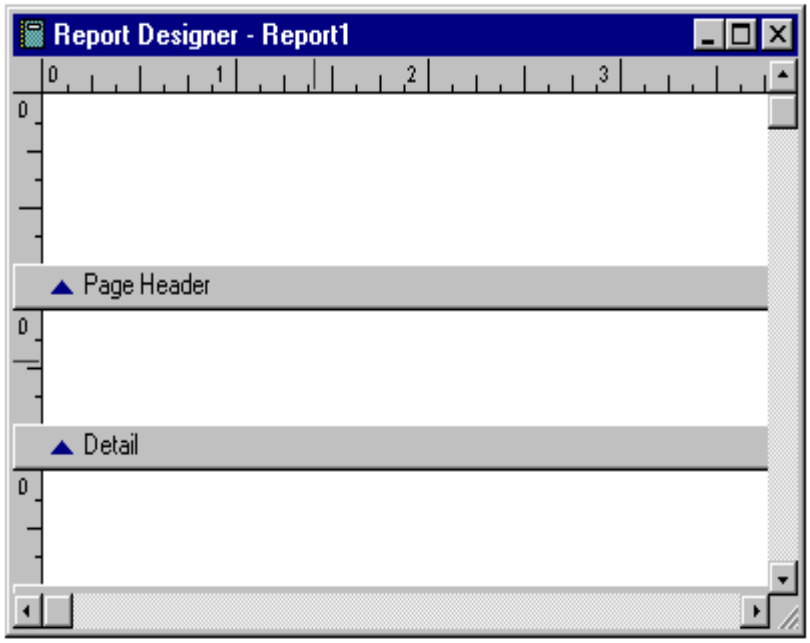


Fig. 9.17: Report Layout Window

You can also create a new report form by typing:

CREATE REPORT Press <RETURN> key in the command window.

A Report Layout Window appears with default title name. You can also type

CREATE REPORT <filename> Press <RETURN> key in the command window to get a new report form named "filename.frx" in the Report Layout.

Choose **Quick Report...** option from the **Report** menu to open the Quick Report dialog box as shown in the figure 9.18. Click the **Fields...** push button from the Quick Report dialog box to open the Field Picker dialog box as shown in the figure 9.19.

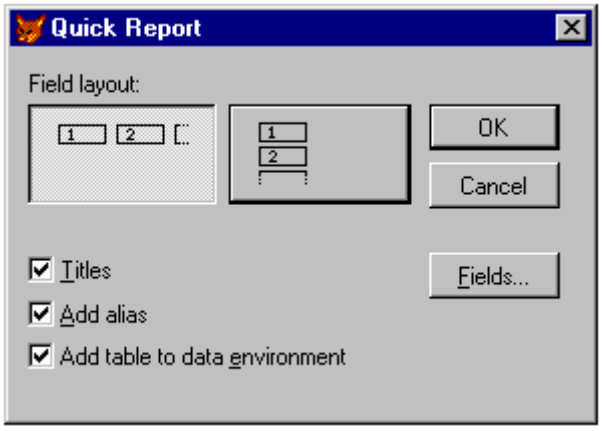


Fig. 9.18: The Quick Report Dialog Box

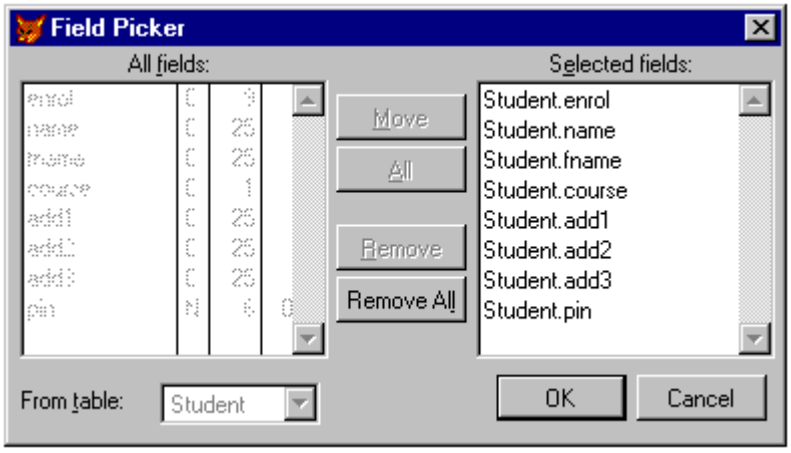


Fig. 9.19: The Field Picker Dialog Box

Double click on the fields, which you want to be included in your report. For example double click on the following fields to move them from All Fields list to Selected Fields list:

Enrol
Name
Fname
Course
Add1
Add2
Add3
Pin

Choose **OK** push button to return to the **Quick Report** dialog box, again choose **OK** push button to return to the **Report Designer** window as shown in the figure 9.20.

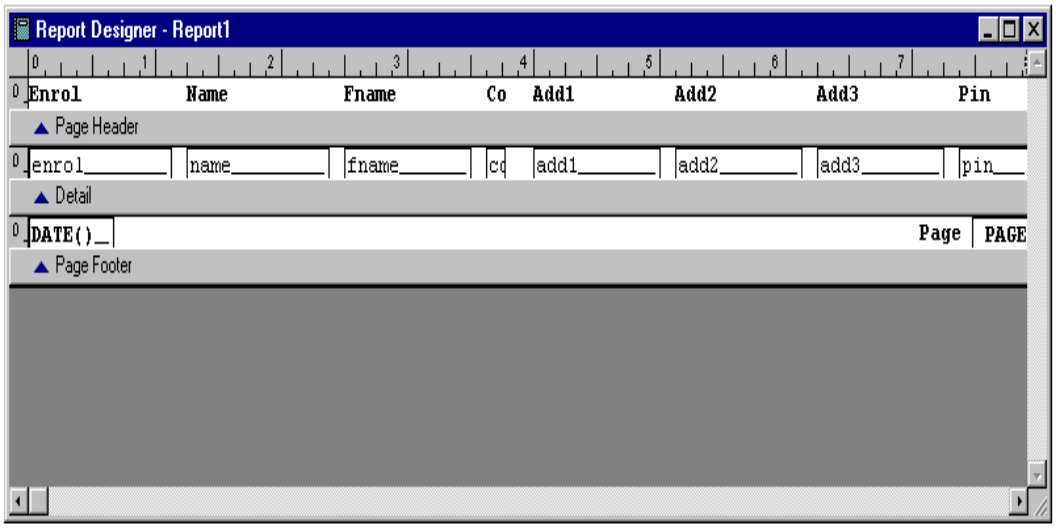


Fig. 9.20: The Report Designer Window

To create some workspace in your report form, click on the **Page Header** band button and drag it down so that the Page Header area looks like as shown in the figure 9.21. Similarly you can create some workspace by clicking on the **Page Footer** band button and drag it down so that the Page Footer band area look like figure 9.21.

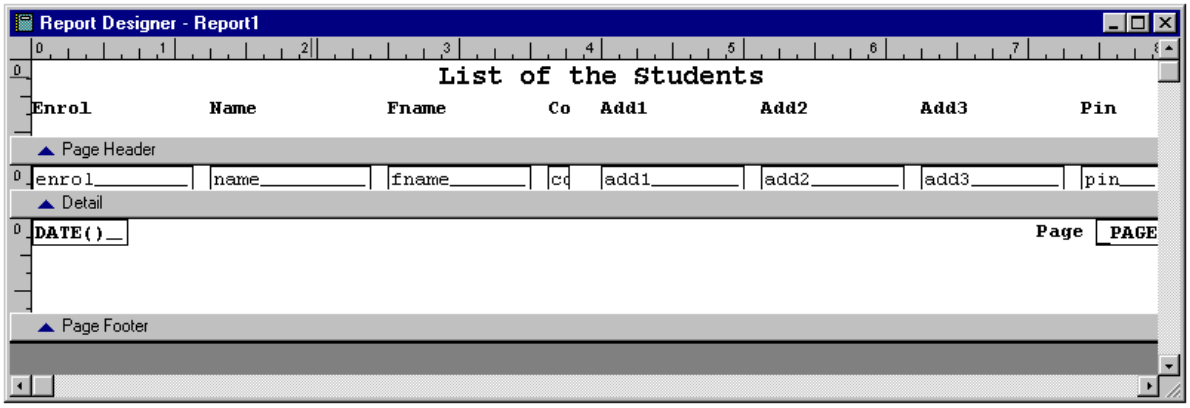


Fig. 9.21: The Report Designer with Added Space

To move field headings in a Report Designer, select the field heading and drag it to the require position. To select several headings at once, hold down the <Shift> Key while clicking on each field and drag them to the required position.

To create a title in the Report Designer, you need to use the **Report Control Toolbar** available in the **View** menu. Click the label tool and move the mouse pointer at the top of the Report Layout Window and type "**List of the Students**". Choosing the **Format** menu, you can **align** the title at the centre and increase the point size and make it bold from the Font Style as per your requirement.

Now, choose **Print Preview** option from the **File** menu or click the **Print Preview tool** to see how the report layout look like as shown in the figure 9.22.

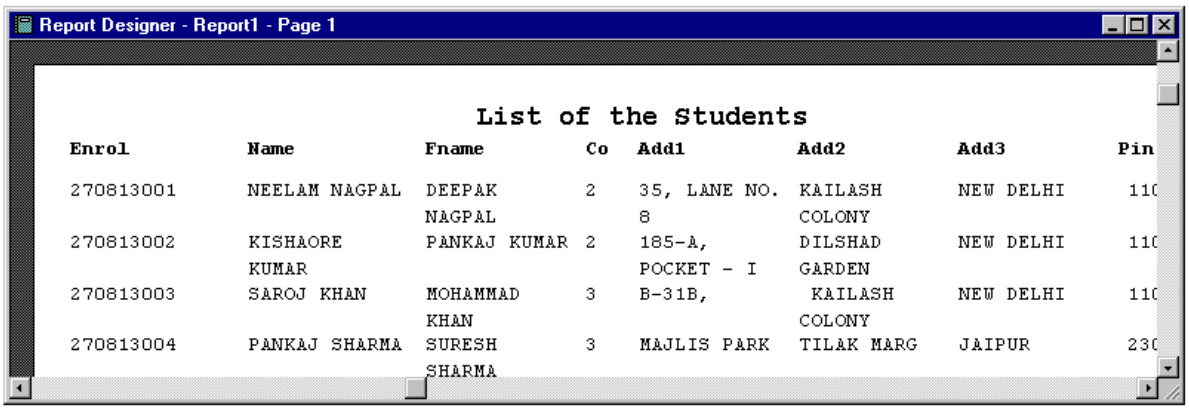


Fig. 9.22: The Print Preview Window with Title

9.6 MODIFYING AN EXISTING REPORT

To modify an existing report form, do the following steps:

- Choose **Open...** option from the **File** menu.
- Select **Report** from the **List Files of Type** popup.
- Choose the report form file you want to open from the **Open** dialog box.

You can also open an existing report through the command window by typing:

MODIFY REPORT <filename> Press <RETURN> Key

The report named "filename.frx appears in the Report Layout window as it was previously saved. Now, you can modify the report as per your requirement in the same way as was discussed during previous section of Creating Report with Report Designer.

9.7 PRINTING A REPORT

Before you print a report, you need to open the report in the **MODIFY REPORT <filename>** mode. After you open an existing report in the Report Layout window, choose **Print** option from the **File** menu or click the **Print** tool to display the **Print Setup** dialog box as shown in the figure 9.23.

Fig. 9.23: The Print Setup Dialog Box

The options available with the Print Setup dialog box are self explanatory, which allows you to select the Printer device, paper orientation and the paper size.

9.8 WHAT YOU HAVE LEARNT

In this lesson you have learned about two types of Report Wizards. The first is general Report Wizard which formats a single table report. The second is one-to-many Report Wizard which uses a parent-child relation between two tables to create a report. You have also learned how to Group the report and to include the total of numeric fields based on a grouping of records in the Report Wizard. You have also seen the limitations of the Report Wizard and beyond these limits how to create a report by using Report Designer.

In the Report Designer you have learned about the basics of creating a custom report, adding report controls to create a professional image and title in your report.

9.9 TERMINAL QUESTIONS

1. Describe two types of reports available with Report Wizard?
2. Listout the various options available while saving a report.
3. Describe the one-to-many report format.
4. What are the steps for adding one more field 'Income' of numeric width 7 in the report crested in figure 9.8.
5. Write steps for typing Title of the report when the report is created using Report Designer.
6. What is the difference between Executive style and Presentation style of reports.

9.10 KEY TO INTEXT QUESTIONS.

1. (a) T (b) F (c) T (d) T
2. on the common field.
3. Print Preview command is used to view the data in advance on the screen before final printing on the paper. The modifications can be done, if the arrangement of data/text is not proper. Print commands takes the final output on the printer and provides a hard copy of it.

10

FUNDAMENTALS OF 'C'

10.1 INTRODUCTION

In this lesson you will be aware with the basic elements used to construct simple C statements. These elements include the C character set, keywords and identifiers, constants, data types, variables, arrays, declarations, expressions and statements. These basic elements are used to construct more comprehensive program components. Some of the basic elements needs very detailed information, however, the purpose of this type of basic elements is to introduce certain basic concepts and to provide some necessary definitions for the topics that follow in next few lessons.

10.2 OBJECTIVES

After going through this lesson you would be in a positions to

- recognize 'C' character set
- recognize keywords and identifiers
- define constants, data types, variables and arrays
- explain the concept of declaration
- describe the expressions and statements

10.3 THE C CHARACTER SET

C uses the uppercase letters A to Z, the lowercase letters a to z, the

digit 0 to 9, and some special characters to form basic program elements (e.g., constants variables, operators, expression) as we discussed above. The special characters are listed below:

!	*	+	\	”	<
#	)	=	;	}	>
^	(	]	,	{	?
&	-	[	:	‘	/
%	_	= =	~	.	(blank)<>

C also uses some combinations of these characters such as \t, \n to represent special conditions such as horizontal tab and newline respectively. These character combinations are called escape sequences. Each escape sequence represents a single character, although it is a combination of two or more characters.

10.4 IDENTIFIERS AND KEYWORDS

Identifiers are names given to various elements of a program, such as variables, functions and arrays. Identifiers consist of digits and letters, in any order but the rule is first character should be a letter. Anyone can use both uppercase and lowercase letters. But uppercase and lowercase letters are not having same meaning e.g. RAM & ram are not same. The underscore character (_) can also be included and is considered to be a letter. Any identifier can start with underscore character, though this is not a good programming practice. Some of the following names are valid identifiers:

She, Z02, computer_02, _wind

Classes, areal, interest_rate, PERIOD

Now, we list out the following names which are not valid identifiers:

2^{nd}, “y”, ship-no, flag error

The first identifier name starting with 2 (i.e. a digit) is not valid, because first character must be a letter, illegal character “ ” is present in the “y” identifier, similarly ship-no, flag error are also invalid identifier name as hyphen & blank character are not allowed in the naming convention of identifiers.

An identifier can be arbitrarily long. Some implementation of C recognize only the first eight characters, though ANSI standard recognizes 31 characters.

Unlike identifiers, keywords are certain reserved words that have, standard, predefined meanings in C. These keywords can be used only for their intended purpose, they cannot be used as programmer-defined identifiers. The standard keywords are listed below:

auto	extern	size of		break	float	static
case	for	struct	const	typedef	if	
switch	char	goto	unsigned	default	long	
continue	int	union	void	do	register	
volatile	double	return	while	else	short	
signed	enum					

It should be kept in mind that keywords are all in lowercase.

INTEXT QUESTIONS

1. What is the purpose of /t and /n character?
2. What is the basic difference between keyword and identifier ?
3. Is student-id is correct for identifier's name or not?

10.5 CONSTANTS

Constant in C refers to fixed values that do not change during the execution of a program. C supports four types of constants. They are integer, character, string & floating-point constants. As the name suggests integer & Floating point constants represent numbers. They are often referred to collectively as numeric-type constants The numeric-type constants must follow the following rules:

- Commas and blank spaces cannot be included within the constant.
- The constant can be preceded by a minus(-) sign if desired.
- The value of a constant cannot exceed specified minimum and maximum bounds. For each type of constant, these bounds will vary from one C compiler to another.

An integer constant refers to a sequence of digits. There are three types of integers, namely, decimal, octal and hexadecimal. A decimal integer constant can consist of any combination of digits taken from

the set 0 through 9, proceded by an optional - or + sign. Various examples of decimal integer constants are:

123, – 345, 0, 5324, +62

An octal integer constant can consist of any combination of digits taken from the set 0 through 7, with a leading 0. Some examples of octal integers are:

0, 01, 0456, 05555,

A hexadecimal integer constant must begin with either 0x or 0x. It can then be followed by any combination of digits taken from the sets 0 through 9 and a through 7 (either upper or lowercase). Note that the letters A through F represent the numbers 10 through 15 respectively. Some examples of hexadecimal integers are:

0x2, 0xgf, 0xbc5, 0x

The largest integer value that can be stored is machine dependent. It is 32767 on 16-bit machines and 2147483647 on 32-bit machines. It is also possible to store larger integer constants on these machines by appending qualifiers such as U, L and UL to the constants. For example

56789U (unsigned integer), 876543217UL (unsigned long integer), 9765467L (Long integer)

Floating point constants is a base-10 number that contains either a decimal point or an exponent (or both). Some of the valid floating point constants are

0. 1. 0.3. 1.555E+8 .121265e18

If an exponent is present, its effect is to shift the location of the decimal point to the right if the exponent is positive or to the left if the exponent is negative. Floating point constants have a much greater range than integer constants. Typically the magnitude of a floating point constant might range from a minimum value of app. 3.4E-38 to a maximum of 3.4E+38. These constants are normally represented as double precision quantities in C. Each floating point constant will typically occupy 2 words (8 bytes) of memory.

Unlike integer constants floating point constants are approximations.

Character constant is a single character, enclosed in apostrophes

(single quotation marks). Some of the valid character constants are 'A', 'y' '4' '$'

Each character constant has its equivalent ASCII value like 'A' has 65, 'y' has 121 '4' has 52 & so on.

Certain nonprinting characters as well as the double quote("), the apostrophe (') , the question mark (?) and the backslash(\) can be expressed in terms of escape sequences. An escape sequence always begins with a backward slash and is followed by one or more special characters. Several character constants, expressed in terms of escape sequences are:

'\n' '\t' '\b' '\\' '\'

The escape sequence \ϕ represents the null character which is used to indicate the end of a string.

String constants consist of any number of consecutive characters enclosed in double quotation marks. Some of the string constants are

"Red", "Mary Marry quite contrary", "2*(J+3)/J" and " "

Sometimes a backslash or a quotation mark must be included as a part of a string constant. These characters must be represented in terms of their escape sequences. The compiler automatically places a null character at the end of every string constant, as the last character within the string. This character is invisible when the string is displayed. One point must be noted here that a character constant 'A' and the corresponding single-character string constant "A" are not equivalent.

10.6 DATA TYPES

C supports different types of data, each of which may be represented differently within the computer's memory. The various basic data types are listed below in tabular form:

Data type	Description	Typical memory requirements
int	integers quantity	2 bytes or 1 word
char	simple character	1 byte
float	floating-point number	1 word (4 bytes)
double	double-precision floating -point number	2 words (8 bytes)

Each data type requires different memory requirements which may vary from one C compiler to another.

C compilers that are written for personal computers represent a word as 4 bytes. Some basic data types can be augmented by using the data type qualifiers short, long, signed & unsigned as discussed above. For example, integer quantities can be defined as short int, long int or unsigned int.

If short int and int both have the same memory requirements (e.g, 2 bytes), then long int will generally have double the requirements. (e.g., 4 bytes) Similarly if int & long int both have the same memory requirements (e.g., 4 bytes) then short int will have half the memory requirements(e.g. 2 bytes).

An unsigned int means all the bits are used to represent the numerical value unlike in the case of ordinary int in which the leftmost bit is reserved for the sign. Thus the size of an unsigned int can be approximately twice as large as an ordinary int. For example if an ordinary int can vary from -32,768 to +32,767 then an unsigned int can vary from 0 to 65,535.

As discussed above that floating point numbers have a decimal point. The C compiler differentiates between floating point numbers & integers because they are stored differently in the computer. Floating point number sometimes are referred to as real numbers. They include all the numbers between the integers. Some of the differences are listed below between floating point numbers and integer.

1. Integer include only whole numbers, but floating point numbers can be either whole or fractional.

2. Integers are always exact, whereas floating point numbers sometimes can lead to loss of mathematical precision.

3. Floating point operations are slower in execution and often occupy more memory than integer operations.

Floating point numbers may also be expressed in scientific notation. For example, the expression 2345.34e6 represents a floating point number in scientific notation. The letter e stands for the word exponent. The exponent is the whole number following the letter e; the part of the number before the letter e is called the mantissa.

The number 2345.34e6 should be interpreted as: 2345.34 times 10 to the 6^{th} power.

The char type is used to represent individual characters. Hence, the char type will generally require only1 byte of memory. With most compilers, a char data type will permit a range of values extending from 0 to 255.

Some compilers permit the qualifier long to be applied to float or to double. e.g long float or long double.

Every identifier that represents a number or a character within a C program must be associated with one of the basic data types before the identifier appears in an executable statement. This is accomplished via a type declaration as described later on.

INTEXT QUESTIONS

4. What are the basic types of constants?
5. Define hexadecimal integer constant briefly.
6. What are two parts of floating point numbers?

10.7 VARIABLES AND ARRAYS

A quantity which may vary during program execution is called a variable. Each variable has a specific storage location in memory where its value is stored. The variable is given a name, the variable name is the “name tag” for the storage location. The value of the variable at any instant during the execution of a program is equal to the number stored in the storage location identified by the name of the variable. The variable name shown in Fig. 10.1 is Sum and its contents is 12.5

Fig 10.1: variable name and its contents

In C, the word identified is used as the general terminology for names given to variables, functions, constants, structures etc.

A variable names may consist of letters, digits and underscore (_) character, subject to the following conditions:

The programmer must follow the rules for naming variables. These are as follows:

1. They must begin with a letter
2. ANSI standard recognizes a length of 31 characters. However, the length should not be normally more than eight characters, since only first eight characters are treated as significant by many compilers.
3. The uppercase & lowercase letters are different e.g., Marry, MARRY , marry are different variables names.
4. Variable name should not be a keyword.
5. No special characters, such as period, comma, semicolon, blanks are permitted in variable names.

The variable name should be chosen to be descriptive of the role that the variable is designed to play. Variable can be assigned any value depending upon their data types. For example,

Sum = 12.5 as shown in the fig 1.1

declares a variable called 'Sum' and assigns a value 12.5 to it. The assignment statement has the following general format

variable = expression.

Notice that the variable name appears to the left side of the equal sign which is called the assignment operator. Only one variable name may appear to the left of the equal sign. A statement such as

C+d=e;

is not acceptable in C language

The array is another kind of variable that is extensively used in C. An array is an identifier that refers to a collection of data items which all share the same name. The data items must all be of the

same type (i.e., all integers, all characters). The individual data items are represented by their corresponding array elements. The individual array elements are distinguished from one another by the value that is assigned to a subscript. For example student is a 10-element array. The first element is referred to as student[0], the second as student[1], and so on. The last element will be student[9].

The number shown in the square brackets is called subscript. For first element the value of subscript is 0, for second element it is 1 & so on. For an n-element array, the subscripts always range from 0 to n-1.

Arrays can be categorized into integer arrays, character arrays, one-dimensional arrays, multidimensional arrays. For now, we will confine our attention to only one type of array: the one dimensional character array, often called a char type array. This type of array is generally used to represent a string. Each array element will represent one character within the string. Thus, the entire array can be thought of as an ordered list of characters.

