DATABASICS

Chapter 1

1.1 OVERVIEW

Data: *Data* is a collection of facts, figures and statistics – related to an object, that can be processed to produce a meaningful information. In an organization, data is an asset that enables the managers to perform an effective and successful operation of management. It gives view of past activities (rise and fall) and enables to make better decisions for future. It is also useful for generating some useful reports, graphs, statistics etc.

Information: The manipulated and processed data is called *information* e.g., the percentage of students results. It refers to all the facts, figures or statistics that are precisely meaningful to the people. So by definition, it is an output of a certain process.

Operations: Manipulation of data (after capturing from different sources) to achieve the required objectives and results is called operation. For this purpose, a software (program) is used to process raw data which is converted to meaningful information. Thus, effectively, a series of actions/operations are performed on raw data to achieve some output or result.

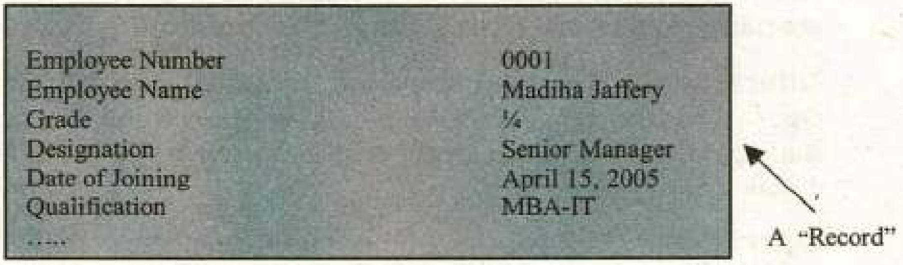
These are categorized into three basic activities :

- **Data Capturing:** Data must be recorded or captured in some form before it can be processed. Data may first be recorded on source documents or given directly through input devices.
- **Data Manipulation:** The following operations may then be performed on the gathered data.
  - Classifying: Organizing data into classes /groups. Items may be assigned predetermined codes, they can be numeric, alphabetic or alphanumeric.
  - Calculations: Arithmetic manipulation of the data.
  - Sorting: Data is arranged in logical sequence (numerically or alphabetically).
  - Summarizing: Masses of data are reduced to a more concise and usable form.

- **Managing the Output Results:** Once the data is captured and manipulated, it may be :
  - ❖ Storing and Retrieval : Data is retained for future reference. Accessing / fetching the stored data and/or information is the *Retrieve* activity.
  - ❖ Communication and Reproduction: Data may be transferred from one location or operation to another, for further processing. It is sometimes necessary to copy or to make duplicate of data. This activity is called *Reproduction.*

1.2 TRADITIONAL FILE SYSTEM

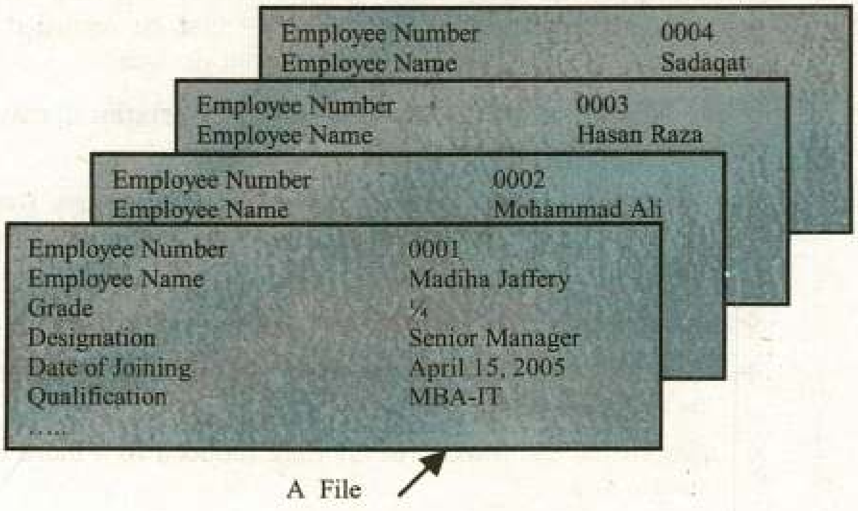
Record: A collection of related fields (facts about something) treated as a single unit is called a record. Let us assume an employee's biographic information in a bank.



Employee Number	0001
Employee Name	Madiha Jaffery
Grade	¼
Designation	Senior Manager
Date of Joining	April 15, 2005
Qualification	MBA-IT
.....	

As it belongs to one employee of the bank, so it is an individual employee's *record* (of biographic information).

File: A collection of related records treated as a single unit is called a file or a data set. If we collect records (as shown below) of all the employees, it becomes a *file* (bio-information) of all the employees of the bank .



Employee Number	0004
Employee Name	Sadaqat

Employee Number	0003
Employee Name	Hasan Raza

Employee Number	0002
Employee Name	Mohammad Ali

Employee Number	0001
Employee Name	Madiha Jaffery
Grade	¼
Designation	Senior Manager
Date of Joining	April 15, 2005
Qualification	MBA-IT
.....	

Files are categorized according to different criteria as discussed below :

File Types

- Usage point of view

❖ **Master File :** These are the latest updated files which never become empty, ever since they are created. They maintain information that remains constant over a long period of time. Whenever the information changes in files/records, it is updated. Methods of updating are adding, deleting or editing records in a file.

❖ **Transaction File:** These are those files in which data prior to the stage of processing is recorded. It may be temporary file, retained till the master file is updated. It may also be used to maintain a permanent record of transaction data.

❖ **Backup File:** These are again permanent files and their purpose is the protection of vital files/data of an organization by creating them using some specific software utilities.

- Functional point of view

For this purpose, the files are given appropriate names, consisting of two parts i.e., *file name* and *file extension*, having a dot "." in between. Normally, the extension is given by the software being used at the time of initial *save*. These files are summarized as under:

❖ **Program files :** These files contain the software instructions i.e. source program files and executable files. The source program files may have the extension as *.com* and the executable files as *.exe.*

❖ **Data files :** These are the files that contain data and are created by the software being used. A few of them are given as under :

Software	**File Types**	
Word Processor	**.doc, .rtf**	(document)
Spread Sheet	**.xls** and **.wks**	(worksheet)
Data base	**.dat**, **.dbf** and **.mdb**	(data files)

* Some other types are as given below:

ASCII/Text files	....	**.txt**
Image files	....	**.tif , .jpg, .eps, .gif, .bmp**
Audio files	....	**.wav, .mid**
Video files	....	**.avi , .mpg**

File Organization (Storage point of view)

* **Sequential Files:** As the name refers, these files are stored or created on the storage media in the order the records are entered i.e., one after another in the sequence. They require comparatively more processing time.

* **Direct or Random Files:** These files reside on the storage media according to the address which is calculated against the value of the key field of the record. Some times, the same address is calculated, which leads to the concept of synonym.

* **Indexed Sequential:** The key field of the records (in a file)are stored separately along with the address of each record. These files can be processed sequentially as well as randomly. They require relatively more space on the storage media but the processing is as fast as random/direct files.

1.3 DATABASES

A *database* is a collection of logically related data sets or files. Normally, these files/datasets are of different nature, used for specific purposes. These may be organized in various ways to meet various processing and retrieval requirements of the organizations or users. For example;

A bank may have separate files for its clients i.e.,

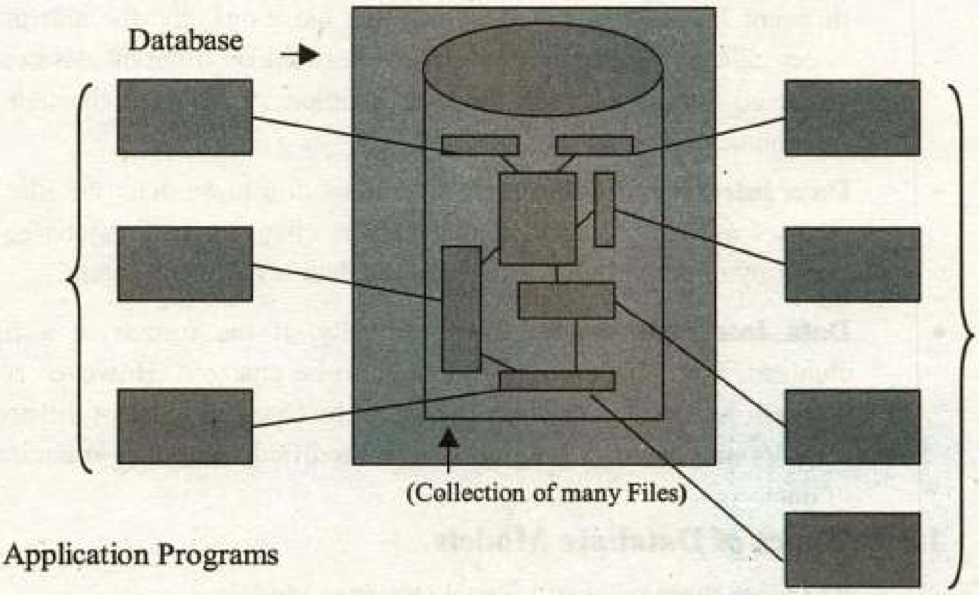
- Savings A/C
- Automobile loan
- Personal loan
- Clients biographic information etc.

The bank's clients/customer database would include records from each of these files. Using a series of programs, data for any client may be added, retrieved or updated depending upon the activity at a particular time.

It is a computerized system whose overall purpose is to maintain information and to make that information available at any time.

A data base system is just a computerized record keeping system. A data base itself can be regarded as a kind of electronic file cabinet, a warehouse or a repository for a collection of computerized data files. The user of the database normally has the following facilities to enjoy.

- Adding new, blank files to the database.
- Inserting new data into the existing files.
- Retrieving data from existing files.
- Updating data in existing files.
- Deleting data from existing files.
- Removing existing files, empty or otherwise from the database.



A Database System

The figure shown above is intended to illustrate the point that a database system involves four major components, namely.

➤ Data	*The Information*
➤ Hardware	*The physical* components i.e.,

- Secondary Storage
- I/O devices
- Device controllers
- I/O channels
- Processors
- Main memory

➤ Software	*User/system* software • Set of programs • Utilities
➤ Personnels	*The people* • Programmers/Analysts • End users • Database Administrators

1.3.1 Database Objectives

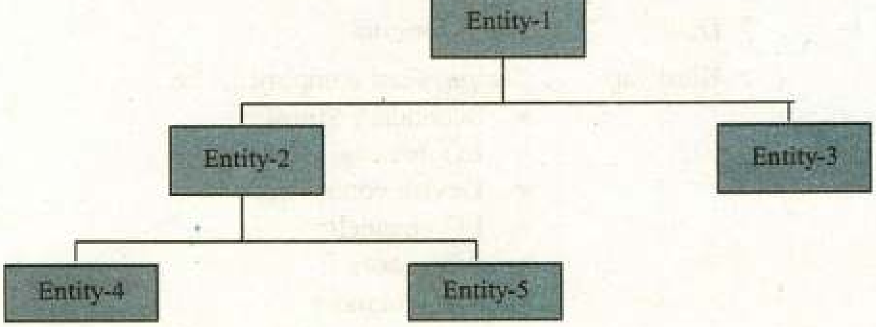
There are at least three main objectives for using the data base organization.

- **Data Integration:** In a database, information is coordinated from different files and operated on a single file. Logically, the information is centralized, physically data may be located on different devices i.e., scattered around over on different locations, connected through data communication links.
- **Data Integrity:** If a data item is contained in more than one file, then all files must be updated if that item is changed. In a database, only one copy of data is kept, therefore, the data is more consistent.
- **Data Interdependence:** Conventionally, if the format of a file is changed, then all the programs have to be changed. However, a data base allows the organization of data to be changed without the need to re-program. It allows programs to be modified without re-organization of data.

1.3.2 Types of Database Models.

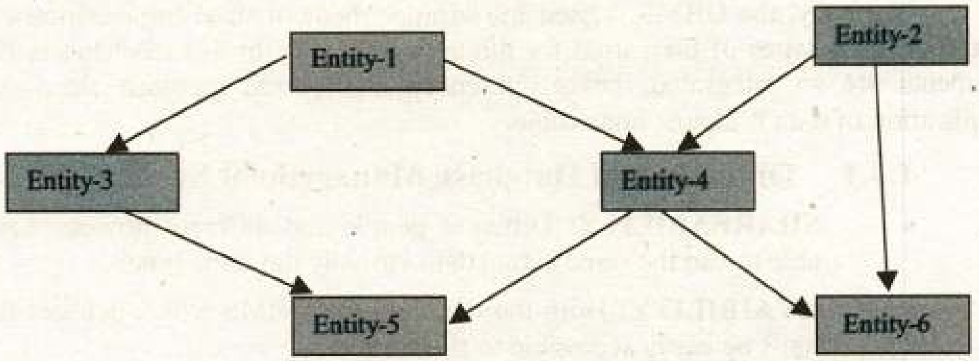
There are three types of Logical Database Models.

- **HIERARCHICAL Model:** This Model has the general shape or appearance of an Organizational Chart.



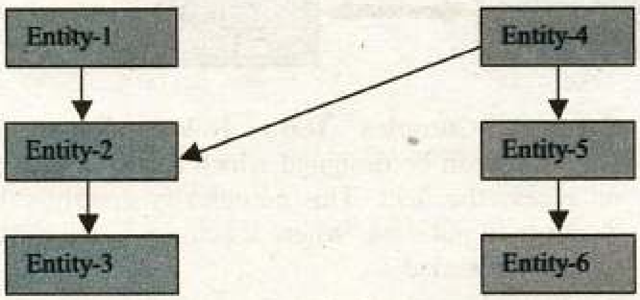
A node on the chart, representing a particular Entity is subordinate at the next highest level, just as on an organizational chart, an employee reports to only boss. This kind of structure is often referred to as an "Inverted Tree", with the top most referred to as the "Root".

- **NETWORK Model:** This is somewhat similar to that of the Hierarchical model but has one major difference.



Subordinate entities, depicted by arrows on the network diagram, may participate in as many subordinate relationships as desired. Therefore a much more complex diagrams may be used to represent the structure of the database. Networks provide more flexibility than a simple hierarchical system in the data relationships may be maintained.

- **RELATIONAL Model:** This system consists of a collection of simple files/Relations (Entities), each of which has no structural or physical connection such as those typically used in hierarchical or network systems.



The various entities possess the interrelationships as depicted by a network like diagram. But these relationships are based on the data content of the entities involved, not by pointer chains or other types of structural connection techniques.

1.4 DATABASE MANAGEMENT SYSTEM

The data management system (a collection of programs) which is used for storing and manipulating databases is called *database management system* (*DBMS*). It is an improvement over the traditional *file management systems*. It uses DBMS software (database manager) which controls the overall structure of a database and access to the data itself.

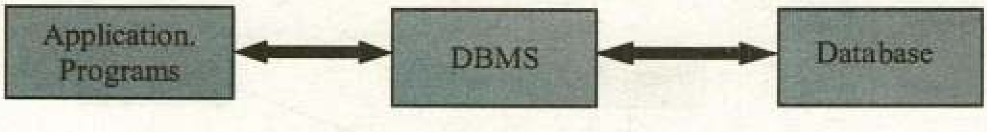
Normally, the DBMS is used for large or medium sized organizations, having heterogeneous types of files, used for different purposes. In this mechanism, the data elements are so integrated, cross referenced and shared amongst them that the duplication of data is almost impossible.

1.4.1 Objectives of Database Management System

- **SHAREABILITY:** Different people and different processes must be able to use the same actual data virtually the same time.
- **AVAIBILITY:** Both the data and the DBMS which delivers the data must be easily accessible to the users.
- **EVOLVABILITY:** The ability of the DBMS to change in response to growing user needs and advancing technology.
- **DATABASE INTEGRITY:** Since data is shared among multiple users, adequate integrity control measures must be maintained.

1.4.2 Advantages of Database Systems

- **Data Independence:** Application programs are not aware of the physical implementation of the data sets. The DBMS sits in between the application programs and the actual data sets that make up the database.



- **Support Complex Data Relationships:** Fairly complex data structures can be designed which allow various ways to logically view or access the data. This complexity greatly enhances the ability of a designer to put data "where it belongs", and provide a path to that data whenever needed.
- **Sophisticated Data Security Features:** Provide enhanced security mechanisms for access to data. Data base security mechanisms typically go much further in adding more extensive security features. If granted Read access to a file/table, the user may see each record in the file, and every data field it contains. Access intent of each application program (read, write, update, delete) can be specified

explicitly. An application program's view of data records may be controlled to the field level.

- **Data Base Backup / Recovery:** Provide sophisticated backup / recovery mechanism.

  Backup / Recovery capabilities often distinguish between true DBMS and a software package that only claims this facility. A DBMS has a logging or recording mechanism that captures information on changes to data within a data base. In case of data base recovery, a utility within the DBMS rebuilds it by using a backup copy of the data and log of changes as input.

- **Advanced Capabilities:** DBMS normally have advance access capability for on-line and ad-hoc reporting capabilities. However, the ability to provide data independence to create complex data structures, to provide security to data access, and to provide backup / recovery capability are the primary requirements of a Database Management Systems.

1.4.3 Disadvantages of Database Systems

Although, the DBMS are very powerful tools to do the job, but they are not more in use. It is because of some disadvantages :

- **Require additional System Overhead:** Additional overhead is required to access data, in case of doing some simple jobs; like reading and processing a tape file, which might take a little time and resources to do the job. If we have to do it on DBMS, it is like "requiring too much to do too little".
- **Additional Training required for Training of Staff:** Application programmers require a sort of precise training to code efficient programs that will run under a DBMS. There is a possibility that in-adequate training or experience of application development staff will lead the creation of grossly inefficient database calls. Quite often, the problem might not be found until the program reaches production *status.* The typical example is that of using proper and improper indexes for accessing the database.
- **Problems can multiply in selecting a wrong type of Dbase Environment:** A later change in structure, forced by changing requirements, can be costly in terms of conversion and testing of existing programs. Hierarchical data base systems are, in particular, more sensitive than network or relational systems towards this kind of problem, and implementing changes costs a great deal. On the other hand, doing these changes on relational data bases are fairly easy and less costly.

- **Data must be considered a corporate resource:** The data in a company's data base no longer belong to one organization alone. True, one organization normally has the primary responsibility for creating a data base. However, as data base systems mature, more companies or organizations can share the same data across applications.
- **A Need of a Dictionary:** In order to share data across application systems, or to simply given end users the ability to identify the location of information they need in order to do their jobs, the internal data contents of a company's data bases need to be documented in a consistent manner. For this purpose, they have to install a data dictionary system, which is another overhead on the DBMS.

1.4.4 Features of a DBMS

- **Data Dictionary:** Some databases have a data dictionary, a procedures document or disk file that stores the data definitions or a description of the structure of data used in the database. The data dictionary may monitor the data being entered to make sure it conforms to the data definition rules i.e., file names, field names, field sizes, data types etc. It may be used for data access authorization for the database users.
- **Utilities:** The DBMS utilities are the software programs that are used to maintain the database by manipulating the data, records and files. Some programs are also used for backup and recovery procedures of the databases.
- **Query Language:** Normally, *SQL* (Structured Query Language) is used for creating table structures, entering data into them and retrieving/updating the selected records, based on the particular criteria and format indicated, within the databases. Typically, the query is in the form of a sentence or English-like command i.e., SELECT, DELETE, CREATE, MODIFY, UPDATE and INSERT commands.
- **Report Generator:** A report generator is a program that is used to produce an on-screen or printed document from the database. The report format can be specified in advance i.e., row headings, column headings, page headers etc. Even the non-experts can create very useful and attractive reports by using this facility.
- **Access Security:** By using this facility, the database administrators can assign specific access privileges for the users of the databases.
- **Backup and Recovery:** It is an important feature available in almost all the DBMS programs. By using this feature, we are able to have the backup of our data and can later, use it to reinstate it in case of data failure, corruption or loss.

Exercise 1c

1. **Fill in the blanks:**

(i) DBMS stands for ____________________________

(ii) A ___________ is a collection of related fields

(iii) A file is a collection of _____________________

(iv) Before processing the data is recorded in ______________.

(v) A __________ is a collection of logically related data

(vi) The data definitions is stored in ___________.

(vii) SQL stands for _____________________

(viii) Hierarchical data Model has the general shape of a(n) _____________.

(ix) Data is a collection of _______, _________ and ________

(x) Processed data is called _________________

2. **Select the correct option:**

(i) Which of the following represents a collection of concepts that are used to describe the structure of a database?

a) data warehouse b) data model
c) data structure d) data type

(ii) Which of the following data model is more flexible?

a) Network data model b) Hierarchical data model
c) Relational data model d) object data model

(iii) Which of the following type of file require largest processing time?

a) Sequential file b) Random file
c) Indexed sequential file d) Direct access file

(iv) Which of the following may be a temporary file?

a) Master file b) Transaction file
c) Backup file d) None of these

(v) SQL is a(n):

a) Unstructured language b) Structured language
c) Object oriented language d) Software

3. Write T for true and F for false statement.

   (i) Data can only be processed through computers

   (ii) The traditional file system approach has many advantages over DBMS approach.

   (iii) Data dictionary is used to view the meanings of database terminology

   (iv) Master file is the latest updated file which never becomes empty, ever since it is created.

   (v) SQL is used to retrieve information from the database based on certain criteria.

   (vi) The Network Data Model is more popular and widely used than Relational Data Model.

   (vii) Indexed sequential files can be processed sequentially as well as randomly.

   (viii) Backup files store data prior to its processing.

4. (ix) Microsoft ACCESS is a relational database management system

   (x) A report generator is used to produce a printed document from the database.

5. a) Differentiate between *Data* and *Information* .

   b) What activities are involved in data processing? Discuss in details.

6. Define file, record and field in details?

7. Describe the file types from usage point of view and functional point of view?

8. How do we organize the files on storage media?

9. In general, what activities are to be performed on the databases? Discuss in details.

10. What are the four major components of the database systems? Write in details.

11. Discuss the objectives of the databases in your own words.

12. Describe the different database models?

13. Discuss the objectives and features of the DBMS?

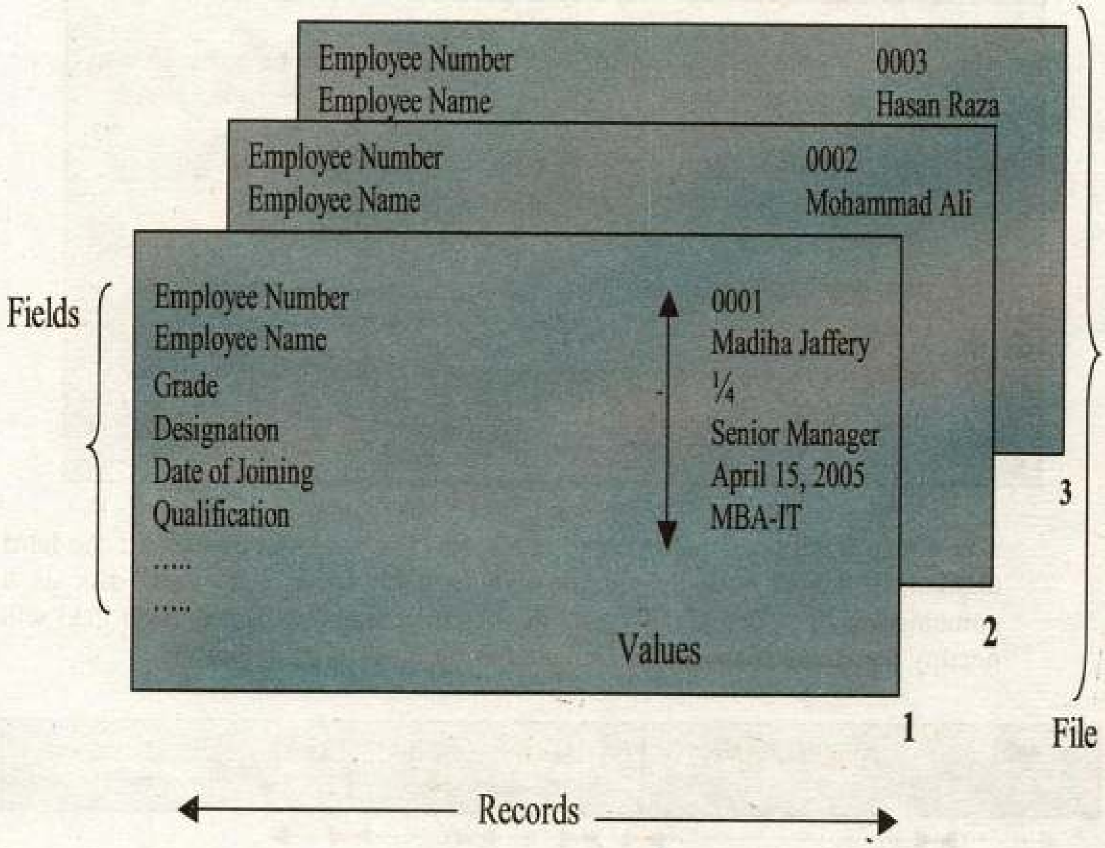
14. What are the advantages and disadvantages of the DBMS?

BASIC CONCEPTS AND TERMINOLOGY OF DATABASES

Chapter 2

2.1 OVERVIEW

In the previous chapter, we have discussed Files and Records in details. In fact, the concept of databases evolved from the old, traditional working of *File Management System* (FMS). Let us see how this *evolution* emerged by considering the same diagram of Record and File.



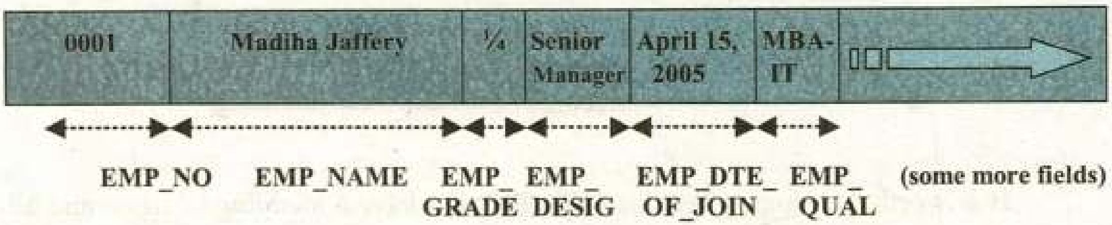
It is worth mentioning here that the file must have a *meaningful* name and all the field names in each record must be *unique* and *meaningful* too. For example, we can give the following names to the above mentioned *file* and its *Record Fields*:

File name	: EMP_BIO_INFO	(Employee's bio-graphic information)
Employee Number	: EMP_NO	
Employee Name	: EMP_NAME	
Grade	: EMP_GRADE	
Designation	: EMP_DESIG	
Date of Joining	: EMP_DTE_OF_JOIN	
Qualification	: EMP_QUAL	
-----	}	*Some more fields*

Basic Definitions

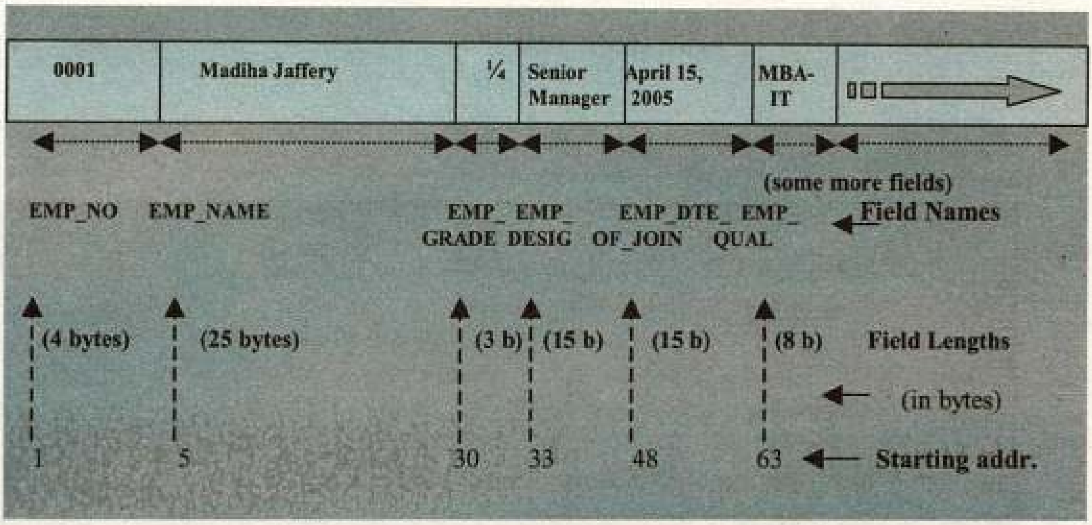
- **Field :** A field is a unit of data consisting of one or more charact i.e., Employee number Employee name or grade of an emplo in a record of the employee.
- **Record :** A collection of related data items treated as a single uni is called a record.
- **File:** A collection of related records treated as a single unit is called a file or a data set.

The above mentioned record can be shown (as it would appear on the hard disk) as follows. Actually, all the data will be saved onto hard disk as a combination of "0"s and "1"s i.e., the machine language form. Each field will occupy the space as would be allocated at the time of its definition.



2.1.1 Data Handling in File Management Systems

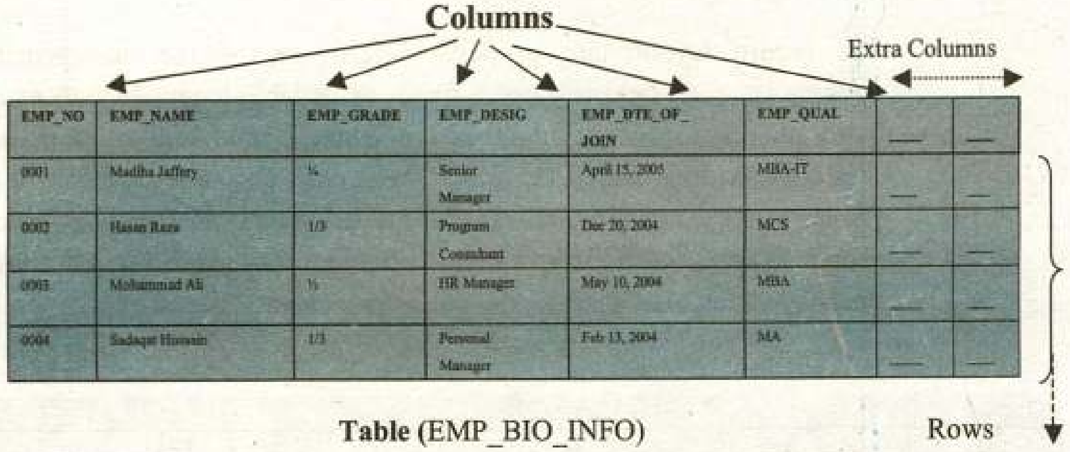
- The record layouts are properly defined in the file management systems i.e. each field is given a fixed- or variable-length sequences of bytes and they are put together contiguously in fixed- or variable-length collections called *records*. Thus, each field corresponds to a proper starting memory address. As the fields have already been given proper lengths, so the values of each field are determined within those memory addresses. See the example given below :



- Thus, the field names are used only as a "names reference" within the programs using them. Their values "flows" with them as the contents of the memory spaces they are occupying.

2.2 ATTRIBUTES, ROWS AND TABLES

In the late 1960s, the researchers came up with an idea of using *Relational Databases* instead of file structures. They gave the idea of defining a file as a "Two Dimensional" array (or *Table* having a unique name), placing all the fields as *columns* (having unique names) of that table and putting each record as a *row* into the table. Each row is also known as a *tuple* or *occurrence* in that table. Following these concepts, we can see the traditional file structure into a new and easy to manage database structure called a *Table* or *Relation*.



EMP_NO	EMP_NAME	EMP_GRADE	EMP_DESIG	EMP_DTE_OF_JOIN	EMP_QUAL	—	—
0001	Madiha Jaffery	¼	Senior Manager	April 15, 2005	MBA-IT	—	—
0002	Hasan Raza	1/3	Program Consultant	Dec 20, 2004	MCS	—	—
0003	Mohammad Ali	½	HR Manager	May 10, 2004	MBA	—	—
0004	Sadaqat Hussain	1/3	Personal Manager	Feb 13, 2004	MA	—	—

Table (EMP_BIO_INFO)

So, consequently, we summarize the above discussion as follows :

- **Data Elements** : The fields or data items in databases are termed as data items, items, attributes or columns in database structures.
- Records in file management structures are termed *as rows or tuples in* database structures.
- Files or datasets in databases are termed as *tables, relations or data objects* in database structures.
- The collection of tables with some traditional files and some other necessary data objects is termed as a *database*.

2.3 RELATION or TABLE

A two dimensional array or table of data containing descriptive information about an entity. The entity must have a unique identifier, which is composed of a combination of one or more attributes, and each attribute must have one and only one value. It is appropriate to define the word *Entity* here.

Entity: An *entity* is any thing about which you want to keep information in the database. Let us consider an example of "Student Information System", which has entities like student, teacher, course list, scholarships, time-tabling etc. Thus, the entities involved in this case are the same and the entity "student" can be defined in the form of database modeling as follows :

- STUDENT (STUD_NO, STUD_NAME, STUD_GENDER_CD, STUD_B_DTE,
  STUD_ADDR, STUD_TEL_NO)

From the above given definition of entity, we can easily construct a two-dimensional array or a relation by converting all the attributes in the brackets into columns of the array, as follows :

Rows

STUD_NO	STUD_NAME	STUD_GENER_CD	STUD_B_DTE	STUD_ADDR	STUD_TEL_NO

Columns

In the above diagram, all the attributes are called columns of the table and all occurrences of the records are called rows.

For example, considering the entity 'STUDENT' as above, following table or relation can be constructed as a part of the database.

STUDENT

STUD_NO	STUD_NAME	STUD_GENER_CD	STUD_B_DTE	STUD_ADDR	STUD_TEL_NO
M001-F05	Ahmad Dar	M	12-10-1980	12, St#5,F10-I	9258481
F009-S04	Saba Afzal	F	31-03-1982	Abcdefgh	9292999
M100-S03	Adil Amin	M	28-02-1980	Gulberg, LHR	5566778

A relation STUDENT having *rows* (records) under *columns* (attributes)

2.3.1 Properties of Relation

A Relation or a Table which is the basis of a Relational DBMS, by definition must have certain inherent characteristics that form the basic for its underlying strength and flexibility. Because of these features, an application implemented by using such a system is much more flexible and can be easily modified when alterations or enhancements to the underlying data base take place. These characteristics are:

- **No duplicate rows exist:** No two rows can be identical. Why to put two rows (records) for the same entity (e.g., a Person). It will also violate the definition of what a relation represents, as it says by definition that there must be a unique key for each row in a relation/table.
- **The order of Rows is insignificant:** There is no ordering or sequencing of the rows in the tables. The relational implementation of the tables support all required access mechanism i.e., it is not necessary to sequence the rows according to the key field.
- **The order of Columns is insignificant:** Again, the order of the columns/attributes in defining a relation/table has no significance. The later insertions of the columns are made at end of the existing columns by the system itself. The system acquires the data (of columns) by their names.
- **Columns/Attributes are all Elemental or Atomic:** All the intersections of Rows and Columns must have a (single) value. The nulls are inserted by the system at the time of later insertion of a column, which should immediately be replaced by zeros/spaces or valid values for that particular column.

2.4 VIEWS

Views are created by using SQL, which is a powerful database language, used for data definition and data manipulation purposes. The purpose of using views is purely to keep the data safe and secure from un-authorized and illegal users. The views provide the descriptions of relations that are not stored, but constructed as needed from stored relations.

To create a view, normally the following create sql command is used :

```
CREATE VIEW    STUDENT_VIEW_01 AS
      SELECT   STUD_NO, STUD_NAME, STUD_ADDR
      FROM     STUDENT
      WHERE    STUD_GENDER_CD = “M”;
```

This will create a *view* from the STUDENT table for only **male** students, which can be used by the users according to the authorization given to them, leaving the original table aside, safe and secure.

2.5 INDEXES

It is another table created by the system developer/DBA containing the key attributes of the table for which the Index is created. It has a very vital role in the data base management systems, especially in RDBMS. The important associations defined in the system make use of this. It helps the system run smooth and fast.

2.6 KEYS

A key is a single or combination of one or more fields and its purpose is to point/retrieve the data rows from the tables, according to the requirement.

Keys are defined in the relations/tables to access or sequence the stored data fast and smooth or to create the links between them.

Let us assume two relations/tables as follows to define different types of key.

PATIENT

PATNO	PNAME	PHCODE	LOCN	STATUS
191	ABID	ME1000	1000101	01
192	SALIM	ME1001	1000102	01
193	FAROOQ	ME1003	1000103	01
198	KASHIF	SU1008	1000104	01
201	ZAHID	ME1000	1000101	09
202	NAEEM	ME1000	1000105	01
213	KHALID	SU1008	1000107	01

PHYSICIAN

PHCODE	PHNAME
ME1000	Dr. Alamgir
ME1001	Prof. Aslam Naqvi
ME1003	Dr. Aftab Chaudhery
ME1008	Dr. Ehsan Haider

2.6.1 Types of KEYS

- **Primary Key:** In a relation, the attribute (column) or a combination of attributes (columns) that uniquely identifies a row or a record. PATNO is the attribute that uniquely identifies each patient and thus can be used as a Primary key for the above defined table 'PATIENT'. On the other hand, PNAME is normally not unique, so it can not be used as a primary key.
- **Secondary Key:** A secondary key is non-unique field that is used as a secondary(alternate) key. In the above table, by using PHCODE (physician code), we can scan the records from the table. If the physician leaves, we can change it with a new one by using update statement.
- **Candidate Key/Alternate Key:** Sometimes, it is unclear which field to select as the primary key. There might exist some additional field (or combination of fields) that also have the uniqueness property. These keys can be termed as Candidate keys or Alternate keys.
  - In addition for the uniqueness requirement, Candidate keys must possess following two properties:

- **Composite /Concatenate Key:** These keys consists of two or more data elements or attributes. Invariably, these are the same as Candidate/Alternate keys except that of uniqueness requirement. In order to make it unique, assign STATUS or another attribute (say patient's ID number).
- **Sort/Control Key:** A sort/Control key is used to physically sequence the stored date according to our need. Multiple attributes can be used as sort fields.
- **Foreign Key:** A foreign key is an attribute in a table whose values must match a primary key in another table. The table in which the foreign key is found is called as *dependent* table and to which it refers is called as *parent* table.

NOTE : Foreign key relationships are the basis for establishing 1:1 or 1:M relationships across the Relations/Tables in a relational database management system.

2.7 THE USER

The *user* or *end-user* is simply a person who uses the computers for his specific need. He might have a moderate knowledge of computers, computer science and information technology, and his need to use the computers may be entertainment, education, or professional tasks. He does not need to know the in-depth knowledge of the computer systems, but instead, he should be aware of the installed software he intends to use.

2.8 THE DATA ADMINISTRATOR

A *data administrator* (DA) is responsible for the entire data of an organization. He normally develops the overall functional requirements for the databases being used in the office. He shares in developing the logical design for each database. He should control and manage the databases, establish the data standards, supervise the data distribution within the organization and communicate with the users when necessary. He should also participate in developing the data dictionary, prepare documentation and conduct user training where needed. Normally, the Data Administrator serves as a bridge between users and data processing staff.

2.9 THE DATABASE ADMINISTRATOR

A *database administrator* (DBA) is responsible for the design, implementation, operation, management and maintenance of the database. He/She must be technically expert on the overall intricacies of the database and DBMS. He is supposed to plan, coordinate and carry out a variety of jobs during all phases of the database projects. He must possess the technical skills because he has to work on the complex software and hardware issues involved and to solve the problems of the system and application experts in the organization. He is also responsible to make sure the database access rights, to safeguard its security and to maintain and fine-tune the database functionality.

Exercise 2c

1. Fill in the blanks:

(i) A(n) _____________ is a two dimensional array containing descriptive information about an entity.

(ii) A(n) _________ is any thing about which the information is kept in the database.

(iii) In a table the order of rows and columns is _______________

(iv) In a relation, the attribute or a combination of attributes that uniquely identifies a record is called _____________.

(v) A(n) __________ describes the characteristics of an entity.

(vi) A(n) ____________ is an attribute in a table whose values must match a primary key in another table.

(vii) A(n) _____________ is responsible for the design, implementation, operation, management and maintenance of the database.

(viii) A(n) __________ consists of two or more attributes.

(ix) A(n) _________ is the dynamic result of one or more relational operations on the base relations to produce another relation.

(x) The _________ refers to raw facts and figures.

2. **Select the correct option:**

(i) Insert command is used to insert:

a) a new table b) a new record

c) a view d) dependencies

(ii) CREATE command is used to create a:

a) table b) view

c) report d) query

(iii) SQL is used for:

a) data definition b) data manipulation

c) data definition and manipulation d) searching records

(iv) The foreign key is found in:

a) parent table	b) dependant table
c) pivot table	d) index table

(v) A table must have:

a) primary key	b) secondary key
c) composite key	d) sort key

3. Mark as True or False
   - (i) The view is not stored in the database.
   - (ii) Two tables can not have the same name in the database.
   - (iii) Index makes the searching of a record faster.
   - (iv) Secondary key must be unique.
   - (v) The primary key can not work as a sort key.
   - (vi) The DBA is responsible for maintaining the database.
   - (vii) A file is a collection of related fields.
   - (viii) DBMS provide more security to protect data than traditional file management systems.
   - (ix) DBMS is a software used to train database administrator.
   - (x) A relation is also termed as a tuple in relational database.
4. How the Records and Files are constructed in traditional File Management System ?
5. How the data is stored and retrieved in FMS (file management system)?
6. How the tables/relations are formed up in DBMS?
7. Discuss the data manipulation in DBMS system.
8. Write down the properties of relations in details.
9. What is a VIEW? How do we create it?
10. What is usage of indexes in FMS and DBMS?

DATABASE DESIGN PROCESS

Chapter 3

3.1 OVERVIEW

Before we design a database for any organization, we have to consider many aspects to find out the practical scenario of owning a database. Few of them are discussed as follows :

Feasibility Study: This is also called preliminary investigation of the required database. It involves the area identification and selection i.e. which area or aspect is to be selected to start with. After the project is selected, it is allocated a specific fund and a proper planning is chalked out for its practical implementation. Side by side, a proper market analysis is also worked out.

Requirements Analysis: During this activity, the requirements are gathered i.e. the possible inputs for the database and the required functionality out of it. The users precisely narrate their needs of the database and the possible domain and restrictions are also chalked out.

Project Planning: A proper schedule is laid down to accomplish this activity. All the cost factors are taken into consideration i.e., the salaries of team members, their logistics involved, other trivial expenses (such as marriage gifts, insurances etc) and hardware costs.

Data Analysis: This is an important analysis aspect while designing a database. It involves the following activities :

(i) Data Flow Diagrams (DFD)
(ii) Decision Tables
(iii) Decision Trees

However, a detailed discussion on these topics is beyond the scope of this book.

3.2 DATA MODELING

Data Modeling is the process of identifying the data objects and the relationships between them.

Ingredients of Data Modeling:

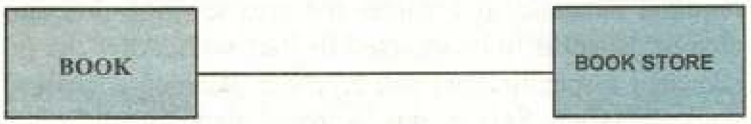
Entities/Objects : A data entity/object is anything that is participating in the system. It is always properly identifiable i.e., a TEACHER, a STUDENT, an AEROPLANE.

Attributes: *Attributes* define the objects, describe their characteristics and in some cases, make references to other objects(s) i.e., attributes for a TEACHER could be :

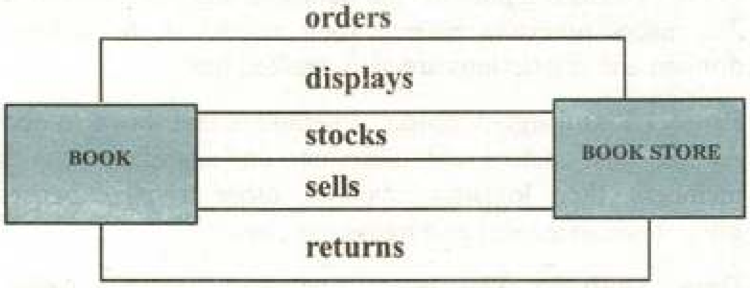
Teacher Name, Gender, Last Degree, Appointment Date, Pay Scale, Nationality, Telephone No. etc.

Relationships: The *relationship* indicates how the Entities/Objects are *Connected* or *Related* to each other.

The Data objects are related/connected to one another in different ways. Consider the data objects *BOOK* and *BOOK STORE* in the following diagrams.



(a) A basic connection between the objects



(b) Relationships between the objects

Following are the different possible and relevant relationships between them:

- A BOOK STORE **orders** BOOK(s).
- A BOOK STORE **displays** BOOK(s).
- A BOOK STORE **stocks** BOOK(s).
- A BOOK STORE **sells** BOOK(s).
- A BOOK STORE **returns** BOOK(s).

It is important to note that :

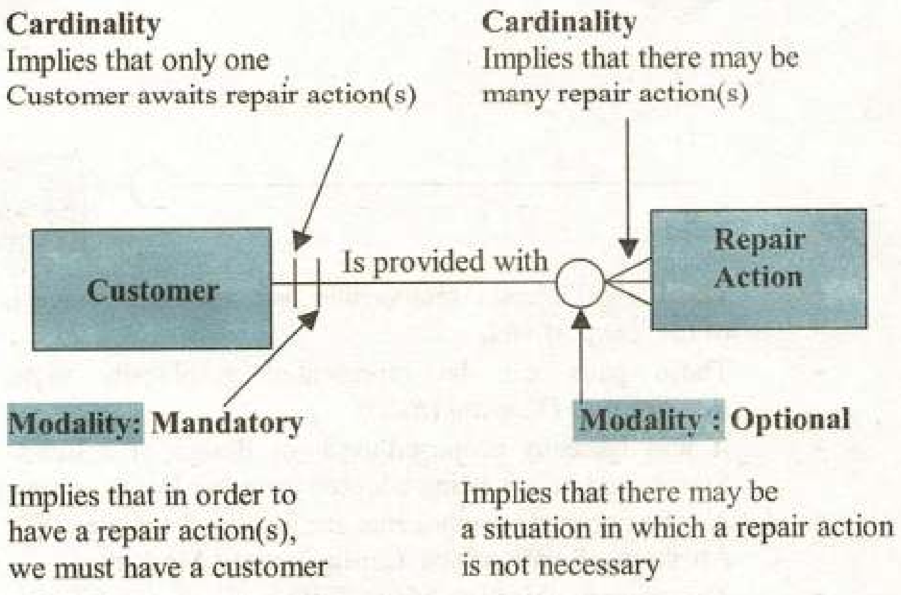
- All the relationships define the relevant *connections* between both objects.
- All the relationships are *bi-directional.*
- We have to consider only the relevant relationship (in the context of the requirement)

Cardinality:

- Whether some occurrence(s) of *object*-1 are related to some occurrence(s) of *object*-2.
- It is expressed as *one* or *many* e.g.,
  - A husband can have only *one* wife and
  - A Father can have *many* children.
- The relationships can be
  - One to one
  - One to many
  - Many to many
  - Recursive
  - None

Modality:

- It defines the nature of the relationship i.e.,
  - **Optional** .... represented by 0
  - **Mandatory** .... represented by 1
- Consider two objects *Customer* and *Repair Action* in a Workshop environment.



Cardinality
Implies that only one Customer awaits repair action(s)

Cardinality
Implies that there may be many repair action(s)

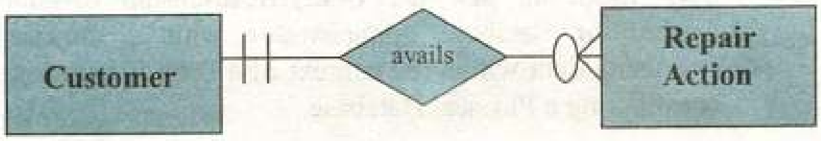
Modality: Mandatory

Implies that in order to have a repair action(s), we must have a customer

Modality : Optional

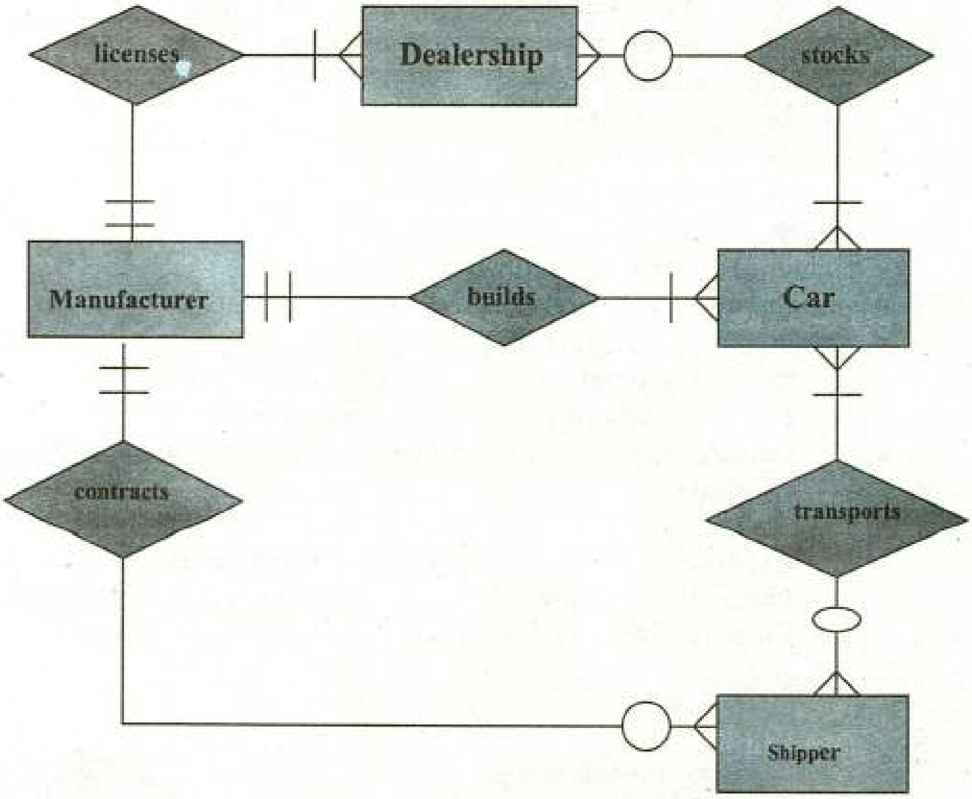
Implies that there may be a situation in which a repair action is not necessary

- A simple *Data Model* can be drawn from the above as follows:



- By connecting all the *Data Objects* alongwith their *Relationships* in the above manner, an *ERD* (Entity Relationship Diagram) is constructed.

Entity-Relation Diagram (An Example)

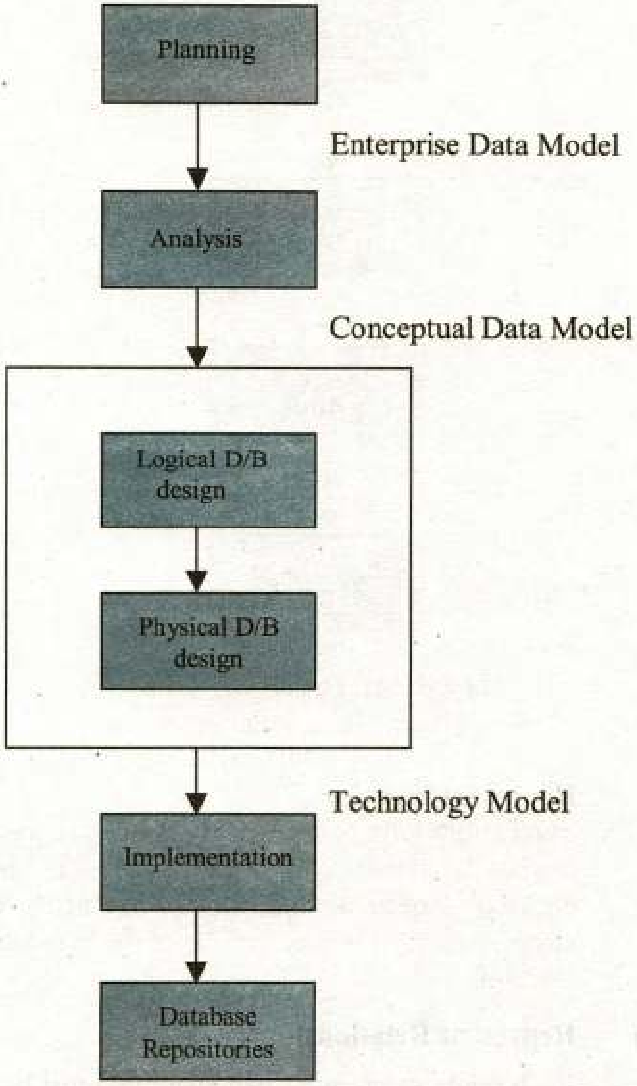


- The Entity/Object – relationship pair discussed above is the objective of the *Data Model*.
- These pairs can be represented graphically using the *Entity-Relationship Diagram* (*ERD*).
- It was basically proposed/used for design of a Relational Database System and now is being adopted for other Database types also.
- A set of primary components are identified for the *ERD*: Data objects, Attributes, Relationships, Cardinality and Modality.
- The primary objective of the *ERD* is to represent *Entities/Objects* and their *relationships / association*.
- Data Modeling and the Entity-Relationship diagram provide the Analyst or database administrator with a concise notation for examining data within the context of a Data Processing Application or constructing a Physical Database.

3.3 DATABASE DESIGN

The major objective of Database design is to map the conceptual data model to an implementation model that a particular DBMS can process with performance that is acceptable to all users throughout the organization. In today's compatible economy, database users require information that is complete and up-to-date and they expect to be able to access this information quickly and easily.

Following is the database development process.



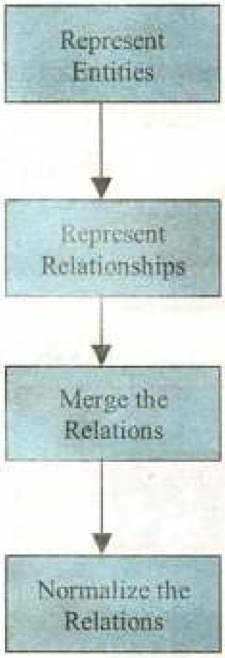
Database design can be divided into the following two phases:

3.3.1 Conceptual (Logical) Database Design:

The process of mapping the conceptual data models (from analysis) to structures that are specific to the target DBMS. If the target environment is a relational DBMS, then the conceptual data models are mapped to normalized relations.

Following diagram presents an overview of logical design process.

Conceptual Data Model
(E-R Diagram)



LOGICAL DATA MODEL
(Normalized Relations)

(i) **Represent Entities:**
Each entity type in the E-R diagram is represented as a relation in the Relational View or Data Model. The identifier of the entity type becomes the Primary key of the relation, and other attributes of the entity type become non-key attributes of the relation.

(ii) **Represent Relationships:**

Each relationship in an E-R diagram must be represented in the relational model. It depends on its nature. For example, in some cases, we represent a relationship by making the primary key of one relation a foreign key of another relation. In other cases, we create a separate relation to represent a relationship.

(iii) Merge the Relations:

In some cases, there may be redundant relations (that is, two or more relations that describe the same entity type). They must be merged to remove the redundancy. This process is also known as View Integration. Suppose we have one relation as:
EMPLOYEE1(EMPNO , NAME,ADDRESS,PHONE)

And another relation as:
EMPLOYEE2(EMPNO , ENAME, EMP-ADDR, EMP-JOB-CODE, EMP-DOB)

Since the two relations have the same primary key (EMPNO) they describe the same entity and may be merged into one relation. The result of merging the relations is the following relation.

EMPLOYEE(EMPNO , NAME,ADDRESS,PHONE,EMP-JOB-CODE,EMP-DOB)

(iv) Normalize the Relations:

The relations that are created in step (i) and (ii) may have unnecessary redundancy and may be subject to anomalies (or errors) when they are updated. Normalization is the process that refines the relations to avoid these problems. (A detailed discussion is given in chapter 4)

3.3.2 Physical Database Design

It is the last stage of the database design process. The major objective of physical database design is to implement the database as a set of stored records, files, indexes and other data structures that will provide adequate performance and ensure database integrity, security and recoverability.

There are three major inputs to Physical database design.

(i) **Logical database structures** (developed during logical database design) i.e., the Normalized Relations.

(ii) **User processing requirements** i.e. size and frequency of use of the data-Base , response time, security, backup, recovery etc.

(iii) **Characteristics** of the DBMS and other components of the computer Operating environment.

