

• Class XII – Computer Science – Notes

• Practicals

• Chapter 01 : Data Basics

1.1 Overview

Q : 01-01-01 : Define Data, Information and Explain Operations on Data ?

Answer :

Data : Data is a collection of facts, figures and statistics - related to an object, that can be processed to produce a meaningful information.

Information : The manipulated and processed data is called information e.g., the percentage of students results. It is an output of a certain process.

Operations : Manipulation of data (after capturing from different sources) to achieve the required objectives and results. For this purpose, a software (program) is used to process raw data which is converted to meaningful information.

These are categorized into three basic activities :

Data Capturing : Data must be recorded or captured in some form before it can be processed.

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 & 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, called Reproduction.

1.2 Traditional File System

Q : 01-02-01 : Define and Explain Record and File ?

Answer :

Record : A collection of related fields (facts about something) treated as a single unit is called a record. Let us see one student's biographic information (record) :

Roll Number 13-3101
Student Name Muhammad Saleem
Class XI

File : A collection of related records treated as a single unit is called a file or a data set. Records of all the students together, make a file.

Q : 01-02-02 : Explain Types of File from Usage Point of View ?

Answer :

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.

Transaction File : Files in which data prior to the stage of processing is recorded. It may be temporary file, retained till the master file is updated.

Backup File : Permanent files, for the purpose of protection of vital data.

Q : 01-02-03 : Explain Types of File from Functional Point of View ?

Answer :

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 .cpp and the executable files as .exe.

Data Files : These files contain data and are created by the software being used. A few of these are : Word Processor .doc, .rtf (document), Spread Sheet .xls and .wks (worksheet), Video files .avi, .mpg etc.

Q : 01-02-04 : Explain Types of File from Storage (File Organization) Point of View ?

Answer :

File Organization (Storage Point of View) :

Sequential Files : Files are stored or created on the storage media in the order the records are entered i.e., one after another in the sequence.

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.

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

1.3 Databases

Q : 01-03-01 : Define & Explain Database ?

Answer :

Database : A database is a collection of logically related data sets or files. 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. The user of the database normally has following facilities :

- 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.

Q : 01-03-02 : Explain Database Objectives ?

Answer :

Data Integration : Information is coordinated from different files and operated on a single file.

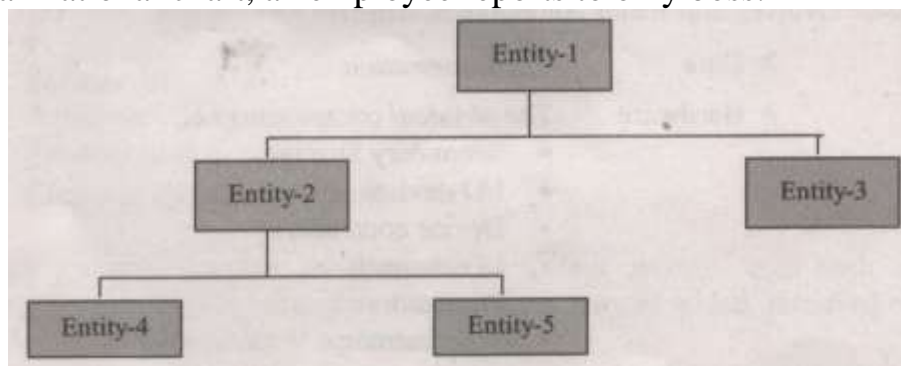
Data Integrity : If a data item is contained in more than one file, then all files must be updated if that item is changed.

Data Interdependence : When the format of a file is changed, then all the programs have to be changed. However, a database 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.

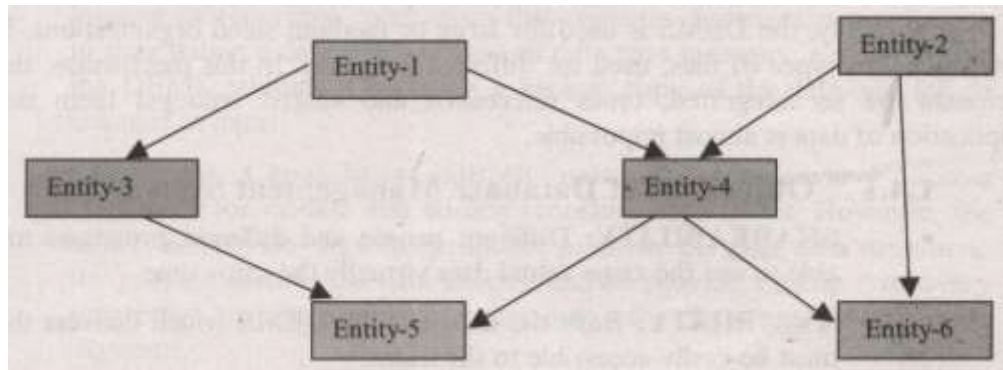
Q : 01-03-03 : Explain Various Types of Database Models ?

Answer :

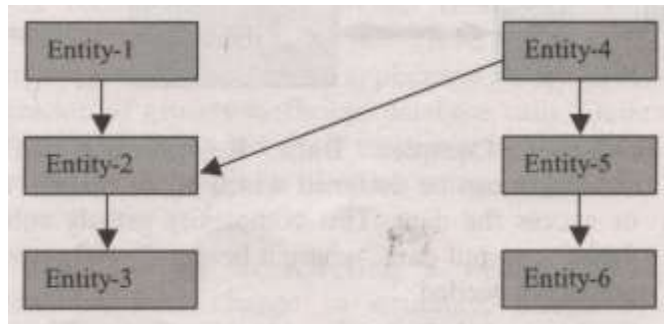
Hierarchical Model : This Model has the general shape or appearance of an Organizational Chart. A node on the chart, is subordinate at the next highest level, just as on an organizational chart, an employee reports to only boss.



Network Model : Similar to Hierarchical model but subordinate entities, depicted by arrows on the network diagram, may participate in as many subordinate relationships as desired.



Relational Model : Collection of simple files/Relations (Entities), each of which has no structural or physical connection like 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.



1.4 Database Management System

Q : 01-03-03 : Describe Database Management System (DBMS) and state Objectives of DBMS ?

Answer :

Database Management System (DBMS) : The data management system (a collection of programs) which is used for storing and manipulating databases is called database management system (DBMS). DBMS software (database manager) controls the overall structure of a database and access to the data itself.

Objectives of Database Management System (DBMS) :

Share Ability : Different people and processes must be able to use the same data at the same time.

Availability : Both the data and DBMS 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.

Q : 01-03-03 : Describe Advantages of Database Management System (DBMS) ?

Answer :

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 structures can be designed

which allow various ways to logically access the data.

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.

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.

Advanced Capabilities : DBMS normally have advance access capability for on-line and ad-hoc reporting capabilities.

Q : 01-03-03 : Describe Disadvantages of Database Management System (DBMS) ?

Answer :

Disadvantages of Database Systems :

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.

Problems can multiply in selecting a wrong type of Database Environment : A later change in structure, forced by changing requirements, can be costly in terms of conversion and testing of existing programs.

Data must be considered a Corporate Resource : The data in a company’s database no longer belong to one organization alone. One organization normally has the primary responsibility for creating a database, many can share the same data across applications.

A Need of a Dictionary : To share data across application systems, the internal data contents of a company’s databases need to be documented in a consistent manner, thus another overhead on the DBMS.

Q : 01-03-03 : Describe Features of Database Management System (DBMS) ?

Answer :

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.

• Chapter 02 : Basic Concepts & Terminology of Databases

2.1 Overview

Q : 02-01-01 : Define Field, Record and File ?

Answer :

Field : A field is a unit of data consisting of one or more characters i.e., Roll Number, Student Name, Grade etc.

Record : A collection of related data items treated as a single unit is called a record.

File : A collection of related records treated as a single unit is called a file or a data set.

2.2 Attributes, Rows and Tables

Q : 02-02-01 : Define Data Elements, Records, Files or Datasets and Databases ?

Answer :

Data Elements : The fields or data items in databases are termed as data items, items, attributes or columns in database structures.

Records : Records in file management structures are termed as rows or tuples in database structures.

Files or Datasets : Files or Datasets in databases are termed as tables relations or data objects in database structures.

Database : The collection of tables with some traditional files and some other necessary data objects is termed as a database.

2.3 Relation or Table

Q : 02-03-01 : Define Relation or Table and describe Entity ?

Answer :

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. Thus, the entities involved in this case are the same and the entity “student” can be defined in the form of database modeling :

STUDENT (STUDENT_NO, STUDENT_NAME, STUDENT_GENDER_CD, STUDENT_BIRTH_DATE, STUDENT_ADDRESS, STUDENT_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.

Q : 02-03-02 : Describe Properties of Relation ?

Answer :

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 (i.e. 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 nulls are inserted by the system at the time of column, which should immediately be replaced by zeros / spaces valid values for that particular column.

2.4 Views

Q : 02-04-01 : Describe Views ?

Answer :

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           STUDENT_NO, STUDENT_NAME,
STUDENT_ADDRESS
FROM             STUDENT
WHERE            STUDENT_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

Q : 02-05-01 : Describe Indexes ?

Answer :

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 database 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

Q : 02-06-01 : Define Key ? Describe Types of Keys ?

Answer :

Key : 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.

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. STUDENT_NO is the attribute that uniquely identifies each student and thus can be used as a Primary key. On the other hand, STUDENT_NAME 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. We can scan the records from the table using secondary key.

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.

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.

Sort / Control Key : A Sort / Control key is used to physically sequence the stored data 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.

Important 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 (RDBMS).

2.7 The User

Q : 02-07-01 : Define User ?

Answer :

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

Q : 02-08-01 : Define The Data Administrator ?

Answer :

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

Q : 02-09-01 : Define The Database Administrator ?

Answer :

The Database Administrator (DBA) : 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.

• Chapter 03 : Database Design Process

3.1 Overview

Q : 03-01-01 : Define Feasibility Study, Requirements Analysis, Project Planning and Data Analysis ?

Answer :

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 find 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, theft 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 :

Data Flow Diagrams (DFD)

Decision Tables

Decision Trees

3.2 Data Modeling

Q : 03-02-01 : Describe Ingredients of Data Modeling ?

Q : 03-02-01 : Describe Entities / Objects, Attributes , Relationships, Cardinality, Modality and ERD (Entity Relationship Diagram) ?

Answer :

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. Important to note are :

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).

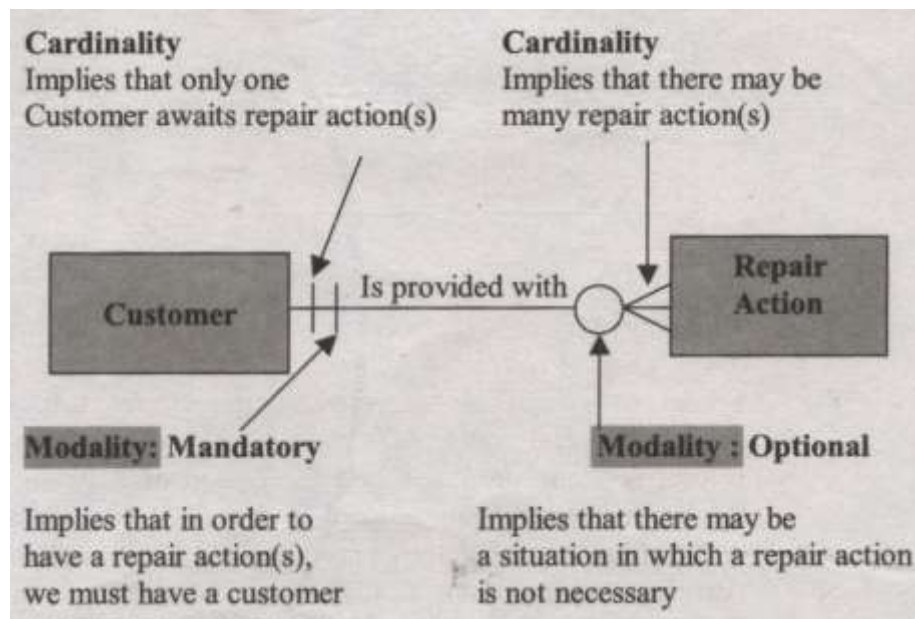
Cardinality : Whether some occurrence(s) of object-1 are related to some occurrence(s) of object-2. It is expressed as one or many. A relationships can be One to One, One to Many, Many to Many, Recursive and None.

Modality : It defines the nature of the relationship :

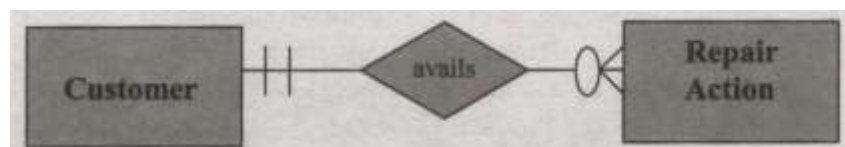
Optional represented by 0

Mandatory represented by 1

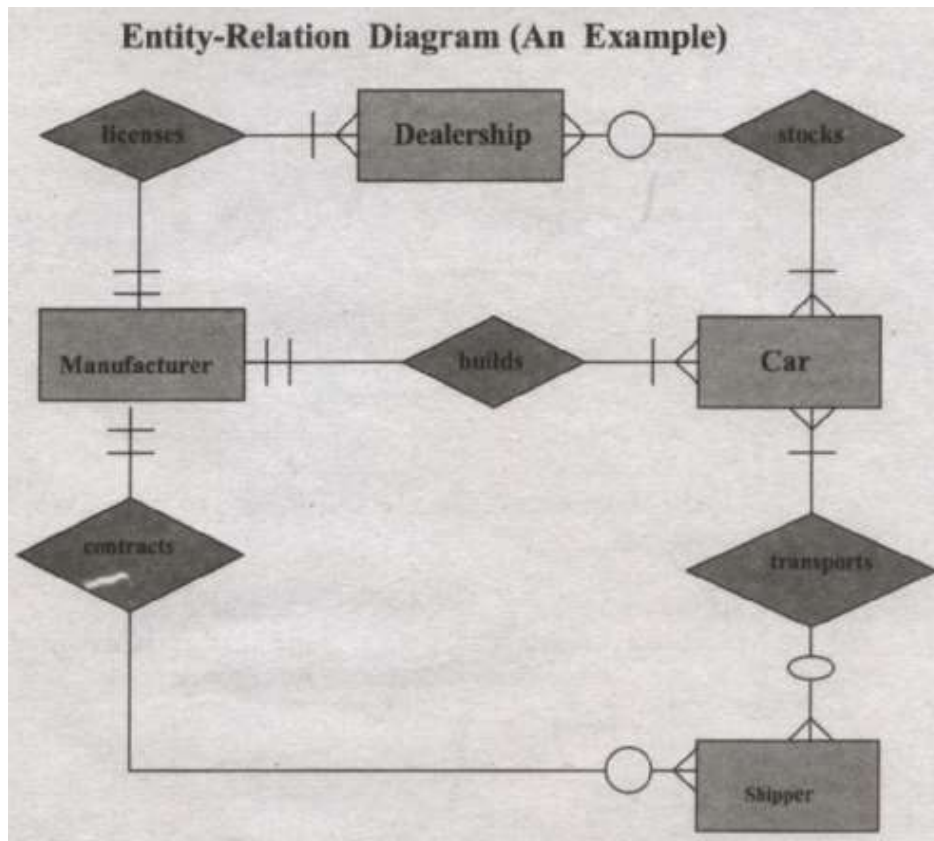
Consider two objects Customer and Repair Action in a Workshop environment :



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



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

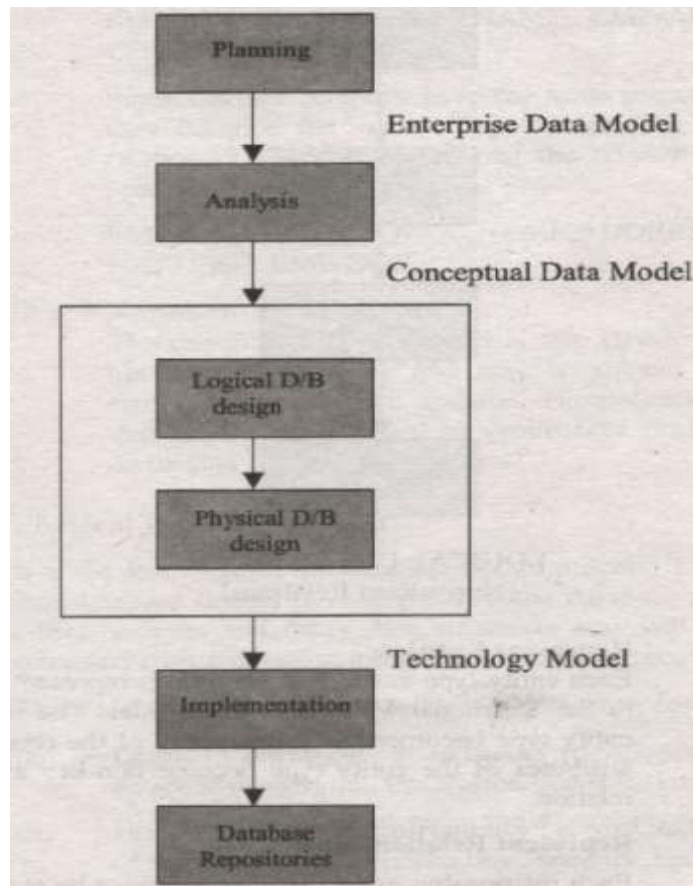


3.3 Database Design

Q : 03-03-01 : Describe major objective of Database Design ?
Explain Database Development process with diagram ?

Answer :

Major Objective of Database Design : [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]. Today, database users require information that is complete and up-to-date and they expect to be able to access this information quickly and easily. The Database Development process in diagram :

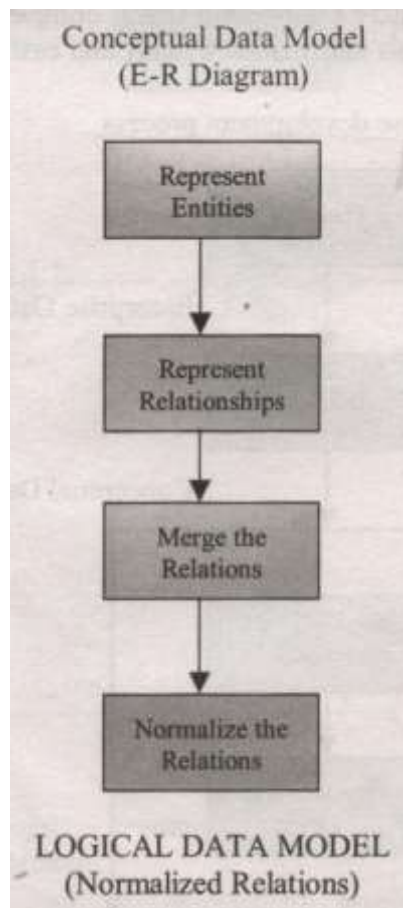


Q : 03-03-02 : Explain Conceptual (Logical) Database Design ?

Answer :

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. Diagram below presents an overview of logical design process :

Conceptual Data Model
(E-R Diagram)



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.

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.

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-JOBCODE, EMP-DOB)

Since the two relations have the same primary key 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, EMPJOB-CODE, EMP-DOB).

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.

Q : 03-03-03 : Explain Physical Database Design ?

Answer :

Physical Database Design : [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 :

Logical Database Structures : Developed during logical database design i.e., the Normalized Relations.

User Processing Requirements : The size and frequency of use of the Database, response time, security, backup, recovery etc.

Characteristics of the DBMS and other components of the computer Operating environment.

Q : 03-03-04 : Explain Components of Physical Database Design ?

Answer :

Components of Physical Database Design :

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.

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 are :

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.

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 accessible.

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.

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.

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.

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.

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.

3.4 Implementation

Q : 03-04-01 : Explain Database Implementation Phase ?

Answer :

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.

• Chapter 04 : Data Integrity and Normalization

4.1 Overview

Q : 04-01-01 : Explain Data Integrity ?

Answer :

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 :

Entity Integrity : It is a constraint on primary values that states that no attribute of a primary key should contain nulls.