Note that an n-character string will require an (n+1) element array, because of the null character(\0) automatically placed at the end of the string as stated earlier.

For example string “National” is to be stored in a one dimensional character array called school. Since National contains 8 characters, school will be an 9-element array. Thus, school[0] will present the letter N, school[1] represent[a] & so on. Their subscript values are given below:

0	N
1	a
2	t
3	i
4	o
5	n
6	a
7	l
8	\0

INTEXT QUESTIONS

7. What is a variable?
8. What is the format of the assignment statement?
9. How many variables may appear on the left of the equal sign in an assignment statement ?
10. Define an array.

10.8 DECLARATIONS

A declaration associates a group of variables with a specific data type. In 'C' language all variables must be declared before they appear in executable statements.

A declaration has following format to declare a variable

<data type> <variable name>;

Each array variable must be followed by a pair of square brackets containing a positive integer which specifies the size of the array.

e.g

```
int x,y, z;
float  sq_root, sq_root1;
char student_name[10];
```

Thus x,y,z are declared to be integer type variables, sq_root & sq_root1 are floating point variables & student_name is character type array whose size is 10 elements.

The programmer can write above declarations in the following manner:

```
int x;
int y;
int Z;
float sq_root;
float sq_root1;
char student_name[10];
```

As stated above, integer type variables can be declared to be short integer for smaller integer quantities or long integer for larger integer quantities. They are declared by writing short int and long int or simply short and long respectively. e.g. short int x,y,z and short x,y,z both are same.

Similarly you can write unsigned int or unsigned as the type indicator.

Initial values can be assigned to variables within a type declaration. The declaration must have a format

<data type><variable name....>=constant of appropriate type;

e.g. int x=10;

char school ='N' ;

float sq_root =0;

From these declarations it is clear that x is an integer type variable and has initial value as 10, school is a character type variable and has initial value as 'N', and float sq–root is a floating point variable and has initial value as 0.

A character-type array can also be initialized within a declaration. For this the square brackets of an array name must be empty. It has following format:

<data type> <array name[ ]> = "<string>";

e.g.char school[] = "National";

This statement means 'school' will be an 9-element character array. The first 8 elements will represent the 8 characters within the word National & the 9^{th} element will represent the null character(\0) which is automatically added at the end of the string.

This can also be written as

Char school[9]="National";

The size of the array must be specified correctly for this type of statements because if the size is small say char school[8] ="National" the characters at the end of the string (in this case, the null character) will be lost and if the size is too large say

char school[18] ="National"

the extra array elements may be assigned spaces, or they may be filled with meaningless characters.

INTEXT QUESTIONS

11. How do you declare a integer of short data type?

12. Can you initialize a char-type array within a declaration ?

10.9 EXPRESSIONS AND STATEMENTS

An expression represents a single data item, such as a number or a character. The programmers can use operators in the expressions in C like other programming languages. Expressions can also represent logical conditions that are either true or false.e.g.

```
x+y
y=z
x=y+z
```

The first expression involves use of addition operator(+), the second expression involve use of assignment operator(=) and third expression involves use of both addition operator & assignment operator.

Statement causes the computer to carry out some action. There are three types of statements in 'C' language. They are expression, compound & control statements.

An expression statement consists of an expression followed by a semicolon. For example, following two expression statements cause the value of the expression on the right of the equal sign to be assigned to the variable on the left.

```
x=5;
x=y+z;
```

A compound statement consists of several individual statements enclosed within a pair of braces ({ and}).

Unlike an expression statement, a compound statement does not end with a semicolon.

Control statements are used to create special program features, such as logical tests, loops and branches.

Symbolic constants are names for a sequence of characters. The characters may represent a numeric constant, a characters constant or a string constant. So, symbolic constant allows a name to appear in place of a numeric or character, or string constant. At the time of compiling each occurrence of a symbolic constant get replaced by its corresponding character sequence.

These are usually defined at the beginning of a program. For example:

#define student class

Where students represents a symbolic name, and class represents the sequence of characters associated with the symbolic name. It should be kept in mind that class does not end with a semicolon, since a symbolic constant definition is not a like C statement. Further if you include semicolon with class, this semicolon would be treated as a part of the numeric, character or string constant that is substituted for the symbolic name.

INTEXT QUESTIONS

13. What is an expression?
14. What is symbolic constant?

10.10 WHAT YOU HAVE LEANRT

In this lesson you have learnt about 'C' character set, keywords and identifier. Now you are well aware about constants and its different types such as int, char, float, signed, unsigned, long etc. There are some rules to declare variable names and arrays that you have learnt in this lesson. The concept of declarations, expression and statements have also been discussed for the benefit of the learners.

10.11 TERMINAL QUESTIONS

1. State the rules for naming identifiers.
2. What are the unsigned integer constants?
3. Explain an escape sequence?
4. Define a subscript? What range of values is permitted for the subscript of a one dimensional n-element array?

10.12 KEY TO INTEXT QUESTIONS

1. /t for tab character and /n for newline character.
2. Identifier are names given to various elements of a program, whereas keywords are reserved words that have standard, pre-defined meaning in C.
3. No, because 'hyphen' is not allowed.
4. Integer, character, string & floating,
5. A hexadecimal integer constant must begin with either x or X. It can then be followed by any combination of digits through 9 and A to F (lowercase or uppercase).
6. Mantissa & exponent part.
7. It is a symbolic name for a memory location in which value is stored.
8. Variable=expression;
9. Only one
10. An array is an identifier refers to a collection of data items which all have the same name .
11. Short int<variable names or short <variable name>
12. Yes
13. Expression represents a single data item, such as a number or a character.
14. Symbolic constants are names for a sequence of characters.

11

OPERATORS AND EXPRESSIONS IN 'C'

11.1 INTRODUCTION

Operators form expressions by joining individual constants, variables, array elements as discussed in previous lesson. C includes a large number of operators which fall into different categories. In this lesson we will see how arithmetic operators, unary operators, relational and logical operators, assignment operators and the conditional operators are used to form expressions.

The data items on which operators act upon are called operands. Some operators require two operands while others require only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variable as operand.

11.2 OBJECTIVES

After going through this lesson you will be able to

- recognize arithmetic operators
- explain unary operators
- define relational, logical, assignment & conditional operators
- explain library functions

11.3 ARITHMETIC OPERATORS

There are five main arithmetic operators in 'C'. They are '+' for additions, '-' for subtraction, '*' for multiplication, '/' for division and '%' for remainder after integer division. This '%' operator is also known as modulus operator. For exponentiation there is no specific operator in 'C' instead there is one library function known as pow to carry out exponentiation.

Operands can be integer quantities, floating-point quantities or characters. The modulus operator requires that both operands be integers & the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero, though the operands need not be integers. Division of one integer quantity by another is referred to as integer division. With this division the decimal portion of the quotient will be dropped. If division operation is carried out with two floating- point numbers, or with one floating-point number. & one integer, the result will be a floating-point quotient.

If we take two variables say x and y and their values are 20 and 10 respectively, and apply operators like addition, subtraction, division, multiplication and modulus on them then their resulted values will be as follows:

x+y	30
x-y	10
x*y	200
x/y	2
x %y	0

If one or both operands represent negative values, then the addition, subtraction, multiplication and division operations will result in values whose signs are determined by the usual rules of algebra.

The interpretation of remainder operation is unclear when one of the operands is negative.

Operands that differ in type may undergo type conversion before the expression takes on its final value. The final result has highest precision possible, consistent with data types of the operands. The following rules apply when neither operand is unsigned.

- If one operand is a floating point type and the other is a char or

an int, the char/int will be converted to the floating point type of the other operand and the result will be expressed as such. So, an operation between an int and a double will result in double.

- If both operands are floating-point types with different precisions, the lower-precision operand will be converted to the precision of the other operand and the result will be expressed in this higher precision.

  Float & double → double

  Float & long double → long double

  Double & long double → long double

- If neither operand is a floating point type or a long int, then both operands will be converted to int & the result will be int.

- If neither operand is a floating point type but one is a long int, the other will be converted to long int & the result will be long int.

The value of an expression can be converted to a different data type if desired. The 'C' programmer also has the choice of explicitly specifying how the values are to be converted in a mixed mode expression. This feature is known in 'C' as **coercion** and may be accomplished using what is called the **cast operator** . The name of data type to which the conversion is to be made is enclosed in parentheses and placed directly to the left of the value to be converted; the word "cast" never is actually used. The example of type casting is as follows:

```
int a=17;
float b;
b=(float)a+ 11.0;
```

The cast operator converts the value of int a to its equivalent float representation before the addition of 11.0 is carried out. The precedence of the cast operator is the same as that of the unary minus operator. The cast operator can be used on a constant or expression as well as on a variable e.g.

```
(int) 7.179
(double) (5*3/8)
   (float)(a+4)
```

In the second example, the cast is performed only after the entire expression within the parentheses is evaluated.

Remember due to type casting, the data type associated with the expression itself is not changed, but it is the value of the expression that undergoes type conversion wherever the cast appears. This is particularly relevant when the expression consists of only a single variable.

The operator within C are grouped hierarchically according to their order of evaluation known as **precedence**. Obviously operations with a higher precedence are carried out before operations having a lower precedence. The natural order can be altered by making use of parentheses.

Arithmetic operators *,/ and % are under one precedence group and +,- are under another precedence group. The operators *, / and % have higher precedence than + and -. In other words, multiplication, division and remainder operations will be carried out before addition and subtraction.

Another important point to consider is the order in which consecutive operations within the same precedence group are carried out. This is known as **associativity**. Within each of the precedence groups described above, the associativity is left to right. In other sense, consecutive addition and subtraction operations are carried out from left to right, as are consecutive multiplication, division and remainder operations.

As stated earlier, the natural precedence of operations can be altered through the use of parentheses, thus allowing the arithmetic operations within an expression to be carried out in any desired order. In fact, parentheses can be nested, one pair within another. In such cases the innermost operations are carried out first, then the next innermost operations, and so on. It is perfectly acceptable to use redundant parentheses if they help to make the expression more meaningful. Whenever parentheses are used, however, be sure that there are as many left parentheses as right.

Let us understand this with the help of one example, say there are 3 variables a,b, and c having values 5,10 and 15 respectively. The different operations on these three variables and their result is as follows:

a+b/c=5

b*c-a=145

a*b/c= 3

(a+c)*b/a=40

In the first expression, division is done first because division has a higher precedence than addition.

In the second expression, the multiplication is done before the subtraction.

In the third expression, the calculation proceeds from left to right, but truncation takes place in the division operation.

Finally in the last case (a+c)*b/a, the contents of the parentheses are evaluated first, then it is multiplied & finally divided. In this case no truncation takes place.

Sometimes it is a good idea to use parentheses to clarify an expression, even though the parentheses may not be required, but use of overly complex expressions should be avoided. Such expressions are difficult to read, and they are often written incorrectly because of unbalanced parentheses.

INTEXT QUESTIONS

1. What is the effect of dividing an integer by a real quantity? Also, what is this effect called?
2. How would you use the cast operator to convert(7*9/11) to double?

11.4 UNARY OPERATORS

'C' includes a class of operators that act upon a single operand to produce a new value. Such operators are known as unary operators. Unary operators usually precedes their single operands, though some unary operators are written after their operands.

The most common unary operator is unary minus, where a minus sign precedes a numerical constant, a variable or an expression.

e.g.

-5,-10, -20(numbers)

x=-y(variable)

Of all the arithmetic operators, the unary minus has the highest precedence level. Thus in an expression such as

```
y=x+z* -b;
```

evaluation commences with the unary minus, which negates the value of b. Then z is multiplied by –b, and finally the addition takes place. The result is stored in the variable y. So, the use of parentheses to separate the two adjacent operators and avoid possible confusion.

```
Y=x+z*(-b);
```

In C, all numeric constants are positive. Thus a negative number is actually an expression, consisting of the unary minus operator, followed by a positive numeric constant.

It is to be noted here that the unary minus operation is distinctly different from the arithmetic operator which denotes subtraction(-). The subtraction operator requires two separate operands.

Two other commonly used unary operators are increment operator, ++, & the decrement operator - -. The increment operator causes its operand to increased by one, whereas the decrement operator causes its operand to be decreased by one. The operand used with each of these operators must be a single variable. For example, x is an integer variable that has been assigned a value of 10. The expression ++ x, which is equivalent to writing x= x+1, causes the value of x to be creased to 11. Similarly the expression --x, which is equivalent to x=x-1, causes the original value of x to be decreased to 9.

The increment and decrement operators can each be utilized in two different ways, depending on whether the operator is written before or after the operand. If the operator precedes the operand, then the value of operand will be altered before it is used for its intended purpose within the program. If, however the operator follows the operand then the value of the operand will be changed after it is used.

For example, if the value of x is initially 10, then if we increase it by saying ++x then the current value of x will be 11.

Similarly for decrement operator the current value of x will be 9 if we say -- x.

Another unary operator is the **sizeof** operator . This operator re-

turns the size of its operand, in bytes. This operator always precedes its operand. The operand may be an expression, or it may be a cast.

e.g sizeof (x);
sizeof (y);

If x is of integer type variable and y is of floating point variable then the result in bytes is 2 for integer type and 4 for floating point variable.

Consider an array school[]= “National”

Then, sizeof (school) statement will give the result as 8.

A cast is also considered to be unary operators. In general terms, a reference to the cast operator is written as (type). Thus, the unary operators we have encountered so far are -,++,- -, sizeof & (type).

Unary operators have a higher precedence than arithmetic operators. Hence, if a unary minus operator acts upon an arithmetic expression that contains one or more arithmetic operators, the unary minus operation will be carried out first. The associativity of the unary operators is right to left.

INTEXT QUESTIONS

3. What are unary operators?
4. How many operands are associated with a unary operators?
5. Describe the difference between increment and decrement operators.
6. What is the associativity of unary operators?
7. What is sizeof operator?

11.5 RELATIONAL, LOGICAL, ASSIGNMENT, CONDITIONAL OPERATORS

Relational operators are symbols that are used to test the relationship between two variables, or between a variable and a constant. The test for equality, is made by means of two adjacent equal signs with no space separating them. ‘C’ has six relational operators as follows:

> greater than

< les than

!= not equal to

>= greater than or equal to

>= less than or equal to

These operators all fall within the same precedence group, which is lower than the unary and arithmetic operators. The associativity of these operators is left-to-right.

The equality operators ==and != fall into a separate precedence group, beneath the relational operators. Their associativity is also from left-to-right.

These relational operator are used to form logical expression represeating condition thet are either true or false. The resulting expression will be of type integer, since true is represented by the integer value and false is represented by the value0.

Let us understand operations of these relational operaters with the help of an example: x=2, y=3, z=4.

x<y	true	1
(x+y) >=z	true	1
(y+z)>(x+7)	false	0
z!=4	false	0
y ==3	true	1

Here, true or false represents logical interpretation and they have values 1 and 0 respectively.

There are two logical operators in C language, they are **and or.** They are represented by && and !! respectively.

These operators are refered to as **logical and, logical or,** respectively. The result of a **logical and** operation will be true only if both operands are true, whereas the result of a **logical or** operation will be true if either operand is true or if both operands are true.

The logical operators act upon operands that are themselves logical expressions. Let us understand it, with the help of a following example:

Suppose x=7, y=5.5 , z= 'w'

Logical expressions using these variables are as follows:

Expression	Interpretation	Value
(x>=6) &&(z= ='w')	true	1
(x>=6)¦¦ (y = =119)	true	1
(x<=6) && (z=='w')	false	0

Each of the logical operators falls into its own precedence group. Logical **and** has a higher precedence than logical **or**. Both precedence groups are lower than the group containing the equality operators. The associativity is left to right.

'C' also includes the unary operator !, that negates the value of a logical expression. This is known as logical negation or logical **NOT** operator. The associativity of negation operator is right to left.

Precedence of operators in decreasing order is as follows:

1. -, ++ - - Operators ! size of (type)
2. * / %
3. + -
4. < <= > > =
5. == !=
6. &&
7. ¦¦

If you have trouble remembering all these rules of precedence you can always resort to parentheses in order to express your order explicitly.

INTEXT QUESTIONS

8. What is the difference between = and == ?
9. What logical operator negates the truth value of the expression to its immediate right?
10. What integer value is equivalent to false?
11. What is the associativity of logical NOT?

There are several different assignment operators in C. All of them are used to form assignment expression, which assign the value of an expression to an identifier. The most commonly used assignment operator is =. The assignment expressions that make use of this operator are written in the form:

identifier=expression

where identifier generally represents a variable and expression represents a constant, a variable or a more complex expression.

Assignment operator = and the equality operator == are distinctly different. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two expressions have the same value. These two operators cannot be used in place of one another.

If the two operands in an assignment expression are of different data types, then the value of the expression on the right will automatically be converted to the type of the identifier on the left. The entire assignment expression will then be of this same data type. For example

A floating point value may be truncated if assigned to an integer identifier, a double-precision value may be rounded if assigned to a floating-point identifier, an integer quantity may be altered if assigned to a shorter integer identifier or to a character identifier.

Multiple assignments of the form

Identifier 1= identifier 2 = - - - -= expression are allowed in 'C'.

In such situations, the assignments are carried out from right to left.

Let us understand this with the help of following example

x=y=10 (x and y are integer variables)

will cause the integer value 10 to be assigned to both x and y. Similarly, x=y=10.5

will cause the integer value 10 to be assigned to both x and y, truncation occurs when the floating point value 10.5 is assigned to the integer variable y.

'C' also contains the five additional assignment operators +=, -=, *=, /= & %=.

expression1+= expression2

is equal to

expression1= expression1 + expression2

Similarly expression1 - = expression2 is equivalent to expression1 = expression1 - expression 2

Assignment operators have a lower precedence than any of the other operators that have been discussed earlier. The decreasing order of precedence is given below, the associativity of assignment operators is right to left.

- ++ - - ! sizeof (type)
* / %
+ -
< <= > >=
= = !=
&&
| |
= + = - = * = /= %=

For example x* = -3 *(y+z)/4

is equivalent to x= x*(-3 *(y+z)/4)

In this example first (y+z) will be evaluated, then multiplication will take place. Then division, and finally multiplication will take place.

INTEXT QUESTIONS

12. What are the 6 assignment operators discussed above ?

13. How can multiple assignments be written in C?

There is one conditional operator(?:) in 'C' language. An expression that makes use of the conditional operator is called a conditional expression.

A conditional expression is written in the form

Expression 1 ? expression 2 : expression 3

When evaluating a conditional expression, expression 1 is evaluated first. If expression 1 is true, then expression 2 is evaluated and

this becomes the value of the conditional expression. If expression 1 is false, then expression 3 is evaluated and this becomes the value of the conditional expression

For example (i< 1) ? 0:200

i is integer variable here.

The expression (i<1) is evaluated first, if it is true the entire conditional expression takes on the value 0. Otherwise, the entire conditional expression takes on the value 200.

Conditional expression frequently appear on the righthand side of a simple assignment statement. The resulting value of the conditional expression is assigned to the identifier on the left.

The conditional operator has its own precedence, just above the assignment operator. The associativity is right-to-left.

Decreasing order of precedence is as follows:

1 - ++ - - ! sizeof (type)

2 * / %

3 + - ?:

4 < <= > >=

5 = = !=

6 &&

7 ! !
! !

8 ?:

9 = += -= *= /= %=

11.6 LIBRARY FUCNTIONS

Library functions carry out various commonly used operations or calculations. Some functions return a data item to their access point, others indicate whether a condition is true or false by returning 1 or 0 respectively, still others carry out specific operations on data items but do not return anything. Features which tend to be computer dependent are generally written as library functions.

Functionally similar library functions are usually grouped together as object programs in separate library files. These library files are supplied as a part of each C compiler.

A library function is accessed simply by writing the function name, followed by a list of arguments that represent information being passed to the function. The arguments must be enclosed in parentheses and separated by commas. The arguments can be constants, variable names or more complex expressions. The parentheses must be present, even if there are no arguments, A function that returns a data item can appear anywhere within an expression in place of a constant or an identifier. A function that carries out operations on data items but does not return anything can be accessed simply by writing the function name, since this type of function reference constitutes an expression statement. In order to use a library function it may be necessary to include certain specific information within the main portion of the program. This information is generally stored in special files supplied with the compiler. Thus, the required information can be obtained simply by accessing these special files. This is accomplished with the preprocessor statement.

```
# include <filename>
```

The list of some commonly used library functions are:

abs(i)	to determine absolute value of i.
exp(i)	raise e to the power i.
log(d)	determine natural logarithm of d.
pow(d1,d2)	returns d1 raised to the d2 power
putchar(c)	send a character to the standard output device
sqrt(d)	return the square root of d.

You can see rest of the library functions available in C in Help topic of 'C'.

INTEXT QUESTIONS

14. Define the associativity of conditional operators.
15. Are the library functions actually a part of the 'C' language?

11.7 WHAT YOU HAVE LEARNT

In this lesson you have learnt about 'C' arithmatic, unary, relational, logical, assignment and conditional operators. The concept of library functions here also been discussed.

11.8 TERMINAL QUESTIONS

1. What is an operator? Describe different types of operators?
2. What is an operand ? What is the relationship between operators & operands ?
3. Summarize the rules that apply to expression whose operands are of different type?
4. When should parentheses be included within the expression?
5. How can the conditional operator be combined with the assignment operator to form an "if-else" type statement?

11.9 KEY TO INTEXT QUESTIONS

1. The integer is temporarily converted to its floating point equivalent before the calculation is performed. This automatic operation is known as implicit conversion.
2. (double) (7*9/11)
3. Operators that act upon a single operand to produce a new value are called unary operator.
4. Only one
5. Increment means increase the value, or add the specified value to the variable's value. Decrement means decrease or less the value of the variable.
6. Right to left
7. It will display the total size of variable or data-type.
8. = is the assignment operator and == is the equality operator.
9. not or !

10. zero
11. right to left
12. =, +=, - =, * =, /= & %=
13. identifier 1 = identifier2 = - - - -= expressions.
14. Right to left
15. No

12

INPUT AND OUTPUT OF DATA

12.1 INTRODUCTION

In 'C' language input and output of data is done by a collection of library functions like getchar, putchar, scanf, printf, gets and puts. These functions permit the transfer of information between the computer and the standard input/output devices. The library function getchar and putchar as the name suggests, allow single characters to be transferred into and out of the computer, scanf and printf permit the transfer of single characters, numerical values and strings, gets and puts facilitate the input and output of strings. An input/output function can be accessed from anywhere within a program simply by writing the function name, followed by a list of parameters enclosed in parentheses. Some input/output functions do not require parameters, but the empty parentheses must appear. 'C' include a collection of header file that provide necessary information in support of the various library functions. The header file stdio.h contains the information about input/output library functions. In this lesson we will discuss some input/output functions in detail.

12.2 OBJECTIVES

After going through this lesson you would be able to

- explain getchar function
- define putchar function

- explain scanf function
- explain printf function
- describe gets & puts function
- use interactive programming

12.3 GETCHAR FUNCTION

getchar function reads a single character from standard input. It takes no parameters and its returned value is the input character.

In general, a reference to the getchar function is written as character variable = getchar();

For example char c;

c= getchar () ;

The second line causes a single character to be entered from the standard input device and then assigned to c.

If an end-of-file condition is encountered when reading a character with the getchar function, the value of the symbolic constant EOF will automatically be returned.

This function can also be used to read multicharacter strings, by reading one character at a time within a multipass loop.

12.4 PUTCHAR FUNCTION

The standard C function that prints or displays a single character by sending it to standard output is called putchar. This function takes one argument, which is the character to be sent. It also returns this character as its result. If an error occurs, an error value is returned. Therefore, if the returned value of putchar is used, it should be declared as a function returning an int.