Components of Physical Database Design:

(i) **Data Volume and Usage Analysis**:

To estimate the size or volume and the usage patterns of the database. Estimates of database size are used to select Physical storage devices and estimate the costs of storage. Estimates of usage paths or patterns are used to select the file organization and access methods, to plan for the use of indexes and to plan a strategy for data distribution.

(ii) **Data Distribution Strategy**:

Many organizations today have distributed computing networks. For these organizations, a significant problem in physical database design is deciding at which nodes (or sites) in the network to physically locate the data.

Basic data Distribution Strategies.

a. **Centralized:** All data are located at a single site. It is fairly easy to do but it has at least three disadvantages.

- data are not readily accessible at remote sites. Data communication costs may be high.
- The database system fails totally when the central system fails.

b. **Partitioned:** The database is divided into partitions (fragments). Each partition is assigned to a particular site. Major advantage of this is that data is moved closer to local users and so is more access-able.

c. **Replicated:** Full copy of database is assigned to more than one site in the network. This approach maximizes local access but creates update problems, since each database change must be reliably processed and synchronized at all of the sites.

d. **Hybrid:** In this strategy, the database is partitioned into critical and non-critical fragments. Non-critical fragments are stored at only one site, while critical fragments are stored at multiple sites.

(iii) **File Organization:**

A technique for physically arranging the records of a file on secondary storage devices. For selecting a file organization, the system designer must recognize several constraints, including the physical characteristics of the secondary storage devices, available operating systems and file management software, and user needs for storing and accessing data. Following is the criteria for selecting file organizations.

- Fast access for retrieval.

- High throughput for processing transactions.
- Efficient use of storage space.
- Protection from failure or data loss.
- Minimizing need for re-organization.
- Accommodating growth.
- Security from un-authorized use.

(iv) Indexes:

An index is a table that is used to determine the location of rows in a table (or tables) that satisfy some condition. They may be created on primary key, secondary key, foreign key etc.

(v) Integrity Constraints:

Database integrity refers to the correctness and consistency of data. It is another form of database protection. While it is related to security and precision, it has some broader implications. Security involves protecting the data from unauthorized operations, while integrity is concerned with the quality of data itself. Integrity is usually expressed in terms of certain constraints which are the consistency rules that the database is not permitted to violate. A few of them are discussed in chapter 4.

3.4 IMPLEMENTATION

In database implementation phase, the builder or the database administrator normally requires a server computer which will be linked with hundreds and thousands of computer users who would want to share and interact with the server (database).

For this purpose, the dba might need the services of network administrators to connect the users with the server. The users are normally given the authorizations / permissions defined by their respective managers so that they can perform the authorized tasks while using the database facilities.

In distributed computing environment, the database servers and users might be thousands of kilometers apart, so a lot of expensive telecommunication links are required to perform the designated tasks. NADRA and CRICKINFO are some of the typical examples of this type of databases.

Exercise 3c

1. Fill in the blanks:

(i) During ___________ phase, the project requirements are gathered and identified.

(ii) DFD stands for ______________.

(iii) The process of identifying data objects and relationship between them is called ______________.

(iv) The number of occurrences of participating entities in a relationship is determined by the ____________ ratio.

(v) Modality determines whether the participation of an entity in a relationship is ___________ or optional.

(vi) ERD stands for _____________.

(vii) In ERD model, a(n)_____________ is represented by a rectangular box.

(viii) In ____________ database systems, all the data is stored at a single site.

(ix) In _________ database multiple copies of the same data are stored at different sites on the network.

(x) In distributed databases, the data is ____________ among various sites.

2. Select the correct option:

(i) Which of the following keys does not hold uniqueness property:
a) candidate key b) foreign key
c) primary key d) secondary key

(ii) An entity related to itself in an ERD model refers to:
a) recursive relationship b) one-to-many relationship
c) many-to-many relationship d) one-to-one relationship

(iii) Database development process involve mapping of conceptual data model into:
a) Object oriented data model b) Network data model

c) Implementation model d) Hierarchical data model

(iv) In ERD model, the relationship between two entities is represented by a:
a) diamond symbol b) rectangular box
c) oval symbol d) line

(v) In hybrid distribution which kind of fragments are stored at only one site:
a) critical fragments b) non-critical fragments
c) critical and non-critical fragments d) only large fragments

3. Write T for true and F for false statement.

(i) In one-to-one relationship only one instance of each entity can participate in the relationship.

(ii) The optional modality is represented by 1.

(iii) One-to-many is a uni-directional relationship.

(iv) In ERD model, a condition is mentioned in a diamond symbol.

(v) ERD is a physical data model.

(vi) In hybrid distribution the database is partitioned in critical and non-critical fragments.

(vii) In distributed databases, the consistency refers to availability of same data at all sites of the network.

(viii) Indexing maximizes the time required to search a piece of information from a database.

(ix) Analysis is less important activity than coding, so minimum time should be spent over analyzing the system.

(x) Relationship defines the logical connection between entities.

4. Describe different steps involved in analysis stage while designing a database.

5. Explain the following with the help of figures:
(i) Entity/Object
(ii) Attribute
(iii) Relationship
(iv) Cardinality
(v) Modality

6. Draw and explain ER diagram for the system of getting admission in your college.

7. Explain the following:
(i) Physical data model
(ii) Conceptual data model

8. What are the components of a logical data model?

9. What elements combined , produce the physical database design? Explain

10. Define and explain the following terms:
(i) Data distribution strategy
(ii) File Organization

11. Define the term Analysis. Briefly discuss the following terms:
(i) Feasibility study
(ii) Requirement Analysis
(iii) Project Planning
(iv) Data Analysis

12. Briefly explain the database design process with the help of a diagram.

DATA INTEGRITY AND NORMALIZATION

Chapter 4

4.1 OVERVIEW

4.1.1 Data Integrity

Database integrity refers to the correctness and consistency of data. It is another form of database protection. While it is related to security and precision, it has some broader implications as well. Security involves protecting the data from unauthorized operations, while integrity is concerned with the quality of data itself. Integrity is usually expressed in terms of certain constraints which are the consistency rules that the database is not permitted to violate. Following two are the most important constraints in relational databases:

(i) **Entity Integrity:** is a constraint on primary values that states that no attribute of a primary key should contain nulls.

(ii) **Referential integrity:** is a constraint on foreign key values that states that if a foreign key exists in a relation, then either the foreign key value must match the primary key value of some tuple in its home relation or the foreign key value must be completely null.

4.1.2 Normalization

Normalization is the process of converting complex data structures into simple and stable data structures. It is based on the analysis of *functional dependence*.

In other words, Normalization is a technique for reviewing the entity/attribute lists to ensure that attributes are stored "where they belong". It is the basis for a relational data base system. In practice, it is simply an applied common sense. More formally stated, it is the process of analyzing the dependencies of attributes within entities. Attributes for each entity are checked consecutively against three sets of rules, making adjustments when necessary to put the entity in First, Second and Third normal form.

First, we discuss what is functional dependence.

"A functional dependency is a particular relationship between two attributes. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B". The functional dependence of B on A is represented by an arrow, as

A B) An attribute may be functionally dependent on two or more attributes rather than a single attribute. For example, consider the relation:

COURSE (<u>STUDID</u> , <u>CRSNO</u> , CRSDATE)

The functional dependency in the relation is represented as follows:

$$\text{STUDID , CRSNO} \longrightarrow \text{CRSDATE}$$

The attribute on the left hand side of arrow is called determinant.

Before NORMALIZATION process, the initial entity/attribute list(s) must be checked for errors or oversights. There may be some hidden problems as:

(i) **Synonyms:** A synonym is created when two different names are Used for the same information (attribute). If an attribute resides in more than one entity, make sure that all entities use the same attribute name. For example, consider the following two entities:

$$\begin{tikzcd}
\underline{\text{ITEM}} & \underline{\text{SUPPLIER}} & \\
\text{Stock\_no} & \text{Supplier\_Id} & \text{(error)} \\
\text{Item\_colour} & \text{Supplier\_Name} & \\
\text{Supplier\_Code} \arrow[uur] & &
\end{tikzcd}$$

We should use Supplier_Code instead of Supplier_Id in SUPPLIER.

(ii) **Homonyms:** A homonym is created when same name is used for Two different attributes. Consider the following example:

<u>CUSTOMER</u>	<u>SUPPLIER</u>	
Company_Name	Company_Name	(error)

We should use Supplier_Name instead of Company_Name in SUPPLIER.

(iii) **Redundant Information:** Storing the same information in two different ways or forms. Consider the example:

<u>Employee</u>	
Employee_Age	(error)
D_O_Birth	

Only one attribute can serve the purpose (The programmer can manipulate the Age by using D_O_Birth as the basis).

(iv) **Mutually Exclusive Data:** Mutually exclusive data exists when attributes occur whose values can be expressed as "yes/no" indicators, can not all be true for any single entity. As an example, consider the proposed attributes of "MARRIED" and "SINGLE" in an Employee entity:

Employee		
Married	(a flag set if the employee is married)	error
Single	(a flag set if the employee is single)	error

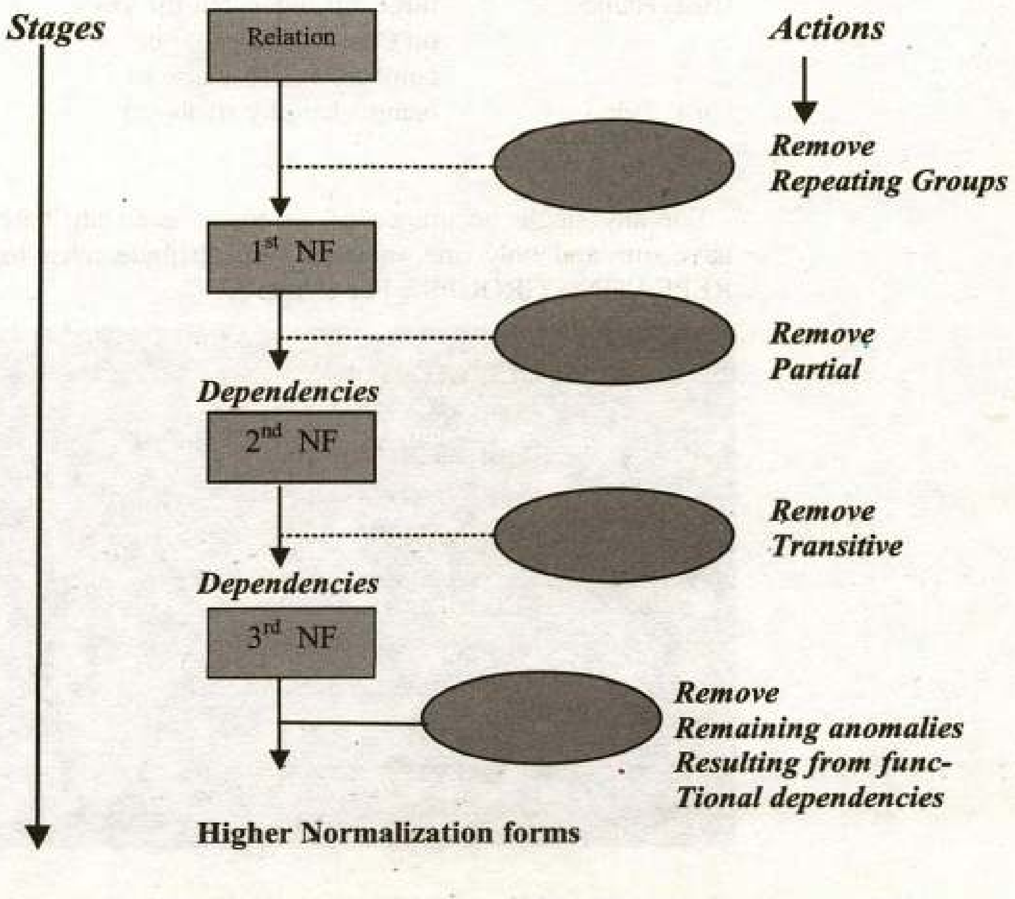
Quite often, errors of this type represent values of a larger category. Whenever possible, resolve the error by creating the larger categorical attribute. In this case, these two elements should be combined into a single attribute of "MARITAL_STATUS" which would have a value of either M (married) or S (single).

Employee	
MARITAL_STATUS	(An indicator of the employees Marital status)

Normalization Steps:

Normalization is often accomplished in steps, each of which corresponds to a normal form. It can be graphically expressed as follows:

(Please see the next page)


Stages
Relation
Actions
Remove Repeating Groups
1st NF
Remove Partial
Dependencies
2nd NF
Remove Transitive
Dependencies
3rd NF
Remove Remaining anomalies Resulting from func-Tional dependencies
Higher Normalization forms

Note: Our scope of study in this semester is only upto 3rd NF.

A Normal Form is a state of a relation that can be determined by applying simple rules, regarding dependencies (or relationship between attributes), to that relation. Following is a brief discussion on different stages:

(i) First Normal Form (1 NF)

"A relation R is in First Normal Form if and only if all underlying domains contain atomic values only".

The pre-requisite is that, A relation has always a primary key associated with it. Thus, we can define it as follows also:

- All entities must have a key, composed of a combination of one or more attributes which uniquely identify one occurance of the entity. For example:

CUSTOMER

Cust_Id	(We can create key on these
Cust_Name	three attributes, but the key
..	on Cust_Name could be
..	cumbersome (because of
Cust_Telno	being a lengthy attribute)
..	
..	

- For any single occurance of an entity, each attribute must have one and only one value or "An attribute must have no REPEATING GROUPS". For example:

Case-1:	<u>DEPARTMENT</u>	
	Dept_No	
	Dept_Name	
	Emp_No	(error)
	Emp_Name	(error)
Case-2:	<u>MAGAZINE</u>	
	Mag_Code	
	Mag_Name	
	Mag_Addr	
	Mag_Telno	
	Issue_Date	(error)

Case-3:	ITEM	
	Item_Code	
	Item_Desc	
	Stock_No	
	Item_Type	
	Weight	
	Colour	
	Qty_Ordered	
	Unit_Price	
	Order_No	(error)
	Cust_Name	

The error exists in the above examples because some Attribute(s) are being repeated for a single occurance of Each record. We should try to avoid this repitition. Following are the steps to achieve this:

Step-1: Whenever repeating groups occur, the repeating Attribute must be removed and placed "where it Belongs", under the entity that it describes.

Step-2: Next, study the relationship of where the Repeating attribute came <u>from</u>, and where the Attribute went <u>to</u> . Determine if the From-To Relationship is 1:M or M:N.

Let us apply these steps in case-1. We end up with the Following two relations.

DEPARTMENT	EMPLOYEE
Dept_No	Emp_No
Dept_Name	Emp_Name

Now ask for the relationship in this case:
"For one department, are there one or more mployees?"
Then repeat the question in reverse. "For one employee, Are there one or many departments?" In this case, one Department has many employees, but one employee has Only one department. Therefore, the relationship is 1:M. When the relationship is 1:M, this is acceptable. No Further adjustment is necessary to make it 1:M or M:1.

But if we apply the above steps in case-3, we end up with The following two relations:

ITEM	ORDER
Item_Code	Order_No
Item_Desc	Cust_Name
Stock_No	Qty_Ordered
Item_Type	Unit_Price
Weight	
Colour	

Now ask the similar question in this case also, we find out that for each item, there are many orders, and for each Order, there are many items. Thus it is M:N relation. The Main problem here is, "where to store the intersection Data i.e., Qty-Ordered and Unit-price". To solve this, Create another entity "ITEM_ORDERED" as:

ITEM-ORDERED
Order_No
Item_No
Qty_Ordered
Unit_Price

And define the key as "the combination of Order_No and Item_No on this entity. This provides us two more tables Having 1:M relationship, which is acceptable.

To have a better understanding about the whole concept, let us consider the following table of data (having repeating groups):

STUD-ID	NAME	DEPT	MONFEE	CRSNO	CDTE
100	Ahmad	Marketing	1000	SPSS	190696
				SURVEYS	100796
140	Nazir	Accounting	1200	TAXACCT	120897
110	Hamid	Inf.Systems	1100	SPSS	140796
				COBOL	220796
190	Rashid	Finance	1200	INVESTMT	200697
150	Hussain	Marketing	1000	SPSS	190697
				SYSANAL	200797

To bring it to first Normal Form, we eliminate the repeating groups from the Table and fill in the missing information (in the cells having no information). Name the table as "STUDENT".

STUDENT

STUD-ID	NAME	DEPT	MONFEE	CRSNO	CDTE
100	Ahmad	Marketing	1000	SPSS	190696
100	Ahmad	Marketing	1000	SURVEYS	100796
140	Nazir	Accounting	1200	TAXACCT	120897
110	Hamid	Inf.Systems	1100	SPSS	140796
110	Hamid	Inf.Systems	1100	COBOL	220796
190	Rashid	Finance	1200	INVESTMT	200697
150	Hussain	Marketing	1000	SPSS	190697
150	Hussain	Marketing	1000	SYSANAL	200797

(ii) **Second Normal Form (2 NF):** A relation is in second normal form (2 NF) if it is in 1 NF and every non-key attribute is fully functionally dependent on the primary key. More precisely:

"To be in 2 NF, every non-key attribute must depend on the key and all parts of the key".

A table (relation) will be in 2NF if any of the following conditions apply.

(a) The primary key consists of only one attribute.
(b) No non-key attributes exist in the relation.
(c) Every non-key attribute is functionally dependant on the full set of primary key attributes.

Now consider the table STUDENT (which is in 1 NF).There are a lot of redundancies in this table, so it is not an acceptable stage. In shorthand notation, it is expressed as:

STUDENT(<u>STUD-</u>ID,NAME,DEPT,MONFEE,CRSNO,CDTE)

The functional dependencies in this relation are the as follows:

STUD-ID ⟶ NAME,DEPT,MONFEE
STUD-ID,CRSNO ⟶ CDTE

The primary key in ii above is the composite key:

STUD-ID + CRSNO.

Therefore, the non-key attributes NAME,DEPT and MONFEE are functionally dependent on part of the primary key (STUD-ID) but not on CRSNO.

A partial functional dependency exists when one or more non-key attributes (such as NAME) are functionally dependant on part (but not all) of the primary key.

The partial functional dependency in the above table creates redundancy in that table, which results in certain anomalies when the table is updated. i.e.,

(i) Insertion Anomaly: To insert a row for the table, we must provide the values for both STUD-ID and CRSNO.

(ii) Deletion Anomaly: If we delete a row for one student, we lose the information that the student completed a course on a particular date.

(iii) Modification Anomaly: If a students monthly fee changes, we must record the change in multiple rows (for students, who have completed more than one course).

To convert a relation to 2 NF, we decompose the relation (having redundant data) into two relations that satisfy one of the conditions described above.

Now, splitting the relation (STUDENT) , we will get the following two relations, namely STUDENT1 and COURSE. This step is done to get rid of the redundant data) The two tables are shown below:

TABLE 1: STUDENT1(STUD-ID,NAME, DEPT, MONFEE)

STUDENT1

STUD-ID	NAME	DEPT	MONFEE
100	Ahmad	Marketing	1000
140	Nazir	Accounting	1200
110	Hamid	Inf.Systems	1100
190	Rashid	Finance	1200
150	Hussain	Marketing	1000

This relation (table) satisfies condition 1 stated above), thus it is in 2 NF.

TABLE 2: COURSE(STUD-ID,CRSNO,CDTE)

COURSE

STUD-ID	CRSNO	CDTE
100	SPSS	190696
100	SURVEYS	100796
140	TAXACCT	120897
110	SPSS	140796
110	COBOL	220796
190	INVESTMT	200697
150	SPSS	190697
150	SYSANAL	200797

Note: The attributes STUD-ID and CRSNO have been concatenated to make a unique key for the relation.

This relation (table 2) satisfies condition 3 above), thus it is in 2 NF.

These two relations are free of anomalies now.

(iii) Third Normal Form (3 NF)

A relation is in third normal form (3 NF) if it is in 2 NF and no transitive dependencies exist:

What is a Transitive Dependency? It is a functional dependency in a relation between two (or more) non-key attributes.

A more precise definition for 3 NF is:"A non-key attribute must not depend on any other non-key attribute" or if a non-key attribute's value can be obtained simply by knowing the value of another non=key attribute, the relation is not in 3 NF.

Consider a relation as follows:

SALES(CUSTNO,NAME,SALESMAN,REGION)

Where CUSTNO is the primary key.

The following functional dependencies exist in the relation.

(a) *CUSTNO -------> NAME,SALESMAN*

(b) *SALESMAN -------> Region (since each salesman is assigned a unique region)*

Notice that SALES is in 2 NF, because the primary kay consists of a single attribute (CUSTNO). However, there is a transitive dependency, because REGION is functionally dependent on SALESMAN which in turn is functionally dependent on CUSTNO. As a result, there are update anomalies in relation SALES.

CUSTNO	NAME	SALESMAN	REGION
8023	AAAA	Ahmad	South
9167	BBBB	Bashir	West
7924	CCCC	Ahmad	South
6837	DDDD	Khalid	East
9596	EEEE	Bashir	West
7018	FFFF	Munir	North

Figure : A relation with Transitive dependency

The Anomalies:

(i) Insertion Anomaly: A new salesman (Abid), assigned to the North region can not be entered until a customer has been assigned to that salesman (since a value of CUSTNO must be provided to insert a row in the table(relation).

(ii) Deletion Anomaly: If customer number 6837 is deleted from the relation, we lose the information that salesman Khalid is assigned to the east region.

(iii) Modification Anomaly: If salesman Ahmad is re-assigned to the east region, several rows must be changed to reflect the fact (two rows in this case).

These anomalies arise as a result of the transitive dependency. This problém (the transitive dependency) can be removed by de-composing the relation SALES into two relations as shown below:

SALE 1

CUSNO	NAME	SALESMAN
8023	AAAA	Ahmad
9167	BBBB	Bashir
7924	CCCC	Ahmad
6837	DDDD	Khalid
8596	EEEE	Bashir
7018	FFFF	Munir

SMAN

SALESMAN	REGION
Ahmad	South
Bashir	West
Khalid	East
Munir	North

SALE1(CUSTNO,NAME,SALESMAN)

SMAN(SALESMAN,REGION)

Now, both the relations (SALE1 & SMAN) are in 3 NF, since no transitive dependency exist. We can verify that the anomalies that exist is SALES are not present in SALE1 and SMAN.

Note: that SALESMAN which is the determinant in the transitive dependency in SALES, became the primary key in SMAN. SALESMAN is also a foreign key in SALE1

Exercise 4c

1. Fill in the blanks:
   (i) Entity integrity constraint states that the ___________ can not be null.
   (ii) _________ key must refer to the primary key in another table or it must be null.
   (iii) Normalization is the process of converting ____________ structures into simple and stable structures.
   (iv) A(n)______________ is a partial relationship between attributes of an entity.
   (v) During the first normal form ________________ groups are removed.
   (vi) To be in 2NF, a relation must be in _____________.
   (vii) In 3NF, no ________________ dependency exists.
   (viii) A(n) ____________ exists when one or more non-key attributes are functionally dependant on part of the primary key.
   (ix) When a new record is added in a relation, it may cause _________ anomaly.
   (x) Referential integrity is a constraint on ____________ key value.

2. Select the correct option:
   (i) In 3NF, which form of dependency is removed?
   a) functional b) non-functional
   c) associative d) transitive

   (ii) In relational database, a table is also called a:
   a) tuple b) relation
   c) file d) schema

   (iii) In 3NF, a non-key attribute must not depend on a(n):
   a) non-key attribute b) key attribute
   c) composite key d) sort key

   (iv) Different attributes in two different tables having same name are referred to as:
   a) synonym b) homonym
   c) acronym d) mutually exclusive

   (v) Every relation must have a:
   a) primary key b) candidate key
   c) secondary key d) composite key

3. Mark as True or False
   - (i) Normalization is the process of converting complex data structures into simple data structures.
   - (ii) A relation is decomposed to convert it from 1NF to 2NF.
   - (iii) The primary key can not be a composite key.
   - (iv) In 2NF, every non-key attribute must depend on the key attribute.
   - (v) A relationship involving three relations is known as a ternary relationship.
   - (vi) A database anomaly leads the database to an inconsistent state.
   - (vii) Partial dependencies are removed in 3NF.
   - (viii) A relation may have multiple primary keys.
   - (ix) In relational database, no relation can exist in isolation.
   - (x) The database is normalized to avoid certain database anomalies.

4. What is meant by data integrity? What are the two types?

5. What do we do to attain entity integrity?

6. Define referential integrity. How can it be achieved?

7. Explain the following terms:
   - (i) Synonym
   - (ii) Homonym
   - (iii) Redundancy
   - (iv) Mutual Exclusiveness of data

8. What is normalization? How it can be used to bring the database in a consistent state?

9. When is a relation in first normal form? Explain with example.

10. What are the conditions for a relation to be in second normal form? Give example.

11. Define transitive dependency. How it can be removed? Explain with the context of normalization.

12. What are the database anomalies? Briefly discuss insertion, deletion and modification anomalies.

13. What anomalies arises due to transitive dependency? Discuss briefly.

14. Define functional dependency? How partial dependencies effect a relation?

15. Convert the ER diagram you have designed in the previous exercise for the admission system of your college to relational database. Also normalize the relations up to third-normal form.

[illegible]

INTRODUCTION TO MICROSOFT ACCESS

Chapter 5

5.1 OVERVIEW

Microsoft Access is one of the most popular and powerful DBMS. It has many built in features to assist you in constructing database and viewing information. MS-Access is much more involved and is a more genuine DBMS than other programs such as Microsoft Works. Microsoft Access is a Relational Database Management System (RDBMS) that you can use to store and manipulate large amount of information. It is easy to understand and its graphical interface helps to create queries, forms, and reports. In other words, even inexperienced programmers can use *Access* to turn a stack of invoices, a card file of customer names, a ledger, and an inventory list into a relational database that makes entering, updating, and reporting information as easy as clicking a button.

MS-Access offers more than just pretty interface for learning how to manage data. You' ll find following benefits and more from using Access:

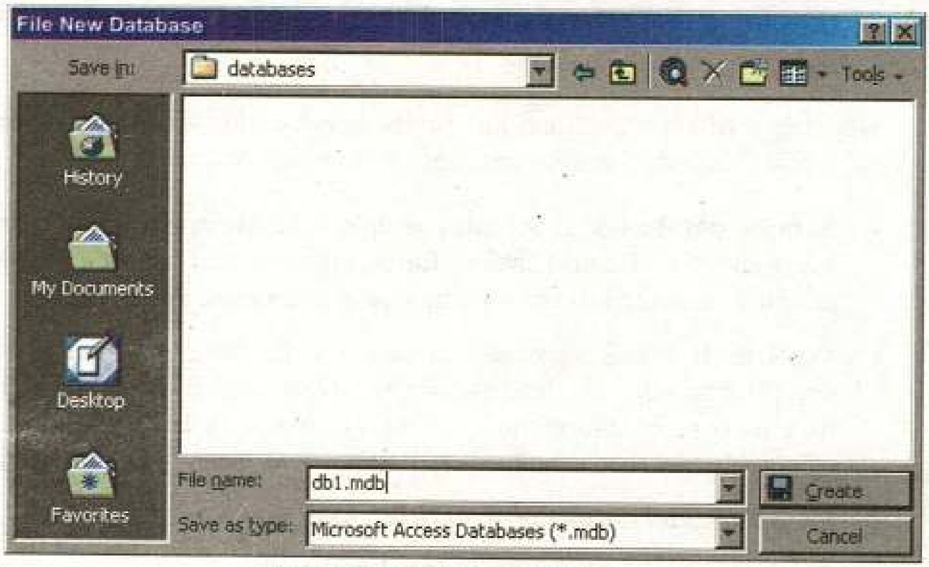
- **Sample databases:** It includes sample database applications to assist you learn about real-world tables, forms, queries, and reports, and how they are interconnected to form a database management system.
- **Wizard:** It makes very easy to create a database. You can choose from several examples of databases in the *Database Wizard* for such storage uses as contact information, inventory control, a ledger, and so on. You can create and then modify these databases to meet your own needs.
- **Keys to understand the structure:** After you have decided how to create and relate tables, you can easily view all the relationships in the database with the graphical interface in the Relationship Window. This makes one of the toughest parts of relational database design much easier and more manageable.
- **Microsoft Office integration:** You can use access with Word, Excel, and other office application to create mail merges, charts, and other helpful uses for your data.
- **Easier programming:** You can use relatively simple code with macros to automate repeated tasks, or you can try more complex and flexible code

with VBA. Access provides graphical shortcuts and hints to help writing easier code.

- **Common Standards:** It uses standards that help applications scale up to work within larger environments. Access uses objects and SQL (Structure Query Language) to make its code from the adaptable to other applications.
- **Redundancy:** *MS-Access* allows you to store, retrieve, sort, analyze, and print information contained in the database. Data may be manipulated without data *redundancy* by defining relationships between sets of data. Databases are often used for product data. *Redundancy* means duplication of data in multiple files. It wastes the storage media of computer.

5.1.1 Creating New Database

Once Access has been accessed, choose the File New Database command from the menu at the top of the screen in order to create a new database. The database window is displayed. Choose the appropriate directory and drive. Then enter an eight-character name for the new database file and click "OK".



File name: db1.mdb
Save as type: Microsoft Access Databases (*.mdb)

Fig.5.1 Defining the File name a Database

When you start up Ms-Access, you get a dialog box asking if you want to open an existing database or create a new one.

- Create a New Database.
- Create a File New Database using wizard
- Open an existing database

Warning - If you have previously created a database, and then create it again using the same name, you will overwrite any work you have done.

5.1.2 Create a database using the Database Wizard

The *Database Wizard* guides you through process of the creation a database; it includes choosing a database template, selecting fields, making customizations, adding pictures, and the database.

(i) When Microsoft Access first starts up, a dialog box is displayed with options to create a new database or to open an existing one. When this dialog box appears, select *Access Database Wizards*, *pages*, and *projects* and then click *OK*. Microsoft Access starts up, when you click the *New Database* on the toolbar.

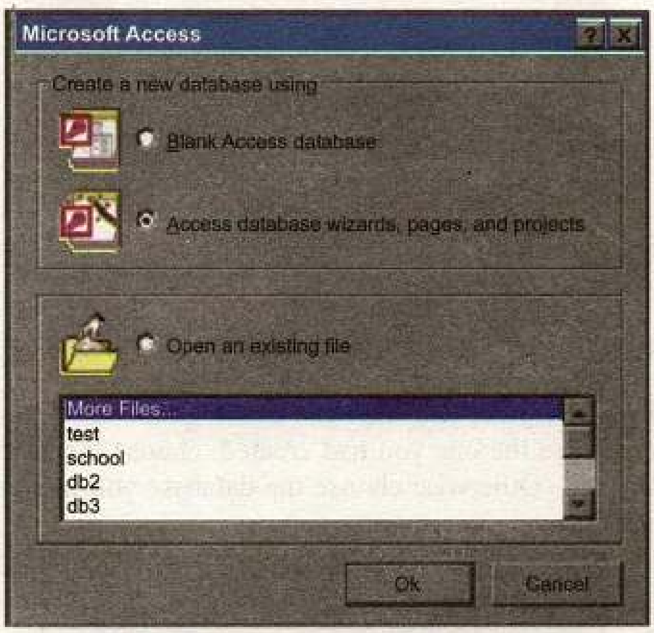


Fig.52 Creating a database using database wizard

(ii) On the *Databases* tab, double-click the icon for the kind of database you want to create.

(iii) Specify a name and location for the database.

(iv) Click *Create* to start defining the new database

5.1.3 Create a database without using the Database Wizard

(i) If you don't want to use the database wizard, you can create a database, you first start the Microsoft Access, a dialog box is displayed with options to create a new database or open an existing one. when this dialog box is displayed.

(ii) Click on the *Blank Access Database*, and then click *OK*.

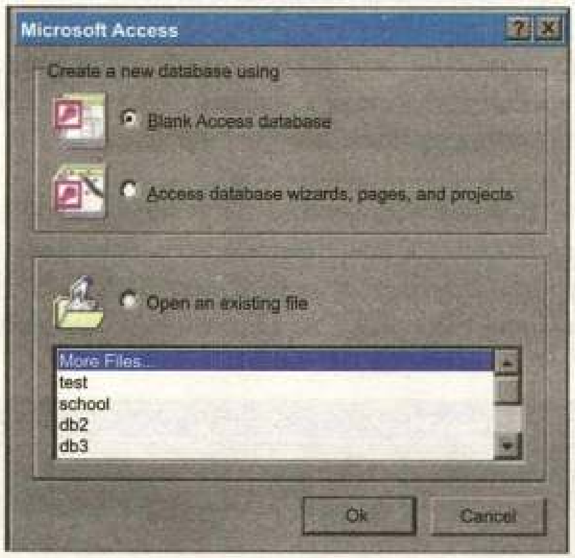


Fig.5.3 Creating a Database without using Database wizard

(iii) Specify a name and location for the database and click **Create**.

5.1.4 Opening Existing Databases

(i) The white box gives you the most existing databases you have used. If you do not see the one you had created, choose the More Files option and click *OK*. Otherwise choose the database you had previously used and click *OK*.

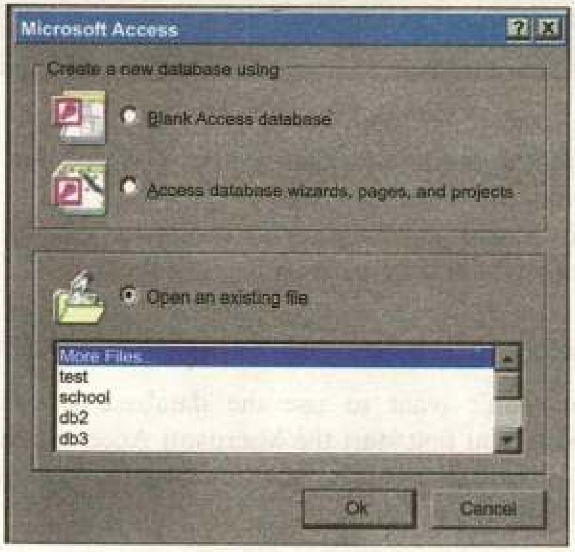


Fig. 5.4 Opening a existing Database

5.1.5 Exiting Microsoft Access

When you finish working in Access, be sure to exit the program properly to avoid damaging your database.

(i) *Click* on *File*. The File menu will appear.

(ii) *Click* on *Exit*. Access will close and you will return to the Widows desktop.

OR

Click on the *Close button* as an alternate way to exit Access in only one step.

5.2 MS-ACCESS APPLICATION WINDOW

The *Access Application Window* follows the standard layout of all Microsoft Application .

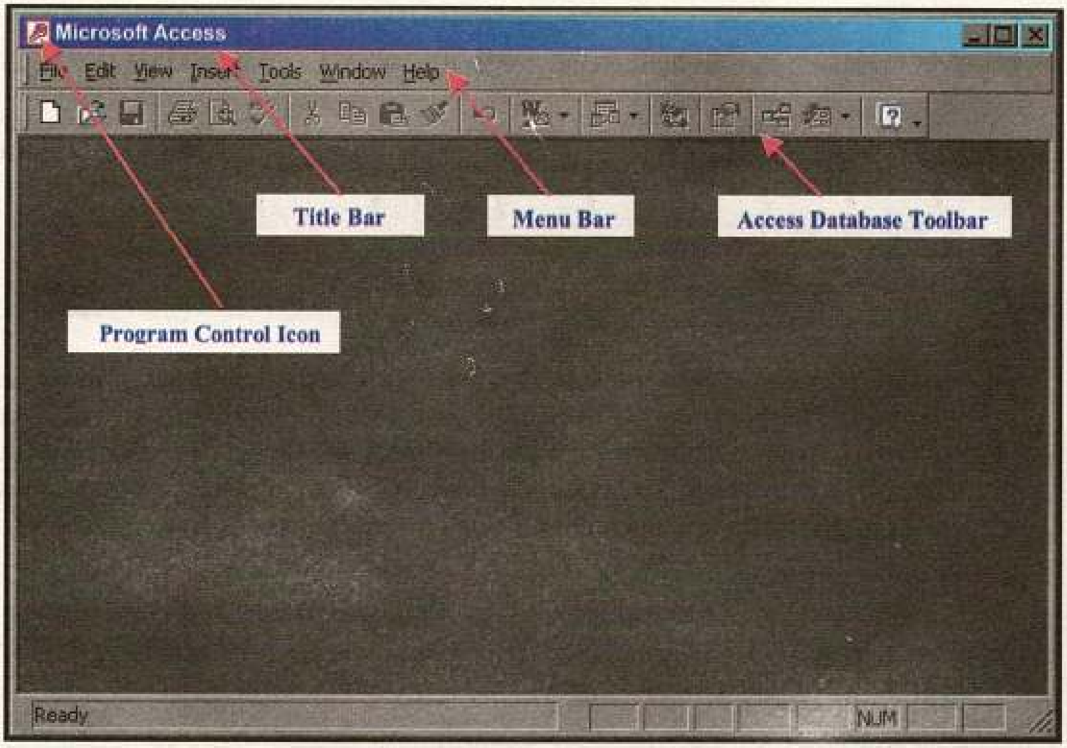


Fig.5.5. Defining Ms-Access Application Windows

Title Bar

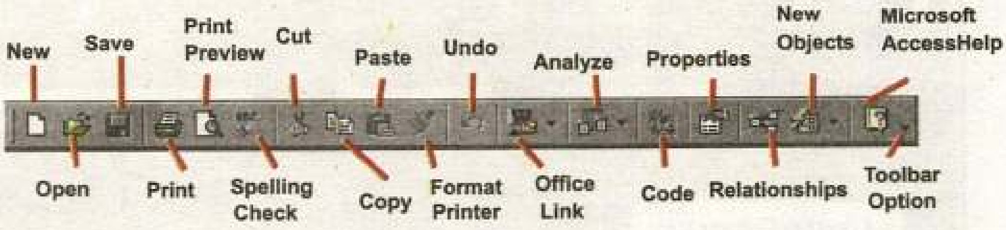
The Title bar identifies the database application you are running in *Microsoft Access*. On the left side of the title bar is program *Control Icon* . If you click this, a menu of the commands to control the Access window is displayed. You can also use the minimize *Maximize* *Restore* *close* buttons on the right side of the Title bar to control the *MS-Access* Window.

Menu Bar

Each word on the menu bar represents a different menu. Each menu contains the commands you use to activate features of Access. If a command is also found on a toolbar then the icon representing the command is displayed in the menu too. This make recognizing commands easier.

Toolbars

Toolbars contain icon button that are shortcuts to the commands in the menu.



Toolbars make it easy to use the program's most common features and functions. The buttons on the toolbar can change depending on the objects selected. When you click on a button on the toolbar, as you open individual tables, queries, forms, and reports within that database, the toolbars change. For example, when you open a table, the table datasheet toolbar appears.

Scroll Bar

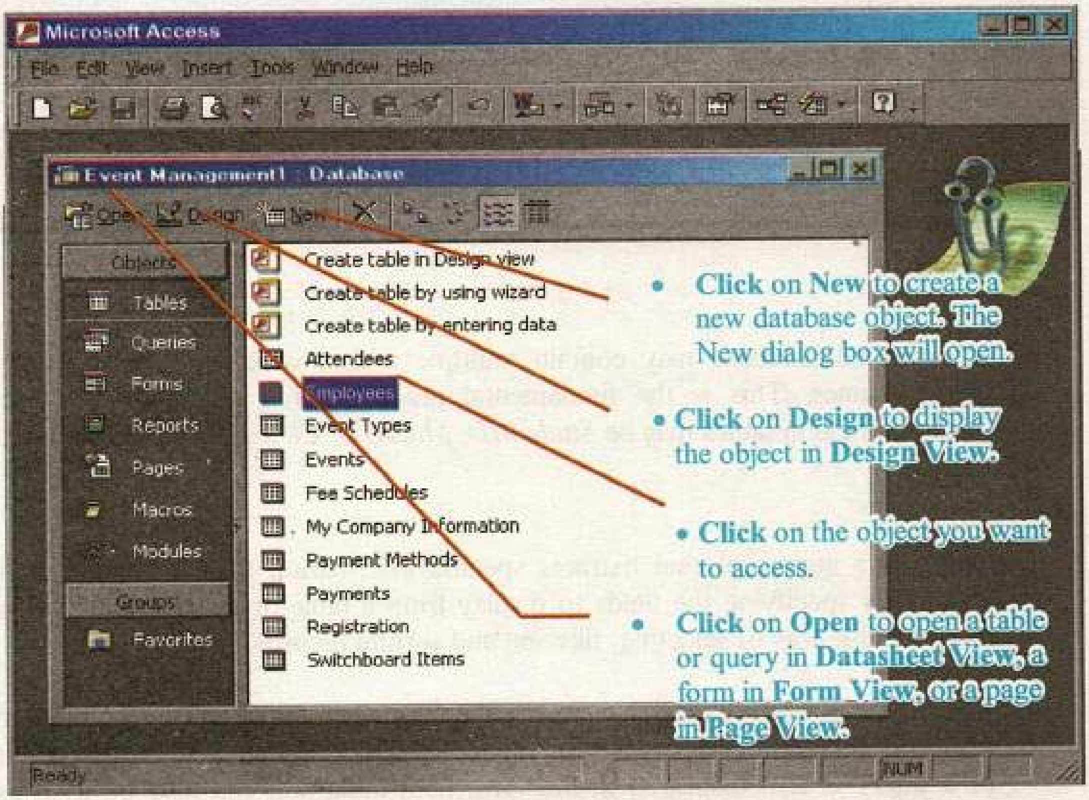
Scroll bars are used to move around the window if its contents do not fit on-screen. You can scroll around the sheet by clicking the scroll arrows at either end of the scroll bar or by dragging the scroll button in the scroll bar.

Status Bar

The status bar display while you are working on an object within a database. CAPS and NUM buttons on keyboard show whether and respectively are on.

5.3 DATABASE WINDOW

The Database Window organizes all of the objects in the database. The default table Listing provides ways to create tables and lists all of the tables in the database. The left side of the *MS-Access* database window includes seven buttons, each corresponding to one of the seven objects that make up an Access database. A database is essentially an organized collection of data. In an Access database, you collect data into the tables by using forms, query tables to analyze their content, create report based on the tables and queries, and design data access page to view your Access data from the web. As an advanced user, you may create macros *(A macro is used to perform the same sequence of steps or automating tasks repeatedly.)* to automate takes or modules (A *module contains an object that stores VBA code.*) to create database applications using Access.



- **Click** on **New** to create a new database object. The New dialog box will open.
- **Click** on **Design** to display the object in **Design View.**
- **Click** on the object you want to access.
- **Click** on **Open** to open a table or query in **Datasheet View,** a form in **Form View,** or a page in **Page View.**

Fig.5.6 Defining the Database window

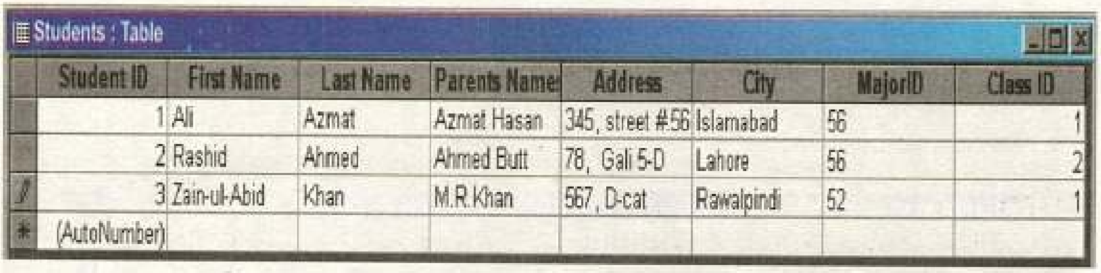
5.4 DATABASE OBJECTS

A Ms-Access database consists of various components called the objects. The database objects are used to store data and to retrieve data from database. The major database objects are:

(i)	Tables	(ii)	Queries
(iii)	Forms	(iv)	Reports

Tables

The most important object of a database is a Table. The data is stored in tables of database. A table is a collection of related data organized in rows and columns. Each row consist a record, and each record consists of columns. The row is divided into columns called field containing different data values of a particular record. A table having recodes of students is given blow. Each record contains fields; *StudentID*, First Name, Parents *Name*, *Address*, *City*, *MajorID*, and *ClassID*.



	Student ID	First Name	Last Name	Parents Name	Address	City	MajorID	Class ID
	1	Ali	Azmat	Azmat Hasan	345, street #56	Islamabad	56	1
	2	Rashid	Ahmed	Ahmed Butt	78, Gali 5-D	Lahore	56	2
✎	3	Zain-ul-Abid	Khan	M.R.Khan	567, D-cat	Rawalpindi	52	1
*	(AutoNumber)							

Fig.5.7 Creating a Database without using Database wizard

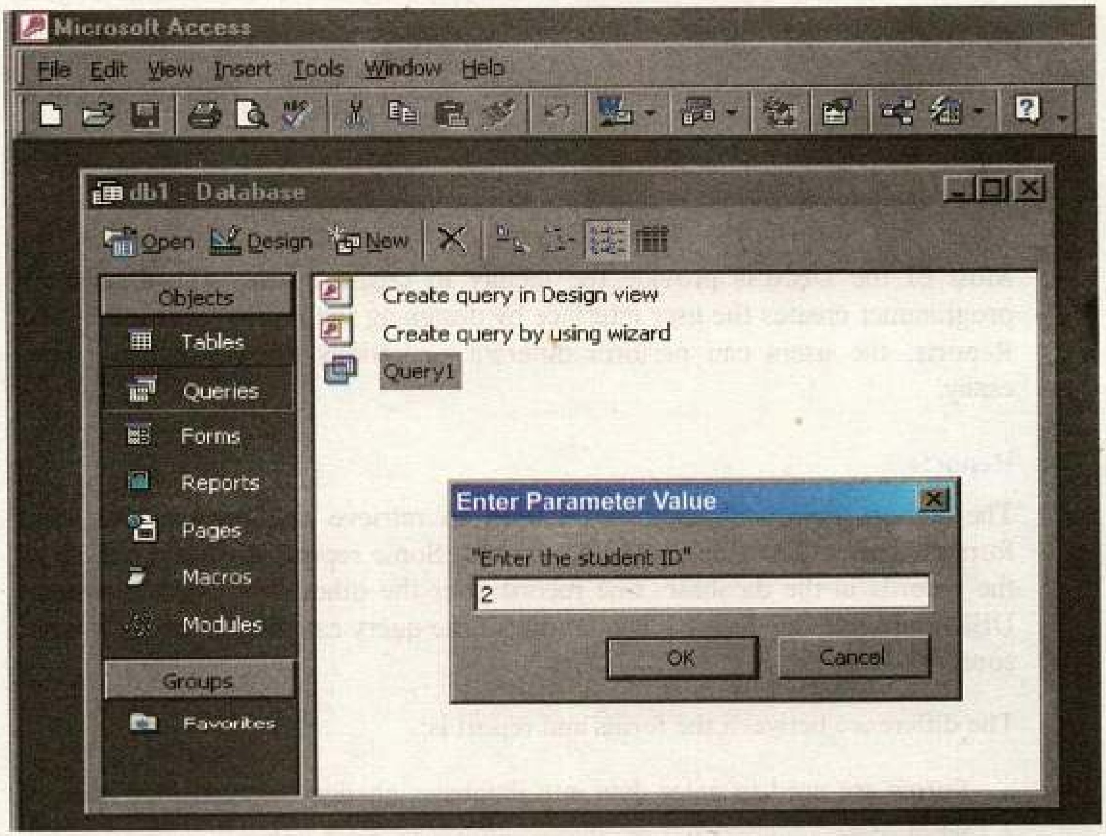
A relational database may contain multiple tables, which are identified by unique names. This is the fundamental property of a relational database. Common field in tables may be *StudentID, MajorID, ClassID* etc.

Queries

Query is a statement that extracts specific information from database. It is created by specifying the fields to display from a table or another query. It is more flexible way of selecting, filtering and sorting records.

The user can also change data in the database that fulfils certain criteria. In addition, queries allow to perform calculations of different fields. The output of a query is also displayed in the form of a table and can also be used as source of records for Forms and Reports.

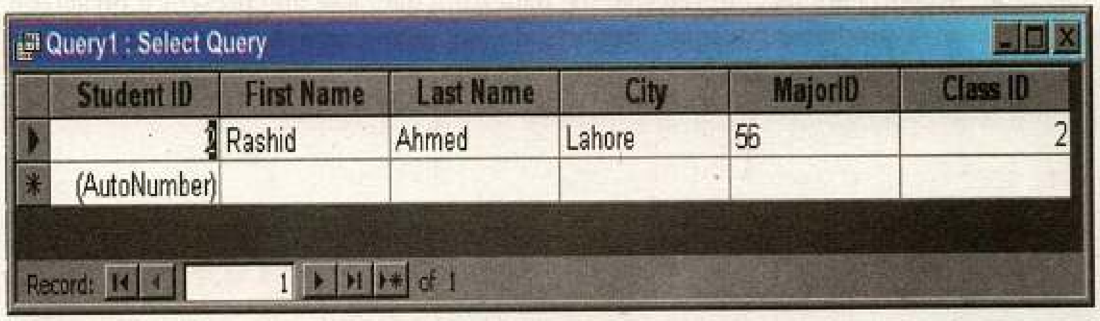
The query allows you to view and analyse data in many different ways. Technically, a query is a stored question or request. You design a query in design view to extract certain information from the database.



"Enter the student ID"

2

Fig.5.8 Creating query in a Database



	Student ID	First Name	Last Name	City	MajorID	Class ID
▶	2	Rashid	Ahmed	Lahore	56	2
*	(AutoNumber)					

Fig.5.9 Select Query

The information appears in Datasheet view, which looks exactly like Datasheet view for a table. The difference between a datasheet for a table and a datasheet for a query is that the query's datasheet can combine information from multiple tables.

Forms

The Form object of database is used to enter data into databases, edit data and view data from database. You can add, update, and delete records in your table by using a form. Form provide:

- an easy method for entering and editing data in tables. Thus the user does not have to work directly with tables.

- facilities to display data retrieved from database tables.

Most of the DBMSs provide the facility to create Forms. The application programmer creates the user interface by designing the Forms. In this way and Reports, the users can perform different operations on the database very easily.

Reports

The Report object of database is used to retrieve and present data in a formatted way. The Report can be printed. Some reports are simply a list of the records in the database, one record after the other. Most of the popular DBMS provide this facility. The output of the query can also be given as input source to Reports.

The difference between the forms and report is:

- Forms are used to enter data into database, change data and view data of databases.
- Reports are used to retrieve the data from database and present it on screen in a predefined format. Reports do not allow user to change data or to enter data into database.

Exercise 5c

1. Fill in the blank:
   - (i) IDE stands for ___________________.
   - (ii) ___________ is basically a computerized record keeping system.
   - (iii) RDBMS stands for ________________.
   - (iv) The __________ object is used to store data in a database.
   - (v) The _________ object is used to retrieve data from a database.
   - (vi) A field with ______ data type is automatically incremented by Access each time a new record is entered.
   - (vii) Each row of a table is divided into columns called ___________.
   - (viii) Each row of a table representing a set of information is called ___________.
   - (ix) The window that is used to display, enter and edit data on the screen is called________.
   - (x) A database consists of ______ major database objects, which are used to store and retrieve data to and from the database.

2. Multiple Choice questions:
   - (i) A database consists of various components called the:
     - a) Tool. b) Properties.
     - c) Entities d) Object
   - (ii) Which of the following object of database is used to retrieve data from database?
     - a) Queries b) Forms
     - c) Reports d) Tables
   - (iii) The output of a query is in the form of a:
     - a) Table b) Form
     - c) Report d) Query
   - (iv) Which of the following object is used to retrieve data from database and present in a formatted way?
     - a) Report b) Form
     - c) Table d) Query
   - (v) Microsoft Access saves the database with the extension:
     - a) .mdb b) .msdb
     - c) .madb d) None of them
   - (vi) A record is a complete set of ______ field.

a) Distinct	b) Related
c) Designed	d) All of them

(vii) In Access, the structure of a table is created in ________ view.

a) Design View	b) Datasheet View
c) a and b both	d) None of them

3. Write T for true and F for false statement.
   - (i) An IDE simplifies the tasks of creating and using a database.
   - (ii) The major objects of database are five.
   - (iii) Forms are provided by database management system to generate reports.
   - (iv) An integrated development environment is an interface that is used by database designers and application programmers to create database applications.
   - (v) To view data in an Access table, the table is displayed in design view.
   - (vi) RDBMS stands for Relational Database Management System.
   - (vii) A request to extract data from a database is called report.
   - (viii) Database design plays an important role in achieving the goals of efficiency, speed and consistency.
   - (ix) The table can be displayed in two views in Access. There are Design view and Datasheet view.
   - (x) The Window in a database IDE that is used to display, enter and edit data on the screen is called form.
4. Define the database objects are used to store and retrieve data.
5. Explain the procedure for creating a new database in Access.
6. Differentiate between Toolbar and Menu bar.
7. What is meant by RDBMS?
8. What is an IDE?
9. Define the use of Toolbar in Microsoft Access.
10. What are the advantages of using a Microsoft Access IDE?
11. Write a procedure to open an existing database file.
12. How is Microsoft Access started or loaded?
13. Differentiate between Form and Report.
14. Define the use of status bar and title bar in Microsoft Access.
15. Describe the Database Window in Microsoft Access

Chapter 6

TABLE AND QUERY

6.1 OVERVIEW

In a relational database the data is stored in tables. The table is the fundamental concept of relational databases. It is also called relation. It is the foundation of every Relational Database Management System is a database object called table.

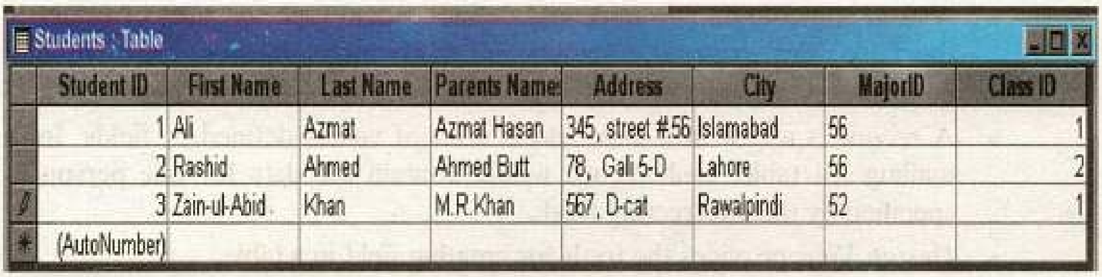
Tables are grids that store information in a database similar to the way an Excel worksheet stores information in a workbook. Access provides three ways to create a table for which there are icons in the Database Window. Double-click on any of the icons to create a table.

Every database consists of one or more tables which store data. Each table has its own unique name and consists of columns and rows. It is a very convenient way to store information.

The columns in a table (also called table fields) have their own unique names and have a pre-defined data type. The field can be a primary key, an index defined on it and it can have certain default value.

The table columns describe the data types, whereas the table rows contain the actual data. Here is an example of a simple database table, containing students' data. The first row, listed in bold, contains the names of the table columns:

Table: **Students**



Student ID	First Name	Last Name	Parents Name	Address	City	MajorID	Class ID
1	Ali	Azmat	Azmat Hasan	345, street #.56	Islamabad	56	1
2	Rashid	Ahmed	Ahmed Butt	78, Gali 5-D	Lahore	56	2
3	Zain-ul-Abid	Khan	M.R.Khan	567, D-cat	Rawalpindi	52	1
(AutoNumber)							

Characteristics of Tables

The tables of a relational database have following characteristics:

- Each cell of the table contains only one value.
- Each column has a distinct name, which is the name of the attribute (field) it represents.
- The order of the columns is immaterial.
- Each row represents a record.
- Each row is distinct; there are no duplicate rows.
- The order of rows is immaterial.

Using a separate table for each entity means that you store that data only once, which makes your database more efficient, and reduces data entry errors. Tables form the foundation of an Access database structure.

Degree of a Relation

The number of fields in a relation is called the degree of a table. Once the table has been created, its degree usually dose not changes, e.g. a table with five fields has a degree of 5.

Cardinality of a Relation

The number of record in a relation is called the cardinality of the relation. The cardinality of a relation changes as new records are added or existing records are deleted, e.g. a table with 50 records has a cardinality of 50.

A Basic Terminology

These words are used often in Assess so you will want to become familiar with them before using the program and this book.

- A *database* is a collection of related data (or record).
- An *object* is a component in the database such as a table, query, form, or macro.
- A *table* is a group of related data organized in fields (columns) and records (rows). By using a common field in two tables, the data can be linked. Many tables can be stored in a single database.
- A *field* is a column in a table and defines a data type for a set of values in the table. For example a mailing list table might include fields for first name, last name, address, city, state, zip code, and telephone number.
- A *record* is a row in a table and is a set of values defined by fields. In a mailing list table, each record would contain the data for one person as specified by the intersecting fields.
- *Design View* provides the tools for creating field in a table.