Referential Integrity : It 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.

Q : 04-01-02 : Explain Normalization ?

Answer :

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

[Normalization is a technique for reviewing the entity / attribute lists to ensure that attributes are stored “where they belong to”. 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].

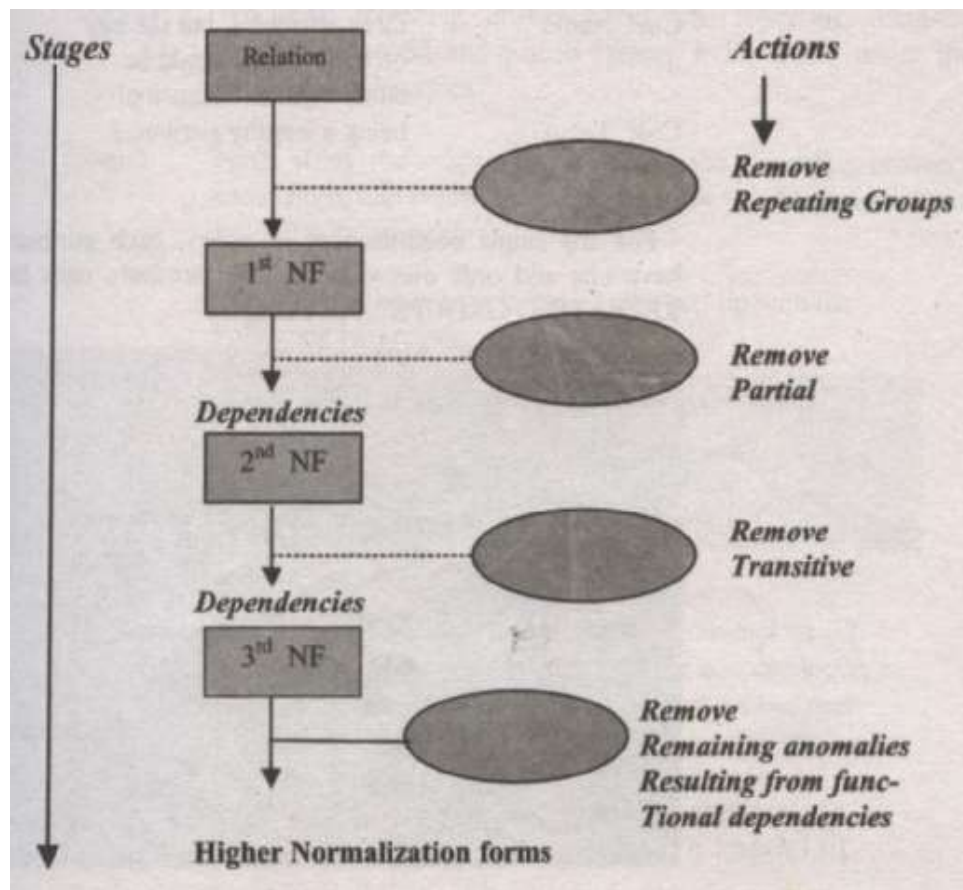
“A functional dependency is a particular relationship between two attributes. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B” The functional dependence of B on A is represented by an arrow. An attribute may be functionally dependent on two or more attributes rather than a single attribute. There may be some hidden problems as :

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.

Homonyms : A homonym is created when same name is used for Two different attributes. **Redundant Information :** Storing the same information in two different ways or forms if only one attribute can serve the purpose.

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.

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



Q : 04-01-03 : Define Normalization and Normal Form ?
Explain First Normal Form (1 NF) ?

Answer :

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

[Normalization is a technique for reviewing the entity / attribute lists to ensure that attributes are stored “where they belong to”. It is the basis for a relational data base system].

Normal Form : 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.

First Normal Form (1 NF) : “A relation R is in First Normal Form if and only if all underlying domains contain atomic values only”.

Relation : The Pre-Requisite is that “A relation has always a primary key associated with it”.

Unique Identification Key : All entities must have a key, composed of a combination of one or more attributes which uniquely identify one occurrence of the entity.

No Repeating Groups : For any single occurrence of an entity, each attribute must have one and only one value or “An attribute must have no REPEATING GROUPS”.

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 from, and where the Attribute went to. Determine if the From-To Relationship is 1:M or M:N.

<u>DEPARTMENT</u>	
Dept_No	
Dept_Name	
Emp_No	(error)
Emp_Name	(error)

<u>DEPARTMENT</u>	<u>EMPLOYEE</u>
Dept_No	Emp_No
Dept_Name	Emp_Name

Q : 04-01-04 : Explain Second Normal Form (2 NF) ?

Answer :

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

Second Definition : “To be in 2 NF, every non-key attribute must depend on the key and all parts of the key”.

Necessary & Sufficient Conditions : A table (relation) will be in 2NF if any of the following conditions apply :

The primary key consists of only one attribute.

No non-key attributes exist in the relation.

Every non-key attribute is functionally dependant on the full set of primary key attributes.

Consider table STUDENT in shorthand notation :

STUDENT(STUD-ID,NAME,DEPT,MONFEE,CRSNO,CDTE)

The functional dependencies in this relation are the as follows :

STUD-ID \longrightarrow NAME, DEPT, MONFEE

STUD-ID,CRSNO \longrightarrow 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.

Partial Functional Dependency : [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].

Anomalies : The partial functional dependency creates redundancy in the table, which results in certain anomalies when the table is updated :

Insertion Anomaly : To insert a row for the table, we must provide the values for both STUDENT-ID (Primary Key) and COURSE-NO (Not Primary Key).

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

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).

Important Note : 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, by splitting the relation, we will get two relations. This step is done to get rid of the redundant data. Anomalies are removed at the end of 2 NF.

Q : 04-01-05 : Explain Third Normal Form (3 NF) and Transitive Dependency ?

Answer :

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].

Transitive Dependency : [It is a functional dependency in a relation between two or more non-key attributes].

Third Normal Form (3 NF) – Second Definition : 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. The Anomalies, Insertion Anomaly, Deletion Anomaly and Modification Anomaly must be related with example data. These anomalies arise as a result of the transitive dependency. This problem (the transitive dependency) can be removed by de-composing the a relation into two relations.

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 \longrightarrow NAME, SALESMAN

(b) SALESMAN \longrightarrow Region (since each salesman is assigned a unique region).

Notice that SALES is in 2 NF, because the primary key 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.

The Anomalies

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

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)).

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

Modification Anomaly : If salesman Ahmad is 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 problem (the transitive dependency) can be removed by de-composing the relation SALES into two relations as shown below:

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

and

SALESMAN	REGION
Ahmad	South
Bashir	West
Khalid	East
Munir	North

SALE1 (CUSTNO, NAME, SALMAN)

SMAN (SALESMAN, REGION)

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

Note : 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.

• Chapter 05 : Introduction To Microsoft Access

5.1 Overview

Q : 05-01-01 : Describe MS Access ? Describe Benefits of MS Access ?

Answer :

Microsoft Access : 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 MS 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.

Benefits of Microsoft Access : MS-Access offers more than just pretty interface for learning how to manage data. Benefits of MS Access are :

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.

5.4 Database Objects

Q : 05-04-01 : Describe MS Access Database Components ?

Answer :

Microsoft Access Database Components : 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 :

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 relational database may contain multiple tables, which are identified by unique names. This is the fundamental property of a relational database.

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 analyze 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. The information appears in Datasheet 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 arid 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 :

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.

• Chapter 06 : Tables and Queries

6.1 Overview

Q : 06-01-01 : Describe MS Access ? Describe Benefits of MS Access ?

Answer :

Microsoft Access

In a relational database the data is stored in tables. The TABLE or RELATION is a fundamental concept of relational databases. It is foundation of every Relational Database Management System.

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.

Q : 06-01-02 : Describe Characteristics of Database Tables ?

Answer :

Characteristics of Tables

The tables of a relational database have following characteristics:

1. Each cell of the table contains only one value.
2. Each column has a distinct name, which is the name of the attribute (field) it represents.
3. The order of the columns is immaterial.
4. Each row represents a record.
5. Each row is distinct; there are no duplicate rows.
6. The order of rows is immaterial.
7. 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.

Q : 06-01-03 : Describe Degree and Cardinality of Relation or Table ?

Answer :

Degree of a Relation or Table

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

Cardinality of a Relation or Table

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.

Q : 06-01-04 : Define Basic Term used in Database ?

Answer :

A Basic Terminology

A **database** is a collection of related data (or record).

An **object** is a component in the database such as a table, 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.

Q : 06-01-05 : Describe MS Access IDE ?

Answer :

Access IDE

IDE stand for Integrated Development Environment. 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.

Q : 06-01-06 : Describe Data Types in MS Access ?

Answer :

Data Types in MS Access

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 fields.

Number — This data type is used to store numbers that are used in mathematical calculations. Several number field sizes are available.

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.

Yes / No - Use this option for True / False, Yes / No, On / Off, or other values that must be only one of two choices.

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 sub-address which is a location in the file or page in the web site; and the text that is displayed as the tool-tip.

Q : 06-01-07 : Describe Query in MS Access ?

Answer :

Query in MS Access

Query mean question or inquiry. 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 d 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.

Q : 06-01-08 : Explain Types of Query in MS Access ?

Answer :

Types of Queries

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 gr 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.

Crosstab 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 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.

Parameter Query

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

• Chapter 07 : Forms and Reports

7.1 Overview

Q : 07-01-01 : Describe Form in MS Access ?

Answer :

Microsoft Access Form

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, 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. 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 in the construction of forms.

Q : 07-01-02 : Describe Types of Form in MS Access ?

Answer :

Microsoft Access Form - Types

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 / sub form (combining the single-form and tabular formats into one form), and graph.

Q : 07-01-03 : Describe Sub Form in MS Access ?

Answer :

Microsoft Access – Sub Form

A sub-form is a form that is placed in a parent form, called the main form. Sub forms are particularly useful to display data from tables and queries that have one-to-many relationships.

Q : 07-01-04 : Describe Reports in MS Access ?

Answer :

Microsoft Access – Reports

Presentation of processed data obtained from a database is called report. The report can be displayed on the screen, printed on the paper or saved on the disk for later retrieval. The main uses of reports :

To display information obtained from a database.

To display result of a query.

To produce output according to the needs of the user.

Q : 07-01-05 : Describe Object Linking in MS Access ?

Answer :

Microsoft Access – Object Linking

Linking objects from another database will create a link to an object in another database while not copying the table to the current database. The latest object data will be retrieved every time we access the link.

• Chapter 08 : Getting Started With C

8.1 Overview

Q : 08-01-01 : Define Computer Program, High Level Programming Language and Low Level Programming Language ?

Answer :

Computer Program : 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.

High Level Programming Language : A computer program is written in a programming language. [A computer programming language that describes the set of statements / commands / instructions nearest / similar to written English language]. The examples are Pascal, Ada, Small Talk, C, C++, Java etc.

Low Level Programming Language : In starting years (1940s and 1950s) computer programs were written in machine language, this was very difficult and time taking for programmers. Later on, Assembly language was introduced that used pneumonics (understandable names for various instructions), and was thus comparatively easier and time saving for programmers.

Q : 08-01-02 : Describe C Language and give its short History ?

Answer :

C Language : Since the emergence of computer, many programming languages have been developed but the effect of C on the computer world is everlasting. The C programming language was developed by Dennis Ritchie in 1972 at AT&T Bell Laboratories.

History : 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. Over the years its power and flexibility have made it popular in industry for a wide 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.

Writing Program in C : 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 most commonly used is Turbo C++.

8.2 Developing A C Program (A Stepwise Approach)

Q : 08-02-01 : Explain C Program Development Process with Diagram ?

Answer :

C Program Development Process :

Turbo C++ (A Compiler for the C language) : Turbo C++ is a Borland International's implementation of a for C language. In addition to a compiler, TC provides a complete (Integrated Development Environment) to create, edit and save programs is called TC editor. 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 prompt or by double clicking the TC shortcut. The menu bar of the IDE contains menus to create, edit, compile, execute (Run) and debug 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.

Creating and Editing a C Program : Open the edit window of the Turbo C++ IDE, select File/New option from menu bar. This window has a double-lined border, and a cursor inside the window represents the starting point to write a program. We can expand this window by clicking the arrow in the upper right corner, or by selecting Windows / 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.

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. Type the name of the file in dialog box 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. 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. When a program is saved with .c extension, the Turbo C++ compiler restricts it to only use standard features of C.

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. 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.

Source Program : The program written in any high level programming language, such as C, is called source program.

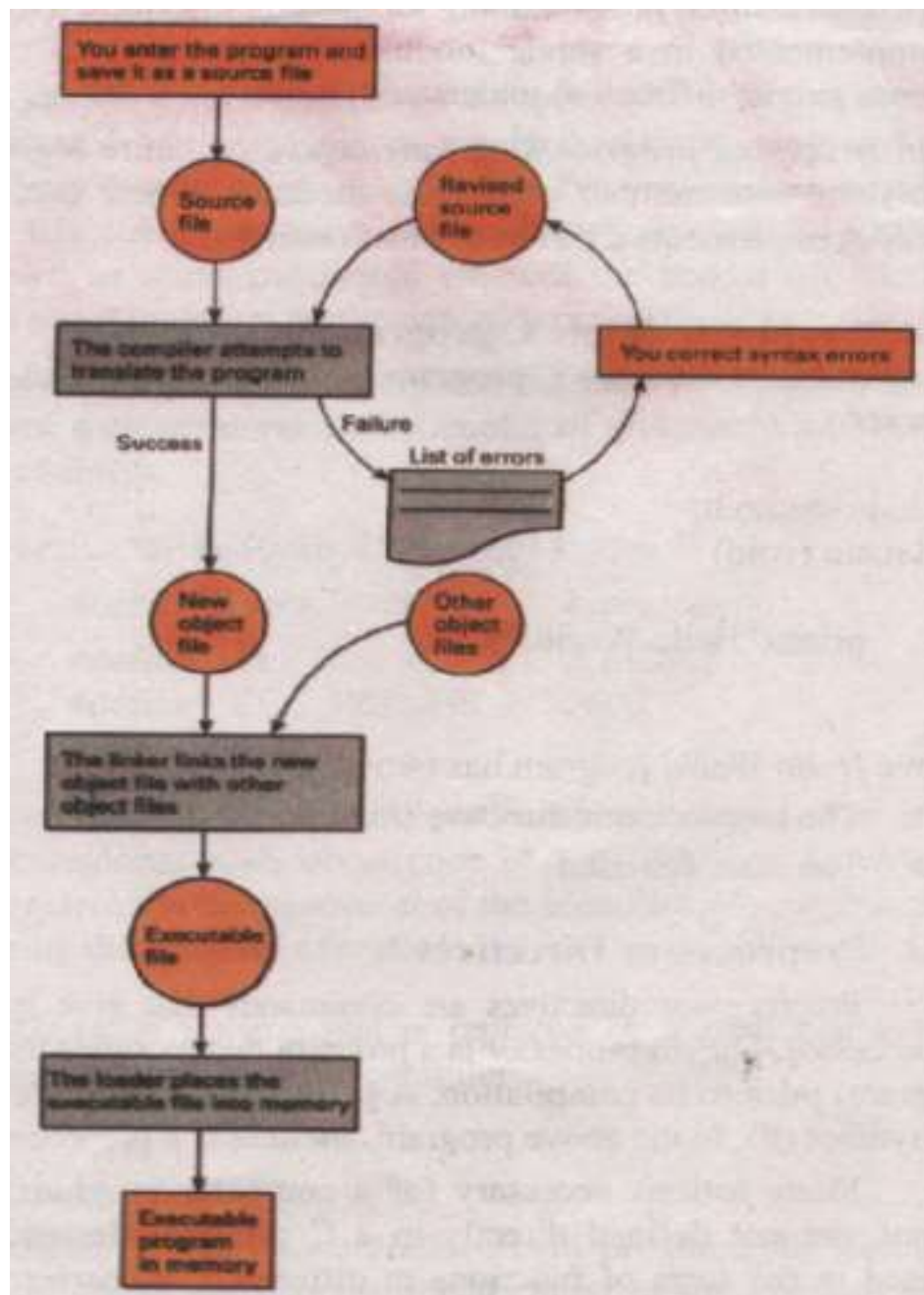
Object Program : The compiler produces an object program from the source program.

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.

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 field 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.



8.3 Basic Structure Of A C Program

Q : 08-03-01 : Describe Unstructured and Structured Programming Languages ? **Answer :**

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.

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.

Q : 08-03-02 : Briefly Explain C Program Structure with Example ? **Answer :**

C Program Structure : 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. 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. We understand the basic structure of the C program with the help of example :

Hello World - A simple C program : 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.

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 programs 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.

#define directive : Another important preprocessor directive is #define directive. It is used to define a constant macro.

#define Directive for Defining Constant Macros :

```
#define    Macro_Name          expression
#define    PI                  3.142857
#define    SECTOR_PER_HOUR    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 : It is a name that is replaced by a particular constant value before the program is sent to the compiler.

FUNCTION main : The 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 not necessary to divide every program into functions. The same functionality may be achieved through a single function. But, every C program must have the function main as the execution of the program starts from there. The main function is actually the entry point of the C programs.

main Function Definition :

```
SYNTAX    :    void main (void)
               {
                   body of main function.
               }
```

In algebra, 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. 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.

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.

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

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.

8.4 Common Program Errors

Q : 08-04-01 : Describe Common Programming Errors ? Briefly explain various types of Programming Errors ?

Answer :

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 :

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, i.e., missing statement terminator (semicolon), using a variable without declaration, missing any of the delimiters.

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.

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

Q : 08-05-01 : Describe the two broad categories of Programming Languages ?

Answer :

Programming Languages : These are used to write computer programs. There are two broad categories of programming languages :

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 words. 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.

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 conform 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.

Q : 08-05-02 : Explain Common Characteristics of High Level Programming Languages ?

Answer :

Common Characteristics of High Level Programming Languages

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 :

English Like Languages : These are English like languages, hence are close to human languages and far from the machine language and are very easy to learn.

Easy to Modify and Debug : Programs written in high level languages are easy to modify and debug, and more readable.

Concentrate on Problem : These languages let the Programmers concentrate on problem being solved rather than human-machine interaction.

Well Defined : These describe a well defined way of writing programs.

Understanding The Machine Architecture : These do not require a deep understanding of the machine architecture.

Machine Independence : 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.

• Chapter 09 : Elements of C

9.1 Overview

Q : 09-01-01 : Describe The Basic Elements (Building Blocks) of C Language ?

Answer :

The Basic Elements (Building Blocks) of C Language : 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. The proper use of basic elements (building blocks) of C language helps a lot to write effective C programs.

Identifiers : These 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 : 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 (stdio.h).

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.

Important Note : 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

Q : 09-02-01 : Describe Keywords ?

Answer :

Keywords : Keywords or reserved words are the words, which have predefined meanings 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 :

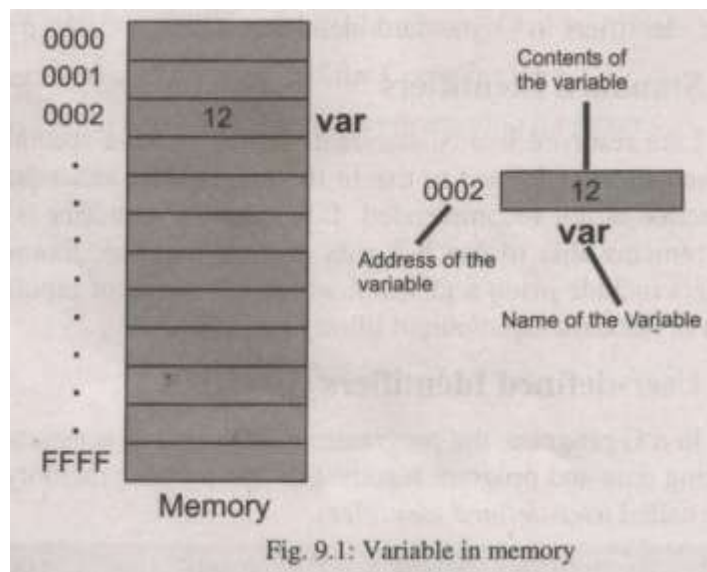
<u>auto</u>	<u>break</u>	<u>case</u>	<u>char</u>	<u>const</u>	<u>continue</u>	<u>default</u>
<u>do</u>						
<u>double</u>	<u>else</u>	<u>enum</u>	<u>extern</u>	<u>float</u>	<u>for</u>	<u>goto</u>
<u>if</u>						

int long register return short signed sizeof
static
struct switch typedef union unsigned void volatile
while

Q : 09-02-02 : Describe Variables ?

Answer :

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 address of the variable can be thought of as the address of that house.



Q : 09-02-03 : How do you declare Variables in C ?

Answer :

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 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 ;

Q : 09-02-04 : Differentiate between Declaring vs Defining a Variable in C ?

Answer :

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.

Important Note 1 : 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.

Important Note 2 : 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.

Q : 09-02-05 : Describe Initializing a Variable ?

Answer :

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 sets aside some memory space for it. This allocated memory space may contain data meaningless (garbage) to the program. The computations involving this variable may produce unexpected results. To avoid this situation, all variables should be declared and initialized 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 ;
```

Q : 09-02-06 : Describe Rules for Naming Variables in C ?