For example

```
putchar ('N');
putchar ('a');
putchar ('t');
putchar ('i');
putchar ('o');
putchar ('n');
putchar ('a');
putchar ('l');
```

When putchar is used, however, each character must be output separately. The parameter to the function calls in the given statements are character constants, represented between apostrophes as usual. Of course, the arguments could be character variables instead.

Two functions that require FILE pointers are getc and putc. These functions are similar to getchar and putchar, except that they can operate on files other than the standard input and output. The getc function is called with one argument, which is a FILE pointer representing the file from which the input is to taken. The expression getc(stdin) is similar to

getchar()

and the expression putc(c, stdout) is same as putchar(c).

INTEXT QUESTIONS

1. What is the difference between input and output ?
2. What are the restrictions related to the putchar function ?
3. Name the counterpart to the putchar function.

12.5 SCANF FUNCTION

Input data can be entered into the computer from a standard input device by means of C library function scanf. This function can be used to enter any combination of numerical values, characters single character and strings. The function returns the number of data items that have been entered successfully.

In general terms, the scanf function is written as

scanf (string, parameter 1, parameter 2..., parameter n);

Where string= string containing certain required formatting information, and Parameter 1, parameter 2.. = parameters that represent the individual input data item.

The control string or string comprises individual groups of characters, with one character group for each input data item. Each character group must start with percent sign (%). In the string, multiple character groups can be contiguous, or separated by white space characters. The conversion character that is used with % sign are

many in number and all have different meaning corresponding to type of data item that is to be input from keyboard.

Some of the conversion characters are listed below:-

Conversion character	Meaning
c	type of data item is single character
d	type of data item is decimal integer
e	type of data item is floating-point value
f	type of data item is floating-point value
h	type of data item is short-integer
i	type of data item is decimal, hexadecimalor octal integer
o	type of data item is octal integer
s	type of data item is string
u	type of data item is unsigned decimal integer
[. . .]	type of data item is string which may include whitespace characters.

The parameters are written as variables or arrays, whose types match the corresponding characters groups in the control string. Each variable name must be preceded by an ampersand (&).

For example: Suppose there are 3 variables char name[10], int roll_no, float marks, then the scanf statement for these 3 variables will be

scanf("%s %d %f," name, &roll_no, &marks);

This statement contains three character groups. %s represents that first parameter is string i.e name. The second character group %d, represents that the parameter is decimal integer value, and third character group %f represents that the parameter is floating point value.

It is to be noted here that roll_no and marks that are only variables not arrays and are preceded by ampersand sign unlike name which is a character array. The order of data items depends upon the requirement of user, but the order of character group should match with the order of data items.

If two or more data items are entered, they must be separated by white space characters. Data items may continue onto two or more lines, since the newline character is considered to be a whitespace character.

It is already stated that s-type conversion character applies to a string terminated by a whitespace character. Therefore, a string that includes whitespace characters cannot be entered in this manner. To do so the s-type conversion character within the control string is replaced by a sequence of characters enclosed in square brackets designated as [...]. Whitespace characters may be included within the brackets, thus accommodating strings that contain such characters.

When the program is executed, successive characters will continue to be read from the standard input device as long as each input character matches one of the characters enclosed within the brackets. The order of the characters within the square brackets need not correspond to the order of the characters being entered. Input characters may be repeated. The string will terminate, however once an input character is encountered that does not match any of the characters within the brackets. A null character \0 will then automatically be added to the end of the string.

Another way to do this is to precede the characters within the square brackets by a circumflex (^). This causes the subsequent characters within the brackets to be interpreted in the opposite manner. Thus, When the program is executed, successive characters will continue to be read from the standard input device as long as each input character does not match one of the characters enclosed within the brackets. If the characters within the brackets are simply the circumflex followed by a **new line** characters, then the string entered from the standard input device can contain any ASCII characters except the newline character. For example:

```
char school [40];

scanf  ("%[^\n]", school);
```

Through this statement any string of undetermined length (but not more than 40 characters) will be entered from the standard input device and assigned to school.

"National Open School" is assigned to array **'school'**. If you want to limit or to restrict the width of the data item you can define it with the help of an unsigned integer indicating the field width is placed

within the control string, between the % and the conversion character.

The data item may be composed of fewer characters than the specified field width. But you cannot exceed the number of characters in the actual data item than specified field width. Any character that extend beyond the specified field width will not be read, incorrectly interpreted as the components of next data item. Such leftover characters may be

For example:

```
int x,y,z;
scanf("%3d % 3d %3d", & x, &y, &z);
```

If the data input from the keyboard is 1,2,3 then the result is

x=1, y=2 , z=3 but suppose if the data input is 123 , 456, 789 then it will result in

x=123, y=456, z=789. If the input is 1234 5678 9 then x=123 y=4 z=567.

The remaining two digits (8 and 9) would be ignored, unless they were read by a subsequent scanf statement.

If the control string contains multiple character groups without interspersed whitespace characters, then whitespace characters within the input data will be interpreted as a data item. To skip this and read the next nonwhitespace character, the conversion group %s should be used.

INTEXT QUESTIONS

4. What function enables a user to input information while the program is in execution ?

5. If numeric or single-character information is being entered by means of scanf function, what symbol must precede the corresponding variable name ?

12.6 PRINTF FUNCTION

The printf function is used to print out a message, either on screen or paper(The letter "f" in printf could stand for either "formatted" or

"function"). It is equivalent to the WRITE statement in Pascal, only more powerful. It is similar to the input function scanf except that its purpose is to display data rather than to enter data into the computer. So, printf function moves data from the computer's memory to the standard output device, whereas the scanf function enters data from the standard input device and stores it in the computer's memory. The general form is:

printf(string, parameter1, parameter2,……, parameter n)

where string refers to a string that contains formatting information, and parameter 1, parameter2… parameter n are arguments that represents the individual output data items. The parameters can be written as constants, single variable or array names or more complex expressions.

Unlike scanf function, the parameters in a printf function do not represent memory addresses and therefore they are not preceded by ampersand (&) sign. The control string or string is composed of individual groups of characters, with one character group for each output data item. Each character group must start with a percent sign like in scanf function followed by a conversion character indicating the type of the corresponding data item. Multiple character groups can be contiguous, or they can be separated by other characters, including whitespace characters. Some of the conversion characters are given below:

c type of data item is displayed as a single character

d type of data item is displayed signed decimal integer

e type of data item is displayed as a floating-point value with an exponent

f type of data item is displayed as a floating-point value without an exponent

o type of data item is displayed as an octal integer without a leading zero.

s type of data item is displayed as a string

u type of data item is displayed as an unsigned decimal integer

x type of data item is displayed as a hexadecimal integer, without the leading ϕx.

Let us consider an example of printf.

Suppose there are 3 variables, char name[10], int roll_no, float marks,

then the printf statement to display all these 3 variables is printf("%s %d %f"' name, roll_no, marks);

Within the printf function, the control string or string is "%s %d %f". It contains 3 characters groups. The first character group %s indicates that the first parameter represents a string. The second character group, %d indicates that the second parameter represents a decimal integer value and the third character group %f indicates that the third parameter represents a floating point value.

It is to be noted here that the parameters are not preceded by ampersands as in scanf. The printf function interprets s-type conversion differently than the scanf function. In the printf function,s-type conversion is used to output a string terminated by the null character (\0). Whitespace characters may be included within the string.

For example

```
char school[40];
scanf("%[^\n]" ,school);
printf("%s",school);
```

If the string entered through keyboard is "National Open School" then it will print it as it is on the screen through printf statement.

A concept of minimum field width in a printf statement, can be specified by preceding the conversion character by an unsigned integer. If the number of characters in the corresponding data item is less than the specified field width, then the leading blanks will be added in front of data item to fill the specified field. But if the number of characters in the corresponding data item is greater than the specified field width then the extra space will be allocated to display the whole data item as contrary to scanf function in which there is a concept of maximum field width as stated earlier.

It is also possible to specify the maximum number of decimal places for a floating point value or the maximum number of characters for a string. This is known as **precision.** This is an unsigned integer that is always preceded by a decimal point. If a minimum field width is specified in addition to the precision then the precision specification follows the field width specification.

For example

```
float a= 456.789;
printf("%7f % 7.3f %7.1 f", a,a,a);
```

The result is 456.789000 456.789 456.7

Minimum field width and precision specification can be applied to character data as well as numerical data. When applied to a string, the minimum field width is interpreted in the same manner as with a numerical quantity. The field width specification will not prevent the entire string from being displayed. The precision specification will determine the maximum number of characters that can be displayed. If the precision specification is less than the total number of characters in the string, the excess rightmost characters will not be displayed. This will occur even if the minimum field width is larger than the entire string, resulting in the addition of leading blanks to the truncated string.

For example

```
char school[9];

printf("%7s %12s % 12.3s  %.3s, school , school,
school,school);
```

Suppose the string is 'National' then the output is

```
Nationa ....  National..... Nat Nat
```

The first string is shown as input even this consists of 8 characters and the field width specification is 7 characters. So it over rides the minimum field width specification. The second string is padded with 4 leading blanks to make 12 as field width specification. Third string is padded with 12 leading blanks and only 3 characters are shown because of 3 character precision specification and the last string also displays only 3 characters but there is no leading blank spaces because of no minimum field width specification. So second string in this case is right-justified in the output.

In addition to the field width, the precision and the conversion character, each character group within the control string can include a flag, which affects the appearance of the output. The flag must be placed immediately after the percent sign. Some of the flags are given below with their common usage.

1. – data item is left-justified within the field

```
int i=346
printf(":%ϕ-6d",i);
output :346
```

2. + a sign (either + or-) will precede each signed int i=346
float f=-3.5
numerical data item; without this flag, only negative data should be there–3. 5e+000: items are preceded by a sign.
printf(":%-+6d%-10.1e":,i,j);
output is:+346 some space

3. Zero causes leading zeros to appear instead of leading blanks, applies only to data items that are right justified.Within a field whose minimum size is larger than the data item.

4. ' ' a blank space will precede each +ve signed numerical data item; flag is override by the + flag if both are present.

5. # causes octal and hexadecimal data items to be preceded by Zero() and x respectively.

In case of strings there is also a usage of flags. Let us see it with the help of examples:

Suppose string is " National Open"

1	printf(":%s:\n", "National Open");	:National Open:	Normal Printing
2	printf(":%9s:\n", "National Open");	:National Open:	Min.field width
3	printf(":%17s:\n", "National Open");	....National Open:	printed in a field wider than the string.
4	printf(":%-17s:\n", "National Open");	:National Open.... ;	left justified printed in a field of 17.
5	printf(":%17.6s:\n", "National Open");	:..........Nation	truncated to 6 characters
6	printf(":%.6s:\n", "National Open");	: Nation:	: truncated to 6 characters
7	printf(":%-17.6s:\n", "National Open");	: Nation.....:	truncated to 6 characters left justified in a field of 12.

Unrecognised characters within the control string or string will be displayed just as they appear. This feature allows us to include label and messages with the output data items if we want.

There is some variation in the features supported by the printf function in different versions of C.

INTEXT QUESTIONS

6. What conversion specification are used to print out the following integer, a floating point number in decimal & scientific notation & a single character ?

7. What is the minimum field width specifics, & where is it located ?

8. When the conversion specification is %, what is its default field width ?

12.7 GETS AND PUTS FUNCTION

'C' contains a number of other library functions that permit some form of data transfer into or out of the computer. Gets and puts functions facilitate the transfer of strings between the computer and the standard input/output devices. Each of these functions accepts a single argument or parameter. The parameter must be a data item that represents a string. The string may include whitespace characters. In the case of gets, the string will be entered from the keyboard and will terminate with a newline characater.

The gets and puts functions are alternative use for scanf and printf for reading and displaying strings.

For example:

```
char school[40];
gets(school);
puts(school);
```

These lines uses the gets and puts to transfer the line of text into and out of the computer. When this program is executed, it will give the same result as that with scanf and printf function for input and output of given variable or array.

12.8 INTERACTIVE PROGRAMMING

Interactive programming means to create an interactive dialog between user and computer. This is some sort of question and answer. This can be created by alternate use of scanf and printf functions.

This type of interactive programming is useful in the case of useful reports or data entry.

12.9 WHAT YOU HAVE LEARNT

In this lesson you have learnt about getchar and putchar function in detail. You are now aware of scanf, printf functions which are used for input of data from the keyboard and display of data to the screen in a formatted way. You have also learnt the concept of gets and puts which are equivalent to scanf and printf except for formatting.

You can do interactive programming for your project work very easily after going through this lesson.

12.10 TERMINAL QUESTIONS

1. What are the commonly used input/output functions in C? How are they accessed ?
2. What is the standard input/output header file called in most versions of C?
3. Explain the purpose of the getchar function.
4. Define the purpose of scanf function.
5. Summarize the meaning of the more commonly used conversion characters within the control string of a scanf function.
6. What is the purpose of the printf function? How is it used within a C program?
7. Explain the precision of an output data item.
8. Summarize the purpose of the flags commonly used within printf.
9. How are unrecognized characters within the control strings of a printf function interpreted ?
10. Explain the use of the gets and puts function ?

12.11 KEY TO INTEXT QUESTIONS

1. Information that enters the computer from the outside is called input, and information produced by the computer and sent to the outside is known as output.

2 It can outputs only one character at a time, and only to the standard output.

3 getchar function

4. scanf function

5. The ampersand (&) symbol.

6. Integer %d, floating point in decimal %f, with scientific notation %e, a single character % c.

7. It specifies the smallest field in which a value is to be printed. It is used in a conversion specification, and if present, it is placed immediately following the % sign.

8. The size of the number being printed (including a minus sign, if the number is negative.)

13

A COMPLETE C PROGRAM

13.1 INTRODUCTION

It is necessary to do some planning before starting actual programming work. This planning will include the logic of the actual program. After logical approach of the program, the syntactic details of the language can be considered. This approach is often referred to as **"top-down"** programming. Top-down program organization is normally carried out by developing an informal outline, consisting of phrases or sentences that are part of English and part of C language. In the initial stages of program development, the amount of C is minimal, consisting only of various elements that define major program components, such as function headings, function references etc. Additional details is then provided by descriptive english material inserted between these elements, often in the form of program comments. The resulting outline is usually referred to as **pseudocode.** Another method sometimes used is the **"bottom-up"** approach. This approach involves the detailed development of these program modules early in the overall planning process.

13.2 OBJECTIVES

After going through this lesson you will be able to

- write a C program
- compile and execute the program
- detect and correct errors

13.3 WRITE C PROGRAM

Once the planning of a program has been formulated, the detailed working development of C program can be considered. This includes translation of each step of the program outline into one or more equivalent C instructions.

There is more to do to write a complete C program than simply arranging the individual declarations and statements in the right order. There are some additional features to be added to enhance the program's readability and its resulting output. There must be logical sequencing of the statements, the use of indentation, the use of comments and the generation of clearly labeled output statements and use of comments for enhancement of program readability.

The use of indentation is closely related to the sequencing of groups of statements within a group. It describes the subordinate nature of individual statements within a group. Comments should always be included within a C program. They can identify certain key items within the program and provide other useful information about the program. The other important characteristic of a well-written program is its ability to generate clear, legible output. When executing an interactive program, the user may not know how to enter the required input data. For example, the user may not know what data items are required, when the data items should be entered, or the order in which they should be entered. Thus a well written interactive program should generate prompts at appropriate times during the program execution in order to provide this information.

Let us consider an example of interactive program:

```
# include < stdio.h>
/ * simple calculation of average marks* /
main ( )
{
 float m1, m2, avg ;
/ * read input data * /
print ("please enter a value for marks m1 :\n");
scanf ("%f", & m1);
printf ("please enter a value for marks m2;");
scanf("%f", & m2);
avg = m1+m2 / 2 ;
```

```
/ * write output * /
printf ( "/n the final value (avg) is: %f ", avg);
}
```

The first line of this program is # include <stdio.h>, <stdio.h> is the file which has definitions of standard input output files, this file must be included in every C program as every program obviously needs printf and scanf statements. The second line, main ( ) function which is followed by curly braces. Now comes the definition and declaration of variables and functions as needed by the programmer. After that there is body of the program which is actually the syntactical conversion of logic of the program. The curly braces of the main function should be end with closed curly braces.

Once the program has been written it must be entered into the computer before it can be compiled and executed. This is usually accomplished by one of two possible methods, the most common being the use of an editor. Most computer systems include an editor that is used to create and alter text file. Some editors are line-oriented, while others are character-oriented. The commands within a line-oriented editor permit various editing operations, such as insert, delete, copy to be carried out on specified lines within the text files. Character-oriented editors are generally used with interactive terminals. They are also referred to as full-screen editors. Any editor can be used to enter a C program and its accompanying data files, though some C compilers are associated with operating systems that include their own editors.

Regardless of particular editor being used, the procedure for creating a new program is to enter the program into a text file line-by-line. The text file is then assigned a file name and stored either in the computer's memory or on an auxiliary storage device. Usually a suffix, such as c is attached to the file name, thus identifying the file as a C program. Such suffixes are called **extensions**. The program will remain in this form until it is ready to be processed by C compiler.

13.4 COMPILING AND EXECUTING THE PROGRAM

Once a complete program has been correctly entered into the computer, it can be compiled and executed. The compilation is usually accomplished automatically in response to a single command say *compile sample*, where sample refers to the name of the file containing the program (sample.c). It may also be necessary to link the

compiled object program (sample.obj)with one or more library routines in order to execute the program. This is done by the command *link sample*. In the UNIX operating system the compile and link steps can both be carried out using the single instruction *cc sample*.c, resulting in an object program called sample.out. The successful compilation and linking of a C program will result in an executable object program that can then be executed in response to an appropriate system command, such *as execute*. In UNIX, the execution can be initiated simply by typing the name of the object program i.e. sample.out

Let us consider the previous example of program average of 2 marks, suppose the name of program is average. After writing the program correctly, compilation must take place. This is done by compiling average. The computer will respond by compiling the program, resulting in a non executable object program called average.obj. Typing errors, syntactic errors, and so on can easily be corrected by reentering the editor and making the required changes. If the compilation is carried out successfully, the next step is to link the program with the program library routines. This is accomplished with the command:-

link average

The result of the linking process will be the creation of an executable objet program, called average.exe. If the compilation has not been successful, it would not have been possible to proceed with the link step. The programmer must then find the source of error, re-enter the editor, and correct the original source program so that it can again be compiled. To exceute the final object program, simply type the program name e.g., average.out.

After this the following interactive dialog will be generated:

Please enter a value for marks m1: 50

Please enter a value for marks m2: 50

The final value (avg) is : 50

C interpreters translate a source program into object code on a line by line basis and then execute the newly generated object code directly. Thus the generation and execution of object code occurs simultaneously and there is no link step. Successful compilation, linking and execution of a C program often requires several attempts because of the presence of errors that are generally present in a new program.

Programming errors often remain undetected until an attempt is made to compile or execute the program. Some particular common errors of this type are improperly declared variables, a reference to an undeclared variable or incorrect punctuation. Such errors are referred to as syntactic or grammatical errors. Most C compilers will generate diagnostic messages when syntactic errors have been detected during the compilation process. Let us consider an example, which contains several syntactic errors.

```
include < stdio.h>
main
{
 float m1, m2, avg ;
/ * read input data * /
printf (“please enter a value for marks m1 :\n”);
scanf (“%f”, m1);
printf (“please enter a value for marks m2;);
scanf(“%f”, & m2);
avg = m1+m2 / 2 ;
/ * write output * /
printf ( (“\n the final value (avg) is: %f ”, avg);
```

After compiling, the errors are as follows:

1. The include statements does not begin with a # sign, main does not include a pair of parentheses.
2. The first scanf statement does not have an ampersand (&) preceding the argument.
3. The 2nd printf statement does not have a closing quotation mark.
4. The program does not end with a closing brace }

These errors are of general type, in the form of C compiler the messages are as follows:

Average.C(2): errors 54: expected ‘(‘ to follow ‘include’

Average.C(7): errors 61: syntax error: identifier ‘scanf’

Average.C(8): errors 61: syntax error: identifier ‘printf’

Average.C(13): fatal error: unexpected EOF

The error message consists of the file name, followed by the line

number where an error has been detected. This information is followed by a numbered reference to an error type, which is then followed by a very brief message. Another common type of error is the **execution error.** Execution errors occur during program execution after a successful compilation e.g. some common execution errors are the generation of an excessively large numerical quantity, division by zero, or an attempt to compute the logarithm or the square root of a negative number. Diagnostic messages will often be generated in this type of situation, making it easy to identify and correct the errors. These diagnostics are sometimes called execution diagnostics, to distinguish them from the compilation diagnostics.

Interpreters operate differently than compilers. Interpreters can detect both compilation-type errors and execution-type errors line by line, approximately at the same time. When an error is detected during the interpretation process, the interpreter will stop and issue an error message, indicating which line was being processed when the error was detected. But interpreters are generally much slower than compilers while executing a program. Interpreters are therefore less suitable for running debugged programs.

INTEXT QUESTIONS

1. Define “top-down” programming?
2. Why are some statement indented within a C program?
3. What is the extension of C program file?
4. What is a syntactic error?

13.5 DETECTION AND CORRECTION OF ERRORS

Syntactic errors and execution errors usually result in the generation of error messages when compiling or executing a program. Error of this type are usually quite easy to find and correct. There are some logical errors that can be very difficult to detect. Since the output resulting from a logically incorrect program may appear to be error free. Logical errors are often hard to find, so in order to find and correct errors of this type is known as **logical debugging.**

To detect errors test a new program with data that will give a known answer. If the correct results are not obtained then the program obviously contains errors. Even if the correct results are obtained,

however you cannot be sure that the program is error free, since some errors cause incorrect result only under certain circumstances. Therefore a new program should receive thorough testing before it is considered to be debugged.

Once it has been established that a program contains a logical error, some ingenuity may be required to find the error. Error detection should always begin with a thorough review of each logical group of statements within the program. If the error cannot be found, it sometimes helps to set the program aside for a while. If an error cannot be located simply by inspection, the program should be modified to print out certain intermediate results and then be rerun. This technique is referred to as **tracing**. The source of error will often become evident once these intermediate calculations have been carefully examined. The greater the amount of intermediate output, the more likely the chances of pointing the source of errors. Sometimes an error simply cannot be located.

Some C compilers include a debugger, which is a special program that facilitates the detection of errors in C programs. In particular a debugger allows the execution of a source program to be suspended at designated places, called **break points,** revealing the values assigned to the program variables and array elements at the time execution stops. Some debuggers also allow a program to execute continuously until some specified error condition has occurred. By examining the values assigned to the variables at the break points, it is easier to determine when and where an error originates.

INTEXT QUESTIONS

5. What is logical debugging?

6. Define tracing.

7. What is a debugger?

13.6 WHAT YOU HAVE LEARNT

In this lesson you learnt how to prepare a complete C program, what files should be included in a C program. You are now aware of the fact that how to compile and execute the C program after writing it through some editor. At last logical and syntactical errors and technique to detect and correct them are also discussed for the benefit of the learners.

13.7 TERMINAL QUESTIONS

1. Explain the advantages of "top-down" programming.
2. Define "bottom-up" programming.
3. What is the difference between a line-oriented editor and a character oriented editor?
4. What is the difference between compilation and execution of a C program?
5. Describe the concept of linking.
6. What are diagnostic message?