- *Datasheet View* allows you to update, edit, and delete information from a table.

6.1.1 Access IDE

IDE stand for **I**ntegrated **D**evelopment **E**nvironment. It is an interface that is used to create a database. An IDE makes the using of database simple, manageable for end users who may not have a complicate programming knowledge of the database system.

Microsoft Access is an example of a database management system. The access IDE simplifies the task of creating, designing good-looking screens with features (i.e. text boxes, list boxes, button, dialog boxes etc.). It provides the facilities for searching, sorting, and retrieving the data.

6.1.2 Starting Microsoft Access

You can build a database in two ways by using the Database Wizard, or by opening an empty database and building all your objects with wizards or from scratch.

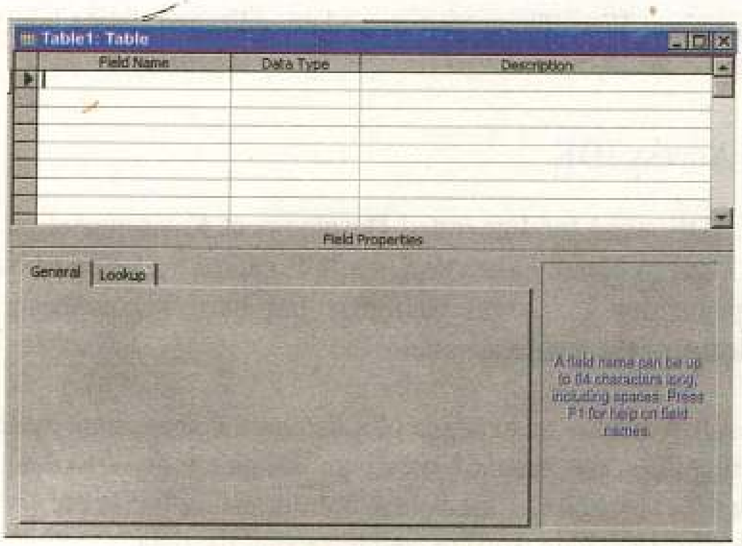
(i) Double click on the Microsoft Access icon on the desktop if its icon in the desktop.

OR

(ii) Click on Start

6.2 TABLE DESIGN VIEW

Design View allows you to define fields in the table before adding any data to datasheet.



A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Fig. 6.1 Table Design View

6.2.1 Datasheet View

When you open a table or query using the database window, it will be displayed in datasheet view. A table or query is opened in Datasheet view to perform different operations on the data in the table such as displaying data, adding new data, searching data etc.

The Datasheet view is like worksheet. When table is opened in Datasheet view, the field names are displayed as header of columns and each row contains a complete record.

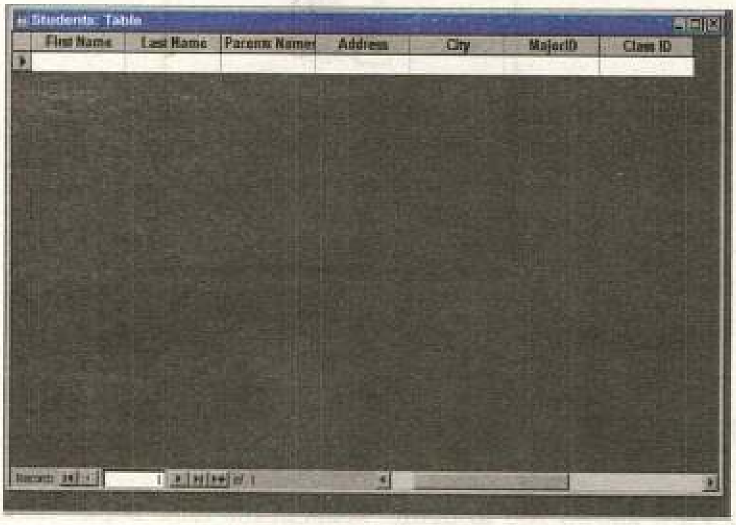


Fig. 6.2 Datasheet View

Switching Views

- To switch views form the datasheet (spreadsheet view) to the design view, simply click the button in the top-left corner of the Access program.

Datasheet View	Design View
▦	✎
Displays the view, which allows you to enter raw data into your database table.	Displays the view, which allows you define fields, data-types, and descriptions into your database table.

Table 6.1

6.3 TABLE CREATION

You can open or create a table in several ways in Microsoft Access i.e.,

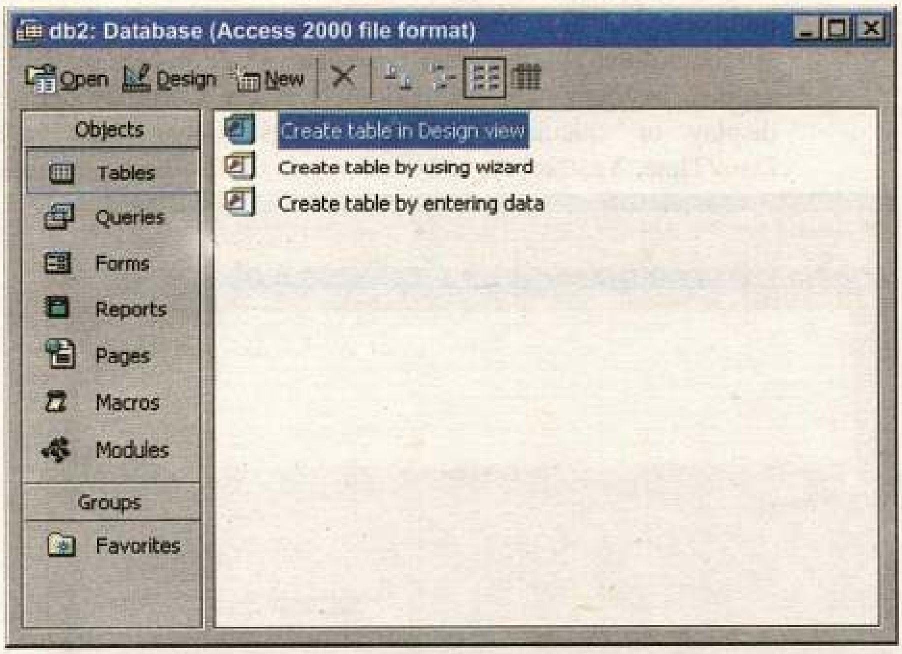

db2: Database (Access 2000 file format)
Open
Design
New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Create table in Design view
Create table by using wizard
Create table by entering data

Fig. 6.3 Creating Database through Different way.

(i) Creating database in Design view.
(ii) Creating database by wizard.
(iii) Creating database by entering data.

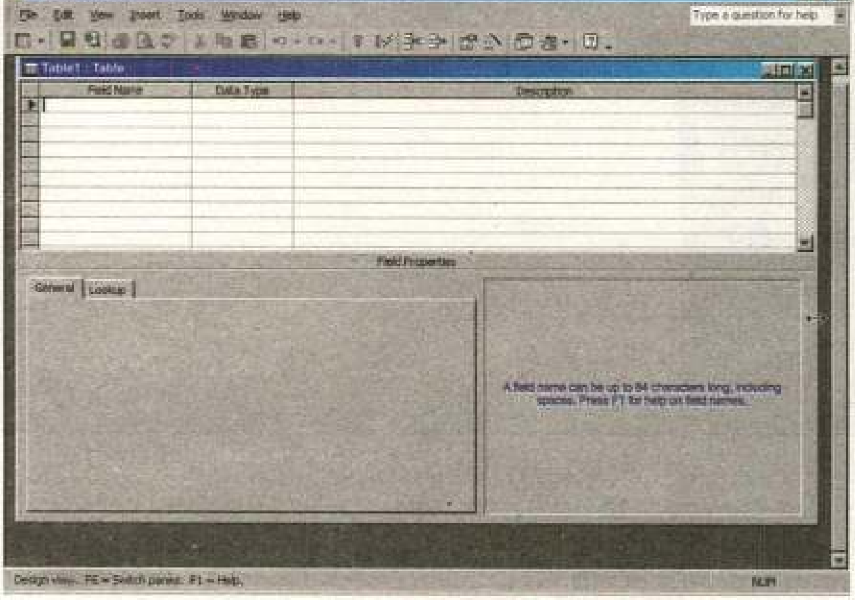
Creating database in Design View

- *Click* on the *Table* object from the list of database object.
- In Database Window, *Double-Click* on "*Create table in Design view*"; a table window in *Design View* is appeared to design the structure of table.

OR

- *Click* on *New*, in Database Window. The New Table dialog box will open.
- *Click* on *Design View Option.*
- *Click* on *OK.* A blank table will open in *Design View.*
- Define each of the fields in your table.
- Under the Field Name column, enter the categories of your table.
- Under Data Type column, enter the type you want for you categories.

  - The attribute of a variable or field determines what kind of data it can hold. For example, in a Microsoft Access database, the Text and Memo field data types allow the field to store either text or numbers, but the Number data type allow the field to store numbers only. Number data type fields store numerical data that is used in mathematical calculations. Use the Currency data type to display or calculate currency values. Other data types are Date/Time, Yes/No, Auto Number, and OLE object (Picture).

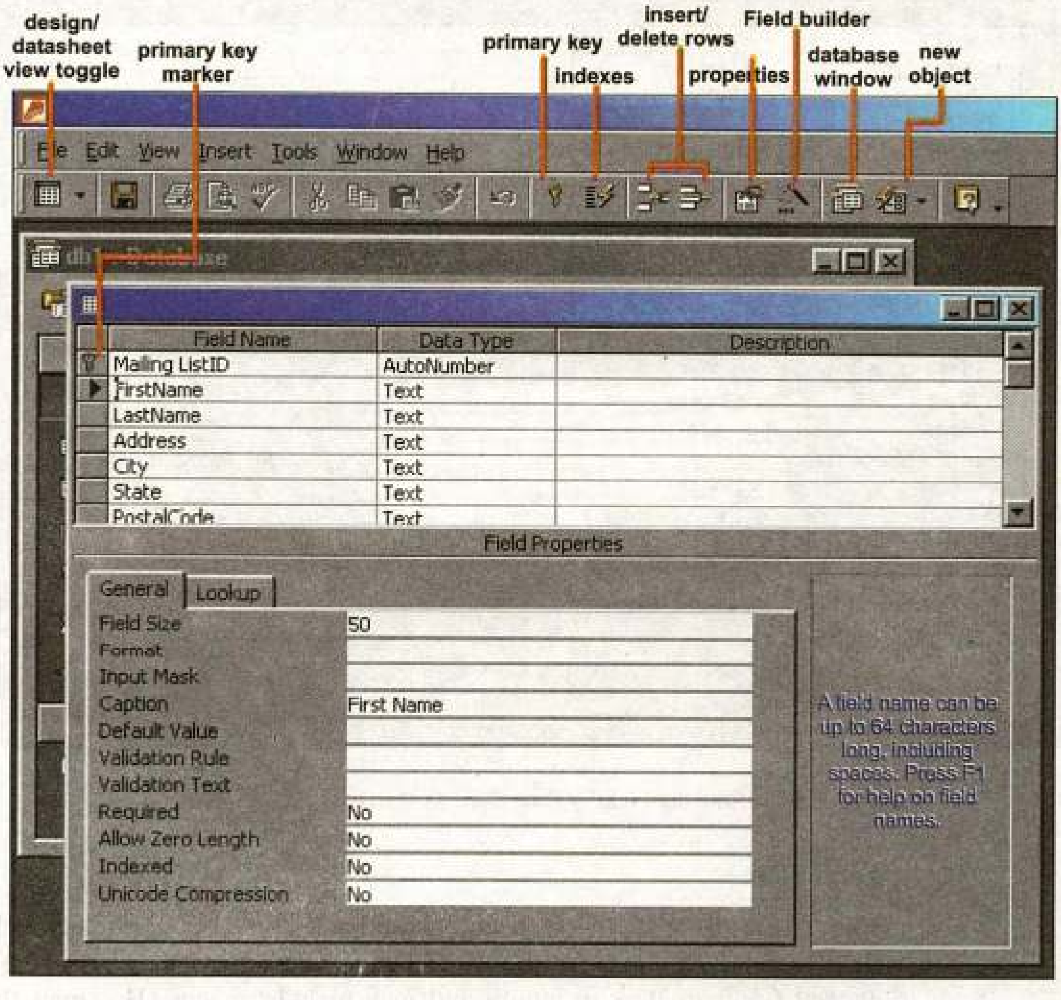


A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Design view. F6 = Switch panes. F1 = Help.

Fig.6.4 Creating Database in Design View

- Under the Description column, enter the text that describes the field. (This field is optional).



Field Name	Data Type	Description
Mailing ListID	AutoNumber	
FirstName	Text	
LastName	Text	
Address	Text	
City	Text	
State	Text	
PostalCode	Text	

Field Size	50
Format	
Input Mask	
Caption	First Name
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	No
Indexed	No
Unicode Compression	No

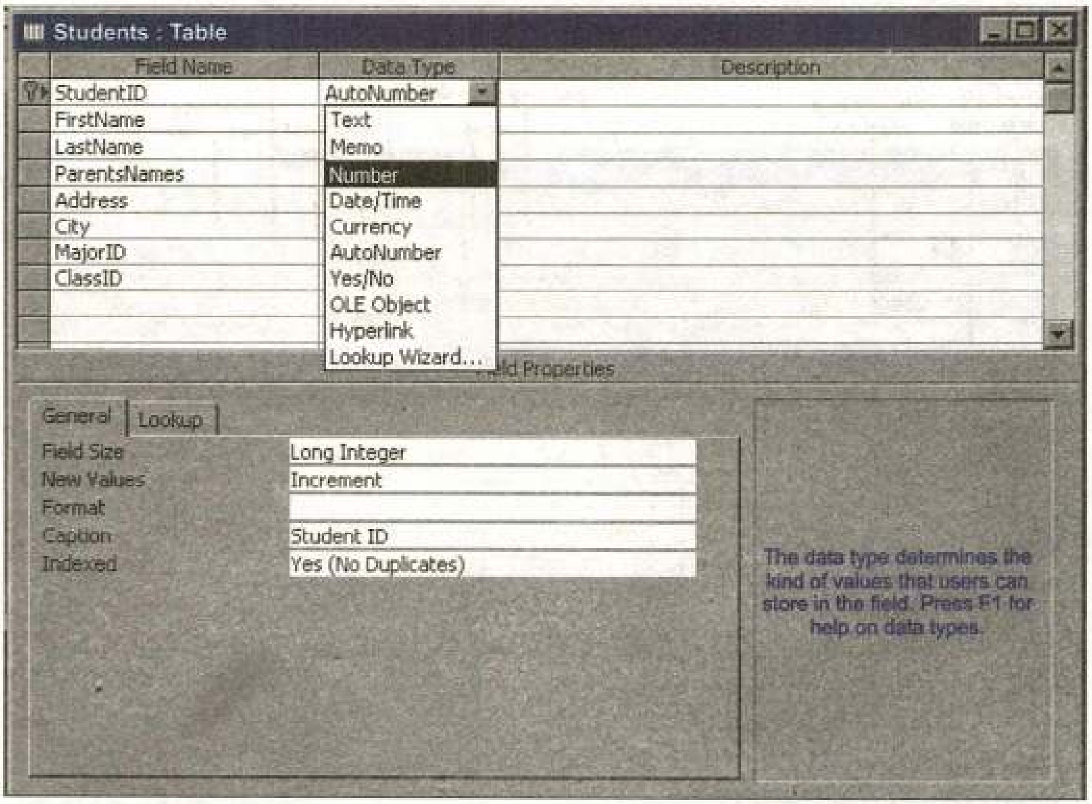
A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Fig. 6.5 Define the fields in Database Design View

Defining Field Properties in Design View

The properties of each field can be set in design View. The window is divided into two parts: a top pane for entering the field name, data type, and an optional description of the field, and a bottom pane for specifying field properties.

To assign the Primary Key, select the field and click the Primary Key button in the toolbar. You can set the remaining properties in the Table Window's lower pane. The following properties are defined briefly.


Students : Table

Field Name	Data Type	Description
StudentID	AutoNumber	
FirstName		
LastName		
ParentsNames		
Address		
City		
MajorID		
ClassID		

Field Properties

General	
Field Size	Long Integer
New Values	Increment
Format	
Caption	Student ID
Indexed	Yes (No Duplicates)

The data type determines the kind of values that users can store in the field. Press F1 for help on data types.

Fig. 6.6 Define the Field' s Data Type in Database Design View

(i) Field Name

This is the name of the field and should represent the contents of the field such as "Name", "Address", "Final Grade", etc. The name cannot exceed 64 characters in length and may include spaces (However, this is not considered a good practice).

(ii) Data Type

Before you start creating a new table in Access, you first consider how you want to break down the information you are organizing into smaller units of data in the table. Dividing the data into units of information is the process of determining the fields. Each field will be assigned a unique field name. Each field is also assigned a data type. Following are the data types available in Ms. Access:

- **Text** - The default type, text type allows any combination of letters and numbers up to a maximum of 255 characters per field record.
- **Memo** - A text type that can store more than 64,000 characters and is used for detailed descriptive feilds0.
- **Number** – This data type is used to store numbers that are used in mathematical calculations. Several number field sizes are available. The most useful are summarized in table 6.3.
- **Date/Time** - A *Date*, *Time*, or combination of both can be specified in this field.
- **Currency** - Monetary values that can be set up to automatically include a dollar sign ($) and correct decimal and comma positions.
- **Auto Number** - When a new record is created, Access will automatically assign a unique integer to the record in this field. From the General options, select Increment if the numbers should be assigned in order or random if any random number should be chosen. Since every record in a datasheet must include at least one field that distinguishes it from all others, this is a useful data type to use if the existing data will not produce such values.
- **Yes/No** - Use this option for *True/False*, *Yes/No*, *On/Off*, or other values that must be only one of two.
- **OLE Object** - An *OLE* (Object Linking and Embedding) object is a sound, picture, or other object such as a Word document or Excel spreadsheet that is created in another program. Use this data type to embed an *OLE* object or link to the object in the database.
- **Hyper link** - A hyperlink will link to a website, or another location in the database. A hyperlink address have up to four parts: the text that is displayed in the field; the path to a file or URL; a subaddress which is a location in the file or page in the web site; and the text that is displayed as the tooltip. The data

consists of up to four parts each separated by the pound sign (#):DisplayText#Address#SubAddress#ScreenTip. The Address is the only required part of the string. Examples:

Internet hyperlink example: Google Home Page#http://www.google.com#

Database link example: #c:\My Documents\database.mdb#MyTable

Description (optional)

Enter a brief description of what the contents of the field are.

Field Properties

Select any pertinent properties for the field from the bottom pane. Properties for each field are set from the bottom pane of the Design View window.

- **Field Size** is used to set the number of characters needed in a text or number field. The default field size for the text type is 50 characters. If the records in the field will only have two or three characters, you can change the size of the field to save disk space or prevent entry errors by limiting the number of characters allowed. Likewise, if the field will require more than 50 characters, enter a number up to 255. The field size is set in exact characters for Text type, but options are give for numbers:
  - **Byte** - Positive integers between 1 and 255
  - **Integer** - Positive and negative integers between -32,768 and 32,767
  - **Long Integer (default)** - Larger positive and negative integers between -2 ,147,483,648 to 2,147,483,647.
  - **Single** - Single-precision floating-point number
  - **Double** - Double-precision floating-point number
  - **Decimal** - Allows for Precision and Scale property control

- **Format** conforms the data in the field to the format specified in the format property. For text and memo fields, this property has two

parts that are separated by a semicolon. The first part of the property
is used to apply to the field and the second applies to empty fields.

Text and memo format.

Text Format			
Format	**Datasheet Entry**	**Display**	**Explanation**
@@@-@@@@	1234567	123-4567	@ indicates a required character or space
@@@-@@@&	123456	123-456	& indicates an optional character or space
<	HELLO	hello	< converts characters to lowercase
>	Hello	HELLO	> converts characters to uppercase
@\!	Hello	Hello!	\ adds characters to the end
@;"No Data Entered"	Hello	Hello	
@;"No Data Entered"	(blank)	No Data Entered	

Table 6.2

- **Number format.** Select one of the preset options from the drop down menu or construct a custom format using symbols explained below:

Number Format			
Format	**Datasheet Entry**	**Display**	**Explanation**
###,##0.00	123456.78	123,456.78	0 is a placeholder that displays a digit or 0 if there is none.
$###,##0.00	0	$0.00	# is a placeholder that displays a digit or nothing if there is none.
###.00%	123	12.3%	% multiplies the number by 100 and added a percent sign

Table 6.3

- **Currency format.** This formatting consists of four parts separated by semicolons:format for positive numbers; format for negative numbers; format for zero values; format for Null values.

Currency Format	
Format	**Explanation**
$##0.00;($##0.00)[Red];$0.00;"none"	Positive values will be normal currency format, negative numbers will be red in parentheses, zero is entered for zero values, and "none" will be written for Null values.

Table 6.4

- **Date format.**

In the table below, the value "1/1/01" is entered into the datasheet, and the following values are displayed as a result of the different assigned formats.

Date Format		
Format	**Display**	**Explanation**
dddd","mmmm d","yyyy	Monday, January 1, 2001	dddd, mmmm, and yyyy print the full day name, month name, and year
ddd","mmm ". " d", '"yy	Mon, Jan. 1, '01	ddd, mmm, and yy print the first three day letters, first three month letters, and last two year digits
"Today is " dddd	Today is Monday	
h:n:s: AM/PM	12:00:00 AM	"n" is used for minutes to avoid confusion with months

Table 6.5

- **Yes/No**

Fields are displayed as check boxes by default on the datasheet. To change the formatting of these fields, first click the Lookup tab and change the Display Control to a text box. Go back to the General tab choices to make formatting changes. The formatting is designated in three sections separated by semicolons. The first section does not contain anything but the semicolon must be included. The second section specifies formatting for Yes values and the third for No values.

Yes/No Format	
Format	**Explanation**
;"Yes"[green];"No"[red]	Prints "Yes" in green or "No" in red

Table 6.6

- **Default Value**

  There may be cases where the value of a field might usually be the same for all records. In this case, a changeable default value can be set to prevent typing the same value numerous times.

- **Indexes**

  Creating indexes allow Access to query and sort records faster. To set an indexed field, select a field that is commonly searched and change the Indexed property to *Yes* (*Duplicates OK*) if multiple entries of the same data value are allowed or *Yes* (*No Duplicates*) to prevent duplicates.

- **Field Validation Rules**

  Validation Rules specify criteria for the data entered in the worksheet. A customized message can be displayed to the user when data that violates the rule is entered. Click the expression builder ("**...**") button at the end of the Validation Rule box to write the validation rule. Examples of field validation rules include **<> 0** (not allow zero values in the record), and??? (only data strings with three characters in length).

- **Input Masks**

  An input mask controls the value of a record and sets it in a specific format. They are similar to the Format property, but instead display the format on the datasheet before the data is entered. For example, a telephone number field can be formatted with an input mask to accept ten digits in the form "(555) 123-4567". The blank field would look like (___) ___-____. An an input mask can be applied to a field by following these steps:

  - In design view, place the cursor in the field that the input mask will be applied to.
  - Click in the white space following **Input Mask** under the **General** tab.
  - Click the "**...**" button to use the wizard or enter the mask such as, (@@@) @@@-@@@@, into the field provided. The following symbols can be used to create an input mask from scratch:

Input Mask Symbols	
Symbol	**Explanation**
A	Letter or digit
0	A digit 0 through 9 without a + or - sign and with blanks displayed as zeros
9	Same as 0 with blanks displayed as spaces
#	Same as 9 with +/- signs
?	Letter
L	Letter A through Z
C or &	Character or space
<	Convert letters to lower case
>	Convert letters to upper case

Table 6.7

Primary Key

Every record in a table must have a primary key that differentiates it from all other record in the table. In some cases, it is only necessary to designate an existing field as the primary key if you are certain that every record in the table will have a different value for that particular field. A social security number is an example of a field whose values will only appear once in a database table.

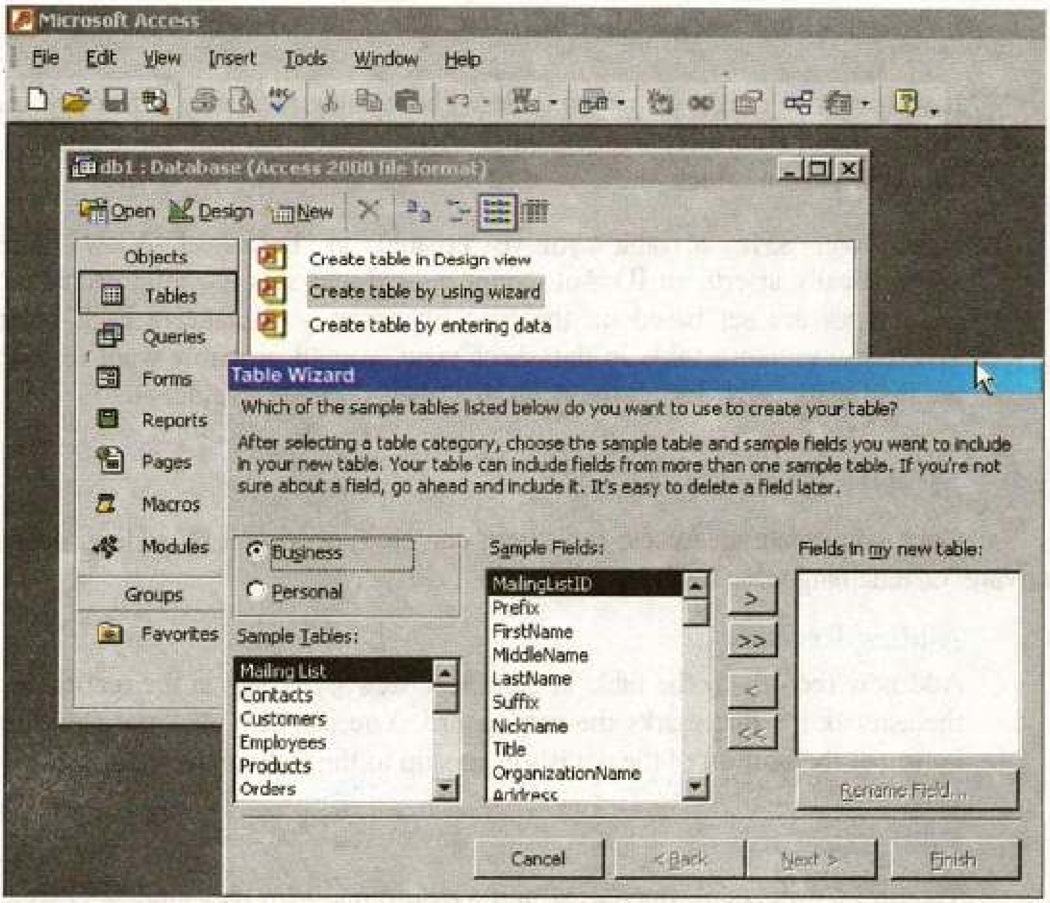
Designate the primary key field by right-clicking on the record and selecting *Primary Key* from the shortcut menu or select *Edit|Primary Key* from the menu bar. The primary key field will be marked with a key image to the left. To remove a primary key, repeat one of these steps.

If none of the existing *fields* in the table produces unique values for every record, a separate field must be added. Ms.- Access will prompts you to create this type of field the first time you save the table if a primary key field has not been assigned. The field is named "ID" and the data type is "autonumber". Since this extra field serves no purpose to you as the user, the autonumber type automatically updates whenever a record is added so there is no extra work on your part. You may also choose to hide this column in the datasheet as explained at later stage in this book.

Creating database through using Wizard

The Access Table Wizard offers an easy way to create tables. Access includes numerous table templates that you can use to create both business and personal database tables. The Wizard can help you create common types of tables, including those that store mailing lists, recipes, investments, video collection etc.

- *Click* on the *Table button* in the main database window.
- *Double-click* on the *"Create table by using wizard"* option. The table Wizard will appear.
- Next, choose the specific field for the table. *Click* on the *Business* or *Personal* option button. Sample business or personal table will appear in the *Sample Tables* scroll box.
- Scroll down the Sample Tables scroll box until you see the table you want to use. *Click* on this *Sample Table*. Sample field, based on the table you choose, will appear in the *Sample Fields* scroll box.
- *Click* on the Sample field from the *Sample Fields* scroll box that you want to include in your table. The field will be selected.



Which of the sample tables listed below do you want to use to create your table?

After selecting a table category, choose the sample table and sample fields you want to include in your new table. Your table can include fields from more than one sample table. If you're not sure about a field, go ahead and include it. It's easy to delete a field later.

Fig. 6.7 Creating Database by using Wizard

- The sample field will move to the field, click on *right arrow button* > or click on the *right double-arrow button* >> for all fields. You can easily to remove a single field by clicking on *left arrow button* < or click on double arrow button << for removing all fields. You can also rename a field after you move it to the Field in my new table scroll box.
- Enter a name for your table in text box. Set the *Primary Key*, it is an important concept in relational database.
- Click on Next if your new table isn't related to any existing tables. If you want to relate your new table to an existing one, Access can create the relationship for you.

Note: A table name can have up to 64 characters including letters, numbers, and spaces.

Create Table by entering data

If you want to create your own table without using the Table Wizard, you can create one in Datasheet View, then let Access analyse it and automatically sea data types and a primary key.

When you save a table you've created in Datasheet View, Access automatically inserts an ID AutoNumber field and sets it as the primary key. Data types are set based on the type of entries you make in each column. When you create a table in datasheet view, you 'll probably want to modify the field names. By default the fields are labelled Field1, Field2, etc.

6.4 MODIFYING A TABLE

Once you create an Access table, you can easily modify it by adding, deleting, moving, or renaming table fields.

Adding Records

Add new records to the table in datasheet view by typing in the record beside the asterisk (*) that marks the new record. You can also click the new record button at the bottom of the datasheet to skip to the last empty record.

Editing Records

To edit records, place the cursor in the record that is to be edited and make the necessary changes. Use the arrow keys to move through the record grid. The previous, next, first, and last record buttons at the bottom of the datasheet are helpful in manoeuvring through the datasheet.

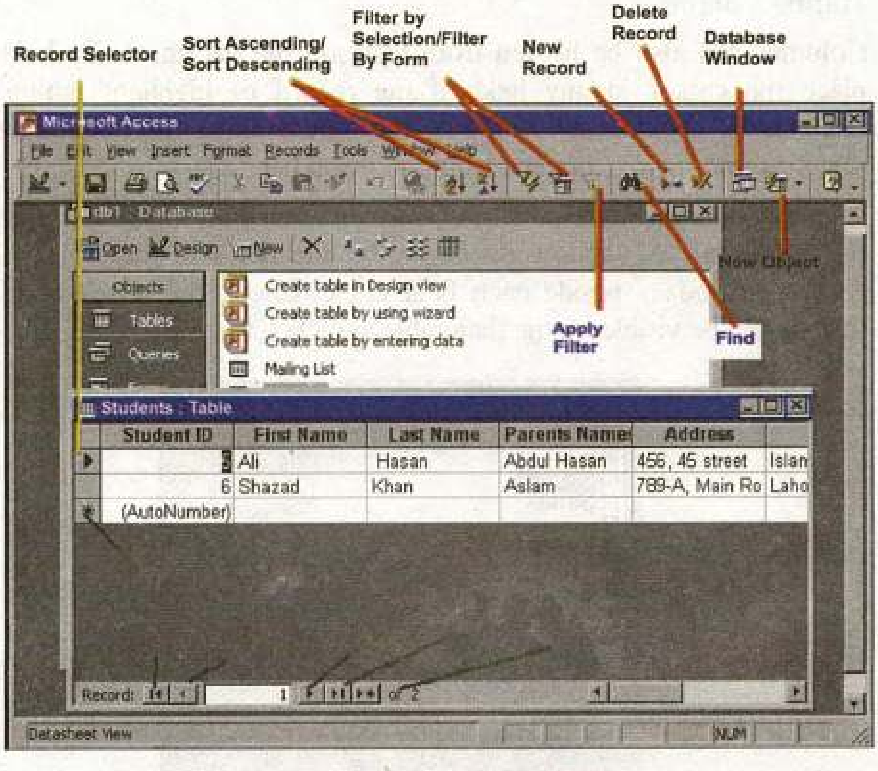
Deleting Records

Delete a record on a datasheet by placing the cursor in any field of the record row and *select Edit|Delete* Record *from the menu bar or click the* Delete Record *button on the datasheet toolbar.*

Inserting and Deleting Fields

Although it is best to add new fields (displayed as columns in the datasheet) in design view because more options are available, they can also be quickly added in datasheet view. Highlight a column by clicking its label at the top of the datasheet and select *Insert|Column* from the menu bar. The new column will be added to the left of the selected column.

Entire columns can be deleted by placing the cursor in the column and selecting *Edit|Delete Column* from the menu bar.


Record Selector
Sort Ascending/ Sort Descending
Filter by Selection/Filter By Form
New Record
Delete Record
Database Window
New Object
Apply Filter
Find
Microsoft Access
File Edit View Insert Format Records Tools Window Help
db1 : Database
Open Design New
Objects
Tables
Queries
Create table in Design view
Create table by using wizard
Create table by entering data
Mailing List
Students : Table
Record: 1 of 2
Datasheet View
NUM

Student ID	First Name	Last Name	Parents Name	Address	
5	Ali	Hasan	Abdul Hasan	456, 45 street	Islan
6	Shazad	Khan	Aslam	789-A, Main Ro	Laho
(AutoNumber)					

Fig.6.8 Defining the Datasheet View

Resizing Rows and Columns

The height of rows on a datasheet can be changed by dragging the grey sizing line between row labels up and down with the mouse. By changing the height of one row, the height of all rows in the datasheet will be changed to the new value.

Column width can be changed in a similar way by dragging the sizing line between columns. Double click on the line to have the column automatically fit to the longest value of the column. Unlike rows, columns on a datasheet can be of different widths. More exact values can be assigned by selecting *Format|Row Height* or *Format|Column Width* from the menu bar.

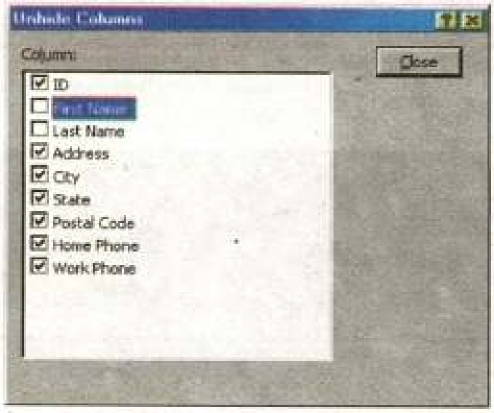
Freezing Columns

Similar to freezing panes in Excel, columns on an Access table can be frozen. This is helpful if the datasheet has many columns and relevant data would otherwise not appear on the screen at the same time. Freeze a column by placing the cursor in any record in the column and select *Format|Freeze* Columns from the menu bar. Select the same option to unfreeze a single column or select *Format|Unfreeze* All Columns.

Hiding Columns

Columns can also be hidden from view on the datasheet. To hide a column, place the cursor in any field of the record or highlight multiple adjacent columns by clicking and dragging the mouse along the column headers, and select *Format|Hide Columns* from the menu bar.

To show columns that have been hidden, select *Format|Unhide Columns* from the menu bar. A window displaying all the fields in the table will be listed with check boxes beside each field name. Check the boxes beside all fields that should be visible on the data table and click the *Close* button.



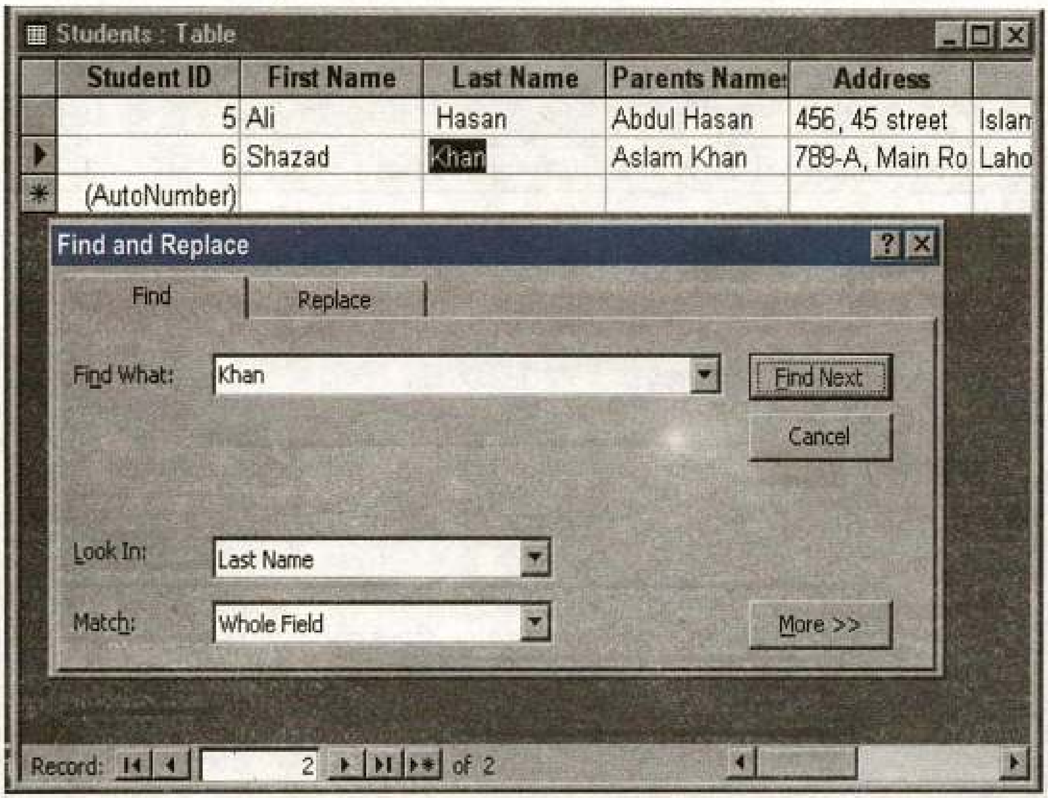
Columns:
- ✓ ID
- ☐ First Name
- ☐ Last Name
- ✓ Address
- ✓ City
- ✓ State
- ✓ Postal Code
- ✓ Home Phone
- ✓ Work Phone

Fig.6.9 Shows the Hiding and Unhidden Columns

Finding Data in a Table

Data in a datasheet can be quickly located by using the *Find* command.

- Open the table in datasheet view.
- Place the cursor in any record in the field that you want to search and select *Edit|Find...* from the menu bar.
- Enter the value criteria in the *Find What*: box.
- From the *Look In*: drop-down menu, define the area of the search by selecting the entire table or just the field in the table you placed your cursor in during step 2.



Students : Table

Student ID	First Name	Last Name	Parents Name	Address	
5	Ali	Hasan	Abdul Hasan	456, 45 street	Islan
6	Shazad	Khan	Aslam Khan	789-A, Main Ro	Laho
(AutoNumber)					

Find and Replace

Find What: Khan

Look In: Last Name

Match: Whole Field

Record: 2 of 2

Fig.6.10 Finding Data in a Table

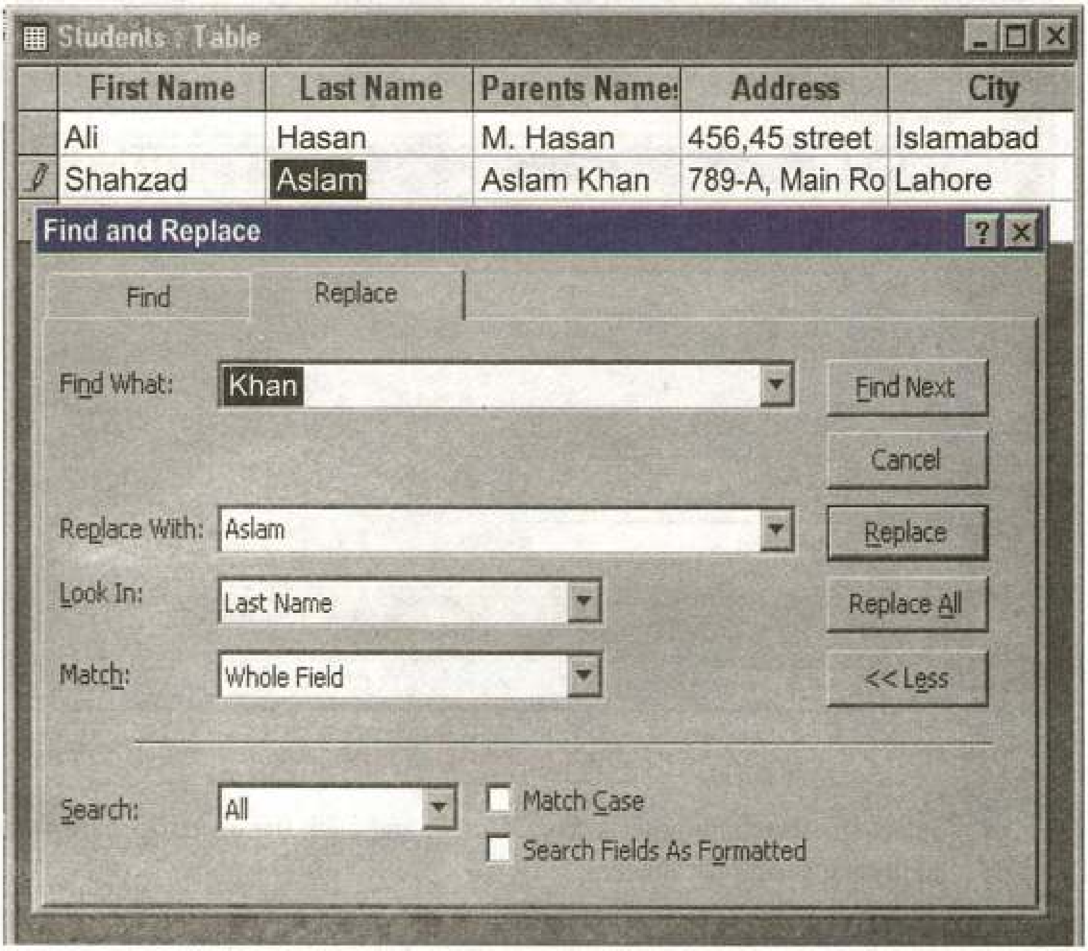
- Select the matching criteria from *Match*: and click the *More* >> button for additional search parameters.
- When all of the search criteria are set, click the *Find Next* button. If more than one records meet the criteria, keep clicking *Find Next* until you reach the desired record.

Replace

The replace function allows you to quickly replace a single occurrence of data with a new value or to replace all occurrences in the entire table.

- Select *Edit|Replace...* from the menu bar (or click the *Replace* tab if the Find window is already open).
- Follow the steps described in the Find procedure to search the data that should be replaced and type the new value of the data in the *Replace With*: box.

Click the *Find Next* button to step through occurrences of the data in the table and click the *Replace* button to make single replacements. Click *Replace All* to change all occurrences of the data in one step.



First Name	Last Name	Parents Name:	Address	City
Ali	Hasan	M. Hasan	456,45 street	Islamabad
Shahzad	Aslam	Aslam Khan	789-A, Main Ro	Lahore

Find What: Khan

Replace With: Aslam

Look In: Last Name

Match: Whole Field

Search: All

☐ Match Case

☐ Search Fields As Formatted

Fig.6.11 Replacing Data in a Table

Check Spelling and AutoCorrect

The spell checker can be used to locate spelling errors in text and menu fields in a datasheet. Select *Tools|Spelling* from the menu bar to activate the spell checker and make corrections just as you would use Word or Excel. The AutoCorrect feature can automatically correct common spelling errors such as two "initial capitals", capitalizing the first letter of the first word, and anything you define. Select *Tools|AutoCorrect* to set these features.

6.5 PRINT A DATASHEET

Datasheets can be printed by clicking the *Print* button on the toolbar or select *File|Print* to set more printing options.

6.6 TABLE RELATIONSHIPS

Relationships can be established among tables by repeating field in more than one table in this way duplication of information can be prevented in database. Follow the steps below to set up a relational database:

- Click the *Relationships* button on the toolbar.
- From the *Show Table* window (click the *Show Table* button on the toolbar to make it appear), double click in the names of the tables you would like to include in the relationships. When you have finished adding tables, click *Close*

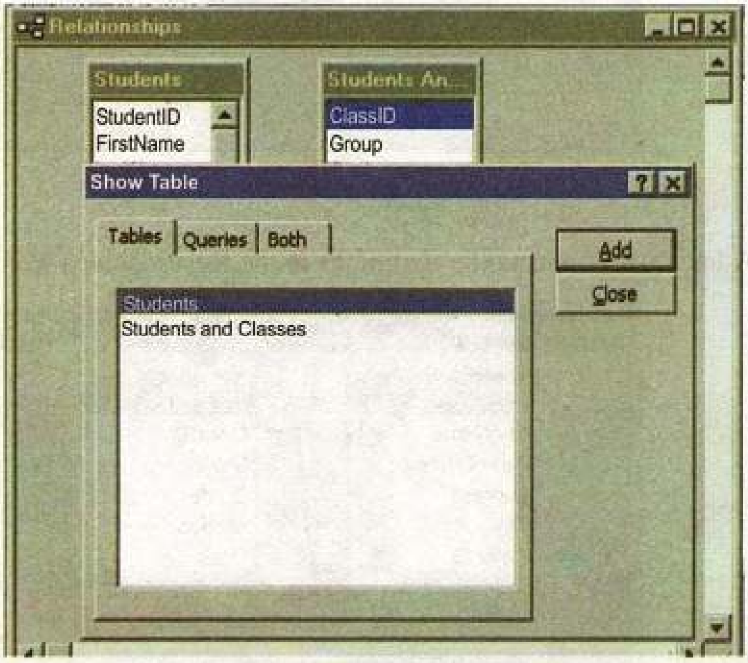


Fig.6.12 Relationship in Show Table

- To link fields in two different tables, click and drag a field from one table to the corresponding field in the other table and release the mouse button. The *Edit Relationships* window will appear. From this window, select different fields if necessary and select an option from "*Enforce Referential Integrity*" if necessary. These options give Access permission to automatically make changes to referencing tables if key records in one of the tables is deleted. Check the *Enforce Referential Integrity* box to ensure that the relationships are valid and that the data is not accidentally lost when a record is added, edited, or deleted. Click *Create* to create the link.

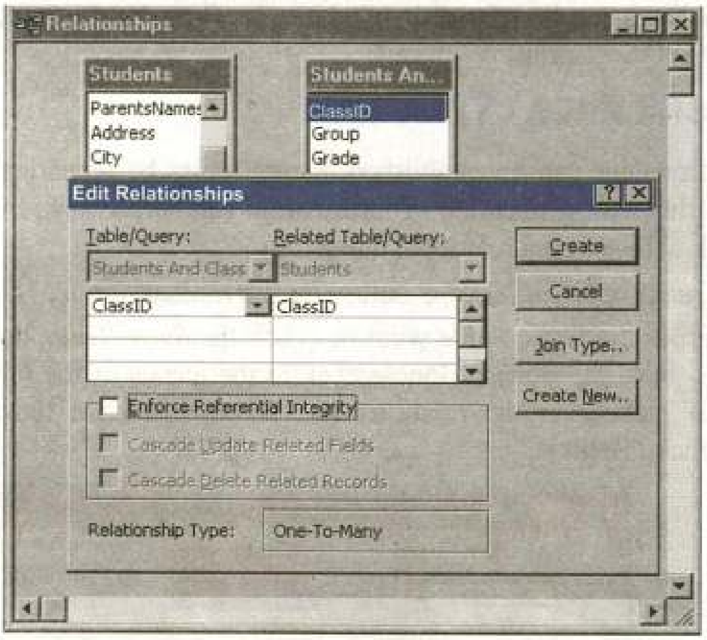


Fig.6.13 Edit Relationship Dialog Box to Define the Relationship between Student Table and Student and Classes.

- A line now connects the two fields in the Relationships window.

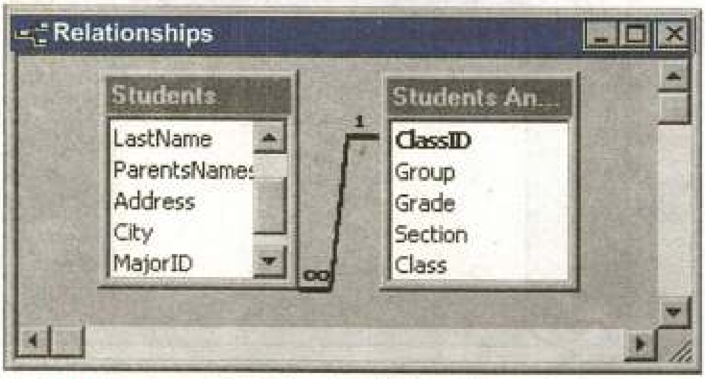
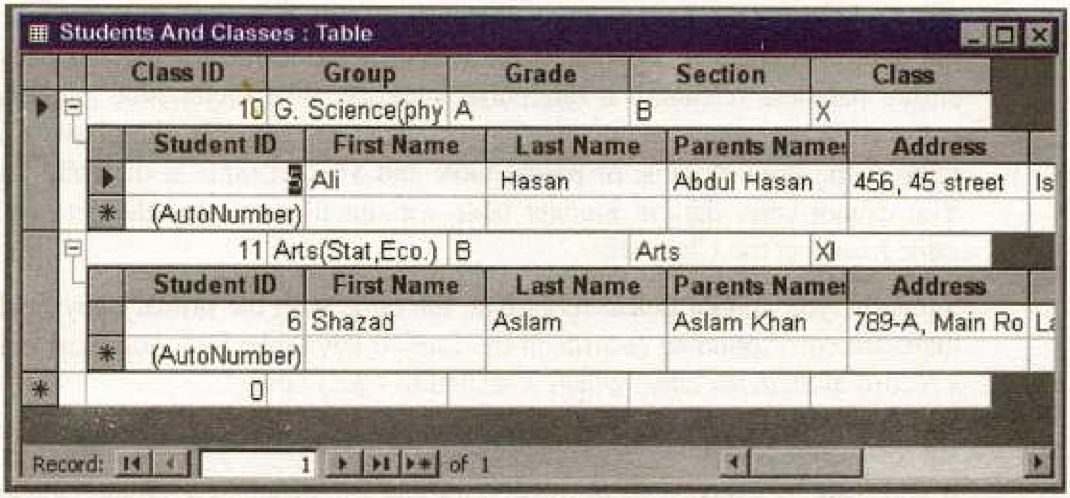


Fig.6.14 Showing the database's Relationship windows

The datasheet of a relational table will provide expand and collapse indicators to view subdatasheets containing matching information from the other table. In the example below, the *student* and *student and classes table* were related and the two can be shown simultaneously using the expand feature. To expand or collapse all subdatasheets at once, select *Format\Subdatasheet\Expand All* or *Collapse All* from the toolbar.


Students And Classes : Table

Class ID	Group	Grade	Section	Class
10	G. Science(phy	A	B	X

Student ID	First Name	Last Name	Parents Name	Address	
5	Ali	Hasan	Abdul Hasan	456, 45 street	Is
(AutoNumber)					

Class ID	Group	Grade	Section	Class
11	Arts(Stat,Eco.)	B	Arts	XI

Student ID	First Name	Last Name	Parents Name	Address	
6	Shazad	Aslam	Aslam Khan	789-A, Main Ro	La
(AutoNumber)					

Class ID	Group	Grade	Section	Class
0				

Record: 1 of 1

Fig.6.15 Showing the relationship between Student table and Student and Classes table.

Relationship and Join

Relationships are really at the center of Relational Database Design for obvious reasons. While there are many benefits to have a relational database, there's also a requirement that you have an understanding of how database design – and relationships in particular – work.

Without a relational database structure, you've got a flat-file. A big block of data – similar to an Excel sheet. While in some cases this "relationship-less" database may be exactly what the Doctor ordered, as soon as you get a substantial amount of data, or throw even a slightly complicated piece of data into the equation, you're destined for a hard time. You simply cannot access the kinds of statistics in a flat-file that you normally would with relational database design, at least not with such ease as you normally would.

Joins are what make relationships work (don't use that line in a bar, you'll get slapped – nobody likes database humor). With something like a flat-file, none of this is a concern to you. Of course, that's not necessarily good because you're probably more concerned with having hundreds or thousands more records than you actually need or can manage. When two tables start to make goo-goo eyes and you think they're ready for a relationship, it's time to make a join!

Referential Integrity

A referential Integrity constraint is a rule that maintains consistency among the rows of two tables. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in the other relation or the foreign key value must be null. It is absolutely crucial that the data contained in a database file is reliable. One method Access uses to ensure database reliability is referential integrity. When referential integrity is enforced, you cannot delete or change related records. In the above *Class table* is the primary table or parent table and *Student table* is the child table. You cannot enter data in Student table without first entering the data of the same record in the *Class table*.

Similarly, you cannot delete records in the table with the primary key field, if there are corresponding records in the foreign key table. First you must delete a record in *Students* table, which is related to *Class* table.

Cascade Update Related Fields and Cascade Delete Related Fields

If you want to override these restrictions and still maintain referential integrity you can select the Cascade Update Related Fields and the Cascade Delete Related Records check boxes in Edit Relationship dialog box. If the Cascade Update Related Fields check box is selected, whenever you change the primary key of a record in the primary table, Access automatically updates the primary key to the new value in all related records. If the Cascade Delete Related Records check box is selected, whenever you delete records in the primary table, Access automatically delete related records in the related table.

6.7 SORTING AND FILTERING

Sorting and filtering allow you to view records in a table in a different way either by reordering all of the records in the table or view only those records in a table that meet certain criteria that you specify.

Sorting

You may want to view records in a table in a different order than they appear such as sorting by date or in alphabetical order. Follow these steps to sort records in a table based on the values of one field:

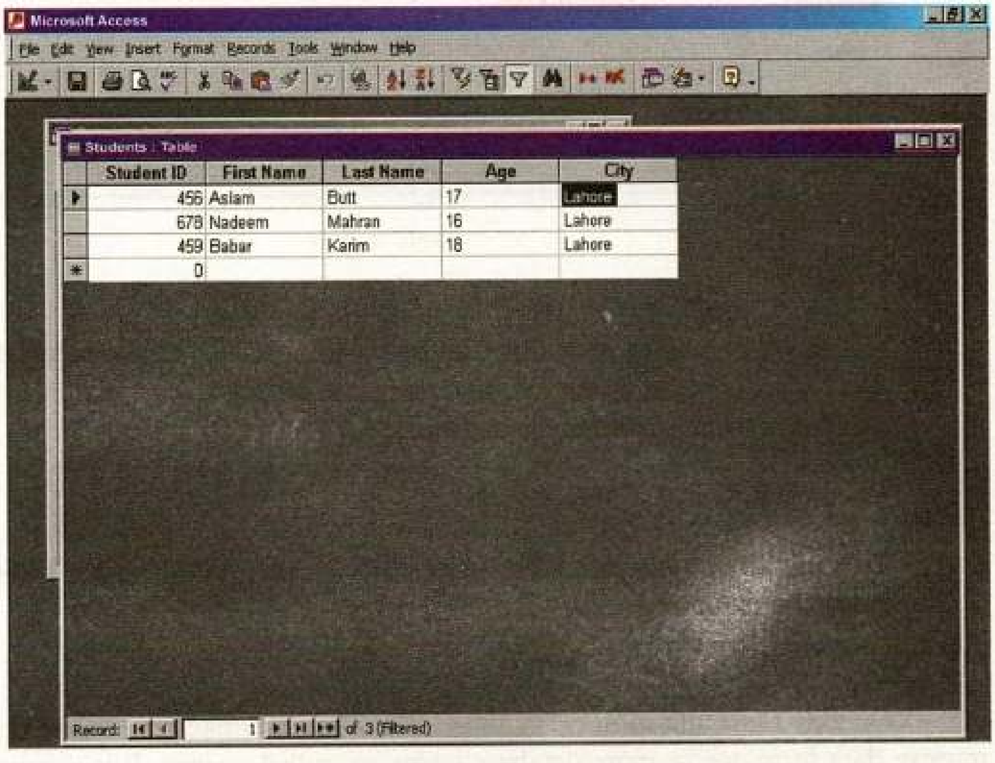
- In table view, place the cursor in the column that you want to sort by.

- Select *Records|Sort|Sort Ascending* or *Records|Sort|Sort Descending* from the menu bar or click the *Sort Ascending* or *Sort Descending* buttons on the toolbar.