Answer :

Rules for Naming Variables in C : 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 :

Allowed Symbols / Characters : A variable name can consists of letters, digits, and the underscore character(_).

The First Symbol / Character : The first character of the name must be a letter or underscore, but use of underscore is not recommended. The second character onwards digits / numeric literals (0 .. 9) are also legal.

The Choices at First Character : A .. Z,
 a .. z,
 Underscore (_)

Total 53.
A .. Z,
a .. z,
Underscore (_)
0 .. 9
Total 63.

The Choices at Later Characters :

Thus 9winner, #home, and 2kgms are invalid names in C.

C is Case Sensitive Language : C is a Case Sensitive Language, thus, the names count, Count and COUNT refer to three different variables.

Keywords : C Language keywords can't be used as variable names e.g., we can not use int, void, signed, or while as variable names.

Length : For many compilers, a C variable name can be up to 31 characters long. (It can actually be longer than that i.e., 255 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 Space : Blank spaces are not allowed in the name e.g., (problem solving) is an invalid variable name in C.

Only One Data Type : A variable can only be declared for one data type.

Important Note : 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.

Q : 09-02-07 : Explain, Why Variable Names should be Readable ?

Answer :

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 as InterestRate or interestRate.

9.4 Constants

Q : 09-04-01 : Explain, Constants in C Language ?

Answer :

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.

The define directive can be used to define constant macros : #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 :

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.

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.

ASCII Character Set : The set of ASCII (American Standard Code for Information Interchange) Characters consists of control, printable and extended which is 0 to 255.

List of ASCII Codes

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
0	null	37	%	74	J	111	o	148	ö	185	¶	222	ü
1	start of heading	38	&	75	K	112	p	149	ò	186		223	ý
2	start of text	39	'	76	L	113	q	150	û	187	§	224	ÿ
3	end of text	40	(77	M	114	r	151	ù	188	¶	225	ÿ
4	end of transmission	41)	78	N	115	s	152	ÿ	189	¶	226	ÿ
5	inquiry	42	*	79	O	116	t	153	Ö	190	¶	227	ÿ
6	acknowledge	43	+	80	P	117	u	154	Ü	191	¶	228	ÿ
7	bell	44	,	81	Q	118	v	155	ø	192	¶	229	ÿ
8	backspace	45	-	82	R	119	w	156	£	193	¶	230	ÿ
9	horizontal tab	46	.	83	S	120	x	157	¥	194	¶	231	ÿ
10	line feed/new line	47	/	84	T	121	y	158	₣	195	¶	232	ÿ
11	vertical tab	48	0	85	U	122	z	159	ƒ	196	—	233	ÿ
12	form feed/new page	49	1	86	V	123	{	160	á	197	†	234	ÿ
13	carriage return	50	2	87	w	124		161	í	198	‡	235	ÿ
14	shift out	51	3	88	X	125	}	162	ó	199		236	ÿ
15	shift in	52	4	89	Y	126	~	163	ú	200	¶	237	ÿ
16	data link escape	53	5	90	Z	127	DEL	164	ñ	201	¶	238	ÿ
17	device control 1	54	6	91	[128	Ç	165	Ñ	202	¶	239	ÿ
18	device control 2	55	7	92	\	129	ü	166	ª	203	¶	240	ÿ
19	device control 3	56	8	93]	130	é	167	º	204	¶	241	ÿ
20	device control 4	57	9	94	^	131	â	168	¿	205	=	242	ÿ
21	negative acknowledge	58	:	95	_	132	ä	169	ƒ	206	¶	243	ÿ
22	synchronous idle	59	;	96	`	133	à	170	¬	207	±	244	ÿ
23	end of transmission block	60	<	97	a	134	á	171	½	208	¶	245	ÿ
24	cancel	61	=	98	b	135	ç	172	¼	209	¶	246	ÿ
25	end of medium	62	>	99	c	136	ê	173	¿	210	¶	247	ÿ

26	substitute	63	?	100	d	137	ë	174	«	211	ℓ	24
27	escape	64	@	101	e	138	è	175	»	212	Ô	24
28	file separator	65	A	102	f	139	ï	176	⋮	213	ƒ	25
29	group separator	66	B	103	g	140	î	177	⋮	214	π	25
30	record separator	67	C	104	h	141	ì	178	⋮	215	‡	25
31	unit separator	68	D	105	i	142	Ä	179		216	‡	25
32	space	69	E	106	j	143	Å	180	↓	217	↓	25
33	!	70	F	107	k	144	É	181	≡	218	└	25
34	"	71	G	108	l	145	æ	182	≡	219	■	25
35	#	72	H	109	m	146	Æ	183	π	220	■	25
36	\$	73	I	110	n	147	ô	184	ƒ	221	■	25

9.5 Data Type

Q : 09-05-01 : Explain, Data Types in C Language ?

Answer :

Data Types : 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 types 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 used in a program may be of different types.

Association : 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.

Pre-Defined and User-Defined Data Types : 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. The important standard data types are :

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.

Data Type	Other Name	Bytes	Range		Range	
Short	Short Int	2	-2^{15}	$2^{15} - 1$	-32768	
Unsigned Short	Unsigned Short Int	2	0	$2^{16} - 1$	0	
Int	Signed	2	-2^{15}	$2^{15} - 1$	-32768	

Unsigned Int	Unsigned	2	0	$2^{16} - 1$	0	
Long	Long Int	4	-2^{31}	$2^{31} - 1$	-2147483648	21474
Unsigned Long	Unsigned Long Int	4	0	$2^{32} - 1$	0	42949

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 65535). 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 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 2.45634×10^5 , where 2.45634 is the mantissa and 5 is the 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.

Data Type	Bytes	Precision	Range	
Float	4	6	10^{-38}	10^{38}
Double	8	15	10^{-308}	10^{308}
Long double	10	19	10^{-4932}	10^{4932}

Working with Floating Point Numbers : While working with floating point numbers, 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 0.000000 1243 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.

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. In C, a character is expressed as enclosed in apostrophes such as 'o', and 'u' etc. Like numbers, characters can also be compared, added and subtracted. Let's look at the following programs 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);
}
```

Program Output :

Sum = 104

```
#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =   '2'   ;
    ch2   =   '6'   ;
    sum   =   ch1   +   ch2   ;
    printf("Sum = %d", sum);
    printf("Sum = %c", sum);
}
```

Program Output :

Sum = 104 Sum = h

```
#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =   '2'   ;
    ch2   =   '6'   ;
    sum   =   ch1   +   ch2   ;
    printf("Sum = %d and Symbol = %c", sum,
sum) ;
}
```

Program Output :

Sum = 104 and Symbol = h

```

-----
#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    2    ;
    ch2   =    6    ;
    sum   =   ch1  +   ch2  ;
    printf("Sum = %d and Symbol = %c", sum,
sum) ;
}

```

Program Output :

Sum = 8 and Symbol =

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. 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 symbol ~). The other codes represent nonprintable control characters. 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.

9.6 Operators in C

Q : 09-06-01 : Explain, Data Types in C Language ?

Answer :

Operators : [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 :

Arithmetic Operators : Arithmetic operators are used to perform arithmetic operations values (numbers). The C language incorporates the following standard arithmetic operators :

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.

Operation	Symbol	Algebraic Expression	Expressions in C
Addition	+	$A + b$	$a + b$
Subtraction	-	$A - b$	$a - b$
Multiplication	*	$A \times b$	$a * b$
Division	/	A / b	a / b
Modulus	%	$a \text{ mod } b$	$a \% b$

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. The six basic relational operators in C are : 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)

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)

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 program. 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, multiplied and the result is assigned to the variable Area. $\text{Area} = \text{height} * \text{width}$;

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.

Important Note : Writing variable to the right side and the expression to the left side of

assignment operator will cause a syntax error.

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 become 11) and then the result will be assigned to the variable i. Hence, the variable j 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.

Compound Assignment Operators : The ++ and -- operators respectively increment and decrement the value of their operand by one. There are five 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 other operators in the same way.

<pre>j = 10; j *= 5;</pre> <p><u>After Execution of statements:</u></p> <ul style="list-style-type: none"> The value of j will be 50 	<pre>j = 10; j /= 5;</pre> <p><u>After Execution of statements:</u></p> <ul style="list-style-type: none"> The value of j will be 2
---	--

Operator Precedence : An operator's precedence determines its order of evaluation in an expression.

Operator	Precedence
	Highest
! (logical NOT)	
*, /, %	
+, -	
<, <=, >, >=	
==, !=	
&&	
= (assignment operator)	
	Lowest

Important Note : The logical NOT operator has the highest priority. It is 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

Q : 09-07-01 : Define and Explain Expression ?

Answer :

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.

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 operand. For example, consider the type of expression involving int or double type operands is of the form :

expr1 operator expr2

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

5 % 2 is 1

Working With Division Operator : The manipulation of division operation is slightly different from other arithmetic operations in C. One should be careful while dividing number, 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.

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.

```
a    =    5 * 0.5 ;
b    =    5 * 0.5 ;
```

Now a becomes 2.5 and b becomes 2.

9.8 Comments

Q : 09-07-01 : Explain importance, usage and style of Comments in C ?

Answer :

Comments : Comments are used to increase the readability of the program. With comments, informative notes are inserted in the program's code, which help in debugging and modifying the program. In C, there are two ways to comment the code.

Single-Line Comments : One can insert single line comments by typing two (forward) slashes (//) at the start of the note :

```
// This program calculates the factorial of the given number
```

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.

```
/*  
    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");  
}
```

Important Note : Comments are completely ignored by the compiler while generating object code.

• Chapter 10 : Input / Output

10.1 Overview

Q : 10-01-01 : Describe Standard Input / Output Library (stdio.h) in C Language ?

Answer :

Standard Input / Output Library : 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.

Q : 10-01-02 : Describe Standard Input / Output Library (stdio.h) Function printf ?

Answer :

Standard Input / Output Library Function printf : To see results of program execution, we must have a way to specify what variable values should be displayed. The standard I/O library (stdio.h) 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 format string is a character string - nothing more and the variables are optional.

The printf() function will take the form :

```
printf(format string, var1, var2, var3, ... );  
printf(format string);
```

The Signatures of a Function : Describes the number, order and type of its arguments, and the return type of the function (signature is identity).

Example : Write a program to calculate and print the area of a Rectangle.

```
#include<stdio.h>  
void main (void)  
{  
    int          height, width, area ;  
    height       = 5 ;  
    width        = 4 ;  
    area         = height * width ;  
    printf ("Area of Rectangle = %d", area) ;  
}
```

Program Outputs : Area of Rectangle = 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 Rectangle and the result is assigned to the variable area. Fifth and the last line of code is the printf() statement. This displays result on the screen. In case of printf, the first parameter is

always a string (e.g., "Area of Rectangle"), 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.

Q : 10-01-03 : Describe Format Specifiers in Standard Input / Output Library (stdio.h) Function printf ?

Answer :

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 :

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 : 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, var3 ;
    var1     = 24.27 ;
    var2     = 41.50 ;
    var3     = var1 + var2 ;
    printf (" %f + %f = %f", var1, var2, var3) ;
}
```

Program Outputs : 24.27 + 41.5 = 65.77

Q : 10-01-04 : Describe Field-Width Specifiers in Standard Input / Output Library (stdio.h) Function printf ?

Answer :

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 : 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 (least significant) on the screen to complete the length of 4. The output of the above statement will be : Area = □□25. □ 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.

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 : We need to 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.

Important Note : 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.nf, where m represents the total field width, and n represents the desired number of decimal places. 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.

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

Q : 10-01-05 : Explain Escape Sequences in Standard Input / Output Library (stdio.h) Function printf() ?

Answer :

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 meanings.

Example : Write a program that will demonstrate the use of escape sequences.

```
#include<stdio.h>
void main (void)
{
    printf ("Name\tRoll No\tMarks");
    printf ("\n-----");
    printf ("\nAmir\t    78\t425");
    printf ("\nTahir\t    23\t385");
}
```

Program Outputs :

Name	Roll No	Marks
Amir	78	425
Tahir	23	385

We used 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 new line and tab escape sequences; there are some others as well :

Escape Sequence	Purpose
<code>\n</code>	New Line
<code>\t</code>	Tab
<code>\b</code>	Backspace
<code>\r</code>	Carriage Return (Enter Key)
<code>\f</code>	Form feed
<code>\'</code>	Single Quote
<code>\"</code>	Double Quote
<code>\\</code>	Backslash
<code>\xdd</code>	ASCII code in hexadecimal notation (each d represents a digit)
<code>\ddd</code>	ASCII code in octal notation (each d represents a digit)

The escape sequence `\b` causes the cursor to move one space left, the form-feed (`\f`) moves to the next page on printer.

Important Note : 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. Statement : `printf` ("Escape Sequence is a \"Cool\" feature of C.");

The output is : Escape Sequence is a "Cool" feature of C.

10.2 Scanf Function

Q : 10-02-01 : Explain Standard Input / Output Library (stdio.h) Function scanf ?

Answer :

Standard Input / Output Library (stdio.h) Function scanf : Most of the programs are interactive in nature. 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 function `scanf` :

Example : Write a program to get the distance in kilometers from user, convert it into meters and display on the screen.

```
#include<stdio.h>
void main (void)
{
    double meter, kilometer ;
    // Ask the user to enter kilometers
    printf ("\nEnter distance in kilometers => ");
    // Take input
    scanf ("%lf", &kilometer);
    meter = kilometer * 1000 ;
```

```
printf (“\n%1f kilometers = %1f meters”, kilometer,
meter) ;
}
```

Program Outputs : Enter distance in kilometers => 90.13
90.130000 kilometers = 90130.000000 meters

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.

Important Note : Instead of the variable name, the scanf function requires address of the variable to store the input value into it. 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 function the address of the variable where the input value is to be stored. 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.



10.3 Character Input

Q : 10-03-01 : Explain Standard Library Functions for Character Input ?

Answer :

Standard Library Functions for 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. Function getch and getche are good examples. These are part of the conio.h (Console Input Output) library.

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 immediately returned by the function to the calling module.

Important Note : Character Constants are always specified within single quotations e.g, 'a', 'x' etc.

Example : Write a program that displays the ASCII code of the character typed by the user.

```
#include<stdio.h>
#include<conio.h>
void main (void)
{
    char ch ;
    // Ask the user to enter any key
    printf (“\nPlease Type a Character => ”) ;
    // Take input
    ch = getche() ;
    printf (“\nThe ASCII code for \'%c\' is %d”, ch, ch) ;
}
```

Program Outputs : Please Type a Character => g
The ASCII code for ‘g’ is 103

Important Note : 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 :

Program Outputs : Please Type a Character =>
The ASCII code for ‘g’ is 103

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 immediately 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 10

Q-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 : (πr^2) : area = PI x radius x radius.

where PI or π is the constant value of 3.14159. (Note: Define a constant macro PI with #define directive).

Answer :

```
#include<stdio.h>
#define PI 3.14159
void main (void)
{
    float radius ;
    double area ;
    // Ask the user to enter radius
    printf (“\nEnter Radius of Circle => ”) ;
    // Take input
    scanf (“%f”, &radius) ;
```

```

        area = PI * radius * radius ;
        printf ("\nArea of Circle = %1f", area) ;
    }

```

Q-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.

Answer :

```

#include<stdio.h>
void main (void)
{
    char ch1, ch2 ;
    float real_num ;
    int i ;
    // Ask the user to enter ch1 and ch2
    printf ("\nEnter First Character => ") ;
    scanf ("%c", &ch1) ;
    printf ("\nEnter Second Character => ") ;
    scanf ("%c", &ch2) ;
    printf ("\nEnter Real Number in Exponential Format
=> ") ;
    scanf ("%e", &real_num) ;
    i = 50 ;
    printf ("\nValue of Integer = %d", i) ;
}

```

Q-11. Write a program that converts a temperature in degrees Fahrenheit to degree Celsius. For conversion, use the formula : Celsius = 5/9 (Fahrenheit – 32).

Answer :

```

#include<stdio.h>
void main (void)
{
    float Celsius, Fahrenheit ;
    // Ask the user to enter Fahrenheit
    printf ("\nEnter Temperature in Fahrenheit => ") ;
    // Take input
    scanf ("%f", &Fahrenheit) ;
    Celsius = (5 / 9) * (Fahrenheit – 32) ;
    printf ("\n%.2f Fahrenheit = %.2f Celsius",
    Fahrenheit, Celsius) ;
}

```

Q-12. Write a program that takes a positive number with a fractional part and round it to two decimal places. For example, 25.4851 would round to 25.49, 62.4431 would round to 62.44.

Answer :

```
#include<stdio.h>
#include<math.h>
void main (void)
{
    float positive_real_num ;
    // Ask the user to enter a positive real number
    printf ("\nEnter a Positive Real Number => ");
    // Take input
    scanf ("%f", &positive_real_num) ;

    printf ("\n%.2f is a Positive Real Number",
    positive_real_num) ;
}
```

• Chapter 11 : Decision Constructs

11.1 Overview

Q : 11-01-01 : Describe the need and organization of Control Structures ?

Answer :

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. Only sequential flow, 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 :

```
{  
    Statement1 ;  
    Statement2 ;  
    .  
    .  
    .  
    Statementn ;  
}
```

Control flows from statement₁ to statement_n, in a logical sequence. 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. 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

Q : 11-02-01 : Describe the IF Statement ? Explain test condition with an example ?

Answer :

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.

Example :

```
#include<stdio.h>
void main (void)
{
    int age, status ;
    printf("Enter the age : ") ;
    scanf("%d", &age) ;
    status = (age > 60) ;
    printf("Status = %d", status) ;
}
```

Note : Value of the variable status will be 1 if the age is greater than 60, otherwise status will be 0.

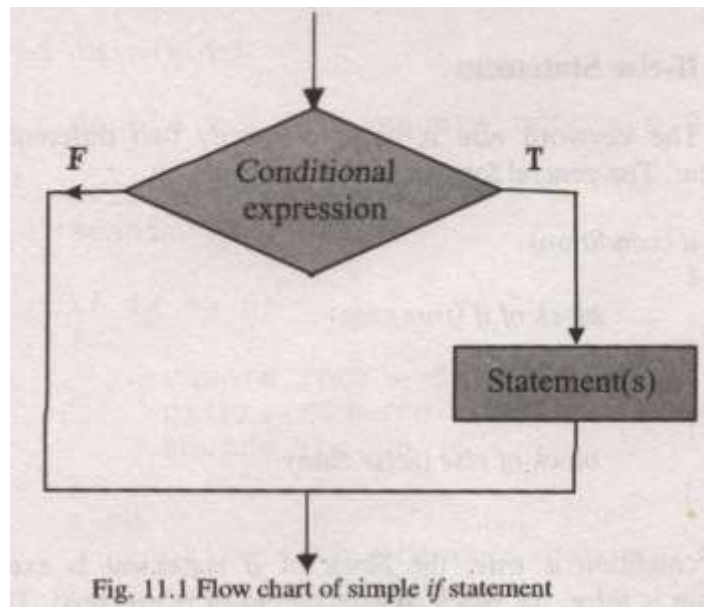
Q : 11-02-02 : Describe Simple IF Statement ? Explain it with diagram and an example ?

