

COMPUTER SCIENCE

(For 11th Class)



Punjab School Education Board

Sahibzada Ajit Singh Nagar

© Punjab Government

Edition 2016 11,000 copies

All rights, including those of translation, reproduction
and annotation etc., are reserved by the
Punjab Government

Translator : PICTES

Co-ordinator : Manvinder Singh Mathoda

**Vetter : Simrat Bhatti
G.G.S.S.S. Sohana
(SAS Nagar)**

WARNING

1. The Agency-holders shall not add any extra binding with a view of charge extra money for the binding. (Ref. CI. No.7 of agreement with Agency-holders)
2. Printing, Publishing, Stocking, Holding or Selling etc., of Spurious Text-Books qua textbooks printed and published by the Punjab School Education Board is a cognizable offence under Indian Penal Code.
(The textbooks of the Punjab School Education Board are printed on paper carrying water mark of the Board.)

Price : ₹ 83.00

Published by : **Secretary**, Punjab School Education Board, Vidya Bhavan Phase-8 Sahibzada
Ajit Singh Nagar-160062 & Printed by M/s Bright Printers Jalandhar City

PREFACE

Punjab School Education Board has been updating the school level syllabi compatible with modern approach and latest research. The previously written text-books are in continuous process of revision according to the latest syllabi. The board has also launched a special drive to prepare new text books as per latest National Policies in this regard. The present book is a part of this prestigious program.

The knowledge in the subject of Computer Science is the need of the hour because its study is essential for enhancement of efficient usage of Science and Technology in every field of modern era. Computerization of every department is done to keep it updated in light of all round development of Information Technology and Communication. The knowledge of Computer Education as well as usage of internet is necessary for everyone to have latest information about different departments to avail facilities of E-Ticketing etc.

Keeping in view of these requirements Punjab School Education Board has introduced Computer Science as a compulsory subject at Elementary and Secondary levels as per guidelines of Punjab Government. This subject is already being taught by PICTES in some Government Schools. The present book is English translation of its Punjabi version prepared according to revised syllabus on the demand of teachers. Every effort has been made to include each requisite information regarding the subject in this book. I hope it will be useful for students and teachers.

All suggestions for the improvement of the book will be highly appreciated.

Chairperson
Punjab School Education Board

CONTENTS

S.NO.	CHAPTER	PAGE NO.
1.	REVIEW OF CLASS X	1 - 16
1.0	Review on Software concepts.	
1.1	System Software : Operating systems, Utilities software, Application Software	
1.2	Review on Excel : (Data types, Formula & Functions) Excel & Financial Data, Count in function, IF, AND, OR & NOT function (Logical), V Look Up Function, V Look up Formula.	
1.3	Review on HTML : Web Pages, Words to Know (Tag, Element, Attribute), What is HTML File, Tables & the Border attributes.	
1.4	Review on Microsoft Access : Manipulating Data, Database of objects (Tables, Queries, Forms, reports, Pages, Data Base Window).	
1.5	Review on Programming Concepts : The various stages of a program development Cycle (Analyzing the Problem, Development of Solution, Coding the solution, Testing the Program), Elements of Programming Language.	
2.	PROGRAMMING IN 'C' LANGUAGE	17 - 36
2.0	Introduction	
2.1	Distinctive Features of C Language	
2.2	The characters Set : Execution of Escape Characters, White Space Characters.	
2.3	Structure of a C Program : Header files, Preprocessor statement, directives, Global Declaration (Main(), Braces, Declarations, Statements). Compiling and execution of C Program.	
2.4	Use of Editor	
2.5	Function : Built in Functions, User Defined Functions	
2.6	Formatted I/O functions [Scorif() & Printf() Function]	
2.7	Starting with (Programming : Installing turbo C, Compiling the program and executing the program.	
3.	CONSTANT, VARIABLES & DATA TYPES	37 - 48
3.0	Introduction	
3.1	Constants / Literals : The types of Constants (Numeric Constant, Character Constant, String Constants)	
3.2	Types of C Variable : Delimiters, Declaration / Initialization of variables (Data types, Varlist, Semicolon, Assigning value to the variables), Storing constant in a variable.	
3.3	Data Types : Built in data types (Integers(int), Character data type (Char), Double, The Void Data Type, The main functions Header	