13.8 KEY TO INTEXT QUESTIONS

1. When the overall program strategy has been clearly established, then the syntactic details of the language can be considered. Such an approach is often referred to as "top-down" programming.
2. The use of indentation is closely related to the sequencing of groups of statements within a program. Indentation explains the subordinate nature of individual statements within a group.
3. 'C'
4. If some variables are improperly declared, a reference to an undeclared variable or incorrect punctuation then such errors are referred to as syntactical errors.
5. To find and correct errors of logical type is known as logical debugging.
6. If an error cannot be located simply by inspection, the program should be modified to print out certain intermediate results and then be rerun. This is known as tracing.
7. Debugger is a special program that facilitates the detection of errors in other C programs. It allows the execution of a source program to be suspended at designated places, called break point.

14

MAKING DECISIONS IN "C"

14.1 INTRODUCTION

So far we have seen that in C programs the instructions are executed in the same order in which they appear in the program. Each instruction is executed once and once only. Programs do not include any logical control structures. Most programs, however, require that a group of instructions be executed repeatedly, until some logical condition has been satisfied. This is known as **looping**. Most of the programs require that a logical test be carried out at some particular point within the program. An action will then be carried out whose exact nature depends upon the outcome of the logical test. This is known as **conditional execution**.

14.2 OBJECTIVES

After going through this lesson you would be able to

- define *'while'* statement, *for* statement and *nested loops*
- explain switch statement and goto statement
- define comma operator

14.3 WHILE STATEMENT

The while statement is used to carry out looping operations. The general form of the statement is

while (expression) statement

The loop operates in the following fashion:

The value of the test expression enclosed in parentheses is evaluated. If the result is true, then the program statement (the body of the loop) is executed. The statement may be a compound statement. Then the test expression, which may be just as complex as any of those found in if statement is evaluated again. If it is again true, the statement is executed once more. This process continues until the test expression becomes false. At that point, the loop is terminated immediately, and program execution continues with the statement (if any) following the while loop. If there are no more statements, the program terminates.

Let us consider an example, which prints the five lines. Each line is numbered by printing out the value of x each time around the loop.

```
/ * A test program for while loop */
include<stdio.h>
main()
{
      int x=1;
while(x<6)
{
print(“This is line number %d of test program\n”,x);
x++;
}
}
```

The variable x is assigned a value of 1. The while loop displays the line number of test program by incrementing the value of x by one each time until the value of x is less than 6. Thus the loop will be repeated 5 times, resulting in 5 consecutive lines of output. Thus, when the program is run, the following output will be generated.

This is line number 1 of test program.

This is line number 2 of test program.

This is line number 3 of test program.

This is line number 4 of test program.

This is line number 5 of test program.

This program can be written more concisely as

```
#include <stdio.h>
main()
        {
        int x=1;
        while (x<6)
printf("This is line number %d of test program\n";x++);
        }
```

When executed, this program will generate the same output as the first program.

All variables used in the test expression of the while statement must be initialized at some point before the while loop is reached. In addition, the body of the loop must do something to change the value of the variable used in the expression being tested. Otherwise the condition would remain true and the loop would never terminate. This situation, known as an infinite loop, is illustrated next.

```
while(x<6)
printf("Something is wrong in this loop\n");
```

The loop is infinite because the value of x is never change. If the test expression starts out being true, it remains true forever. In the previous program, the value of x determines how many times the loop executes. Therefore, the second statement in the body of the loop (x++;) serves to increment x by 1. Since x starts out with a value of 1 and is incremented by 1, at some point it must become equal to 6. Then the condition in the while loop becomes false and the loop is terminated.

If x were initialized to a value of 6, the condition in the while loop becomes false to begin with , and the body of the loop would not be executed at all. It is clear, then, that the while statement employs a loop pre-test. The loop condition is tested before each iteration, and therefore before the loop is entered at all. If the condition initially fails, the loop is skipped entirely.

If the test expression involves two variables and joined with 'and' or 'OR' operator

```
        #include <stdio.h>
```

```
main()
      {
      int x=1;
int a=4;
while (x<6 && a>3)
{
printf(" This is line number %d of test program \n",x);
x++;
   }
     }
```

The loop executes only while x is less than 6 and a is greater than 3. If at least one of these conditions becomes false, the loop terminates. If at least one of the conditions is false when the loop is first encountered, the loop is skipped entirely.

The value of a is not changed in the loop, but the value of x is incremented in a way that will make the loop terminate eventually. If the value of a is less than or equal to 3, then the initial test fails and the while loop is skipped entirely.

There is another way to use while loop, instead of incrementing the value of variables inside the loop, you can also decrement it. For example, if you want to print the line number of test program in reverse order then test expression changes and the program is as follows:

```
#include <stdio.h>
main()
{
            int x = 0
            printf("This is line number of test program in reverse
            order");
            while(x>=0)
      {
            printf("%d\n",x);
            x - - ;
      }
}
```

There is an option for test expression in while loop, instead of giving some constant value to test variable you can even given it some variable also. The value of this variable can be input from the user interactively e.g. if the user wants to calculate the average of numbers, but 'how many numbers?' this number can be input from the user itself and then this variable can be used in test expression.

Let us consider an example.

```
#include <stdio.h>
main()
{
int number, n=1;
float x, average, sum=/0;
printf("How many  number ?");
scanf("%d", & number);
while(n<=number)
{
scanf("%f", &x);
sum=sum+x;
++n;
}
average=sum/number;
printf("\n The average is %f  \n", average );
}
```

The output of this program is as follows:

How may number? 6

1.

2.

3.

4.

5.

6.

The average is 3.500000

do……while loop

The do… while loop differs from its counter part, the while loop in that it makes what is called a loop **post-test**. That is the condition is not tested until the body of the loop has been executed once. In the while loop, by contrast, the test is made on entry to the loop rather than at the end. The effect is that even if the condition is false when the do-while loop is first encountered, the body of the loop is executed at least once. If the condition is false after the first iteration, the loop terminates. If the first iteration has made the condition true, however the loop continues.

The general form of the do….while loop is as follows:

do

Statement;

while (test expression);

The fact that the while clause is located after the statement reflects the fact that the test is made after the statement is executed.

If the body of the loop is a single statement, it must be terminated with a semicolon. For example:

do

a=a+10;

while (a<b);

This semicolon marks the end of the inner statement only not of the entire loop construct. In every situation that requires a loop, either one of these two loops can be used. Let us consider an example:

```
#include <stdio.h>
main()
{
int numbers, n=1;
float x,average ,sum=0;
printf("How many number ?");
scanf("%d", &numbers);
do
{
scanf("%f; &x);
sum=sum+x;
```

```
++n;
}
while (n<=numbers);
average=sum/numbers;
printf("\n The average is %f\n", average);
}
```

The output of this program is same as that with while loop.

INTEXT QUESTIONS

1. What happens if the condition in a while loop is initially false?
2. What is the minimum number of times the body of a do... while loop is executed?
3. What is the essential difference between a while and a do.... While loop?
4. Define syntax for do.... While loop?

14.4 FOR STATEMENT

The for statement is the most commonly used looping statement in 'C'. This statement includes an expression that specifies an initial value for an index, another expression that determines whether or not the loop is continued and the third expression that allows the index to be modified at the end of each pass.

The general form of the for statement is

for (expression1; expression2; expression3) statement

Expression1 is the initialization expression, usually an assignment, which is performed once before the loop actually begins execution. Expression 2 is the test expression, exactly like the one used in the while loop, which is evaluated before each iteration of the loop and which, determines whether the loop should continue or be terminated. Finally, expression 3 is the modifier statement, which changes the value of the variable used in the test. This expression is executed at the end of each iteration, after the body of the loop is executed. Statement is the body of the loop, which may as usual be compound. The three loops expressions are separated by two semicolons. No semicolon should be placed after expression 3.

The first expression of the for loop can be omitted if the variable is initialized outside the loop. If one or more expressions are omitted from the for loop, the two semicolons still must appear, even if they are not preceded or followed by anything. Let us understand the concept of for loop with the help of an example:

```
#include <stdio.h>
main()
{
int x;
for(x=1; x<=10; x ++)
printf("This is line number %d of test program\n",x);
}
```

The output for this program is as follows:

```
This is line number 1 of test program
This is line number 2 of test program
This is line number 3 of test program
This is line number 4 of test program
This is line number 5 of test program
This is line number 6 of test program
This is line number 7 of test program
This is line number 8 of test program
This is line number 9 of test program
This is line number 10 of test program
```

In order to print the tenth line, the test expression must use the <= relational operators. If < alone were used, the loop would print only nine lines.

Let us consider one more program to print first five even numbers through use of for loop.

```
#include <stdio.h>
main()
{
    int i;
    for(i=2; i<=10; i=i+2)
```

```
printf("%d\n", i);
}
```

The output of this program is as follows:

2

4

6

8

10

In the for loop, if the second expression is omitted, however it will be assumed to have a permanent value of 1 (true) thus the loop will continue indefinitely unless it is terminated by some other means, such as a break or a return statement.

INTEXT QUESTIONS

5. Which is the better loop to use, the for loop or the while loop?
6. What is a special advantage of the for loop?
7. What separates the three expressions of a for statement?
8. How does the for loop operate?

14.5 NESTED LOOPS

Loops can be nested or embedded one within another. The inner and outer loops need not be generated by the same type of control structure. It is essential, however that one loop be completely embedded within the other there can be no overlap. Each loop must be controlled by a different index. Let us consider an example of nested for loops:

```
#include <stdio.h>
main()
{
int i,j,n,sum;
for(i=1;i<=5; i++)
{
printf("\nEnter a positive number:");
scanf("%d",&n);
```

```
            sum=0;
            for(j=i; j<n; j++)
            {
            sum=sum+j;
            printf("\n The sum of the integers from 1 to %d
is:%d\n",          n,sum);
            }
}

  }
```

The output is as follows: -

Enter a positive number: 5
The sum of the integers from 1 to 5 is: 15

Enter a positive number: 10
The sum of the integers from 1 to 10 is: 55

Enter a positive number: 15
The sum of the integers from 1 to 15 is: 120

Enter a positive number: 20
The sum of the integers from 1 to 20 is: 210

Enter a positive number: 5
The sum of the integers from 1 to 25 is: 325

In this program firstly, outer for loop will be executed, if the expression (test) is true, sum variable is initialized to zero every time and then inner loop will be executed. Inner loop will be executed till the test expression of inner loop satisfies. Then if the expression is false, the pointer again goes to outer loop and then it will reexecute. For nested loops any other loop structures could also have been selected.

14.6 IF...ELSE STATEMENT

C allows decisions to be made by evaluating a given expression as true or false. Depending upon the result of the decision, program execution proceeds in one direction or another. This is carried out in 'C' by 'if' statement. The simplest form of the if statement is as follows:

if (expression)

Statement;

'if' statement will execute if the given expression is true. If the user wants to add more than one statement then there must be pair of curly braces after 'if'.

```
if  (expression)
{
statement;
statement;
}
Let us consider an example of if statement.
#include <stdio.h>
main()
{
char grade;
printf("Enter a character value for grade:");
scanf("%c", & grade);
if(grade = = 'A')
printf("The grade is excellent \n");
printf("Thanks for using this program\n");
}
```

The output is as follows:

Enter a character value for grade: A

The grade is excellent

thanks for using this program

If we re-run the program with different value for grade than the output is

Enter a character value for grade: C

Thanks for using this program

In the second case, the value for grade is 'C' other than 'A' which is in test expression of if statement, thus the first printf statement will not execute.

In the previous program only the general message is printed if the

user selects a grade other than 'A'. If the user wants to display other message when enter a grade other than 'A', you have to add the else clause of if statement. The general form of if..else is as follows.

```
if (statement)
Statement 1;
else
Statement 1;
```

Both if and else clause are terminated by semicolons. Let us consider an example of if…else statement.

```
#include <stdio.h>
main()
{
        char grade;
        printf("Enter a character value for grade:");
        scanf("%c", &grade);
        if(grade= ='A')
        printf("grade is excellent \n");
        else
        printf("grade is other than excellent\n");
}
```

The user can use compound statements both in if and else statements. The first printf is executed if and only if grade is equal to 'A', if grade is not equal to 'A', the first printf is ignored, and the second printf, the one following the word else, is executed.

A clause of the if statement may itself contain another if statement, this construct known as nesting of if statements. Let us consider an example nested if..else statement:

```
#include <stdio.h>
main()
{
int year;
printf("Enter a year to check for leap year");
scanf("%d", & year);
if (year %4= = O),
```

```
if(year %100 != 0)
printf("%d is a leap year:\n", year);
else
if(year %400 = =)
printf("%d is a leap year \n",year);
else
printf("%d is not a leap year \n", year);
else
printf ("%d is not a leap year\n", year);
}
```

It is very important to be sure which else clause goes with which if clause. The rule is that each else matches the nearest if preceding it which has not already been matched by an else. Addition of braces prevents any association between the if statement within the braces and the else clause outside them. Even where braces are not necessary, they may still be used to promote clarity.

INTEXT QUESTIONS

9. In an if statement, if two separate statements are to be executed when the comparison is true, what must be done with them?

10. What is the function of the else clause in an if statement?

14.7 SWITCH STATEMENT

The switch statement causes a particular group of statements to be chosen from several available groups. The selection is based upon the current value of on expression that is included within the switch statement. The general form of the switch statement is

switch (expression) statement

Where expression results in an integer value. Expression may also be of type char, since individual characters have equivalent integer values. The embedded statement is generally a compound statement that specifies alternate courses of action. Each alternative is expressed as a group of one or more individual statements within the overall embedded statement. For each alternative, the first statement within the group must be preceded by one or more case labels. The case labels identify the different groups of statements and

distinguish them from one another. The case labels must therefore be unique within a given switch statement.

Thus, the switch statement is in effect an extension of the familiar if...else statement. Rather than permitting maximum of only two branches, the switch statement permits virtually any number of branches.

In general terms, each group of statements is written as

```
case expression1;
case expression2;
      :
      :
case expression m:
Statement 1
Statement 2
      :
Statement n
```

Where expression 1, expression 2.... Expression n represent constant, integer valued expressions. Each individual statement following the case labels may be either simple or complex. When the switch statement is executed, the expression is evaluated and control is transferred directly to the group of statements whose case-label value matches the value of the expression. If none of the case-label value matches the value of the expression, then none of the groups within the switch statement will be selected. In this case control is transferred directly to the statement that follows the switch statement.

Let us consider an example of switch statement:

```
      #include <stdio.h>
      main()
{
      char vowel;
      printf("Enter a character");
      scanf("%c",&vowel);
      switch(vowel)
{
      case'a'
```

```
        case'A':
                printf("vowel");
                break;
        case 'e' :
        case 'E':
                printf("vowel");
                break;
        case 'i' :
        case 'I' :  printf("vowel"); break;
        case' o' :
        case 'O' :
                printf("vowel");
                break;
        case 'u' :
        case 'U' :
                printf("vowel");
}
     }
```

One of the labeled groups of statements within the switch statement may be labeled default. This group will be selected if none of the case labels matches the value of the expression. The default group may appear anywhere within the switch statement. If none of the case labels matches the value of the expression and default group is not present, then the switch statement will take no action.

```
#include <stdio.h>
main()
{
        char vowel;
        printf("Enter a character");
        scanf("%c", & vowel);
        switch(vowel)
{
        case 'a':
        case 'A':
              printf("vowel");
```

```
            break;
        case'e':
        case 'E':
            printf("vowel");
            break;
        case 'i' :
        case 'I':
            printf("vowel");
            break
        case'o':
        case 'O':
            printf("vowel");
            break;
        case'u'
        'u':
            printf("vowel");
            break;
        default:
            printf("Not a vowel");
        }
}
```

The default label can be placed anywhere within the body of the switch statement; it need not be the last label. In fact, the labels of a switch statement can appear in any order, depending on the logic of the program. It is desirable for several different values of the switch variable to cause execution of the same set of statements. This can be accomplished by including several labels in succession with no intervening statements.

14.8 THE BREAK STATEMENT

The break statement is used to force fully terminate loops or to exit from a switch. It can be used within a while, a do-while, for or a switch statement. The format is simple as

```
break;
```

without any embedded expression or statements. The break statement causes a transfer of control out of the entire switch statement, to the first statement following the switch statement.

If a break statement is included in a while, in do while or in for loop, then control will immediately be transferred out of the loop when the break statement is encountered. Thus provides a convenient way to terminate the loop if an error or other irregular condition is detected. Let us consider a program segment of break statement in while loop.

```
#include <stdio.h>
main()
{
        int x, sum=O;
        printf("Enter any number");
        scanf("%d" &x);
        while(x<=50)
        {
                if (x<zero)
        {
        printf("error value because of negative value");
        break;
        }
                scanf("%d", &x);
        }
```

The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered, instead the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop. It can also be included within a while, do while or a for statement as like break statement. It is also written simply as

continue;

without any embedded statement or expression.

14.9 THE COMMA OPERATOR

Comma operator is used primarily in conjunction with the for statement. This operator permits two different expressions to appear in situation where only one expression would ordinarily be used.

For example:

for (expression 1a, expression 1b; expression 2; expression 3) statement.

where expression 1a and expression 1b are the two expressions, separated by comma operator, where only one expression would normally appear.

Let us consider an example:

```
#include<stdio.h>
main()
{
        int i,j,n;
      printf("Enter number for a table");
        scanf("%d", &n);
        printf("\n");
        for(i=0, j=n; i<n; i++, j– – )
        printf("%3d+%3d= %3d\n", i,j,n);
}
```

The output is as follows:

Enter number for a table 20

```
 0+20=20
 1+19=20
 2+18=20
 3+17=20
 4+16=20
 5+15=20
 6+14=20
 7+13=20
 8+12=20
 7+13=20
 8+12=20
 9+11=20
```

10+10=20

11+9=20

12+8=20

13+7=20

14+6=20

15+5=20

16+4=20

17+3=20

18+2=20

19+1=20

20+0=20

The comma operator has the lowest precedence. Thus, the comma operator falls within its own unique precedence group, beneath the precedence group containing the various assignment operators. Its associatively is left-to-right

14.10 THE GOTO STATEMENT

The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program. It is written as

goto label;

Where label is an identifier used to label the target statement to which control will be transferred. Control may be transferred to any other statement within the program. The target statement must be labeled, and the label must be followed by a colon. Thus the target statement will appear as

Label : statement

No two statements cannot have the same label

Goto statements has many advantages like branching around statements or groups of statements under certain conditions, jumping to the end of a loop under certain conditions, thus bypassing the remainder of the loop during the current pass, jumping completely out of a loop under certain conditions, thus terminating the execution of a loop.

14.11 WHAT YOU HAVE LEARNT

In this lesson, you have learnt about different types of loops like for, do..while, while. You are now familiar with if-else statement and switch statement. You are now also familiar with comma operators, break and continue statement and goto statement. You can very well use all of them in a 'C' program to make a program interactive and user friendly.

14.12 TERMINAL QUESTIONS

1. What is meant by looping? Describe two different forms of looping.
2. What is the purpose of while statement?
3. How is the execution of a while loop terminated?
4. How many times will a for loop be executed, what is the purpose of the index in a for statement?
5. What is the purpose of the switch statement?
6. Compare the use of the switch statement with the use of nested if else statement?
7. What is the use of break and continue?
8. What is the purpose of goto statement. Explain their usage in the program.

14.13 KEY TO INTEXT QUESTIONS

1. The while loop is skipped over
2. One
3. A while loop performs its test before the body of the loop is executed, whereas a do..loop makes the test after the body is executed.
4. do

   Statement;

   while (test expression);
5. It depends on the nature of the problem to be solved, and upon the preference of the programmer

6. The initialization, testing and modifier expressions of the loop all are specified within a single set of parentheses.
7. Semicolons
8. The first expression is executed once. Then the second expression is tested, if it is true, the body of the loop is executed. Then the third expression is executed. Again, second expression is evaluated, till the second expression is true, the body of the loop is executed, followed by the third expression.
9. They must be combined into a compound statement, using the curly braces { and }.
10. If present, it is followed by a statement that is executed only if the condition being tested proves to be false.

15

FUNCTIONS IN 'C'

15.1 INTRODUCTION

In the earlier lessons we have already seen that C supports the use of library functions, which are used to carry out a number of commonly used operations or calculations. C also allows programmers to define their own functions for carrying out various individual tasks. In this lesson we will cover the creation and utilization of such user-defined functions.

15.2 OBJECTIVES

After going through this lesson you will be able to

- explain of function
- describe access to function
- define parameters data types specification
- explain function prototype and recursion
- define storage classes – automatic, external, static variables

15.3 MODULAR APPROACH

The use of user-defined functions allows a large program to be broken

down into a number of smaller, self-contained components, each of which has some unique, identifiable purpose. Thus a C program can be modularized through the intelligent use of such functions. There are several advantages to this modular approach to program development. For example many programs require a particular group of instructions to be accessed repeatedly from several different places within a program. The repeated instruction can be placed within a single function, which can then be accessed whenever it is needed. Moreover, a different set of data can be transferred to the function each time it is accessed. Thus, the use of a function avoids the need for redundant (repeating) programming of the same instructions. The decomposition of a program into individual program modules is generally considered to be an important part of good programming.

15.4 DEFINING A FUNCTION

The question arises what is a function? So, function is a self-contained program segment that carries out some specific well-defined task. Every C program consists of one or more functions. The most important function is main. Program execution will always begin by carrying out the instruction in main. The definitions of functions may appear in any order in a program file because they are independent of one another. A function can be executed from anywhere within a program. Once the function has been executed, control will be returned to the point from which the function was accessed. Functions contains special identifiers called parameters or arguments through which information is passed to the function and from functions information is returned via the return statement. It is not necessary that every function must return information, there are some functions also which do not return any information for example the system defined function printf.

Before using any function it must be defined in the program. Function definition has three principal components: the first line, the parameter declarations and the body of the functions.

The first line of a function definition contains the data type of the information return by the function, followed by function name, and a set of arguments or parameters, separated by commas and enclosed in parentheses. The set of arguments may be skipped over. The data type can be omitted if the function returns an integer or a character. An empty pair of parentheses must follow the function name if the function definition does not include any argument or parameters.

The general term of first line of functions can be written as:

data-type function-name (formal argument 1, formal argument 2...formal argument n)

The formal arguments allow information to be transferred from the calling portion of the program to the function. They are also known as parameters or formal parameters. These formal arguments are called actual parameters when they are used in function reference. The names of actual parameters and formal parameters may be either same or different but their data type should be same. All formal arguments must be declared after the definition of function. The remaining portion of the function definition is a compound statement that defines the action to be taken by the function. This compound statement is sometimes referred to as the body of the function. This compound statement can contain expression statements, other compound statements, control statements etc. Information is returned from the function to the calling portion of the program via the return statement. The return statement also causes control to be returned to the point from which the function was accessed.

In general terms, the return statement is written as

```
return expression;
```

The value of the expression is returned to the calling portion of the program. The return statement can be written without the expression. Without the expression, return statement simply causes control to revert back to the calling portion of the program without any information transfer. The point to be noted here is that only one expression can be included in the return statement. Thus, a function can return only one value to the calling portion of the program via return. But a function definition can include multiple return statements, each containing a different expression. Functions that include multiple branches often require multiple returns.

It is not necessary to include a return statement altogether in a program. If a function reaches the end of the block without encountering a return statement, control simply reverts back to the calling portion of the program without returning any information.

Let us consider an example of function without returning any information.

```
#include <stdio.h>
        main()
{
        int x,y;
        maxi(int, int); /*function declaration*/
        printf("Enter two integer values");
        scanf("%d %d"' &x,&y);
        maxi(x,y); /*call to function*/
}
        maxi(x,y)       /*function definition*/
        int x,y;
{
        int z;
        z=(x>=y)?x:y;
        print("\n\n Maximum value %d",z);
        return;
}
```

Body of the function maxi

This 'maxi' function do not return any value to the calling program, it simply returns the control to the calling programs, so if it is even not present, then also program will work efficiently.