Answer :

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. This flow chart describes a situation where a simple if statement can be used.



Example : A program that calculates the square root of a number input by user.

```

#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 %f = %f", x, square_root) ;
    }
}
  
```

As the square root of negative numbers is imaginary, therefore the program finds the square root of positive numbers only. If a negative number is input, the program does nothing and terminates. We use a function `sqrt()` from math library. This includes the definition of `sqrt()` and many other mathematical functions.

Important Note : A flow chart is the pictorial representation of a program and its logic.

Q : 11-02-03 : Describe IF-ELSE Statement ? Explain it with diagram and an example ?

Answer :

IF-ELSE Statement : If statement is the simplest form of decision constructs. It allows a statement or a set of statements to be executed conditionally. The keyword `else` is used to specify two different choices with if statement. The general form of if-else statement is :

```

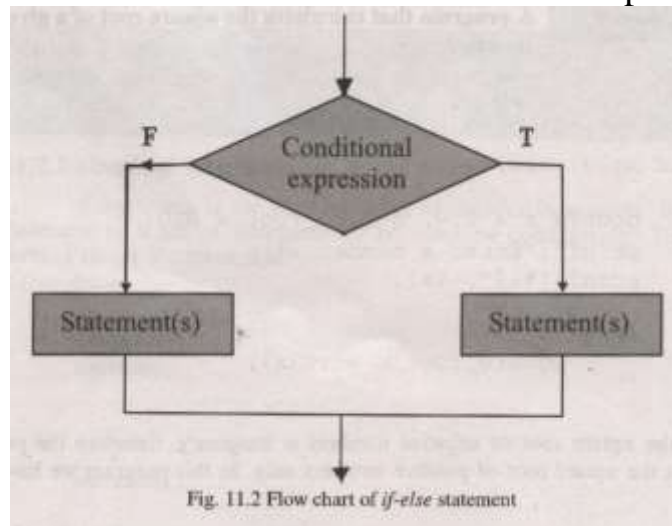
if (condition)
{
    Block of if (TRUE CASE)
}
  
```

```

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. This flow chart explains the idea :



Example : A program that calculates the square root of a number input by user using if-else.

```

#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 %f = %f", x, square_root) ;
    }
    else
        printf("Square Root of a Negative number can't 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 of a Negative number can't be calculated" will be displayed.

Important Note : 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 error message "Illegal else without matching if" will appear. 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.

Q : 11-02-04 : Describe Nested IF Statement ? Explain it with diagram and an example ?

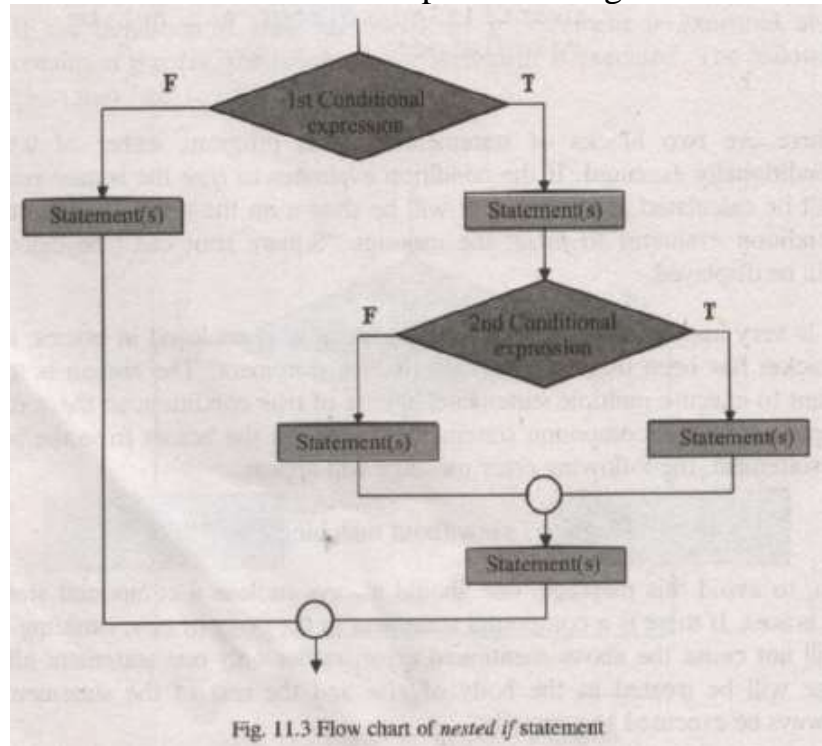
Answer :

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 (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 :

```
if (condition1)  
{  
    Block of Outer if Begins  
    if (condition2)  
    {  
        Block of INNER if  
    }  
    Block of Outer if Ends  
}
```

Important Note : 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.

Flowchart : This flow chart of nested if statement explains the logic.



Example : 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 the largest", a);
    else
        printf("%d is the largest", c);
}
else
{
    if (b > c)
        printf("%d is the largest", b);
    else
        printf("%d is the 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 contain other if statements (nested ifs are boxed). This sort of arrangement of multiple if statements is called nested-if statements.

Execution Sequence : 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 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 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, 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 greater than a (from the first condition), but b is smaller than c, therefore we conclude that c is the largest number. We can implement decision making in the program using statement.

Q : 11-02-05 : Compare Nested if and Sequence of ifs and explain with example ?

Answer :

Comparison of Nested if and Sequence of ifs : Due to the complexity of nested if statement, beginners usually prefer to use a sequence of if statements rather than a single nested statement. Sometimes, it is useful to use a nested if instead of sequence of ifs. 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.

Example : 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.

Q : 11-02-06 : Describe if-else-if Statement with data flow diagram and example ?

Answer :

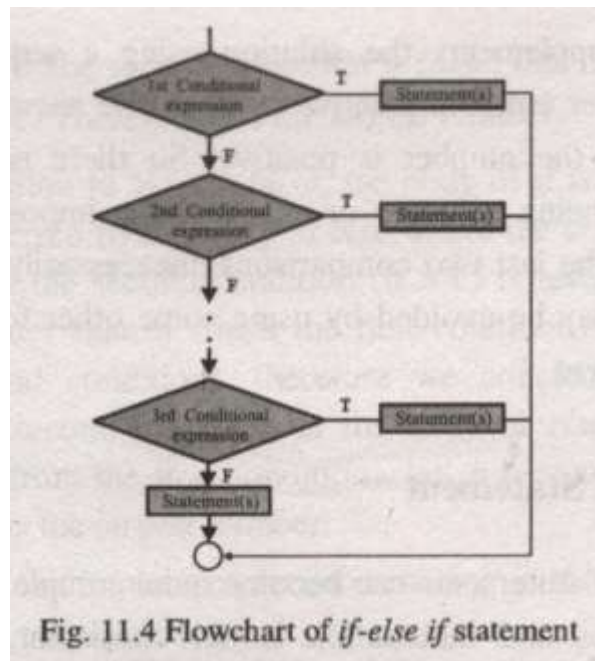
if-else-if Statement : Nested if statements can become quite complex, if there are more than three alternatives and indentation is not consistent, it may be difficult to determine the logical structure of the if statement. In such situations, if statement with multiple alternatives (if-else-if) can be a good option. The general form of if-else-if statement is :

```

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 skipped. If a condition is FALSE, the statement following it is skipped and the next condition is tested. If all conditions are FALSE, then statement_k following the last else is executed. **Flowchart :** It shows the execution flow of the program through if-else-if statement.



Example : A program that inputs a number from the user and determines whether it is positive, negative or zero. The program is re-written using if-else-if statement :

```

#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 if-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

Q : 11-03-01 : Illustrate the use of Logical Operators giving example ?

Answer :

Use of Logical Operators : Three logical operators AND, OR, and NOT play an important role in constructing certain compound conditions to be used with if statement. Now, we'll see how complex program logic can be simplified using logical operators :

Example : 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 && a > c)
        printf("%d is the largest", a);
    else if (b > a && b > c)
        printf("%d is the largest", b);
    else
        printf("%d is the largest", c);
}

```

There may be situations where the use of logical operators may simplify the program logic. We just need to concentrate on the underlying problem.

11.4 Switch Statement

Q : 11-04-01 : Illustrate the use of Switch Statement giving example ?

Answer :

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 :
        statementsk ;
}

```

The value of expression is compared to each case label. The statement 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 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 : Write a program that inputs a character from the user and checks whether it is a vowel or a consonant.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    char ch ;
    printf("Enter an ALPHABET => ");
    ch = getch( ) ;
    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's NOT a Vowel, It's a
CONSONANT") ;
            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.

Important Note 1 : The value of expression in switch statement must be of type int or char, but not of type float or double.

Important Note 2 : If the value of expression in switch statement is of float or double, the compiler will generate error message : “Switch selection expression must be of integral type”.

11.5 Conditional Operator

Q : 11-05-01 : Illustrate the use of Conditional Operator / Statement (ternary operator) giving example ?

Answer :

Conditional Operator / Statement : 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(void)
{
    int    a,    b ;
    printf(“Enter TWO numbers => ”) ;
    scanf(“%d %d”, &a, &b) ;
    a > b ? printf(“%d is larger”, a) : printf(“%d is larger”, b) ;
}
```

11.6 Locating a Point in The Coordinate Plane

Case Study : 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. 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 at 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’ step-by-step procedure to solve any problem in finite number of steps].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 output describing the position of the point in the coordinate 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 ($x > 0$ && $y > 0$), the point will lie in the 1st quadrant.

If x-coordinate is negative ($x < 0$) and y-coordinate is positive ($y > 0$), the point will lie in the 2nd quadrant.

If both x- and y-coordinates are negative ($x < 0$ && $y < 0$), the point will lie in the 3rd quadrant.

If x-coordinate is positive ($x > 0$) and y-coordinate is negative ($y < 0$) the point will lie in the 4th quadrant.

Program : A program that accepts TWO numbers x and y from the user and determines / displays the quadrant on which the point (x, y) lies.

```
#include<stdio.h>
void main(void)
{
    int    x,    y ;
    printf("Enter values of x and y coordinates => ");
    scanf("%d %d", &x, &y) ;
    if (x == 0)
    {
        if (y == 0)
            printf("The point lies on THE ORIGIN") ;
        else
            printf("The point lies on Y-AXIS") ;
    }
    else if (x > 0)
    {
        if (y == 0)
            printf("The point lies 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 11

Q-9. A year is a leap year if it is divisible by 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”.

Answer :

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    unsigned int year = 0 ;
    printf(“Enter The Year => ”) ;
    scanf(“%d”, &year) ;
    if (year % 4 == 0)
    {
        if (year % 100 == 0)
        {
            if (year % 400 == 0)
            {
                printf(“%d is a Leap Year”, year) ;
            }
            else
            {
                printf(“%d is a NOT Leap Year”, year) ;
            }
        }
        else
        {
            printf(“%d is a Leap Year”, year) ;
        }
    }
    else
    {
        printf(“%d is Not a Leap Year”, year) ;
    }
}
```

Q-10. Write a program that inputs temperature and displays a message according to the following data :

<u>Temperature</u>	<u>Message</u>
Greater Than 35	Hot Day
Between 25 and 35 (Both Inclusive)	Pleasant Day
Less Than 25	Cool Day

Answer :

```
#include<stdio.h>
#include<conio.h>
```

```

void main(void)
{
    int    temerature ;
    printf("Enter Temperature in Celsius => ");
    scanf("%d", &temerature) ;
    if (temperature > 35)
    {
        printf("%d Degrees Celsius ! Its a HOT Day", temperature) ;
    }
    else if (temperature >= 25)
    {
        printf("%d Degrees Celsius ! Its a PLEASANT Day",
            temperature) ;
    }
    else
    {
        printf("%d Degrees Celsius ! Its a COOL Day", temperature)
        ;
    }
}

```

Q-11. Write a program that inputs obtained marks of a student, calculates percentage (assume total marks are 1100 or may get input from user), and displays his / her grade. The grade should be calculated according to the following rules :

<u>Percentage</u>	<u>Grade</u>
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

Answer :

```

#include<stdio.h>
#include<conio.h>
void main(void)
{
    int    marks ;
    float percent ;
    printf("Enter Obtained Marks => ");
    scanf("%d", &marks) ;
    percent = (marks / 1100) * 100 ; // Assume Total Marks = 1100
    if (percent >= 80)
    {
        printf("A+") ;
    }
}

```

```

else if (percent >= 70)
{
    printf("A") ;
}
else if (percent >= 60)
{
    printf("B") ;
}
else if (percent >= 50)
{
    printf("C") ;
}
else if (percent >= 40)
{
    printf("D") ;
}
else if (percent >= 33)
{
    printf("E") ;
}
else
{
    printf("F") ;
}
}

```

Q-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 display the message "Wrong Choice". [Use switch statement to implement the solution].

Answer :

```

#include<stdio.h>
#include<conio.h>
void main(void)
{
    float  num1, num2, result ;
    float result2 ;
    char choice ;
    printf("Enter The Tow Numbers => ") ;
    scanf("%d %d", &num1, &num2) ;
    printf("1.  Add\n") ;
    printf("2.  Subtract\n") ;
    printf("3.  Multiply\n") ;
    printf("4.  Divide\n") ;
}

```

```

printf("Please Enter Your Choice From 1 To 4 ==> ");
choice = getche();
switch(choice)
{
    case '1':    result = num1 + num2 ;
                printf("\nAddition %d + %d = %d", num1,
                    num2, result) ;
                break ;
    case '2':    result = num1 - num2 ;
                printf("\nSubtraction %d - %d = %d", num1,
                    num2, result) ;
                break ;
    case '3':    result = num1 * num2 ;
                printf("\nMultiplication %d * %d = %d",
                    num1, num2, result) ;
                break ;
    case '4':    if (num1 > num2)
                {
                    Result2 = num1 / num2 ;
                    printf("\nDivision %d / %d = %f", num1,
                        num2, result2) ;
                }
                else
                {
                    Result2 = num2 / num1 ;
                    printf("\nDivision %d / %d = %d", num2,
                        num1, result2) ;
                }
                break ;
    default :    printf("\nIn-Valid Choice : Try 1 To 4") ;
} // end of switch block
} // end of main block

```

• Chapter 12 : Loop Constructs

12.1 Overview

Q : 12-01-01 : Describe Iteration and Loop ?

Answer :

Iteration or Loop : There are problems whose solution may require executing a statement or a set of statements repeatedly. 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. [Iteration is the third type of program control structure (sequence, selection, iteration), and the repetition of statements in a program is called a loop]. Loop control statements are while, do-while, and for.

12.2 While Statement

Q : 12-02-01 : Describe While Statement with Flowchart and example ?

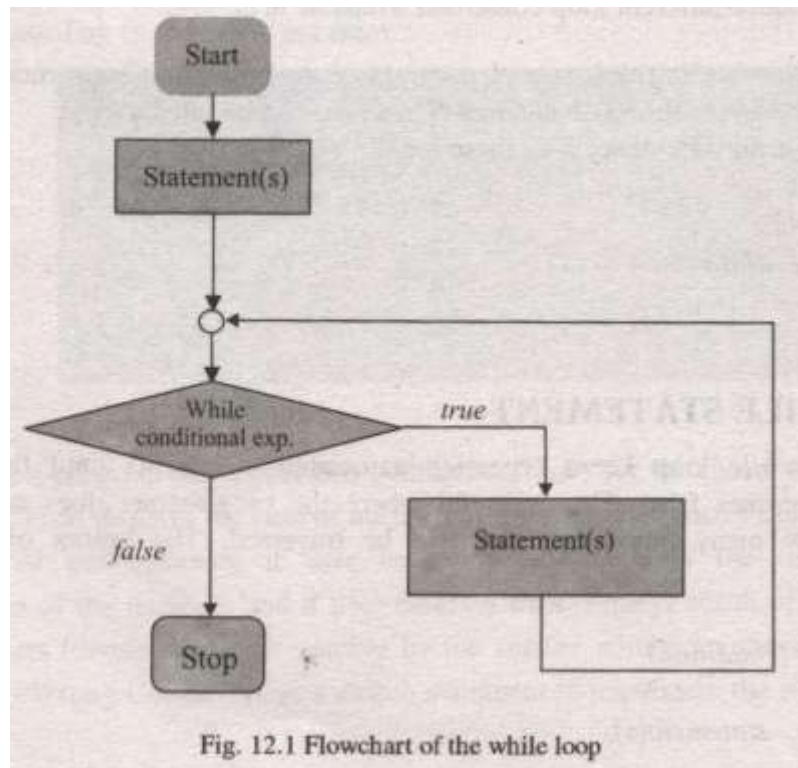
Answer :

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 before starting next iteration.

Flowchart of The While Loop :



Example : Write a program to print digits from 1 to 10 using while loop.

```

#include<stdio.h>
void main(void)
{
    int count ;
    count = 1 ;
    while (count <= 10)
    {
        printf("%d\n", count) ;
        count = count + 1 ;
    }
}

```

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 (next) 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 by 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 transfers

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.

Important Note : 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 loop and is incremented or decremented inside the loop body.

12.3 Do-While Loop

Q : 12-03-01 : Describe Do-While Statement with Flowchart and example ?

Answer :

While Statement / 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 while loop example.

Example 1 : Write a program to print digits from 1 to 10 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 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 working (W) or dead (D). 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. Force the user to enter a valid input value. This could be achieved by using a do-while loop.

```
#include<stdio.h>
void main(void)
{
    char  state ;
    do
    {
        printf("\nPlease Enter Current State Working (W) or Dead (D) => ");
        scanf("%c", &state);
    } while (state != 'w' && state != 'W' && state != 'd' && state != 'D');
    if (state == 'w' || state == 'W')
        printf("\nThe State of your phone is WORKING");
    else if (state == 'd' || state == 'D')
        printf("\nThe State of your phone is DEAD");
}
```

This program demonstrates a scenario where an invalid input is not processed, until the user enters a valid input (d or w or 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 != 'W' && state != 'd' && state != 'D'). It is a compound condition which is comprised of four sub conditions.

Important Note : 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, the user enters an invalid input (suppose e), all four sub conditions evaluate to true (because e is not equal to w, W, d or D). Therefore the compound condition also evaluates to true and the control flow returns back to the start of loop body. This process continues until the user enters a w or W or d or D, one of the sub conditions evaluates to false causing the compound condition to be evaluated to false and the control flow exits the loop body.

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.

12.3a FOR Loop – A Poetic Structure

Q : 12-03a-01 : Describe FOR Statement, its Execution Sequence, with flowchart and example ?

Answer :

FOR Statement / Loop : 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 :

```
for (initialization expression ; test condition ; increment / decrement
expression)
{
    statement(s) ; // loop body
}
```

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.