3.4	Tokens (Identifiers, Keywords, constants, operators): Keywords and identifiers, Identifiers, Types modifier or Qualifier, Unsigned int, Double long.	
4.	OPERATORS AND EXPRESSIONS	49 - 61
4.0	Introduction	
4.1	Operators and Expressions : Expression, Binary, Operator, Arithmetic Operators, Operations & Hierarchical order.	
4.2	Relational & Logical Operator : Relational Operator, Logical Operators (Logical AND (&&) Operators, Logical OR (::), Logical NOT(!)), Assignment Operators, Increment & Decrement Operators, Ternary Operators, Size of () of Operators, Bitwise Operator.	
5.	CONTROL FLOW (PART - 1)	62 - 73
5.0	Introduction : Branching, Looping Jumping	
5.1	Decision making statements : If statements, if else statement	
5.2	Switch Statement	
5.3	The Break Statement	
5.4	Continue Statement	
6.	CONTROL FLOW (PART -2)	74 - 84
6.0	Introduction	
6.1	Control loop Structures : The while statements, Do while statement, For statement (Loop)	
7.	ARRAYS (PART -1)	85 - 101
7.0	Introduction	
7.1	Declaring and Initialization of array : Initializing arrays	
7.2	Some special rules : Entering Data into an Array	
7.3	Copying arrays	
7.4	Accessing the values of an array.	
7.5	Manipulation of array elements : Sum of Elements.	
8.	ARRAYS (PART-2)	102 - 115
8.0	Introduction	
8.1	Two dimensional array : Declaration of Two dimensional arrays, Let us see representation of 2 dimensional array, Initialization of 2-dimensional array, Initialization of the two-dimensional array elements.	
8.2	Memory Occupation in Two dimensional Arrays Elements.	
8.3	Multidimensional Array Character type : Accessing elements of Multidimensional Arrays, Initialization of char type multi dimensional arrays, two dimensional array of char word processing	
8.4	Single Character input output	
8.5	#Define director.	

9. DESKTOP PUBLISHING

116 - 122

- 9.0 Introduction of DTP
- 9.1 Print Documents
- 9.2 Printing Methods : Offset Printing (Lithography), Laser Printing
- 9.3 Fonts
- 9.4 Scaling, Tracking and Leading
- 9.5 Frames
- 9.6 Page Layouts
- 9.7 WYS/WYG (What-you-see-is-what-you-get)
- 9.8 Advantages of DTP over word processor.
- 9.9 Documents Planning : Page layout, Style, Margin, Header and Footer, Font.

COMMON PROGRAMMING ERRORS.

123 - 134

Review of Class X

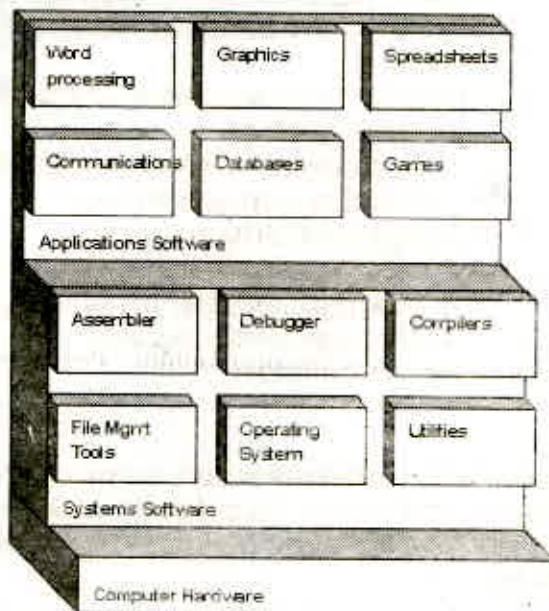
1.0 Review on Software Concepts

A program is a set of instructions, given to the computer in a particular sequence for solving a given problem. Without software, a computer is like a car without petrol.

The broad categories of software are:

1.1 System Software

System software are those software that are used to operate the system e.g.: Windows NT, Unix, Linux.



Note : Some Systems Software is built into the computer. It helps to setup the computer and start it, For Example To read the data written in ROM chips and BIOS.