To sort by more than one column (such as sorting by date and then sorting records with the same date alphabetically), highlight the columns and select one of the sort methods stated above.

Filter by Selection

This feature will filter records that contain identical data values in a given field such as filtering out all of the records that have the value "Smith" in a name field. To Filter by Selection, place the cursor in the field that you want to filter the other records by and click the *Filter by Selection* button on the toolbar or select *Records|Filter|Filter By Selection* from the menu bar. In the figure, the cursor is placed in the City field of the second record that displays the value "Lahore" so the filtered table will show only the records where the city is Lahore.


Microsoft Access
File Edit View Insert Format Records Tools Window Help
Students : Table
Record: 1 of 3 (Filtered)

	Student ID	First Name	Last Name	Age	City
▶	456	Aslam	Butt	17	Lahore
	679	Nadeem	Mahran	16	Lahore
	459	Babar	Karim	18	Lahore
*	0				

Fig.6.16 Filtering Data in table

Filter by Form

If the table is large, it may be difficult to find the record that contains the value you would like to filter by so using Filter by Form may be advantageous instead. This method creates a blank version of the table with drop-down menus for each field each menu contains the values found in the records of that field. Under the default *Look for* tab of the Filter by Form window, click in the field to enter the filter criteria. To specify an alternate criteria if records contain one of two specified values, click the *Or* tab at the bottom of the window and select another criteria from the drop-down menu. More *Or* tabs will appear after one criteria is set to allow you to add more alternate criteria for the filter. After you have selected all of the criteria you want to filter, click the *Apply Filter* button on the toolbar.

Following methods can be used to select records based on the record selected by that do not have exactly the same value. Type these formats into the field where the drop-down menu appears instead of selecting an absolute value.

Filter by Form	
Format	**Explanation**
Like "*Street"	Selects all records that end with "Street"
<="G"	Selects all records that begin with the letters A through G
>1/1/00	Selects all dates since 1/1/00
<> 0	Selects all records not equal to zero

Table 6.7

Saving A Filter

The filtered contents of a table can be saved as a query by selecting *File|Save As Query* from the menu bar. Enter a name for the query and click *OK*. The query is now saved within the database.

Remove a Filter

To view all records in a table again, click the depressed **Apply Filter** toggle button on the toolbar.

6.8 INTRODUCTION TO QUERIES

Queries mean question or inquires. The questions like statements that are to retrieve data form one or more database tables are called queries. It is a powerful and flexible way of selecting, filtering and sorting records.

Queries select records from one or more tables in a database; these selected records can be viewed, analyzed, and sorted on a common datasheet. The resulting collection of records, called a *dynaset* (short for dynamic subset), is saved as a database object and can therefore be easily used in future.

The query will be updated whenever the original tables are updated. Types of queries are *select queries* that extract data from tables based on specified values, *find duplicate* queries that display records with duplicate values for one or more of the specified fields, and *find unmatched* queries display records from one table that do not have corresponding values in a second table.

Types of Queries

In general, there are five types of query: Select queries, Action queries, Crosstab queries, Parameter queries and SQL queries.

Select Queries

A select query gathers, collates and presents information in usable forms. It retrieves data from one or more tables and displays the results in a datasheet where you can update the records. You can also use a select query to group records and calculate sums, counts, averages, and other types of totals.

Action Queries

An action query makes changes in specified records of an existing table, or creates a new table. There are four types of action queries:

- **Delete Queries:**

  A delete query deletes a group of records from one or more tables.

- **Update Queries:**

  An update query makes changes to a group of records in one or more tables.

- **Append Queries:**

  An append query adds a group of records from one or more tables to the end of one or more tables.

Crosstabe Queries

There are crosstab queries to calculate and restructure data for easier analysis of your data. Crosstab queries calculate a sum, average, count, or other type of computation for data. These queries are grouped by two types of information one down the left side of the datasheet and another across the top.

Parametic Queries

A parameter query is a query that when run displays its own dialog box prompting you for information. Parameter queries are also used as the basis for forms and reports.

Create a Query in Design View

Follow these steps to create a query in Design View:

- From the Queries page on the Database Window, click the **New** button.

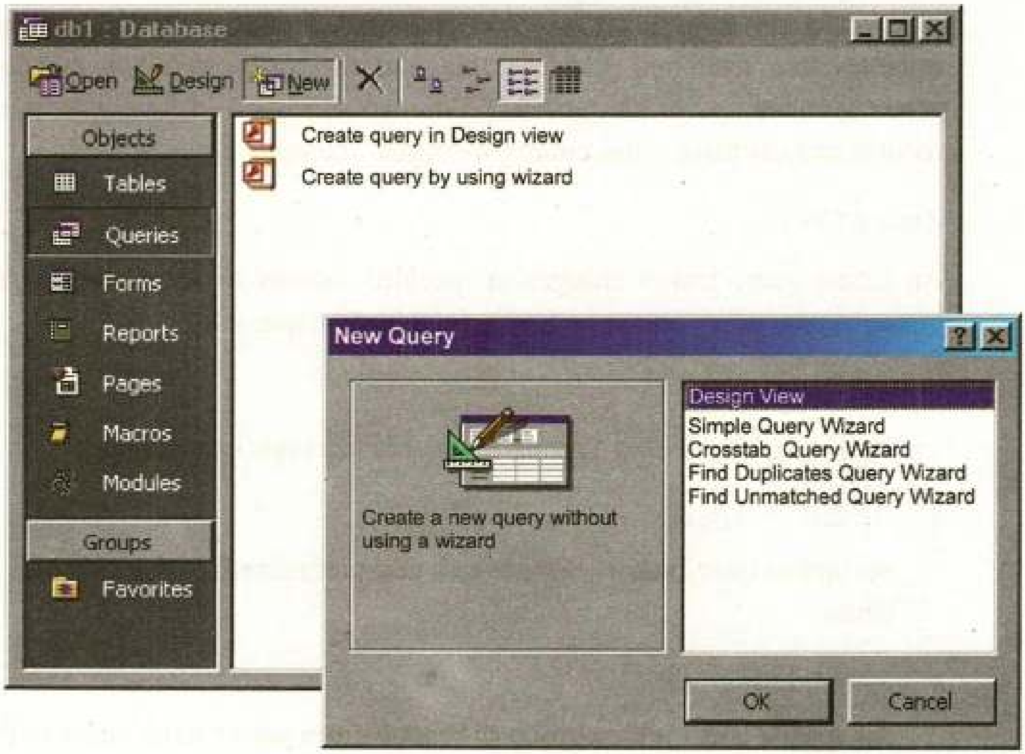


Fig.6.17 Creating a Query in Design View

- Select Design View and click **OK**.
- Select tables and existing queries from the **Tables** and **Queries** tabs and click the **Add** button to add each one to the new query.
- Click **Close** when all of the tables and queries have been selected.

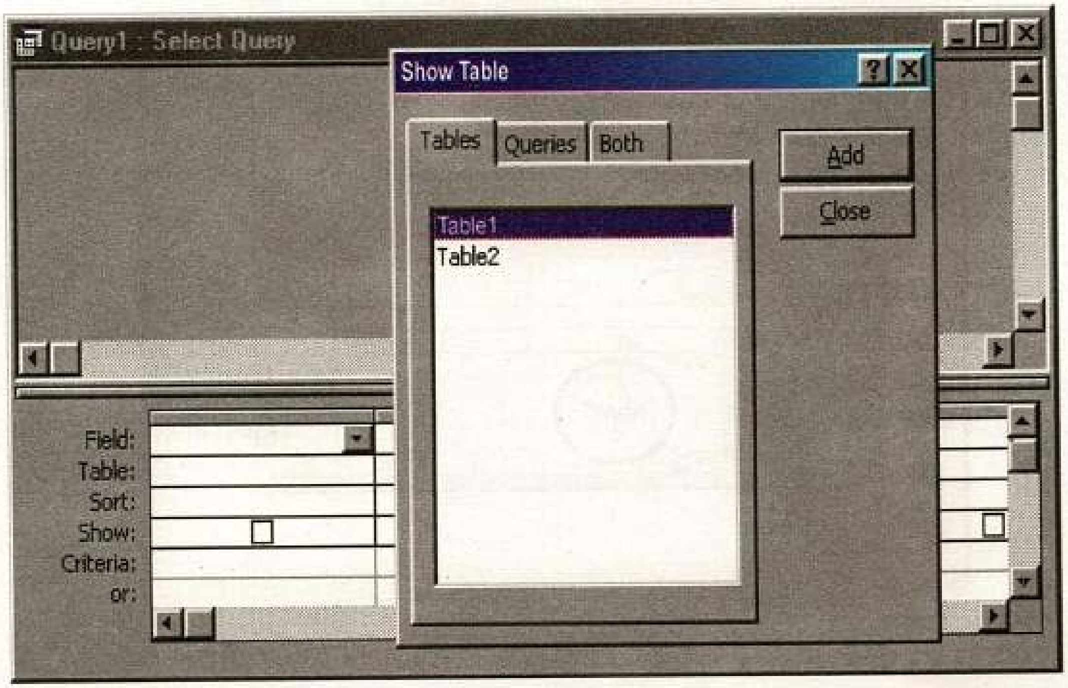
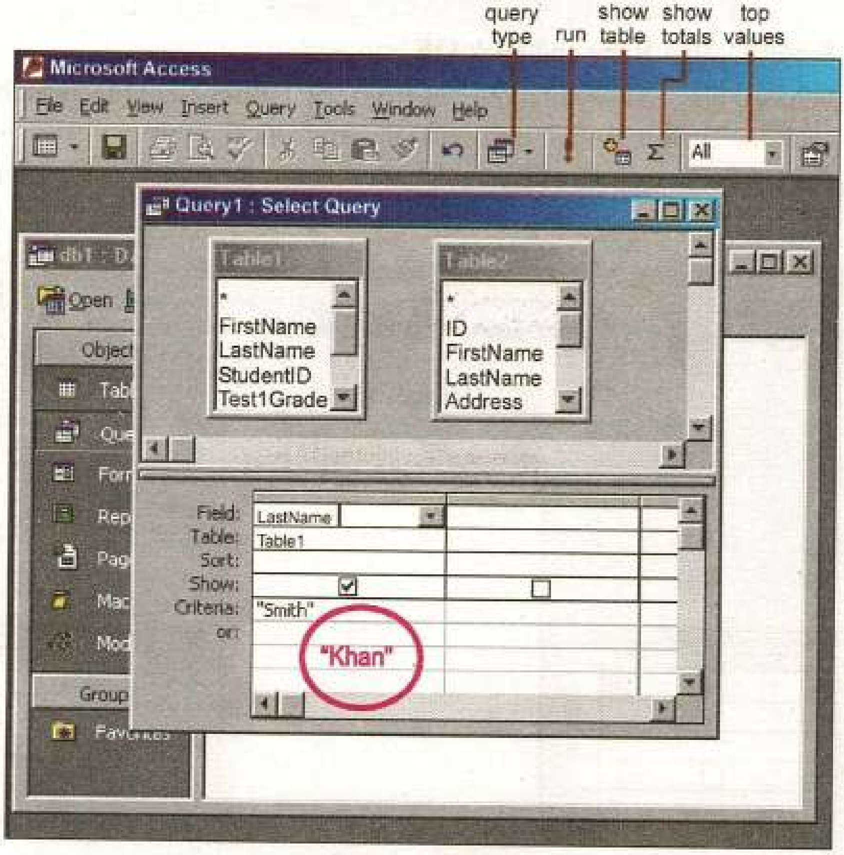


Fig.6.18. Add a Table to the Query in Design View

- Add fields from the tables to the new query by double-clicking the field name in the table boxes or selecting the field from the *Field*: and *Table*: drop-down menus on the query form. Specify sort orders if necessary.
- Enter the criteria for the query in the *Criteria*: field. The following table provides examples for some of the wildcard symbols and arithmetic operators that may be used. The *Expression Builder* can also be used to assist in writing the expressions.
- After you have selected all of the fields and tables, click the *Run* button on the toolbar.
- Save the query by clicking the *Save* button.



Field:	LastName	
Table:	Table1	
Sort:		
Show:	✓	☐
Criteria:	"Smith"	
or:	"Khan"	

Fig.6.19 Specifying Criteria

Specifying Criteria

Once you've selected all your query fields, you can narrow your query to include only data that matches specific criteria. You may want to display only records with certain field values, for example. A query that display only employees in a certain state is an example of the use of criteria or indicate what values not to include.

Wildcards

Wildcards offer a way of setting criteria based on patterns or partial words rather than exact matches. For example, the criterion A* in a *First Name* field specify name beginning with the letter A, such as Ahmed Ali. The following are the most common wildcard operators:

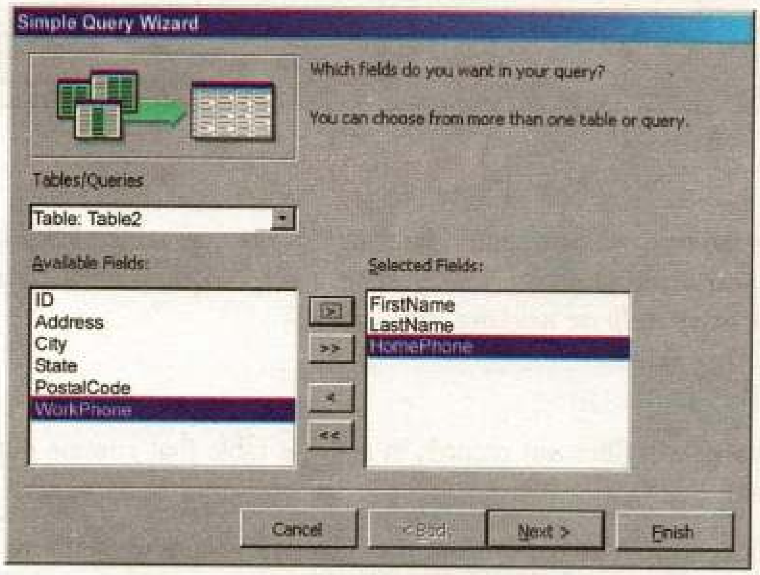
Query Wildcards and Expression Operators	
Wildcard / Operator	**Explanation**
? Street	The question mark is a wildcard that takes the place of a single letter.
43th *	The asterisk is the wildcard that represents a number of characters.
<100	Value less than 100
>=1	Value greater than or equal to 1
<>"XI"	Not equal to (all classes besides XI)
Between 1 and 10	Numbers between 1 and 10
Is Null **Is Not Null**	Finds records with no value. or all records that have a value
Like "a*"	All words beginning with "a"
>0 And <=10	All numbers greater than 0 and less than 10
"Khan" Or "Ahmed"	Values are Khan or Ahmed

Table 6.8

Query Wizard

Ms-Access' Query Wizard will easily assist you to begin creating a select query.

- Click the *Create query by using wizard* icon in the database window to have Access step you through the process of creating a query.


Simple Query Wizard

Which fields do you want in your query?

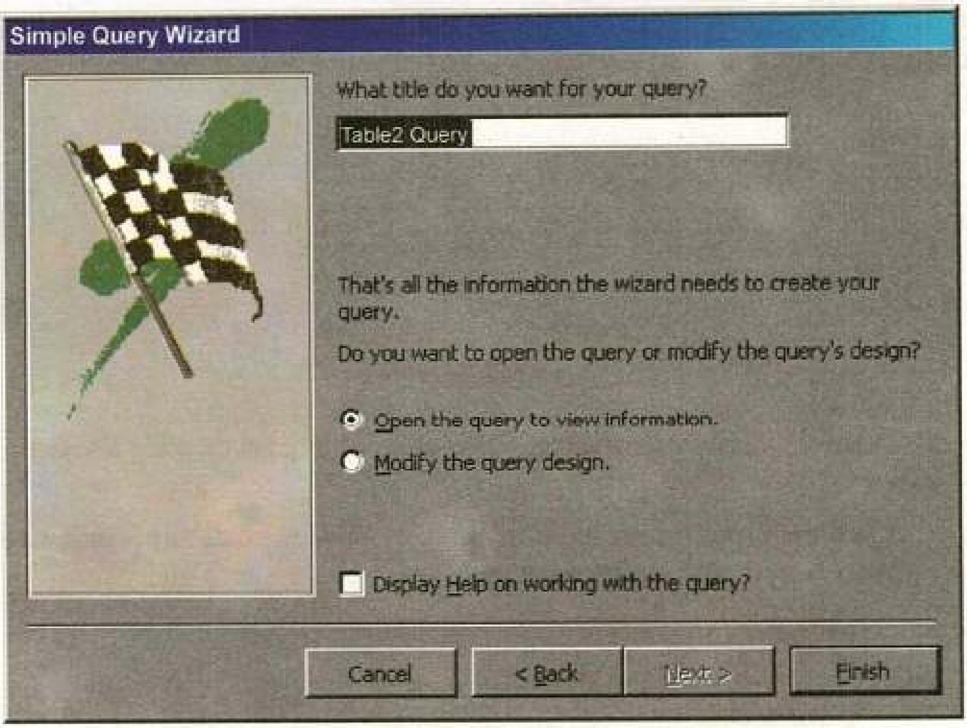
You can choose from more than one table or query.

Tables/Queries: Table: Table2

Available Fields:	Selected Fields:
ID	FirstName
Address	LastName
City	HomePhone
State	
PostalCode	
WorkPhone	

Fig.6.20 Creating query by using wizard

- From the first window, select fields that will be included in the query by first selecting the table from the drop-down *Tables/Queries* menu. Select the fields by clicking the > button to move the field from the Available Fields list to Selected Fields or Click the double arrow button >> to move all of the fields to Selected Fields. Select another table or query to choose from more fields and repeat the process of moving them to the Selected Fields box. Click *Next* > when all required fields have been selected.



What title do you want for your query?

Table2 Query

That's all the information the wizard needs to create your query.

Do you want to open the query or modify the query's design?

[✓ Open the query to view information.]

Modify the query design.

☐ Display Help on working with the query?

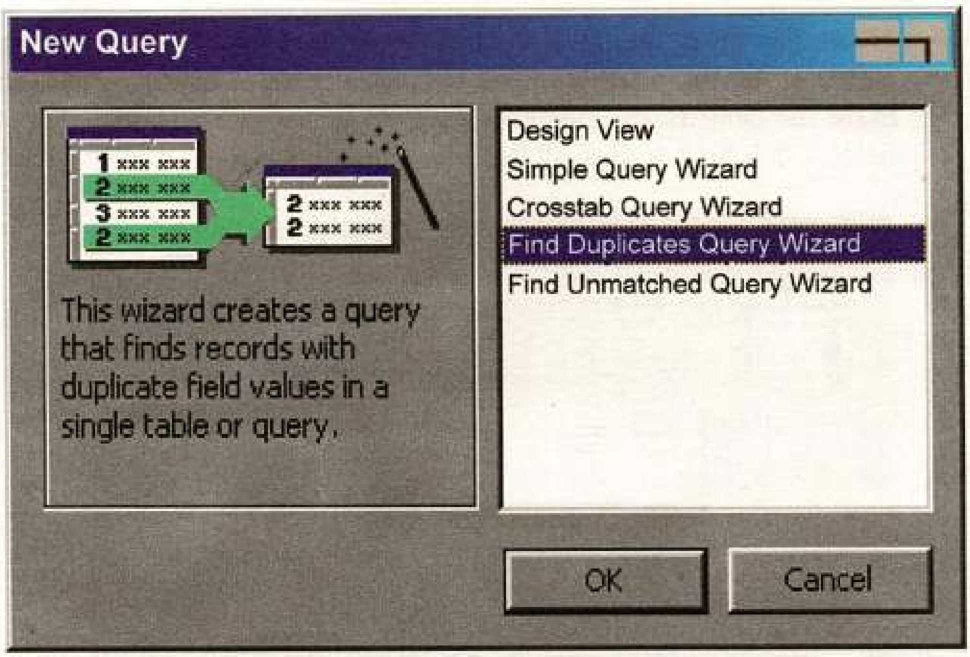
Fig.6.21 Finishing query wizard

- On the next window, enter the name for the query and click *Finish.*
- If you want to automatically open a help window click on check box of "*Display Help on working with query*?".

Find Duplicates Query

This query will filter out records in a single table that contain duplicate values in a field.

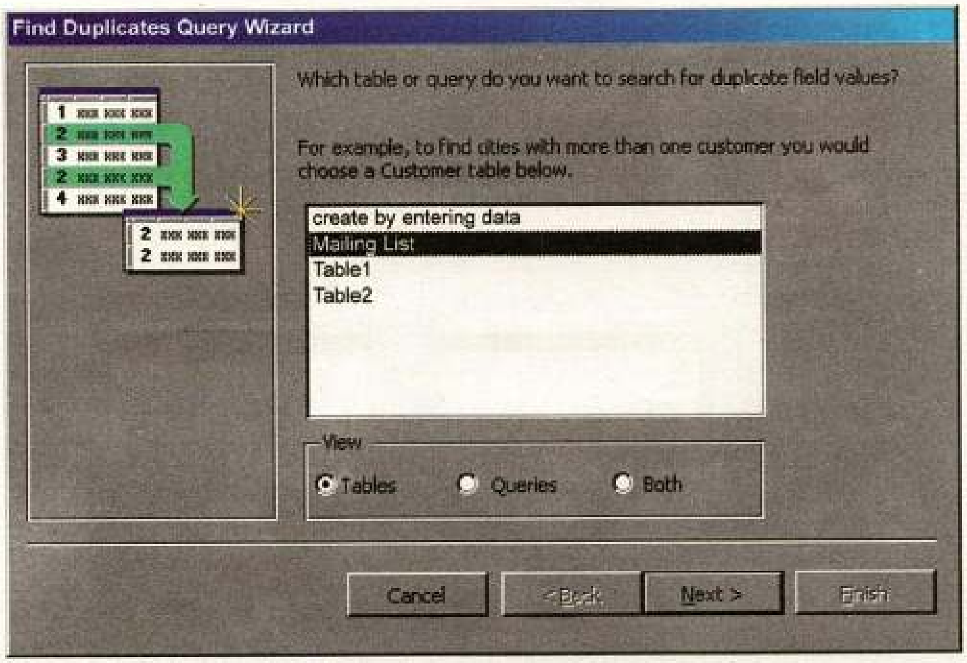
- Click the *New* button on the Queries database window, select *Find Duplicates Query Wizard* from the *New Query* window and click *OK*.



This wizard creates a query that finds records with duplicate field values in a single table or query.

Fig. 6.22 Creating a Query that finds records with Duplicate Field

- Select the table or query that the find duplicates query will be applied to from the list provided and click *Next* >.

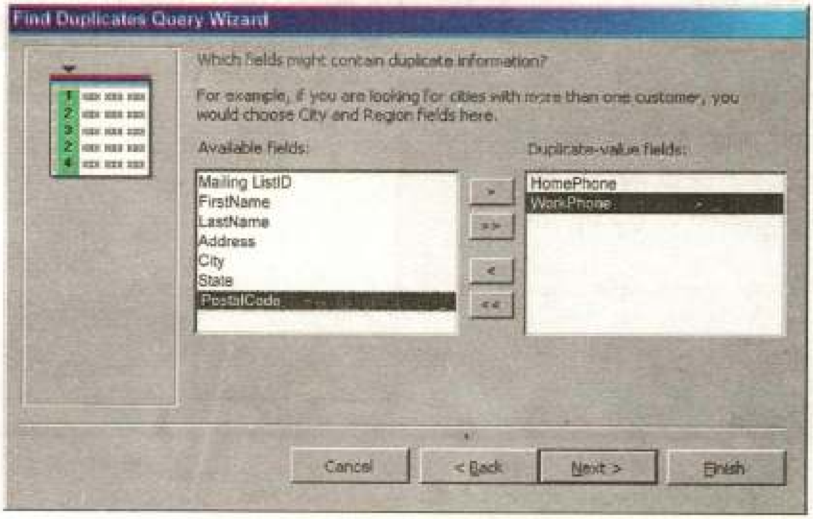


Which table or query do you want to search for duplicate field values?

For example, to find cities with more than one customer you would choose a Customer table below.

Fig.6.23 Defining Table or Query to find Duplicate Field Value.

- Select the fields that may contain duplicate values by highlighting the names in the Available fields list and clicking the > button to individually move the fields to the Duplicate-value fields list or >> to move all of the fields. Click **Next >** when all fields have been selected.



Which fields might contain duplicate information?

For example, if you are looking for cities with more than one customer, you would choose City and Region fields here.

Available fields:

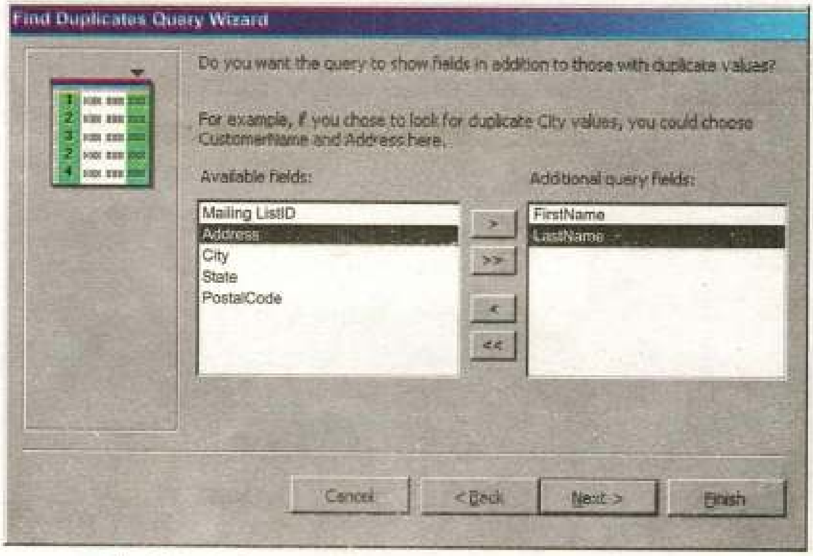
Mailing ListID
FirstName
LastName
Address
City
State
PostalCode

Duplicate-value fields:

HomePhone
WorkPhone

Fig.6.24 Definging Fields that have more than one Duplicate Information

- Select the fields that should appear in the new query along with the fields selected on the previous screen and click *Next* >.



Do you want the query to show fields in addition to those with duplicate values?

For example, if you chose to look for duplicate City values, you could choose CustomerName and Address here.

Available fields:

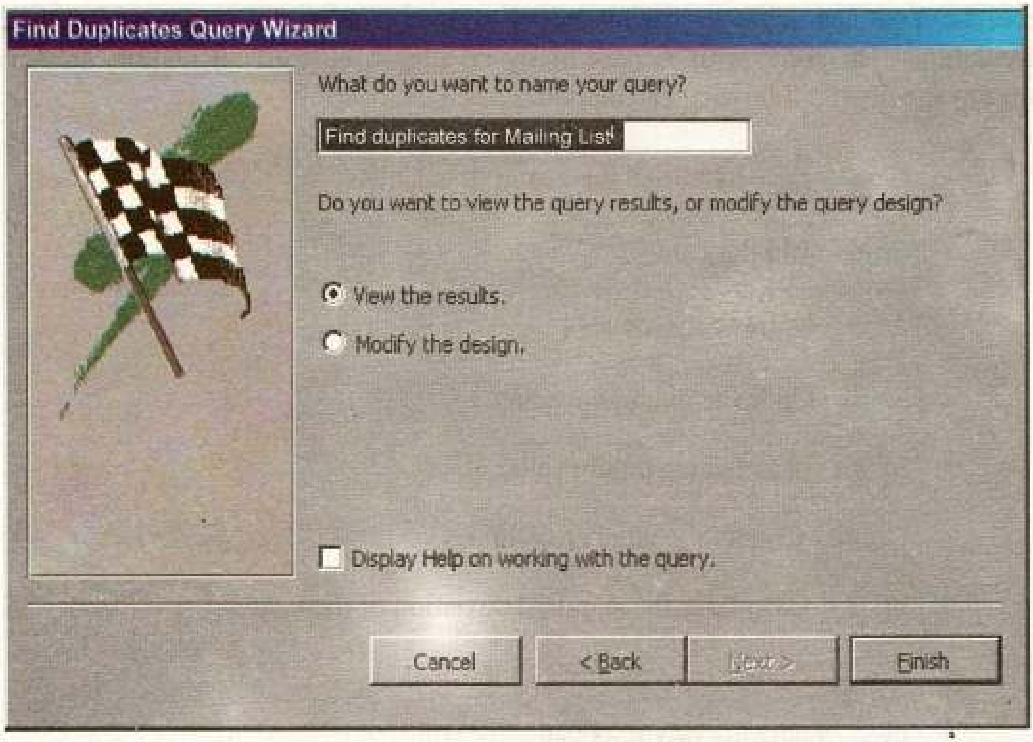
Mailing ListID
Address
City
State
PostalCode

Additional query fields:

FirstName
LastName

Fig.6.25 A Query to Show Fields for Duplicate Information.

- Name the new query and click *Finish*.


Find Duplicates Query Wizard

What do you want to name your query?

Find duplicates for Mailing List

Do you want to view the query results, or modify the query design?

(●) View the results.

( ) Modify the design.

[ ] Display Help on working with the query.

Fig.6.25 Finishing the Find Duplicate Query

Delete a table from the Query

- To delete a table from the query, click the table's title bar and press the **Delete** key on the keyboard.

Sorting Query Field

By default, query fields are not sorted. You can, however, sort any field in either ascending or descending order.

- Click on the *Sort row* in the field column you want to sort. A down arrow will appear to the right of the field.
- Click on the *down arrow.* A menu will appear.
- Click on *Ascending / Descending or not sorted* to sort in ascending/descending order or not sort the field.

6.9 PERFORMING CALCULATION IN A QUERY

Using a query is a convenient way to perform a calculation on a group of records. You can perform calculation in a query either by using the predefined calculations that come with Access or by creating a custom calculation. In Access, you can specify the following calculation types:

- **Group By:** Identified the group to calculate.
- **Sum:** Add the values.
- **Avg:** Average the values.
- **Min:** Finds the minimum value.
- **Max:** Finds the maximum value.
- **Count:** Counts the number of values.
- **StDev:** Calculates the standard deviation of the value.
- **Var:** Calculate the variance of the value.
- **First:** Finds the first field value.
- **Last:** Finds the last field value.
- **Expression:** Creates a calculate field through an expression.
- **Where:** Indicates criteria for a field not included in the query.

Exercise 6c

1. Fill in the blank

(i) The __________ is specified in table to avoid duplicate entries of records.

(ii) In Microsoft Access, the data of the table is displayed in __________ view.

(iii) In the relational model, ________ is the basic structure in which data is stored.

(iv) The __________ is a graphical representation of the structure of a database.

(v) The _______ button shows the current record in the table.

(vi) If the primary key is made up of a group of two or more fields, it is called ______.

(vii) _________ are special characters that are used in queries to specify the criteria.

(viii) In Microsoft Access, the output of a query is in the form of a ___________.

(ix) A query that involves two tables is called ___________.

(x) The wildcard character _________ is used to specify any number of characters.

(xi) The ___________ data type is used when the field is to contain text consisting of about 300 characters.

(xii) The number of rows in a table of a relational database is called the ___________ of the table.

(xiii) The number of columns in a table is called the _________ of the table.

(xiv) A query that only retrieve and displays data is called _______.

(xv) The wildcard character _______ is used to specify a single digit.

2. Select the correct option.

(i) The data in table is entered in:

a) Design View
b) Normal View
c) Datasheet View
d) Layout View

(ii) How many types of relationship?.

a) 2 b) 3 c) 4 d) 5

(iii) In a relational database, a single piece of information is called:

a) Field b) Record
c) Entity d) Attribute

(iv) The rule that a record from a table cannot be deleted if its associated record exits in a related table is called ________ rule.

a) Referential integrity
b) Entity – relationship
c) Normalization
d) All of them

(v) How many table views are available in Microsoft Access.

a) 4 b) 3 c) 2 d) 1

(vi) To find a four-character name that starts with H, the criteria is specified as.

a) H*4 b) H?4 c) H???? d) H####

(vii) Which of the following buttons of Find and Replace dialog box is clicked to start the search process?

a) Find b) Find Next c) Search d) Next

(viii) As in Design view, you can move from field to field in the Table window in Datasheet view using _______ button:

a) Tab b) Esc c) Enter d) Spacebar

(ix) The relationship between countries and their capitals is an example of _______ relationships.

a) one-to-one b) one-to-many

c) many-to-many d) None of them

(x) The wildcard _____ Sal[ei]ma.

a) Saleemà b) Salima

c) both a and b d) None of them

3. Write T for true and F for false statement.

(i) If for each entity in B, and for each entity in B, there is only one related entity in A, then the relationship between the entities is one-to-many.

(ii) Date/Time data type is used when data such as data of birth or time of day is to be stored.

(iii) The primary key is assigned in a table to avoid duplicate entries of records.

(iv) An append query adds a group of records from one or more tables to the end of one or more tables.

(v) The currency data type cannot be used in calculations.

(vi) The most commonly used type of query is Scratch Query.

(vii) A query is used to extract specific information from a database.

(viii) The wildcard character "?" is used to specify any number of characters.

(ix) Fields are displayed as check boxes by default on the datasheet.

(x) An action query makes changes in specified records of an existing table, or creates a new table.

4. Define the different data types available in Microsoft Access.

6. Define the primary key.

7. What is Referential Integrity?

8. What are relationships?
9. How are relationship defined in Microsoft Access?
10. Explain the options in Find and Replace dialog box.
11. Differentiate between Relationship and Join.
12. Define different types calculation in a query and also specifies the some Functions.
13. What is query? Discuss its uses and advantages.
14. Explain the criteria in a query. How is it specified?
15. What is a join? Explain its purpose.
16. Differentiate between Sorting and Filtering.
17. What are wildcards?
18. Define the various types of queries.
19. How can you create a query in Design View?
20. Create a Query in Design View Create a Query in Design View.

MICROSOFT ACCESS-FORMS AND REPORTS

Chapter **7**

7.1 OVERVIEW

Access Form creates the user interface to your table. Although you can use Datasheet to perform many of the same functions as forms, Forms are used as an alternative way to enter data into a database table. It provides a different way of viewing table data. Access enables us to create forms that can be used to enter, maintain, view, and print data.

The Form is constructed from a collection of individual design elements called controls or control objects. Controls are the components we see in the windows dialog boxes of the access and other windows applications like buttons, check boxes etc. We use text boxes to enter and edit data, labels to hold field names and object frame to display graphics.

A Form Wizard is provided to assist you in the construction of forms. Four types of forms can be created. These include single-column (displaying one record at a time in a vertical format), tabular (displaying multiple records in a row-and-column format), main/subform (combining the single-form and tabular formats into one form), and graph.

Create Form by Using Wizard

To create a form using the assistance of the wizard and follow these steps:

- Click the *Create form by using wizard* option on the database window.
- From the *Tables/Queries* drop-down menu, select the table or query whose datasheet the form will modify. Then, select the fields that will be included on the form by highlighting each one the *Available Fields* window and clicking the single right greater symbol button > to move one field at a time to the *Selected Fields* window. To move all of the fields to Select Fields, click the double greater symbol >>. If you make a mistake and would like to remove a field or all of the fields from the Selected Fields window, click the left arrow < or left double arrow << buttons. After the proper fields have been selected, click the *Next* > button to move on to the next screen.

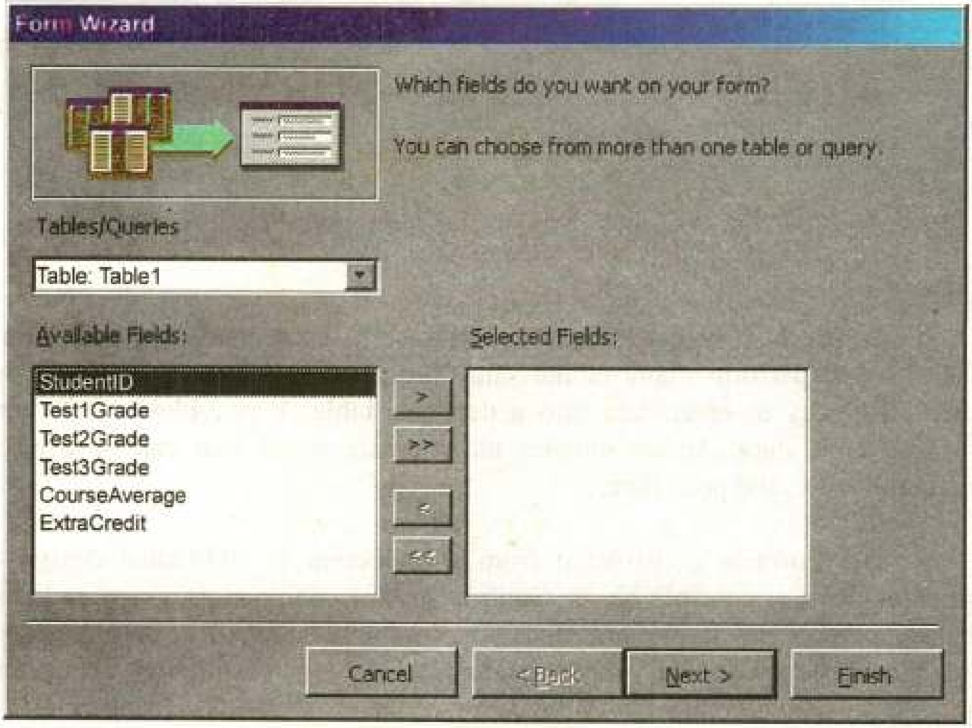
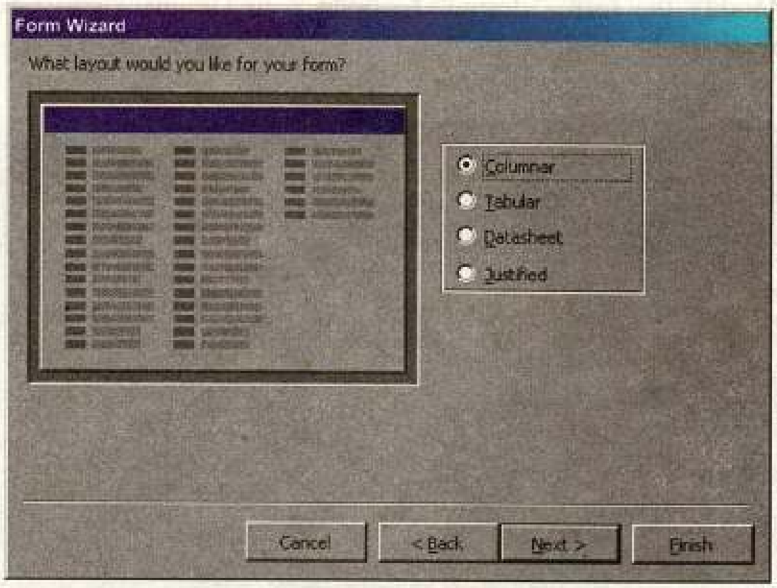


Fig.7.1 Ist Step of Form Wizard

- In the second step we have the screen to select the layout of the form.
  - **Columnar Form:** A single record is displayed at one time with labels and form fields listed side-by-side in columns. In a columnar form, each field appears in a separate line with a label to its left; only one record is shown on each screen. The wizard fills the first column with as many fields as will fit and so forth.
  - **Justified Form:** A single record is displayed with labels and form fields are listed across the screen
  - **Tabular Form:** Multiple records are listed on the page at a time with fields in columns and records in rows. Tabular forms display fields in horizontal row, with field labels at the top of the form. Each new row represents a new record. Tabular forms are best when you want to display just a few relatively narrow fields and you want to see several records at once. To avoid spending most of your time scrolling back and forth in a tabular form, add just a few fields to the form.
  - **Datasheet Form:** A datasheet form initially displays data in datasheet view, much as it appears when you open a table, or run a query, or when you use the *Form view* toolbar button to switch to datasheet view

in any form. This type of form is often used as the basis for sub forms.

- Click the *Next* > button to move on to the next step.

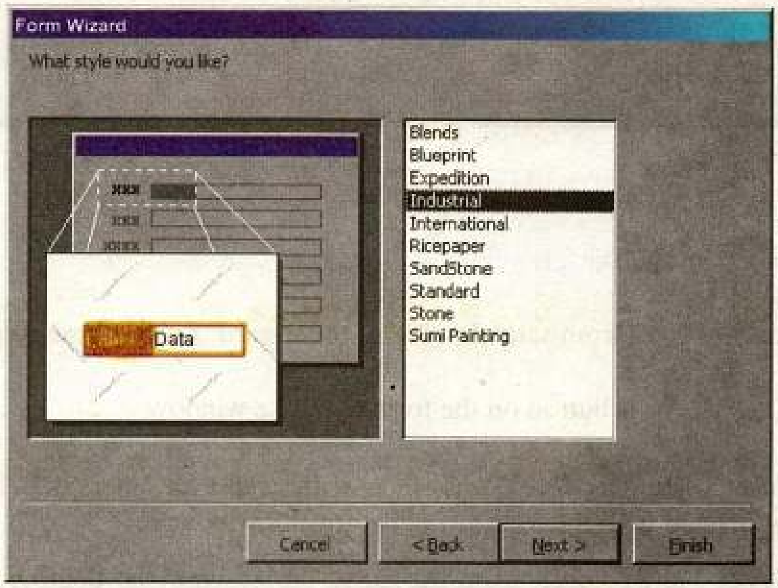

Form Wizard

What layout would you like for your form?

- [✓ Columnar]
- Tabular
- Datasheet
- Justified

Fig.7.2 Defining Different types of Layout

- Select a visual style for the form from the next set of options and click *Next* >.

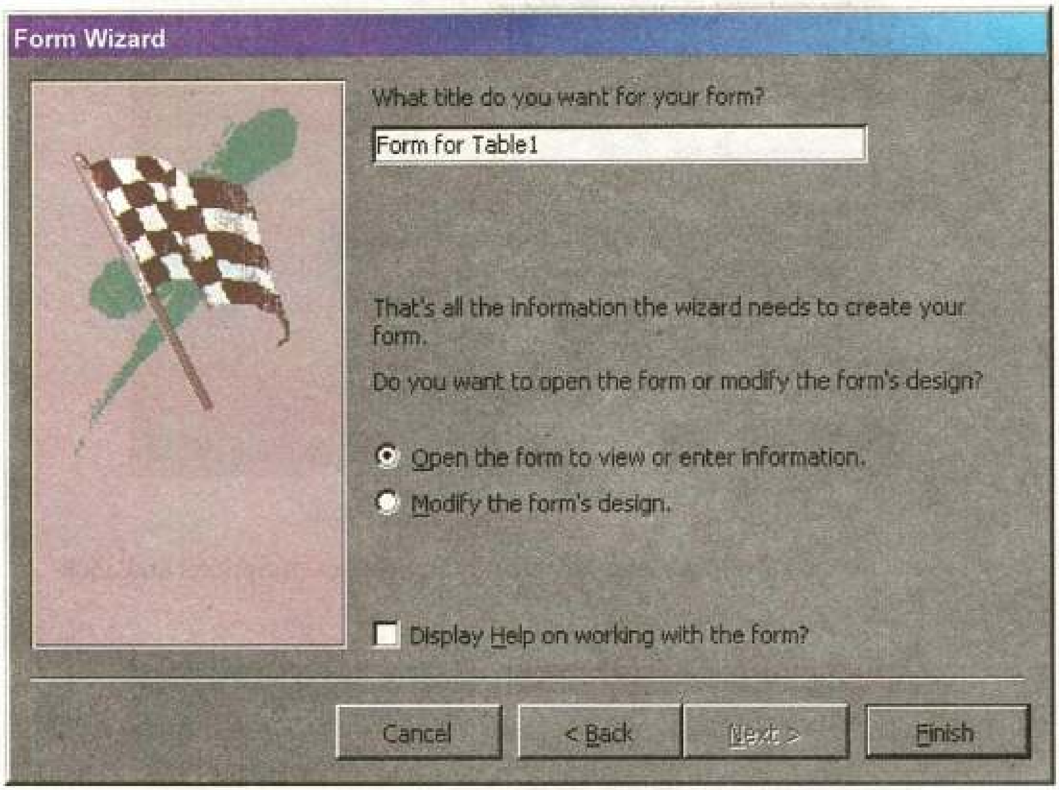

Form Wizard

What style would you like?

- Blends
- Blueprint
- Expedition
- [✓ Industrial]
- International
- Ricepaper
- SandStone
- Standard
- Stone
- Sumi Painting

Fig.7.3 Choosing a Form Styles in Form Wizard

- On the final screen, enter the title you want to display on your form in the text box and than name the form in the space provided. Select "Open the form to view or enter information" to open the form in Form View or "Modify the form's design" to open it in Design View. Click *Finish* to create the form.


Form Wizard

What title do you want for your form?

Form for Table1

That's all the information the wizard needs to create your form.

Do you want to open the form or modify the form's design?

[✓ Open the form to view or enter information.]

Modify the form's design.

Display Help on working with the form?

Fig.7.4 Finishing the Form in Form Wizard

Create Form in Design View

To create a form from scratch without the wizard, follow these steps:

- Click the *New* button on the form database window.

- Select "Design View" and choose the table or query the form will be associated with the form from the drop-down menu.

- Select *View\Toolbox* from the menu bar to view the floating toolbar with additional options.

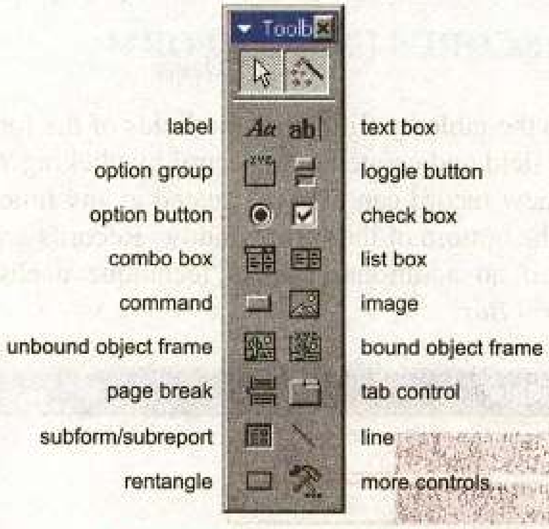


Fig.7.5 Defining tools of Toolbox

- Add controls to the form by clicking and dragging the field names from the Field List floating window. Access creates a text box for the value and label for the field name when this action is accomplished. To add controls for all of the fields in the Field List, double-click the Field List window's title bar and drag all of the highlighted fields to the form.

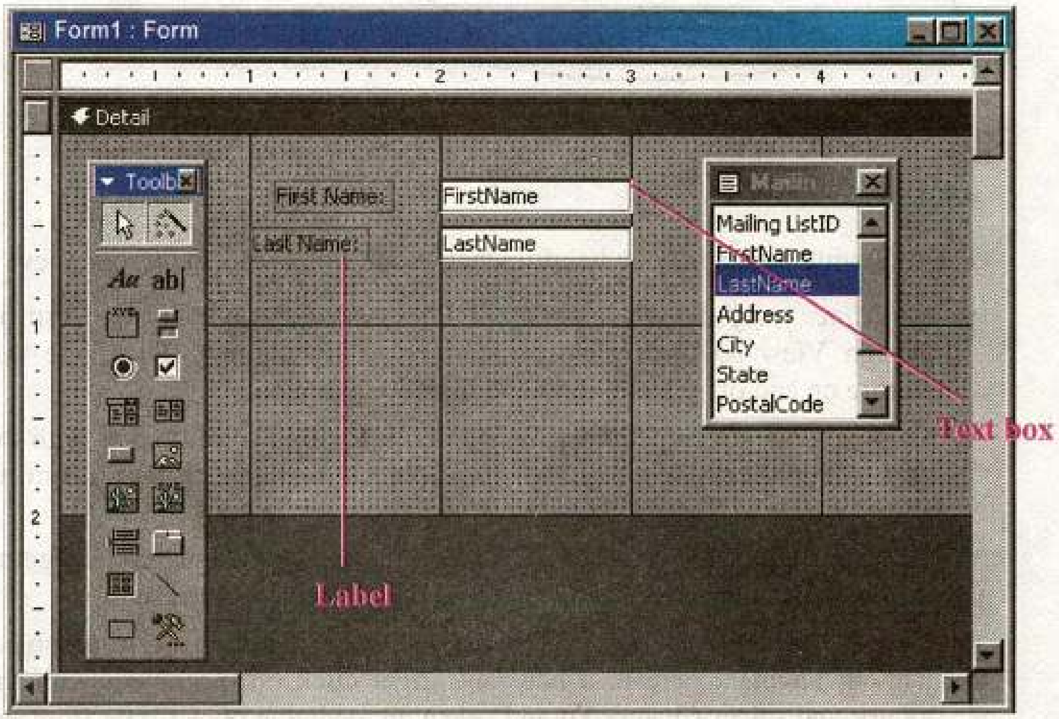
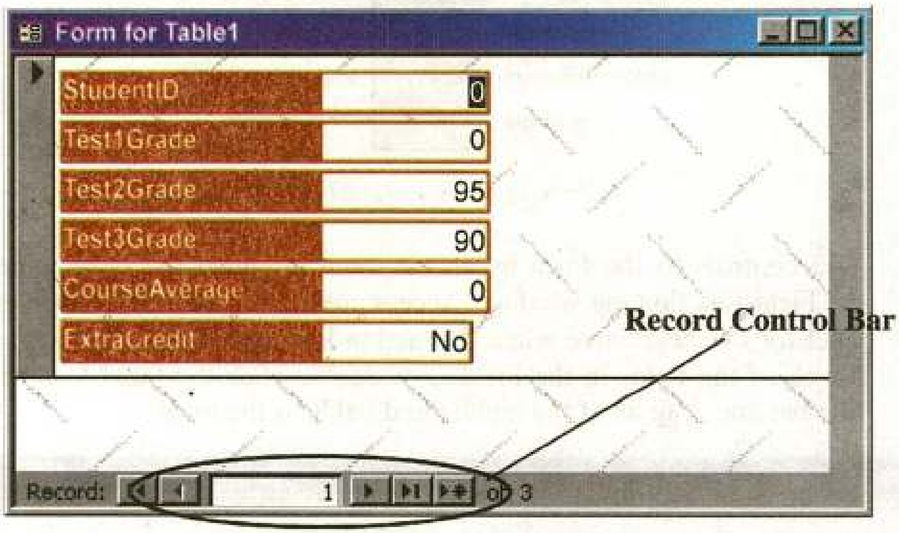


Fig.7.6 Creating a Form by Design View

7.2 ADDING RECORDS USING A FORM

Input data into the table by filling out the fields of the form. Press the *Tab* key to move from field to field and create a new record by clicking *Tab* after the last field of the last record. A new record can also be created at any time by clicking the *New* **Record** button ▶* at the bottom of the form window. Records are automatically saved as they are entered so no additional manual technique needs to be executed, by clicking *Record Control Bar*.



StudentID	0
Test1Grade	0
Test2Grade	95
Test3Grade	90
CourseAverage	0
ExtraCredit	No

Record: 1 of 3

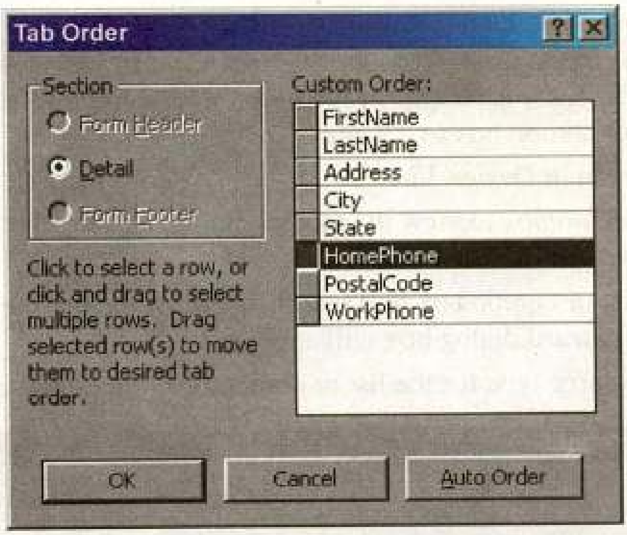
Fig.7.7 Adding the record in a Form Design View

Editing Forms

The following points may be helpful when modifying forms in Design View.

- *Grid lines* - By default, a series of lines and dots underlay the form in Design View so form elements can be easily aligned. To toggle this feature on and off select View|Grid from the menu bar.
- *Snap to Grid* - Select *Format|Snap* to *Grid* to align form objects with the grid to allow easy alignment of form objects or uncheck this feature to allow objects to float freely between the grid lines and dots.
- *Resizing Objects* - Form objects can be resized by clicking and dragging the handles on the edges and corners of the element with the mouse.
- *Change form object type* - To easily change the type of form object without having to create a new one, right click on the object with the mouse and select Change To and select an available object type from the list.

- *Label/object alignment* - Each form object and its corresponding label are bounded and will move together when either one is moved with the mouse. However, to change the position of the object and label in relation to each other (to move the label closer to a text box, for example), click and drag the large handle at the top, left corner of the object or label.
- *Tab order* - Alter the tab order of the objects on the form by selecting *View|Tab Order...* from the menu bar. Click the grey box before the row you would like to change in the tab order, drag it to a new location, and release the mouse button. This can also be done by using Tab Property in the properties.



Section: Form Header ( ) / [✓ Detail] / Form Footer ( )

Custom Order:

- FirstName
- LastName
- Address
- City
- State
- HomePhone
- PostalCode
- WorkPhone

Click to select a row, or click and drag to select multiple rows. Drag selected row(s) to move them to desired tab order.

Fig.7.8 Defining the Tab Order in a Form

- **Form Appearance:** Change the background color of the form by clicking the *Fill/Back Color* button on the formatting toolbar and click one of the color swatches on the palette. Change the color of individual form objects by highlighting one and selecting a color from the *Font/Fore Color* palette on the formatting toolbar. The font and size, font effect, font alignment, border around each object, the border width, and a special effect can also be modified using the formatting toolbar:

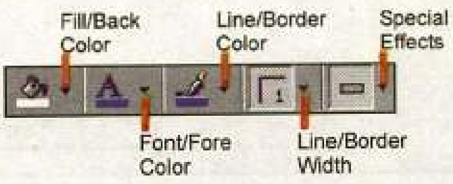


Fig.7.9 Defining the Formatting Toolbar

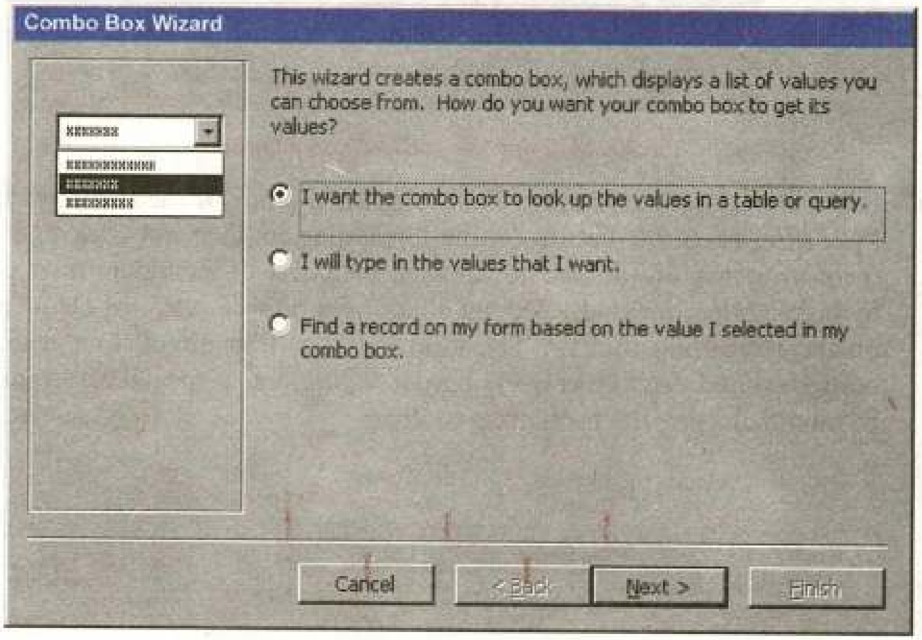
- **Page Header and Footer:** Headers and footers added to a form will only appear when it is printed. Select these sections by choosing *View|Page Header/Footer* on the menu bar. Page numbers can also be added to these sections by selecting *Insert|Page Numbers*. A date and time can be added from *Insert|Date and Time....* Select *View|Page Header/Footer* again to hide these sections from view in Design View.

7.3 LIST AND COMBO BOXES

If there are small, finite number of values for a certain field on a form, using combo or list boxes may be a quicker and easier way of entering data. These two control types differ in the number of values they display. List box values are all displayed during data entering while the combo box values are not displayed until the arrow button is clicked to open it as shown in these examples:

By using a combo or list box, the name of the cities does not need to be typed for every record. Instead, it simply needs to be selected from the list. Follow these steps to add a list or combo box to a form:

- Open the form in *Design View*.
- Select *View|Toolbox* to view the toolbox and make sure the "Control Wizards" button is pressed in.
- Click the list or combo box tool button and draw the outline on the form. The combo box wizard dialog box will appear.
- Select the source type for the list or combo box values and click *Next >*.