```
for (stmt1; stmt2 ; stmt3)
    stmt4 ;
```

Execution Sequence : The four statements / expressions are :

Stmt1 : Initialization Statement(s) (may or may not exist).

Stmt2 : Test / Boolean Expression (may or may not exist).

Stmt3 : Stepping Statement(s) (may or may not exist).

Stmt4 : Body Block / Body Statement (may or may not exist).

On Entry To The Loop, Sequence of EXECUTION is :

Step 1 : Stmt1(if exists) shall be executed ONLY ONCE.

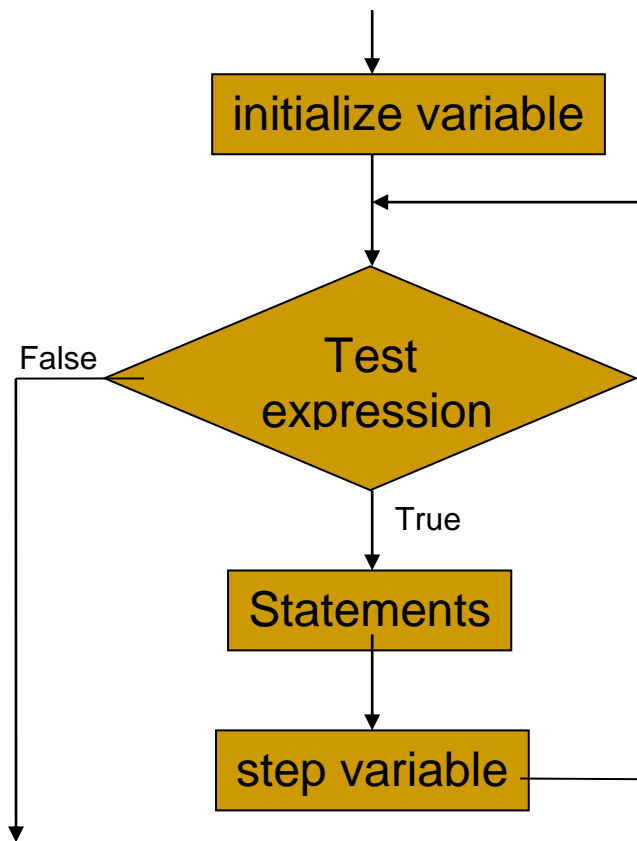
Step 2 : Stmt2(if exists) shall be evaluated for a BOOLEAN value, if ZERO go to Step 5.

Step 3 : Stmt4(if exists) shall be executed.

Step 4 : Stmt3(if exists) shall be executed and go to Step 2.

Step 5 : Exit THE “FOR” LOOP.

Important Note : If stmt3 is not changed / stepped, the loop continues for ever; such a loop is called infinite loop.



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 increment / decrement expression is evaluated. After evaluation “increment / decrement” expression, the test condition is checked again and if it is true then the body of the loop is 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.

Example : Write a program to print digits from 1 to 10 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. Any of these can be omitted but we must be aware of the consequences as : `for (; ;)` ; is a legal statement and will compile and run but will never terminate; this is called INFINITE LOOP.

12.4 Nested Loop

Q : 12-04-01 : Describe Nested Loop Statement with example ?

Answer :

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.

Example : Write a program that will print asterisks (*) according to the given pattern :

```
*****
*****
*****
****
***
**
*

#include<stdio.h>
void main(void)
{
    int    outer, inner ;
    for (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.

Q : 12-04-02 : Define Sentinel Value ? Write an example program to show data entry by a user to calculate average marks of a student ?

Answer :

Sentinel Value : [Sentinel Value is an end marker that follows the last item in a list of items]. Many programs require a list of items to be entered by the user. Often, 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, 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 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 :

Get the first line of data.

While the sentinel value has not been encountered.

Process the data line.

Get another line of data.

Example : Write a program to find the average marks of the students in a class.

```
#include<stdio.h>
```

```
void main (void)
```

```
{
```

```
    int sum = 0, marks = 0, 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("Enter 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 checked; 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 : $\text{average} = \text{sum} / (\text{float})\text{total_students}$; 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 (`float`) before the integer variable name (i.e. `total_students`) causes the 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

Q : 12-05-01 : Describe goto Statement ? Explain it with Example Program ?

Answer :

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 statement; in any other context, the labeled statement is executed without regard to the label. The general form of the goto statement is as :

```
goto label ;
label: statement ;
```

Example : Write a program to calculate the square root of a positive number (handle negative numbers properly).

```
#include<math.h>
#include<stdio.h>
void main (void)
{
```

```

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.

Important Note : Use of goto statement is not appreciated in C.

Exercise 12

Q-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”.

Answer :

```

#include<math.h>
#include<stdio.h>
void main (void)
{
    unsigned int num ;
    printf("Please Enter a Positive Number To Check
    Primality => ") ;
    scanf("%f", &num) ;
    if (num % 2 == 0)
    {
        printf("The Number %d is NOT Prime", num);
        exit(0);
    }
    else
    {
        for (int i = 3 ; i < sqrt(num) ; i += 2)
        {
            if (num % i == 0)
            {
                printf("The Number %d is NOT
                Prime", num);
                exit(0);
            }
        }
        printf("The Number %d is Prime", num);
    }
}

```


Q-10. Write a program that displays the first 15 even numbers.

Answer :

```
#include<stdio.h>
void main (void)
{
    for (int i = 1 ; i < 16 ; i ++)
    {
        printf("\n%d", i*2);
    }
}
```

Q-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
```

Answer :

```
#include<stdio.h>
void main (void)
{
    unsigned int num ;
    printf("Please Enter a Positive Number To Print Table
=> ") ;
    scanf("%f", &num) ;
    for (int i = 1 ; i < 11 ; i ++)
    {
        printf("\n%d x %d = %d", i, num, i * num) ;
    }
}
```

Q-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.

Answer :

```
#include<stdio.h>
void main (void)
{
    int num = 0 , total = 0 ;
    do
```

```

    {
        printf("\nPlease Enter a Number (0 To 15) =>
    ");
        scanf("%f", &num);
        total += num;
    } while (num < 0 || num > 15);
    printf("\nThe Total = %d", total);
}

```

Q-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

```

Answer :

```

#include<stdio.h>
void main (void)
{
    for (int i = 0 ; i < 10 ; i++)
    {
        printf("\n");
        for (int j = 0 ; j <= i ; j++)
            printf("\t%d", j);
    }
}

```

Q-14. Write a program the produces the following output :

```

0 1
1 2
2 4
3 8
4 16
5 32
6 64

```

Answer :

```

#include<math.h>
#include<stdio.h>
void main (void)
{
    for (int i = 0 ; i <= 6 ; i++)
        printf("\n\t %d \t %d", i, pow(2, i));
}

```

• Chapter 13 : Functions in C

13.1 Overview

Q : 13-01-01 : Describe Unstructured and Structured Programming ?
Define Function and differentiate between Unstructured and Structured Programming ?

Answer :

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 : 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.

Unstructured Programming : When the whole program logic is contained in a single main function, the style of writing programs is known as unstructured programming.

Structured Programming : 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.

Difference between Unstructured and Structured Programming : In Unstructured Programming, 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. Whereas in Structured Programming, the entire logic of the program is divided into number of smaller modules, where each module (piece of code) implements a different functionality.

13.2 Importance of Functions

Q : 13-02-01 : Describe Importance / Benefits of Use of Functions ?

Answer :

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.

Benefits of Functions : The use of functions provides several benefits :

Easier to Understand : 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.

Reusability 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.

Parallel Development of The Software : Different programmers working on one large project can divide the workload by writing different functions, hence ensuring the parallel development of the software.

Repeated Execution : Functions can be executed as many times as necessary from different places in the program.

Easy Debugging : When an error arises, rather than examining the whole program, the infected function is debugged only.

13.3 Types of Functions

Q : 13-03-01 : Explain the TWO Types of Functions ?

Answer :

Types of Functions : There are two types of functions in C :

Built-In 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 re-invent 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. Some of the built-in functions, e.g., stdio library and some of the functions defined in it, printf and scanf, which are defined in the library of standard input / output, similarly we have getch and getche functions which are defined in the library of console input / output.

Important Note : To use a Built-In Function in C, we need to include the header file defining the function. To use printf() and scanf(), we have to include stdio.h file in our program.

User-Defined Functions : Built-In Functions are not sufficient for solving every kind of problem. A programmer may need to write own functions depending on the nature of problem being solved. Such functions are called user-defined functions.

13.4 Writing Functions In C

Q : 13-04-01 : Explain the procedure of writing a Function in C ?

Answer :

Writing Functions In C : Every function in C has almost the same basic structure as that of the main() function. 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 :

```
return_type FunctionName (parameter_list) // number, order and types of
parameters
{
    Executable Statement (s) ;
```

```
        return expression ;  
    }
```

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 and the parameters of the function enclosed in parentheses.

The Return Type : 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 ()
```

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.

The return Statement : The return statement is used to specify the value returned 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

Q : 13-05-01 : Explain the Function Prototype in C ?

Answer :

Writing Functions In C : 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) ;
```

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.

Important Note : Function Prototype looks like Function Header but with a Semi Colon (;) at the end.

13.6 Calling A Function

Q : 13-06-01 : Explain the Function Call in C ?

Answer :

Function Call In C : 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

Q : 13-07-01 : Explain Local Variables in C, their Life Time, Memory Allocation / De-allocation and The Scope of a Local Variable with an example program ?

Answer :

Local Variables And Their Scope In C :

Life Time of The Variable : When the program executes, all variables are created in memory for a limited 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 life time of the variable.

Memory Allocation / De-allocation : Operating System (Memory Management Module) manages the allocation (reserve space in RAM) and de-allocation (free space in RAM) of memory for all variables in our program. Destroying a variable means returning the memory allocated to a variable back to the operating system for other programs. The value stored in such a variable is lost for ever.

The Scope of a Variable : 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.

Local Variables : All variables declared within a block (within the extent of a pair of curly braces) 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 :

```

#include<stdio.h>
void main (void)
{
    int nCount = 0 ;
    if (nCount == 0)
    {
        int chk ;
        chk = 10 ;
    }
    printf(“%d”, chk) ;
}

```

Explanation : Two variables are used 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

Q : 13-08-01 : Explain Global Variables in C, their Life Time, Memory Allocation / De-allocation and The Scope of a Local Variable with an example program ?

Answer :

Global Variables : The variables which are declared outside all blocks i.e. outside / above the main() and all other functions are called global variables and have global scope.

Global Scope : The global variables 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.

Life Time of Global Variables : The lifetime of global variables is from the start of the program till its termination. They exist in memory from the start to the end of the program.

Example :

```

#include<stdio.h>
#include<conio.h>
void main (void)

```

```

{
    int nCount = 1 ;
    clrscr( ) ; // a function of conio.h used to clear screen
    while (nCount <= 10)
    {
        int chk = 10 ;
        printf(“%d\t”, chk) ;
        chk = chk + 1 ;
        nCount++ ;
    }
}

```

Important Note : 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 re-initialized each time it is created. Thus the previous value will be lost for ever.

Explanation : In the example program above, each repetition of the loop prints the same value of the variable. The addition to the value of chk will have no effect, because at the end of execution 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 chk variable to be destroyed it 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.

Example :

```

#include<stdio.h>
#include<conio.h>
void Counter (void) ;
int nCount = 0 ;
void main (void)
{
    for (int n = 0 ; n <= 10 ; n += 2)
        Counter () ;
    printf(“nCount = %d”, nCount) ;
}
void Counter (void)
{
    nCount++ ;
}

```

Explanation : 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

Important Note : The variable nCount is declared outside the functions main() and Counter(), but these two functions can access and manipulate it as if it was declared within these. The nCount is created in the memory before the start of execution of main() and exists until the execution of the program ends.

13.9 Functions Without Arguments

Q : 13-09-01 : Explain Functions Without Arguments with an example program ?

Answer :

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.

Example : Write a function named Print_Asterisks that will print asterisks (*) according to the pattern shown and invoke a function call from the function main to print the asterisks.

```
*****
*****
****
***
**
*

#include<stdio.h>
void Print_Asterisks (void) ; // function prototype
void main(void)
{
    // Function call
    Print_Asterisks ( ) ;
}
void Print_Asterisks (void) // function header
{
    // Function Body Starts
    int inner ;
    for (int outer = 7 ; outer >= 1 ; outer--)
    {
        inner = 1 ;
        while ( inner <= outer)
        {
            printf(“*”) ;
            inner++ ;
        }
        printf (“\n”) ;
    }
    // Function Body Ends
}
```

Explanation : The next line to the `#include<stdio.h>` 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. 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 / Parameters

Q : 13-10-01 : Explain Functions that Return a Value and Accept Arguments / Parameters with example program(s) ?

Answer :

Functions that Return a Value and Accept Arguments / Parameters : We may need a function that could return a value and arguments could be passed to it. We have seen a number of such built-in library functions e.g., `sqrt()`, `toupper()`, `tolower()` etc. 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 comma separated list which specifies the data type and the name of each parameter in the list.

Example :

```
#include<stdio.h>
#include<conio.h>
int Add (int num1, int num2) ;
void main (void)
{
    int a, b ;
    int sum ;
    clrscr( ) ; // clears 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 num1, int num2)
```

```

    {
        return num1 + num2 ;
    }

```

Explanation : If we enter 12 and 15 for a and b respectively then the output of the of the program will be :

12 + 15 = 27

The 6th 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., int num1 and int num2) 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 function. 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 num1 and num2 respectively. The function Add() returns the sum of the two values to the main function which is then assigned to the variable sum.

Example :

```

#include<stdio.h>
#include<conio.h>
float Area_of_Triangle (int base, int altitude) ; // Function
Prototype
void main (void)
{
    int a, b ;
    float area ;
    clrscr( ) ; // clears the screen
    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 = %.2f", area) ;
}
float Area_of_Triangle (int base, int altitude) // Function
Header
{
    return (0.5 * base * altitude) ;
}

```

Explanation : Suppose we enter 25 and 45 for altitude and base respectively, then the program output will be :

Area of Triangle = 562.50

Exercise 13

Q-8. Write a program that calls 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.

Answer :

```
#include<math.h>
#include<stdio.h>
void Draw_Horizontal( ) ; // Function Prototype
void Draw_Vertical( ) ; // Function Prototype
void main (void)
{
    clrscr( ) ; / clears screen
    Draw_Horizontal( ) ; // draws upper horizontal line
    Draw_Vertical( ) ; // draws two vertical lines
    Draw_Horizontal( ) ; // draws lower horizontal line
}
void Draw_Horizontal( ) // Function Header
{
    // Go to new line and display 25 stars
    printf("\n");
    for (int i = 0 ; i < 25 ; i++)
        printf("*");
}
void Draw_Vertical( ) // Function Header
{
    for (int i = 0 ; i < 15 ; i++)
        printf("\n*                *") ; // 23 Spaces
        between stars
}
// We can write Draw_Horizontal( ) function like this also –
have only one
void Draw_Horizontal( ) // Function Header
{
    // Go to new line and display 25 stars
    printf("\n*****") ;
}
}
```

Q-9. Write a program that prompts the user for the Cartesian coordinates of two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ 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 :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Answer :

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```

float Distance(int x1, int y1, int x2, int y2) ;
void main (void)
{
    int x1, y1, x2, y2 ;
    float distance ;

    clrscr( ) ; // clears screen

    printf("\nEnter x-Coord of Point 1 => ") ;
    scanf("%f", &x1) ;
    printf("\nEnter y-Coord of Point 1 => ") ;
    scanf("%f", &y1) ;
    printf("\nEnter x-Coord of Point 2 => ") ;
    scanf("%f", &x2) ;
    printf("\nEnter y-Coord of Point 2 => ") ;
    scanf("%f", &y2) ;

    distance = Distance(x1, y1, x2, y2) ;
    printf("\nDistance Between P1 and P2 = %.2f",
distance) ;
}
float Distance(int x1, int y1, int x2, int y2)
{
    return (sqrt(((x2 - x1) * (x2 - x1)) + ((y2 - y1) * (y2 -
y1)))) ;
}

```

Q-10. Write a program that prompts the user to enter a number and then reverses 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.

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
int Reverse(int n) ;
void main (void)
{
    int a, b ;

    clrscr( ) ; // clears screen

    printf("\nEnter The Number To Reverse => ") ;
    scanf("%f", &a) ;
    b = Reverse(a) ;
    printf("\n%d Reversed = %d", a, b) ;
}
int Reverse(int n)
{

```

```

int negative = 0 ;
int first = 0, second = 0, third = 0, fourth = 0, fifth = 0
;
if (n < 0)
{
    negative = 1 ;
    n *= -1 ;
}
first      =    n % 10 ;
n          =    n / 10 ;
second     =    n % 10 ;
n          =    n / 10 ;
third      =    n % 10 ;
n          =    n / 10 ;
fourth     =    n % 10 ;
n          =    n / 10 ;
fifth      =    n % 10 ;
n          =    n / 10 ;

b          = fifth * 1 ;
b          += fourth * 10 ;
b          += third * 100 ;
b          += second * 1000 ;
b          += first * 10000 ;

if (negative = 1)
    b = b * -1 ;
return b ;
}

```

Q-11. Write a function named Draw_Asterisks that will print asterisks (*) according to the pattern shown and make a function call from the function main to print the asterisks pattern.

```

*****
*****
*****
***
*
#include<stdio.h>
void Draw_Asterisks (void) ; // function prototype
void main(void)
{
    // Function call
    Draw_Asterisks ( ) ;
}
void Draw_Asterisks (void) // function header
{
    // Function Body Starts

```

```

int inner ;
for (int outer = 9 ; outer >= 1 ; outer -= 2)
{
    inner = 1 ;
    while (inner <= outer)
    {
        printf("*") ;
        inner++ ;
    }
    printf ("\n") ;
}
// Function Body Ends
}

```

Q-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.

Answer :

```

#include<math.h>
#include<stdio.h>
int Is_Prime(int num) ;
void main (void)
{
    unsigned int a ;
    printf("Enter a Positive Number > 2 To Check
    Primality => ") ;
    scanf("%d", &a) ;
    if (a < 3)
        printf("In-appropriate Number %d to Check
        Primality", a) ;
    else
    {
        int result = Is_Prime(a) ;
        if (result == 0)
            printf("The Number %d is NOT Prime",
            a);
        else
            printf("The Number %d is Prime", a);
    }
}

int Is_Prime(int num)
{
    if (num % 2 == 0)
        return 0 ;
    else
    {
        for (int i = 3 ; i <= sqrt(num) ; i += 2)
            if (num % i == 0)

```

```

        return 0 ;
    return 1 ;
}
}

```

Q-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 divided (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 (terminate). Write four functions Add(), Subtract(), Multiply() and Divide() to complete the task.

Answer :

```

#include<stdio.h>
#include<conio.h>
void Add(int x, int y) ;
void Subtract(int x, int y) ;
void Multiply(int x, int y) ;
void Divide(int x, int y) ;
void main (void)
{
    int a, b ;
    char choice ;

    clrscr( ) ; // clears screen

    printf("\nEnter First Integer => ") ;
    scanf("%f", &a) ;
    printf("\nEnter Second Integer => ") ;
    scanf("%f", &b) ;
    printf("\nEnter Your Choice 1 .. 4 => ") ;
    scanf("%f", &choice) ;
    switch(choice)
    {
        case '1' : Add(a, b) ; break ;
        case '2' : Subtract(a, b) ; break ;
        case '3' : Multiply(a, b) ; break ;
        case '4' : Divide(a, b) ; break ;
        default : printf("\n%c is a Bad Choice",
            choice) ; break ;
    }
}

void Add(int x, int y) ;
{
    printf("%d + %d = %d", x+y) ;
}

void Subtract(int x, int y) ;
{

```



```

        printf(“%d - %d = %d”, x-y) ;
    }
void Multiply(int x, int y) ;
{
    printf(“%d * %d = %d”, x*y) ;
}
void Divide(int x, int y) ;
{
    if (x > y)
    {
        if (y == 0)
            printf(“Error – Divisor Can’t be ZERO”)
            ;
        else
            printf(“%d / %d = %.2f”, x / y) ;
    }
    else if (y > x)
    {
        if (x == 0)
            printf(“Error – Divisor Can’t be ZERO”)
            ;
        else
            printf(“%d / %d = %.2f”, y / x) ;
    }
    else
        printf(“Both Numbers are EQUAL thus result =
        1”)
    }
}

```

Q-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.

Answer :

```

#include<conio.h>
#include<stdio.h>
long int Factorial(int n) ;
void main (void)
{
    unsigned int a ;
    long int factorial ;
    printf(“Enter a Valid Number > 1 To Calculate
    Factorial => ”) ;
    scanf(“%f”, &a) ;
    if (a < 2)
        printf(“In-Valid Number < 2 To Calculate
        Factorial”) ;
    else

```

```

        {
            factorial = Factorial(a) ;
            printf(“%d Factorial = %ld”, a, factorial) ;
        }
    }
long int Factorial(int n)
{
    long int fact = 1 ;
    if (n == 1 || n == 0)
        fact = 1 ;
    else
    {
        for (int i = n ; i >= 2 ; i--)
            fact *= i ;
    }
    return fact ;
}

```

Q-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 calls the function GCD to compute the greatest common divisor of the numbers entered.