1.1.1 Operating System

It is a part of System Software and it lies between the hardware and the application software. The operating system is a program that conducts the communication between the various pieces of hardware like the video card, sound card, printer the motherboard and the applications i.e.

1. **It is used by the computer's hardware to work with its parts. Tells the computer how to :**
 - Display information on the screen, & to use a printer.
 - Store information on a secondary storage device.
2. **The operating system, is usually located on a disk.**
 - It can be either on the hard disk, a floppy disk, or CD-ROM disk.
 - Must be loaded into RAM before it can be used.

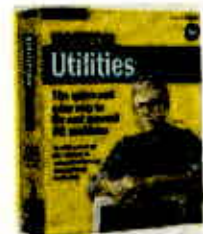
The system software that controls peripherals is called drivers i.e. this software's is used to run the peripheral attached to a system. Some of drivers are in operating system itself. An Operating system works with application software.

- Does basic tasks, like printing a document or saving a file.
- The operating system starts the application software so that it can be used.

1.1.2 Utilities software

Utilities allow you to complete certain tasks on your computers. Examples of some of these tasks are file organization.

- There is Specific purpose application software used to help a computer work well or to avoid problems.
- Some utility programs are built into the operating system.
 - Scandisk in the Windows operating system
 - Disk Formatting software
- Examples of utility programs
 - Anti-Virus software
 - Disk maintenance software
- File Management programs
- Security software



1.1.3 Application Software

Application software programs work with the operating system software to help you to use computer to do specific types of work such as word processing to type a letter.

- Examples of general-purpose application software are Word Processing software, Database software, Spreadsheet software, Desktop Publishing software



1.2 Review on Excel

Excel is an electronic spreadsheet program that can be used for storing, organizing and manipulating data. When you look at the Excel screen you see a rectangular table or grid of rows and columns. The horizontal rows are identified by numbers (1,2,3.....) and the vertical columns with letters of the alphabet (A,B,C.....). For columns beyond 26, columns are identified by two or more letters such as AA, AB, AC..... The intersection point between a column and a row is a small rectangular box known as a cell. A cell is the basic unit for storing data in the spreadsheet. Because an Excel spreadsheet contains thousands of these cells, each is given a cell reference or address to identify it. The cell reference is a combination of the column letter and the row number such as A3, B6, A34.

1.2.1 Data Types, Formula & Functions

The types of data that a cell can hold include numbers, text or formulas. Just as in math class, formulas are used for calculations- usually involving data contained in other cells. Excel and other electronic spreadsheets include a number of built in formulas used for common tasks known as functions.

1.2.2 Excel & Financial Data

Spreadsheets are often used to store financial data. Formulas and functions that are used on this type of data include. Performing basic mathematical operations such as summing column and rows of figures. Finding values such as profit or loss Calculating

repayment plans for land or mortgages, finding average, maximum or minimum values in a specified range of data. Some other uses of excel are :

The information garnered in a spreadsheet can easily be incorporated into electronic presentations, web pages, or printed of in report form.

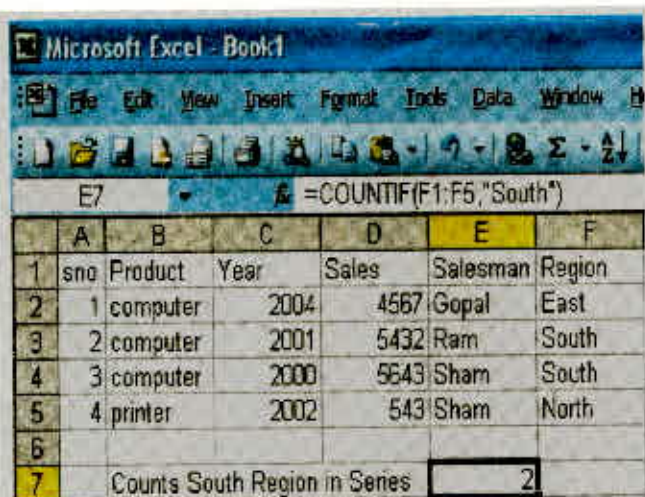
Excel formulas are one of the most useful features of the program. Formulas can be simple as adding two numbers or can be complex calculations needed for high end business projections. Once you learn the basic format of creating a formula, Excel does all the calculations for you. Let us use some of functions as.