Combo Box Wizard

This wizard creates a combo box, which displays a list of values you can choose from. How do you want your combo box to get its values?

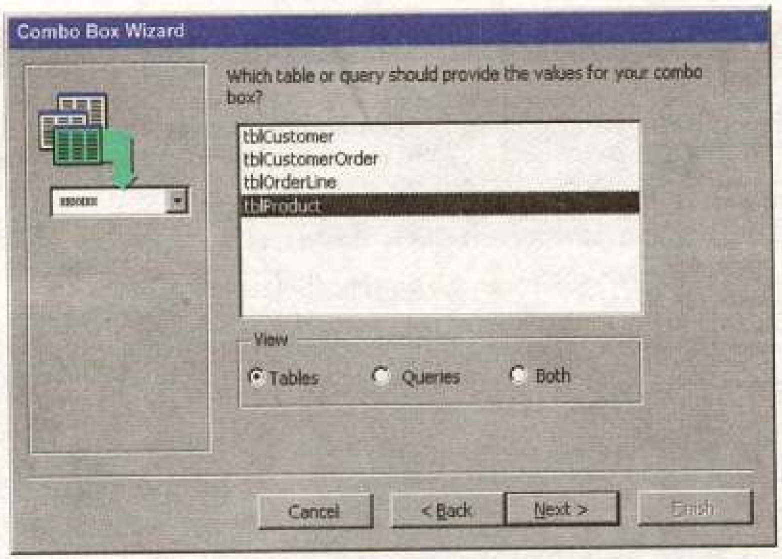
(●) I want the combo box to look up the values in a table or query.

( ) I will type in the values that I want.

( ) Find a record on my form based on the value I selected in my combo box.

Fig.7.10 Ist Step of Combo Box Wizard

- Depending on your choice in the first dialog box, the next options will vary. If you chose to look up values from a table or query, the following box will be displayed. Select the table or query from which the values of the combo box will come from. Click *Next* > and choose fields from the table or query that was selected. Click *Next* > to proceed.


Combo Box Wizard

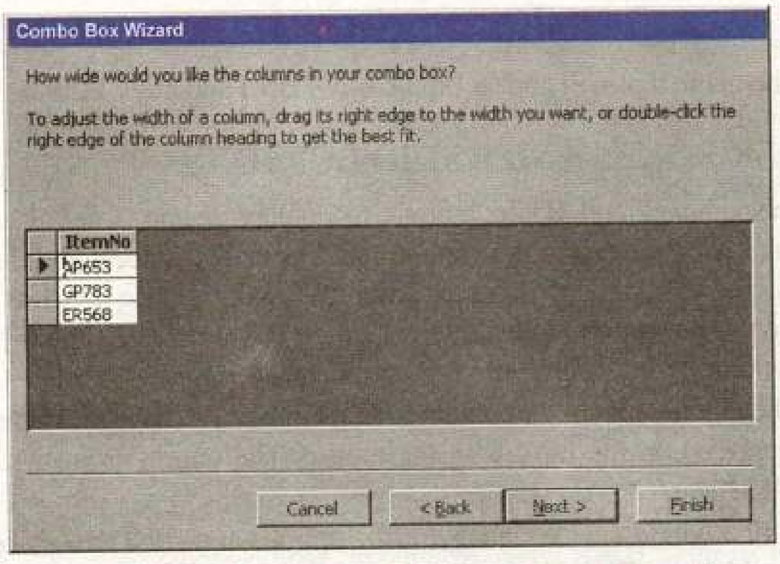
Which table or query should provide the values for your combo box?

tblCustomer
tblCustomerOrder
tblOrderLine
tblProduct

View: [✓ Tables] Queries Both

Fig.7.11 Choosing the Table in Combo Box Wizard

- On the next dialog box, set the width of the combo box by clicking and dragging the right edge of the column. Click *Next* >.


Combo Box Wizard

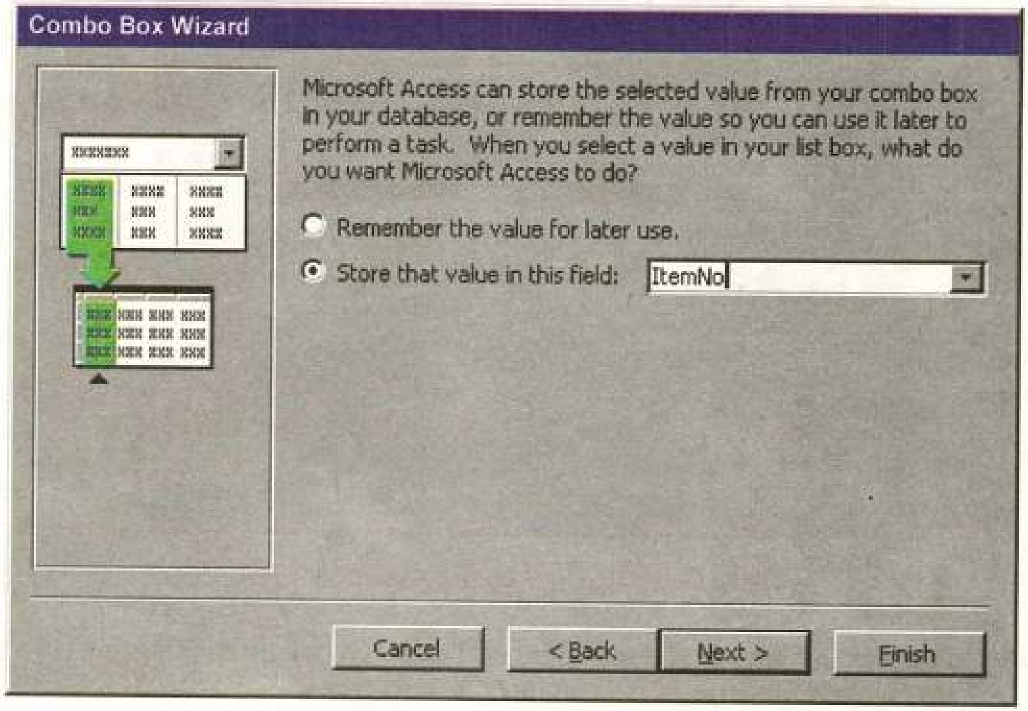
How wide would you like the columns in your combo box?

To adjust the width of a column, drag its right edge to the width you want, or double-click the right edge of the column heading to get the best fit.

ItemNo
AP653
GP783
ER568

Fig.7.12 Viewing the Column in Combo Box Wizard

- The next dialog box tells Access what to do with the value that is selected. Choose "Remember the value for later use" to use the value in a macro or procedure (the value is discarded when the form is closed), or select the field that the value should be stored in. Click *Next* > to proceed to the final screen.



Microsoft Access can store the selected value from your combo box in your database, or remember the value so you can use it later to perform a task. When you select a value in your list box, what do you want Microsoft Access to do?

Remember the value for later use. [✓ Store that value in this field: ItemNo]

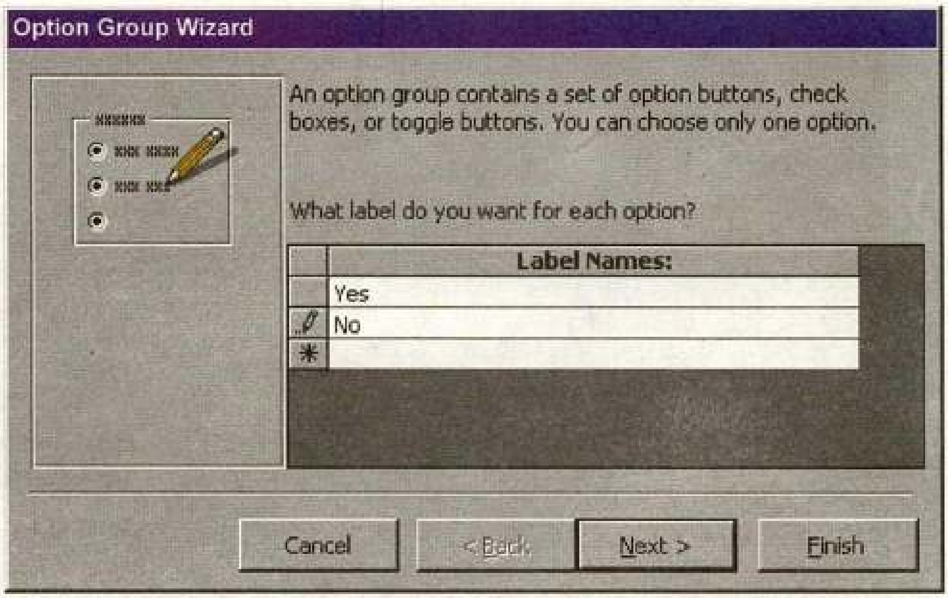
Fig.7.13 Entering data in Combo Box

- Type the name that will appear on the box's label and click **Finish**.

7.4 CHECK BOXES AND RADIO BUTTONS

Use check boxes and Radio buttons to display yes/no, true/false, or on/off values. Only one value from a group of radio buttons can be selected while any or all values from a check box group can be chosen. Typically, these controls should be used when five or less option are available. Combo boxes or lists should be used for long lists of options. To add a checkbox or option group:

- Click the *Option Group* tool on the toolbox and draw the area where the group will be placed on the form with the mouse. The option group wizard dialog box will appear.
- On the first window, enter labels for the options and click the tab key to enter additional labels. Click *Next* > when finished typing labels.



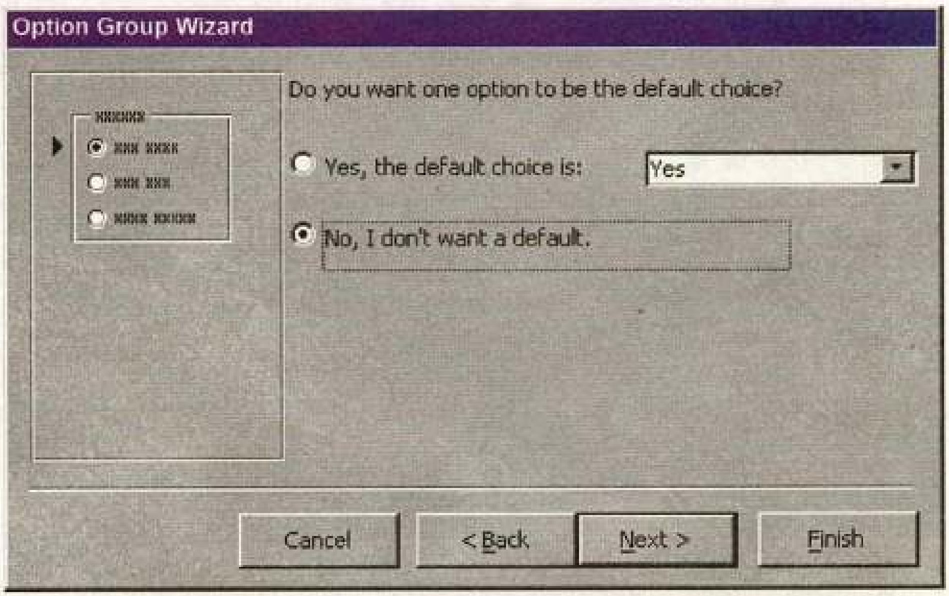
An option group contains a set of option buttons, check boxes, or toggle buttons. You can choose only one option.

What label do you want for each option?

Label Names:
Yes
No

Fig. 7.14 Choosing the option in Option Group Wizard.

- On the next window, select a default value if there is any and click *Next* >.



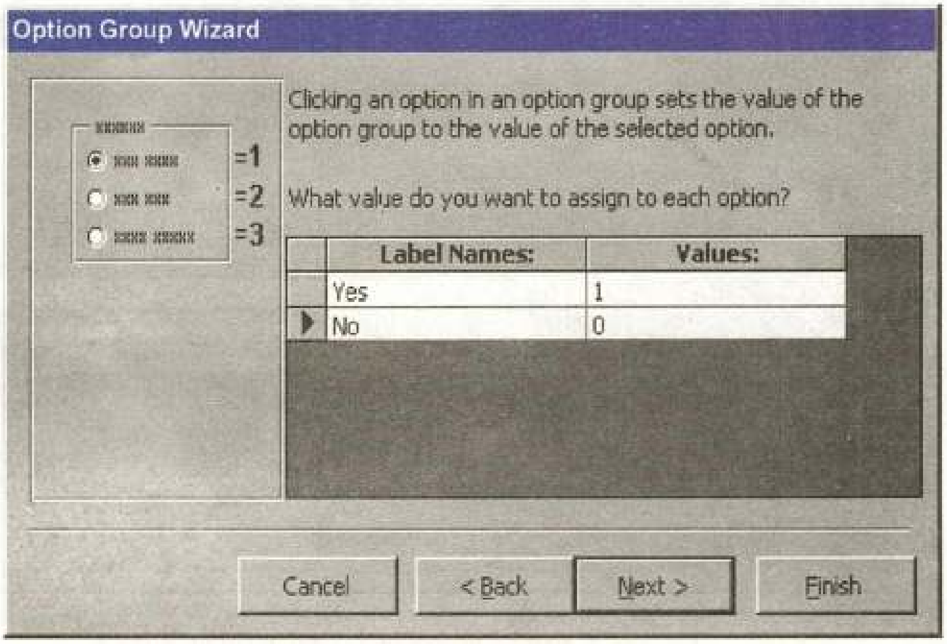
Do you want one option to be the default choice?

Yes, the default choice is: Yes

[✓ No, I don't want a default.]

Fig.7.15 Defining the default option in Option Group Wizard

- Select values for the options and click *Next* >.


Option Group Wizard

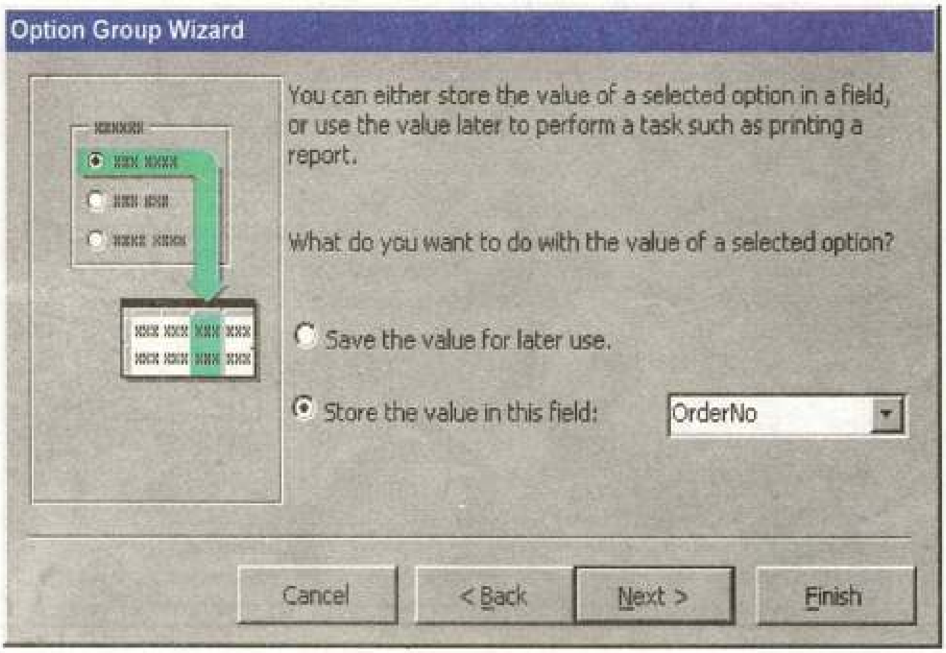
Clicking an option in an option group sets the value of the option group to the value of the selected option.

What value do you want to assign to each option?

Label Names:	Values:
Yes	1
No	0

Fig.7.16 Selecting the option Values in Option Group Wizard

- Choose *what* should be done with the value and click *Next* >.


Option Group Wizard

You can either store the value of a selected option in a field, or use the value later to perform a task such as printing a report.

What do you want to do with the value of a selected option?

Save the value for later use.

[✓ Store the value in this field:] OrderNo

Fig.7.17 Storing the value of a selected option

- Choose the type and style of the option group and click Next >.

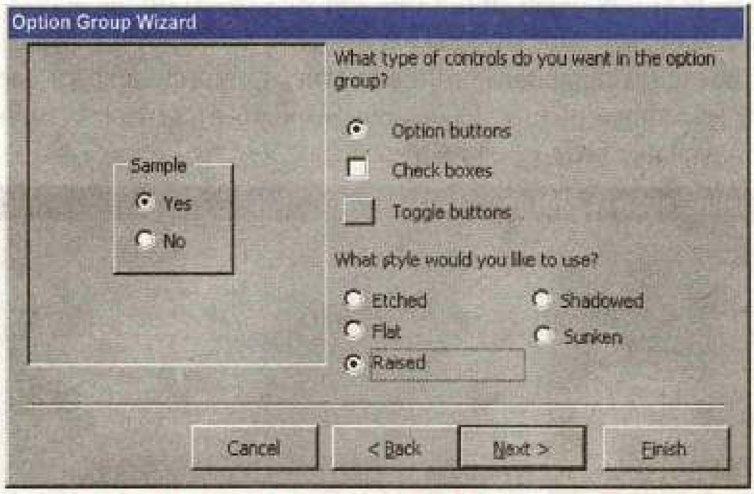


Fig.7.18 Specifying the controls and style of Option Button

- Type the caption for the option group and click *Finish*.

Command Buttons

In this example, a command button beside each record is used to open another form.

- Open the form in Design View and ensure that the Control Wizard button on the toolbox is pressed in.
- Draw the button on the form. The Command Button Wizard will then appear.
- On the first dialog window, action categories are displayed in the left list while the right list displays the actions in each category. Select an action for the command button and click *Next* >.

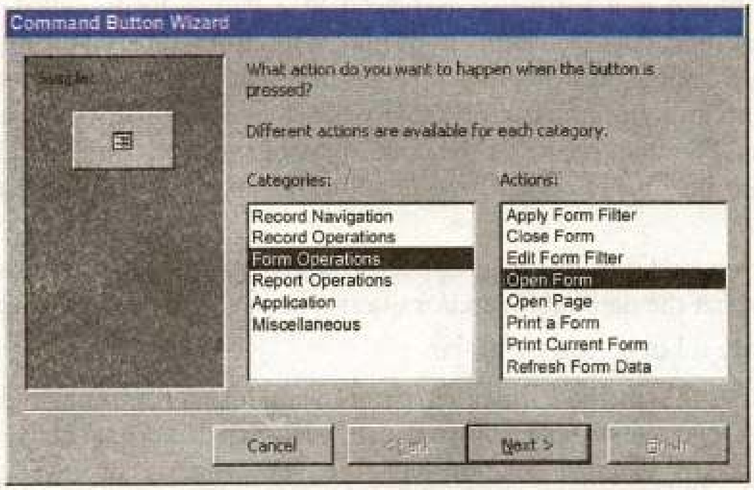
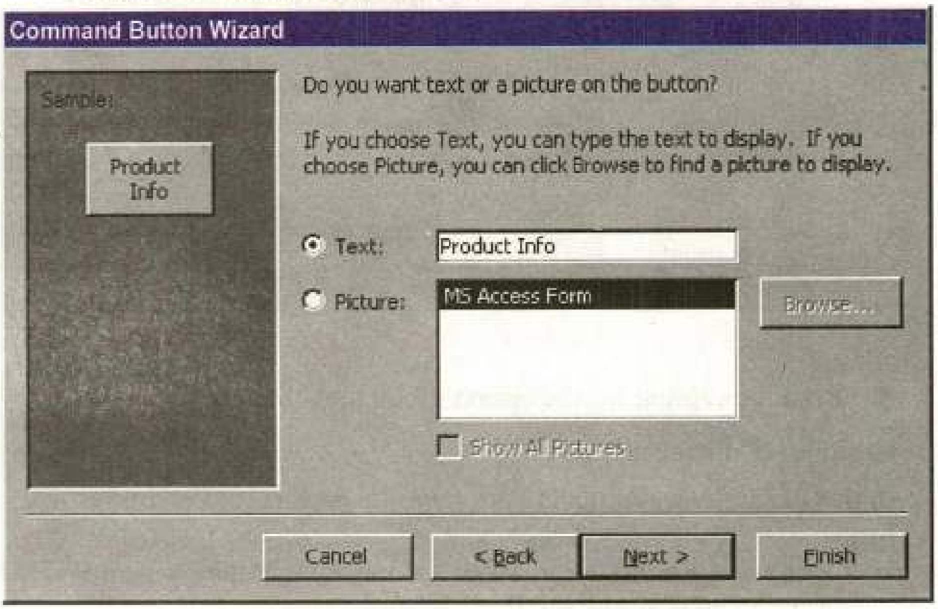


Fig.7.19 Defining the Action to command button

- The next few pages of options will vary based on the action you selected. Continue selecting options for the command button.
- Choose the appearance of the button by entering text for caption of button or selecting a picture. Check the *Show All Pictures* box to view the full list of available images. Click *Next >*.


Command Button Wizard

Sample:

Product Info

Do you want text or a picture on the button?

If you choose Text, you can type the text to display. If you choose Picture, you can click Browse to find a picture to display.

Text: Product Info

Picture: MS Access Form

Fig.7.20 Finishing Command Button Wizard

- Enter a name for the command button and click *Finish* to create the button.

7.5 SUBFORM

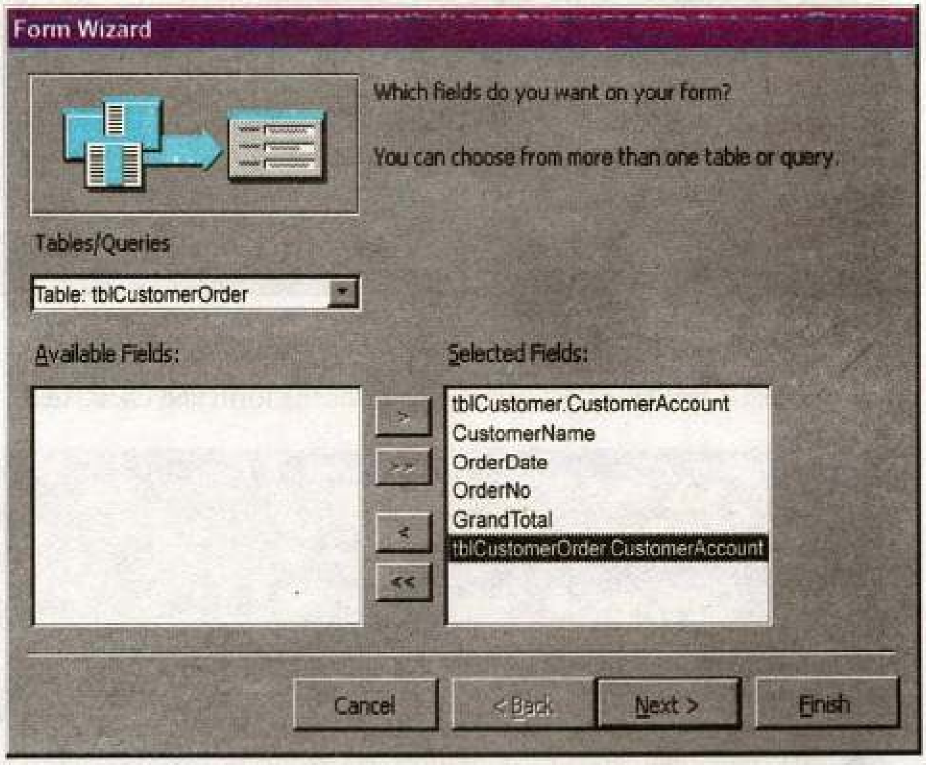
A subform is a form that is placed in a parent form, called the main form. Subforms are particularly useful to display data from tables and queries that have one-to-many relationships. For example, in the sample below, data on the main form is drawn from an item information table while the subform contains all of the orders for that item. The item record is the "one" part of this one-to-many relationship while the orders are the "many" side of the relationship since many orders can be placed for the one item.

The remainder of this page explains three methods for creating subforms and they assume that the data tables and/or queries have already been created.

Create a Form and Subform

- Use this method if neither parent and child form has already been created. A main form and subform can be created automatically using the form wizard if table relationships are set properly or if a query involving multiple tables is selected. For example, a relationship can be set between a table containing customer information and one listing customer orders so the orders for each customer are displayed together using a main form and subform. Follow these steps to create a subform within a form:

- Double-click *Create form by using wizard* on the database window.
- From the *Tables/Queries* drop-down menu, select the first table or query from which the main form will display its data. Select the fields that should appear on the form by highlighting the field names in the *Available Fields* list on the left and clicking the single greater symbol **>** button or click the double greater



Which fields do you want on your form?

You can choose from more than one table or query.

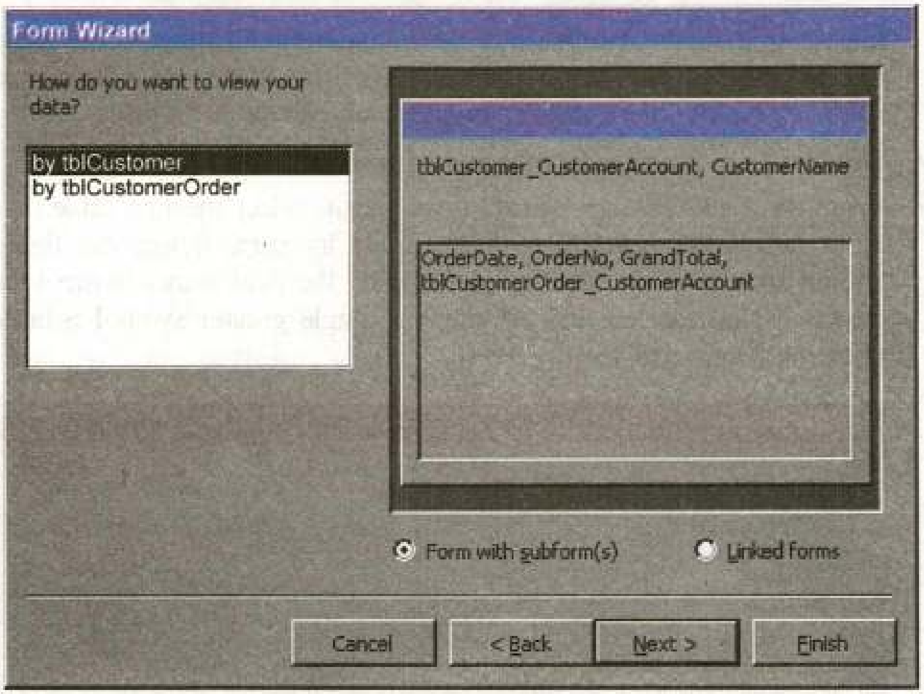
Tables/Queries: Table: tblCustomerOrder

Selected Fields:
tblCustomer.CustomerAccount
CustomerName
OrderDate
OrderNo
GrandTotal
tblCustomerOrder.CustomerAccount

Fig.7.21 Creating Subform from Form Wizard

- From the same window, select another table or query from the *Tables/Queries* drop-down menu and choose the fields that should appear on the form. Click *Next* to continue after all fields have been selected.

Choose an arrangement for the forms by selecting *form with subform(s)* if the forms should appear on the same page or *Linked forms* if there are many controls on the main form and a subform will not fit. Click *Next* to proceed to the next page of options.


Form Wizard

How do you want to view your data?

by tblCustomer
by tblCustomerOrder

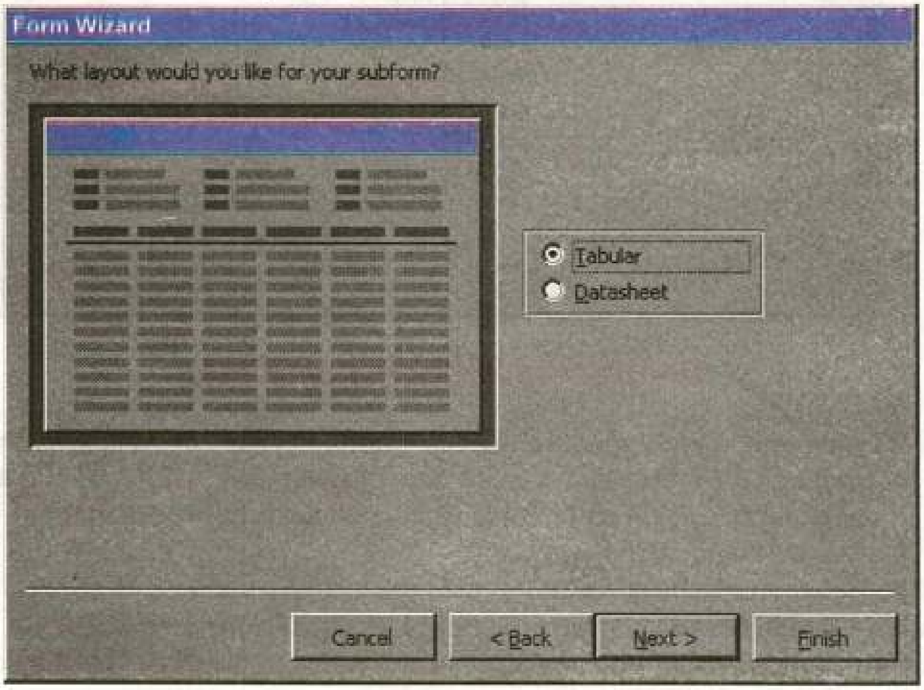
tblCustomer_CustomerAccount, CustomerName

OrderDate, OrderNo, GrandTotal, tblCustomerOrder_CustomerAccount

[✓ Form with subform(s)] Linked forms

Fig.7.22 Choosing the Form Views

- Select a tabular or datasheet layout for the form and click **Next**.

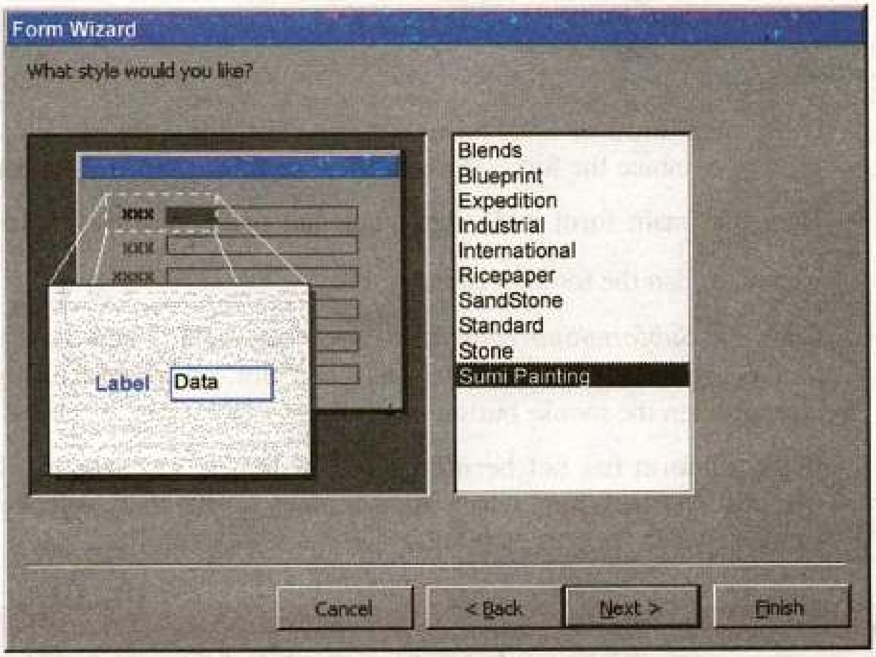

Form Wizard

What layout would you like for your subform?

[✓ Tabular] Datasheet

Fig.7.23 Selecting the layout of subform

- Select a style for the form and click *Next*.

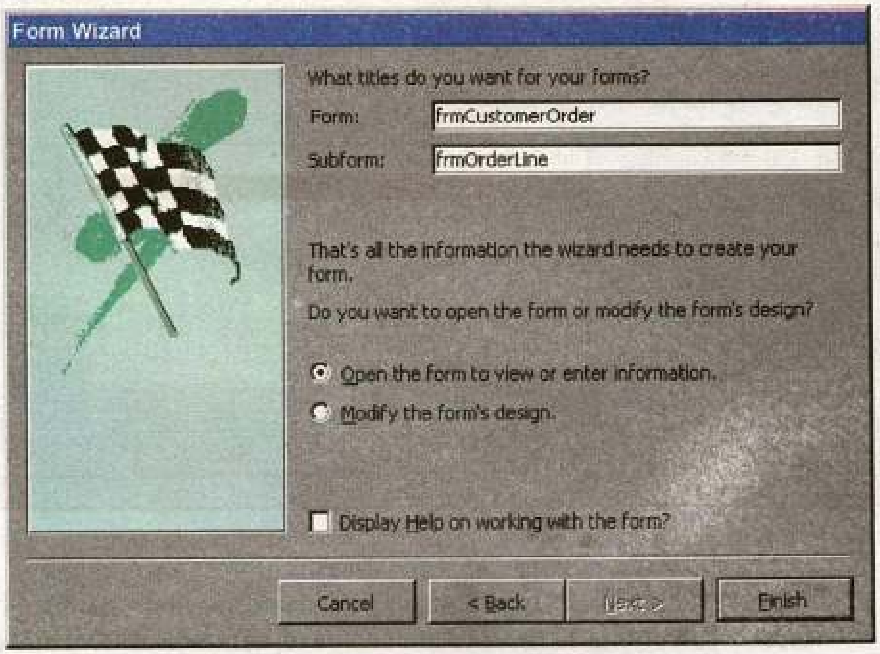

Form Wizard

What style would you like?

Blends
Blueprint
Expedition
Industrial
International
Ricepaper
SandStone
Standard
Stone
Sumi Painting

Fig.7.24 Specifing the style of Subform

- Enter the names for the main form and subform. Click *Finish* to create the forms.


Form Wizard

What titles do you want for your forms?

Form: frmCustomerOrder

Subform: frmOrderLine

That's all the information the wizard needs to create your form.

Do you want to open the form or modify the form's design?

[✓ Open the form to view or enter information.]

Modify the form's design.

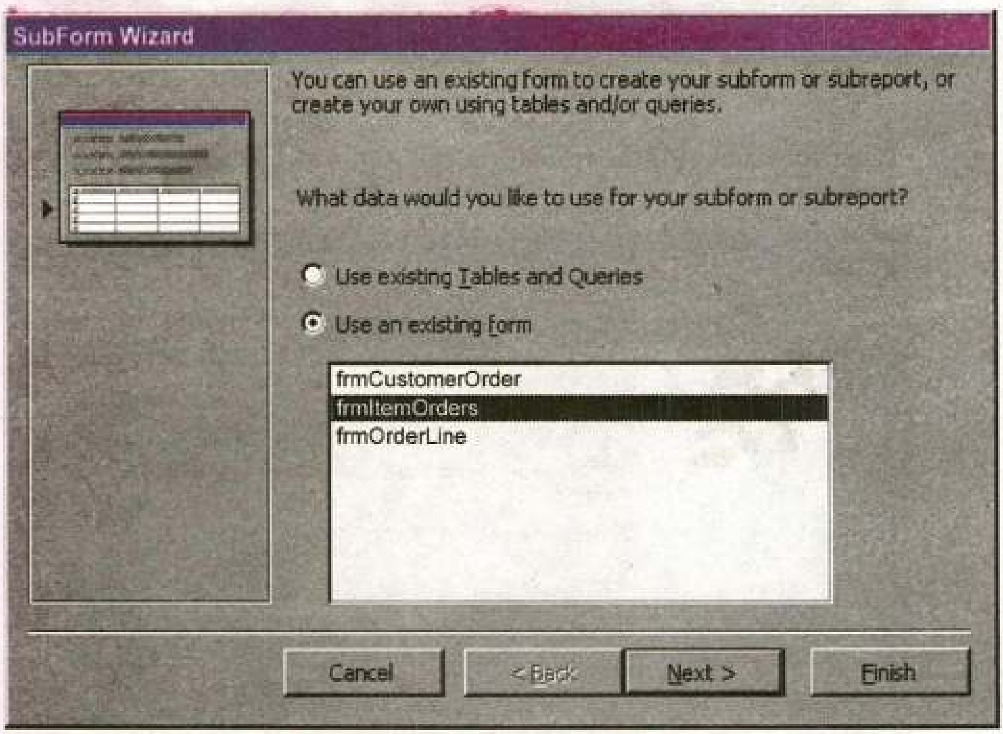
Display Help on working with the form?

Fig.7.25 Finishing the Subform

- New records can be added to both tables and queries at once by using the new combined form.

Subform Wizard

- If the main form or both forms already exist, the Subform Wizard can be used to combine the forms. Follow these steps to use the Subform Wizard:
- Open the main form in *Design View* and make sure the *Control Wizard* button on the toolbox is pressed in.
- Click the *Subform/Subreport* icon on the toolbox and draw the outline of the subform on the main form. The Subform Wizard dialog box will appear when the mouse button is released.
- If the subform has not been created yet, select "Use existing Tables and Queries". Otherwise, select the existing form that will become the subform. Click *Next* to continue.


SubForm Wizard

You can use an existing form to create your subform or subreport, or create your own using tables and/or queries.

What data would you like to use for your subform or subreport?

( ) Use existing Tables and Queries
(•) Use an existing form

frmCustomerOrder
frmItemOrders
frmOrderLine

Fig.7.26 Creating Subform by using SubForm Wizard

- The next dialog window will display table relationships assumed by Access. Select one of these relationships or define your own and click *Next*.

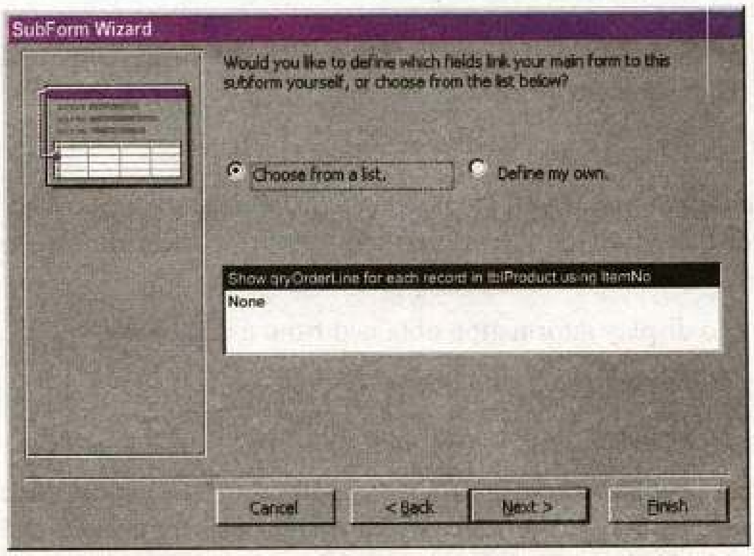

SubForm Wizard
Would you like to define which fields link your main form to this subform yourself, or choose from the list below?
Choose from a list.
Define my own.
Show qryOrderLine for each record in tblProduct using ItemNo
None
Cancel
< Back
Next >
Finish

Fig.7.27 Finishing Subform by SubForm Wizard

- On the final dialog box, enter the name of the subform by choosing option "Choose from list" option or by "define my own" and click *Finish*.

7.6 DRAG-AND-DROP METHOD

Use this method to create subforms from two forms that already exist. Make sure that the table relationships have already been set before proceeding with these steps.

- Open the main form in *Design View* and select *Window|Tile Vertically* to display both the database window and the form side-by-side as shown below.

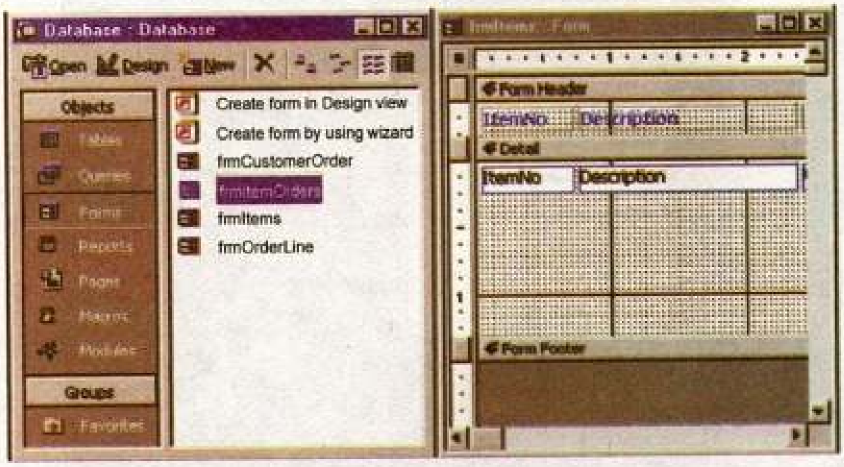


Fig.7.28 Defining Drag & Drop Method

Drag the form icon beside the name of the subform onto the detail section of the main form design.

7.7 REPORTS

Presentation of processed data obtained from a database is called report. The report can be displayed on the screen, on the paper or on the disk. The following are the main uses of reports:

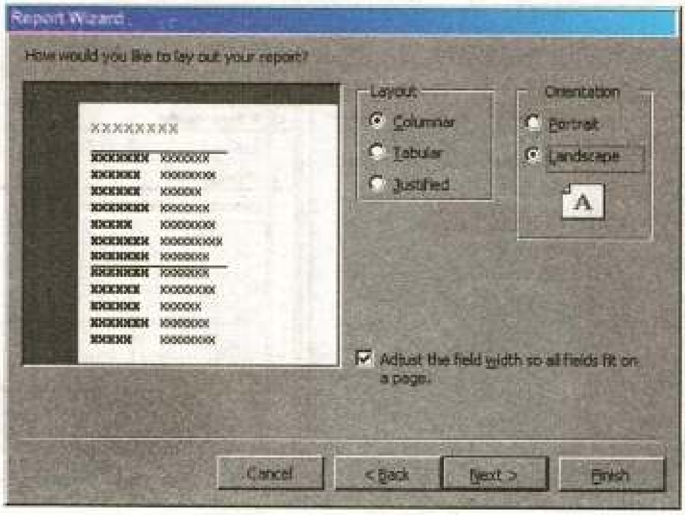
(i) To display information obtained from a database.

(ii) To display result of a query.

(iii) To produce output according to the needs of the user.

The final product of most database application is a report. Access combines data in tables and queries to produce a report that can be printed and distribute to people who need or request it. Reports provide means for creating printed copies of the information of database. Some reports consist of a single page, such as, order acknowledgement and invoice. Multi-page Access reports are more common than the single-page reports. These reports include catalogs, general ledgers, financial statements and examination result sheets. A report simply retrieved data from a database and presents it in a formatted form. It can retrieve data from one or more tables of a database. For example, in a school database, a report can be created to print the names and phone numbers of all students. The names and phone numbers of the students may store in two different tables. The report retrieved the data from the two tables and presents it in a predefined manner. Standard reports come in two basic varieties, that is, columnar and tabular.

Columnar Reports



Report Wizard

How would you like to lay out your report?

Layout: [✓ Columnar] Tabular Justified

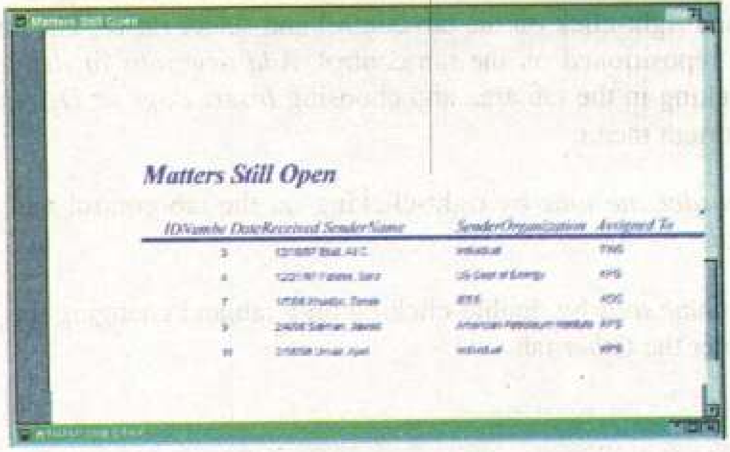
Orientation: Portrait [✓ Landscape]

[✓] Adjust the field width so all fields fit on a page.

Fig. 7.29(a) Colummar Report

In these reports, the values of each field in each record of a table or a query are listed in one long column of text boxes. A label indicates the name of the field and a text box to the right of the label provides the values. Columnar report spreads the information for a single record over many rows. Layout of a columnar report is shown below. It resembles the layout of a form.

Tabular Reports


Matters Still Open

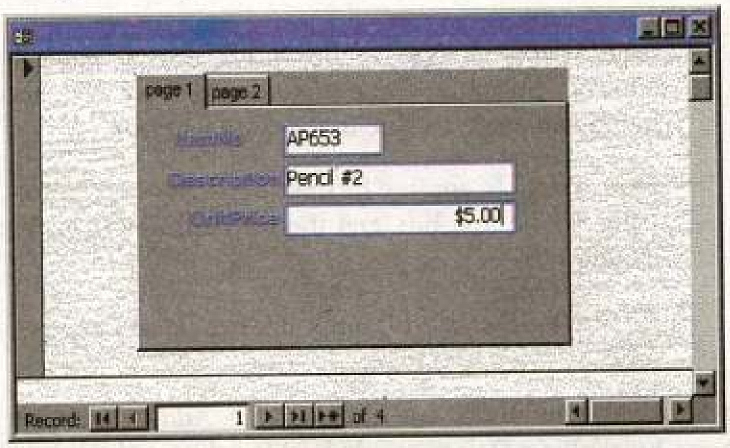
IDNumber	DateReceived	SenderName	SenderOrganization	Assigned To
3	12/18/97	Blatt, Al C.	Individual	[illegible]
4	12/21/97	[illegible]	US Dept of Energy	KPS
7	1/7/98	[illegible]	[illegible]	KDC
9	2/4/98	[illegible]	American Petroleum Institute	KPS
10	2/18/98	[illegible]	Individual	[illegible]

Fig.7.29(b) Tabular Report

These reports provide a column for each field of the records in rows under the column header. If you have more columns than can fit on one page, additional pages print in sequence until all columns are printed, then the next group of records is printed.

Multiple-Page Forms Using Tabs

Tab controls allow you to easily create multi-page forms. Create a tab control by following these steps:



PartNo	AP653
Description	Pencil #2
UnitPrice	$5.00

Record: 1 of 4

Fig. 7.29(c) Multiple page Report

- Click the *Tab Control* icon on the toolbox and draw the control on the form.

- Add new controls to each tab page the same way that controls are added to regular form pages and click the tabs to change pages. Existing form controls cannot be added to the tab page by dragging and dropping. Instead, right-click on the control and select *Cut* from the shortcut menu. Then right-click on the tab control and select *Paste*. The controls can then be repositioned on the tab control. *Add new tabs or delete tabs* by right-clicking in the tab area and choosing *Insert Page* or *Delete Page* from the shortcut menu.

- *Reorder the tabs* by right-clicking on the tab control and selecting *Page Order*.

- *Rename tabs* by double-clicking on a tab and changing the *Name* property under the *Other* tab.

Conditional Formatting

Special formatting that depends on the control's value can be added to text boxes, lists, and combo boxes. A default value can set along with up to three conditional formats. To add conditional formatting to a control element, follow these steps:

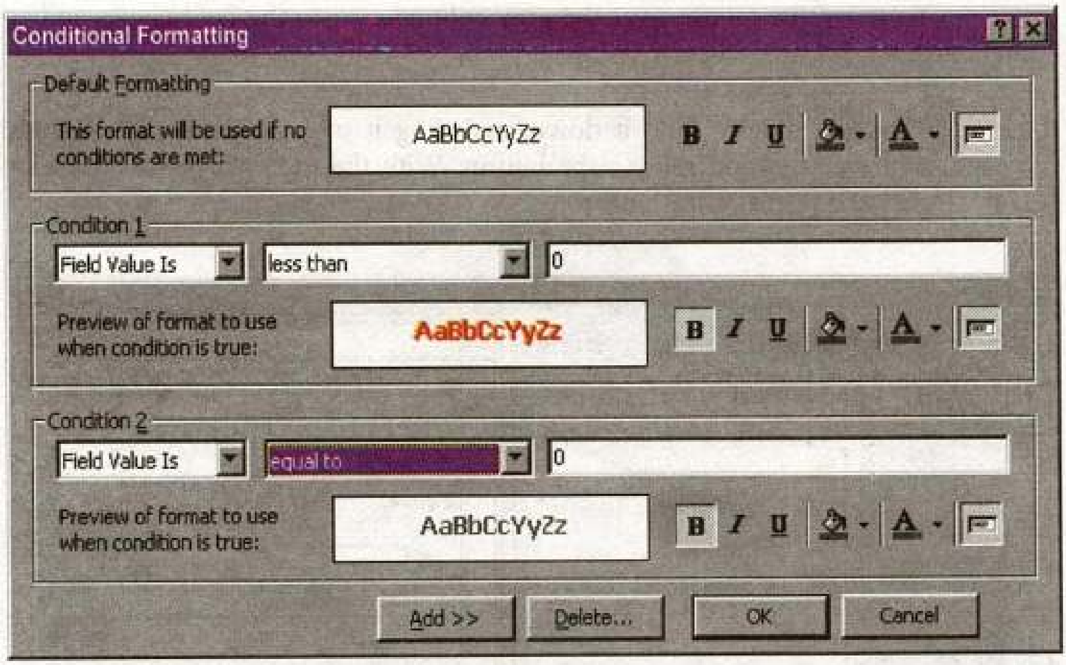
- Select the control that the formatting should be applied to and select *Format | Conditional Formatting* from the menu bar.
- Under *Condition* 1, select one of the following condition types:

    - *Field Value Is* applies formatting based upon the value of the control. Select a comparison type from the second drop-down menu and enter a value in the final text box.

    - *Expression Is* applies formatting if the expression is true. Enter a value in the text box and the formatting will be added if the value matches the expression.

    - *Field Has Focus* will apply the formatting as soon as the field has focus.

- Add additional conditions by clicking the *Add >>* button and delete conditions by clicking *Delete...* and checking the conditions to erase.



This format will be used if no conditions are met: AaBbCcYyZz

Field Value Is | less than | 0

Preview of format to use when condition is true: AaBbCcYyZz

Field Value Is | equal to | 0

Preview of format to use when condition is true: AaBbCcYyZz

Fig.7.30 Defining Conditional Formatting

Password Text Fields

To modify a text box so each character appears as an asterisk as the user types in the information, select the text field in Design View and click *Properties*. Under the *Data* tab, click in the *Input Mask* field and then click the button [...] that appears. Choose "Password" from the list of input masks and click *Finish*. Although the user will only see asterisks for each character that is typed, the actual characters will be saved in the database.

Change Control Type

If you decide the type of a control needs to be changed, this can be done without deleting the existing control and creating a new one although not every control type can be converted and those that can have a limited number of types they can be converted to. To change the control type, select the control on the form in Design View and choose *Format*|Change *To* from the menu bar. Select one of the control types that is not grayed out.

Composite Primary Keys

To select two fields for the composite primary key,

- To move the mouse over the gray column next to the field names and note that it becomes an arrow.

- Click the mouse, hold it down, and drag it over all fields that should be primary keys and release the button. With the multiple fields highlighted, click the primary key button.

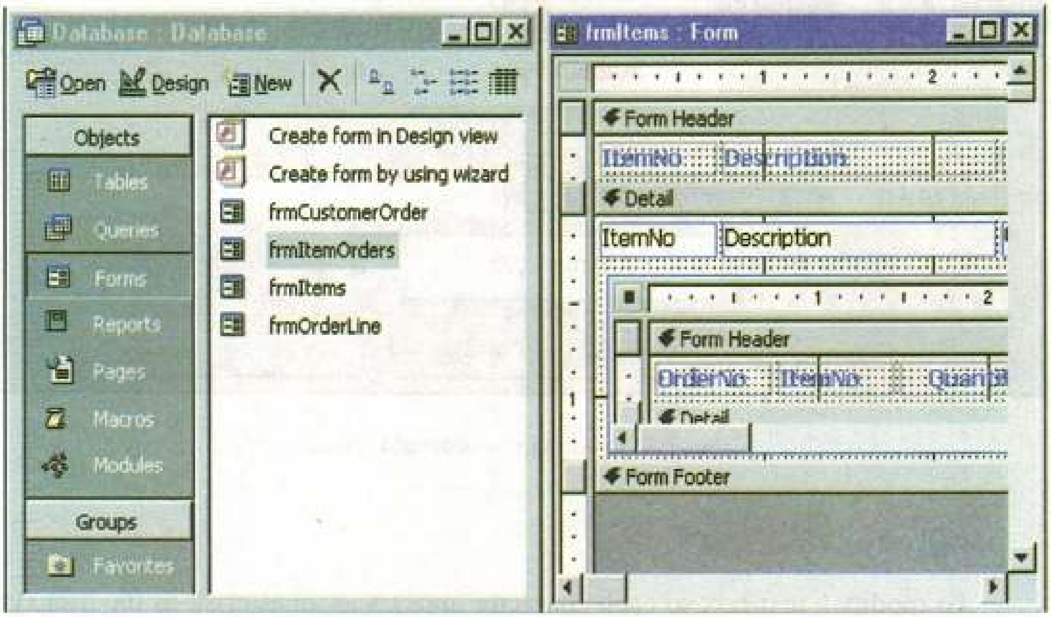


Fig.7.31 Showing composite Primary Key

Reports will organize and group the information in a table or query and provide a way to print the data in a database.

Using the Wizard

Create a report using Access' wizard by following these steps:

- Double-click the "Create report by using wizard" option on the Reports Database Window.
- Select the information source for the report by selecting a table or query from the *Tables/Queries* drop-down menu. Then, select the fields that should be displayed in the report by transferring them from the *Available Fields* menu to the *Selected Fields* window using the single right greater symbol button > to move fields one at a time or the double greater symbol button >> to move all of the fields at once. Click the *Next >* button to move to the next screen.

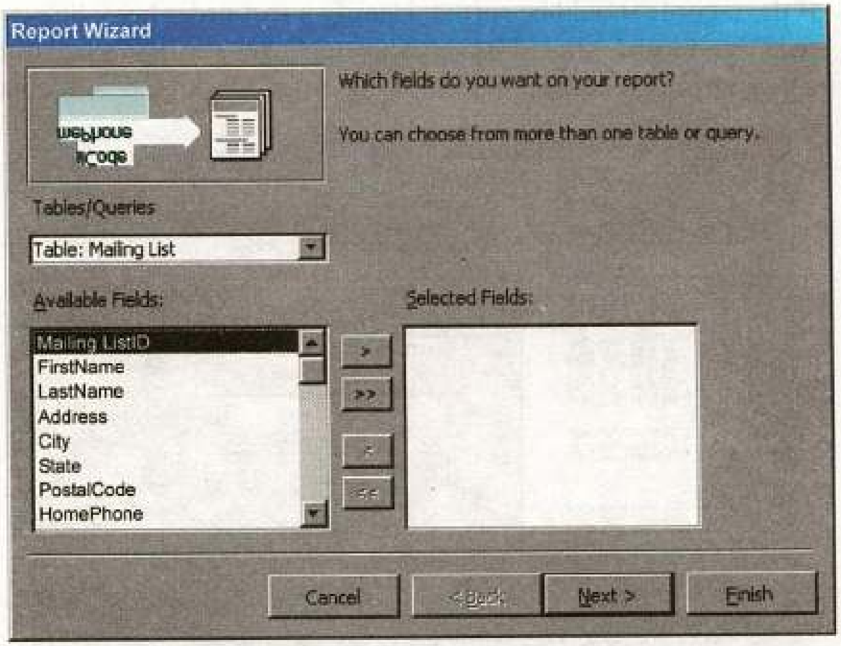


Fig.7.32 Choosing a table in Report Wizard

- Select fields from the list that the records should be grouped by and click the right greater symbol button > to add those fields to the diagram. Use the *Priority* buttons to change the order of the grouped fields if more than one field is selected. Click *Next* > to continue.