Answer :

```

#include<math.h>
#include<stdio.h>
int GCD(int m, int n) ;
void main (void)
{
    int a, b ;
    int gcd ;
    printf(“Enter TWO integers To Calculate GCD => ”) ;
    scanf(“%d %d”, &a, &b) ;
    if (a < 1 || b < 1)
        printf(“In-Valid Numbers To Calculate GCD”) ;
    else
    {
        gcd = GCD(a, b) ;
        printf(“GCD(%d, %d) = %d”, a, b, gcd) ;
    }
}
int GCD(int m, int n)
{
    int temp ;
    if (m > n)
    {
        if (m % n == 0)
            return n ;
    }
}

```

```
        temp = n ;
        n = m % n ;
        m = temp ;
    }
    else if (n > m)
    {
        if (n % m == 0)
            return m ;
        temp = m ;
        m = n % m ;
        n = temp ;
    }
    else
        return n ;
}
```

• Chapter 14 : File Handling in C

14.1 Overview

Q : 14-01-01 : Define permanent storage / file ?

Answer :

Permanent Storage / File : When we write programs to work with temporary data, the user has to enter data each time the program is executed. The data is stored on permanent storage in the form of files. [A file is a set of related records stored on a permanent storage medium].

14.2 The Stream

Q : 14-02-01 : Explain The Concept of Streams in C ?

Answer :

The Stream : C does not have any built-in method of performing file I/O, however the C standard library (stdio.h) 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 new line 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).

Important Note : In C, a file refers to a disk file, the screen, the keyboard, a port, a file on tape, etc.

14.3 NewLine and EOF Marker

Q : 14-03-01 : Explain NewLine and EOF Marker in C ?

Answer :

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 having ASCII Value 255 or xFF or all bits of the byte are 1s). 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. A specimen text file layout (Organization of text in a text file .txt) on storage is :

I		I	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

14.4 Opening a File

Q : 14-04-01 : Explain File Opening in C, its Modes, and File Pointer with an example program ?

Answer :

File Opening : 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 :

```
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 can, however be, other reasons for failure so don't assume that is what went wrong for certain. Example is :

```
FILE *fp ;
if ((fp = fopen("myfile", "r")) == NULL)
{
    printf("Error Opening File\n");
    exit(1);
}
```

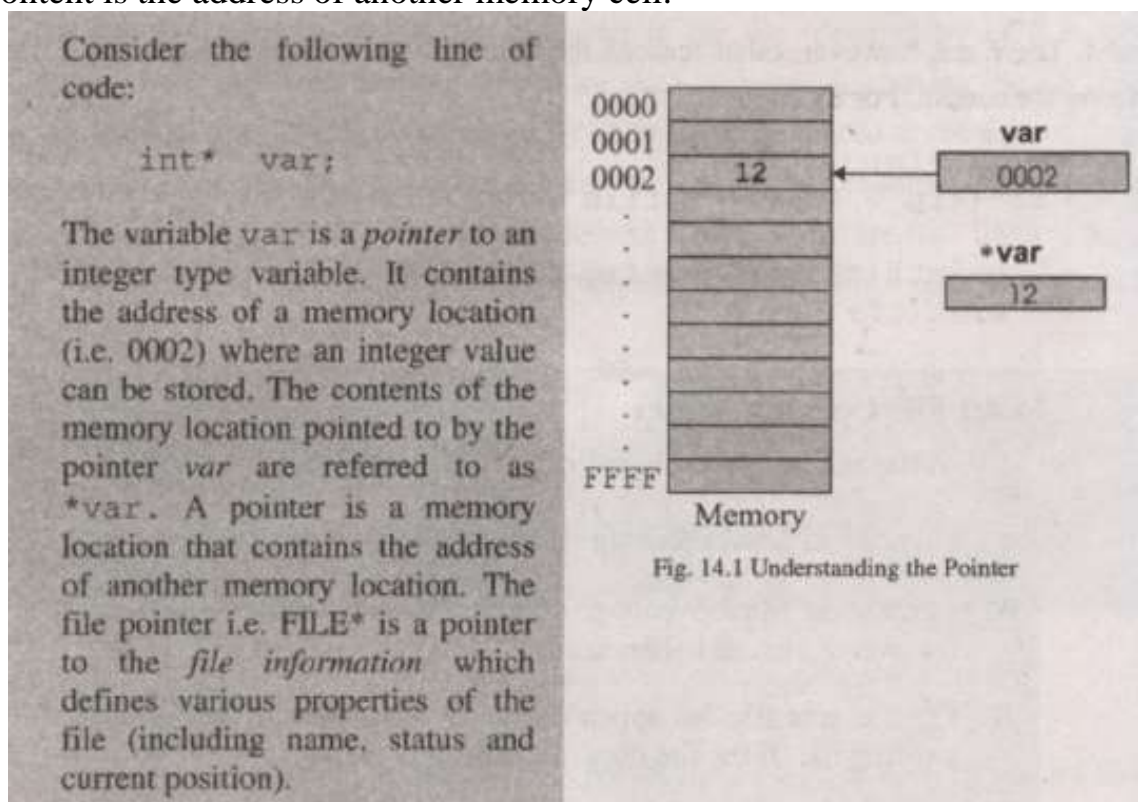
File Opening Modes : A file can be opened in following modes :

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.

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.



Example : Write a program to demonstrate the use of pointers :

```
#include<stdio.h>
#include<stdio.h>
void main (void)
{
    int* var ;
```

```

int num = 25 ;
clrscr( ) ;
var = &num ;
printf (“\nAddress 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.

14.5 Closing A File

Q : 14-05-01 : Explain File Closing in C ?

Answer :

File Closing : When a program has no further use of a file, it should close it with `fclose()` library function. The syntax of `fclose()` 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

Q : 14-06-01 : Explain the procedure for Reading and Writing Characters to a File in C ?

Answer :

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 : Write a program that reads a file and then writes its contents to another file (copy file).

```

#include<stdio.h>
void main (void)
{
    FILE *input ;
    FILE *output ;
    int ch ;
    // Try to open the input file. If it fails, print an ERROR
    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 file
    else if ((output = fopen("bfile.txt", "w") == NULL)
    {
        printf("Cant 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 both 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).

14.7 String Handling

Q : 14-07-01 : Explain String Handling (including declaration, initialization and assignment) in C ?

Answer :

String Handling in C : We display strings on screen with printf() function. 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. [C 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]. The general form of declaring arrays in C :

Data_type array_name[n]

The data_type specify the type of data that is stored in every memory location of the array, array_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). 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]    = 37.0 ;  
balls[1]    = 0 ;  temperature[2]    = 26.5 ;  
balls[4]    = 6 ;  temperature[3]    = 19.6 ;
```

Declaring and Initializing String Variables : 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 CityName[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	\32	L	O	v	e	\32	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 (space is ASCII 32).

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

Q : 14-08-01 : Explain String Handling in Text Files (including `fputs()` and `fgets()` functions) in C ?

Answer :

String Handling in Text Files in C : 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 : Write a program that accepts name and telephone numbers of your friends and write them in a file "Contacts.txt".

```
#include<stdio.h>
#include<string.h>
void main (void)
{
    FILE *ptrFile ;
    char name[30] ;
    char tel[12] ;
    if ((ptrFile = fopen("d:\\Contacts.txt", "w")) == NULL)
        printf("Can't open Contacts.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 file
    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
Hamad Rehan !9214578

```

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 field, whereas the data for the next field follow this separator.

Example : 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.

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.

Example : Write a program that will append records in `contacts.txt` file.

```

#include<stdio.h>
#include<conio.h>
void main (void)
{
    FILE* ptrFile ;
    char name[30] ;
    char tel [12] ;
    if ((ptrFile = fopen("d:\\Contacts.txt", "a")) == NULL)
        printf("Can't open Contacts.txt for writing / appending !\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 file
fclose (ptrFile) ;
}
}

```

Important Note : This program seems very much similar to the program in previous example except that it opens contacts.txt in append mode, so new records are added at the end of the contacts.txt file.

14.9 Formatted I/O

Q : 14-09-01 : Explain Formatted I/O in Files (including fprintf() and fscanf() functions) in C ?

Answer :

Formatted I/O in Files in C : 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 : Write a program (using Formatted I/O) that accepts name and telephone numbers of your friends and write them in a file "Contacts.txt".

```

#include<stdio.h>
#include<string.h>
void main (void)
{

```

```

FILE *ptrFile ;
char name[30] ;
char tel[12] ;
if ((ptrFile = fopen("d:\\Contacts.txt", "w")) == NULL)
    printf("Can't open Contacts.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) ;
    // this line replaces 4 lines commented below
    // fputs(name, ptrFile) ;
    // fputs("!", ptrFile) ;
    // fputs(tel, ptrFile) ;
    // fputs ("\n", ptrFile) ;
    }
    }while(strlen(name) > 0) ;
    // Close the file
    fclose (ptrFile) ;
    }
}

```

Exercise 14

Q-8. Write a program to merge the contents of two text files.

Answer :

Program Merge Files :

Important Note : Open Files a.txt for input and b.txt for appending. Read a.txt character by character and append each character at the end of b.txt (merge files).

```

#include<stdio.h>
void main (void)
{
    FILE *input ;
    FILE *appendoutput ;
    int ch ;

```

```

// Try to open the input file. If it fails, print an ERROR
message.
if ((input = fopen("a.txt", "r")) == NULL)
    printf("Can't open a.txt for reading ! \n") ;
// Now try to open the output file. If it fails, close the
input file
else if ((appendoutput = fopen("b.txt", "a") == NULL)
{
    printf("Cant open b.txt for merging / appending
! \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, appendoutput) ;
}
// Close both the files
fclose(input) ;
fclose(appendoutput) ;
}

```

Q-9. Write a program that counts the total number of characters in a text file. [Note: consider the blank. space a character]

Answer :

Program Count Characters in a File :

Important Note : Open File a.txt for input read it character by character and keep counting. At the end of the file print / display # of characters counted.

```

#include<stdio.h>
void main (void)
{
    FILE *input ;
    int ch, count = 0 ;
    // Try to open the input file. If it fails, print an ERROR
message.
    if ((input = fopen("a.txt", "r")) == NULL)
        printf("Can't open a.txt for reading/counting
characters! \n") ;
    else
    {
        while ((ch = getc(input)) != EOF) // count ch.
            count++ ;
    }
}

```

```

    }
    printf("\n # of Characters in a.txt = %d", count) ;
    // Close the file
    fclose(input) ;
}

```

Q-10. Write a program that counts the number of words in a text files and display the count on the screen.

Answer :

Program Count Words in a File :

Important Note : Open File a.txt for input read it character by character and keep counting spaces only that separate words from each other. At the end of the file print / display # of words counted.

```

#include<stdio.h>
void main (void)
{
    FILE *input ;
    int ch, count = 0 ;
    // Try to open the input file. If it fails, print an ERROR
    message.
    if ((input = fopen("a.txt", "r")) == NULL)
        printf("Can't open a.txt for reading/counting
        characters! \n") ;
    else
    {
        while ((ch = getc(input)) != EOF) // count ch.
            if (ch == ' ') count++ ;
    }
    count++ ;
    printf("\n # of Words in a.txt = %d", count) ;
    // Close the file
    fclose(input) ;
}

```


• Chapter 15 : C Language & Practical Programs

15.1 Year 2007 Onwards

/*

C Language Programs

Hello World : C program that displays the phrase Hello World ! on the screen.

Page : 136

*/

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
void main (void)
```

```
{
```

```
    printf("Hello World!");
```

```
}
```

/*

The above Hello World program has two parts :

The preprocessor directive : #include<stdio.h>

The main function.

*/

/*

#define directive : Another important preprocessor directive is #define directive. It is used to define a constant macro.

#define Directive for Defining Constant Macros :

```
#define Macro_Name expression
```

```
#define PI 3.142857
```

```
#define SECTOR_PER_HOUR 3600
```

*/

/*

In C, a character is expressed as enclosed in apostrophes such a 'o', and 'u' etc.

Like numbers, characters can also be compared, added and subtracted. Let's look at the following programs to understand the concept :

Page : 151

*/

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
void main(void)
```

```
{
```

```
    char ch1, ch2, sum ;
```

```
    ch1 = '2' ;
```

```

        ch2  =    '6'    ;
        sum  =    ch1    +    ch2    ;
        printf("Sum = %d", sum);
    }
    /*

```

Program Output :

Sum = 104

Note : Output is not 8 as you may expect.

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    '2'    ;
    ch2   =    '6'    ;
    sum   =    ch1    +    ch2    ;
    printf("Sum = %d", sum);
    printf(" Sum = %c", sum)      ;
}
/*

```

Program Output :

Sum = 104 Sum = h

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    '2'    ;
    ch2   =    '6'    ;
    sum   =    ch1    +    ch2    ;
    printf("Sum = %d and Symbol = %c", sum, sum)  ;
}
/*

```

Program Output :

Sum = 104 and Symbol = h

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    2    ;

```

```

        ch2    =    6        ;
        sum    =    ch1    +    ch2    ;
        printf("Sum = %d and Symbol = %c", sum, sum)    ;
    }
    /*

```

Program Output :

Sum = 8 and Symbol =

```

    */
    /*

```

Page : 159

This program prints a single line message.

Author : Student

Year : 2015

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    // Following line prints a message
    printf("This is my first program");
}
    /*

```

Write a program to calculate and print the area of a Rectangle.

Page : 164

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int            height, width, area ;
    height         = 5 ;
    width = 4 ;
    area           = height * width ;
    printf ("Area of Rectangle = %d", area) ;
}
    /*

```

Program Output : Area of Rectangle = 20

```

    */
    /*

```

Write a program that adds two floating point numbers and shows their sum on the screen.

Page : 165

```

    */
#include<stdio.h>
#include<conio.h>

```

```
#include<math.h>
void main (void)
{
    float        var1, var2, var3 ;
    var1          = 24.27 ;
    var2          = 41.50 ;
    var3          = var1 + var2 ;
    printf ("%f + %f = %f", var1, var2, var3) ;
}
/*
```

Program Outputs : 24.27 + 41.5 = 65.77

*/

/*

Write a program that will demonstrate the use of escape sequences.

Page : 167

*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    printf ("Name\tRoll No\tMarks") ;
    printf ("\n-----") ;
    printf ("\nAmir\t\t 78\t425") ;
    printf ("\nTahir\t\t 23\t385") ;
}

/*
```

Write a program to get the distance in kilometers from user, convert it into meters and display on the screen.

Page : 169

*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    double meter, kilometer ;
    // Ask the user to enter kilometers
    printf ("\nEnter distance in kilometers => ") ;
    // Take input
    scanf ("%lf", &kilometer) ;
    meter = kilometer * 1000 ;
    printf ("\n%lf kilometers = %lf meters", kilometer, meter) ;
}
/*
```

Program Outputs : Enter distance in kilometers => 90.13

90.130000 kilometers = 90130.000000 meters

*/

/*

Write a program that displays the ASCII code of the character typed by the user.

Page : 171

*/

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
void main (void)
```

```
{
```

```
    char  ch ;
```

```
    // Ask the user to enter any key
```

```
    printf ("\nPlease Type a Character => ") ;
```

```
    // Take input
```

```
    ch    = getche() ;
```

```
    printf ("\nThe ASCII code for \'%c\' is %d", ch, ch) ;
```

```
}
```

/*

Program Outputs : Please Type a Character => g
 The ASCII code for 'g' is 103

*/

/*

Q : 10:09 : 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 : (πr^2) : area = PI x radius x radius.

where PI or π is the constant value of 3.14159. (Note: Define a constant macro PI with #define directive).

Page : 174

*/

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
#define PI 3.14159
```

```
void main (void)
```

```
{
```

```
    float radius ;
```

```
    double area ;
```

```
    // Ask the user to enter radius
```

```
    printf ("\nEnter Radius of Circle => ") ;
```

```
    // Take input
```

```
    scanf ("%f", &radius) ;
```

```
    area = PI * radius * radius ;
```

```
    printf ("\nArea of Circle = %1f", area) ;
```

```
}
```

/*

Q : 10:10 : 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.

Page : 174

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    char ch1, ch2 ;
    float real_num ;
    int i ;
    // Ask the user to enter ch1 and ch2
    printf ("\nEnter First Character => ") ;
    scanf ("%c", &ch1) ;
    printf ("\nEnter Second Character => ") ;
    scanf ("%c", &ch2) ;
    printf ("\nEnter Real Number in Exponential Format => ") ;
    scanf ("%e", &real_num) ;
    i = 50 ;
    printf ("\nValue of Integer = %d", i) ;
}
/*

```

Q : 10:11 : Write a program that converts a temperature in degrees Fahrenheit to degree Celsius. For conversion, use the formula : $Celsius = 5/9 (Fahrenheit - 32)$.

Page : 174

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float Celsius, Fahrenheit ;
    // Ask the user to enter Fahrenheit
    printf ("\nEnter Temperature in Fahrenheit => ") ;
    // Take input
    scanf ("%f", &Fahrenheit) ;
    Celsius = (5 / 9) * (Fahrenheit - 32) ;
    printf ("\n%.2f Fahrenheit = %.2f Celsius", Fahrenheit, Celsius) ;
}
/*

```

Q : 10:12 : Write a program that takes a positive number with a fractional part and round it to two decimal places. For example, 25.4851 would round to 25.49, 62.4431 would round to 62.44.

Page : 174

```

*/

```

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float positive_real_num ;
    // Ask the user to enter a positive real number
    printf ("\nEnter a Positive Real Number => ");
    // Take input
    scanf ("%f", &positive_real_num) ;
    printf ("\n%.2f is a Positive Real Number", positive_real_num) ;
}
/*

```

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.

Page : 176

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int age, status ;
    printf("Enter the age : ");
    scanf ("%d", &age) ;
    status = (age > 60) ;
    printf("Status = %d", status) ;
}
/*

```

A program that calculates the square root of a number input by user.

Page : 177

```

*/
#include<stdio.h>
#include<conio.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 %f = %f", x, square_root) ;
    }
}

```

```
}  
/*
```

A program that calculates the square root of a number input by user using if-else.

Page : 179

```
*/  
#include<stdio.h>  
#include<conio.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 %f = %f", x, square_root) ;  
    }  
    Else  
    {  
        printf("Square Root of a Negative number can't be calculated") ;  
    }  
}  
/*
```

A program that accepts three numbers from the user and displays the largest number.

Page : 181

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.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 the largest", a) ;  
        }  
        else  
        {  
            printf("%d is the largest", c) ;  
        }  
    }  
    else
```



```

        {
            if (b > c)
            {
                printf("%d is the largest", b) ;
            }
            else
            {
                printf("%d is the largest", c) ;
            }
        }
    }
}
/*

```

A program that inputs a number from the user and determines whether it is positive, negative or zero.

Page : 182

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.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") ;
    }
}
/*

```

A program that inputs a number from the user and determines whether it is positive, negative or zero. The program is re-written using if-else-if statement :

Page : 184

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.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") ;
}
}
/*

```

A program that accepts three numbers from the user and displays the largest number.

Page : 185

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.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 the largest", a) ;
    }
    else if (b > a && b > c)
    {
        printf("%d is the largest", b) ;
    }
    else
    {
        printf("%d is the largest", c) ;
    }
}
/*

```

Write a program that inputs a character from the user and checks whether it is a vowel or a consonant.

Page : 186

```

*/

```

```

#include<stdio.h>
#include<conio.h>
#include<math.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's NOT a Vowel, It's a CONSONANT");
            break ;
    }
}
// OR
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    char ch ;
    printf("Enter an ALPHABET => ");
    ch = getche( ) ;
    switch (ch)
    {

```

```

        case 'a' : case 'A' : case 'e' : case 'E' : case 'i' : case 'I' : case 'o' : case
        'O' : case 'u' : case 'U' :
            printf("It's a Vowel");
            break ;
        default :
            printf("It's NOT a Vowel, It's a CONSONANT");
            break ;
    }
}
/*

```

Write a program that inputs a character from the user and checks the type of character / symbol.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    char ch ;
    printf("Enter an ALPHABET => ");
    ch = getch();
    switch (ch)
    {
        case 'a' : case 'A' : case 'e' : case 'E' : case 'i' : case 'I' : case 'o' : case
        'O' : case 'u' : case 'U' :
            printf("It's a Vowel");
            break ;
        case 'b' : case 'B' : case 'c' : case 'C' : case 'd' : case 'D' : case 'f' : case
        'F' : case 'g' : case 'G' :
        case 'h' : case 'H' : case 'j' : case 'J' : case 'k' : case 'K' : case 'l' : case
        'L' : case 'm' : case 'M' :
        case 'n' : case 'N' : case 'p' : case 'P' : case 'q' : case 'Q' : case 'r' : case
        'R' : case 's' : case 'S' :
        case 't' : case 'T' : case 'v' : case 'V' : case 'w' : case 'W' : case 'x' :
        case 'X' : case 'y' : case 'Y' : case 'z' : case 'Z' :
            printf("It's a Consonant");
            break ;
        case '0' : case '1' : case '2' : case '3' : case '4' : case '5' : case '6' : case
        '7' : case '8' : case '9' :
            printf("It's a Numeric Character");
            break ;
        default :
            printf("It's a Special Character");
            break ;
    }
}
/*

```

Use of Conditional operator.