Countif Functions

The **COUNTIF** functions is similar to the AVERAGE & the SUM function. You specify the range you wish to include & Excel counts only the number of values contained in the range. The COUNTIF function allows you to use criteria to focus in on the specific values you want to be counted. The syntax of the COUNTIF function is in the following form.

=COUNTIF(range, criteria)

Where range is the location of all the values the COUNTIF is to choose from & criteria are the expressions, text or values that define which cells will be counted.



The screenshot shows the Microsoft Excel interface. The formula bar at the top displays the formula `=COUNTIF(F1:F5,"South")` in cell E7. Below the formula bar is a table with the following data:

	A	B	C	D	E	F
1	sno	Product	Year	Sales	Salesman	Region
2	1	computer	2004	4567	Gopal	East
3	2	computer	2001	5432	Ram	South
4	3	computer	2000	5543	Sham	South
5	4	printer	2002	543	Sham	North
6						
7		Counts South Region in Series				2

On the formula bar type **=COUNTIF(F1:F5,"South")**
Result : Counts South Region in Series 2

IF, AND, OR & NOT function (Logical)

If all the functions in Excel, the if functions is most important. In fact, it's the IF statement in most software programs that really give computers logic & power. We can also use this powerful tool with Excel's IF function. The **if function** performs the tests to determine if the condition is true or false. If true, then it returns a certain value or takes a certain action, & it returns a different values or takes a action according to the Else portion.

The following is the syntax for the IF function:

= IF (Condition, Value-if-True, Value - IF-False)

Condition:

Value - IF - True

Value - IF - False

If function, can be used in conjunction with other functions. The AND, OR and NOT functions are used to create compound conditional tests. You may remember that the IF functions evaluates whether a condition is true or false. Using the AND, OR and NOT functions, we can evaluate whether computer conditions are true or false.

Follow the steps below to create the condition test using the IF function.

- Create the following worksheet as shown on the right side.
- Type the following formula in cell G4 :

= IF(F4>37, "Tour", "No Tour")

- Copy the formula in cell G4 to cells G5 through G8 by using the mouse pointer. Point to the small box in the lower-right corner of cell G4 until the mouse pointer from a thick cross cursor to a thin cross cursor. Hold & drag the mouse pointer to cell G8.

G4		=IF(F4>37,"Tour","No Tour")						
	A	B	C	D	E	F	G	H
1	Cricket Analysis		Wicket Cup					
2								
3	Cricketer	World Cup	Wills Cup	Pepsi Cup	Average	Age	Tour	No Tour
4	Jaffer	28	21	18	22.33	45	Tour	
5	A Kumble	30	26	23	26.33	32		
6	Dravid	24	34	31	29.67	35		
7	Tendulkar	34	41	32	35.67	41		
8	Dhoni	12	23	18	17.67	34		

Worksheet with IF(Condition)

Look at the formula in the formula Bar. The formula simply states that if the bowler's age is greater than 37, then the condition is true and returns the text value Tour. However, if the bowler's age is not greater than 37, then the condition is false and returns the text value No Tour.

In the following table to find the average of the age, construct the group condition by using if() & AND function.

- In cell H3, type the following columns heading: Age or Avg.
- Type the following formula in cell H4:

= IF(AND(F4>37,E4>=26), "Tour", "No Tour"))

- Copy the formula in cell H4 to cells H5 through H8 by using the mouse pointer. Point to the small box in the lower-right corner of cell H4 until the mouse pointer from a thick cross cursor to a thin cross cursor. Hold and drag the mouse pointer to cell H8.

H4		=IF(AND(F4>37,E4>=26),"Tour","No Tour")						
	A	B	C	D	E	F	G	H
1	Cricket Analysis		Wicket Cup					
2								
3	Cricketer	World Cup	Wills Cup	Pepsi Cup	Average	Age	Tour	No Tour
4	Jaffer	28	21	18	22.33	45	Tour	
5	A Kumble	30	26	23	26.33	32	No Tour	
6	Dravid	24	34	31	29.67	35	No Tour	
7	Tendulkar	34	41	32	35.67	41	Tour	
8	Dhoni	12	23	18	17.67	34	No Tour	

Worksheet with IF(AND) Condition

Look at the formula in the formula bar. The formula simply states that if the bowler age is greater than 37 and the bowler's average is equal to or greater than 26, then the condition is true and return the text value Tour. However, if the bowler's age is not greater than 37 and the bowlers average is not equal to or greater than 26, then the condition is false and returns the text value No Tour.