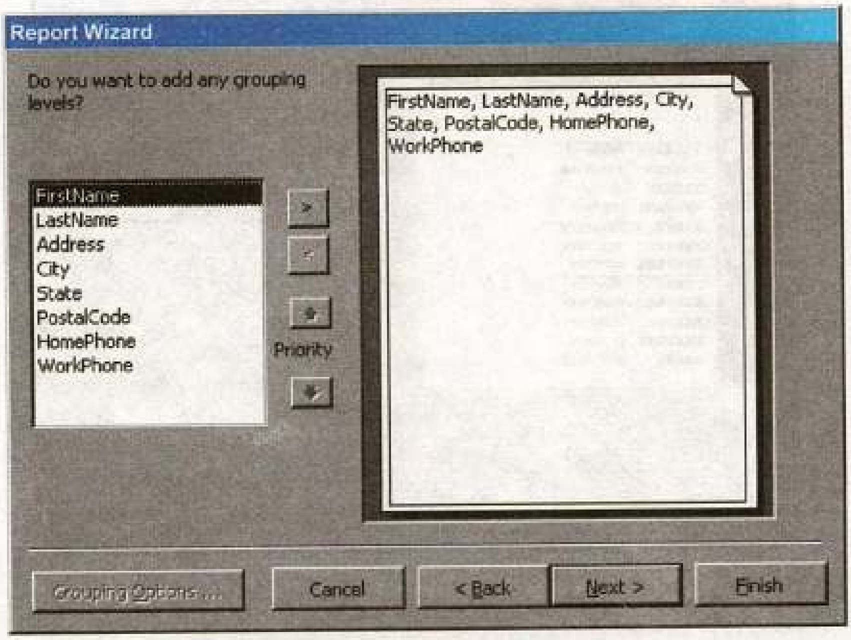
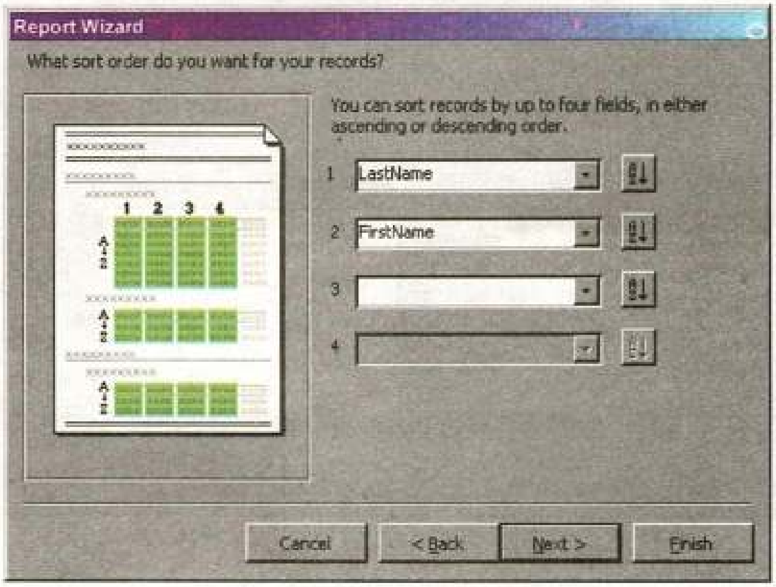


Fig.7.33.Grouping the fields in Report Wizard

- If the records should be sorted, identify a sort order here. Select the first field that records should be sorted by and click the A-Z sort button to choose from ascending or descending order. Click *Next >* to continue.


Report Wizard

What sort order do you want for your records?

You can sort records by up to four fields, in either ascending or descending order.

1 LastName

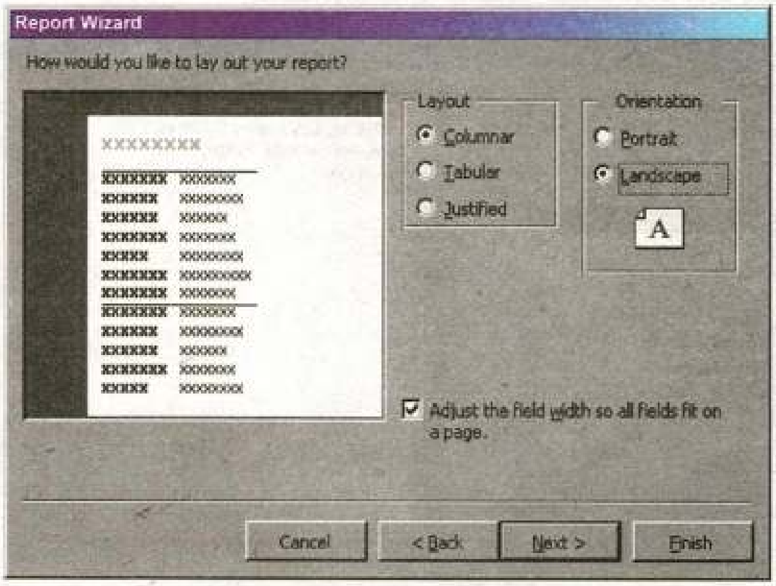
2 FirstName

3

4

Fig.7.34 Sorting the record in ascending and descending ordere in Report Wizard

- Select a layout and page orientation for the report and click *Next >*.


Report Wizard

How would you like to lay out your report?

Layout: [✓ Columnar] Tabular Justified

Orientation: Portrait [✓ Landscape]

[✓] Adjust the field width so all fields fit on a page.

Fig.7.35 Choosing a Layout and paper orientation in Report Wizard

- Select a color and graphics style for the report and click Next >.

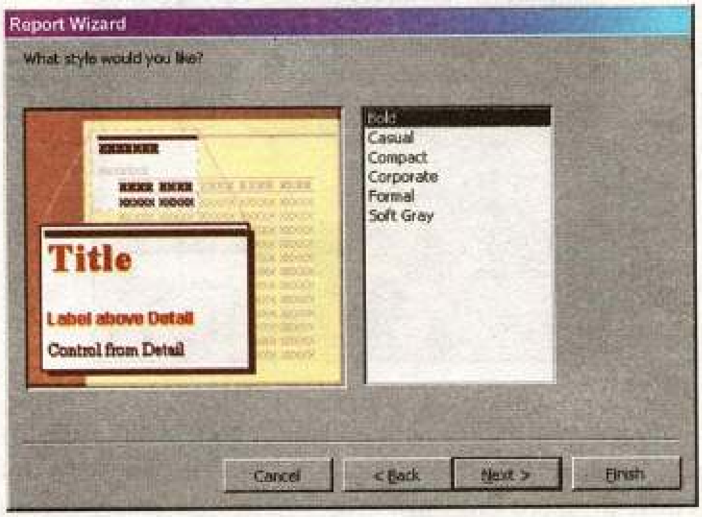
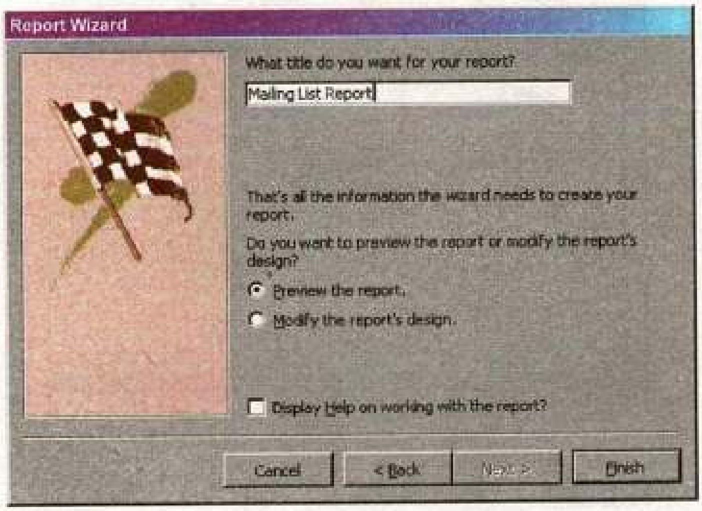


Fig.7.37 Choosing a style in Report Wizard

- On the final screen, name the report and select to open it when needed. Click the *Finish* button to create the report.



Mailing List Report

That's all the information the wizard needs to create your report.

Do you want to preview the report or modify the report's design?

[✓ Preview the report.]

Modify the report's design.

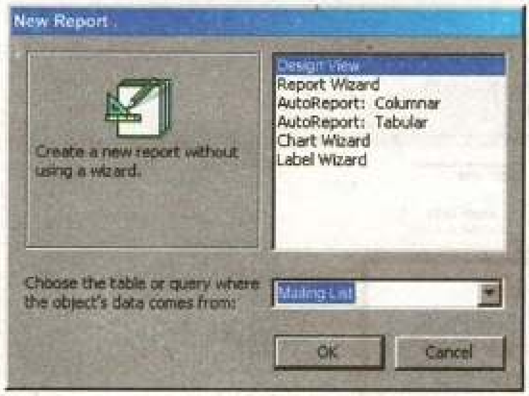
Display Help on working with the report?

Fig.7.35 Finishing in Report Wizard

Create in Design View

To create a report from scratch, select Design View from the Reports Database Window.

- Click the *New* button on the Reports Database Window. Highlight "Design View" and choose the data source of the report from the drop-down menu and click *OK*.


New Report

Create a new report without using a wizard.

Design View
Report Wizard
AutoReport: Columnar
AutoReport: Tabular
Chart Wizard
Label Wizard

Choose the table or query where the object's data comes from: Mailing List

OK Cancel

Fig.7.38 Choosing a table table by using Design View

- You will be presented with a blank grid with a Field Box and form element toolbar that looks similar to the Design View for forms. Design the report in much the same way you would create a form. For example, double-click the title bar of the Field Box to add all of the fields to the report at once. Then, use the handles on the elements to resize them, move them to different locations, and modify the look of the report by using options on the formatting toolbar. Click the Print View button at the top, left corner of the screen to preview the report.

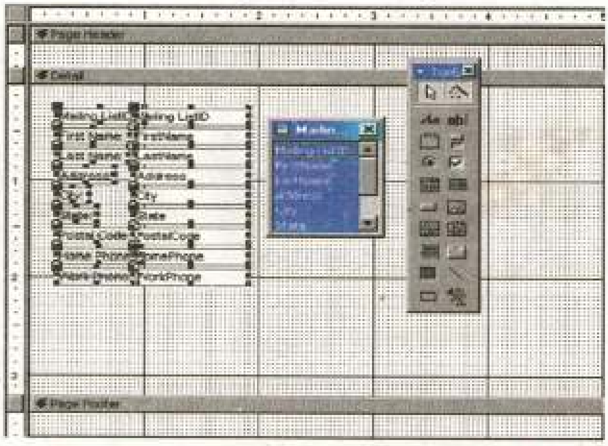


Fig.7.39 Creating report by using Design View

Printing Reports

Select *File|Page Setup* to modify the page margins, size, orientation, and column setup. After all changes have been made, print the report by selecting *File|Print* from the menu bar or click the *Print* button on the toolbar.

7.8 LINKING

Unlike importing, linking objects from another database will create a link to an object in another database while not copying the table to the current database. Create a link by following these steps:

- Open the destination database.
- Select *File|Get External|Link Tables...* from the menu bar.
- Choose the database that the table is located in and click the *Link* button.

A window listing the tables in the database will then appear. Highlight the table or tables that should be linked and click *OK*. A link to the table will appear in the Database Window as a small table icon proceeded by a small right arrow.

This is going to be the form that will be displayed when the application starts, and will enable the user to navigate to the other parts of the database. As such, it is only going to have buttons on it. Access does have a built-in Switchboard Manager Add-In. The switchboard form will also perform any required actions on the start-up of the database.

7.9 CREATING A SWITCHBOARD IN ACCESS

- Open one of your databases.
- Click on *Tools*, then *Database Utilities*, and finally *Switchboard Manager*.
- If no Switchboard form exists, Access will display a message telling you that no Switchboard exists and will ask if you want to create one.
- Click *Yes* to display the Switchboard Manager screen, and click the *Edit* button to edit your options.
- From this screen, you can change the Switchboard default name from Main Switchboard to whatever you'd like.
- Click the *New* button to display the Edit Switchboard Item dialog box.
- In the *Text* field, type a brief description of the first item you want to add.
- In the *Command* field, select the appropriate option from the drop-down list. The option you select will determine what options you'll receive in the third drop-down list.
- For example, if you choose Open Form In Edit Mode, the Switchboard Manager will display a list of your database's forms. Choose the form and click OK.
- Repeat this process until you've added all the items you want to the Switchboard form and click Close.

If you want to give users a second option for closing the Switchboard (besides the Close box), create a new Switchboard item named Exit and associate it with the command Exit Application; that option will close the Switchboard and the database. The final step is telling Access 2000 to run the Switchboard form when you open the database. To do so:

- Right-click on the database window and choose *Startup* from the context menu.
- Click the drop-down arrow for *the Display Form/ Page* option, choose *Switchboard*, and click *OK*.

The next time you open this database, Access will run the Switchboard form.

7.10 KEYBOARD SHORTCUTS

Keyboard shortcuts can save time and the effort of constantly switching from the keyboard to the mouse to execute simple commands. Print this list of Access keyboard shortcuts and keep it by your computer for a quick reference.

Note: A plus sign indicates that the keys need to be pressed at the same time.

Action	Keystroke
Database actions	
Open existing database	CTRL+O
Open a new database	CTRL+N
Save	CTRL+S
Save record	SHIFT+ENTER
Print	CTRL+P
Display database window	F11
Find and Replace	CTRL+F
Copy	CTRL+C
Cut	CTRL+X
Paste	CTRL+V
Undo	CTRL+Z
Help	F1
Toggle between Form and Design view	F5
Other	
Insert line break in a memo field	CTRL+ENTER
Insert current date	CTRL+;
Insert current time	CTRL+:
Copy data from previous record	CTRL+'
Add a record	CTRL++
Delete a record	CTRL+-

Action	Keystroke
Editing	
Select all	CTRL+A
Copy	CTRL+C
Cut	CTRL+X
Paste	CTRL+V
Undo	CTRL+Z
Redo	CTRL+Y
Find	CTRL+F
Replace	CTRL+H
Spell checker	F7
Toggle between Edit mode and Navigation mode	F2
Open window for editing large content fields	SHIFT+F2
Switch from current field to current record	ESC
Navigating Through a datasheet	
Next field	TAB
Previous field	SHIFT+TAB
First field of record	HOME
Last field of record	END
Next record	DOWN ARROW
Previous record	UP ARROW
First field of first record	CTRL+HOME
Last field of last record	CTRL+END

Exercise 7c

1. Fill in the blank:

(i) Forms are used to __________, __________ and ____________ data in database.

(ii) The easiest and quick way to create a form in Microsoft Access is by using ___________.

(iii) ____________ is used to display multiple records at a time in tabular format.

(iv) A _________ is a form, which is displayed within the main form.

(v) There are basically ______ layouts of forms in Microsoft Access.

(vi) The _______ object is used only to retrieve data from a database and display it on the screen or print it on the printer. It cannot be used to edit data in a database.

(vii) _________ are can compare, summarize, and sùbtotal large sets of data.

(viii) In Microsoft Access, a report can be created in any one of the three layouts. These layouts are _________, __________, and __________.

(ix) Reports are the finished result of your data input to the database backed through the forms as ___________ end.

(x) The ____________ object is used only to retrieve data from a database and displays it on the screen or print it on the printer. It cannot be used to edit data in a database.

2. Choose the Correct Option.

(i) Forms are designed for :

a) Input Data b) Manipulate Data
c) Accepting Change d) All of them

(ii) A form that contains the sub form is called ____________.

a) Form b) Main Form
c) Report d) None of them

(iii) You can drag the __________ bar to move the property sheet window around on your screen.

a) Title bar b) Status bar
c) Scroll bar d) All of them

(iv) How many are basic layouts of forms in Microsoft Access.

a) 2 b) 3
c) 4 d) 5

(v) The forms are the ________end of our database in Microsoft Access.

a) Back end b) Front end
c) both and b d) None of them

(vi) A __________ Auto form displays one record at a time..

a) Tabular b) Columnar

Chapter 8 GETTING STARTED WITH C

8.1 OVERVIEW

A computer is a device that follows the instructions given to it. A well-defined set of instructions given to the computer is called a *computer program.* A computer program is written in a programming language. Since the emergence of computer, many programming languages have been developed but the effect of C on the computer world is everlasting. This book will remain incomplete without describing the history of the C. That's why before going into detail; let us have an overview of the history of C.

History of C

The **C** programming language was developed by Dennis Ritchie in 1972 at AT & T Bell Laboratories. It was derived from an earlier programming language named *B*. The B was developed by Ken Thompson in 1969-70 and provided the basis for the development of C. The C was originally designed to write system programs under UNIX®operating system. But over the years its power and flexibility have made it popular in industry for a vide range of applications. The earlier version of C was known as K&R (Kernighan and Ritchie) C. As the language further developed, the ANSI (American National Standards Institute) developed a standard version of the language known as ANSI C.

8.2 DEVELOPING A C PROGRAM (A STEPWISE APPROACH)

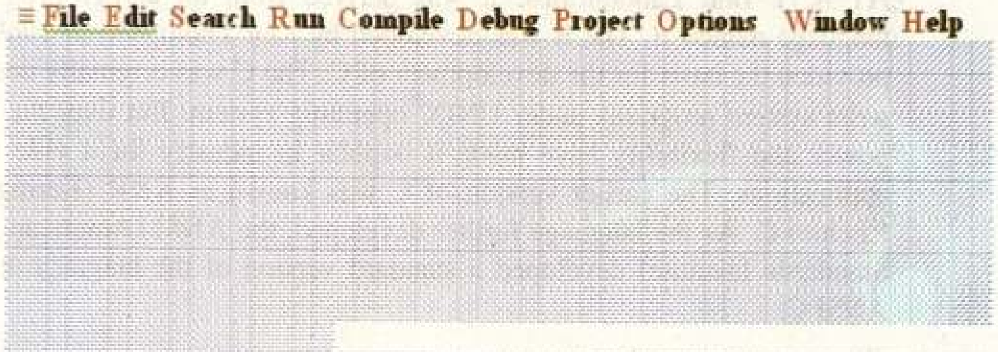
Writing a program in C is not too difficult; however it requires a good understanding of the development environment of C language. The programmer should also have the knowledge of steps required to prepare a C program for execution.

As a first step, install a compiler for the C language on the computer so that the source program can be compiled and executed. Many compilers for C language are available from number of vendors. Any of them can be used, but we recommend using Turbo C++.

8.2.1 Turbo C++ (A Compiler for the C language)

Turbo C++ is a Borland International's implementation of a compiler for C language. In addition to a compiler, TC provides a complete IDE (Integrated Development Environment) to create, edit and save programs is called TC editor (Fig. 8.1). It also provides a powerful debugger that helps in detecting and removing errors in the program.

Once the TC (Turbo C) has been installed, it is very easy to write C programs in its editor. The IDE can be invoked by typing **tc** on the DOS prompt or by double clicking the TC shortcut. The menu bar of the IDE contains menus to create, edit, compile, execute (Run) and debug a C program. A menu can be opened by either clicking the mouse on it or pressing the first highlighted character of the name of the menu in conjunction with the *Alt* key. For example to open *File* menu, press *Alt+F* (hold down Alt key and then press F key).


≡ File Edit Search Run Compile Debug Project Options Window Help

8.2.2 Creating and Editing a C Program

To write the first C program, open the *edit* window of the Turbo C++ IDE. This can be done by selecting *File|New* option from the menu bar. A window appears on the screen (Fig. 8.2). This window has a double-lined border, and a cursor inside the window represents the starting point to write a program.

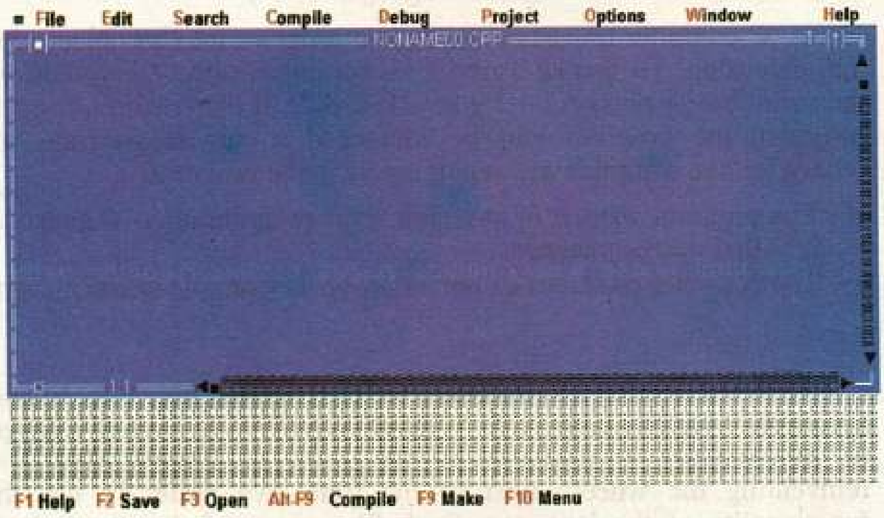


Fig. 8.2 Create, Edit and Save a Program

We can expand this window by clicking the arrow in the upper right corner, or by selecting *Window|Zoom* from the menu bar. We can also navigate through the program by using the vertical and horizontal scroll bars or by using arrow keys.

8.2.3 Saving a C Program

After writing the C program, we should save it on the disk. This can be done by selecting *File|Save* command from the menu bar or pressing the *F2* key. When we select *File|Save*, a dialogue box will appear. At the top of this dialogue box, there is a text box with caption *Save File As*. Type the name of the file in it and press the Enter key. The default path for saving the file is BIN folder. The TC assigns a default name NONAME00.cpp to the file (Fig. 8.2). To save the file in a specific folder / location with a different file name, one has to specify the absolute path.

Note:

Turbo C++ is a compiler for C++ programming language – an extension to C. Therefore it can compile programs of both C and C++. When we save a program with .cpp extension, it can use many additional features that are not supported in ANSI C. *As this course is designed just for C, not C++, therefore it is suggested to always save the programs with .c extension.* When a program is saved with .c extension, the Turbo C++ compiler restricts it to only use standard features of C.

8.2.4 Compiling a C Program

The computer does not understand source program because instructions in the program are meaningless to the microprocessor, as it understands only the machine language. A program that is to be executed must be in the form of machine language.

C compiler translates the source program into an object program with .obj extension. To invoke Turbo C++ compiler, select *Compile|Compile* from the menu bar or press *Alt + F9* key (Fig. 8.2). If there is no error in the source program, the program will be translated to object program successfully otherwise, the compiler will report errors in the program.

- The program written in any high level programming language, such as C, is called *source program.*
- The compiler produces an *object program* from the source program

8.2.5 Linking a C Program

While writing a C program, the programmer may refer to many files to accomplish various tasks such as input/output etc. In case of C language, a lot of functionality is available in the form of library files. Rather than reinventing the wheel, most of the times we prefer to use the built-in functionality of the language. Such files are needed to be linked with the object file, produced by the compiler, before execution of the program.

Linking is the process in which the object file produced by the compiler is linked to many other library files by the linker. The *linker* is a program that combines the object program with additional object files that may be needed for the program to execute and save the final machine language program as an executable file on disk. In Turbo C++, the linker can be invoked by selecting *Compile|Link* from the menu bar.

The *Linker* combines different library files to the object file and produces an executable file with .exe extension

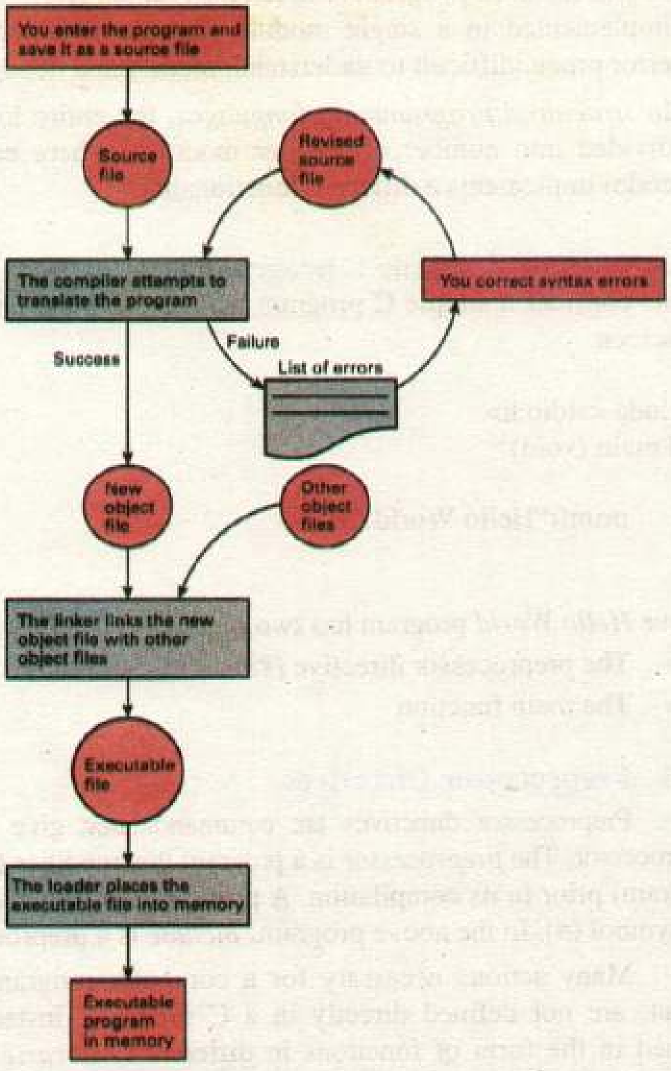
8.2.6 Executing a C program

After successfully compiling and linking the program, we are now ready to execute it. For execution the program must be loaded into memory. This is done by the loader. Loader is a program that places executable file in memory. In Turbo C++, this is done by selecting *Run|Run* from the menu bar or pressing *Ctrl+F9* key.

When the program is run, the screen flickers for a moment and the output screen will disappear in a flash. To see the program's output select *Window|User Screen* or press *Alt+F5*. The normal DOS output screen will appear. Flowchart 8.3 describes the steps required to prepare a C program for execution.

Setting the Output and Source Directories

By default, Turbo C++ places the object and executable files in the BIN subdirectory of the TC directory. This is not the right place to put these files. These files should be placed in the same directory where the source file (with .c extension) was created. To do so, select the *Option|Directories* from the menu bar. A window appears with four fields captioned *Include Directories, Library Directories, Output Directories* and *Source Directories*. The Include Directories filed should already be set to drive:\TC\INCLUDE and Library Directories should be set to drive:\TC\LIB, where the *drive*: is the drive in which the directory TC is placed. It can be C, D, or E etc. We need to set the *output directory* field to source file directory e.g., D:\MyPrograms etc. this is where the compiler will put .obj file and the linker will put .exe file.


You enter the program and save it as a source file
Source file
Revised source file
The compiler attempts to translate the program
You correct syntax errors
Failure
Success
List of errors
New object file
Other object files
The linker links the new object file with other object files
Executable file
The loader places the executable file into memory
Executable program in memory

8.3 BASIC STRUCTURE OF A C PROGRAM

The structure of a C program is very flexible which increases the power of the language. C is a structured programming language; therefore it provides a well-defined way of writing programs. As discussed earlier in this chapter that a C program is combined with many other files before execution. The linker does this job. But we have to specify these files to be linked. How can this be done? To answer this question and to understand the basic structure of the C program, we proceed with the following example:

- In *unstructured programming languages*, the entire logic of the program is implemented in a single module (function), which causes the program error prone, difficult to understand, modify and debug.
- In *structured programming languages*, the entire logic of the program is divided into number of smaller modules, where each module (piece of code) implements a different functionality.

Hello World – A simple C program

Let us consider a simple C program that displays the phrase *Hello World*! on the screen.

```
#include <stdio.h>
void main (void)
{
        printf("Hello World!");
}
```

The above *Hello World* program has two parts:

- The preprocessor directive (#include <stdio.h>)
- The main function

8.3.1 Preprocessor Directives

Preprocessor directives are commands that give instructions to the C preprocessor. The *preprocessor* is a program that modifies the C program (source program) prior to its compilation. A preprocessor directive always begins with the symbol (#). In the above program, *include* is a preprocessor directive.

Many actions necessary for a computer program, such as input and output, are not defined directly in a C program. Instead, these actions are defined in the form of functions in different C *libraries*. Each library has a standard header file, which is referred to with *.h* extension. In the above program, the *stdio.h* refers to the header file containing the definition of standard input/output functions.

The *include* directive gives a program access to a library. This directive causes the preprocessor to insert definitions from a standard header file into a program before compilation. Hence, the statement **#include<stdio.h>** gives the program access to standard input and output functions.

# include Directive for Defining Identifiers from Standard Libraries

SYNTAX:	#include<*standard header file*>
EXAMPLE:	#include <stdio.h>
	#include <math.h>

The *include* directive tells the compiler where to find the meanings of standard identifiers (e.g., *printf* in the Hello World program) used in the program. These meanings are described in files called *standard header files*. The header file *stdio.h* contains information about standard input and output functions such as *scanf* and *printf*, whereas the header file *math.h* contains information about common mathematical functions.

Another important preprocessor directive is #define directive. It is used to define a constant *macro*. Examples of this macro will be discussed in subsequent chapters.

#define Directive for Defining Constant Macros

SYNTAX:	#define Macro_Name		*expression*
EXAMPLE:	#define	PI	3.142857
	#define	SEC_PER_HR	3600

The *expression* may be constant, arithmetic expression or a string. C preprocessor replaces each occurrence of the identifier *Macro_Name* with *value of expression*. The expression of the identifier *Macro_Name* can not be changed during the program execution.

Constant Macro is a name that is replaced by a particular constant value before the program is sent to the compiler.

8.3.2 FUNCTION main

As shown in the above *Hello World* program, the definition of the *main* function comes next to the specification of the *#include* preprocessor directive. In fact, *main* is the function where the execution of the C program begins. Every C program has a *main* function. The rest of the lines of program forms the *body* of the main function, the body is enclosed in braces { and }.

C programs are divided into units called functions. This division is usually done on the basis of functionality, where every function carries out a single task. However, it is no necessary to divide every program into functions. The same functionality may be achieved through a single function. But, every C program must have the functions *main* as the execution of the program starts from there. In this way we can say that the main function is actually the entry point of the C programs.

main Function Definition

SYNTAX:

```
void main (void)
{
        body of main function
}
```

As we know from the algebra that every function returns a single value and may accept one or more arguments (parameters). There is some resemblance between an algebraic function and the *main* function. The definition of the function *main* starts with a reserved word *void*. This *void* represents the data type of the value that is returned by the function, which means the function main returns nothing. The second **void** enclosed in parenthesis describes that the function *main* does not accept any argument. However arguments can be passed to the main function and it can also return a value. But the discussion of this issue is out of scope of this book. You may find the topic in detail in many other books.

Body of the function (enclosed in braces) consists of C language statements, which are used to implement the program logic. There are many types of C statements that help programmers to write C programs. We shall learn much more about writing programs in C in next chapters.

8.3.3 Delimiters

Next to the function definition are braces, which indicate the beginning and end of the function body. These braces are called *delimiters*. The opening brace { indicates the beginning of a block of code (set of statements) while the closing brace } represents the end of a block of code.

8.3.4 Statement Terminator

Every statement in a C program terminates with a *semicolon* (;). If any of the statement is missing the statement terminator, the compiler will report it the following error message.

```
Statement missing ;
```

Note: Always be careful about the semicolon while writing C program statement.

8.3.5 Function printf

The last statement in the *Hello World* program is *printf* function. It is used to display the output of the program on the screen. See detail in chapter 3.

8.4 COMMON PROGRAMMING ERRORS

The programmer may come across errors while writing a computer program. In programming languages, these errors are called "bugs", and the processing of finding and removing these bugs is called *debugging*.

When the C compiler detects an error, it displays an error message describing the cause of the error. There are three types of programming errors, these are: Syntax error, Runtime error, and Logic error.

8.4.1 Syntax Errors

A syntax error occurs when the program violates one or more grammar rules of C language. The compiler detects these errors as it attempts to translate the program. If a C statement has syntax error, it can not be translated and the program could not be executed.

There can be many causes of syntax errors, for example, missing statement terminator i.e., the semicolon, using a variable without declaration, missing any of the delimiters i.e., { or } etc.

8.4.2 Runtime Errors

A runtime error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero. Runtime errors are detected and displayed by the computer during the execution of a program. When a runtime error occurs, the computer stops executing the program and displays a diagnostic message.

8.4.3 Logical Errors

Logical errors occur when a program follows a faulty algorithm. The compiler can not detect logical errors; therefore no error message is reported from the compiler. Moreover, these errors don't cause the program to be crashed, that's why these are very difficult to detect. One can recognize logical errors by just looking at the wrong output of the program. Logical errors can only be detected by thorough testing of the program.

8.5 PROGRAMMING LANGUAGES

Programming languages are used to write computer programs. There are two broad categories of programming languages i.e., low level programming languages and high level programming languages. We discuss them briefly to have an overview of both:

8.5.1 Low Level Languages

Low level languages are divided into two broad categories i.e., machine language and assembly language. *Machine language* is the native language of the computer. The computer does not need any translator to understand this language. Programs written in any other language must be converted to machine language so that the computer can understand them. Every machine language instruction consists of strings of binary 0s and 1s. As it is very difficult for human beings to remember long sequences of 0s and 1s, therefore writing programs in machine language are very difficult and error prone. So, it was thought to replace the long sequences of 0s and 1s in machine language with English like word. This idea provided the basis for the development of assembly language.

In *assembly language*, machine language instructions (long sequences of 0s and 1s) are replaced with English like words known as mnemonics (pronounced as Ne-Monics). An assembler (language translator for assembly language programs) is used to translate an assembly language programs into machine language.

8.5.2 High Level Languages

Programming languages whose instructions resemble the English language are called *high level languages*. Every high level language defines a set of rules for writing programs called *syntax* of the language. Every instruction in the high level language must confirm to its syntax. If there is a syntax error in the program, it is reported by the language translator (compiler or interpreter). The program does not translate into machine language unless the error is removed.

Common high-level languages include C, C++, Java, Pascal, FORTRAN, BASIC, and COBOL etc. Although each of these languages was designed for a specific purpose; all are used to write variety of *application software*. Some of these languages such as C and C++ are used to write *system software* as well. Each of these languages has some advantages and disadvantages over the other e.g., FORTRAN has very powerful mathematical capabilities while the COBOL is ideal for writing business applications, C and C++ are very handy for writing system software while Java is equipped with strong network programming features. Besides having different features, all high level programming languages have some common characteristics are:

- These are English like languages, hence are close to human languages and far from the machine language and are very easy to learn
- Programs written in high level languages are easy to modify and debug, and more readable
- These languages let the Programmers concentrate on problem being solved rather than human-machine interaction
- These describe a well defined way of writing programs
- These do not require a deep understanding of the machine architecture
- High level languages provide machine independence. It means programs written in a high level language can be executed on many different types of computers with a little modification. For example, programs written in C can be executed on Intel® processors as well as Motorola processors with a little modification.

Exercise 8c

1 Fill in the blanks:

(i) The set of instruction given to the computer to solve any kind of problem is called ______________

(ii) ANSI stands for ______________________

(iii) The program written in high level language is known as __________

(iv) ________ is a program that places the executable file in memory

(v) In __________programming language , the entire logic of the program is implemented in a single module

(vi) ___________are command that give instruction to C preprocessor

(vii) __________ is a name that is replaced by particular constant before program is sent to the compiler

(viii) The _________ directive gives a program access to a library file

(ix) C program is divided into units, called ___________

(x) Every statement in a C program terminates with a ______

(xi) A language translator for Assembly language is called ___________

(xii) A set of rule for writing program in high level language is known as____________

2 Choose the correct option:

(i) C is a:
a) High Level Language b) Low Level Language
c) Assembly Language d) Machine Language

(ii) Turbo C++ can compile:
a) C++ programs only b) C and C++ programs
c) Turbo C programs only d) Turbo C++ programs only

(iii) Debug is the process of:
a) Creating bugs in program b) Identifying and removing errors
c) Identifying Errors d) Removing Errors

(iv) C was designed to write programs for:
a) Windows operating system b) Solaris operating system
c) Unix operating system d) OS/2 operating system

(v) Preprocessor directives are commands for:
a) Microprocessor b). Language processor
c) C preprocessor d) Loader

(vi) The expression in define directive:
   a) can only be changed at the end of the program
   b) can not be changed
   c) can not be changed but can be redefine
   d) can not be assigned a value

(vii) Which of the following language requires no translator to execute the program:
   a) C b) C++
   c) Machine language d) Assembly language

(viii) .exe file is produced by the:
   a) Linker b) Loader
   c) Compiler d) Interpreter

(ix) Which of the following key is used to save a file?
   a) F2 b) F3
   c) F5 d) F9

(x) void occupy how many bytes in memory?
   a) zero b) one
   c) two d) four

3 Write T for true and F for false statement:

(i) The C programming language was developed by Dennis Ritchie in 1972.
(ii) C was derived from earlier programming language named B.
(iii) The B was developed by Ken Thomson in 1980.
(iv) The short-key for compiling a program is Alt+F9.
(v) The compiler produces the source file from an object file.
(vi) The linker is a program that combines the object program with additional files.
(vii) The short-key for executing the C program is Alt + F5.
(viii) In structured programming the entire program is divided into smaller modules.
(ix) The value of a constant can be changed during the program execution.
(x) High level language provide machine independence.

4 Briefly describe the history of C.

5 List two reasons why it would be preferable to write a program in C rather than machine language.

6 What necessary steps taken to prepare a C program for execution? Explain with diagram.

7 Define a *bug*. Discuss some debugging features of Turbo C++.

8 While writing a C program, how many types of errors can occur? Discuss briefly. Which one is the most difficult to locate and remove? Justify your answer.

9 What is a programming language? Discuss the two main categories of programming languages.

10 Describe characteristics of high-level programming languages.

11 Briefly describe the basic structure of a C program.

12 How would you create, edit, compile, link and execute a C program? Discuss briefly.

13 Differentiate the following:

(i) Preprocessor Directive and the Compiler

(ii) Structured and Unstructured programming languages

(iii) Linker and Loader

ELEMENTS OF C

Chapter **9**

9.1 OVERVIEW

Computer does not do anything on its own. The most important skill to learn is how intelligently one can program it. It can be used to solve multidimensional problems. The C being a magic tool for writing programs is used to solve variety of problems; a beginner just need practice and more practice of writing programs to master it.

Here we shall introduce the basic elements of C language. These are actually the building blocks of every C program. The proper use of these basic elements helps a lot to write effective C programs.

9.1.1 Identifiers

Identifiers are the names used to represent variables, constants, types, functions, and labels in the program. Identifiers in C can contain any number of characters, but only the first 31 are significant to C compiler. There are two types of identifiers in C: standard identifiers and user-defined identifiers.

9.1.2 Standard Identifiers

Like reserved words, standard identifiers have special meanings in C, but these can be redefined to use in the program for other purposes, however this practice is not recommended. If a standard identifier is redefined, C no longer remains able to use it for its original purpose. Examples of standard identifiers include *printf* and *scanf*, which are names of input/output functions defined in standard input/output library i.e., *stdio*.h.

9.1.3 User-defined Identifiers

In a C program, the programmer may need to access memory locations for storing data and program results. For this purpose memory cells are named that are called *user-defined identifiers*.

C is a *case sensitive* language. This means that C compiler considers uppercase and lowercase letters to be distinct characters. For example, the compiler considers SQUARE_AREA and Square_Area as two different identifiers referring to different memory locations.

9.2 KEYWORDS

Keywords or *reserved words* are the words, which have predefined meaning in C. There are 32 words defined as keywords in C. These have predefined uses and cannot be used or redefined for any other purpose in a C program. They are used by the compiler as an aid to compile the program. They are always written in lower case. A complete list of ANSI C keywords is given at the end of this chapter.

Keywords or **reserved words** cannot be redefined in the program.

Variables

Variables are named memory locations (memory cells), which are used to store program's input data and its computational results during program execution.

As the name suggests, the value of a variable may change during the program execution. We are familiar with the concept of *variable* with reference to algebra. The variables are created in memory (RAM); therefore the data is stored in them temporarily. One should not mix the contents of a variable with its address. These are entirely different concepts. The contents of a variable can be thought of as the residents of your neighboring house, while the address of the variable can be thought of as the address of that house (Fig. 9.1).

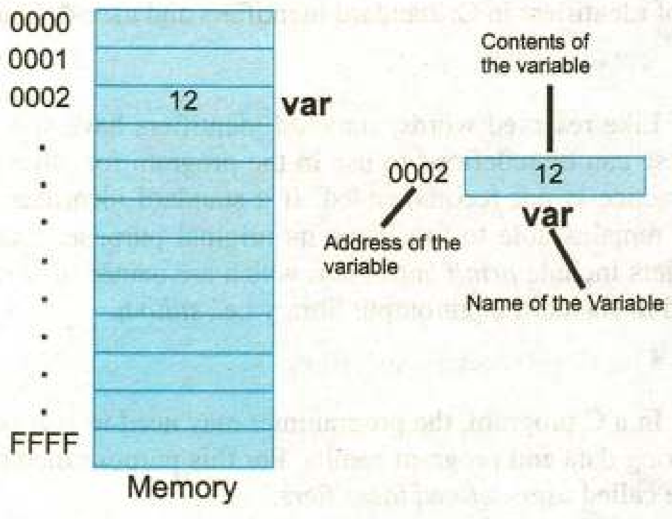


Fig. 9.1: Variable in memory

9.2.1 Declaring a Variable in C

C is a *strongly typed* language i.e. all variables must be declared before being used. The compiler will report an error if an undeclared variable is used in a program. A variable is declared in C by specifying its type (data type) and name. The syntax takes the form:

```
data type        variable_name;
```

data type refers to the type of the data being stored in the variable. For example

```
int       kgs;
double    length;
```

A list of names of variables separated by commas is specified with the *data type* to declare more variables of the same data type such as:

```
int         marks, total_students, no_of_rooms;
double      kgs, length, volume, height;
```

9.2.2 Declaring vs Defining a Variable

Variable declaration tells the compiler the name of the variable to be used in the program and the type of information stored in it. In a C program, the

```
int  volume;
char    ch;
```

variable declarations tell the compiler the name of two variables (volume and ch) used to store an integer and character data respectively.

A variable declaration does not set aside memory location for the data to be stored. It just informs the compiler the name of the variable and the type of data to be stored in it, while the *definition of the variable* that set aside memory location for the variable.

However in C, the variable declaration statement not only declares the variable but also defines it as in case of above two statements. It does not mean that the declaration of a variable can not be separated from its definition in C. C is such a powerful language that provides us all what we need for developing a program. But the discussion is out of scope of this book. We will learn more on this topic in next classes.

9.2.3 Initializing a Variable

Assigning a value to a variable at the time of its declaration is called initializing a variable. In a C program, when a variable is declared, the compiler set aside some memory space for it. This allocated memory space may contain data meaningless to the program (also called garbage). The computations involving this variable may produce unexpected results. To avoid this situation, all variables should be declared before being used. In C, a variable can also be initialized at the time of its declaration e.g.,

```
int     count;      (variable declaration and definition)
count = 125;        (variable initialization)
char    ch = 'z';   (variable declaration, definition and initialization)
float   weight = 75.8;
```

9.2.4 Rules for Naming Variables

To use variables in C programs, we must know the rules to choose names for them. Here are some important rules for naming a variable in C:

- A variable name can consists of letters, digits, and the underscore character (_)
- The first character of the name must be a letter. The underscore is also a legal first character, but its use is not recommended. Therefore, 9winner, #home, and 2kgms are invalid names in C.
- C is a case sensitive language. Thus, the names *count* and *COUNT* refers to two different variables.
- C keywords can't be used as variable names e.g., we can not use **int, void,** *signed*, or *while* as variable names.
- For many compilers, a C variable name can be up to 31 characters long. (It can actually be longer than that, but the first 31 characters of the name are significant) e.g., Turbo C++ restricts the maximum length of a variable name to 31 characters. Hence, the names *problem_solving_techniques_in_C* (31 characters) and *problem_solving_techniques_in_C_language* (40 characters) would appear to be the same to the compiler. The compiler does not differentiate these two names because the first 31 characters of both are the same.
- Blank spaces are not allowed in the name e.g., *problem solving* is an invalid variable name in C.
- A variable can only be declared for one data type.

C programmers commonly use only lowercase letters in variable names, although this isn't required. Using all-uppercase letters is usually reserved for the names of constants.

Variable Names should be Readable

Let's consider a program that calculates loan payments could store the value of the prime interest rate in a variable named interest_rate. The variable name helps make its usage clear. We could also create a variable named *x* or even *Ahmed*; it doesn't matter to the C compiler. Many naming conventions are used for variable names. We've seen one style: interest_rate. Using an underscore to separate words in a variable name makes it easy to interpret. Another style is to capitalize the first letter of each word. Instead of interest_rate, the variable would be named InterestRate.

9.4 CONSTANTS

A constant is a quantity whose value can not be changed. Unlike a variable, the value stored in a constant can't be changed during program execution. As discussed in chapter 1, the *define directive* can be used to define constant macros, for example:

```
#define   PI   3.142857
```

defines a constant i.e. *PI*, whose value will remain unchanged during the program execution. C has two types of constants; numeric constants and character constants, each with its own specific uses.

9.4.1 Numeric Constants

Numeric constants consist of numbers. It can be further divided into integer and floating point constants. Integer constants represent values that are counted and do not have a fractional part e.g., +56, -678, 8, etc. Floating point constants represent values that are measured e.g., - 4.786, 5.0, 0.45 etc.

9.4.2 Character Constants

A character constant is a single alphabet, a single digit or a single symbol enclosed within apostrophes e.g., 'A' is a valid character constant. e.g., 'A', 'I', '5','=' etc. The maximum length of a character constant is 1 character.

9.5 DATA TYPE

In C, the *data type* defines a set of values and a set of operations on those values. We know that computer program manipulates various types of data. The data is given to the program as input. The data is processed according to the program instruction and output is returned. In program designing, the data and its type are defined before designing the actual program used to process the data. The type of each data value is identified at the beginning of program design. The values or data used in a program may be of different types.

In a C program, we may need to process different type of data. Therefore, variables should be capable of storing data of different types. This is done by associating certain data types to the variables.

To work with different types of data, C defines some standard data types and also allows us to define our own data types known as *user-define data types*. In C, a standard data type is one that is predefined in the language such as int, double, char, float etc. Let's look at some of the important standard data types:

9.5.1 Data Types for Integers (int, short, long)

In C, the data type **int** is used to represent integers – the whole numbers. This means that *int* variables store number that have no fractional parts such as 1128, 1010, 32432 etc. Along with standard type *int*, the compiler also supports two more types of integer i.e. *short int* and *long int*, often abbreviated to just *short* and *long*. In addition to these types, an integer variable can be *signed* or *unsigned*. If not mentioned, all integer variables are considered to be signed. The data types *signed int* and *int* can handle both signed and unsigned whole numbers such as 245, 1010, and -232 etc, where as the data type *unsigned* can not handle negative numbers.

Because of the limited size of the memory cell, all integers can not be represented by *int*, *short* or *long*. When a variable of type *int* is declared, the compiler allocates two bytes of memory to it. Therefore, only the numbers ranging from -2^{15} through $2^{15} - 1$ (i.e. -32768 to 32767) can be represented with the *int* type variables. The variable of type *unsigned int* can handle numbers ranging from 0 through 2^{16}-1 (i.e. 0 to 65635). The *long* is used to represent larger integers. It occupies four bytes of memory and can hold numbers ranging from -2^{31} through 2^{31}-1 (i.e. – 2147483648 to 2147483647).

Data Type	Bytes	Other Names	Range of Values
Int	2	Signed	-32768 to 32767
unsigned int	2	Unsigned	0 to 65635
Short	2	short int	-32768 to 32767
unsigned short	2	Unsigned short int	0 to 65635
Long	4	Long int	– 2147483648 to 2147483647
unsigned long	4	Unsigned long int	0 to 4294967295

Table 9.1 Data types for integers

9.5.2 Data Types for Floating Point Numbers (float, double, long double)

Floating point numbers are the numbers that have a fractional part e.g. 12.35874, 0.54789, and -8.64, 101.003 etc. The floating point numbers are represented in computer in a format that is analogous to scientific notation (floating-point format). The storage area occupied by the number is divided into two sections: the *mantissa* and the *exponent*. Mantissa is the value of the number and the exponent is the power to which it is raised.

For example, in exponential notation the number 245634 would be represented as 2.45634 x 10^5, where 2.45634 is the mantissa and 5 is exponent. However in C, the representation of scientific notation is slightly different e.g. the above number will be represented as 2.45634e5. We don't express the exponent as the power of 10, rather the letter **e** or **E** is used to separate exponent from the mantissa. If the number is smaller than 1 (one), then exponent would be negative. For example, the number 0.00524 will be represented in computer as 5.24E-3.

ANSI C specifies three floating point types that differ in their memory requirements: *float*, *double*, and *long double*. These data types provide higher precision than the data types used for integers. The following table shows memory requirement, precision and the range of values that can be represented by data types *float, double and long double*.

Data Type	Memory Req. (in bytes)	Precision (in digits)	Range of Values
float	4	6	10^{-38} to 10^{38}
Double	8	15	10^{-308} to 10^{308}
long double	10	19	10^{-4932} to 10^{4932}

Table 9.2 data types for floating point numbers

9.5.3 Data Type for Characters – char

The data type *char* is used to represent a letter, number, or punctuation mark (plus a few other symbols). A char type variable occupies 1 *byte* in memory and can represent individual characters such as 'a', 'x', '5', and '#' etc. (the character '5' is manipulated quite differently than the integer **5** in the computer, so one should not consider both the same. we shall thoroughly discuss the topic in next chapters). In C, a character is expressed as enclosed in apostrophes such as 'a', 'e', 'i', 'o', and 'u' etc.

Like numbers, characters can also be compared, added and subtracted. Let's look at the following program to understand the concept:

```
#include <stdio.h>
void main(void)
{
        char   ch1, ch2, sum;
        ch1 = '2';
        ch2 = '6';
        sum = ch1 + ch2;
        printf("Sum = %d", sum);
}
```

Output:

```
Sum = 104
```

In fact, characters are stored in the form of ASCII (American Standard Code for Information Interchange) code. When we add, subtract or compare two characters, then instead of characters their ASCII codes are manipulated. In above example, because the ASCII codes of characters '2' and '6' are 50 and 54 respectively therefore the sum is 104. In ASCII, the printable characters have codes from 32 (code for a blank or space) to 126 (code for the symbol ~). The other codes represent nonprintable control characters. The complete list of ASCII charters with their codes is given at the end of this chapter.

The *signed* and *unsigned* keywords can be used with *char* like they can with *int*. Signed characters can represent numbers ranging from -128 though 127, while unsigned characters can represent numbers from 0 to 255. English alphabets, numbers and punctuation marks are always represented with positive numbers.

Working with floating point numbers

While working with floating point numbers, we may encounter some problems. For example, manipulation of very large and very small floating point numbers may show unexpected results. When we add a large number and a small number, the larger number may *cancel out* the smaller number; resulting in a *cancellation error* e.g. the result of addition of 1970.0 and 0.0000001243 may compute to 1970.000000 on some computers.

When two very small numbers are manipulated, the result may be too small to be represented accurately, so it will be represented as zero. This phenomenon is called *arithmetic underflow*. Similarly, manipulation of two very large numbers may result to a too large number to be represented. This phenomenon is call *arithmetic overflow*.

9.6 OPERATORS IN C

Operators are symbols, which are used to perform certain operations on data. C is equipped with a rich set of operators. These include arithmetic operators, relational operators, logical operators, bitwise operators, and many others. We shall discuss only the first three of these operators.

9.6.1 Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on values (numbers). The C language incorporates the following standard arithmetic operators:

Operation	**Symbol**	**Algebraic Expression**	**Expressions in C**
Addition	+	A + b	a + b
Subtraction	-	A – b	a – b
Multiplication	*	A × b	a * b
Division	/	A / b	a / b
Modulus	%	a mod b	a % b

The use of first four operators is straightforward. The last operator is *modulus* (also called *remainder* operator). Contrary to the *division* operator, which returns the quotient, it returns the remainder of an integral division. For example, if a, and b are two integers having values 8 and 3 respectively, then the express *a % b* will be evaluated to 2, which is the remainder of integral division.

9.6.2 Relational Operators

Relational operators are used to compare two values. These operators always evaluates to *true* or *false*. They produce a *non-zero* value (in most cases 1) if the relationship evaluates to *true* and a *0* if the relationship evaluates to *false*. There are the following six basic relational operators in C:

Suppose *a*, *b* and *c* are three integer variables having values 123, 215 and 123 respectively then:

Operation	Symbol	Expression	Evaluation
Equal to (comparison)	==	a == c	true (non-zero)
Less than	<	b < a	false (zero)
Greater than	>	a > c	false (zero)
Less than or Equal to	<=	a <= b	true (non-zero)
Greater than or Equal to	>=	b <= a	false (zero)
Not Equal to	!=	a != b	true (non-zero)

9.6.3 Logical Operators

Logical operators combine two or more relational expressions to construct compound expressions. The logical operators are && (logical *AND*), || (logical *OR*), and **!** (logical *NOT*).

The first logical operator && (logical AND) when combines two conditions, evaluates to *true* if both the conditions are true, otherwise it evaluates to *false*. The second logical operator || (logical OR) when combines two conditions, evaluates to *true* if any one of the conditions evaluates to true, otherwise evaluates to *false*. Similarly, the third logical operator ! (logical NOT) when applied to a condition, reverse the result of the evaluation of the condition, this means that if the condition evaluates to true, the logical NOT operator evaluates to **false** and vice versa. The three logical operators can take the following general forms:

```
exp1  &&  exp2
exp1  ||  exp2
     !(expression)
```

Example:

Suppose, if the salary of an employee in an organization is less than Rs.10,000 and he/she is married than he/she will be given an additional relief allowance. In a C program, let we have two variables; *salary* (an *int* type variable and *status*) a *char* type variable representing the marital status of the employee. To evaluate this condition, we can use the && operator:

```
salary < 1000 && status ==  'M'
```

Suppose, the organization changes his policy and decides to give relief allowance to all those employees whose salary is less than 5000 **or** who are married. To find out all those employees who fulfill the criteria, we can evaluate the following condition:

```
salary  < 5000 || status ==  'M'
```

In the meanwhile, the organization feels that their employees are facing problems because of low salary packages and also the organization has to spend a huge amount in the form of pension and retirement bonus. Keeping in view all these problems, the organization introduces revised pay scales in which it offers high salary packages to all those employees whose service period has not exceeded 10 years. In a C program, let we have a variable service_period (a float type variable). To find out all those employees whose service period has not exceeded 10 years we can evaluate the following condition:

```
!(service_period > 10)
```

9.6.4 Assignment Operator

In addition to basic C operators (arithmetic, logical, and relational) there are some other important operators, which one should know to write C programs. These includes assignment operator, increment and decrement operators.

The assignment operator is used to store a value or a computational result in a variable. In C, the symbol = represents the assignment operator e.g. in the following statement, values of the two variables, height and width, are multiplied and the result is assigned to the variable **Area.**

```
Area = height * width
```

The value to the right side of the operator is assigned to the variable on the left side of the assignment operator. This statement is also called assignment statement.

Assignment Statement
The assignment statement takes the general form:

variable = *expression*
The *expression* on the right side of the operator is evaluated first and the result is then assigned to the variable on the left side of the operator. The *expression* can be a variable, a constant or arithmetic, relational or logical expression.

Note: Writing variable to the right side and the expression to the left side of assignment operator will cause a syntax error.

9.6.5 Increment and Decrement Operators

The **increment operator** increases the value of its operand by one. It is denoted by the symbol ++ e.g. `count++`, where `count` is a variable. The effect of this expression is equivalent to the following expression:

```
    count = count + 1;
```

When ++ precedes its operand, it is called *prefix increment*. When the ++ follows its operand, it is called *postfix increment*. Consider the following statements:

```
    j = 10;
    i = ++j;
```

The first statement assigns the value of 10 to the variable *j*. In the second statement, first the value of *j* will be incremented by one (i.e. the value of j will be 11) and then the result will be assigned to the variable *i*. Hence, the variable *i* will be assigned the value of eleven (11). Therefore, after execution of the two statements, both the variables will have the same value i.e. eleven (11). Now, consider the following statements:

```
    j = 10;
    i = j++;
```

The execution of the first statement will take place as in above case. In the second statement, first the value of *j* (i.e. 10) will be assigned to the variable *i*, and then the value of *j* will be incremented by one. Hence, the variable *i* will be assigned the value of 10. Therefore, after execution of the two statements, the variable *j* will have the value of 11 and the variable **i** will have the value of 10.

C also provides a *decrement operator* denoted by the symbol `--` e.g. `count--`. The effect of this expression is equivalent to the following expression:

```
    count = count - 1;
```

It can be used as either the *prefix* operator or *postfix* operator in the same way as the increment operator is used. Consider the effect of the following set of statements:

`j = 10;`
`i = --j;`
	`j = 10;`
	`i = j--;`

<u>After Execution of statements:</u>	<u>After Execution of statements:</u>
• The value of *j* will be 9 and • The value of *i* will also be 9	• The value of *j* will be 9 and • The value of *i* will be 10

Note: Both *increment* and *decrement* operators are unary operators.