Page : 187

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main(void)  
{  
    int    a,    b ;  
    printf("Enter TWO numbers => ") ;  
    scanf("%d %d", &a, &b) ;  
    a > b ? printf("%d is larger", a) : printf("%d is larger", b) ;  
}  
/*
```

A program that accepts TWO numbers x and y from the user and determines / displays the quadrant on which the point (x, y) lies.

Page : 188

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main(void)  
{  
    int    x,    y ;  
    printf("Enter values of x and y coordinates => ") ;  
    scanf("%d %d", &x, &y) ;  
    if (x == 0)  
    {  
        if (y == 0)  
            printf("The point lies on THE ORIGIN") ;  
        else  
            printf("The point lies on Y-AXIS") ;  
    }  
    else if (x > 0)  
    {  
        if (y == 0)  
            printf("The point lies 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") ;
    }
}
/*

```

Q : 11:9 : A year is a leap year if it is divisible by 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".

Page : 192

```

/*
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    unsigned int year = 0 ;
    printf("Enter The Year => ") ;
    scanf("%d", &year) ;
    if (year % 4 == 0)
    {
        if (year % 100 == 0)
        {
            if (year % 400 == 0)
            {
                printf("%d is a Leap Year", year) ;
            }
            else
            {
                printf("%d is a NOT Leap Year", year) ;
            }
        }
        else
        {
            printf("%d is a Leap Year", year) ;
        }
    }
    else
    {
        printf("%d is Not a Leap Year", year) ;
    }
}
/*

```

Q : 11:10 : 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 (Both Inclusive)	Pleasant Day
Less Than 25	Cool Day

Page : 192

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    int    temerature ;
    printf("Enter Temperature in Celsius => ");
    scanf("%d", &temerature) ;
    if (temperature > 35)
    {
        printf("%d Degrees Celsius ! Its a HOT Day", temperature) ;
    }
    else if (temperature >= 25)
    {
        printf("%d Degrees Celsius ! Its a PLEASANT Day", temperature) ;
    }
    else
    {
        printf("%d Degrees Celsius ! Its a COOL Day", temperature) ;
    }
}
/*

```

Q : 11:11 : Write a program that inputs obtained marks of a student, calculates percentage (assume total marks are 1100 or may get input from user), 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

Page : 192

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    int    marks ;

```

```

float percent ;
printf("Enter Obtained Marks => ");
scanf("%d", &marks);
percent = (marks / 1100) * 100 ; // Assume Total Marks = 1100
if (percent >= 80)
{
    printf("A+");
}
else if (percent >= 70)
{
    printf("A");
}
else if (percent >= 60)
{
    printf("B");
}
else if (percent >= 50)
{
    printf("C");
}
else if (percent >= 40)
{
    printf("D");
}
else if (percent >= 33)
{
    printf("E");
}
else
{
    printf("F");
}
}
/*

```

Q : 11:12 : 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 display the message "Wrong Choice". [Use switch statement to implement the solution].

Page : 192

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{

```



```

float num1, num2, result ;
float result2 ;
char choice ;
printf("Enter The Tow Numbers => ");
scanf("%d %d", &num1, &num2) ;
printf("1. Add\n");
printf("2. Subtract\n");
printf("3. Multiply\n");
printf("4. Divide\n");
printf("Please Enter Your Choice From 1 To 4 ==> ");
choice = getche() ;
switch(choice)
{
    case '1':    result = num1 + num2 ;
                printf("\nAddition %d + %d = %d", num1, num2, result) ;
                break ;
    case '2':    result = num1 - num2 ;
                printf("\nSubtraction %d - %d = %d", num1, num2, result) ;
                break ;
    case '3':    result = num1 * num2 ;
                printf("\nMultiplication %d * %d = %d", num1, num2, result) ;
                break ;
    case '4':    if (num1 > num2)
                {
                    result2 = num1 / num2 ;
                    printf("\nDivision %d / %d = %f", num1, num2, result2) ;
                }
                else
                {
                    result2 = num2 / num1 ;
                    printf("\nDivision %d / %d = %d", num2, num1, result2) ;
                }
                break ;
    default :    printf("\nIn-Valid Choice : Try 1 To 4") ;
} // end of switch block
} // end of main block
/*

```

Chapter 12

```

*/
/*

```

Write a program to print digits from 1 to 10 using while loop.

Page : 194

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)

```

```

{
    int count ;
    count = 1 ;
    while (count <= 10)
    {
        printf("%d\n", count) ;
        count = count + 1 ;
    }
}
/*

```

Write a program to print digits from 1 to 10 using do-while loop.

Page : 195

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    int count ;
    count = 1 ;
    do
    {
        printf("%d\n", count) ;
        count = count + 1 ;
    } while (count <= 10) ;
}
/*

```

Your telephone connection may be in any of two states working (W) or dead (D).

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. Force the user to enter a valid input value. This could be achieved by using a do-while loop.

Page : 196

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    char state ;
    do
    {
        printf("\nPlease Enter Current State Working (W) or Dead (D) => ");
        scanf("%c", &state) ;
    } while (state != 'w' && state != 'W' && state != 'd' && state != 'D') ;
    if (state == 'w' || state == 'W')
        printf("\nThe State of your phone is WORKING") ;
}

```

```

        else if (state == 'd' || state == 'D')
            printf("\nThe State of your phone is DEAD");
    }
/*

```

Write a program to print digits from 1 to 10 using “for” loop.

Page : 198

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    int    count ;
    for (count = 1 ;    count <= 10 ;    count++)
        printf ("%d\n", count) ;
}
/*

```

Write a program that will print asterisks (*) according to the given pattern :

```

*****
*****
*****
****
***
**
*

```

Page : 199

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    int    outer, inner ;
    for (outer = 7 ;    outer >= 1 ; outer--)
    {
        inner = 1 ;
        while ( inner <= outer )
        {
            printf ( "*" ) ;
            inner++ ;
        }
        printf ( "\n" ) ;
    }
}
/*

```

Write a program to find the average marks of the students in a class.

Page : 200

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int sum = 0, marks = 0, 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("Enter Marks of at least one student to Calculate
Average\n") ;
}
/*

```

Write a program to calculate the square root of a positive number (handle negative numbers properly).

Page : 202

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    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)) ;
}
/*

```

Q : 12:9 : 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”.

Page : 206

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    unsigned int num ;
    printf(“Please Enter a Positive Number To Check Primality => ”) ;
    scanf(“%f”, &num) ;
    if (num % 2 == 0)
    {
        printf(“The Number %d is NOT Prime”, num);
        exit(0);
    }
    else
    {
        for (int i = 3 ; i < sqrt(num) ; i += 2)
        {
            if (num % i == 0)
            {
                printf(“The Number %d is NOT Prime”, num);
                exit(0);
            }
        }
        printf(“The Number %d is Prime”, num);
    }
}
/*
```

Q 12:10 : Write a program that displays the first 15 even numbers.

Page : 206

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    for (int i = 1 ; i < 16 ; i ++ )
    {
        printf(“\n%d”, i*2);
    }
}
/*
```

Q : 12:11 : 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 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Page : 206

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    unsigned int num ;
    printf("Please Enter a Positive Number To Print Table => ");
    scanf("%f", &num) ;
    for (int i = 1 ; i < 11 ; i ++)
    {
        printf("\n%d x %d = %d", i, num, i * num) ;
    }
}
/*
```

Q : 12:12 : 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.

Page : 206

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int num = 0 , total = 0 ;
    do
    {
        printf("\nPlease Enter a Number (0 To 15) => ");
        scanf("%f", &num) ;
        total += num ;
    } while (num < 0 || num > 15) ;
    printf("\nThe Total = %d", total) ;
}
/*
```

Q : 12:13 : 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
```

Page : 206

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    for (int i = 0 ; i < 10 ; i++)
    {
        printf("\n") ;
        for (int j = 0 ; j <= i ; j++)
            printf("\t%d", j) ;
    }
}
/*
```

Q : 12:14 : Write a program the produces the following output :

```
0   1
1   2
2   4
3   8
4   16
5   32
6   64
```

Page : 206

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    for (int i = 0 ; i <= 6 ; i++)
        printf("\n\t %d \t %d", i, pow(2, i)) ;
}
/*
```

Chapter 13

```
*/
/*
```

The Scope of a Variable : 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.

Local Variables : All variables declared within a block (within the extent of a pair of curly braces) 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.

Page : 212

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main (void)  
{  
    int nCount = 0 ;  
    if (nCount == 0)  
    {  
        int chk ;  
        chk = 10 ;  
    }  
    printf(“%d”, chk) ;  
}  
/*
```

Global Variables : The variables which are declared outside all blocks i.e. outside / above the main() and all other functions are called global variables and have global scope.

Global Scope : The global variables 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.

Life Time of Global Variables : The lifetime of global variables is from the start of the program till its termination. They exist in memory from the start to the end of the program.

Page : 213

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main (void)  
{  
    int nCount = 1 ;  
    clrscr( ) ; // a function of conio.h used to clear screen  
    while (nCount <= 10)  
    {  
        int chk = 10 ;  
        printf(“%d\t”, chk) ;  
        chk = chk + 1 ;  
        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

Page : 213

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void Counter (void) ;  
int nCount = 0 ;  
void main (void)  
{  
    for (int n = 0 ; n <= 10 ; n += 2)  
        Counter () ;  
    printf("nCount = %d", nCount) ;  
}  
void Counter (void)  
{  
    nCount++ ;  
}  
/*
```

Write a function named Print_Asterisks that will print asterisks (*) according to the pattern shown and invoke a function call from the function main to print the asterisks.

```
*****  
*****  
****  
***  
**  
*
```

Page : 215

```
*/  
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void Print_Asterisks (void) ; // function prototype  
void main(void)  
{
```

```

        // Function call
        Print_Asterisks ( ) ;
    }
void Print_Asterisks (void) // function header
{
    // Function Body Starts
    int inner ;
    for (int outer = 7 ; outer >= 1 ; outer--)
    {
        inner = 1 ;
        while ( inner <= outer)
        {
            printf(“*”);
            inner++ ;
        }
        printf (“\n”) ;
    }
    // Function Body Ends
}
/*

```

Write a program that takes 2 numbers, uses a function to add these, and then display the result.

Page : 216

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
int Add (int num1, int num2) ;
void main (void)
{
    int a, b ;
    int sum ;
    clrscr( ) ; // clears 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 num1, int num2)
{
    return num1 + num2 ;
}
/*

```

Write a program using a function to calculate the area of a Triangle.

Page : 217

```

*/
#include<stdio.h>

```

```

#include<conio.h>
#include<math.h>
float Area_of_Triangle (int base, int altitude) ; // Function Prototype
void main (void)
{
    int a, b ;
    float area ;
    clrscr( ) ; // clears the screen
    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 = %.2f", area) ;
}
float Area_of_Triangle (int base, int altitude) // Function Header
{
    return (0.5 * base * altitude) ;
}
/*

```

Q : 13:8 : Write a program that calls 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.

Page : 221

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void Draw_Horizontal( ) ; // Function Prototype
void Draw_Vertical( ) ; // Function Prototype
void main (void)
{
    clrscr( ) ; / clears screen
    Draw_Horizontal( ) ; // draws upper horizontal line
    Draw_Vertical( ) ; // draws two vertical lines
    Draw_Horizontal( ) ; // draws lower horizontal line
}
void Draw_Horizontal( ) // Function Header
{
    // Go to new line and display 25 stars
    printf("\n") ;
    for (int i = 0 ; i < 25 ; i++)
        printf("*") ;
}
void Draw_Vertical( ) // Function Header
{
    for (int i = 0 ; i < 15 ; i++)

```

```

        printf("\n*          *") ; // 23 Spaces between stars
    }
// We can write Draw_Horizontal( ) function like this also – have only one
void Draw_Horizontal( ) // Function Header
{
    // Go to new line and display 25 stars
    printf("\n*****") ;
}
/*

```

Q : 13:9 : Write a program that prompts the user for the Cartesian coordinates of two points P1 (x1, y1) and P2 (x2, y2) 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 :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Page : 221

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
float Distance(int x1, int y1, int x2, int y2) ;
void main (void)
{
    int x1, y1, x2, y2 ;
    float distance ;

    clrscr( ) ; // clears screen

    printf("\nEnter x-Coord of Point 1 => ") ;
    scanf("%f", &x1) ;
    printf("\nEnter y-Coord of Point 1 => ") ;
    scanf("%f", &y1) ;
    printf("\nEnter x-Coord of Point 2 => ") ;
    scanf("%f", &x2) ;
    printf("\nEnter y-Coord of Point 2 => ") ;
    scanf("%f", &y2) ;

    distance = Distance(x1, y1, x2, y2) ;
    printf("\nDistance Between P1 and P2 = %.2f", distance) ;
}
float Distance(int x1, int y1, int x2, int y2)
{
    return (sqrt(((x2 - x1) * (x2 - x1)) + ((y2 - y1) * (y2 - y1)))) ;
}
/*

```

Q : 13:10 : Write a program that prompts the user to enter a number and then reverses 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.

Page : 221

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
int Reverse(int n) ;
void main (void)
{
    int a, b ;

    clrscr( ) ; // clears screen

    printf("\nEnter The Number To Reverse => ");
    scanf("%f", &a) ;
    b = Reverse(a) ;
    printf("\n%d Reversed = %d", a, b) ;
}
int Reverse(int n)
{
    int negative = 0 ;
    int first = 0, second = 0, third = 0, fourth = 0, fifth = 0 ;
    if (n < 0)
    {
        negative = 1 ;
        n *= -1 ;
    }
    first      =    n % 10 ;
    n          =    n / 10 ;
    second     =    n % 10 ;
    n          =    n / 10 ;
    third      =    n % 10 ;
    n          =    n / 10 ;
    fourth     =    n % 10 ;
    n          =    n / 10 ;
    fifth      =    n % 10 ;
    n          =    n / 10 ;

    b          =    fifth * 1 ;
    b          +=    fourth * 10 ;
    b          +=    third * 100 ;
    b          +=    second * 1000 ;
    b          +=    first * 10000 ;

    if (negative = 1)
        b = b * -1 ;
}
```

```

        return b ;
    }
    /*

```

Q : 13:11 : Write a function named Draw_Asterisks that will print asterisks (*) according to the pattern shown and make a function call from the function main to print the asterisks pattern.

```

*****
*****
*****
***
*

```

Page : 221

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
void Draw_Asterisks (void) ; // function prototype
void main(void)
{
    // Function call
    Draw_Asterisks ( ) ;
}
void Draw_Asterisks (void) // function header
{
    // Function Body Starts
    int inner ;
    for (int outer = 9 ; outer >= 1 ; outer -= 2)
    {
        inner = 1 ;
        while (inner <= outer)
        {
            printf(“*”);
            inner++ ;
        }
        printf (“\n”) ;
    }
    // Function Body Ends
}
    /*

```

Q : 13:12 : 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.

Page : 222

```

    */
#include<stdio.h>
#include<conio.h>
#include<math.h>
int Is_Prime(int num) ;

```

```

void main (void)
{
    unsigned int a ;
    printf("Enter a Positive Number > 2 To Check Primality => ") ;
    scanf("%f", &a) ;
    if (a < 3)
        printf("In-appropriate Number %d to Check Primality", a) ;
    else
    {
        int result = Is_Prime(a) ;
        if (result == 0)
            printf("The Number %d is NOT Prime", a);
        else
            printf("The Number %d is Prime", a);
    }
}

int Is_Prime(int num)
{
    if (num % 2 == 0)
        return 0 ;
    else
    {
        for (int i = 3 ; i <= sqrt(num) ; i += 2)
            if (num % i == 0)
                return 0 ;
        return 1 ;
    }
}

```

Q : 13:13 : 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 divided (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 (terminate). Write four functions Add(), Subtract(), Multiply() and Divide() to complete the task.

Page : 222

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void Add(int x, int y) ;
void Subtract(int x, int y) ;
void Multiply(int x, int y) ;
void Divide(int x, int y) ;
void main (void)
{

```

```

int a, b ;
char choice ;

clrscr( ) ; // clears screen

printf("\nEnter First Integer => ");
scanf("%f", &a) ;
printf("\nEnter Second Integer => ");
scanf("%f", &b) ;
printf("\nEnter Your Choice 1 .. 4 => ");
scanf("%f", &choice) ;
switch(choice)
{
    case '1' : Add(a, b) ; break ;
    case '2' : Subtract(a, b) ; break ;
    case '3' : Multiply(a, b) ; break ;
    case '4' : Divide(a, b) ; break ;
    default : printf("\n%c is a Bad Choice", choice) ; break ;
}
}
void Add(int x, int y) ;
{
    printf("%d + %d = %d", x+y) ;
}
void Subtract(int x, int y) ;
{
    printf("%d - %d = %d", x-y) ;
}
void Multiply(int x, int y) ;
{
    printf("%d * %d = %d", x*y) ;
}
void Divide(int x, int y) ;
{
    if (x > y)
    {
        if (y == 0)
            printf("Error – Divisor Can't be ZERO") ;
        else
            printf("%d / %d = %.2f", x / y) ;
    }
    else if (y > x)
    {
        if (x == 0)
            printf("Error – Divisor Can't be ZERO") ;
        else
            printf("%d / %d = %.2f", y / x) ;
    }
}

```



```

    }
    else
        printf("Both Numbers are EQUAL thus result = 1")
    }
}
/*

```

Q : 13:14 : 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.

Page : 222

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
long int Factorial(int n) ;
void main (void)
{
    unsigned int a ;
    long int factorial ;
    printf("Enter a Valid Number > 1 To Calculate Factorial => ") ;
    scanf("%f", &a) ;
    if (a < 2)
        printf("In-Valid Number < 2 To Calculate Factorial") ;
    else
    {
        factorial = Factorial(a) ;
        printf("%d Factorial = %ld", a, factorial) ;
    }
}
long int Factorial(int n)
{
    long int fact = 1 ;
    if (n == 1 || n == 0)
        fact = 1 ;
    else
    {
        for (int i = n ; i >= 2 ; i--)
            fact *= i ;
    }
    return fact ;
}
/*

```

Q : 13:15 : 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 calls the function GCD to compute the greatest common divisor of the numbers entered.

Page : 222

*/

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
int GCD(int m, int n) ;
void main (void)
{
    int a, b ;
    int gcd ;
    printf("Enter TWO integers To Calculate GCD => ") ;
    scanf("%d %d", &a, &b) ;
    if (a < 1 || b < 1)
        printf("In-Valid Numbers To Calculate GCD") ;
    else
    {
        gcd = GCD(a, b) ;
        printf("GCD(%d, %d) = %d", a, b, gcd) ;
    }
}
int GCD(int m, int n)
{
    int temp ;
    if (m > n)
    {
        if (m % n == 0)
            return n ;
        temp = n ;
        n = m % n ;
        m = temp ;
    }
    else if (n > m)
    {
        if (n % m == 0)
            return m ;
        temp = m ;
        m = n % m ;
        n = temp ;
    }
    else
        return n ;
}
/*

```

Chapter 14

```
*/
```

```
/*
```

Write a program to demonstrate the use of pointers.

Page : 226

```
*/
```

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int* var ;
    int num = 25 ;
    clrscr( ) ;
    var = &num ;
    printf (“\nAddress 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) ;
}
/*

```

Write a program that reads a file and then writes its contents to another file (copy file).