VLOOKUP Function

Sometimes you need to look up a value in another table. For example, if you're a teacher keeping grade sheets, you might want Excel to look up the letter grades corresponding to your students test score averages. For this work VLOOK UP Function can be used.

VLOOK Up (LOOK-up-value.....)

VLOOKUP Formula

Create the following table :

The table needs to be set up so that the values you are looking up (in this example, the averages) are in the left most column, as shown in Figure below:

H5	=VLOOKUP(76,J4:K8,2)										
	A	B	C	D	E	F	G	H	I	J	K
1	Annual Report					Class IX					
2											
3	Name	Maths	English	Science	Bio	Total	Average	Grade		Lookup Table	
4	Jaffer	45	45	56	67	213	53.25	F		0	F
5	A Kumble	90	56	67	90	303	75.75	C		60	D
6	Dravid	70	90	93	89	342	85.50	B		70	C
7	Tendulkar	67	78	56	67	268	67.00	D		80	B
8	Dhoni	45	56	67	78	246	61.50	F		90	A
9	Laxman	89	90	89	89	357	89.25	B			
10											

- Click the cell in which you want the result to appear; then click the Paste Function button.
- In the **All or Lookup & Reference** categories, double-click the VLOOKUP functions. The VLOOKUP dialog box appears, as shown in Figure.
- Click the **Look-up-value box** and click the cell that contains the value you want to look up (in this case, the average)
- Click the Table array box and drag to select the lookup table.
- In the **Col-index-num** box, type the number of the lookup table column. As in this case we have 2 columns.
- In the **Range-lookup** box, decide whether you want to find the closest match or an exact match. For the closest match (which is appropriate in this case); leave the box empty. For an exact match (for example, in an unsorted list), type false. We will type the value in the increasing order in the look up table and then on OK press enter.
- After this, copy the formula on the other downside rows of the table and press the Enter Key.

1.3 Review on HTML (Hyper text Markup Language)

HTML Language is used to create documents on the Word Wide Web. HTML defines the structures and layout of a Web document by using a variety of tags and attributes. The correct structure for an HTML document starts with <HTML><HEAD><TITLE> enter here what document is about </title></Head><Body> and ends with </Body></HTML>.

All the information you'd like to include in your Web page fits in between the <BODY>and </BODY> tags. There are hundreds of other tags used to format and lay our the information in a Web Page. Tags are also used to specify hypertext links.

1.3.1 Web Pages

Web pages have many uses. Here are some important facts about why web pages are so useful.

- A cheap and easy way to spread information to a large audience.
- Another medium to market your business.
- Let the world know about you with a personal website.

Words to Know

- Tag -** Used to specify ("mark-up") regions of HTML documents for the web browser to interpret. Tags look like this : <tag>
- Element -** A complete tag, having an opening<tag> and a closing </tag>.
- Attribute -** Used to modify the value of the HTML element. Elements will often have multiple attributes.

1.3.1 What is HTML File

An HTML file is a text file containing small **markup tags**. The markup tags tell the Web browser **how to display** the page.

An HTML file must have an **htm** or **html** file extension.

An HTML file can be created using a **simple text editor**.

If you are running Windows, start Notepad :

Type in the following text on the left side. Save the file as "mypage.htm". Start your Internet browser. Select "Open" (or "Open Page") in the File menu of your browser. A dialog box will appear. Select "Browse" (or "Choose File") and locate the HTML file you just created - "mypage.htm" - select it and click "Open".	<pre><html> <head> <title> Title of page</title> </head> <body> This is my first HTML page. This text is bold </body> </html></pre>
---	--

now you can see an address in dialog box, for example "C:\MyDocuments\mypage.htm". Click OK, and the browser will display the page.