Most C compilers permit the keyword void to appear as a type specifies when defining a function that does not return anything. So the function definition will look like this if void is add to it

void maxi (int, int);

INTEXT QUESTIONS

1. Distinguish between a user-defined and one supplied, functions in the C library.
2. In what sense does the user-defined function feature of 'C' extends its repertoire?
3. How many values can a function return?

15.5 ACCESSEMENT OF A FUNCTION

A function can be accessed by specifying its name, followed by a list

of parameters or arguments enclosed in parentheses and separated by commas. If the function call does not require any arguments an empty pair of parentheses must follow the function's name. The function call may appear by itself or it may be one of the operands within a more complex expression. The parameters in the body of the functions are called **actual arguments** as stated earlier, they may be expressed as constants, single variables or more complex expressions.

Let us consider another example of function.

```
#include <stdio.h>
        main()
{
        int a,b,c;
        printf("Enter two numbers");
        scanf("%d%d", &a,&b);
        c=sum_v(a,b,);
        printf("\n The sum of two variables is %d\n,",c);
}
        sum_v(a,b)
        int a,b
{
        int d;
        d=a+b;
        return d;
}
```

This program returns the sum of two variables a and b to the calling program from where sum_v is executing. The sum is present in the variable c through the 'return d' statement. There may be several different calls to the same function from various places within a program. The actual parameters may differ from one function call to another. Within each function call, the actual arguments must correspond to the formal arguments in the function definition, i.e. the number of actual arguments must be same as the number of formal arguments and each actual argument must be of the same data type as its corresponding formal argument.

Let us consider an example.

```
#include <stdio.h>
        main()
{
        int a,b,c,d;
        printf("\n Enter value of a=");
        scanf("%d", &a);
        printf("\n Enter value of b=");
        scanf("%d",&b);
        printf("\n Enter value of c=");
        scanf("%d", &c);
        d=maxi(a,b);
        printf("\n maximum =%d", maxi(c,d));
}
        maxi(x,y);
        int x,y
{
        int z;
        z=(x>=y)? x:y;
        return z;
}
```

The function maxi is accessed from two different places in main. In the first call actual arguments are a, b and in the second call c, d are the actual arguments.

If a function returns a non-integer quantity and the portion of the program containing the function call precedes the function definition, then there must be a function declaration in the calling portion of the program. The function declaration effectively informs the compiler that a function will be accessed before it is defined. A function declaration can be written as.

```
datatype function name ( );
```

Function calls can span several levels within a program; function A can call function B so on.

15.6 PASSING ARGUMENT TO A FUNCTION

Arguments can be passed to a function by two methods, they are called passing by value and passing by reference. When a single value is passed to a function via an actual argument, the value of the actual argument is copied into the function. Therefore, the value of the corresponding formal argument can be altered within the function, but the value of the actual argument within the calling routine will not change. This procedure for passing the value of an argument to a function is known as passing by value.

Let us consider an example

```
#include <stdio.h>
    main()
    {
        int x=3;
        printf("\n x=%d(from main, before calling the
        function"),x);
        change(x);
        printf("\n\nx=%d(from main, after calling the
        function)",x);
    }
        change(x)
        int x;
    {
        x=x+3;
        printf("\nx=%d(from the function, after being
        modified)",x);
        return;
}
```

The original value of x (i.e. x=3) is displayed when main begins execution. This value is then passed to the function **change**, where it is sum up by 3 and the new value displayed. This new value is the altered value of the formal argument that is displayed within the function. Finally, the value of x within main is again displayed, after control is transferred back to main from change.

```
x=3 (from main, before calling the function)
```

x=6 (from the function, after being modified)

x=3 (from main, after calling the function)

Passing an argument by value allows a single-valued actual argument to be written as an expression rather than being restricted to a single variable. But it prevents information from being transferred back to the calling portion of the program via arguments. Thus, passing by value is restricted to a one-way transfer of information.

Arrays are passed differently than single-valued entities. If an array name is specified as an actual argument, the individual array elements are not copied to the function. Instead the location of the array is passed to the function. If an element of the array is accessed within the function, the access will refer to the location of that array element relative to the location of the first element. Thus, any alteration to an array element within the function will carry over to the calling routine.

15.7 SPECIFICATION OF DATA TYPES OF ARGUMENTS

The calling portion of a program must contain a function declaration if a function returns a non-integer value and the function call precedes the function definition. Function declaration may be included in the calling portion of a program even if it is not necessary. It is possible to include the data types of the arguments within the function declaration. The compiler will then convert the value of each actual argument to the declared data type and then compare each actual data type with its corresponding formal argument. Compilation error will result if the data types do not agree. We had already been use, data types of the arguments within the function declaration. When the argument data types are specified in a function declaration, the general form of the function declaration can be written as

data-type function name (argument type1, argument type2, ... argument type n);

Where data-type is the data type of the quantity returned by the function, function name is the name of function, and argument type/refer to the data types of the first argument and so on. Argument data types can be omitted, even if situations require a function declaration.

Most C compilers support the use of the keyword void in function definitions, as a return data type indicating that the function does

not return anything. Function declarations may also include void for the same purpose. In addition, void may appear in an argument list, in both function definitions and function declarations, to indicate that a function does not require arguments.

15.8 FUNCTION PROTOTYPES AND RECURSION

Many C compilers permits each of the argument data types within a function declaration to be followed by an argument name, that is

data-type function name (type1 argument 1, type 2 argument2... type n argument n); Function declarations written in this form are called **function prototypes**.

Function prototypes are desirable, however, because they further facilitate error checking between the calls to a function and the corresponding function definition. Some of the function prototypes are given below:

int example (int, int); or int example (int a, int b);

void example 1(void); or void example 1(void);

void fun (char, long); or void fun (char c, long f);

The names of the arguments within the function declaration need not be declared elsewhere in the program, since these are "dummy" argument names recognized only within the declaration. "C" language also permits the useful feature of **'Recursion'**.

Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied. The process is used for repetitive computations in which each action is stated in terms of a precious result. In order to solve a problem recursively, two conditions must be satisfied. The problem must be written in a recursive form, and the problem statement must include a stopping condition. The best example of recursion is calculation of factorial of a integer quantity, in which the same procedure is repeating itself. Let us consider the example of factorial:

```
#include <stdio.h>
    main()
    {
        int number;
        long int fact(int number);
```

```
        printf("Enter number");
        scanf("%d", & number);
        printf("Factorial of number is % d\n", fact(number));
    }
        long int fact(int number)
{
        if(number <=1)
        return(1);
        else
        return(number *fact(number-1));
    }
```

The point to be noted here is that the function 'fact' calls itself recursively, with an actual argument (n-1) that decrease in value for each successive call. The recursive calls terminate the value of the actual argument becomes equal to 1.

When a recursive program is executed, the recursive function calls are not executed immediately. Instead of it, they are placed on a stack until the condition that terminates the recursion is encountered. The function calls are then executed in reverse order, as they are popped off the stack.

The use of recursion is not necessarily the best way to approach a problem, even though the problem definition may be recursive in nature.

INTEXT QUESTIONS

4. Can a function be called from more than one place within a program?
5. What is the purpose of the keyword void in a function declaration?
6. When a function is accessed, must the names of the actual arguments agree with the name of the arguments in the corresponding function declaration?
7. What is recursion?

15.9 STORAGE CLASSES – AUTOMATIC, EXTERNAL, STATIC VARIABLES

There are four different storage-class specification in 'C', automatic, external, static and register. They are identified as auto, extern, static and register respectively.

Automatic variables are always declared within a function and are local to the function in which they are declared, that is their scope is confined to that function. Automatic variables defined in different functions will therefore be independent of one another. The location of the variable declarations within the program determine the automatic storage class, the keyword auto is not required at the beginning of each variable declaration.

These variables can be assigned initial value by including appropriate expressions within the variable declarations. An automatic variable does not retain its value once control is transferred out of its defining function. It means any value assigned to an automatic variable within a function will be lost once the function is exited. The scope of an automatic variable can be smaller than an entire function. Automatic variables can be declared within a single compound statement.

External variables are not confined to single functions. Their scope extends from the point of definition through the remainder of the program. External variable are recognized globally, that means they are recognized throughout the program, they can be accessed from any function that falls within their scope. They retain their assigned values within their scope. Therefore, an external variable can be assigned a value within one function and this value can be used within another function. With the use of external variables one can transfer the information between functions.

External variable definitions and external variable declarations are not the same thing. An external variable definition is written in the same manner as an ordinary variable declaration. The storage-class specifier extern is not required in an external variable definition, because these variables will be identified by the location of their definition within the program. An external variable declaration must begin with the storage class specifier extern. The name of the external variable and its data type must agree with the corresponding external variable definition that appears outside of the function. The declaration of external variables cannot include the assignment of initial values. External variables can be assigned initial values as

a part of the variable definitions, but the initial values must be expressed as constants rather than as expression. These initial values will be assigned only once, at the beginning of the program. If an initial value is not included in the definition of an external variable, the variable will automatically be assigned a value of zero.

Static variables are defined within individual functions and therefore have a same scope as automatic variables, i.e. they are local to the functions in which they are defined. Static variables retain their values throughout the program. Thus, if a function is exited and re-entered later, the static variables defined within that function will retain their former values. Static variables are defined within a function in the same manner as automatic variables, but its declaration must begin with the static storage class designation. They cannot be accessed outside of their defining function. Initial values can be included in static variable declarations. The initial value must be expressed as constants, not expression, the initial values are assigned to their respective variables at the beginning of program,execution. The variables retain these values throughout the program, unless different values are assigned during the program. This is all for storage classes auto, extern and static.

Let us consider and example of static variables:

static int a;

If the keyword static is replaced with the **keyword** auto, the variable is declared to be of storage class auto. If a static local variable is assigned a value the first time the function is called, that value is still there when the function is called second time.

```
        display_number()
{
        static int number=2;
        printf("number=%d\n", number);
        number++;
}
```

When the first time display_number is called, it prints the value 2, to which number is initialized. Then number is incremented to 3, and terminates. The second time display_number is called, it prints the value of 3. On the third call, the value printed is 4 and so on. Point to be noted here is that the initialization is not performed after the first call. An initialization used in a declaration occurs only

once-when the variable is allocated. Since a static variable is allocated only once, the initialization occurs only during the entire program, no matter how many times the function is called. When the display_number function in the example is called the second time, the value found in the variable number is the value left there by the previous call to the function.

15.10 WHAT YOU HAVE LEARNT

In this lesson you have learnt about functions, how to define and declare functions, what are the functions different data types. You are now able to execute function from different places of a program. You are now familiar with useful feature of functions say recursion. In this lesson we are also discussed different storage classes like auto, extern and static.

15.11 TERMINAL QUESTIONS

1. What is meant by a function call?
2. What is the purpose of the return statement?
3. What are the three principal components of a function definition?
4. Can multiple expressions be included in a return statement? Can multiple statements be included in a function?
5. How are argument data types specified in a function declaration?
6. Name the four storage-class specifications in C?
7. What is the importance or use of a automatic variable?
8. Differentiate between external and static variable?
9. What is the scope of a static external variable?
10. Does static variables retain their values even after exiting the function?

15.12 KEY TO INTEXT QUESTIONS

1. The statements of a user-defined function are written as part of program itself, according to the programmer's requirements.
2. A function may be designed to carry out some frequently used task, and called upon when needed merely by specifying the name of the function.

3. One
4. Yes
5. It means the function does not return any value to the calling program
6. It is not necessary
7. Recursion is a process by which a function calls itself repeatedly.

16

ARRAYS

16.1 INTRODUCTION

As we stated earlier that the variables are the entities in 'C' which are used to hold data in memory. But the concept of variables does not solve the indeterminate number of values is to be stored and operated upon. In case of storing say 100 values, it is a very difficult task to store 100 different variable names with their values. Arrays are the solution to this problem. Arrays are nothing but a single name to a whole group of similar data. The position in terms of arrays is known as subscript. Each subscript must be expressed as a non-negative integer.

y[0]	y[1]	y[2]	y[3]

The number of subscripts determines the dimension of the array.

16.2 OBJECTIVES

After going through this lesson you will be able in a position to

- define AN ARRAY
- process AN ARRAY
- pass ARRAYS TO FUNCTIONS
- process MULTIDEMNSIONAL ARRAYS AND STRINGS.

16.3 DEFINING AN ARRAY

Arrays are defined in much the same manner as ordinary variables except that each array name must be followed by a size specification. For a one-dimensional array, the size is specified by a positive integer expression enclosed in square brackets.

For example

Storage-class data-type array name[expression];

```
int a[10];
char text[100];
static char text[100];
```

The array's size can be defined in terms of a symbolic constant rather than a fixed integer quantity.

The following example reads a one-dimensional character array. and convert it to uppercase and then print it.

```
# include <stdio.h>
main()
{
char str[20];
int i;
printf("Enter any line of 9 characters");
scanf("\s", str);
scan("%s", str);
for(i=0;, i<20; ++i)
{
putchar(toupper(str[i] ));
}
}
```

Automatic arrays, unlike automatic variables, cannot be initialized. But the definitions of external and static arrays can include the assignment of initial values if required. The initial values must appear in the order in which they will be assigned to the individual array elements, enclosed in braces and separated by commas. The general

form is

storage class data-type array. name[expression]={value1, value2 - - - , value n};

For example, char school[4] ={'O','P','E','N'};

```
int marks[10]={1,2,3,4,5,6,7,8,9,10};
static float y[3]={o,0.3,0.2};
```

All individual array elements that are not assigned explicit initial values will automatically be set to zero. This includes the remaining elements of an array in which certain element have been assigned non zero values.

The array size need not be specified explicitly when initial values are included as a part of an array definition. With a numerical array, the array size will automatically be set equal to the number of initial values included within the definition.

For example

```
int array[]={4,8,3,7,5};
```

Since the square brackets following the array name are empty, the compiler determines how many elements to allocate for the array by counting the number. of values within the curly braces. This approach can help to avoid errors. If the dimension is specified explicitly, and the curly braces contain more initialization values than are needed, a syntax error is flagged by the compiler.

The case of strings are different. The array size specification in this case is usually omitted. The proper array size will be assigned automatically. This will include a provision for the null character.

If a program requires a one-dimensional array declaration the declaration is written in the same manner as the array definition with the following exception.

1. The square brackets may be empty, since the array size will have been specified as a part of the array definition. Array declaration are customarily written in this form.

2. Initial values cannot be included in the declaration. Following are the examples of defining an External array.

```
int a[]= { 1,2,0};              /*external array definition */
char text []="open";            /*external array definition  */
extern void dummy(void);        /* external function declaration */
```

16.4 PROCESSING AN ARRAY

Single operations involving entire array are not permitted in 'C'. If two arrays are same in every respect then different operations must be carried out on an element by element basis. This usually accomplished within a loop, where each pass through the loop is used to process one array element. The number of passes through the loop will therefore equal the number of array elements to be processed.

Let us consider an example of processing an array.

```
#include <stdio.h>
main()
{
int number,i;
float avg, sum=0;
float arr[20];
printf("How many numbers will be averaged");
scanf("%d", & number);
printf("\n");
for (i=0; i<number; ++i);
printf ("Enter number")
{
scanf("%f", & arr[i]);
sum +=arr[i];
}
avg= sum/number;
printf("\n The average is %f\n", avg);
}
```

Here, each value gets entered in the array "arr" one-by-one so arr [0] element will fill first then arr[1] - - and so on.

16.5 PASSING ARRAYS TO FUNCTIONS

An array name can be used as an argument to a function, thus permitting the entire array to be passed to the function.

To pass an array to a function, the array name must appear by itself, without brackets or subscripts, as an actual argument or parameter within the function call. The corresponding formal parameter is written in the same manner, though it must be declared as an array within the formal argument declarations. When declaring a one-dimensional array as a formal argument, the array name is written with a pair of empty square brackets. The size of the array is not specified within the formal argument declaration.

Let us consider an example of passing arrays to a function

```
# include <stdio.h>
main()
{
int n, i;
float avg;
float avr[20]; /*array definition */
float average();/* function declaration */
printf("Enter number");
scanf("%d", &n);
printf("\n");
for (i=0; i<n; ++i)
{
printf("Enter number");
scanf("%f",& arr[i]);
}
        avg=average (n,arr);
        printf("The average of numbers is %f", avg);
}
        float average(a,x)   /*function DEFINITION */
        int a;                  /*formal argument declaration */
        float x[ ];             /* formal argument (array) declaration*)
{
    float sum ; int i=0;
    float avg=0;
    for (i=0; i<a; ++i)
    {
```

```
        sum+=x[i];
}
        avg=sum/a;
        return avg;
}
```

Within main there is a call to the function average. This function call contains two actual argument—the integer variable n, and the one dimensional, floating-point array arr. The arr appear as an ordinary variable, within the function call.

In the first line of the function definition, there are 2 formal arguments, called a and x.

When an array is passed to a function, however, the values of the array element are not passed to the function. But the array name is interpreted as the address of the first array element. This address is assigned to the corresponding formal argument when the function is called. The formal argument therefore becomes a pointer to the first array element. Arguments passed in this manner are said to be passed by reference rather than by value. When a reference is made to an array element within the function , the value of the element's subscript is added to the value of the pointer to indicate the address of the specified array element. Therefore, any array element can be accessed from within the function . If an array element is altered within the function., the alteration will be recoganised in the calling portion of the program.

With the return statement array cannot be used. If the elements of an array are to be passed back to the calling portion of the program, the array must either be defined as an external array or it must be passed to the function as a formal argument.

INTEXT QUESTIONS

1. What is an array ?
2. How is the integer array x, containing 50 elements, declared in 'C'?
3. What is the subscript of the first element of an array in 'C' ?.
4. What are the rules for naming array ?

5. Is it possible to declare and initialize an array in 'C' simultaneously ? If so, how ?

6. Must the elements of an array be read in or printed out in order of subscript ?

16.6 MULTIDIMENSIONAL ARRAYS

The arrays we have used so far have been one dimensional. The elements of the array could be represented either as a single column or as a single row.

A two dimensional array is a grid containing rows and columns, in which each element is uniquely specified by means of its row and column coordinates. Multidimensional arrays are defined in much the same manner as one dimensional arrays, except that a separate pair of square brackets is required for each subscript. Thus a two-dimensional array will require two pairs of square brackets, a three dimensional array will require three pairs of square brackets and so on. A multidimensional array definition can be written as

storage-class data-type array-name[expression1][expression 2] ——[expression n];

The two dimensional array is like a matrix where position of each element is specified by both the column and row numbers.

For example:

Matrix x

		0	1	2	3	Column
Row	0	4	5	1	0	
	1	2	6	9	1	
	2	5	2	6	7	

Suppose if want to locate '9' then its specification is x[1][2]

The row subscript generally is specified before the column subscript. In C, each subscript be written within its own separate pair of brackets.

For example: Definition of Multidimensional arrays are given below.

float table[10][10];

char string[10]20];

If a multidimensional array definition include the assignment of initial values , then care must be given to the order in which the initial values are assigned to the array element. The rule is that the last subscript increases most rapidly, and the first subscript increases least rapidly. Thus, the elements of a two dimensional array will be assigned by rows, that is , the elements of the first row will be assigned, then the elements of the second row, and so on.

Suppose int sample [3][4]={1,2,3,4,5,6,7,8,9,10,11,12}

Thus sample [0][0]=1 sample[0][1]=2 sample [0][2]=3 sample [0][3]=4 sample[1][0]=5

sample[1][1]=6 sample[1][2]=7 sample[1][3]=8 sample[2][0]=9 sample[2][1]=10 sample[2][2]=11 sample[2] [3]=12

You can write another definition for array sample as given below:

```
int sample[3][4]={
    {1,2,3,4},
    {5,6,7,8,},
    {9,10,11,12}
    };
```

If the definition is given like this

```
int sample[3][4]={
{1,2,3},
{4,5,6},
{7,8,9}
};
```

Then this definition assigns values only to the first three elements in each row. Rest last 3 element of each row gets a value of zero.

If the above definition is written in another way like

int sample [3][4]={1,2,3,4,5,6,7,8,9};

then here also 3 elements will be assigned zeros, but the order of the assignments will be different. In this case the last 3 elements will get a value zero.

Multidimensional arrays are processed in the same manner as one dimensional arrays, on an element-by-element basis.

Let us understand the example of multidimensional arrays of adding two tables of numbers.

```
#include <stdio.h>
main()
{
int rows, cols;
int a [10][20], b [10][20], c [10][20],
void read (int a [ ][20], int rows, int cols);
void sum (int a [][20], int b [ ][20], int c [ ][20], int rows, int-
cols);
void write( int c [ ][20], int rows, int cols);
printf( “Enter number of rows and columns”);
scanf(“%d %d”, & rows, & cols);
printf(“\n” First Table” );
read (a,rows, cols);
printf(“\n\n Second table” );
read(b, rows, cols);
sum(a,b,c, rows, cols);
write(c, rows, cols);
}
void read(int a[ ] [20], int m, int n)
{
int row, col;
for(row-0; row<m; ++ row)
{
for(col=0; col<n; ++ cols)
scanf(“%d”, & a[row][col]);
}
return;
```

```
}
void sum (int a[ ][20], int b[ ] [20], int c[ ] [20], int m, int n)
{
int row, col;
for(row=0; row<m; ++ row)
for (col=0; col<n; ++ col)
c[row][col]=a[row][col]+b[row][col];
return;
}
void write(int a[ ][20], int m, int n)
{
int row, col;
for(row=0; row<m; ++row){
for(col=0; col<n; ++col)
printf("%d", a[row][col]);
printf("\n");
}
return;
}
```

INTEXT QUESTIONS

7. How are multidimensional arrays defined ?

8. If all the elements of a two-dimensional array are initialized in the declaration of the array both subscripts may be omitted , is it is true or false ?

16.7. ARRAYS AND STRINGS

Strings can be represented as a one-dimensional character-type array. Each character within the string will be stored within one element of the array. For example,

char name[ ]= {'N', 'A', 'T','I','O','N','A','L','\0'};

Each character in the array occupies one byte of memory and the last character is always '\0'. The terminating null is important, because it is the only way the functions that work with a string can

know where the string ends. In facts, a string which is not terminated by a '\0' is not really a string but merely a collection of characters. The above string can be also initialized as

char name[ ]= "National";

In this declaration '\0' is not necessary. C inserts the null character automatically.

Let us consider an example to demonstrate printing of string

```
#  include <stdio.h>
main()
{
char name [ ]="National";
int i= 0;
while(i<=8)
{
printf("%c", name [i];
i ++;
}
}
```

The output is **National.**

With the help of null character this program can be written in the following manner:

```
#include <stdio.h>
main()
{
char name [ ]=**"National";**
int i=0;
while (name [i]!='\0')
{
printf ("%c", name[i ] );
i++;
}
}
```

printf() does not print the '\0'. The %s used in printf() is a format specification for printing out a string.

```
#include<stdio.h>
main()
{
char name[25];
printf("Enter your name");
scanf("%s", name);
printf("%s", name);
}
```

It will print the same string as entered from the user. The length of string should not exceed the dimension of the character array. This is because the C compiler doesn't perform bounds checking on character arrays. scanf() is not capable of receiving multi word strings. The way to get around this limitation is by using the function gets (). The example is shown below:

```
#include <stdio.h>
main()
{
char name [25];
printf("Enter your full name");
gets(name);
puts(name);

}
```

Following is an example to print the length of string using gets and null character.

```
# include <stdio.h>
main()
{
int i;
char name [30];
printf("enter some string");
gets(name);
for(i=0; name [i]!= '\0'; i++);
```

```
printf("The length of the string is %d", i);
}
```

The length of the string can be calculated with the help of standard library function strlen().

```
#include <stdio.h>
main()
{
int i;
char name [30];
printf("enter some string");
gets(name);
i=strlen(name);
printf("The length of the string is %d",i);

}
```

There are strcpy(), strcmp, strcat() standard library functions all of these have different usuages.

```
# include <stdio.h>
main()
{
char name [ ]="National";
char name1[20];
strcpy(name1, name);
printf("\n Original string=%s", name);
printf("\n Copied  string =%s", name1);
}
```

output is : **Original string=National**

Copied string= National

strcpy() function copies one string to another.