Compound Assignment Operators

The `++` and `--` operators respectively increment and decrement the value of their operand by one. There are four other compound assignment operators that can increment or decrement the value of their operand by other than one. These are `+=`, `-=`, `*=` and `/=` operators.

For example, the statement `j += 5` increases the value of *j* by 5 and the statement `j -=5` decreases the value of j by 5. Similarly, we can also use the operators `*=` and `/=` in the same way:

`j = 10;`
`j *= 5;`
	`j = 10;`
	`j /= 5;`

<u>After Execution of statements:</u>	<u>After Execution of statements:</u>
• The value of *j* will be 50	• The value of *j* will be 2

The operator `*=` multiplies the value of j by 5 and assign the result to it, whereas the operator `/=` divide the value of j by 5 and assign the result to it. The statement `j *= 5` is equivalent to `j = j * 5`, and the statement `j /= 5` is equivalent to `j = j / 5`.

9.6.6 Operator Precedence

An operator's precedence determines its order of evaluation in an expression. Table 9.3 lists the precedence of all C operators discussed so far, from highest to lowest.

Operator	**Precedence**
	Highest
! (logical NOT)	↓
`*, /, %`	
`+, -`	
`<, <=, >, >=`	
`==, !=`	
&&	
`||`	
= (assignment operator)	Lowest

Table 9.3 Operator Precedence

The table shows that the logical NOT operator has the highest priority. It is a unary operator – unary operators are those operators that have just one operand. Then comes arithmetic operators, relational operators, logical AND, logical OR and the assignment operators, which are all binary operators – *binary operators* are those operators that have two operands.

9.7 EXPRESSION

An *expression* is the combination of operators and operands. The operand may either be a constant or a variable e.g., a + b, 7 + m etc.

An expression, in which only arithmetic operators operate on operands, is known as *arithmetic expression*. To solve different mathematical problems, one needs to write arithmetic expressions. Arithmetic expressions involve integers and floating point numbers, which are manipulated with arithmetic operators.

9.7.1 Data Type of an Expression

The data type of expression (in fact, the data type of the result of expression) depends on the types of its operands. For example, consider the type of expression involving int or double type operands is of the form:

exp1 operator exp2

If both the operands are of type int, the result of the expression will be evaluated to an int type value. However, in case of mixed-type expression, one must be careful. A *mixed-type expression* is one in which operands are of different data types e.g. if the type of operand1 is int and the type of operand2 is double then the expression will always be evaluated to a double type value.

Arithmetic Operator	**Examples**
+	2 + 3 is 5 2.0 + 3.0 is 5.0
-	3 – 2 is 1 3.0 – 2.0 is 1.0
*	2 * 3 is 6 2.0 * 3.0 is 6.0
/	5 / 2 is 2 (integral division, because both the operands are integers) 5.0 / 2.0 is 2.5
%	5 / 2 is 2

Working with Division Operator

The manipulation of division operation is slightly different from other arithmetic operations in C. One should be careful while dividing numbers, as the result of division of two integers may not be an integer. C handles the division intelligently.

When the divisor and the dividend both are integers, the fractional part of the quotient (if exists) is truncated. For example the value of 7.0/2.0 is 3.5 but the value of 7/2 is the integral part of the result i.e. 3. Similarly, the value of 198.0/100.0 is 1.98, but the value of 198/100 is the integral part only i.e. 1. That's how C performs the division operation. So, to get the accurate result, at least one floating point number should be involved in division operation. Otherwise, integral division will take place and we will get the integral value.

Note: When a type double expression or value is assigned to a type int variable, the fractional part is truncated since it can not be represented in int type variable. For example, let **a** and **b** be two variables of type double and int respectively. Let,

a = 5 * 0.5;
b = 5 * 0.5;

a	**b**
2.5	2

9.8 Comments

Comments are used to increase the readability of the program. With comments, informative notes are inserted in the program's code, which helps in debugging and modifying the program. In C, there are two ways to comment the code; one can insert single line comments by typing two (forward) slashes at the start of the note such as:

```
//    This program calculates the factorial of the given number
```

These are called *single-line* comments. The other method is referred to as *multi-line* comments. Multi-line comments are used to add such informative notes which extend to multiple lines.

Multi-line comments can be inserted to the code (program code) by placing /* characters at the beginning of the comments (informative note), this is called opening of the multi-line comments. Each multi-line comment must end with letters */. Omitting ending letters (*/) will cause the whole program code beneath the opening letters (/*) for comments to be commented.

Note: It is worth mentioning that comments are completely ignored by the compiler while generating object code.

Example:

```
/*
 This program prints a single line message
  Author:     Student
  Date:       14-05-2005
*/
void main (void)
{
        // the following line of code prints a message

        printf(" This is my first program");
}
```

Exercise 9c

1. Fill in the blanks:
   - (i) The first character of a variable name must be a ____________.
   - (ii) _________operates on two operands.
   - (iii) Named memory cells which are used to store program's input and output are called_________.
   - (iv) The maximum length of a character constant is ______character.
   - (v) The value of a variable of type *int* ranges from________ to ________.
   - (vi) The value of an *unsigned int* variable ranges from 0 to __________.
   - (vii) The value of a float variable ranges from _______ to ________.
   - (viii) The symbol for logical OR operator is ___________.
   - (ix) _________are used to increase the readability of the program.
   - (x) Multi-line comments start with _______.

2. Choose the correct option:

   (i) Variables are created in:

   a) RAM b) ROM
   c) Hard Disk d) Cache

   (ii) Which of the following is a valid character constant?

   a) a b) "b"
   c) '6' d) =

   (iii) Which of the following data type offers the highest precision?

   a) float b) long int
   c) long double d) unsigned long int

   (iv) When the result of the computation of two very small numbers is too small to be represented, this phenomenon is called:

   a) Arithmetic overflows b) Arithmetic underflow
   c) Truncation d) Round off

   (v) The symbol '=', represents:

   a) Comparison operator b) Assignment operator
   c) Equal-to operator d) None of these

   (vi) Which of the following operators has lowest precedence?

   a) ! b) +
   c) = d) ==

   (vii) Relational operators are used to:

   a) Establish a relationship among variables
   b) Compare two values

c) Construct compound condition
d) Perform arithmetic operations

(viii) C is a strongly typed language, this means that:
e) Every program must be compiled before execution
f) Every variable must be declared before it is being used
g) The variable declaration also defines the variable
h) Sufficient data types are available to manipulate each type of data

(ix) The logical not operator, denoted by !, is a:
a) Ternary operator b) Uniary operator
c) Binary operator d) Bitwise operator

(x) a += b is equivalent to:
a) b += a b) a =+ b
c) a = a + b d) b = b + a

3. Write T for true and F for false statement.
(i) printf and scanf are standard identifiers.
(ii) In C language, all variables must be declared before being used.
(iii) Standard data types are not predefined in C language.
(iv) The double data type required 4 bytes in memory.
(v) In Scientific notation, the exponent represents the value of the number and mantissa represents the power to which it is raised.
(vi) The symbol for modulus operator is %.
(vii) The symbol = is used to compare two values.
(viii) Operator precedence determines the order of evaluation of operators in an expression.
(ix) For many compilers a C variable name can be up to 31 characters.
(x) C program can only use lowercase letters in variable names.

4. What is an identifier? Discuss the two types of identifiers in C.
5. What is a variable? Discuss the difference between declaring and defining a variable.
6. Write down rules for naming variables in C.
7. Differentiate the following:
(i) Constant and variable
(ii) Character constant and Numeric constant
(iii) Standard data type and User defined data type
(iv) Keyword and Identifier
8. What is a data type? Discuss various C data types to manipulate integers, floating point numbers and characters.
9. How many types of operators are available in C? Describe briefly. Also mention their precedence.

10. What data type would you use to represent the following items: number of children at your school, a letter grade on an exam, and the average marks of your class?
11. Which of the following are valid variable names in C?
    (i) income
    (ii) total marks
    (iii) double
    (iv) average-score
    (v) room#
    (vi) _area
    (vii) no_of_students
    (viii) long
    (ix) Item
    (x) MAX_SPEED
12. Write a note on the following:
    (i) Arithmetic Expression
    (ii) Comments in C
13. Let w, x, y, and z be the name of four type float variables, and let a, b and c be the names of three type int variables. Each of the following statements contains one or more violations of the rules for forming arithmetic expressions. Rewrite these statements so that it is consistent with these rules.
    (i) z = 4.0 w * y;
    (ii) y = yz;
    (iii) a = 6b4;
    (iv) c = 3(a + b);
    (v) z = 7w + xy;
14. Assume that you have the following variable declarations:

```
int  a, b, c, d, p;
float  v, w, x, y, z;
```

Evaluate each of the following statements assuming `a is 2, z is -1.3, c is 1, d is 3, y is 0.3E+1.`

(i) v = a * 2.5 / y;
(ii) w = a / y;
(iii) p = a / d;
(iv) x = (a + c) / (z + 0.3);
(v) b = d / a + d % a;
(vi) y = c / d * a;

Chapter 10

INPUT / OUTPUT

10.1 OVERVIEW

In previous chapters, we have studied the basics of C programs. This lesson covers basic input and output features of C language. Usually input and output form an important part of any program. To be more interactive, a program needs to be able to accept data and show results.

In C, the standard input/output library provides functions to perform input and output operations. By standard input and output, we mean the keyboard and monitor respectively. In C, these input/output operations are performed by two standard input/output functions, these are printf( ) and scanf( ). These functions can be accessed by including the standard input/output library (stdio.h) in the program. Let us have an overview of standard input/output functions.

10.1.1 printf Function

To see results of program execution, we must have a way to specify what variables values should be displayed. The standard library function *printf* (pronounced as print-eff) is used for formatted output. It takes as arguments a format string and an optional list of variables to output. The values of variables are displayed according to the specifications in the format string.

The printf ( ) function will take the form:

printf(*format string*, *var1*, *var2*, *var3*, ...);

printf(*format string);*

The *format string* is a character string – nothing more and the variables are optional. The easiest way to understand this is by example.

The **signature** of a function describes the number and type of its arguments, and the return type of the function.

Example 1 **Write a program to calculate and print the area of a square.**

```
#include <stdio.h>
void  main( )
{
   int  height, width, area;
   height = 5;
   width = 4;
   area = height * width;
   printf("Area of Square = %d", area);
}
```

Here's the **output** of the program:

```
Area of Square = 20
```

In the above program, the first line is the variable declaration statement. In second and third lines, values are assigned to the variables *height* and *width*. Fourth line of code describes the arithmetic expression for calculating the area of the square and the result is assigned to the variable *area*. Fifth and the last line of code is the printf( ) statement, which displays result on the screen. In case of *printf*, the first parameter is always a string (e.g.,"Area of Square"), which should be enclosed in double quotes. This string is called the *format string*. Format string may include any number of format specifiers such as %d. The list of variables separated by commas, whose values are to be displayed in the result, will follow the format string.

10.1.2 Format Specifier

Format specifiers specify the format in which the value of a variable should be displayed on the screen. Format specifiers are specified in the format string.

For instance, in the above program the printf( ) statement contains the symbol %d, which is format specifier for the variable *area*. For different types of variables, different format specifiers are used. Here is the list of format specifiers:

Symbol	Data Type
%d	int, short
%f	float
%lf	double
%e	float, double (Exponential Notation)
%g	Floating point (%f or %e, whichever is shorter)
%c	char
%s	Character string
%u	unsigned short, unsigned int
%x	Unsigned hexadecimal integers
%o	Unsigned octal integer
%i	Integers
%ld	long integer

Example 2 **Write a program that adds two floating point numbers and shows their sum on the screen.**

```
#include <stdio.h>

void main(void)
{
        float var1, var2, res;

        var1 = 24.27;
        var2 = 41.50;
        res = var1 + var2;
        printf("%f + %f = %f", var1, var2, res);
}
```

Here's the *output* of the program

```
24.27 + 41.5 = 65.77
```

10.1.3 Field-width Specifier

In a C program, the number of columns used to display a value on the screen is referred to as *field-width*. Field-width specifiers describe the number of columns that should be used to print a value.

Formatting Integers

We simply need to add a number between the % and d of the %d format specifier in the printf format string. This number specifies the field-width or the number of columns to be used for the display of the value. The statement

printf("Area = %4d", area);

indicates that four columns will be used to display the value of *area*. Suppose the value of the variable *area* is 25. Two extra spaces will be padded before 25 on the screen to complete the length of 4. The output of the above statement will be as follows:

Area = □□25

Here, □ represents a blank space. This space will not be displayed as a printed character in actual output. In this way the value of 25, which requires two spaces to be displayed, will occupy four spaces (columns) on the screen. The reason is that the format specifier for area (%4d) allows spaces for four digits to be printed. Because the value of area is 25, therefore its two digits are *right justified*, preceded by two blank spaces.

The following table shows how values are displayed using different format specifiers.

Value	Format Specifier	Displayed Output	Value	Format Specifier	Displayed Output
786	%4d	□786	-786	%4d	-786
786	%5d	□□786	-786	%5d	□-786
786	%6d	□□□786	-786	%6d	□□-786
786	%1d	786	-786	%2d	-786

The last row of this table shows that C expands the field width if it is too small for the integer value displayed.

Formatting Floating Point Numbers

For format specification of floating point numbers, we must indicate both the total *field width* needed and the number of *decimal places* desired. The total field width should be large enough to accommodate all digits before and after the decimal point e.g., to display 15.245 and 0.12 the total field width should be six and four respectively. It should be noted that for numbers smaller than zero, a zero is always printed before the decimal point. Therefore the total field width should include a space for the decimal point as well as for the minus sign if the number can be negative.

The general form for the format specifier for a floating point value will be *%m.n*f, where *m* represents the total field width, and n represents the desired number of decimal place. For instance, the statement

printf("Height = 6.2f", height);

indicates that the total field width for the value of the variable *height* is 6, and the accuracy is of two decimal places. The value of *height* will be rounded off to two decimal places and will be displayed *right justified* in 6 columns. While being rounded off, if the third digit of the value's fractional part is 5 or greater, the second digit is increased by one otherwise the third digit is discarded.

Remember: A format specifier always begins with the symbol %

The following table shows how values are displayed using different format specifiers.

Value	Format Specifier	Displayed Output	Value	Format Specifier	Displayed Output
-25.41	%6.2f	-25.41	.123	%6.2f	□□0.12
3.14159	%5.2f	□3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	□□3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	□3.14159
.6789	%4.2f	0.69	-0.007	%4.2f	-0.01
-.007	%8.3f	□□-0.007	-0.007	%8.5f	-0.00700
-.007	%.3f	-0.007	-3.14159	%.4f	-3.1416

10.1.3 Escape Sequences

Escape sequences are characters which are specified in the format string of the *printf* statement in combination with a backslash (\). These cause an escape from the normal interpretation of a string so that the next character is recognized as having a special meaning. For example, consider the following program

Remember: Escape sequence characters always begins with a backslash (\)

Example 3 **Write a program that will demonstrate the use of escape sequences.**

```
#include <stdio.h>

void main(void)
{
     printf("Name\t\tRoll_No\t\tMarks");
     printf("\n------------------------------------ ");
     printf("\nAmir\t\t  78\t\t425");
     printf("\nTahir\t\t  23\t\t385");
}
```

Here's the output of the program

```
Name            Roll No         Marks
-----------------------------------------------
Amir            78              425
Tahir           23              385
```

In the above program, we use two escape sequences. These are **\t** and **\n**. The escape sequence **\n** causes the text to print from the start of the next line, whereas **\t** inserts a tab space between two words. In addition to newline and tab escape sequences; there are some others as well. Here is a list of them:

Escape Sequence	**Purpose**
\n	New Line
\t	Tab
\b	Backspace
\r	Carriage Return (Enter Key)
\f	Form feed
\'	Single Quote
\"	Double Quote
\\	Backslash
\xdd	ASCII code in hexadecimal notation (each d represents a digit)
\ddd	ASCII code in octal notation (each d represents a digit)

We have discussed the purpose of first two escape sequences. The escape sequence \b causes the cursor to move one space left, the form-feed (\f) moves to the next page on printer. It is important to note that one can not display a single or double quote on the screen without using the escape sequences \' and \". The reason is that the format string of the *printf* function is enclosed in a double quote. When a double quote is specified in the format string, it is treated as the closing double quote. That's why, single and double quotes are always written with backslash. For example the statement

```
printf("Escape Sequence is a \"Cool\" feature of C.");
```

Here's the *output*

```
Escape Sequence is a "Cool" feature of C.
```

10.2 SCANF FUNCTION

Most of the programs are interactive in nature. Till now, we did not learn a way to write interactive programs. C is featured with a range of functions to accept user input in variety of forms. The *scanf* (pronounced as scan-eff) function is versatile as it is equally good for numeric as well as string input.

It takes as arguments a format string and a list of variables to hold the input values. Here is the syntax of *scanf* function:

scanf(*format string*, &*var1*, &*var2*, &*var3*, ...);

Let us consider the following program to understand the working of scanf function:

Example 4 **Write a program to convert the distance in kilometers into meters.**

```
#include <stdio.h>

void main(void)
{
      double meter, kilometer;
      // prompt the user to enter kilometers
      printf("Enter distance in kilometers >");
      // take input
      scanf("%lf", &kilometer);
      meter = kilometer * 1000;
      printf("\n%lf kilometers = %lf meters",
      kilometer, meter);
}
```

Here's the sample *output* of the program

```
Enter distance in kilometers >40.5
40.500000 kilometers = 40500.000000 meters
```

In the above program, first line is the declaration of variables *meter* and *kilometer*. The next executable statement is *printf*, which displays a message for the user to enter distance in kilometers. The next is the *scanf* statement. When the program reaches this line of code, the flow of execution stops until the user enters a value. The format string of *scanf* is "%lf" which tells the scanf what kind of data to copy into variable *kilometer*. The format string of scanf consists of a list of format specifiers only; no other value or text can be specified in it.

Instead of the variable name, the *scanf* requires address of the variable to store the input value into it.

Notice that in the call to scanf, the name of variable *kilometer* is preceded by an ampersand character (&). In C, & is actually the *address of* operator. In scanf, the *address of* operator (&) tells the scanf funciton the address of the variable where the input value is to store. If & is omitted, the scanf will not be able to locate the variable in memory, hence it will be unable to store the value into the variable and the program will find a garbage value in the variable.

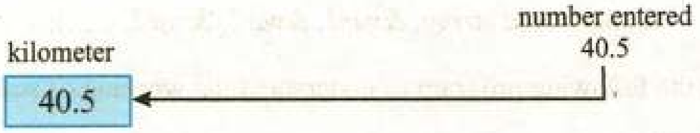


Fig. 10.1 Input value is stored in variable

10.3 CHARACTER INPUT

In C, there are many functions to accept character input. The versatile *scanf* can also be used for this purpose. But *scanf* requires pressing the return key at the end of input value. In some cases, it is desirable to input characters without pressing the return key. For example, in a game while controlling the movement of a space ship through arrow keys we can't afford to press return key each time after pressing an arrow key. To overcome such situations, C is equipped with many other functions specialized for character input. *getch* and *getche* are examples of such functions. These are part of the conio (console input output) library.

10.3.1 getch and getche Functions

The *getch* and *getche* functions are very handy in character manipulation. In contrast to the *getch* function which does not echo the character typed, the *getche* function echo the typed character. Both of these functions do not accept any argument. However they return the typed character to the calling function. One does not need to press the return key (ENTER key) after typing the character. The moment a character is typed, it is imidiately returned by the function to the calling module.

Note it that the character constants are always specified within single quotations e.g. 'a', '@', '7', etc.

Example 5 **Write a program that displays the ASCII code of the character typed by the user.**

```
#include <stdio.h>
#include <conio.h>
void main( )
{
     char ch;

     printf("Please type a character :");
     ch = getche( );
     printf("\nThe ASCII code for \'%c\' is %d", ch, ch);
}
```

Here's the **output** of the program

```
Please type a character :a
The ASCII code for 'a' is 97
```

Note it that, in this program we include a new header file *conio.h*. This file contains the definition of functions getch() and getche(). In this program, the statement

ch = getche();

can be replaced with the statement

ch = getch();

In the later case, the typed character will not be shown on the screen and the output will be as follows:

```
Please type a character :
The ASCII code for 'a' is 97
```

When the 3rd line of the program is executed, it waits for a character to be typed. As soon as a character is typed, the very next line executes imidiately without waiting for the return key to be typed. And the function getche() returns the typed character to the main function, where it is assigned to the variable **ch**.

Exercise 10c

1. Fill in the blanks:
   - (i) The_______________ function does not display characters on the output screen.
   - (ii) _____________ is an input function.
   - (iii) %x is a format specifier for __________________.
   - (iv) Escape sequences always begins with a _______________.
   - (v) The printf function is defined in ____________________.
   - (vi) The ASCII code for *Escape* key is ____________________.
   - (vii) The escape sequence ________ represents the carriage return.
   - (viii) There are total __________ columns on the output screen.
   - (ix) The symbol for address of operator is _____________.
   - (x) \dddd is used to print ASCII code in _____________ notation.

2. Choose the correct option:
   - (i) The function getche() is defined in:
     - a) stdio.h b) string.h
     - c) math.h d) conio.h
   - (ii) The escape sequence for backslash is:
     - a) \ b) \b
     - c) \\ d) \t
   - (iii) The format specifier %u is used for:
     - a) integer b) unsigned short
     - c) unsigned float d) unsigned long int
   - (iv)
     - a) Arithmetic overflows b) Arithmetic underflow
     - c) Truncation d) Round off
   - (v) The symbol '=', represents:
     - a) Comparison operator b) Assignment operator
     - c) Equal-to operator d) None of these
   - (vi) Which of the following operators has lowest precedence?
     - a) ! b) +
     - c) = d) ==
   - (vii) Relational operators are used to:
     - a) Establish a relationship among variables
     - b) Compare two values
     - c) Construct compound condition
     - d) Perform arithmetic operations
   - (viii) C is a strongly typed language, this means that:
     - a) Every program must be compiled before execution
     - b) Every variable must be declared before it is being used
     - c) The variable declaration also defines the variable

d) Sufficient data types are available to manipulate each type of data

(ix) The logical not operator, denoted by **!**, is a:

a) Ternary operator b) Uniary operator
c) Binary operator d) Bitwise operator

(x) a += b is equivalent to:

a) b += a b) a =+ b
c) a = a + b d) b = b + a

3. Write T for true and F for false statement:
   (i) printf and scanf are standard identifiers.
   (ii) In C language you must declare all variables before using them.
   (iii) Standard data types are not predefined in C language.
   (iv) The double data type required 4 bytes in memory.

4. What do we mean by standard input and output? Illustrate the use of printf() and scanf() functions.

5. Illustrate the difference between format specifiers and field-width specifiers with examples.

6. Define the term 'escape sequence'. List names and uses of any five escape sequences.

7.

a) Show the output displayed by the following program when the data entered are 10 and 15.

```
#include <stdio.h>
void main()
{
        int m, n;
        printf("Enter two numbers(separated by
        comma):");
        scanf("%d %d", m, n);
        m = m + 10;
        n = 5 * m;
        printf("m = %d\t\t\t n = %d\n", m, n);
}
```

b) Show the contents of memory (for variables `'a'` and `'b'`) before and after the execution of the above program.

c) Write the program in example 5 using *scanf* function.

8.

a) Show how the value -17.246 would be printed using the formats %8.4f, %8.3f, %8.2f, %8.1f, %8.0f, and %0.2f.

b) Assuming x (type double) is 21.335 and y (type int) is 200, show the output of the following statements (on paper). For clarity, use the symbol □ to denote a blank space.

```
printf("x is %6.2f \t y is %4d\n", x, y);
printf("y is %d\n", y);
printf("x is %.1f\n", x);
```

c) If the variables a, b, and c are 307, 408.558 and -12.31, respectively, write a statement tha t will display the following line: (for clarity, the symbol □ shows a blank space)

□□307□□□□□408.56□□□□-12.3

9. Write a program that asks the user to enter the radius of a circle and then computes and displays the circle's area. Use the formula

area = PI × radius × radius

where PI is the constant value of 3.14159. (Note: Define a constant macro PI with #define directive)

10. Write a program that stores the values 'A', 'U', 3.456E10 and 50 in separate memory cells. Your program should get the first three values as input data, but use an assignment statement to store the last value.

11. Write a program that converts a temperature in degrees Fahrenheit to degrees Celsius. For conversion, use the following formula

celsius = 5/9 (fahrenheit – 32)

12. Write a program that takes a positive number with a fractional part and rounds it to two decimal places. For example, 25.4851 would round to 25.49, and 62.4431 would round to 32.44.

DECISION CONSTRUCTS

Chapter 11

11.1 OVERVIEW

While writing C programs, the programmer may need to choose a path to execute one or more statements through the program based on a certain criterion; Decision constructs provide a way to meet this requirement. This chapter introduces various selection structures available in C, which can be used in such situations.

11.1.1 Control Structures

Control structures are statements used to control the flow of execution in a program or function. The C control structures enable us to group individual instructions into a single logical unit with one entry point and one exit point.

Program instructions can be organized into three kinds of control structures to control execution flow i.e. *sequence, selection and repetition*. All programs, whether simple or complex, use these control structures to implement the program logic.

Until now we have been using only sequential flow, which is also called default flow. In case of *sequence* structure, instructions are executed in the same order in which they are specified in the program. A *compound statement* refers to a group of statements enclosed in opening and closing braces such as:

```
{
        statement1 ;
        statement2 ;
            .
            .
            .
        statementn ;
}
```

Control flows from *statement*$_1$ to *statement*$_n$ in a logical sequence. Here we shall discuss the *selection structure* in detail, and in the next chapter the *repetition structure* will be discussed. Solutions of some problems require steps with two or more alternative course of action. A *selection structure* chooses which statement or a block of statements is to execute. In C, there are two basic selection statements:

- *if* – else
- *switch*

There are some variations of *if – else* structure which will be discussed here. The third control structure is *repetition, which* is also called iteration or loop. It is a control structure, which repeats a statement or a group of statements in a program. C provides different types of loop structures to be used in various situations. These include *for* loop, *while* loop, and *do-while* loop.

11.2 IF STATEMENT

if is one of the keywords in C language. It is used to select a path flow in a program based on a condition. A *condition* is an expression that either evaluates to true (usually represented by 1) or false (represented by 0). The result of this evaluation can be assigned to a variable. For example, consider the following program:

```
#include <stdio.h>
void main ( )
{
        int age, status;
        printf("Enter the age :");
        scanf("%d", &age);
        status = (age > 60);
        printf("Status = %d", status);
}
```

The value of the variable status will be *1* if the age is greater 60, otherwise status will be *0*. Here's the *output* of the program:

Case 1 (When age is less than 60)	**Case 2** (When age is greater than 60)
Enter the age :45	Enter the age :78
Status = 0	Status = 1

11.2.1 Simple *if* Statement

if statement is the simplest form of decision constructs. It allows a statement or a set of statements to be executed conditionally. The general form of simple *if* statement is:

```
if (condition)
{
        statement1 ;
        statement2 ;
                .
                .
                .
        statementn ;
}
```

The statement(s) in the block of *if statement* are executed if the condition is *true*; otherwise these are skipped. If there are more statements in *if* block then these should be enclosed in braces as a compound statement. However, in case of a single statement the braces are optional.

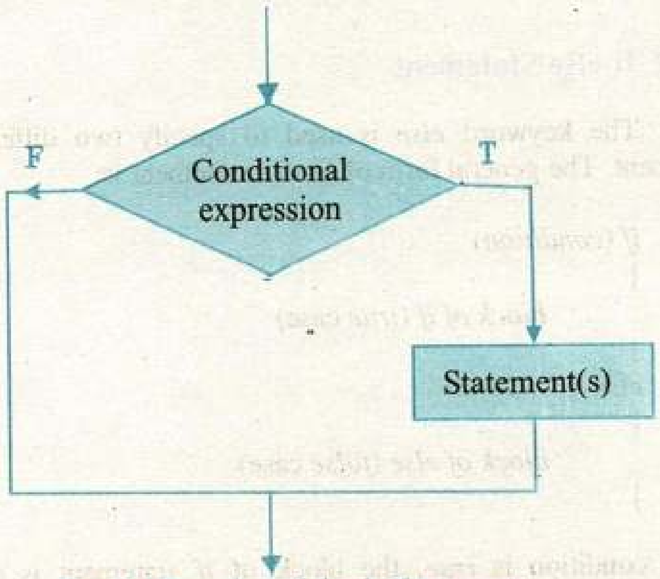


Fig. 11.1 Flow chart of simple *if* statement

This flow chart describes a situation where a simple *if* statement can be used.

Example 1 **A program that calculates the square root of a given number**

```
#include <stdio.h>
#include <math.h>

void main(void)
{
      double x = 0.0, square_root = 0.0;
      printf("Enter a number >");
      scanf("%lf", &x);

      if (x > 0)
            square_root = sqrt(x);
}
```

As the square root of negative numbers is imaginary, therefore the program finds the square root of positive numbers only. In this program we have used *math* library. This includes the definition of sqrt and many other mathematical functions. We shall discuss it later in this book.

A **flow chart** is the pictorial representation of a program.

11.2.2 If-else Statement

The keyword *else* is used to specify two different choices with *if* statement. The general form of *if-else* statement is:

```
if (condition)
{
        block of if (true case)
}
else
{
        block of else (false case)
}
```

If the condition is *true*, the block of *if* statement is executed and if the condition is *false*, the block of *else* statement is executed. The following flow chart may help to explain the idea.

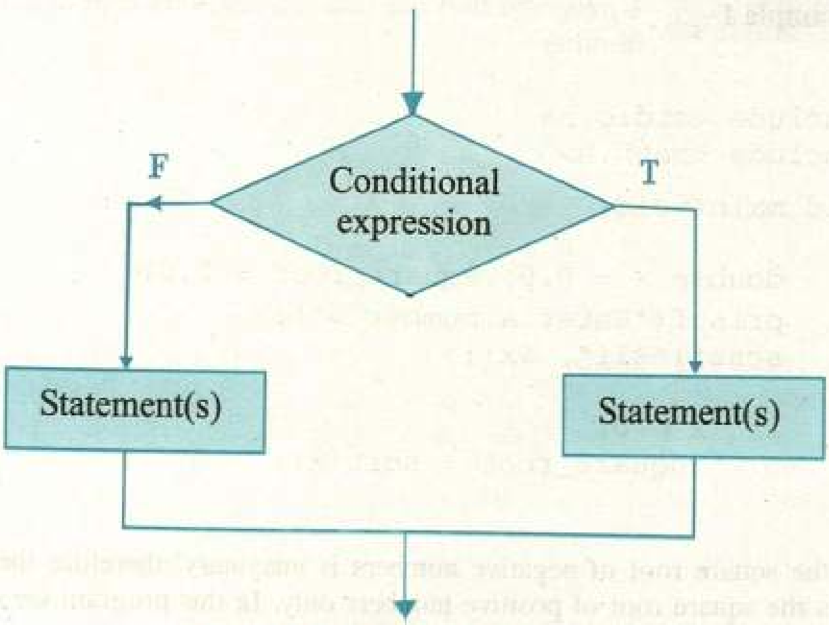


Fig. 11.2 Flow chart of *if-else* statement

The program in example 1 can be written using *if-else* statement as follows:

```
#include <stdio.h>
#include <math.h>
        void main(void)
        {
                double x = 0.0, square_root = 0.0;

                printf("Enter a number >");
                scanf("%lf", &x);

                if (x >= 0)
                {
                        square_root = sqrt(x);
                        printf("Square root of %lf = %lf", x,
                        square_root);
                }
                else
                        printf("Square root can not be
                        calculated");
        }
```

There are two blocks of statements in this program, either of which is conditionally executed. If the condition evaluates to *true* the square root of **x** will be calculated and the output will be shown on the screen and in case the condition evaluates to *false*, the message "Square root can't be calculated" will be displayed.

It is very important to note that the block of *if* is enclosed in braces, and no bracket has been used in the body of *else* statement. The reason is that we want to execute multiple statements in case of *true* condition, so these must be represented as a compound statement. If we omit the braces from the body of *if* statement, the following error message will appear:

Illegal `else` without matching `if`

So, to avoid this message, one should always enclose a compound statement in braces. If there is a compound statement in the body of *else*, omitting braces will not cause the above-mentioned error, rather only one statement after the else will be treated as the body of *else* and the rest of the statements will always be executed sequentially.

11.2.3 Nested if Statement

Nested if statement means an *if* statement inside another *if* statement. Nesting can be done up to any level. The programmer may use as many *if* statements inside another *if* statement as (s)he wants. However, the increase in the level of nesting also increases the complexity of the *nested if* statement. The general form of *nested if* statement is as follows:

if (*condition*)

```
{
        if (condition)
        {
                block of inner if
        }
}
```

block of outer **if**

The *else* statement is optional, it may or may not be used with *outer* or *inner* **if** statement. However, if used, its block can also contain other if statements.

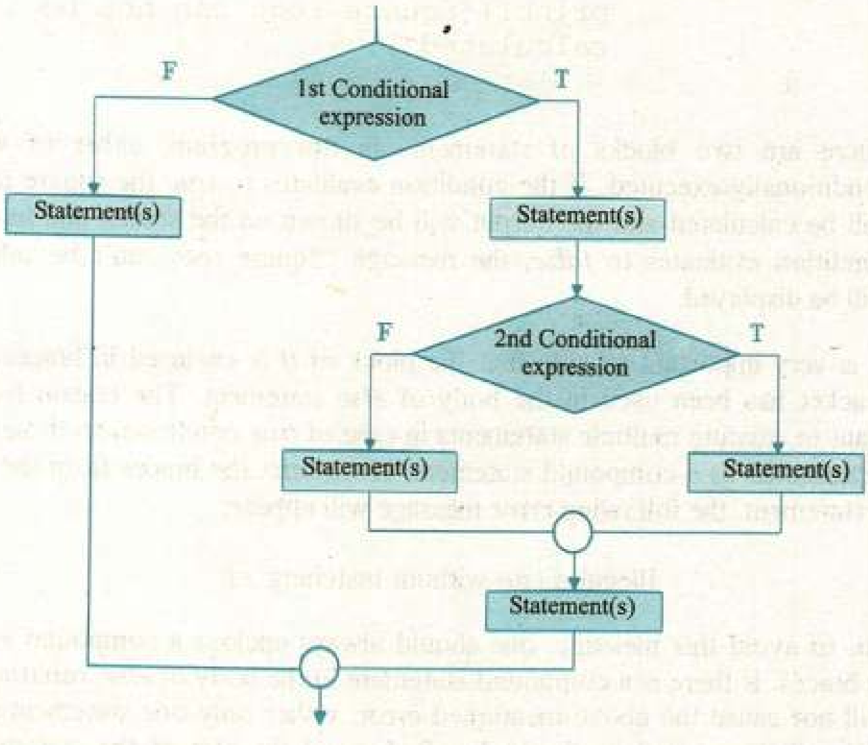


Fig. 11.3 Flow chart of *nested if* statement

Example 2 **A program that accepts three numbers from the user and displays the largest number.**

```
#include <stdio.h>

void main(void)
{
    int a, b, c;

    printf("Enter three numbers >");
    scanf("%d %d %d", &a, &b, &c);

    if (a > b)
    {
          if(a > c)
               printf("%d is largest", a);
          else
               printf("%d is largest ", c);

    }
    else
    {
     if (b > c)
          printf("%d is largest", b);
     else
          printf("%d is largest", c);
    }
}
```

This is a simple program that compares three numbers to find the largest one. The body of *if* and the body of *else*, both contains other if statements (nested *if*s are boxed). This sort of arrangement of multiple *if* statements is called *nested if*.

The execution of the nested *if* proceeds as follows: The first condition (a > b) is tested, if it is *true* then the second condition (a > c) is tested, if it is also *true*, the rest of the conditions will be skipped and we conclude that *a* is the largest number. If the second condition is *false*, this means *a* is greater than *b* but smaller than **c**. Therefore **c** is the largest number.

If the first condition (a > b) is *false*, the body of *if* is skipped and the flow of control is transferred to the body of *else* where the condition (b > c) is tested. If it is *true* then the second condition (b > c) is tested, if it is also *true*, this means *b* is greater than *a* (from the first condition) and *b* is greater than *c* (from the second condition), therefore we conclude that *b* is the largest number. If the second condition in the body of *else* is *false*, it means *b* is greater than *a* (from the first condition), but *b* is smaller than *c*, therefore we conclude that **c** is the largest number.

In this way, we can implement decision making in the program using nested *if* statement.

11.2.4 Comparison of Nested if and Sequence of ifs

It is due to the complexity of *nested if* statement that beginners usually prefer to use a sequence of *if* statements rather than a single nested *if* statement. However, sometimes it is useful to use a *nested if* instead of sequence of ifs such as in example 2. In case of *nested if* statement, when the control flow reaches a logical decision, the rest of the conditions are skipped. Whereas in a sequence of *if* statements, all conditions are tested in any case. This can be understood by the following example.

Example 3 **A program that inputs a number from the user and determines whether it is positive, negative or zero.**

```
#include <stdio.h>

void main(void)
{
        int num;

        printf("Enter a number >");
        scanf("%d", &num);

        if (num > 0)
                printf("The number is positive");
        if (num < 0)
                printf("The number is negative");
        if (num == 0)
                printf("The number is zero");
```

This program implements the solution using a sequence of *if* statements. Suppose, the user enters a positive number, the answer is decided in the first comparison i.e. the number is positive. So there is no need to check the number for its being negative or zero as it is impossible. But, this solution suggests doing the last two comparisons unnecessarily, wasting the CPU time. This situation may be avoided by using some other form of *if* statement such as *if-else* statement.

11.2.5 if-else if Statement

Nested if statements can become quite complex, if there are more than three alternatives and indentation is not consistent, it may be difficult to determine the logical structure of the *if* statement. In such situations, *if* statement with multiple alternatives (if-else if) can be a good option. The general form of *if-else if* statement is as follows:

```
if (condition₁)
        statement₁;
else if (condition₂)
        statement₂;
            .
            .
            .
else if (conditionₙ)
        statementₙ;
else
        statementₖ;
```

The test conditions in *if statement with multiple alternatives* are executed in sequence until a *true* condition is reached. If a condition is true, the statement(s) following it is executed, and the rest of the alternatives are skipped. If a condition is false, the statement following it is skipped and the next condition is tested. If all conditions are falls, then statement$_k$ following the last *else* is executed. The following flowchart shows the execution flow of the program through *if-else if* statement.

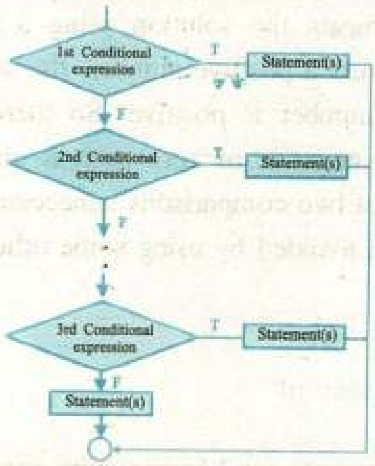

1st Conditional expression
T
Statement(s)
F
2nd Conditional expression
T
Statement(s)
F
3rd Conditional expression
T
Statement(s)
F
Statement(s)

Fig. 11.4 Flowchart of *if-else if* statement

The program in *example 3* can be re-written using *if-else if* structure as follows:

```
#include <stdio.h>
void main(void)
{
     int num;
     printf("Enter a number >");
     scanf("%d", &num);
     if (num > 0)
          printf("The number is positive");
     else if (num < 0)
          printf("The number is negative");
     else
          printf("The number is zero");
}
```

It can be seen that the *else-if* construct has greatly simplified the program, while preserving the efficiency as well. If a positive number is entered, we will reach the answer in the first comparison and rest of the conditions will be skipped.

11.3 USE OF LOGICAL OPERATORS

We have studied the three logical operators (AND, OR, and NOT represented by *&&*, ||, and **!** respectively) in chapter 2. These operators play an important role in constructing certain compound conditions to be used with *if* statement. Until now, we have been using simple conditions with *if* statement and its variations. In this section, we shall observe how complex program logic can be simplified using logical operators.

To grasp the concept, we re-write the program in *example* 2 using logical AND operator. This will demonstrate how much the task is simplified.

```
#include <stdio.h>
void main (void)
{
          int a, b, c;
          printf("Enter three numbers >");
          scanf("%d %d %d", &a, &b, &c);
          if ( a > b && a > c)
                    printf("%d is largest", a);
          else if( b > a && b > c)
                    printf("%d is largest", b);
          else
                    printf("%d is largest", c);
}
```

There may be situations where the use of logical operators may simplify the program logic. We just need to concentrate the underlying problem. We shall see more examples of using logical operators in next chapters.

11.4 SWITCH STATEMENT

The *switch* statement can also be used in C to select one of many alternatives. Like *if* statement, it is also a conditional statement that compares the value of an expression against a list of cases. The case label can be an integral or character constant. Similarly, the value of the expression must be an integer or a character; it must not be a double or float value. Here's the syntax of switch statement.

```
switch(expression)
{
          case val1:
                    statements1;
                    break;
          case val2:
                    statements2;
                    break;
                    .
                    .
                    .
          case valn:
                    statementsn;
                    break;
```

```
        default:
                statements_k;
    }
```

The value of expression is compared to each *case* label. The statements following the first label with a matching evaluated value are executed until a *break* statement is encountered. The break causes the rest of the switch statement to be skipped. If all *break* statements are omitted from the switch statement, the code from the first true *case* down to the end of the switch statement will execute sequentially. A *default* label may be used to provide code to be executed if none of the *case* label is matched. However the position of *default* label is not fixed. It may be placed before the first *case* statement or after the last *case*. The *switch* statement is especially useful when the selection is based on the value.

Example 4 **Write a program that inputs a character from the user and checks whether it is a vowel or a consonant.**

```
#include <stdio.h>
void main(void)
{
     char ch;
     printf("Enter an alphabet");
     ch = getche();
     switch(ch)
     {
          case 'a':
          case 'A':
               printf("It's a vowel");
               break;
          case 'e':
          case 'E':
               printf("It's a vowel");
               break;
          case 'i':
          case 'I':
               printf("It's a vowel");
               break;
          case 'o':
          case 'O':
```

```
                    printf("It's a vowel");
                    break;
                case 'u':
                case 'U':
                    printf("It's a vowel");
                    break;
                default:
                    printf("It is not a vowel");
                    break;
            }
    }
```

In this program, we have used two *case* labels, one after the other, without having any other statement between them. This sort of arrangement of *case* labels works equivalent to the logical OR operator i.e., whether the value of the variable *ch* is 'a' or 'A', the code following the labels *case* '*a* and *case* 'A' will execute.

- The value of *expression* in *switch* statement must be of type *int* or *char*, but not of type *float* or *double*.
- If the value of *expression* in *switch* statement is of type *float* or *double*, the compiler will generate the following error message:

  Switch selection expression must be of integral type

11.5 CONDITIONAL OPERATOR

Conditional operator is used as an alternative to the if-else statement. It is a ternary operator (requires three operand). Its general form is:

conditional expression? *true-case statement* **:** *false-case statement*;

The *expression* may be a relational or logical expression. If the expression is true then the true-case statement will be executed otherwise the false-case statement is executed. e.g.,

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a, b;
    printf("Please enter two numbers: ");
    scanf("%d %d", &a, &b);
    a > b ? printf("%d is larger", a) : printf("%d is
    larger", b);
}
```

11.6 CASE STUDY: Locating a point in the coordinate plane

PROBLEM Write a program that input x- and y-coordinates of a point in the coordinate plane. Based on these values, it then determines where it lies, on x- or y-axis, or in any of the four quadrants.

SOLUTION The first step towards the solution of any problem (simple or complex) is to understand it. Think on the problem from all aspects; identify input and output requirements and different types of restrictions on the data. After analyzing the problem, try to develop a simple solution. Don't indulge yourself in unnecessary details. You can trace the solution on the paper in the form of an *algorithm*. ***An algorithm is a step-by-step procedure to solve any problem in finite number of steps.*** Let's work out the problem described above:

It is clear from the figure 11.1 that each point will lay on either of the two axes or in any of the four quadrants.

Input Values: The program will input the values of x-coordinate and y-coordinate. These values will be of type *int*.

Output: The program will output a message describing the position of the point in the coordinate plane.

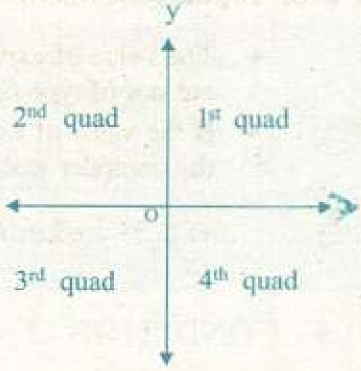


Fig. 11.1 -Co-ordinate plane

What do we know?

- If both x- and y-coordinates are zero, the point will be at the origin.
- If x-coordinate is zero and y-coordinate is non-zero, then the point will be on y-axis.
- If y-coordinate is zero and x-coordinate is non-zero, then the point will be on x-axis.
- If both x- and y-coordinates are positive (> 0), the point will lie in the 1st quadrant.
- If x-coordinate is negative (< 0) and y-coordinate is positive (> 0), the point will lie in the 2nd quadrant.
- If both x- and y-coordinates are negative (< 0), the point will lie in the 3rd quadrant.
- If x-coordinate is positive (> 0) and y-coordinate is negative (< 0), the point will lie in the 4th quadrant.

Now, keeping in view all this information, let's try to write a C program to solve this problem:

Program:

```
#include <stdio.h>
void main(void)
{
        int x, y;
        printf("Enter value for x- and y-coordinates
>");
        scanf("%d %d", &x, &y);
        if(x  == 0)
        {
                if(y  == 0)
                        printf("The   point   lie   on   the
origin");
                else
                        printf("The point lie on y-axis");
        }
        else if(x > 0)
        {
                if(y  == 0)
                        printf("The point lie on x-axis");
                else if(y > 0)
                        printf("The   point   lies   in   1st
                        quadrant");
                else
                        printf("The   point   lies   in   4th
                        quadrant");
        }
        else
        {
                if(y  == 0)
                        printf("The point lies on x-axis");
                else if(y > 0)
                        printf("The   point   lies   in   2nd
                        quadrant");
                else
                        printf("The   point   lies   in   3rd
                        quadrant");
        }
}
```

Exercise 11c

1. Fill in the blanks:

   (i) ________________ control the flow of execution in a program.
   (ii) ______________ is a group statements enclosed in braces.
   (iii) *if* statement is used to select a pathflow in a program based on a ________________.
   (iv) A ____________ is a pictorial representation of a program.
   (v) ________________ refers to a structure with one if statement inside another if statement.
   (vi) _____________ statement can be used as an alternative to if-else if statement.
   (vii) In switch statement the value of expression must be ________ or ___________.
   (viii) The _____________statement switches the control outside the block in which it is used.
   (ix) The purpose of ___________ label in switch-case statement is the same as that of *else* in *if-else* statement.
   (x) A condition is an expression that is either __________ or ___________.

2. Write T for true and F for false statement.

   (i) An arithmetic expression can not be used as condition in **if** statement.
   (ii) The conditional operator is a ternary operator.
   (iii) Logical operators are used to compare values.
   (iv) A switch statement can not be used within the block of an *if* statement.
   (v) The break statement stops the execution of a program for a moment and then resumes.
   (vi) In sequence structure, statements are executed in the same order in which they appear in the program.
   (vii) A false condition always evaluates to zero.
   (viii) Use of the statement terminator at the end of an *if* statement causes a syntax error.
   (ix) The switch expression can not be of float or double type.
   (x) C is an unstructured programming language.

3. What is a *control structure*? Briefly describe the basic control structures for writing programs.

4. How many selection statements are available in C? Discuss the difference between them.

5. Write the general form of the following statements:
   (i) *if* statement with one alternative
   (ii) *if* statement with two alternatives
   (iii) *if* statement with multiple alternatives
   (iv) *switch* statement

6. Rewrite the program given in the example 4 using *if* statement.
7. Attempt the following parts:
   (i) Assuming x is 10.0 and y is 15.0, what are the values of the following conditions:
      a) x != y
      b) x < x
      c) x >= y – x
      d) x == y + x – y

      [*Hint*: The value of a true condition is **1** and a false condition is *0*]
   (ii) Write an expression to test each of the following relationships.
      a) *age* is from 18 to 25.
      b) *temperature* is less than 40.0 and greater than 25.0.
      c) *year* is divisible by 4 (Hint: use %).
      d) *speed* is not greater than 80.
      e) y is greater than x and less than z.
      f) *w* is either equal to 6 or not greater than 3.

   Note: The bold face words represent variables' names.

   (iii) Write assignment statement for the following:
      a) Assigns a value of 1(one) to the variable *test* if *k* is in the range –*m* through +*m*, inclusive. Otherwise, assigns a value of zero.
      b) Assigns a value of one (1) to the variable *lowercase* if *ch* is a lowercase letter; otherwise, assigns a value of zero (0).
      c) Assigns a value of one (1) to the variable *divisor* if *m* is a divisor of *n*; otherwise, assigns a value of zero(0).

   Note: The bold face words represent variables' names.

8. Write an interactive program that contains an *if* statement that may be used to computer the area of a square (*area* = *side*^{2}) or a triangle (*area* = ½ × *base* × *height*) after prompting the user to type the first character of the figure name (S or T).

9. A year is a leap year if it is divisible four, except that any year divisible by 100 is a leap year only if it is divisible by 400. Write a program that inputs a year such as 1996, 1800, and 2010, and displays "Leap year" if it is a leap year, otherwise displays "Not a leap year".

10. Write a program that inputs temperature and displays a message according to the following data:

Temperature	Message
Greater than 35	Hot day
Between 25 and 35 (inclusive)	Pleasant day
Less than 25	Cool day

11. Write a program that inputs obtained marks of a student, calculates percentage (assume total marks are 1100), and displays his/her grade. The grade should be calculated according to the following rules:

Percentage	Grade
More than or equal to 80	A+
Between 70 (inclusive) and 80	A
Between 60 (inclusive) and 70	B
Between 50 (inclusive) and 60	C
Between 40 (inclusive) and 50	D
Between 33 (inclusive) and 40	E
Less than 33	F

12. Write a program that inputs two numbers and asks for the choice of the user, if user enters 1 then displays the sum of numbers, if user enters 2 then displays result of subtraction of the numbers, if user enters 3 then displays the result of the multiplication of the numbers and if user enters 4 then displays result of the division of the numbers (divide the larger number by the smaller number), otherwise displays the message "Wrong Choice". [Use a switch statement to implement the solution]

LOOP CONSTRUCTS

Chapter 12

12.1 OVERVIEW

We often encounter problems whose solution may require executing a statement or a set of statements repeatedly. In such situations, we need a structure that would allow repeating a set of statements up to fixed number of times or until a certain criterion is satisfied. In C, Loop statements fulfill this requirement. This chapter will provide the basis for writing iterative solutions to certain problems. Here, we shall introduce different loop constructs available in C.

Iteration is the third type of program control structure (sequence, selection, iteration), and the repetition of statements in a program is called a *loop*. There are three loop control statements in C, these are:

- *while*
- *do-while*
- *for*

12.2 WHILE STATEMENT

The *while* loop keeps repeating associated statements until the specified condition becomes false. This is useful where the programmer does not know in advance how many times the loop will be traversed. The syntax of the *while* statement is:

```
while (condition)
{
        statement(s);
}
```

The *condition* in the *while loop* controls the loop iteration. The statements, which are executed when the given condition is true, form the *body of the loop*. If the condition is *true*, the body of the loop is executed. As soon as it becomes false, the loop terminates immediately.

Example 1 **Write a program to print digits from 1 to 10.**

```
#include <stdio.h>

void main(void)
{
     int count;

     count = 1;
     while( count <= 10)
     {
          printf("%d\n", count);
          count = count + 1;
     }
}
```

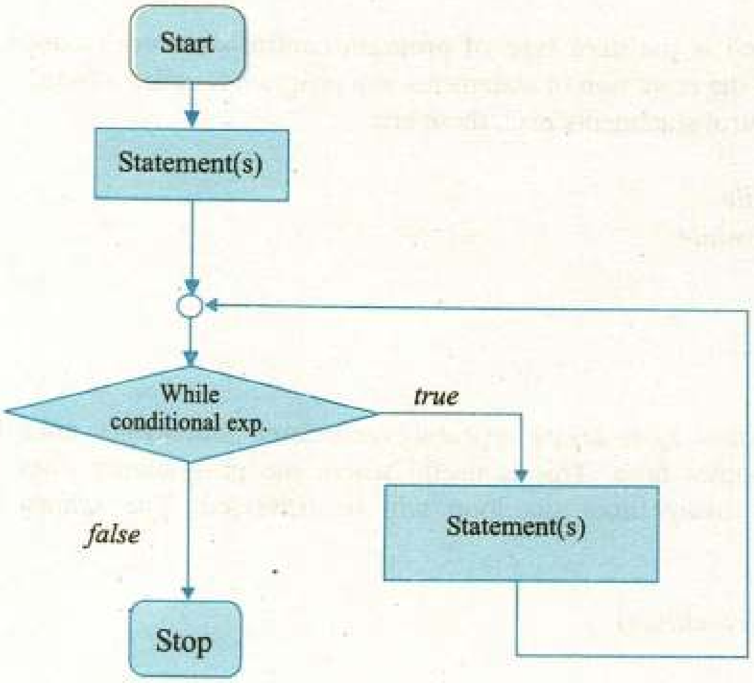


Fig. 12.1 Flowchart of the while loop

This is a simple program which demands iterative solution. It does not make sense to use ten printf statements to print ten digits; if so, what if we have to print digits from 1 to 1000? Should we write one thousand printf statements to accomplish the task? Certainly not, the right way to come up to the solution is to use a loop, which would execute ten times. Each time the loop executes, a number is printed which is incremented by one for every iteration until the required list of numbers is printed.

In this program, we use a variable *count* which is initialized to 1. The condition (count <= 10) depends on the value of this variable. Until the condition is *true*, the control will enter the *body of the loop*, and as soon as it becomes *false*, the control will exit from the loop and will jump to the next statement to the body of the loop. First time, when the condition is checked, it is found *true* as the value of *count* which is one, is less than ten. The control enters the body of the loop and the number "1" is printed. The next line of code increments the value of *count* by one, which becomes "2". After that, the control will immediately jump to the *while* statement where again the condition is tested which is still found *true*, as 2 is less than 10. The control again enters the body of the loop, and the number "2" is printed. The value of the variable *count* again increases by one and becomes "3". The control again transfer to the *while* statement. This process continues until the value of *count* becomes "11", making the condition *false*. When the condition becomes *false*, the control will exit from the loop.

The *count* is the *loop control variable.* A variable whose value controls the number of iterations is known as *loop control variable*. The compound statement, which is enclosed in braces, is the *body of the loop*.

In *while loop*, the *loop control variable* is always initialized outside the *body of the loop* and is incremented or decremented inside the *loop body*.

12.3 DO-WHILE LOOP

This is very similar to the *while loop* except that the test occurs at the end of the loop body. This guarantees that the loop is executed at least once. This loop is frequently used where data is to be read; the test then verifies the data, and loops back to read again if it was unacceptable. The syntax of the do-while statement is:

```
do
{
        statement(s);
}while (condition);
```

The important point about this loop is that unlike *while* loop, it ends with a semicolon. Omitting the semicolon will cause a syntax error. Let us re-write the program in *example 1* using *do-while* loop.