Example

The first tag in your HTML document is <html>. This tag tells your browser that this is the start of an HTML document. The last tag in your document is </html>. This tag tells your browser that this is the end of the HTML document.

- <head> :** The text between the <head> tag & the </head> tag is header information. Header information is not displayed in the browser window.
- <title>:** The text between the <title> tag is the title of your document. The title is displayed in your browser's caption.
- <body>:** The text between the <body> tag is the text that will be displayed in your browser.
- :** The text between the and tags will be displayed in a bold font.

Let us see how to create a table using tags and to display data in it:

Tables are defined with the `<table>` tag. A table is divided into rows (with the `<tr>` tag), and each row is divided into data cells (with the `<td>` tag). The letters td stands for "table data", which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

```
<table border ="1">
<tr>
<td> row 1, cell 1 </td>
<td> row 1, cell 2</td>
</tr>
<tr>
<td> row 2, cell 1 </td>
<td> row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Tables & the Border attributes

If you do not specify a border attributes the table will be displayed without any borders. Sometimes this can be useful, but most of the time, you want the borders to show. To display a table with borders, you will have to use the border attributes.

```
<table border ="1">
<tr>
<td> Row 1, cell 1</td>
<td> Row 1, cell 2</td>
</tr>
</table>
```

Headings in a Table : Headings in a table are defined with the `<th>` tag.

```
<table border ="1" ?
<tr>
<th> Heading </th>
<th> Another Heading</th>
</tr>
<tr>
<td> row 1, cell 1</td>
<td> row 1, cell 2</td>
</tr>
<tr>
<td> row 2, cell 1</td>
<td> row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser.

Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Problem : Write program to display the output as

row 1, cell 1	row 1, cell 2
row 2, cell 1	

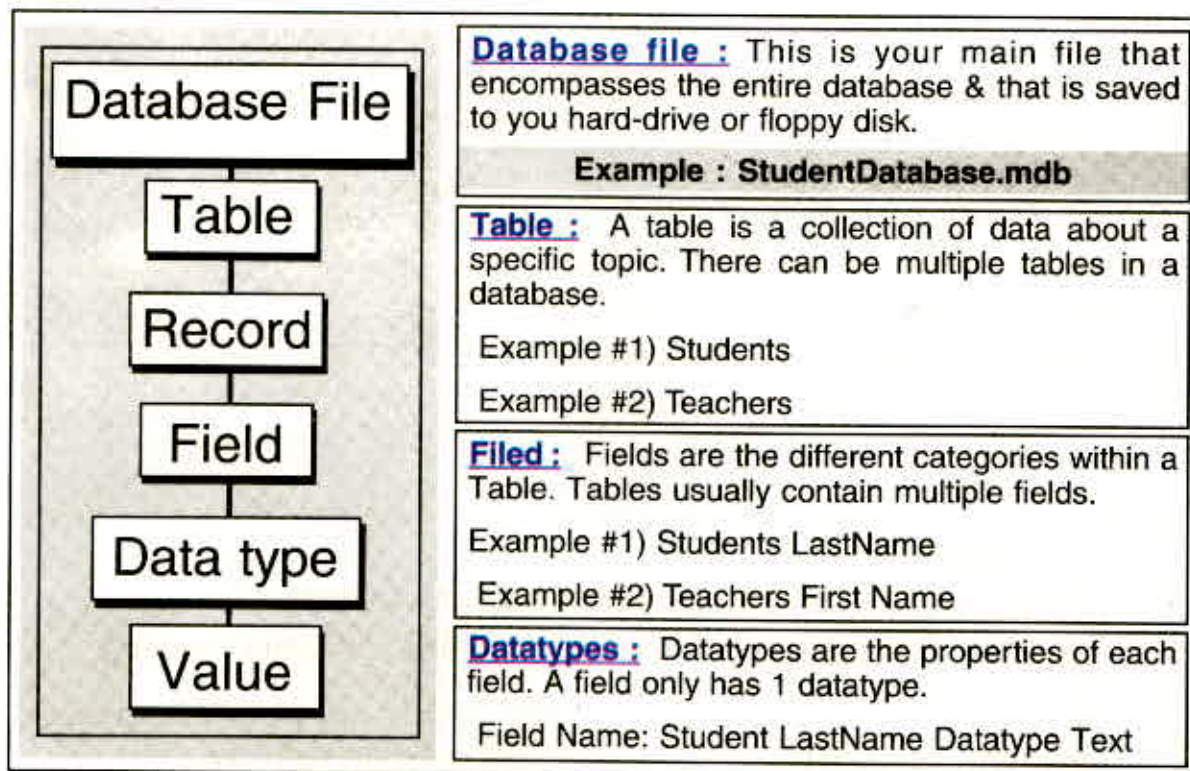
Ans : Empty Cells in a Table

Table cells with no content are not displayed very well in most browsers.