Similarly, strcmp() function compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first. If the two strings are identical, strcmp() returns a value zero. If they are not, it returns the numeric

difference between the ASCII values of the non-matching character.

```
#include <stdio.h>
main()
{
char name[ ]="National";
char name1[ ]= "Open";
int i, j, k;
i=strcmp(name,"National");
j=strcmp(name, name1);
k= strcmp(name, "National School");
printf("\n %d%d%d", i,j,k);
}
```

The output is as follows:

0,1,-32

In the first call to strcmp(), the 2 strings are identical, "National" and "National" and the value returned by strcmp() is zero. In the second call, the first character of "National", does not match with first character of "Open" and the result is 1, which is the numeric difference between ASCII value of N and 0. In the third call to strcmp() "National" does ASCII not match with National school, because the null character at the end of National does not match the blank in National school. The values returned is –32, which is the value of null character minus the ASCII value of space i.e '\0' minus ' ', which is equal to –32.

You can do rest of the standard library functions by yourself.

Two dimensional Array of characters.

Let us first consider an example of 2D array of character.

```
#include <stdio.h>
main()
{
char sample[6][10]= {
"Vaishali"
```

```
"Akanksha"
"Shivansh"
"Tanishq"
"Anmol"
"Yash"
};
int i, flag, a;
char name[10];
printf("Enter your name");
scanf("%s", name);
flag=False;
for(i=0, i<5;i++)
{
a = strcmp(&sample[i][0], name);
if(a= = 0)
{
printf("You are a right candidate");
flag =True;
break;
}
}
if(flag= =False)
printf("Wrong candidate");
}
```

In this program note that how the two dimensional character array has been initialised. The order of the subscripts in the array declaration is important. The first subscript gives the number of names in the array, while the second subscript gives the length of each item in the array.

The function strcmp() is also used to compare two strings which will give the value 0 if strings are equal. The names would be stored in the memory as shown in fig. Each string ends with '\0'. The arrangement is somewhat similar to that of a two dimensional numeric array.

1001	V	a	i	s	h	a	l	i	\0	
1011	A	k	a	n	k	s	h	a	\0	
1021	S	h	i	v	a	n	s	h	\0	
1031	T	a	n	i	s	h	q	\0		
1041	A	n	m	o	l	\0				1060 (last location)
1051	Y	a	s	h	\0					

Two dimensional character Array.

As scan from the above pattern some of the names do not occupy all the bytes reserved for them. For example, even though 10 bytes are reserved for storing the name "Vaishali", it occupies only 9 bytes. Thus 1 byte go waste. Similarly, for each name there is some amount of wastage. This wastage of memory can be avoided using arrays of pointers which we will discuss in next chapter.

16.8 WHAT YOU HAVE LEARNT

In this lesson you have learnt about arrays, what are one-dimensional array and two dimensional arrays. Now you can easily pass an array to any function. You have also learnt about multidimensional arrays and strings.

16.9 TERMINAL QUESTIONS

1. When passing an array to a function, how must the array argument be written?
2. How can a list of strings be stored within a two dimensional array ?
3. Write a program to compare two strings entered by the user without help of strcmp() function.
4. Write a program to copy a string without help of strcpy() function.
5. Write a program to enter strings from user and then sort them in alphabetical order.(You can use string standard library functions)
6. Write a program to count the number of vowels in a string entered from the user.
7. Write a program to convert a string from lowercase to uppercase.
8. Write a program to join two strings e.g. Suneeta, Gupta.

16.10 KEY TO IN-TEXT QUESTIONS

1. Array is an ordered collection of elements that share the same name.

2. int x[50];

3. Zero

4. The same as for naming regular variables or functions. An array cannot have the same name as a variable or function within the same program.

5. Yes, the array declaration is followed immediately by an equal sign. This is followed by the list of values to be assigned enclosed in braces.

6. No, the elements of an array may be accessed in any order at all.

7. Storage-class data type array name[expression 1][expression 2] - - — [expression n];

8. False, only the first (row) subscript can be omitted, with first pair of square brackets left empty. The second subscript must always be explicitly specified.

17

POINTERS

17.1 INTRODUCTION

As you know by now, variables are stored in memory. Each memory location has a numeric address, in much the same way that each element of an array has its own subscript. Variable names in C and other high-level languages enable the programmer to refer to memory locations by name, but the compiler must translate these names into addresses. A pointer is a variable that represents the location of a data item, such as a variable or an array element. Pointers are used frequently in C. Pointers can be used to pass information back and forth between a function and its reference point. In particular, pointers provide a way to return multiple data items from a function via function arguments. Pointers also permit references to other functions to be specified as arguments to a given function. This has the effect of passing functions as arguments to the given function.

17.2 OBJECTIVES

After going through this lesson you will be in a position to

- explain address operator
- define pointer declarations

- describe pointers and one-dimensional arrays
- define pointers and multidimensional arrays
- explain arrays of pointers

17.3 THE ADDRESS OPERATOR

Within a computer's memory, every stored data item occupies one or more contiguous memory cells. The number of memory cells required to store a data item depends on the type of data item. For example, a single character will typically be stored in 1 byte of memory, an integer requires two contiguous bytes. A floating point number may require four contigueous bytes, and a double precision quantity may require eight contiguous bytes.

Suppose 'a' is a variable that represents some particular data item. The compiler will automatically assign memory cells for this data item. The data item can be accessed if we know the location of the first memory cell. The address of a's memory location can be determined by the expression &v, where & is a unary operator called the **address operator**, that evaluates the address of its operand. Now let us assign the address of 'a' to another variable,

Thus

pa= &a;

This new variable is called a pointer to 'a', since it "points" to the locations where 'a' is stored in memory. However, pa represents a's address, not its value. Thus, pa is referred to as a pointer variable.

Address of a $\rightarrow$ value of a

pa a

The data item represented by 'a' (i.e the data item stored in a's memory cells) can be accessed by the expression *pa, where * is a unary operator, called the **indirection operator**, that operates only on a pointer variable. Therefore, *pa and 'a' both represent the same data item (ie the contents of the same memory cells). Further more if we write pa= &a and b = *pa, then 'a' and 'b' will both represents the same value i.e the value of a will indirectly be assigned to 'b'

Let us consider an example :

```
# include <stdio.h>
main ( )
{
int a=3;
int b;
int *pa; /* to an integer */
int *pb;/*pointer to an integer */
pa=&a;/* assign address of a to pa */
b=*pa;/* assign value of a to b*/
pb=&b;/* assign address of b to pb */
printf ("\n% d %d % d %d", a, &a,pa, *pa);
printf ("\n% d %d % d %d", b, &b, pb, *pb);
}
```

The output is :

```
3    F8E   F8E   3
3    F8C   F8C   3
```

The unary operators & and * are members of the same precedence group as the other unary operators i.e. -, ++, —, ! , size of and(type). The address operator (&) must act upon operands associated with unique addresses, such as ordinary variables or single array elements. Thus, the address operator cannot act upon arithmetic expression, such as 2*(a+b). The indirection operator(*) can only act upon operands that are pointers.

Let us consider an example

```
# include <stdio.h>
main()
{
int i=3,* j;k;   /*j is a pointer to integer */
j=&i;      /*j points to the location of i*/
k=*j ;          /*assign to k the value pointed to by j*/
*j=4            /assign 4 to the location pointed to by j*/
printf("i=%d, *j=%d, k=%d\n," i, *j, k);
}
```

The output is as follows:

i=4,*j=4, k=3

17.4 POINTER DECLARATION

Pointer variables, like all other variables, must be declared before they may be used in a C program. The interpretation of a pointer declaration is somewhat different than the interpretation of other variable declarations. When a pointer variable is declared, the variable name must be preceded by an asterisk(*). This identifies the fact that the variable is a pointer. The data type that appears in the declaration refers to the object of the pointer, i.e the data item that is stored in the address represented by the pointer, rather than the pointer itself.

Thus, a pointer declaration may be written in general terms as datatype *ptr;

Where ptr is the name of the pointer variable and data-type refers to the data type of the pointer's object.

For example,

```
float a,b;
float *pb;
```

The first line declares a and b to be floating-point variables. The second line declares pb to be a pointer variable whose object is a floating-point quantity i.e. 'pb' point to a floating point quantity. 'pb' represents an address, not a floating-point quantity.

Within a variable declaration, a pointer variable can be initialized by assigning it the address of another variable.

17.5 POINTERS AS ARGUMENTS

Pointers are often passed to a function as arguments. This allows data items within the calling portion of the program to be accessed by the function, altered within the function, and then returned to the calling portion of the program in altered form.

When an argument is passed by value, the data item is copied to the function. Thus any alteration made to the data item within the function is not carried over into the calling routine when an argument is passed by reference, however the address of a data item is passed to the function. The contents of that address can be accessed freely,

either within the function or within the calling routine. Moreover, any change that is made to the data item will be recognized in both the function and the calling routine. Thus, the use of a pointer as a function argument permits the corresponding data item to be altered globally from within the function.

If formal arguments are pointers, then each must be preceded by an asterisk. Also, if a function declaration is included in the calling portion of the program, the data type of each argument that corresponds to a pointer must be followed by an asterisk.

Let us consider an example of passing by reference

```
#include <stdio.h>
main ()
{
int i,j;
i=2;
j=5;
printf("i=%d and j=%d\n", i,j);
/* Now exchange them */
swap(&i,&j);
printf("\nnow i=%d and j=%d\n", i,j);
}
swap(i,j)
int *i,*j;
{
int temp=*i; /* create temp and store into it the value  pointed to by "i"*/
*i=*j; /*The value pointed to by j is stored in the location pointed to by i*/
*j=temp;/* Assign temp to the location pointed to by j*/
}
```

The output is as follows:

i=2 and j=5

Now i=5 and j=2

If the formal parameters i and j of the swap function were declared merely as integers, and the main function passed only i and j rather

than their addresses, the exchange made in swap would have no effect on the variables i and j in main. The variable temp in swap function is of type int, not int*. The values being exchanged are not the pointers, but the integers being pointed to by them. Also temp is initialized immediately upon declaration to the value pointed to by i.

Let us consider it again with the help of another example:-

The following program counts the no. of vowels and consonants in a line of text with the help of pointers.

```
# include <stdio.h>
main()
{
char str[80];
int vowels=0;
int consonants=0;
void scan_line(char str[ ], int *v, int *c);
printf("Enter a line of text\n");
scanf("%[ "\n]" , str);
scan_line(str, &vowels, &consonants);
printf("\No. of vowels %d",vowels);
printf("\No. of consonants %d", consonants);
}
void scan_line(char str[ ], int * v, int *c)
{
char c;
int count=0;
while ((c=toupper(str[count]))!='\0')
{
if(c = = A !!     c= =E !!   c= =I !!     c= =O !!     c= =U)
++*v;
else if(c>='A'&& c<='Z')
++*c;
++ count;
}
```

```
return;
}
```

It is possible to pass a portion of an array, rather than an entire array, to a function. To do so, the address of the first array element to be passed must be specified as an argument. The remainder of the array, starting with the specified array element, will then be passed to the function.

INTEXT QUESTIONS

1. What numeric value is associated with every memory location ?
2. How are variable names translated to their corresponding addresses?
3. What must be done to a pointer variable before it can be put to use?
4. What is the role played by the & operator in a call to the scanf function ?
5. Can addresses be stored ? If so, where ?

17.6 POINTERS AND ONE MIDEMNSIONAL ARRAYS

Array name is really a pointer to the first element in that array. Therefore, if x is a one-dimensional array, then the address of the first array element can be expressed as either &x[0] or x. Similarly, the address of the second array element can be written as either & x [1] or as (x+1) and so on. Here we should keep it in mind that the expression (x+1) is a symbolic representation for an address specification rather than an arithmetic expression. When writing the address of an array element in the form(x+i), there is no need to concern with the number of memory cells associated with each type of array element, the C compiler adjusts for this automatically. The programmer must specify only the address of the first array element and the number of array elements beyond the first.

Since &x[i] and (x+i) both represent the address of the i^{th} element of x then x [i] and *(x+i) both represent the contents of that address. The two terms are interchangeble.

When assigning a value to an array element such as x[i] the left side of the assignment statement may be written as either x[i]or *(x+i).

Thus a value may be assigned directly to an array element, or it may be assigned to the memory area whose address is that of the array element . Thus expression such as x,(x+i) and &x [i] cannot appear on the left side of an assignment statement.

Numerical array elements cannot be assigned initial values if the array is defined as a pointer variable. Therefore a conventional array definition is required if initial values will be assigned to the elements of a numerical array.

Let us assume that x is to be defined as a one-adimentional, 10 element array of integers. It is possible to define x as a pointer variable rather than as an array. Thus we can write int *x; instead of int x[10];

However x is not automatically assigned a memory block when it is defined as a pointer variable, To assign sufficient memory for x ,we can make use of the library function malloc as follows :

x= malloc (10*size of (int));

This function reserves a block of memory whose size is equivalent to the size of an integer quantity. The function return a pointer to a character. If the declaration is to include the assignment of initial values, then x must be defined as an array rather then as a pointer variable for example

```
int x[10]= {1,2,3,4,5,6,7,8,9,10};
   or int x[]= {1,2,3,4,5,6,7,8,9,10};
```

For character arrays you can write them in the form of pointer as char*x="this is string";

17.7 ARITHMETICS ON POINTERS

An integer value can be added to or subtracted from a pointer variable. For example, px is a pointer variable representing the address of some variable x. We can write expressions such as a ++ px ,- - px (px +3),(px+i) and (px – i) where i is an integer variable. Each expression will represent an address located some distance from the original address represented by px. The exact distance will be the product of the integer quantity and the number. of bytes or words associated with the data item to which px points. One pointer variable can be subtracted from another provided both variables points to elements of the same array. The resulting value indicates the number of words or bytes separating the corresponding array elements.

Pointer variables can be compared provided both variables point to objects of the same data type. Such comparisons can be useful when both pointer variables point to elements of the same array. The following points must be kept in mind about operations on pointers:-

- A pointer variable can be assigned the address of an ordinary variable.
- A pointer variable can be assigned the value of another pointer variable provided both pointers point to objects of the same data type.
- An integer quantity can be added to or subtracted from a pointer variable.
- A pointer variable can be assigned a null (zero) value.
- One pointer variable can be subtracted from another provided both pointers point to elements of the same array.
- Two pointer variables can be compared provided both pointers point to objects of the same data type.

17.8 POINTERS AND MULTIDIMENSIONAL ARRAYS

A two dimensional array, is actually a collection of one dimensional arrays. Therefore, we can define a two dimensional array as a pointer to a group of contiguous one-dimensional arrays. A two dimensional array declaration can be written as

data type (*ptr) [expression2];

Notice the parentheses that surround the array name and the preceding asterisk in the pointer version of each declaration. These parentheses must be present. Without them we would be defining an array of pointers rather than a pointer to a group of arrays, since these particulars symbols (i.e. The square brackets and asterisk) would normally be evaluated right-to-by.

For example, if x is a two dimensional integer array having 10 rows and 20 columns, then we can declare x as

int(*x)[20];

rather than int x(10) (20);

Similarly, three dimmensional integer array can be written as int (*x) (20) (30)

An individual array element within a multi-dimensional array can be accessed by repeatedly using the indirection operator. If n is a 2D array having 10 rows and 20 columns, then item in row 2, column 5 can be accessed by writing either

x[2][5]

or

((x+2)+5)

Here, (x+2) is a pointer to row 2. Therefore, the object of this pointer *(x+2), refers to the entire row. Since row 2 is a one-dimensional array, *(x+2) is actually a pointer to the first element in row 2. We now add 5 to this pointer. Hence,(*(x+2)x5) is a pointer to element 5 in row 2. The object of this pointer, (*(x+2)+5) therefore refers to the item in column 5 of row2 which is x[2][5].

INTEXT QUESTIONS

6. What is meant by passing by reference ?

7. Why is only the address of an array's first element necessary in order to access the entire array ?

8. If an array is declared as

   char x[10][12];

   What is referred to by x[5] ?

17.9 ARRAYS OF POINTERS

A multi-dimensional array can be expressed in terms of an array of pointers rather than as a pointer to a group of contiguous arrays. In such cases the newly defined array will have one less dimension than the original multi-dimensional array. Each pointer will indicate the beginning of a separate (n-1) dimensional array.

In general terms, a 2D array can be defined as a one dimensional array of pointers by writing.

data type *array [expression 1];

In this type of declaration, array name and its preceding asterisk are not enclosed in parentheses. Thus, a right-to-left rule first associates the pairs of square brackets with array, defining the named object as an array. The preceding asterisk then establishes that the array will contain pointers.

For example x is a 2D integer array having 10 rows and 20 columns. We can define x as a one-dimensional array of pointers by writing

int *x[10];

An individual array element, such as x[2][5], can be accessed by writing *(x[2]+5)

In this expression, x[2] is a pointer to the first element in row 2, so that (x[2] +5) points to element 5 within row 2. The object of this pointer, *(x[2]+5), therefore refers to x[2][5].

Pointer arrays offer a particularly convenient method for storing strings. In this situation, each array element is a character-type pointer that indicates the beginning of a separate string. Each individual string can be accessed by referring to its corresponding pointer.

Let us consider an example of array of pointers: (To sort list of strings)

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int i, n=0;
char *x[10];
int reorder(int n, char *x[]);
printf(“Enter each string on a separate line,)
"Type\'END\' to finish”);
do
{
x[n]=malloc(12*size of (char));
printf(“\nstring %d”, n+1);
scanf(“%s”, x[n]);
```

```
}
while ( stringcmp(x[n++], “END”));
reorder(- - n,x);
printf (or \n\n Recordered List:\n");
for(i=0; i<n; ++i)
printf(“\n string %d:%s”, i+1, x[i]);
}
reorder(int n, char * x[])
{
char *temp;
int i, item;
for(item=0; item <n-1; ++ item)
for(i=item +1; i<n; ++i)
if (stringcmp(x[item], x[i]>0)
{
temp=x[item];
x[item]=x[i];
x[i]=temp;
}
return;
}
```

An advantage to use arrays of pointers is that a fixed block of memory need not be reserved in advance, as is done when initializing a conventional array.

If some of the strings are particularly long, there is no need to worry about the possibility of exceeding some maximum specified string length. Arrays of this type are ofttern referred to as **ragged arrays.**

17.10 PASSING FUNCTIONS TO OTHER FUNCTIONS

When a function declartion appears within another function, the name of the function being declared becomes a pointer to that function. Such pointer can be passed to other functions as arguments. This has the effect of passing one function to another , as though the first function was a variable. The first functiion can then be accessed within the second function. Successive calls to the second function can pas different pointers to the second funtion.

When a function accepts another function's name as an argument a formal argument declaration must identify that argument as a pointer to another function. A formal argument that is a pointer to a function can be declared as

data-type (*function name)();

Where data-type refers to the data type of the quantity returned by the function.

This function can then be accessed by means of the indirection operator. To do so the indirection operator must precede the function name. Both the indirection operator and the function name must be enclosed in parentheses that is

(*function-name)(argument 1, argument 2..., argument n);

In the last, we mention that pointer declaration can become complicated and some care is required for its interpretation. This is particularly take in case of declarations that involve functions or arrays.

One difficulty is the dusl use of parentheses. In particular, parentheses are used to indicate functions and they are used for nesting purposes within more complicated declarations. Thus the declaration

int *p(int a);

Indicates a function that accepts an integer argument, and returns a pointer to an integer. On the other hand, the declaration.

int(*p)(int a);

indicates a pointer to a function that accepts an integer argument and returns an integer. In this last declaration, the first pair of parentheses is used for nesting and the second pair is used to indicate a function.

17.11 WHAT YOU HAVE LEARNT

In this lesson you have learnt about pointers. You can now declare pointers very effciently. Now you can pass pointers to a function. You can do operations on pointer. Now you are familiar with arrays of pointers. You can use pointers with one as well as multi dimensional arrays.

17.12 TERMINAL QUESTIONS

1. What is meant by the address of a memory cell?
2. What kind of information does a pointer variable represent ?
3. How is a pointer variable declared ?
4. How can a function return a pointer to its calling routine ?
5. How is a multidimensional array defined in terms of a pointer to a collection of contiguous arrays of lower dimensioanlity ?
6. A 'C' program contains the following declaration :

   Static int x[8]= {10,20,30,40,50,60,70,80};

   (a) What is the meaning of x?

   (b) What is the meaning of (x+2) ?

   (c) What is the value of *x?

   (d) What is the value of (*x+2) ?

   (e) What is the value of *(x+2) ?

7. Write a program to find the length of a string using the concept of pointers
8. Write a program to copy a string using pointers.
9. Write a program to concatenate two strings using pointers.
10. Write a program to compare two strings using pointers.

17.13 KEY TO IN-TEXT QUESTIONS

1. its address
2. with the & operator
3. After it has been declared, it must be initialized
4. It passes the address of the variable being input, making it possible for the contents of that location to be changed.

5. Addresses can be stored in pointer variables.

6. Passing by reference is the term used to refer to the passing of a variable's address to a function.

7. Since all the elements of an array are of the same type, and all the elements are stored continguously in memory, the location of the first element can be used to find the second element, the third element, and so forth.

8. x[5] is a pointer to the sixth row (row #5) of the array.

18

STRUCTURES AND UNIONS

18.1 INTRODUCTION

We studied earlier that array is a data structure whose element are all of the same data type. Now we are going towards structure, which is a data structure whose individual elements can differ in type. Thus a single structure might contain integer elements, floating– point elements and character elements. Pointers, arrays and other structures can also be included as elements within a structure. The individual structure elements are referred to as members. This lesson is concerned with the use of structure within a 'c' program. We will see how structures are defined, and how their individual members are accessed and processed within a program. The relationship between structures and pointers, arrays and functions will also be examined. Closely associated with the structure is the union, which also contains multiple members.

18.2 OBJECTIVES

After going through this lesson you will be able to

- explain the basic concepts of structure
- process a structure
- use typedef statement
- explain the between structures and pointers

- relate structure to a function
- explain the concept of unions

18.3 STRUCTURE

In general terms, the composition of a structure may be defined as

```
struct tag
{ member 1;
member 2;

------------

-------------

member m; }
```

In this declaration, struct is a required keyword ,tag is a name that identifies structures of this type.

The individual members can be ordinary variables, pointers, arrays or other structures. The member names within a particular structure must be distict from one another, though a member name can be same as the name of a variable defined outside of the structure. A storage class, however, cannot be assigned to an individual member, and individual members cannot be initialized within a structure-type declaration. For example:

```
struct student
{
char name [80];
int roll_no;
float marks;
};
```

we can now declare the structure variable s1 and s2 as follows:

```
struct student s1, s2;
```

s1 and s2 are structure type variables whose composition is identified by the tag student.

It is possible to combine the declaration of the structure composition with that of the structure variable as shown below.

```
storage- class struct tag
```

{
member 1;
member 2;
- ——
- —-
- member m;
} variable 1, variable 2 --------- variable n;

The tag is optional in this situation.