```
#include <stdio.h>

void main(void)
```

```
{
        int count;
        count = 1;
        do
        {
              printf("%d\n", count);
              count = count + 1;
        } while( count <= 10);
    }
```

Here, we achieve the same objective as in previous example but in a different way. The keyword do let the program flow to move into the *body of the loop* without checking any test condition. It means, whatever is written in the *loop body* always will be executed at least once. At the completion of execution of the *body of the loop*, the test condition is checked. If it is found *true*, the control is transferred to the first statement in the *body of the loop*, and if the condition is evaluated to *false*, the loop terminates immediately and the control moves to the very next instruction outside the loop.

The *do-while* loop is of great importance in situations where we need to execute certain statements at least once.

Example 2 **Your telephone connection may be in any of two states i.e., working or dead. Write a program that reads the current state of the telephone line; the user should enter w for working state and d for dead state. Any input, other than *w* or *d*, will be considered invalid. We want to force the user to enter a valid input value. This could be achieved by using a *do-while* loop. Let us consider the following program:**

```
#include <stdio.h>
void main( void )
{
     char state;
     do
     {
            printf("\nPlease enter the current state
            of the telephone line (enter \'w\' for
            working and \'d\' for dead) >");
            scanf("%c", &state);

        }while (state != 'w' && state != 'd');
    }
```

This program demonstrates a scenario where an invalid input is not processed. Until the user enters a valid input (d or *w*), the program repeatedly shows him (or her) the message for the valid input to be entered.

Here, the key point is the correct understanding of the test condition (state != 'w' && state != 'd'). It is a compound condition which is comprised of two sub conditions i.e. state != 'w' and state != 'd'. It should be noted that if two or more conditions are combined using logical AND operator to form a compound condition, the compound condition will be *true* only if all the sub conditions are *true* and if any of the sub conditions is *false*, the compound condition evaluates to *false*. Therefore, when the user enters an invalid input (suppose *e*), the first sub condition state != 'w' evaluates to *true* (because *e* is not equal to *w*), similarly the second sub condition state != 'd' also evaluates to *true*. Since both the sub conditions are true, therefore the compound condition also evaluates to *true* and the control flow returns back to the *printf* statement in the *body of the loop*. This process continues until the user enters a *w* or *d*. When the user enters a *d* or a *w*, one of the sub conditions evaluate to *false* causing the compound condition to be evaluated to *false* and the control flow exit the loop.

while vs do-while

- In *while* loop, the *body of the loop* may or may not execute depending on the evaluation of the test condition.
- In *do-while* loop, first the *body of the loop* is executed and then the test condition is checked. Hence it always executed at least once.

The *for* statement is another way of implementing loops in C. Because of its flexibility, most programmers prefer the *for* statement to implement loops. The syntax of the *for* loop is as follows:

for (*initialization expression; test condition; increment/decrement expression*)
{
    statement(s);
}

There are three expressions in *for* loop statement, these are

- Initialization of the loop control variable
- Test condition
- Change (increment or decrement) of the loop control variable

The *initialization expression* is executed in only the first iteration. Then the *loop condition* is tested. If it is *true*, the statements in the body of the loop are executed. After execution of the body of the loop, the increment/decrement expression is evaluated. It is very important to note that the initialization expression is only executed for the first iteration. For second and next iterations, the loop condition is tested, if it is true then the body of the loop is executed and then the increment/decrement expression is evaluated. After evaluation of the increment/decrement expression, the test condition is checked again and if it is true then the body of the loop executed. This process continues as long as the *loop condition* is true. When this condition is found to be *false*, the *for loop* is terminated, and the control transfers to the next statement following the *for loop*. Usually, we increment or decrement the loop control variable in the increment/decrement expression. Let us re-write the program in *example1* using *for* loop.

```
#include <stdio.h>
void main(void)
{
     int count;
     for (count = 1; count <= 10; count++)
          printf("%d\n", count);

}
```

There are three expressions in *for* loop statement, separated by semicolons. Two of the expressions i.e. *initialization expression* and *increment/decrement expression* are optional. We may omit these expressions. In this case, the *for* statement will be written as follows:

```
for (; count <= 10;)
```

The loop condition is mandatory. This can not be omitted. In this case we must have to initialize the loop control variable outside the *for* statement and it should be incremented or decremented inside the *loop body*.

12.4 NESTED LOOP

Nested loop means a loop inside the body of another loop. Nesting can be done up to any level. But, as the level of nesting increases, the complexity of the *nested loop* also increases. There is no restriction on the type of loops (while, do-while, or for) that may be placed in the body of other loops. For example, we can place one or more *while* or *do-while* loops in the body of *for loop*. Similarly, one or more *for loops* can be placed in the body of while or do-while loop.

Example 3 **Write a program that will print asterisks (*) according to the pattern shown in the fig. 12.2.**

```
*******
******
*****
****
***
**
*
```

Fig. 12.2 asterisks pattern

```
# include <stdio.h>

void main(void)
{
     int inner;

     for(int outer=7; outer>=1; outer--)
     {
          inner = 1;
          while( inner <= outer)
          {
               printf("*");
               inner++;
          }
          printf("\n");
     }
}
```

In this program, a *while* loop is used inside the body of *for* loop, which shows a nested loop. The outer loop is controlled by the loop control variable i.e., *outer*. The outer loop is executed seven times. For each iteration of the outer loop, the inner loop executes until the value of the inner loop control variable i.e., *inner* is less than or equal to the value of the variable *outer*. It should be noted that each time a new iteration for the outer loop starts, the variables used in the inner loop are re-initialized and re-processed.

For the first iteration of the outer loop, the variable '*outer*' is initialized to 7, and in all next iterations it is decremented by 1. This process continues until the value of the variable is greater than or equal to 1. For the first iteration of the outer loop, the inner loop executes seven times, and for the 2nd iteration it executes six times, similarly for the last seventh iteration, the inner loop executes just one time. Each time when the inner loop is terminated, the statement `printf("\n")` moves the cursor to the start of the new line.

Note:

Many programs require a list of items to be entered by the user. Often, we don't know how many items the list will have. For example, to find the average marks of a class, we have to input the marks of every student of the class. Similarly to calculate the sum of a series, we have to input the list of numbers in the series. There are so many other situations where the solution demands to enter a list of items to process. Loops are very useful to develop solutions for such problems. Each time the loop body is repeated, one or more data items are input. But, often we don't know how many data items will be input by the user. Therefore, we must find some way to signal the program to stop reading and processing new data.

One way to do this is to instruct the user to enter a unique data value, called a *sentinel value*, after the last data item. The loop condition tests each data item and causes loop exit when the sentinel value is read. Choose the sentinel value carefully; it must be a value that could not normally occur as data. The general form of a sentinel-controlled loop is:

1. Get the first line of data
2. While the sentinel value has not been encountered
3. Process the data line
4. Get another line of data

Sentinel Value is an end marker that follows the last item in a list of items

Example 4 **Write a program to find the average marks of the students in a class.**

```
#include <stdio.h>

void main(void)
{
     int sum = 0, marks, total_students = 0;
     float average;
     do{
          printf("Enter marks of the student (or any -ve
          number to quit) >");
          scanf("%d", &marks);

          if (marks >= 0)
          {
              total_students++;
              sum += marks;
          }
      } while(marks >= 0);
      if (total_students > 0)
      {
          average = sum / (float)total_students;
          printf("The average marks of the class are:
          %f\n", average);
      }
      else
          printf("Please enter the marks of at least one
          student to calculate average\n");
}
```

This program demonstrates a typical implementation of sentinel loop. Size of the class does not matter, whatever it is, the average will be calculated in the same way. Here *any negative number* may act as the sentinel value because no student can have negative marks. However, zero would not be a wise option because there can be a student with zero marks.

The program reads the marks until the user enters a negative number. For every valid input (zero and +ve numbers) the control switches to the body of the while loop. In the loop body, the *total_students* is incremented by one and the *sum* is accumulated. As soon as a negative number is entered, the *sentinel while* loop is terminated. The next line to the end of the *while loop* is an *if* statement, which checks the count for total students i.e., *total_students* to ensure that the marks of at least one student have been entered. Omitting this *if* statement may crash the program. It is because of the formula for calculating average where the *sum* is divided by *total_students*. When marks of any students are not entered, the value of *total_students* is zero and calculating average for zero students will result in a runtime error of division by zero. So, to avoid this possible error first the value of the variable *total_students* is cheked; if its greater than zero then the average is calculated otherwise a message is shown to the user to enter the marks of at least one student. Now, notice the average formula i.e.,

```
average = sum / (float)total_students;
```

We have used the keyword *float* in parenthesis before the variable *total_students*. The reason is that both the variables *sum* and *total_students* are integers. So, their division will be integral division in which the fractional part is truncated. Hence, the result will not be accurate. Writing *float* in parenthesis i.e. (*float*) before the integer variable name (i.e. *total*_students) causes the integer variable to temporarily act as a float variable for this particular calculation. It is done to preserve the fractional part in the result. The integer variable (*total_students*) will act as an integer for all other calculations. The effect of this change is strictly associated with that particular calculation. This phenomenon is known as *type casting*.

12.5 GOTO STATEMENT

The *goto* statement performs an unconditional transfer of control to the named label. The label must be in the same function. A label is meaningful only to a goto statement; in any other context, the labeled statement is executed without regard to the label.

The general form of the goto statement is as follows:

goto label;

....

....

label: statement

Example 5 Write a program to calculate the square root of a positive number. Also handle negative numbers properly.

```
#include <math.h>
#include <stdio.h>

void main()
{
     float num;

positive:
     printf("Please Enter a positive number: ");
     scanf("%f", &num);

     if (num < 0)
          goto positive;
     else
          printf("Square root of %0.2f is %0.2f", num,
          sqrt(num));
}
```

If the user enters a negative number, the control transfers to the label *positive*.

Exercise 12c

1. Fill in the blanks:

   (i) There are ______________ types of loop in C.

   (ii) The loop condition controls the loop ______________.

   (iii) In ______________ loop, first the body of the loop is executed and then the test condition is checked.

   (iv) ______________ means a loop within the body of another loop.

   (v) The ______________ statement performs an unconditional transfer of control to the named label.

   (vi) Repetition of statements in a program is called ______________.

   (vii) There are ______________ expressions in for loop statement.

   (viii) The body of the while loop executes only if the specified condition is ______________.

   (ix) Increase in the level of nesting increases the ______________ of the nested loop.

   (x) A ______________ is meaningful only to a goto statement.

2. Write T for true and F for false statement.

   (i) There is no difference between while and do-while loop.

   (ii) The body of a while loop may or may not execute.

   (iii) The do-while loop always executes at least once.

   (iv) var++ is an example of prefix increment.

   (v) The condition of an infinite loop never becomes true.

   (vi) Initialization expression is optional in *for* loop.

   (vii) `for( i = 1; i <= 10; i++);` is an infinite loop.

   (viii) Loop is a decision making construct.

   (ix) A *while* loop can not be used in the body of a *for* loop.

   (x) In type casting, a variable of one type behaves as the variable of another type temporarily.

3. Define a loop. How many loops are available in C? Compare the following loops:
   a) while loop and do-while loop
   b) while loop and for loop

4. What is a sentinel controlled loop and how it is implemented? Discuss some of the situations where it can be useful.

5. Write the output of the following program fragments:

(a)
```
k = 0;
        while (k <= 5)
        {
                printf("%3d %3d\n", k, 10 – k)
                k++;
        }
```

(b) Trace the output of the following piece of code.
```
        j = 10;
        for (int i = 1; i <= 5; ++i)
        {
                printf("%d  %d\n", i, j);
                j -= 2;
        }
```

6. Correct the following code segments according to the given instructions:
   a) Insert braces where they are needed and correct errors if any. The corrected code should accept five integers and should display their sum.

```
count = 0;
while (count <= 5);
count += 1;
printf("Next Number >");
scanf("%d", &next_num);
next_num += sum;
printf("%d numbers were added; \n", count);
printf("Their sum is %d.\n", sum);
```

b) Rewrite the following code segment using a do-while statement

```
sum = 0;
for (odd = 1; odd < n; odd = odd + 2)
          sum = sum + odd;
printf("Sum of the positive odd numbers less than %d is %d\n", n,
sum);
```

7. Trace the output of the following program segments, assuming m is 3 and n is 5:

a)
```
for ( k = 1; k <= n;  ++k)
{
          for (j = 0; j < k; ++j)
          {
                    printf("*");
          }
          printf("\n");
}
```

b)
```
for (k = n; k > 0; --k)
{
          for (j = m; j > 0; --j)
          {
                    printf("*");
          }
          printf("\n");
}
```

c) (a) Re-write the program in example3 by replacing the inner *while* loop with

- a *for* loop
- a *do-while* loop

(b) Re-write the program in example3 by replacing the outer *for* loop with

- a *while* loop
- a *do-while* loop

9. Write a program that inputs a number and displays the message "Prime number" if it is a prime number, otherwise displays "Not a prime number".

10. Write a program that displays the first 15 even numbers.

11. Write a program that inputs a number, and displays its table according to the following format:

    Suppose the number entered is 5, the output will be as follows:

```
        5 * 1 = 5
        5 * 2 = 10
        5 * 3 = 15
.
.
.
5 * 10 = 50
```

12. Write a program using do-while loop that repeatedly prompts for and takes input until a value in the range 0 through 15 inclusive is input. The program should add all the values before exiting the loop and displays their sum at the end.

13. Write a program that produces the following output:

```
0
0  1
0  1  2
0  1  2  3
0  1  2  3  4
0  1  2  3  4  5
```

14. Write a program the produces the following output:

```
0     1
1     2
2     4
3     8
4     16
5     32
6     64
```

FUNCTIONS IN C 13

13.1 OVERVIEW

The idea of modular programming is the result of inspiration from the hardware manufacturing where replicable components of different items are available. If a component of an item gets out of order, it is replaced with a newer one. Many different components from different manufacturers can be combined together to form a hardware device such as computers, cars, and washing machines. Functions are the building blocks of C programs. They encapsulate pieces of code to perform specific operations. Functions allow us to accomplish the similar kinds of tasks over and over again without being forced to keep adding the same code into the program. Functions perform tasks that may need to be repeated many times.

In programs we have seen so far, the whole program logic was contained in a single *main* function. This style of writing programs is known as *unstructured programming*. Recall chapter 8 where we have discussed the difference between unstructured and structured programming. Here we shall discuss structured programming approach. It is a modular way of writing programs. The whole program logic is divided into number of smaller modules or functions. The *main* function calls these functions where they are needed. A *function* is a self-contained piece of code with a specific purpose.

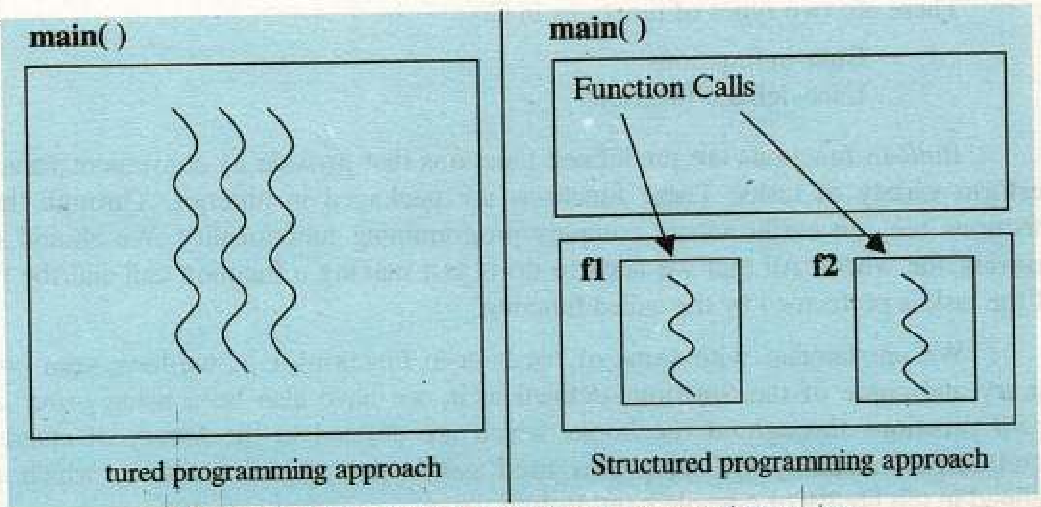


Fig. 13.1 Difference between structured and unstructured approaches

The above figure demonstrates the idea of structured and unstructured programming.

13.2 IMPORTANCE OF FUNCTIONS

A program may have repetition of a piece of code at various places. Without the ability to package a block of code into a single function, programs would end up being much larger. But the real reason to have functions is to break up a program into easily manageable chunks. The use of functions provides several benefits. Some of them are:

- They make programs significantly easier to understand and maintain. The main program can consist of a series of function calls rather than countless lines of code.
- Functions increase reusability of the code. Well written functions may be reused in multiple programs. The C standard library is an example of the reuse of functions.
- Different programmers working on one large project can divide the workload by writing different functions, hence ensuring the parallel development of the software.
- Functions can be executed as many times as necessary from different places in the program
- When an error arises, rather than examining the whole program, the infected function is debugged only.

13.3 TYPES OF FUNCTIONS

There are two types of functions in C:

(ii) Built-in functions
(iii) User-defined functions

Built-in functions are predefined functions that provide us convenient ways to perform variety of tasks. These functions are packaged in libraries. Through these functions we can easily access complex programming functionality. We should not reinvent the wheel. All that we need to do is just making a function call and the rest of the task is performed by the called function.

We are familiar with some of the built-in functions, e.g., we have seen *ctype* library and some of the functions defined in it, we have also been using *printf* and *scanf* functions throughout the book, which are defined in the library of standard input/output, similarly we have also used *getch* and *getche* functions which are defined in the library of console input/output.

To use a built-in function in C, we must have to include in wer program the header file containing its declaration. For example to user printf( ) and scanf( ), we have to include stdio.h file in wer program.

Built-in functions are not sufficient for solving every kind of problem. A programmer may need to write his/her own functions depending on the nature of problem being solved. Such functions are called *user-defined functions*.

13.4 WRITING FUNCTIONS IN C

We are familiar with the *main*( ) function, which is the mandatory part of every C program. In 8th chapter, we have introduced the structure of the *main* function. Every function in C has almost the same basic structure. A function in C consists of a *function header* which identifies the function followed by the body of the function between curly braces containing the executable code for the function. Every function in C is written according to the following general form:

```
returen_type FunctionName (parameter_list)
{
    Executable Statement(s)

    return expression;
}
```

13.4.1 Function Header

The first line of function definition is called the function header i.e.

```
return_type FunctionName (parameter_list)
```

It consists of three parts:

- The type of the return value
- The name of the function
- The parameters of the function enclosed in parentheses

The `return_type` can be any valid data type. If the function does not return a value, the return type is specified by the keyword **void**. A function that has no parameter specifies the keyword **void** as its parameter list. Hence, a function that has no parameter and does not return any value to the calling function will have the header:

```
void FunctionName (void)
```

However the keyword *void* is optional. The above function header for a function that has no argument can be re-written as follows:

```
void FunctionName()
```

13.4.2 The Function Body

Variables declaration and the program logic are implemented in the function body. Function body makes use of the arguments passed to the function. It is enclosed in curly braces. A function can be called in the body of another function.

13.4.3 The `return` Statement

The *return* statement is used to specify the value retuned by a function. The general form of *return* statement is:

```
return [expression];
```

When the *return* statement is executed, *expression* is evaluated and returned as the value of the function. Execution of the function stops when the *return* statement is executed, even if there are other statements still remaining in the function body. If the type of the return value has been specified as *void* in the function header then there is no need to use a return statement.

13.5 FUNCTION PROTOTYPE

The compiler must know functions used in the program. That's why we include corresponding header files in the source program before using built-in functions such as stdio.h and conio.h etc. A header file contains the prototypes of the functions provided by the library. The compiler actually needs enough information to be able to identify the function that we are using. A *function prototype* is a statement that provides the basic information that the compiler needs to check and use a function correctly. It specifies the parameters to be passed to the function, the function name, and the type of the return value. The general form of the function prototype is as follows:

```
return_type FunctionName (parameter_list);
```

We might be surprising at the above statement. It looks like the function header; yes it is, but with a semicolon at the end.

The prototype for a function which is called from another function must appear before the function call statement. Functions prototypes are usually placed at the beginning of the source file just before the function header of the *main* function.

13.6 CALLING A FUNCTION

Function call is a mechanism that is used to invoke a function to perform a specific task. A function call can be invoked at any point in the program. In C the function name, the arguments required and the statement terminator (;) are specified to invoke a function call.

When function call statement is executed, it transfers control to the function that is called. The memory is allocated to variables declared in the function and then the statements in the function body are executed. After the last statement in the function is executed, control returns to the calling function.

13.7 LOCAL VARIABLES AND THEIR SCOPE

When the program executes, all variables are created in memory for a limited time period. They come into existence from the place where they are declared and then they are destroyed. The duration in which a variable exists in memory is called *lifetime of the variable*.

Note: Operating system manages the allocation and de-allocation of memory for all variables in wer programs. So, by *destroying a variable* we mean returning the memory allocated to a variable back to the operating system for other programs.

The *scope of a variable* refers to the region of a program in which it is accessible. The name of a variable is only valid within its scope. So a variable can not be referred outside its scope. Any attempt to do so will cause a compiler error.

All variables that we have declared so far have been declared within a block – that is, within the extent of a pair of curly braces. These are called *local variables* and have *local scope*. The scope of a local variable is from the point in the program where it is declared until the end of the block containing its declaration.

Example 1

```
#include <stdio.h>
void main()
{
	int nCount = 0;

	if (nCount = 0)
	{
		int chk;

		chk = 10;
	}
	printf("%d", chk);
}
```

We have used two variables in this program; these are *nCount*, and *chk*. Both of these are local variables. But, they have different scope. The scope of the variable *nCount* is the block of *main*( ) function i.e., from its point of declaration to the end of the *main*( ) function. Whereas the scope of the variable *chk* is the block of *if* statement i.e., from its point of declaration until the end of the block of *if* statement.

These variables can only be referenced within their respective scopes. Any reference made to them outside of their scopes would be illegal, thus the program causes the following *compiler error*.

'chk' : undeclared identifier

This is because in the last *printf( )* statement of the program, the variable chk is referenced outside of *if* block i.e., out of its scope, which is illegal. The lifetime of local variables is the duration in which the program control remains in the block in which they are declared. As soon as the control moves outside of their scope, these variables are destroyed.

13.8 GLOBAL VARIABLES AND THEIR SCOPES

The variables which are declared outside all blocks i.e. outside the main( ) and all other functions are called *global variables* and have *global scope*. They are accessible from the point where they are declared until end of the file containing them. It means they are visible throughout all the functions in the file, following their point of declaration. The lifetime of global variable is until the termination of the program. They exist in memory from the start to the end of the program.

Note: It is very important to understand that every time the block of statements containing a declaration for a local variable is executed, the variable is created anew, and if we specify an initial value for the local variable, it will be reinitialized each time it is created.

Example:

```
#include <stdio.h>
void main()
{
        int nCount = 1;

        clrscr();
        While(nCount <= 10)
        {
                int chk = 10;
                printf("%d\t", chk);

                chk = chk + 1;
                nCount++;
        }
}
```

In this program, each repetition of the loop prints the same value of the variable *chk*. The addition to the value of *chk* will have no effect, because at the end of execution of the body of the loop, the control moves outside the loop body (which is also the scope of *chk* variable) and returns to the *while* statement; this causes the the *chk* variable to be destroyed in each repetition. The *chk* variable is again created in the next repetition and gets destroyed at the end of the repetition. This process continues until the loop condition is true.

Output:

```
10    10    10    10    10    10    10    10    10    10
```

Example 2

```
#include <stdio.h>
void Counter(void);
int nCount = 0;

void main()
{
        for (int n = 0; n <= 10; n+=2)
            Counter();

        Printf("nCount = %d", nCount);
    }
    void Counter(void)
    {
        nCount++;
    }
```

This is a simple program which demonstrates the use of global variable. Here, we have declared a global variable i.e., *nCount* outside the *main* and the *Counter* functions. This is not contained in any block. The global variable *nCount*, the function *main*, and the function *Counter* all are defined in the same file. Because, the variable *nCount* is declared on top of the two functions, therefore it is visible within them. The function *Counter*, increments the value of *nCount* by one each time it is called. The main() executes a loop six times and call the function *Counter* to increment the value of nCount. The value of the variable nCount is printed as the final *output* of the program i.e.,

Output:

```
nCount = 6
```

Note: The point to be noted here is that the variable **nCount** is declared outside the functions main() and Counter(), but they manipulate it as if it was declared within them. The nCount is created in memory before the start of execution of the main() and exists until the execution of the program ends.

13.9 FUNCTIONS WITHOUT ARGUMENTS

The simplest type of function is one that returns no value and no arguments are passed to them. The return type of such functions is void and the Parameter_List may either be empty or containing the keyword void. Lets consider the following example.

Example 3 **Write a function named Print_Asterisks that will print asterisks (*) according to the pattern shown in the fig. 13.2. and invoke a function call from the function main to print the asterisks.**

```
*******
******
*****
****
***
**
*
```

Fig. 13.2 asterisks pattern

```
#include <stdio.h>

void Print_Asterisks(void); //function
                            // prototype
void main(void)
{
    // Function call
    Print_Asterisks();
}
void Print_Asterisks(void) //function header
{
    int inner;

    for(int outer=7; outer>=1; outer--)
    {
          inner = 1;
          while( inner <= outer)
          {
                printf("*");
                inner++;
          }
          printf("\n");
    }
}
```

We have discussed this program in the previous chapter, but here we have followed a different approach. The next line to the #include directive is the *prototype* for the function Print_Asterisks( ). It tells the compiler about the function, its return_type and number of parameters (void) in this case. Our *main* function consists of just one line of code i.e.,

```
Print_Asterisks();
```

It represents a *function call* to the function Print_Asterisks( ). We can think of a function as a worker who takes necessary steps to accomplish the task assigned to him.

Similarly, the function `Print_Asterisks()`is capable of printing asterisks in a specific order. When the function call statement is executed, the control is immediately transferred to the Print_Asterisks function. Memory is allocated to the variables *inner* and *outer*. Then comes the *for* and *while* loops, which print asterisks. When the task is completed, the control is transferred to the function *main* from the function Print_Asterisks, and the memory allocated to the variables *inner* and *outer* is returned to the operating system again. Then, the control is transferred to the next statement to the function call statement in the calling function i.e., main(). As there is no statement in the *main* function other than the function call, so the program will terminate.

13.10 FUNCTIONS THAT RETURN A VALUE AND ACCEPT ARGUMENTS

So far, we have discussed simple functions that return no value to the calling function. However, we may need a function that could return a value and arguments could be passed to it. In previous chapters, we have seen a number of such built-in functions e.g., sqrt(), toupper(), tolower() etc. Here, we shall learn to write these types of functions in C. Let's consider the general form of function header:

```
return_type FunctionName(parameter_list)
```

The *return_type* specifies the data type of the value that the function returns. *Parameter_list* is a coma separated list which specifies the data type and the name of each parameter in the list.

Example 4

```
#include <stdio.h>
int Add(int n1, int n2);
void main()
{
        int a, b;
        int sum;
        clrscr();          //clears   the   previous   output
                           from //the screen
        printf("Enter values for 'a' and 'b' >:");
        scanf("%d %d", &a, &b);

        sum = Add(a, b);
        printf("%d + %d = %d", a, b, sum);
    }
    int Add(int n1, int n2)
    {
        return n1 + n2;
    }
```

Suppose the user enters 12 and 15 for *a* and *b* respectively then the *output* of the of the program will be:

```
12 + 15 = 27
```

The 7th line of code in the *main* function is a *function call* to the *Add*( ) function. The Add( ) requires two parameters of type *int* to be passed to it. In the function call, we have passed two variables i.e., *a* and *b* of type *int* to the function. These arguments (i.e., variables *a* and *b*) are called *actual arguments* or *actual parameters* of the function. These are local variables and their scope is the body of *main* function. Whereas the parameters specified in the function header (i.e., *n1* and *n2*) are called *formal arguments* or *formal parameters* of the function and their scope is the body of *Add* function. These are also called dummy arguments.

When parameters are passed to a function, the value of actual parameters is copied in the formal parameters of the functions. The function uses its formal parameters for processing data passed to it. Any change made to the value of formal parameters does not affect the value of actual parameters. Here, the values of *a* and *b* are copied in *n1* and *n2* respectively. The function *Add* returns the sum of the two values to the *main* function which is then assigned to the variable *sum*.

Example 5

```
#include <stdio.h>
float Area_of_Triangle(int base, int altitude);
void main()
{
    int a, b;
    float area;

    printf("Enter value for altitude: ");
    scanf("%d", &a);
    printf("Enter value for base: ");
    scanf("%d", &b);
    area = Area_of_Triangle(a, b);
    printf("Area of triangle is %.2f", area);
}

float Area_of_Triangle(int base, int altitude)
{
    return (0.5*base*altitude);
}
```

Output

Suppose the user enters 25 and 45 for altitude and base respectively, then the output of the program will be:

```
Area of triangle is 562.50
```

Exercise 13c

1. Fill in the blanks:
   (i) A __________ is a self contained piece of code.
   (ii) Pre-defined functions are packaged in ______________.
   (iii) A ______________ provide basic information about the function to the compiler.
   (iv) The duration for which a variable exists in memory is called its _________.
   (v) ___________ of a variable refers to the region of the program where it can be referenced.
   (vi) __________ variables are declared outside all blocks.
   (vii) A function can not return more than ______ value(s) through **return** statement.
   (viii) The parameters specified in the function header are called __________ parameter.
   (ix) The parameters passed to a function in the function call are called _________ parameters.
   (x) Functions help to achieve ___________ programming.

2. Choose the correct option:
   (i) Function prototypes for built-in functions are specified in:
   a) source files b) header files
   c) object files d) image files

   (ii) Global variables are created in:
   a) RAM b) ROM
   c) hard disk d) cache

   (iii) Which of the following is true about a function call?
   a) Stops the execution of the program
   b) Transfers control to the called function
   c) Transfers control to the *main* function
   d) Resumes the execution of the program

   (iv) Which of the following looks for the prototypes of functions used in a program?
   a) linker b) loader
   c) compiler d) parser

(v) Memory is allocated to a local variable at the time of its:
a) declaration b) destruction
c) definition d) first reference

(vi) The name of actual and formal parameters:
a) may or may not be same b) must be same
c) must be different d) must be in lowercase

(vii) Formal arguments are also called:
a) actual arguments b) dummy arguments
c) original arguments d) referenced arguments

(viii) printf() is a:
a) built-in function b) user-defined function
c) local function d) keyword

(ix) A built-in function:
a) can not be redefined b) can be redefined
c) can not return a value d) should be redefined

(x) In a C program, two functions can have:
a) same name
b) same parameters
c) same name and same parameters
d) same name but different parameters

3. Write T for true and F for false statement.
(i) In C, arguments can be passed to a function only by value.
(ii) There can be multiple *main* functions in a C program.
(iii) A function can be called anywhere in the program.
(iv) In C, every function must return a value.
(v) A user-defined function can not be called in another user-defined function.
(vi) A function can be called only once in a program.
(vii) Scope of a local variable is the block in which it is defined.
(viii) Global variables exist in memory till the execution of the program.
(ix) An unstructured program is more difficult to debug than a structured program.
(x) Function body is an optional part of the function.

4. What is a function? How many types of functions are used in C? Discuss the difference between them.

5. Differentiate the following:
   (i) Function Definition and Function Declaration
   (ii) Global and Local variables
   (iii) Scope and Lifetime of a variable
   (iv) Function prototype and Function header
   (v) Formal parameters and Actual parameters of a function

6. How is a function call made in a C program? Discuss briefly.

7. Answer the following:
   (i) What is the purpose of a function argument?
   (ii) How many (maximum) values can a function return using **return** statement?
   (iii) When is a function executed, and where should a function prototype and function definition appear in a source program?
   (iv) Write three advantages of functions.

8. Write a program that call two functions Draw_Horizontal and Draw_Vertical to construct a rectangle. Also write functions Draw_Horizontal to draw two parallel horizontal lines, and the function Draw_Vertical to draw two parallel vertical lines.

9. Write a program that prompts the user for the Cartesian coordinates of two points (x_1, y_1) and (x_2, y_2) and displays the distance between them. To compute the distance, write a function named Distance( ) with four input parameters. The function Distance( ) uses the following distance formula to compute the distance and return the result to the calling function:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

10. Write a program that prompts the user to enter a number and then reverse it. Write a function Reverse to reverse the number. For example, if the user enters 2765, the function should reverse it so that it becomes 5672. The function should accept the number as an input parameter and return the reversed number.

11. Write a function named *Draw_Asterisks* that will print asterisks (*) according to the pattern shown in the following and make a function call from the function *main* to print the asterisks pattern.

```
*********
*******
*****
***
*
```

12. Write a function Is_Prime that has an input parameter i.e num, and returns a value of 1 if *num* is prime, otherwise returns a value of 0.

13. Write a complete C program that inputs two integers and then prompts the user to enter his/her choice. If the user enters 1 the numbers are added, for the choice of 2 the numbers are divided, for the choice of 3 the numbers are multiplied, for the choice of 4 the numbers are divide (divide the larger number by the smaller number, if the denominator is zero display an error message), and for the choice of 5 the program should exit. Write four functions Add(), Subtract(), Multiply() and Divide() to complete the task.

14. Write a program that prompts the user to enter a number and calls a function Factorial() to compute its factorial. Write the function Factorial() that has one input parameter and returns the factorial of the number passed to it.

15. Write a function GCD that has two input parameters and returns the greatest common divisor of the two numbers passed to it. Write a complete C program that inputs two numbers and call the function GCD to compute the greatest common divisor of the numbers entered.

FILE HANDLING IN C

Chapter 14

14.1 OVERVIEW

So far we have been writing programs to work with temporary data. The user had to enter data each time the program was executed. All programs, we have seen in previous chapters, were unable to store data and results permanently. The data is stored on permanent storage in the form of files. A *file* is a set of related records. Here, we shall explore the basic file handling features of C.

14.2 THE STREAM

Although C does not have any built-in method of performing file I/O, however the C standard library (*stdio*) contains a very rich set of I/O functions providing an efficient, powerful and flexible approach for file handling.

A very important concept in C is the *stream*. A *stream* is a logical interface to a *file*. A *stream* is associated to a file using an *open operation*. A *stream* is disassociated from a file using a *close operation*. There are two types of streams:

- **Text Stream:** A text stream is a sequence of characters. In a text stream, certain character translations may occur (e.g., a newline may be converted to a carriage return/line-feed pair). This means that there may not be a one-to-one relationship between the characters written and those in the external device
- **Binary Stream:** A binary stream is a sequence of bytes with a one-to-one correspondence to those on the external device (i.e., no translations occur). The number of bytes written or read is the same as the number on the external device. (However, an implementation-defined number of bytes may be appended to a binary stream (e.g., to pad the information so that it fills a sector on a disk).

Note: In C, a **file** refers to a disk file, the screen, the keyboard, a port, a file on tape, and so on.

14.3 NEWLINE AND EOF MARKER

A *text file* is a named collection of characters saved in secondary storage e.g. on a disk. A text file has no fixed size. To mark the end of a text file, a special *end-of-*

file character is placed after the last character in the file (denoted by *EOF* in C). When we create a text file using a text editor such as *notepad*, pressing the ENTER key causes a newline character (denoted by \n in C) to be placed at the end of each line, and an EOF marker is placed at the end of the file. For example, consider the organization of text in a text file in the following figure; There are four lines of text, each ends with a newline character i.e.,\n except the last one which ends with an end of file marker i.e., EOF.

I		l	o	v	e		P	a	k	i	s	t	a	n	\n			
I		a	m		a		s	t	u	d	e	n	t	\n				
I		w	o	r	k		h	a	r	d	\n							
M	a	y		A	l	l	a	h		b	l	e	s	s		u	s	EOF

Fig. 14.1 Organization of text in a text file

14.3 OPENING A FILE

Before reading from or writing to a file, it must be opened. All standard file handling functions of C are declared in *stdio.h*. Thus it is included in almost every program. To open a file and associate it with a *stream*, the *fopen*() function is used. Its prototype is shown here:

```
FILE* fopen(const char* filename, const char* mode);
```

The fopen() function takes two parameters. The first is the name of the file. If the file is not in the current directory then its absolute path is be given. In this case, we need to escape the backslashes (i.e., use \\ instead of \) in the absolute path. For example:

```
fopen("c:\\Program Files\\MyApplication\\test.txt", "r");
```

The second parameter of fopen() function is the open "mode". It needs to be a string – not just a character. (Use double quotes, not single quotes). The "r" means we wish to open the file for reading (input). We could use a "w" if we wanted to open the file for writing (output).

The fopen() function returns the NULL pointer if it fails to open the file for some reason. The most common reason for fopen() to fail is that the file does not exist. There are, however, other reasons for failure so don't assume that is what went wrong for certain. For example:

```
FILE *fp;
if ((fp = fopen("myfile", "r")) == NULL)
{
  printf("Error opening file\n");
  exit(1);
}
```

14.4.1 File Opening Modes

A file can be opened in any of the following modes:

r	Open a text file for reading. The file must already exist.
W	Open a text file for writing. If the file already exists its contents are overwritten. If it does not exist, it will be created.
A	Open a text file for append. Data is added to the end of the existing file. If the file does not exist, it is created.
R+	Open a text file for both reading and writing. The file must already exist.
W+	Open a text file for reading and writing and its contents are overwritten. If the file does not exist, it is created.
A+	Open a text file for both reading and appending. If the file does not exist, it is created for both reading and writing.

14.4.2 The File Pointer

A file pointer is a variable of type FILE that is defined in *stdio.h*. To obtain a file pointer variable, a statement like the following is used:

```
FILE* fp;
```

We know the symbol '*' as the arithmetic multiplication operator. But, it has entirely different meaning when used with a data type such as int, double, or FILE. It represents a *pointer* to the variable of type with which it is used e.g. int* represents a pointer to an integer, float* represents a pointer to a float variable, and FILE* represents a pointer to a variable of type FILE. Conceptually, a *pointer* is a memory cell whose content is the address of another memory cell.

Consider the following line of code:

```
int*  var;
```

The variable `var` is a *pointer* to an integer type variable. It contains the address of a memory location (i.e. 0002) where an integer value can be stored. The contents of the memory location pointed to by the pointer *var* are referred to as `*var`. A pointer is a memory location that contains the address of another memory location. The file pointer i.e. FILE* is a pointer to the *file information* which defines various properties of the file (including name, status and current position).

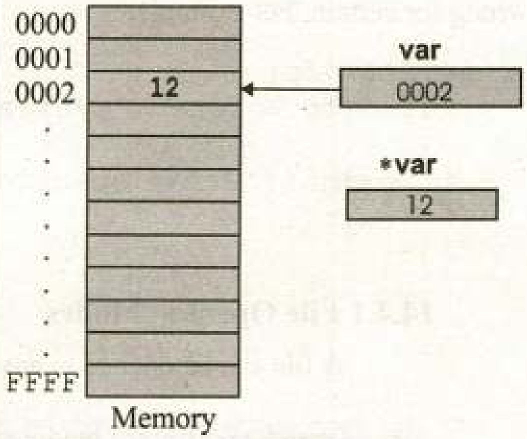


Fig. 14.1 Understanding the Pointer

Example 1 Consider the following program that demonstrates the use of pointers.

```
#include <stdio.h>
#include <conio.h>

void main(void)
{
     int* var;
     int num = 25;

     clrscr();
     var = &num;
     printf("Address of variable num is %x", &num);
     printf("\nContents (i.e. value) of num is %d",
     num);
     printf("\nAddress of memory location pointed to
     by var is %x", var);
     printf("\nContents of memory pointed to by var
     is %d", *var);
     }
```

It is clear from the program that a pointer type variable stores the address of a memory location containing the value, not the value itself. The address of the variable *num* i.e., *fff4* may be different when you would execute this program on your computer. This is because a different memory location may be assigned to the variable num each time the program is executed.

Output: Here is the output of the program:

```
Address of variable num is fff4
Contents (i.e. value) of num is 25
Address of memory location pointed to by var is fff4
Contents of memory pointed to by var is 25
```

14.5 CLOSING A FILE

When a program has no further use of a file, it should close it with **fclose()** library function. The syntax of fclsoe() is as follows:

```
int fclose(FILE* fp)
```

The *fclose*() function closes the file associated with `fp`, which must be a valid *file pointer* previously obtained using *fopen*(), and disassociates the *stream* from the file. It also destroys structure that was created to store information about file. The *fclose*() function returns *0* if successful and *EOF* (end of file) if an error occurs.

14.6 READING AND WRITING CHARACTERS TO A FILE

Once a file has been opened, depending upon its opening mode, a character can be read from or written to it by using the following two functions.

```
int getc(FILE* fp)
int putc(int ch, FILE* fp)
```

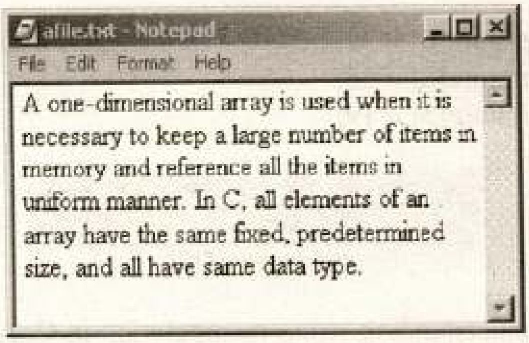
The *getc*() function reads the next character from the file and returns it as an integer and if error occurs returns *EOF*. The *getc*() function also returns *EOF* when the end of file is encountered.

The *putc*() function writes the character stored in the variable *ch* to the file associated with *fp* as an unsigned *char*. Although *ch* is defined as an *int* yet we may use a *char* instead. The *putc*() function returns the character written if successful or *EOF* if an error occurs.

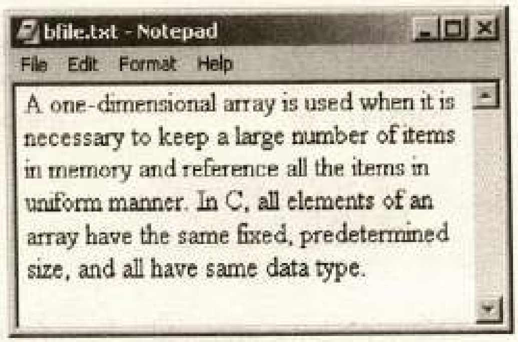
Example 2 **Write a program that reads a file and then writes its contents to** another file.

```
#include <stdio.h>

void main(void)
{
  FILE *input;
  FILE *output;
  int   ch;

  // Try to open the input file. If it fails, print a
  // message.
 if ((input = fopen("afile.txt", "r")) == NULL)
  {
    printf("Can't open afile.txt for reading!\n");
  }

  // Now try to open the output file. If it fails,
  // close the input.
  else if ((output = fopen("bfile.txt", "w")) ==
NULL)
  {
    printf("Can't open bfile.txt for writing!\n");
    fclose(input);
  }

  // If the files opened successfully, loop over the
  // input one character at a time.
else
{
     while ((ch = getc(input)) != EOF)
     {
           // Process ch and output it.
           putc(ch, output);
     }

     // Close the files
     fclose(input);
     fclose(output);
 }
}
```

Output: This program copies the contents of afile.txt to bfile.txt, both files are in current directory (i.e. the directory in which this .c file resides). The following figure shows the output of the program:



```
A one-dimensional array is used when it is
necessary to keep a large number of items in
memory and reference all the items in
uniform manner. In C, all elements of an
array have the same fixed, predetermined
size, and all have same data type.
```



```
A one-dimensional array is used when it is
necessary to keep a large number of items
in memory and reference all the items in
uniform manner. In C, all elements of an
array have the same fixed, predetermined
size, and all have same data type.
```

Fig. 14.2 Afile.txt is copied to bfile.txt by the program

14.7 STRING HANDLING

Until now, we have not discussed the topic of string handling. This book will remain incomplete without having a discussion on strings. In most of the programs we have to work with strings. For example, we may want to keep a list of names and telephone numbers of our friends, a shopkeeper may need to prepare records of items and their prices in his shop, and a law-enforcement agency might be interested in keeping records of criminals including their names, pictures, telephone numbers and addresses; in all of these cases we need to handle strings. So, in this section we shall see how strings are handled in a C program.

In different programs, we have been displaying strings on screen with *printf()* function. But still we are not familiar with string variables – the way C stores a string in a variable. Unlike variables of different numeric data types, C follows a different approach to handle strings. It stores a string as an *array* of characters. An **array** is a group of contiguous memory locations, which can store data of the same data type. Let us see how we can declare an array in C? The general form is:

data_type arr_name[n];

The *data_type* specify the type of data that is stored in every memory location of the array, arr_name describe the *array name*, and *'n'* is the subscript of array which shows the total number of memory locations in the array. For example, the statements:

```
int balls[6];
double temperature[10];
```

define two arrays named *balls* and *temperature* (two sets of six and ten contiguous memory locations as shown below). In *balls* we can store *six* integer values, whereas in *temperature* we can store *ten* floating point values. Each value of array can be accessed via its subscripts. For example, consider the following statements:

```
balls[0] = 4;          temperature[0] = 37;
balls[1] = 0;          temperature[2] = 26;
balls[4] = 6;          temperature[3] = 19;
```

0	1	2	3	4	5
4	0			6	

balls[6]

0	1	2	3	4	5	6	7	8	9
37		26	19						

temperature[10]

Now we have prepared a base for understanding string manipulation in C. As strings are array of characters in C, that's why it was necessary to have a concept of arrays. We shall not prolong our discussion on arrays as it is out of scope of this book. You will study more about this topic in next classes. However, here we shall briefly discus strings – the array of characters.

14.7.1 Declaring and Initializing String Variables

As we mentioned earlier, a string in C is implemented as an array. So declaring a string variable is the same as declaring an array of type *char*, such as:

```
char name[16];
```

the variable *name* can hold string from 0 to 15 characters long. The last character of every string in C is `'\0'`, the null terminator which indicates the end of the string. In this way the C let us manipulate each character of the string individually. Like variables of other data types, the strings can also be initialized:

char name[16] = "Lahore";

[0]	[1]	[2]	[3]	[4]	[5]	[6]	7	8	9	10	11	12	13	14	15
L	a	h	o	r	e	\0									

Notice the above figure showing the memory arrangement for the string variable *name*; the name[6] contains the character `'\0'`. This is the *null character* that marks the end of the string. This end marker allows the strings to have variable lengths. The rest of the memory locations in the array remains empty and are not allocated to any other variables. All of the C's string handling functions simply ignore whatever is stored in the cells following the *null character*. The following figure shows another string, longer than the previous, that the variable *name* can store.

char name[16] = "I love Pakistan";

[0]	[1]	[2]	[3]	[4]	[5]	[6]	7	8	9	10	11	12	13	14	15
I		l	o	v	e		P	a	k	i	s	t	a	n	\0

Notice that, in the initialization statement of the string we did not put a *null character* (\0) at the end. When we initialize a string, a null character is added at the end of it by default.

14.7.2 String Assignment

Assigning a value to a string variable is not as simple as assignment to other variables. For example, we can assign an integer value to a variable of type **int** and a floating point value to a variable of type *float* by using assignment operator (i.e., =). But, it does not work with strings. So the following statement will cause an error:

name = "I love Pakistan";

As *name* does not consist of a single memory location – it is an array. So, different characters are put in different memory locations of the array. This is done by copying every character of the string to respective index (subscript) of the array. For this purpose C provide a library for handling string manipulation i.e., library of string.h. Most of the string manipulation functions of C are part of this library. There is a function named *strcpy* which is used to copy a string to an array of characters (i.e., string variable). The syntax of **strcpy** is as follows:

```
char* strcpy(char* dest, const char* source);
```

Hence, the following statement will successfully copy the string to the variable *name*.

```
strcpy(name, "I love Pakistan");
```

14.8 STRING HANDLING IN TEXT FILES

When working with *text files*, C provides four functions which make file operations easier. The first two are called *fputs*() and *fgets*(), which write or read a string from a file, respectively. Their prototypes are:

```
    int fputs(char *str, FILE *fp)
    char *fgets(char *str, int num, FILE *fp)
```

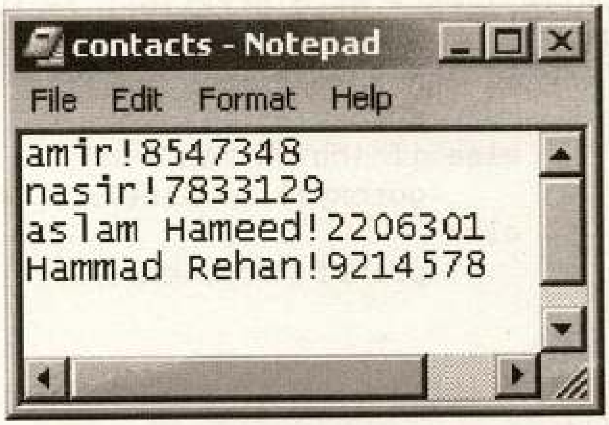
The *fputs*() function writes the string pointed to by *str* to the file associated with *fp*. It returns *EOF* if an error occurs and a non-negative value if successful. The null that terminates *str* is not written and it does not automatically append a carriage return/linefeed sequence.

The *fgets*() function reads string of characters from the file associated with *fp* into a string pointed to by *str* until num-1 characters have been read, a new line character (\n) is encountered, or the end of file (EOF) is encountered. The function returns *str* if successful and a *null pointer* if an error occurs.

Example 3 **Write a program that accepts name and telephone numbers of your friends and write them in a file.**

```
#include <stdio.h>
#include <string.h>

void main(void)
{
   FILE *ptrFile;
   char name[30];
   char tel[11];
   if ((ptrFile = fopen("d:\\Contacts.txt", "w")) ==
NULL)
   {
      printf("Can't open bfile.txt for writing!\n");
   }
   // If the file opened successfully, Get the name
   // and telephone number and store them in the file
   else
   {
      do
      {
         printf("Enter the name(or press ENTER to quit):
         ");
         gets(name);
         if (strlen(name) > 0 )
         {
            printf("Enter telephone number (max 10
            characters): ");
            gets(tel);
            // write name and telephone number to file
            fputs(name, ptrFile);
            fputs("!", ptrFile);
            fputs(tel, ptrFile);
            fputs("\n", ptrFile);
        }
    }while(strlen(name) > 0);
    // Close the files.
    fclose(ptrFile);
  }
}
```

This program demonstrates the typical use of strings in text files. A sentinel loop reads name and telephone numbers unless the user enters an empty string for the name. In addition to *fgets*() and *fputs*(), this program makes use of a new string handling function i.e., *gets*(). The *gets* function accepts a string from keyboard and assigned it to the variable *tel* (an array of characters). The contents of the file *contacts.txt* are as follows:



```
amir!8547348
nasir!7833129
aslam Hameed!2206301
Hammad Rehan!9214578
```

Fig. 14.3 Contents of Contacts.txt

Here an exclamation sign (!) separates the name and the telephone number fields in each record. We may use another symbol such as a colon (:), as a separator. In text files, a separator is used to mark the end of the data for one filed, whereas the data for the next field follow this separator.

Example 4 **Write a program that will read the contacts.txt file, and displays its contents on the screen.**

```
#include <stdio.h>
#include <conio.h>

void main(void)
{
     FILE* ptrFile;
     char ch;
     int line = 3;
     clrscr();

     if((ptrFile = fopen("d:\\contacts.txt", "r")) ==
     NULL)
          printf("can not open file");
     else
     {
          printf("Name");
          gotoxy(35,1);
          printf("Phone#\n");
          printf("-----------------------------------\n");
          while ((ch = getc(ptrFile)) != EOF)
          {
               if (ch == '!')
                    gotoxy(35, line);
               else if (ch == '\n')
                    gotoxy(1, ++line);
               else
                    printf("%c", ch);
          }
     }
     fclose(ptrFile);
     getch();
```

The function *gotoxy*() moves the cursor to a specified location on the screen. To use this function, the conio.h file must be included in the program. Its syntax is:

```
gotoxy(int col, int row)
```

The arguments of the gotoxy() function specify the coordinates of the screen where the cursor should move to.

Output: This program reads the file contacts.txt and displays its contents on the screen. The following is the output of the program:

```
NAME                                    Phone#
--------------------------------------------------------
amir                                    8547348
nasir                                   7833129
aslam Hameed                            2206301
Hammad Rehan                            9214578
```

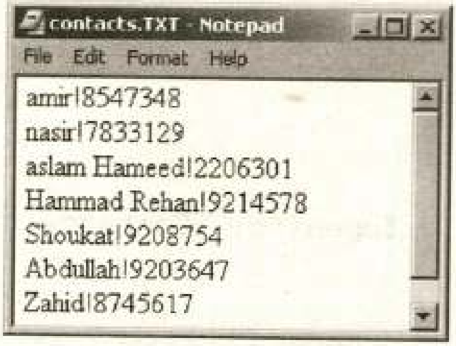
The process of *appending* a file is same as that of writing a file, just open the file in append mode. Consider the following example:

Example 5 **Write a program that will append records in contacts.txt file.**

```
#include <stdio.h>
#include <string.h>

void main(void)
{
   FILE *ptrFile;
   char name[30];
   char tel[11];
   if ((ptrFile = fopen("d:\\Contacts.txt", "a")) ==
NULL)
   {
      printf("Can't open bfile.txt for writing!\n");
   }
   // If the file opened successfully, get the name
   // and telephone number and append them in the file
   else
   {
      do
      {
         printf("Enter the name(or press ENTER to quit):
         ");
         gets(name);
```

```
        if (strlen(name) > 0 )
        {
            printf("Enter telephone number (max 10
            characters): ");
            gets(tel);
            // write name and telephone number to file
            fputs(name, ptrFile);
            fputs("!",ptrFile);
            fputs(tel, ptrFile);
            fputs("\n", ptrFile);
        }
    }while(strlen(name) > 0);
    // Close the files.
    fclose(ptrFile);
  }
}
```

This program seems very much similar to the program in example3 except that it opens *contacts.txt* in append mode, so new records are added at the end of the *contacts.txt* file. The following figure shows the contents of *contacts.txt* after appending three records:



```
amir!8547348
nasir!7833129
aslam Hameed!2206301
Hammad Rehan!9214578
Shoukat!9208754
Abdullah!9203647
Zahid!8745617
```

Fig. 14.4 Contents of Contacts.txt after adding three more records

14.9 FORMATTED I/O

The other two file handling functions to be covered are *fprintf*() and *fscanf*(). These functions operate exactly like *printf*() and *scanf*() except that they work with files. Their prototypes are:

```
    int fprintf(FILE *fp, char *control-string, ...)
    int fscanf(FILE *fp, char *control-string ...)
```

Instead of directing their I/O operations to the console, these functions operate on the file specified by *fp*. Otherwise their operations are the same as their console-based relatives. The advantages to *fprintf*() and *fscanf*() is that they make it very easy to write a wide variety of data to a file using a text format.

Example 6 Example3 can be re-written using formatted I/O as follows:

```
#include <stdio.h>
#include <string.h>
void main(void)
{
   FILE *ptrFile;
   char name[30];
   char tel[11];

   if ((ptrFile = fopen("d:\\contacts.txt", "w")) ==
NULL)
   {
     printf("Can't open bfile.txt for writing!\n");
   }
   // If the file opened successfully, Get the name
   // and telephone number and store them in the file
   else
   {
     do
     {
      printf("Enter the name(or press ENTER to quit):" );
        gets(name);
        if (strlen(name) > 0 )
        {
          printf("Enter telephone number (max 10
          characters): ");
          gets(tel);
          // write name and telephone number to file
          fprintf(ptrFile, "%s!%s\n", name, tel);
        }
     }while(strlen(name) > 0);
     // Close the file.
     fclose(ptrFile);
   }
}
```

Exercise 14c

1. Fill in the blanks:
   - (i) A ___________ can store text only.
   - (ii) EOF stands for ________________________.
   - (iii) The ____________function is used to open a file.
   - (iv) An opened file must be _______________ before terminating the program.
   - (v) A file opened in __________ mode can be read and appended.
   - (vi) A file pointer is a variable of type______________.
   - (vii) A pointer is a memory location whose contents points to __________ memory location.
   - (viii) In C, every valid string ends with a ____________.
   - (ix) A string is an ____________ of characters.
   - (x) The fopen() returns a __________, if it fails to open a file for some reason.

2. Choose the correct option:
   - (i) A file is stored in:
     - a) RAM b) hard disk
     - c) ROM d) cache
   - (ii) Which of the following mode open only an existing file for both reading and writing:
     - a) "w" b) "w+"
     - c) "r+" d) "a+"
   - (iii) Which of the following functions is used to write a string to a file?
     - a) puts() b) putc()
     - c) fputs() d) fgets()
   - (iv) On successfully closing a file, the fclose() returns:
     - a) NULL b) 0 (zero)
     - c) 1 (one) d) FILE pointer
   - (v) An array subscript should be:
     - a) int b) float
     - c) double d) an array

3. Write T for true and F for false statement.
   - (i) A picture can not be stored in a text file.
   - (ii) EOF marks the end of a string.
   - (iii) A null character marks the end of a text file.
   - (iv) Text files are stored in a FILE* (file pointer).
   - (v) The name of the array points to its first element.
   - (vi) Array subscript is used to access array elements.
   - (vii) An array of characters can store data of any data type.
   - (viii) A binary file is a group of contiguous memory locations.
   - (ix) C can handle text files only.
   - (x) When an existing file is opened in "w" mode, its contents are over-written.

4. Can a file be used for both input and output by the same program?

5. What is a stream? Illustrate the difference between text and binary streams.

6. How many modes are there for opening a file in C? Discuss characteristics of different file opening modes.

7. What is a file pointer? Briefly explain the concept.

8. Write a program to merge the contents of two text files.

9. Write a program that counts the total number of characters in a text file. [**Note:** consider the blank space a character]

10. Write a program that counts the number of words in a text files and display the count on the screen.