Page : 228

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE *input ;
    FILE *output ;
    int ch ;
    // Try to open the input file. If it fails, print an ERROR 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 file
    else if ((output = fopen(“bfile.txt”, “w”) == NULL)
    {
        printf(“Cant 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 both the files
    fclose(input) ;
    fclose(output) ;
}

```

```
/*
```

Write a program that accepts name and telephone numbers of your friends and write them in a file “Contacts.txt”.

Page : 232

```
*/
```

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE *ptrFile ;
    char name[30] ;
    char tel[12] ;
    if ((ptrFile = fopen(“d:\\Contacts.txt”, “w”)) == NULL)
        printf(“Can’t open Contacts.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 file
        fclose (ptrFile) ;
    }
}
```

```
/*
```

Write a program that will read the contacts.txt file, and displays its contents on the screen.

Page : 234

```
*/
```

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```

#include<string.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() ;
}
/*

```

Write a program that will append records in contacts.txt file.

Page : 235

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE* ptrFile ;
    char name[30] ;
    char tel [12] ;
    if ((ptrFile = fopen("d:\\Contacts.txt", "a")) == NULL)
        printf("Can't open Contacts.txt for writing / appending !\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 file
fclose (ptrFile) ;
}
/*

```

Write a program (using Formatted I/O) that accepts name and telephone numbers of your friends and write them in a file "Contacts.txt".

Page : 237

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE *ptrFile ;
    char name[30] ;
    char tel[12] ;
    if ((ptrFile = fopen("d:\\Contacts.txt", "w")) == NULL)
        printf("Can't open Contacts.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) ;
        // this line replaces 4 lines commented below
        // fputs(name, ptrFile) ;
        // fputs("!", ptrFile) ;
        // fputs(tel, ptrFile) ;
        // fputs ("\n", ptrFile) ;
    }
    }while(strlen(name) > 0) ;
    // Close the file
    fclose (ptrFile) ;
}
}
/*

```

Q : 14:8 : Write a program to merge the contents of two text files.

Program Merge Files :

Important Note : Open Files a.txt for input and b.txt for appending. Read a.txt character by character and append each character at the end of b.txt (merge files).

Page : 239

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE *input ;
    FILE *appendoutput ;
    int ch ;
    // Try to open the input file. If it fails, print an ERROR message.
    if ((input = fopen("a.txt", "r")) == NULL)
        printf("Can't open a.txt for reading ! \n") ;
    // Now try to open the output file. If it fails, close the input file
    else if ((appendoutput = fopen("b.txt", "a") == NULL)
    {
        printf("Cant open b.txt for merging / appending ! \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, appendoutput) ;
    }
    // Close both the files
    fclose(input) ;
}

```

```

        fclose(appendoutput) ;
    }
    /*

```

Q 14:9 : Write a program that counts the total number of characters in a text file.
[Note: consider the blank. space a character]

Program Count Characters in a File :

Important Note : Open File a.txt for input read it character by character and keep counting. At the end of the file print / display # of characters counted.

Page : 239

```

    /*
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE *input ;
    int ch, count = 0 ;
    // Try to open the input file. If it fails, print an ERROR message.
    if ((input = fopen("a.txt", "r")) == NULL)
        printf("Can't open a.txt for reading/counting characters! \n") ;
    else
    {
        while ((ch = getc(input)) != EOF) // count ch.
            count++ ;
    }
    printf("\n # of Characters in a.txt = %d", count) ;
    // Close the file
    fclose(input) ;
}
    /*

```

Q 14:10 : Write a program that counts the number of words in a text files and display the count on the screen.

Program Count Words in a File :

Important Note : Open File a.txt for input read it character by character and keep counting spaces only that separate words from each other. At the end of the file print / display # of words counted.

Page : 239

```

    /*
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main (void)
{
    FILE *input ;
    int ch, count = 0 ;

```



```

// Try to open the input file. If it fails, print an ERROR message.
if ((input = fopen("a.txt", "r")) == NULL)
    printf("Can't open a.txt for reading/counting characters! \n");
else
{
    while ((ch = getc(input)) != EOF) // count ch.
        if (ch == ' ') count++;
}
count++;
printf("\n # of Words in a.txt = %d", count);
// Close the file
fclose(input);
}
/*

```

Chapter 15 : BISE Practical

```

*/
/*
Q      :      15-01-01      :      Write a program in C Language to Convert Fahrenheit
into Celsius by using formula  $C = 5 / 9 (F - 32)$ .
*/

```

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float f, c ;
    printf("Please Enter Temperature in Fahrenheit => ");
    scanf("%f", &f);
    c = (5 / 9) * (f - 32);
    printf("\nTemperature in Celsius = %f", c);
}
/*

```

OR Like This

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float f;
    printf("Please Enter Temperature in Fahrenheit => ");
    scanf("%f", &f);
    printf("\nTemperature in Celsius = %f", (5 / 9) * (f - 32));
}
/*

```

Q : 15-01-02 : Write a program in C Language which displays the Maximum and Minimum from N numbers (three numbers taken from user), also show its output.

*/

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
void main (void)
```

```
{
```

```
    unsigned int a, b, c;
```

```
    printf("Please Enter Value of Integer Number a => ");
```

```
    scanf("%d", &a);
```

```
    printf("Please Enter Value of Integer Number b => ");
```

```
    scanf("%d", &b);
```

```
    printf("Please Enter Value of Integer Number c => ");
```

```
    scanf("%d", &c);
```

```
    if (a >= b)
```

```
    {
```

```
        if (a >= c)
```

```
            printf ("Maximum Value = %d", a);
```

```
        else
```

```
            printf ("Maximum Value = %d", c);
```

```
        if (b >= c)
```

```
            printf ("Minimum Value = %d", c);
```

```
        else
```

```
            printf ("Minimum Value = %d", b);
```

```
    }
```

```
    else
```

```
    {
```

```
        if (b >= c)
```

```
            printf ("Maximum Value = %d", b);
```

```
        else
```

```
            printf ("Maximum Value = %d", c);
```

```
        if (c >= a)
```

```
            printf ("Minimum Value = %d", a);
```

```
        else
```

```
            printf ("Minimum Value = %d", c);
```

```
    }
```

```
}
```

```
/*
```

Q : 15-01-03 : Write a program in C Language to Print The Output as shown below by using goto and Nested Loop statement.

```
1    2    3
4    5    6
```

```

7      8      9
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    unsigned int num = 1 ;
    outer_loop:
    if (num == 9)
        goto stop_printing ;
    inner_loop:
        printf ("%d\t", num) ;
        num++ ;
        if (num % 3 != 0)
            goto inner_loop ;
        else
            goto outer_loop ;
    stop_printing ;
}
/*

```

Q : 15-01-04 : Write a program in C Language to Calculate The Product of First 20 Odd Numbers.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    unsigned int odd, product ;
    for (int i = 0 ; i < 20 ; i ++ )
    {
        product = product * (i * 2 + 1) ;
    }
    printf("Product of First 20 Odd Numbers = %d", product) ;
}
/*

```

Q : 15-01-05 : Write a program in C Language to Calculate and Print The Area of a Geometrical Figure (Square, Rectangle, Right-Angle Triangle, Circle with appropriate formula). (Take Values from user.). One of following required in EXAM (not all simultaneously).

```

*/
// Area of Square
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)

```

```

{
    float s, area ;
    printf("Please Enter Length of Side of Square => ");
    scanf("%f", &s);
    area = s * s;
    printf("\nArea of Square = %f", area);
}

// Area of Rectangle
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float length, width, area ;
    printf("Please Enter Length of Rectangle => ");
    scanf("%f", &length);
    printf("Please Enter Width of Rectangle => ");
    scanf("%f", &width);
    area = length * width;
    printf("\nArea of Rectangle = %f", area);
}

// Area of Right-Angle Triangle
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float base, perpendicular, area ;
    printf("Please Enter Base of Triangle => ");
    scanf("%f", &base);
    printf("Please Enter Perpendicular of Triangle => ");
    scanf("%f", &perpendicular);
    area = base * perpendicular / 2;
    printf("\nArea of Right-Angle Triangle = %f", area);
}

// Area of Circle
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    float radius, area ;
    printf("Please Enter Radius of Circle => ");
    scanf("%f", &radius);
    area = (22 / 7) * radius * radius;
    printf("\nArea of Circle = %f", area);
}

```

/*
Q : 15-01-06 : Write a program in C Language to Print out The
Output as below.

```

*
* *
* * *
* * * *
* * * * *
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void Print_Asterisks (void) ; // function prototype
void main(void)
{
    // Function call
    Print_Asterisks ( ) ;
}
void Print_Asterisks (void) // function header
{
    // Function Body Starts
    int inner ;
    for (int outer = 1 ; outer <= 5 ; outer++)
    {
        inner = 1 ;
        while ( inner <= outer)
        {
            printf(“* ”) ;
            inner++ ;
        }
        printf (“\n”) ;
    }
    // Function Body Ends
}
*/

```

Q : 15-01-07 : Write a program in C Language to Print Following
Output with the Help of Nested Loop statement.

```

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
*/

```

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    unsigned int num ;
    printf("Please Enter a Number => ") ;
    scanf("%f", &num) ;
    for (int i = 1 ; i <= num ; i ++)
    {
        for (int j = 1 ; j <= num ; j ++)
        {
            printf("%d\t", i) ;
        }
    }
} // Enter Value 9 and see result
/*

```

Q : 15-01-08 : Write a program in C Language that takes three distinct integral values as input from the user. Then program will display the entered numbers in **Descending** order. If user enters 2, 6 & 4 then program will display the numbers as 6, 4 & 2.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, b, c ;
    printf("Please Enter Three Numbers => ") ;
    scanf("%d %d %d", &a, &b, &c) ;
    if (a > b & a > c)
    {
        if (b > c)
            printf("\n%d, %d & %d", a, b, c) ;
        else
            printf("\n%d, %d & %d", a, c, b) ;
    }
    else if (b > a & b > c)
    {
        if (a > c)
            printf("\n%d, %d & %d", b, a, c) ;
        else
            printf("\n%d, %d & %d", b, c, a) ;
    }
    else
    {
        if (a > b)

```

```

        printf("\n%d, %d & %d", c, a, b) ;
    else
        printf("\n%d, %d & %d", c, b, a) ;
}
} // Enter Values 2, 6 & 4 and see result
/*

```

Q : 15-01-09 : Write a program in C Language that takes three digit single integral value (100 – 999) as input from the user. Then program will determine whether the entered number is palindrome or not. **Note** : A number is said to be palindrome if number will remain same when reversed. 552 is not a palindrome but 505, 252, 373 are all palindromes.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a ;
    printf("Please Enter a Number => ") ;
    scanf("%d", &a) ;
    if (a % 10 == a / 100)
        printf("%d is a Palindrome", a) ;
    else
        printf("%d is NOT a Palindrome", a) ;
}
/*

```

Q : 15-01-10 : Write a program in C Language that displays the following pattern on the output window / Console using **nested loop**.

```

1   2   3   4
2   3   4
3   4
4

```

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    printf("\n") ;
    for (int i = 1 ; i <= 4 ; i++)
        for (int j = i ; j <= 4 ; j++)
        {
            printf("%d\t", j) ;
        }
    printf("\n") ;
}
}

```

/*

Q : 15-01-11 : Write a program in C Language that takes an integral value as input from the user. The program will decide whether the entered number is **Composite** or not (by displaying appropriate message) using **while** loop. Note : A number is said to be composite, if it is greater than 1 and has more than two distinct factors.

*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, i = 2, isComposite = 0 ;
    printf("Please Enter a Number ==> ");
    scanf("%d", &a) ;
    while (i < a)
    {
        if (a % i == 0)
        {
            isComposite = 1 ;
            break;
        }
        i++;
    }
    if (isComposite == 1)
        printf("%d is a Composite Number", a) ;
    else
        printf("%d is NOT a Composite Number", a) ;
}
```

/*

Q : 15-01-12 : Write a program in C Language that takes three distinct integral values as input from the user. Then program computes and displays the **middle number** among the three entered numbers. If user enters 2, 6 & 4 then program will display the number 4.

*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, b, c ;
    printf("Please Enter Three Numbers ==> ");
    scanf("%d %d %d", &a, &b, &c) ;
    if (a > b & a > c)
    {
        if (b > c)
            printf("\n%d is the middle number", b) ;
    }
}
```



```

        else
            printf("\n%d is the middle number", c) ;
    }
    else if (b > a & b > c)
    {
        if (a > c)
            printf("\n%d is the middle number", a) ;
        else
            printf("\n%d is the middle number", c) ;
    }
    else
    {
        if (a > b)
            printf("\n%d is the middle number", a) ;
        else
            printf("\n%d is the middle number", b) ;
    }
} // Enter Values 2, 6 & 4 and see result
/*

```

Q : 15-01-13 : Write a program that takes a floating/decimal point value (percentage of a student in Inter Exam from 0 - 100) as input from the user. Then program will display the status of the student (**Fail** (0% – 40%), **Conditionally Pass** (41% - 59%) or **Pass** (above 60%)) on the basis of input value.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
    float percentMarks ;
    printf("Enter Obtained Percentage Marks => ") ;
    scanf("%f", &percentMarks) ;
    if (percent >= 60)
    {
        printf("Pass") ;
    }
    else if (percent >= 41)
    {
        printf("Conditionally Pass") ;
    }
    else
    {
        printf("Fail") ;
    }
}
/*

```

Q : 15-01-14 : Write a program in C Language that displays the following pattern on the output window / Console using **nested loop**.

```
1   2   3   4
    4   5   6
      6   7
        7
```

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int additive = 1, highest ;
    printf("\n") ;
    for (int i = 1 ; i <= 4 ; i++)
    {
        for (int j = 1 ; j < i ; j++)
            printf("\t") ;
        for (int k = 0 ; k <= 4 - i; k++)
        {
            printf("%d\t", k + additive) ;
            highest = k + additive ;
        }
        additive = highest ;
        printf("\n") ;
    }
}
/*
```

Q : 15-01-15 : Write a program in C Language that takes three distinct integral values as input from the user. Then program computes and displays the **smallest number** among the three entered numbers. The program should enforce the user to enter different values same numbers shall not be compared.

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, b, c ;
    do
    {
        printf("\nPlease Enter Three Distinct Numbers => ") ;
        scanf("%d %d %d", &a, &b, &c) ;
    } while (a == b || a == c || b == c) ;
    if (a > b & a > c)
    {
        if (b > c)
```

```

        printf("\n%d is the smallest number", c) ;
    else
        printf("\n%d is the smallest number", b) ;
}
else if (b > a & b > c)
{
    if (a > c)
        printf("\n%d is the smallest number", c) ;
    else
        printf("\n%d is the smallest number", a) ;
}
else
{
    if (a > b)
        printf("\n%d is the smallest number", b) ;
    else
        printf("\n%d is the smallest number", a) ;
}
} // Enter Values 2, 6 & 4 and see result
/*

```

Q : 15-01-16 : Write a program in C Language that takes an integral value as input from the user. The program will decide whether the entered number is **Prime** or not (by displaying appropriate message) using **while** loop. Note : A number is said to be composite, if it is greater than 1 and has no more than two distinct factors.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, i = 2, isPrime = 1 ;
    printf("Please Enter a Number To Test Primality => ") ;
    scanf("%d", &a) ;
    while (i < a)
    {
        if (a % i == 0)
        {
            isPrime = 0 ;
            break;
        }
        i++;
    }
    if (isPrime == 1)
        printf("%d is a Prime Number", a) ;
    else
        printf("%d is NOT a Prime Number", a) ;
}

```

/*
Q : 15-01-17 : Write a program in C Language that displays the following pattern on the output window / Console using **nested loop.**

```

}      }      }
?      ?      ?
#      #      #
@      @      @

```

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    printf("\n");
    for (int i = 1 ; i <= 4 ; i++)
    {
        for (int j = 1 ; j <= 3 ; j++)
        {
            if (i == 1)
                printf("{}\t");
            else if (i == 2)
                printf("? \t");
            else if (i == 3)
                printf("#\t");
            else
                printf("@\t");
        }
        printf("\n");
    }
}
*/

```

Q : 15-01-18 : Write a program in C Language that calculate the sum of all **two digit Prime numbers (from 11 to 97) using any loop and then displays the computed sum on the output window / Console. **Note** : A number is said to be **Prime**, if it is greater than 1 and has no more than two distinct factors.**

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int isPrime = 1;
    long int totalPrime = 0 ;
    for (int i = 11 ; i <= 97 ; i++)
    {
        isPrime = 1;
        for (int j = 2 ; j < i; j++)

```

```

        {
            if (i % j == 0)
            {
                isPrime = 0 ;
                break;
            }
        }
        if (isPrime == 1)
            totalPrime += i;
    }
    printf("The SUM of 2 Digit Primes = %d", totalPrime) ;
}
/*

```

Q : 15-01-19 : Write a program in C Language that takes an integral value as input from the user. The program calculates and displays all the factors / divisors of the entered number on the output window / Console using **while loop**.

```

/*
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, i = 1, j ;
    printf("Please Enter a Number To Display its Factors => ") ;
    scanf("%d", &a) ;
    printf("\nThe Factors of %d => ", a) ;
    while (i <= a)
    {
        if (a % i == 0)
            printf(" %d, ", i);
        i++;
    }
}
/*

```

Q : 15-01-20 : Write a program in C Language that displays the following pattern on the output window / Console using **nested loop**.

```

1    3    5    7
2    4    6    8
3    5    7    9
4    6    8    10
5    7    9    11
6    8    10   12
7    9    11   13
8    10   12   14
9    11   13   15
10   12   14   16
*/

```

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    printf("\n");
    for (int i = 1 ; i <= 10 ; i++)
        for (int j = i ; j <= i+6 ; j += 2)
        {
            printf("%d\t", j) ;
        }
        printf("\n");
    }
}
/*

```

Q : 15-01-21 : Write a program in C Language that computes the sum of the following series : $4/5 - 6/8 + 8/11 - 10/14 + 12/17 - 14/20 + \dots$ up to n terms using any loop where 'n' (number of terms) is entered by the user as input. The program also displays the computed sum on output window / Console.

```

*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int n ;
    double sumSeries = 0.0 ;
    printf("\nThe Series 4/5 - 6/8 + 8/11 - 10/14 + ..... \n");
    printf("\nPlease Enter Number of Terms To SUM => ");
    scanf("%d", &n) ;
    for(int i = 4, j = 1; j <= n; i+=2, j++)
    {
        if(j % 2 == 0)
        {
            sumSeries -= i / (i+j) ;
            printf("%d/%d + ", i, i+j);
        }
        else
        {
            sumSeries += i / (i+j) ;
            printf("%d/%d - ", i, i+j);
        }
    }
    printf (" = %ld", sumSeries);
}
/*

```

Q : 15-01-22 : Write a program in C Language that takes two integral values as input from the user. The program computes successive / iterative sum of 1st number upto 2nd number using **while loop**. The program also displays the result on the output window / Console like for 3 and 4 entry : 3 + 3 + 3 + 3 (4 Times) = 12.

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    int a, b, i = 1, sum = 0 ;
    printf("Please Enter a Number for Successive Summation => ");
    scanf("%d", &a) ;
    printf("Please Enter a Number for Iterations => ");
    scanf("%d", &b) ;
    while (i <= b)
    {
        printf (" + %d, ", i) ;
        sum += a ;
        i++;
    }
    printf (" (%d Times) = %d", b, sum) ;
}
/*
```

Q : 15-01-23 : Write a program in C Language that calculate the sum of all **three digit EVEN numbers** (from 100 to 998) using any loop and then display the computed sum on the output window / Console.

```
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main (void)
{
    long int sum = 0 ;
    for (int i = 2 ; i <= 998 ; i+=2)
    {
        sum += i ;
    }
    printf("The SUM of EVEN Numbers from 100 - 998 = %ld", totalPrime) ;
}
}
```