```
struct student {
char name [80];
int roll_no;
float marks;
}s1,s2;
```

The s1, s2, are structure variables of type student.

Since the variable declarations are now combined with the declaration of the structure type, the tag need not be included. As a result, the above declaration can also be written as

```
struct{
char name [80];
int roll_no;
float marks ;
} s1, s2, ;
```

A structure may be defined as a member of another structure. In such situations, the declaration of the embedded structure must appear before the declaration of the outer structure. The members of a structure variable can be assigned initial values in much the same manner as the elements of an array. The initial values must appear in the order in which they will be assigned to their corresponding structure members, enclosed in braces and separated by commas. The general form is

storage-class struct tag variable = { value1, value 2,-------, value m};

A structure variable, like an array can be initialized only if its storage class is either external or static.

e.g. suppose there are one more structure other than student.

```
struct dob
{ int month;
int day;
int year;
 };
struct student
{ char name [80];
int roll_no;
float marks;
struct dob d1;
};
static struct student st = { " param", 2, 99.9, 17, 11, 01};
```

It is also possible to define an array of structure, that is an array in which each element is a structure. The procedure is shown in the following example:

```
struct student{
    char name [80];
    int roll_no ;
    float marks ;
    } st [100];
```

In this declaration st is a 100- element array of structures.

It means each element of st represents an individual student record.

18.4 PROCESSING A STRUCTURE

The members of a structure are usually processed individually, as separate entities. Therefore, we must be able to access the individual structure members. A structure member can be accessed by writing

variable.member name.

This period (.) is an operator, it is a member of the highest precedence group, and its associativity is left-to-right.

e.g. if we want to print the detail of a member of a structure then we

can write as

printf(“%s”,st.name); or printf(“%d”, st.roll_no) and so on. More complex expressions involving the repeated use of the period operator may also be written. For example, if a structure member is itself a structure, then a member of the embedded structure can be accessed by writing.

variable.member.submember.

Thus in the case of student and dob structure, to access the month of date of birth of a student, we would write

st.d1.month

The use of the period operator can be extended to arrays of structure, by writing

array [expression]. member

Structures members can be processed in the same manner as ordinary variables of the same data type. Single-valued structure members can appear in expressions. They can be passed to functions and they can be returned from functions, as though they were ordinary single-valued variables.

e.g. suppose that s1 and s2 are structure variables having the same composition as described earlier. It is possible to copy the values of s1 to s2 simply by writing

s2=s1;

It is also possible to pass entire structure to and from functions though the way this is done varies from one version of 'C' to another. Let us consider an example of structure:

```
#include <stdio.h>
struct date {
int month;
int day;
int year;
};
struct student{
```

```
char name[80];
char address[80];
int roll_no;
char grade;
float marks;
struct date d1;
}st[100];
main()
{
int i,n;
void readinput (int i);
void writeoutput(int i);
printf(“Student system”);
printf(“How many students are there ?”);
scanf(“%d” &n);
for (i=0; i<n; ++i){
readinput (i);
if( st[i].marks <80)
st[i].grade=’A’;
else
st[i].grade='A'+;
}
for (i=0; i<n; ++i)
writeoutput(i);
}
void readinput (int i)
{
printf(“\n student no % \n”, i+1);
printf(“Name:”);
scanf(“%[^\n]”, st[i].name);
printf(“Address:”);
scanf(“%[^\n]", st[i].address);
```

```
printf("Roll number");
scanf("%d", &st[i].roll_no);
printf("marks");
scanf("%f",&st[i].marks);
printf("Date of Birth {mm/dd/yyyy)");
scanf("%d%d%d", & st[i].d1.month & st[i].d1.day, & st[i].d1.year);
return;
}
void writeoutput(int i)
{
printf("\n Name:%s",st[i].name);
printf("Address %s\n", st[i].address);
printf("Marks % f \n", st[i].marks);
printf("Roll number %d\n", st[i].roll_no);
printf("Grade %c\n",st[i].grade);
return;
}
```

It is sometimes useful to determine the number of bytes required by an array or a structure. This information can be obtained through the use of the sizeof operator.

INTEXT QUESTIONS

1. What is the main reason for using structure ?
2. What special keyword is used in defining a structure ?
3. Define structure tag and what is its purpose?
4. In what two ways can a structure variable be declared?

18.5 USER-DEFINED DATA TYPES (Typedef)

The typedef feature allows users to define new data types that are equivalent to existing data types. Once a user-defined data type has been established, then new variables, arrays, structure and so on, can be declared in terms of this new data type. In general terms, a new data type is defined as

typedef type new type;

Where type refers to an existing data type and new-type refers to the new user-defined data type.

e.g. typedef int age;

In this declaration, age is user- defined data type equivalent to type int. Hence, the variable declaration

age male, female;

is equivalent to writing

int age, male, female;

The typedef feature is particularly convenient when defining structures, since it eliminates the need to repeatedly write struct tag whenever a structure is referenced. As a result, the structure can be referenced more concisely.

In general terms, a user-defined structure type can be written as

typedef struct
{ member 1;
member 2:
- - - -
- - - -
member m;
}new-type;

The typedef feature can be used repeatedly, to define one data type in terms of other user-defined data types.

18.6 STRUCTURES AND POINTERS

The beginning address of a structure can be accessed in the same manner as any other address, through the use of the address (&) operator.

Thus, if variable represents a structure type variable, then & variable represents the starting address of that variable. We can declare a pointer variable for a structure by writing

type *ptr;

Where type is a data type that identities the composition of the structure, and ptr represents the name of the pointer variable. We can then assign the beginning address of a structure variable to this pointer by writing

```
ptr= &variable;
```

Let us take the following example:

```
typedef struct {
char name [ 40];
int roll_no;
float marks;
}student;
student s1,*ps;
```

In this example, s1 is a structure variable of type student, and ps is a pointer variable whose object is a structure variable of type student. Thus, the beginning address of s1 can be assigned to ps by writing.

```
ps = &s1;
```

An individual structure member can be accessed in terms of its corresponding pointer variable by writing

```
ptr →member
```

Where ptr refers to a structure- type pointer variable and the operator → is comparable to the period (.) operator. The associativity of this operator is also left-to-right.

The operator → can be combined with the period operator (.) to access a submember within a structure. Hence, a submember can be accessed by writing

```
ptr → member.submember
```

18.7 PASSING STRUCTURES TO A FUNCTION

There are several different ways to pass structure–type information to or from a function. Structure member can be transferred individually , or entire structure can be transferred. The individual structures members can be passed to a function as arguments in the function call; and a single structure member can be returned via the return statement. To do so, each structure member is treated the same way as an ordinary, single- valued variable.

A complete structure can be transferred to a function by passing a structure type pointer as an argument. It should be understood that a structure passed in this manner will be passed by reference rather than by value. So, if any of the structure members are altered within the function, the alterations will be recognized outside the function. Let us consider the following example:

```
# include <stdio.h>
typedef struct{
char *name;
int roll_no;
float marks ;
} record ;
main ( )
{
void adj(record *ptr);
static record stduent={“Param”, 2,99.9};
printf(“%s%d%f\n”, student.name,
student.roll_no,student.marks);
adj(&student);
printf(“%s%d%f\n”, student.name,
student.roll_no,student.marks);
}
void adj(record*ptr)
{
ptr → name=”Tanishq”;
ptr → roll_no=3;
ptr → marks=98.0;
return;
}
```

Let us consider an example of transferring a complete structure, rather than a structure-type pointer, to the function.

```
# include <stdio.h>
typedef struct{
```

```
char *name;
int roll_no;
float marks;
}record;
main()
{
void adj(record stduent); /* function declaration */
static record student={“Param,” 2,99.9};
printf(“%s%d%f\n”,
student.name,student.roll_no,student.marks);
adj(student);
printf(“%s%d%f\n”,
student.name,student.roll_no,student.marks);
}
void adj(record stud) /*function definition */
{
stud.name=”Tanishq”;
stud.roll_no=3;
stud.marks=98.0;
return;
}
```

INTEXT QUESTIONS

5. What rules govern the use of the period (.) operator?
6. What is meant by an array of structure?
7. What characteristic must a structure have in order to be initialized within its declaration?
8. What is the meaning of the arrow operator?

18.8 UNIONS

Union, like structures, contain members whose individual data types may differ from one another. However, the members that compose a union all share the same storage area within the computer’s memory

, whereas each member within a structure is assigned its own unique storage area. Thus, unions are used to conserve memory.

In general terms, the composition of a union may be defined as union tag{

member1;

member 2;

- - -

member m

};

Where union is a required keyword and the other terms have the same meaning as in a structure definition. Individual union variables can then be declared as storage-class union tag variable1, variable2, -----, variable n; where storage-class is an optional storage class specifier, union is a required keyword, tag is the name that appeared in the union definition and variable 1, variable 2, variable n are union variables of type tag.

The two declarations may be combined, just as we did in the case of structure. Thus, we can write.

Storage-class union tag{

member1;

member 2;

- - -

member m

}variable 1, varibale2, ..., variable n;

The tag is optional in this type of declaration.

Let us take a 'C' program which contains the following union declaration:

union code{

char color [5];

int size ;

}purse, belt;

Here we have two union variables, purse and belt, of type code.

Each variable can represent either a 5–character string (color) or an integer quantity (size) of any one time.

A union may be a member of a structure, and a structure may be a member of a union.

An individual union member can be accessed in the same manner as an individual structure members, using the operators (→) and.

Thus if variable is a union variable, then varibale.member refers to a member of the union. Similarly, if ptr is a pointer variable that points to a union, then ptr→ member refers to a member of that union.

Let us consider the following C program:

```
# include <stdio.h>
main()
union code{
char color;
int size;
};
struct {
char company [10];
float cost ;
union code detail;
}purse, belt;
printf("%d\n", sizeof (union code));
purse.detail.color='B';
printf("%c%d\n", purse.detail.color,purse.detail.size);
purse.detail.size=20;
printf("%c%d\n", purse. detail.color,purse.detail.size);
}
```

The output is as follows:

```
2
B- 23190
@ 20
```

The first line indicates that the union is allocated 2 bytes of memory

to accommodate an integer quantity. In line two, the first data item [B] is **meaningful,** but the second is not. In line three, the first data item is meaningless, but the second data item [20] is meaningful. A union variable can be initialized, provided its storage class is either external or static. Only one member of a union can be assigned a value at any one time. Unions are processed in the same manner, and with the same restrictions as structures. Thus, individual union members can be processed as though they were ordinary variables of the same data type and pointers to unions can be passed to or from functions.

18.9 WHAT YOU HAVE LEARNT

In this lesson you have learnt about structures. Structures contain the variables of different data types. You can also pass structures to functions. You have also learnt about unions. All members of a union share the same storage area within the computer's memory.

18.10 TERMINAL QUESTIONS

1. What is a structure ? How does a structure differ from an array ?
2. What is a member ? What is the relationship between a member and a structure ?
3. How can a structure variable be declared ?
4. How is an array of structures initialized ?
5. How is a structure member accessed ? How can a structure member be processed ?
6. What is the precedence of the period operator ? what is its associativity ?
7. What is the purpose of the typedef feature ?
8. What is a Union ?
9. How does a union differ from a structure ?
10. How can an entire structure be passed to a function ?

18.11 KEY TO INTEXT QUESTION

1. Because it is helpful to group together non-homogenous data into a single entity.
2. struct
3. A structure tag is a name associated with a structure template. This makes it possible later to declare other variables of the same structure type without having to rewrite the template itself.
4. By preceding the variable name with the keyword struct and either a previously defined structure tag or a template.
5. An individual member of a structure is accessed by following the name of the structure variable with the period operator and the member name. The member name must be explicitly specified.
6. It is an array in which each element is a structure.
7. It must be of the static storage class.
8. The notation a b means (*a).b, that is, " the member named b of the structure pointed to by a.

19

DATA FILES

19.1 INTRODUCTION

Many applications require that information be written to or read from an auxillary storage device. Such information is stored on the storage device in the form of data file. Thus, data files allow us to store information permanently and to access later on and alter that information whenever necessary. In C, a large number of library functions is available for creating and processing data files. There are two different types of data files called stream-oriented (or standard) data files and system oriented data files. We shall study stream-orinted data files only in this lesson.

19.2 OBJECTIVES

After going through this lesson you will be able to

- open and close a data file
- create a data file
- process a data file

19.3 OPENING AND CLOSING A DATA FILE

We must open the file before we can write information to a file on a disk or read it. Opening a file establishes a link between the program and the operating system. The link between our program and

the operating system is a structure called FILE which has been defined in the header file “stdio.h”. Therefore, it is always necessary to include this file when we do high level disk I/O. When we use a command to open a file, it will return a pointer to the structure FILE. Therefore, the following declaration will be there before opening the file,

FILE *fp each file will have its own FILE structure. The FILE structure contains information about the file being used, such as its current size, its location in memory etc. Let us consider the following statements,

```
FILE *fp;
fp=fopen(“Sample.C,” “r”);
```

fp is a pointer variables, which contains the address of the structure FILE which has been defined in the header file “stdio.h”. fopen() will oepn a file “sample.c” in ‘read’ mode, which tells the C compiler that we would be reading the contents of the file. Here, “r” is a string and not a character.

When fopen() is used to open a file then, it searches on the disk the file to be opened. If the file is present, it loads the file from the disk into memory. But if the file is absent, fopen() returns a NULL. It also sets up a character pointer which points to the first character of the chunk of memory where the file has been loaded.

Reading A file:

To read the file’s contents from memory there exists a function called fgetc(). This is used as:

```
s=fgetc(fp);
```

fgetc() reads the character from current pointer position, advances the pointer position so that it now points to the next character, and returns the character that is read, which we collected in the variable s. This fgetc() is used within an indefinite while loop, for end of file. End of file is signified by a special character, whose ascii value is 26.

While reading from the file, when fgetc() encounters this Ascii special character, instead of returning the characters that it has read, it returns the macro EOF. The EOF macro has been defined in the file “stdio.h”.

When we finished reading from the file, there is need to close it.

This is done using the function fclose() through the following statement:

```
fclose(fp);
```

This command deactivates the file and hence it can no longer be accessed using getc().

'C' provides many different file opening modes which are as follows:

1. "r" Open the file for reading only.
2. "w" Open the file for writing only.
3. "a" Open the file for appending (or adding) data to it.
4. "r+" The existing file is opened to the beginning for both reading and writing.
5. "w+" Same as "w" except both for reading and writing.
6. "a+" Same as "a" except both for reading and writing.

INTEXT QUESTIONS

1. Distinguish briefly between input and output.
2. What is the major difference between the end of a string and the end of a file ?
3. What is EOF, and what value does it usually have ?
4. What must be done to a file before it can be used ?
5. How does the fopen function work?

19.4 CREATING A DATA FILE

A data file must be created before it can be processed. A stream-oriented data file can be created in two ways. The first one is to create the file directly using a text editor or word processor. The second one is to write a program that generates information in a computer and then writes it out to the data file. We create unformatted data file with the second method.

Let us consider following program in C.

```
#include "stdio.h"

main()

{ FILE *fs,
```

```
char ch;

fs=fopen(“sample1.c”, “w”);

do

putc (toupper (ch=getchar (,), fs);

while (ch!= '\h');

fclose (fs);
```

This program starts with definning the stream pointer fs, indicating the beginning of the data–file buffer area. A new data file called sample1.c is then opened for writing only. Next a do–while loop reads a series of characters from the keyboard and writes their uppercase equivalents to the data file. The putc function is used to write each character to the data file. Notice that putc requires specification of the stream pointer fs as an argument.

The loop continues as long as a newline character (\n) is not entered from the keyboard. Once a newline character is detected, loop comes to an end and data file is closed. After executing the program, the data file sample1.C will contain an uppercase equivalent to the line of text entered into the computer from the keyboard. For example, if the orgional file contains the following text "param" is a good boy. Then the data file sample1.c will contain the following text.

```
PARAM IS A GOOD BOY.
```

19.5 PROCESSING A DATA FILE

To execute any program we are required to first enter the program, compile it, and then execute it. Instead of the program prompting for entering the source and target filenames it can be through command prompt in the form:

```
C> filecopy sample1.c sample2.c
```

where sample1.c is the source filename and sample2.c is target filename.

The second option is possible by passing the source filename and target filename to the function main(). Let us first consider an example:

```
#include “stdio.h>
main( int argc,char * argv [])
```

```
{
FILE *fs , *ft;
```

The arguments which are passed on to main() at the command prompt are called command line arguments. These are named as argc & argv. argv is an array of pointer to strings and argc is an int whose value is equal to the number of strings to which argv points.

When the program is executed, the strings on the command line are passed to main(). The string at the command line are stored in memory and address of the first string is stored in argv[0], address of the second string is stored in argv[1] and so on. The no. of strings given on the command line are stored in argc.

We can use fputc() in while loop as follows:

```
while(!feof(fs))
{
ch=fgetc(fs);
fputc(ch, ft);
}
```

Here, feof() is a macro which returns a 0 if end of file is not reached. When the end of file is reached feof() returns a non-zero value, ! makes it 0.

You can read or write strings of characters from and to files instead of single character by the help of fputs(). Let us consider an example.

```
#include “stdio.h”
main()
{
        FILE * fp;
        char str[80];
        fp=fopen(“sample.txt” , “w”) ;
        if ( fp = = NULL)
{
        puts(“cannot open file”);
        exit();
```

```
}
        printf("\n enter a few lines of text :\n");
        while(  strlen(gets(str))>0)
{
        fputs(str, fp);
 fputs("\n");
}
fclose(fp);
}
```

fputs () function writes the contents of the array to the disk . Since fputs() does not automatically add a new line character the end of the string, it must be explicitly done. Let us consider an example of reading strings from a disk file.

```
#include "stdio.h"
main()
{
        FILE *fp;
        char str[80];
fp=fopen("sample.txt,"r");
If (fp= = NULL)
{
        puts(" Cannot open file");
exit();
}
while(fgets(str,  79,fp)!=NULL)
printf("%s",s);
fclose(fp);
}
```

The function fputs() takes three arguments. The first is the address where the string is stored, second is the maximum length of the string. The third argument is the pointer to the structure FILE.

For formatted reading and writing of characters, strings, integers, floats, there exists two functions, fscanf and fprintf(). Let us con-

sider an example:

```
#include “stdio.h”
main()
{
FILE *fp;
char choice= ‘y’;
char name[20];
int age;
float basic;
fp=fopen (“Emp.dat”, “w”);
if(fp= =NULL)
{
puts(“Cannot open file”);
exit();
}
while (choice = = ‘y’)
{
printf(“\n” Enter name, age and basic salary \n”);
scanf(“\ %s%d %f,”  name, &age, & basic);
fprintf(fp,”%s %d %f \n”,name, &age, &basic );
printf(“Another details(y/n)”);
fflush(stdin);
ch= getche() ;
}
 fclose (fp);
}
```

fpritnf() , writes the values of three variables to the file. If we want to read the details which we had entered through fprintf() then we have to use fscanf() within the loop. Let us consider an example:

```
#include “stdio.h”
main()
```

```
{
        FILE* fp;
        char name (4ϕ);
        int age;
float basic;
fp=fopen("Emp.dat", "r");
if (fp = = NULL)
{
puts("Cannot open file");
exit ();
}
while (fscanf(fp, "%s %d %f", name, &age, &basic)!=EOF)
printf("\n %s %d %f", name,&age, &basic);
fclose(fp);
}
```

fscanf() function is used to read the data from the disk.

The program which we have used till now, used text modes in which the file is opened. But now we will discuss the binary mode also which are different from text mode in the following manner:

- Handling of newlines
- representation of end of file
- storage of numbers

The format is

e.g. fp=fopen("Sample.txt", "rb");

In case of binary modes, there is no special character present to mark the end of file. The binary mode files keep track of the end of file from the number of characters present in the directory entry of the file. The file that has been written in binary mode must be read back only in binary mode.

If large amount of numerical data is to be stored in a disk file, using text mode may turn out to be inefficient. The solution is to open the file in binary mode. Then, only each number would occupy same number of bytes on disk as it occupies in memory.

RECORD I/O IN FILES

If you want to write a combination of dissimilar data types, you have to use structures let us consider an example

```
        #include “stdio.h”
main()
{
        FILE *fp;
        char ch= ‘y’;
        struct emp
{
char name[40];
int age;
float basic;
};
struct emp el;
fp= fopen( “Emp.dat:”,”w”);
if(fp = = NULL)
{
        puts(“Cannot oepn file”);
        exit();
}
        while(ch= = ‘y’ )
{
        printf(“\n Enter name, age, and basic salary”);
        scanf(     “%s%d%f”, e.name, &e.age, &e.basic);
        fprintf(fp, “%s%d%f\n, e.name, e.age, e.basic);
printf(“want to add more record (y/n)”);
fflush(stdin);
ch=getch();
}
fclose(fp);
}
```

In this program, the variable(basic) which is number would occupy more number of bytes, since the file has been opened in text mode. This is because when the file is opened in text mode, each number is stored as a character string. Let us consider the following examples which is receiving records from keyboard and writing them to a file in binary mode.

```
# include “stdio.h”
main()
{
FILE *fp;
char ch=’y’;
struct emp
{
char name[40];
int age;
float basic;
};
struct  emp e;
fp=fopen(“emp.dat”, “wb”);
if (fp= = NULL)
{
puts(“Cannot open file”);
exit();
}
while(ch= =’y’)
{
printf(“\n Enter name, age and basic salary”);
scanf(“%s%d%f”, e.name, &e.age,&e.basic);
fwrite (&e, sizeof (e), 1,fp);
printf(“want to continue (y/n)”);
fflush(stdi^);
ch= getch();
}
```

```
fclose(fp);
}
```

The first argument is the address of the structure to be write to the disk. The second argument is the size of the structure in bytes. sizeof() operator gives the size of the variable in bytes. The third argument is the number of such structures that we want to write at one time. The last argument is the pointer to the file we want to write to. Let us consider an example of reading records from binary file.

```
#include "stdio.h"
main()
{
    FILE *fp;
    struct emp
    {
    char name [40];
int age;
float basic;
};
struct emp e;
    fp =fopen ("Emp. Dat", "rb");
if(fp= = NULL)
{
puts("Cannot open file");
exit();
}
while (fread(&e, sizeof (e),1,fp) = =1)
printf ("\n %s%d%f," e.name, e.age, e.basic);
fclose (fp);
}
```

The function fread() returns the number of records read. If end of file reaches, it returns a 0.

INTEXT QUESTIONS

6. Distinguish between printf and fprintf ?

7. What is append mode, and what letter is used to specify if?

8. How does write mode differ from append mode when an existing file is being written to ?

Rewind() & fseek()

The rewind() function places the pointer to the beginning of the file, irrespective of where it is present right now. The syntax is

rewind(fp);

Where fp is the file pointer.

Pointer movement is of utmost importance since fread() always reads that record where the pointer is currently placed. Similarly , fwrite() always writes the record where the pointer is currently placed.

The fseek() function move the pointer from one record to another.

The syntax is

fseek(fp,no-of-bytes, position);

Here, fp is the file pointer, no-of-bytes is the integer number that indicates how many bytes you want to move & position is the position of the pointer from where you want to move e.g. it may be current position, beginning of file position, & End of file position.

e.g. to move the pointer to the previous record from its current position, the function is

fseek(fp, -recsize, SEEK-CUR);

Here no-of-bytes is stored in variable recsize which itself is a record size, SEEK_CUR is a macro defined in “stdio.h”. Similarly SEEK_END, SEEK_SET can be used for end of file and beginning of file respectively

If we wish to know where the pointer is positioned right now, we can use the function ftell(). It returns this position as a long int which is an offset from the beginning of the file. The value returned by ftell()

can be used in subsequent calls to fseek(). The sample call to ftell() is show below:

```
p=ftell(fp);
```

DETECTING ERRORS

To detect any error that might have occurred during a read/write operation on a file the standard library function ferror() can be used. e.g.

```
        while(!feof(fp))
        {
           ch=fgetc(fp);
        if (ferror())
        {
        printf("Error in reading file");
        break;
        }
        - - - -
- - - -
```

If error occurs, ferror() returns a non-zero value & the if block gets executed.

To redirect the output of a program, from the screen to a file let us see an example:

```
#include "stdio.h"
main()
{
char ch;
while((ch=getc(stdin))!=EOF)
putc(ch,stdout);
}
```

On compiling this program we would get an executable file sample exe. Normally when we execute this file, the putc() function will cause what ever we type to be printed on screen, until you don't type ctrl+z. But through DOS this can be done using redirection.

c>sample.exe>sample1.txt.

Here, the output of sample.exe is redirecting to the file sample1.txt. So, '>' symbol is used for redirection (output). Similarly you can use '<' for input redirection e.g c>smple.exe< Newsample.txt

19.6 WHAT YOU HAVE LEARNT

In this lesson you have learnt about different file opening modes and before that commands to open and close a file. You are now very well know about file pointer. You can write or read to or from the file with the help of functions fread (and fwrite). You can also change the position of file pointer with the help of command fseek(). The function ftell() tells you the current position of file pointer.

You can also use symbol '>' and '<' for output redirection and input redirection respectively.

19.7 TERMINAL QUESTIONS

1. In which file the macro FILES are defined?
2. If a file is opened for reading, it is necessary that the file must exist. Is it true or false?
3. Write a program to count the number of words in a given text file.
4. While using the statement,

   fp=fopen("my.c", "r");

   What happens if my.c exists on the disk?
5. Write a program which will append or add the records in a student file, and also modifies the details of any student.
6. What role does the fseek function play, and how many arguments does it have?
7. If a C library does not contain a rewind function, how can it always be implemented?

19.8 KEY TO INTEXT QUESTIONS

1. Information that enters computer from the outside is called input, whereas information produced by the computer and sent to outside world is known as output.

2. The end of a string is always indicated by a null character, whereas the manner in which the end of file is indicated depends on the storage device and the computer.

3. EOF is a constant returned by many I/O functions to indicate that the end of an input file has been reached. Its value on most computers is –1.

4. It must be opened by the program.

5. The fopen function has two parameters. The first is a string specifying the file name. The second is a string that specifies the mode in which the file is to be opened. The function returns a FILE pointer associated with the opened file.

6. The print ϕ function can send output only to the standard output file, whereas fprintf can send its output to any opened output file.

7. Append mode is used to add characters to the end of a file that already exists. It is specified by using the mode string “a” as the second parameter to fopen.

8. When write mode is used, the former contents of the file are erased. With append mode, the contents of the file are returned and new information is added to the end of the file.

20

C PREPROCESSOR

20.1 INTRODUCTION

We have already seen many features provided by C language. Yet another unique feature of the C language is the preprocessor. The C preprocessor provides several tools that are not available in other high–level languages. The programmer can use these tools to make his program more efficient in all respect.

20.2 OBJECTIVES

After going through this lesson you will be able to

- explain preprocessor working
- explain the # define Directive
- define constants
- explain macros
- write the various directions such as # undef, #include, #fdef, #ifdef, #ifndef, # else, #if

20.3 HOW THE PREPROCESSOR WORKS

When you issue the command to compile a C program, the program is run automatically through the preprocessor. The preprocessor is

a program that modifies the C source program according to directives supplied in the program. An original source program usually is stored in a file. The preprocessor does not modify this program file, but creates a new file that contains the processed version of the program. This new file is then submitted to the compiler. Some compilers enable the programmer to run only the preprocessor on the source program and to view the results of the preprocessor stage. All preprocessor directives begin with the number or sharp sign (#). They must start in the first column, and no space is required between the number sign and the directive. The directive is terminated not by a semicolon, but by the end of the line on which it appears.

The C preprocessor is a collection of special statements called directives, that are executed at the beginning of the compilation process. The #include and # define statements are preprocessor directives.

20.4 THE # DEFINE DIRECTIVE

The #define directive is used to define a symbol to the preprocessor and assign it a value. The symbol is meaningful to the preprocessor only in the lines of code following the definition. For example, if the directive #define NULL 0 is included in the program, then in all lines following the definition, the symbol NULL is replaced by the symbol. If the symbol NULL is written in the program before the definition is encountered, however, it is not replaced.

The # define directive is followed by one or more spaces or tabs and the symbol to be defined. The syntax of a preprocessor symbol is the same as that for a C variable or function name. It cannot be a C keyword or a variable name used in the program; if it is so a syntax error is detected by the compiler. For example, suppose a program contains the directive

```
#define dumb 52
```

which in turn is followed by the declaration

```
int dumb;
```

This would be translated by the preprocessor into

```
int 52;
```

which would be rejected by the compiler. The preprocessor does not check for normal C syntax, except for identifying quotation marks. It merely substitutes symbols where it finds them. The sym-

bol being defined is followed by one or more spaces or tabs and a value for the symbol. The value can be omitted, in which case the symbol is defined but not given a value. If this symbol is used later in the program, it is deleted without being replaced with anything.

If a # define directive does not fit on a single line, it can be continued on subsequent lines. All lines of the directives except the last must end with a backslash(\ ) character. A directive can be split only at a point where a space is legal.

20.5 CONSTANTS

A common use for defined symbols is the implementation of named constants. The following are examples of constants in C:

23, 'a', "hello"

Any of these can be assigned to a defined preprocessor symbol:

```
#define INTEGER 23
#define     CHARACTER 'a'
```

A defined symbol can specify only a complete constant. For example, if a program contains the definition

```
#define NULL 0
```

the number 120 cannot be represented by 12 NULL. A defined symbol can be recognized only if it is delimited by white space, punctuation or operators.

20.6 MACROS

When a constant is associated with a defined symbol, the characters making up the constants are assigned , not the numeric value of the constant. The definition

```
#define EOF-1
```

really assigns the string "-1" to EOF. The preprocessor replaces the symbol EOF by the characters "-1" without any consideration of the meaning of the two characters.

When a preprocessor symbol is defined as a segment of text, it is more generally called a macro. Macros are very useful in making C code more readable and compact. For example, some programmers include the following definitions in all their programs:

```
# define and &&
# define or ¦¦ ||
```

These definitions enable the programmer to write more readable code, such as the following:

```
If (a<b or c>d and e<f)
```

A comment can be included at the end of a macro or constant definition:

```
# define same 1 /* Nothing */
```

A comment can be inserted at any point in a C program where white space is permitted without adverse effect to the program.

Macros also can be used as abbreviation for lengthy and frequently used statements. The one restriction on macros is that, even though their definitions can occupy more than one line, they cannot include a newline. That is, when a macro is expanded all the resulting text is placed on the same line.

The preprocessor permits macros to have arguments, just as functions do; the difference is that the arguments are strings rather than values. Consider the following definition.

```
#define Decrement(x) if (x>0)x - = 1
```

when this macro is expanded, the string passed to the argument x replaces all occurrences of x in the symbol's value. That is, the statement

```
Decrement(a);
```

is expanded to

```
    if(a>0) a- = 1;
```

and the statement

```
Decrement(b);
```

becomes

```
    if (b>0) b - =1;
```

Notice that the macro definition itself does not include a semicolon. When the macro is invoked later, it is followed by an explicitly specified semicolon:

```
Decrement(a);
```

The argument names used in a macro are local to the macro. Thus there is no conflict between a macro's formal parameter names and identifiers used in the program itself. Any symbols used in the macro definition which are not argument name or names of other macros are left unchanged, and are assumed to be variable names or other C symbols.

The argument supplied to a macro can be any series of characters.

The comma is the delimiter that separates arguments in a macro.

A macro can be defined in terms of another macro, as in the following example of nested macros:

```
#defined control "%d\n"
#define printint(x) printf(control,x)
#define test(x) if (x>0) printint(x)
```

If a program contains the statements

```
Test(w);
```

Where w is an integer variable, the statement goes through the following conversion steps:

i) `if(w>0) printint(w);`

ii) `if (w>0) printf(CONTROL,w);`

iii) `if(w>0) printf ("%d\n,"w);`

INTEXT QUESTIONS

1. What, in general terms, is the role played by the C preprocessor ?
2. Which symbol always precedes preprocessor directives ?
3. Where on the line must all preprocessor directives begin ?
4. Where can a preprocessor directive be written ?
5. Which symbol is used to separate the arguments of a macro ?

20.7 USE OF DIRECTIVES: # undef, # include

(a) The # undef Directive:-

If a preprocessor symbol has already been defined, it must be undefined before being redefined. This is accomplished by the #undef directive, which specifies the name of the symbol to be undefined. It is not necessary to perform the readefinition at the beginning of a program. A symbol can be redefined in the middle of a program so that it has one value in the first part and another value at the end.

A symbol need not be redefined after it is undefined.

(b) The #include Directive:-

Often a programmer accumulates a collection of useful constants and macro definitions that are used in almost every program. It is desirable to be able to store these definitions in a file that can be inserted automatically into every program. This task is performed by the #include directive.

A file is inserted at a point where the #include directive is encountered. Such files are called header files, and by convention their names end with the character.h (as in stdio.h).

20.8 CONDITIONAL COMPILATION: #ifdef, #ifndef and #endif

Removing statements by hand would be quite tedious and could also lead to error. For this reason, the preprocessor provides directives for selectively removing section of code. This process is known as conditional compilation.

#define RECORD–FILE

If the # ifdef directive tests whether a particular symbol has been defined before the #ifdef is encountered. It does not matter what value has been assigned to the symbol. In fact, a symbol can be defined to the preprocessor without a value.

If the #ifdef directive is encountered after this definition it produce a true result. If, however, the directive #undef RECORD–FILE is encountered before the directive #ifdef RECORD–FILE then the preprocessor considers the symbol RECORD–FILE to be undefined and the #ifdef directive returns a false value.

If an #ifdef returns a true value, all the lines between the #ifdef and the corresponding #endif directive are left in the program. If those lines contains preprocessor directives, the directives are processed. In this way, conditional compilation directives can be nested. If the #ifdef evaluates as false, the associated lines are ignored, including any preprocessor directives that are included.

The statements need not all be grouped in one place. The #ifdef and #fendif directives can be used as many times as required. The preprocessor also provides the directive #ifndef, which produces a true result if a symbol is not defined. This makes it possible to use a single symbol to switch between two versions.

Conditional compilation can be used to select preprocessor directives as well as C code. For example suppose a header file is included in a program. and a certain preprocessor symbol (say, FLAG) may or may not be defined in that header file. If the programmer wants FLAG never to be defined, then the #include directive can be followed by

```
#ifdef FLAG
#undef FLAG
#endif
```

This ensures that, even if the symbol is defined in the header file, its definition is removed. It is not sufficient merely to work on

```
#undef FLAG
```

because if FLAG is not defined, the directive is erroneous. If FLAG should always be defined, then we would write

```
#ifndef FLAG
 #define FLAG
 #endif
```

We could, of course, give FLAG a value. We cannot simply write

```
#define FLAG
```

since if FLAG is already defined, an error probably will result.

20.9 THE #ELSE DIRECTIVE

This directive functions in much the same way as the else clause of an if statement.

All lines between an #ifdef or an #ifndef directive and the corresponding # else clause are included if the #ifdef or #ifndef is true. Otherwise, the lines between the #else and the corresponding #endif are included.

The # else directive and the lines that follow it can be omitted, but never the #endif directive. No other text can be included on the same line as the #else or #endif directive.

20.10 THE # IF DIRECTIVE

The #if directive tests an expression. This expression can be of any form used in a C program, with virtually any operators, except that it can include only integer constant values No variables, or function calls are permitted, nor are floating point, character or string constants.

The #if directive is true, if the expression evaluates to true (non–zero). Any undefined preprocessor symbol used in the #if expression is treated as if it has the value ϕ. Using a symbol that is defined with no value does not work with all preprocessors, and an attempt to do so might result in an error.

INTEXT QUESTIONS

6. What is the characteristics of the #else and the #endif directives?

7. How is the #ifndef directive used?

8. State the reason for using the #ifdef directive.

20.11 WHAT YOU HAVE LEARNT

In this lesson, you have learnt about preprocessors. The preprocessor is a program which modifies the C source program according to instructions provided. Remember that all preprocessor directives begin with the symbol #. There are different preprocessor directives. You also learnt about Macros. This lesson also discussed about conditional compilation.

20.12 TERMINAL QUESTIONS

1. How can a #define directive be continued to a new line ?

2. Where can a preprocessor directive be split ?

3. What is a macro, and what is it used for ?
4. What can a macro argument consist of ?
5. What role is played by the #undef directive?
6. How is the # include directive used ?
7. What is conditional compilation ?
8. How does the #if directive operate ?

20.13 KEY TO INTEXT QUESTIONS

1. It provides for the implementation of macros and conditional compilation.
2. The # sign.
3. The first column.
4. Anywhere in a program, so long as it starts in the first column of a line and is not on the same line as another directive or C statement.
5. The comma.
6. They only allow that text can be included on the lines they occupy.
7. It passes lines of code to the compiler only if the specificed preprocessor symbol is not defined.
8. The #ifdef directive is used in order to pass certain lines of code of the compiter, only if the specified preprocessor symbol is defined before the #ifdef directive is encounted.

21

SETTING UP A COMPUTER BUSINESS CENTRE

21.1 INTRODUCTION

After completing this course there are two options available to you:-

First, you may find a job as a DTP Operator, Data Processing Assistant, Designing Assistant, Computer Operator, Programming Asstt. etc. Such jobs are available in government organizations as well as private companies. The second option is to set up your own business. By setting up your own DTP unit, you become your own employer and provide jobs for others.

The selection of the option lies with you : Either go for a job or set up an enterprise. Both the options have their own plus and minus points : Salaried jobs provide fixed income without investing any money. Business on the other hand involves investing money and taking risk. However, the income in the business is comparatively more than in wage jobs. Further, income generally commensurate with the amount of money invested and the amount of hard-work you put in.

You will be wondering how will you manage to set up your own business without any experience and sufficient money. Once you decide to set up your own Computer Business Centre (CBC), it is not difficult to do so. Guidance and help is available at different sources.

In this lesson you must be thinking about the utility of this course in the terms of earning. This is quite natural.

21.2 OBJECTIVES

After going through this lesson, you would be able to

- identify major inputs to a business unit
- identify various components of capital required for a CBC
- estimate the profits of a CBC
- learn various sources of help and finance for setting up a CBC
- adopt good business practices required for an entrepreneur

21.3 INPUTS TO A BUSINESS

The important inputs to a business are :-

- Will Power
- Place
- Capital
- Finance

Will Power

This is the first and the foremost input required for setting up a business unit. You must make up your mind well before starting any business, as business involves risk. A person who wants to set up a business either in trade or industry is called an entrepreneur. An entrepreneur makes best use of the available machinery, manpower and money for producing goods or service. A unit set up by an entrepreneur is called an enterprise. CBC is also an enterprise which provides services to others.

Place

Place for a CBC unit involves generally a room of the minimum size of 20 sq. metre. The space should be sufficient for installing computer and accessories, office space and space for visitors. The space may be your own or rented. For a CBC unit, it is better if the place is near to a commercial complex or government offices. The office space must have electricity and telephone connection.

Capital

Once you decide on the place of your business, you require money. Money invested in the business is called **Capital**. Two types of capital are required: Capital is **fixed** when it involves procuring land, building, equipment, machinery, vehicles, furniture and fixtures, etc. **Working capital** is required for meeting day to day expenses of an enterprise. These include purchase of raw material, stationery, payment of bills like water, electricity, telephone, wages of staff/labourer and traveling/conveyance charges.

Fixed capital

Fixed capital generally remains constant over a longer period and is required only in the beginning stage. On the other hand, working capital is required frequently. It is implied that after sometime money starts coming out of the business and thus working capital is met out of the turnover.

For a basic CBC the fixed capital involves the following equipement and furniture.

Item	*Quantity*
Computer (with VDU, CD drive, UPS and Keyboard)	1
Scanner (optional)	1
Laser Printer (optional)	1
Inkjet Printer / DM - Printer	1
CD Writer	1
Computer Table	1
Computer Chair	1
Office Table	1
Chairs for staff and visitors	4
Internet Centre	1

Presently this minimum requirement may be met within about **One lakh** rupees.

(For datails see model calculation given below)

Working Capital

Working capital is required for purchase of raw material (like paper, cartridges, floppies, CDs, etc.), for making payments to staff, electricity and telephone bills, conveyance and travel, etc. In DTP business the returns are quick and generally turnover starts within a month. It is estimated that the working capital of Rs. 12,500 is sufficient for a month.

In the table below, we have given a model for working out the fixed capital, working capital and expected profit. Please note that the estimate below is only an illustration. Keeping in view your specific requirement, you have to make your own budget.

Model Calculation of Investment for One DTP Unit

A.	**Fixed Capital**	
(1)	Land and Building (one room)	rented
(2)	Equipment (1 Computer with VDU, UPS, Keyboard and CD drive)	40,000
(3)	Scanner	6,000
(4)	Laser Printer (Basic Model)	15,000
(5)	Inkjet Printer / Dot Matrix Printer	6,000
(6)	Software (Licensed)	15,000
(7)	Furniture	16,000
(8)	Miscellaneous	2,000
	Total	1,00,000

B. Working Capital (for one month)		
(1) Rent of room		3,000
(2) Electricity, water bill, etc.		1,000
(3) Telephone/Internet charge		2000
(4) Wages of one Assistant/Helper		3000
(5) Raw Material (Paper, floppy, cartridge, etc.)		3000
(6) Conveyance/Travel		2000
(7) Miscellaneous Expenses		500
(8) Contingences		500
	Total	15,000
Total Investment (Fixed Capital + Working Capital)		**1,15,000**
C. Calculation of Income over Expenditure		
Expected Turnover (for one month) [75 pages per day Rs. 15/- Per page]		30,000
Less Expenses		
Repayment of loan		3,000
Rent, Wages, Telephone etc.		15,000
Depreciation of the equipment (@ 20% Industry Norms)		850
Provision for Taxe/Service charge		1000
Provision for bad debt		150
	Total	20,000
Surplus of Income over expenditure (profit)		10,000

Thus you can expect a profit of Rs. 10,000 per month by setting up one unit. If you set up more units, the profit will be proportionately higher as the overhead expenses like rent, electricity, telephone etc. will be remain the same.

As already said all this is only an estimate. In actual conditions the profit may vary, it may increase or decrease. Much of it depends on your hard-work, wisdom and intelligence.

21.4 FINANCE

Having moved by the **profit motive**, you may now be thinking of establishing your own for DTP unit. For this, you will require money. If you are lucky to have the money

arranged from your own/family sources, it is fine. Otherwise, the following sources may be tapped.

Banks

Banks, whether national, private or cooperative, provide loans under different schemes. The rate of interest and terms generally vary from bank to bank. Interest rates are decreasing steadily as well as the process is becoming easier.

Government Department

Various departments and ministries of the Central/State Government have different schemes under which financial assistance loans are available. Prime Minister Rozgar Yojana (PMRY) is one such scheme. Some of the departments which provide financial assistance are:

Department of Rural Development

Department of Small Scale Industries

Department of Tribal Development

Department of Minorities Development

Department of Women Development

Small Scale Industries Institute (Govt. of India) provides valuable help in preparing project report and procuring machinery and raw material. This institute has its head-quarter in Okhla (New Delhi) and branch offices in all state capitals and major cities. Addresses of this institute and other government departments may be obtained from the nearest Employment Exchange or the Office of the District Collector.

Finance is generally available for Fixed Capital, i.e., one time only. Generally up to 80% finance is available and rest you have to arrange from your resources. This difference is called the **Seed Money.** In our model calculation, we had estimated that fixed capital of approximately Rs. 1,00,000 is required for setting up a CBC. Out of this, loan upto Rs. 80,000 will be financed by the banks/institutions and Rs. 20,000 will have to be arranged by you.

For working capital banks may grant you **Overdraft** facility. Thus, by investing about 35,000/- (Rs. 20,000 + Rs. 15,000) you get a profit of over Rs. 10,000/- per month which is reasonably good. However, profit can be increased by increasing business which will be the result of improved efforts and hard work put in the direction.

21.5 ACTIVITIES FOR COMPUTER BUSINESS CENTRE

The main activities which a computer Business center may carry on are

Word Processing: You may do Word Processing jobs like preparing personal/ business letters of individuals, small traders and business firms. Bio-data/resume for the purpose of jobs and marriages may also be prepared. Some documents like Income Tax Return, Application for PAN, Passport Forms, Application form for enrolling as voter may be stored as a **Template** and used frequently for your customers.

You may also contact the Free-lance writers, Journalists and Researchers and prepare their articles and reports. Similarly Projects and Dissertations of the students may be documented.

Data Processing: Individuals and small business firms generally do not have their own computer facilities for storing their data. You may contact them and maintain this data for them. Data may relate to:-

- Personal Address Books
- List of Suppliers
- List of Vendors
- Inventory
- Record of Investments
- Record of Taxes like Sales Tax, Excise and Custom Duty

Accounting: If you have an aptitude for the accounting work, you may like to maintain the accounts of individuals professionals like Doctors, Lawyers, Free-lance Writers, Artists etc and small business firms.

For this you need, is some basic knowledge of Accounting and a Software package. These days many accounting software package are available in the market. The commonly used ones are MUNIM and TALLY

Diversified Activities: At a CBC, the following activities may also be undertaken

Fax facilities

Fax facilities may be made available to you customers by installing a separate fax machine or through a fax-card installed in the computer. A fax machine may cost you about Rs 10,000-20,000 depending on the model you choose. Fax machine with a Plain-Paper mode is better than a Thermal-paper one.

The fax card costs only a few hundred rupees and may be used in conjunction with your existing computer, scanner and printer.

Internet Cafe

You may provide internet surfing service to your customers. This service requires

installing a few move computers and proportionate space for making cubicles/ cabins. Internet customer may use this service for downloading information, e-mail, internet telephony and retrieving small queries like getting examination results or for finding the status of the train reservation.

Many more diversifications like Call Centre Operation, Medical Transcription, Computerised Design Centre, On-line Trading Centre (For shares and Securities etc.) may be thought of by your own imagination and growing experience. Always keep yourself up-dated with the new technological advances. News papers, Professional Journals, Internet, Professional Conferences/Associations are a good source of knowledge. Make the best use of them.

21.6 GOOD BUSINESS PRACTICES

Other than the financial aspects of a business, an entrepreneur should also take care of good business practices. These practices not only pay in the long run but also create harmony and cordial environment in the society. These practices are:

(1) Be polite to the customers. Always greet and receive them properly. Remember that **Service with a Smile** does not cost but brings reputation and goodwill to your business.

(2) Be punctual and adhere to the Delivery Schedule. If the job of a customer is not ready by scheduled time, inform her/him in advance and apologise.

(3) Be honest in dealings. Charge reasonably and honestly from the customers. Make prompt payments to your suppliers.

(4) Pay your dues and taxes in time. It is our earnest duty to pay the taxes honestly. Taxes are the revenue for the government. This money is used for the welfare of the society.

(5) Pay reasonably to your staff. Payment to them should commensurate with their qualification, experience and the output they produce. Besides you have to comply by the **Minimum Wages Act** in this regard.

(6) Try to explore new business opportunities. Diversify your business.

Please note that scope and opportunities in any business are indefinite and so with the CBC, only sky is the limit.

Wish you a happy business.