1.4 Review on Microsoft Access

Microsoft Access is a powerful program to create and manage your database. It has many built in features to assist you in constructing and viewing your information. Access is much more involved and is a more genuine database application than other programs such as Microsoft Works.

Some keywords involved in this process are : Database File, Table, Record, fields, Data-type. Here is the Hierarchy that Microsoft Access uses in breaking down a database.



1.4.1 Manipulating Data

Adding new row in record : Simply drop down to a new line & enter the information.

Updating a record : Simply select the record or field you want to update & change its data with what you want.

Deleting a Line in record : Simply select the entire row & press the Delete Key on the keyboard.

Relationship

If you've multiple tables in Microsoft Access database, and you want to extract information from more than one table. The first step in this process is to define relationships between you tables. After you've done that, you can create queries, forms and reports to display information from several tables at once. A relationship works by matching data in key fields—usually a field with the same name in both tables. In most cases, these matching fields are the primary key from one table, which provides a unique identifier for each record, and a foreign key in the other table.

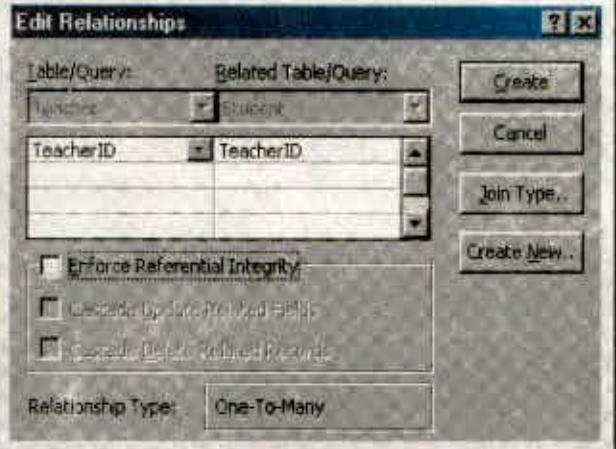


Figure : showing relationship b/w 2 tables

Example : Teachers can be associated with the students they're teaching By creating a relationship between the teachers table and the student's table using the **TeacherID** fields.

Having the met the criteria above, follow these steps for creating relationships between tables.

1. In the database window view, at the top, click on Tools--> Relationships.
2. Select the Tables you want to link together, by clicking on them and selecting the Add Button.
3. Drag the primary key of the Parent table (Teacher Table in this case), and drop it into the same field in the Child table <Student table in this case>.



Select Enforce Referential Integrity

- When the Cascade Update Related Fields check box is set, changing a primary key value in the primary table automatically updates the matching value in all related records.
- When the Cascade Delete Related Records check box is set, deleting a record in the primary table deletes any related records in the related table.
- Click Create and Save the Relationship.

1.4.2 Access Database of Objects

To understand how to use Access, you must first understand a few basic database concepts. A database is a collection of information regarding a certain topic. A database will help you organize this information in a logical manner for easy understanding. The database in Access is a term for the container that holds all the data and its associated objects. The seven main objects in Access include tables, queries, forms,

reports, pages, macros, and modules. Although some other computer database programs might call the object that actually holds the data a database, Access calls this object a table.

Access can only work with one database at a time, but in that database there can be hundreds of objects, such as tables, queries, and forms. They are all stored in one Access file. The heart of the Access database is the table. A new database is created to which Access assigns an **.MDB** extension and it will appear in the **Database** window.

Tables

(A table is used to hold the raw data of the database). You enter your data into tables. Next the table organizes this data into rows and columns. The table list is the default view when you open a database in Access.

Queries

A query is used to extract only certain information from a database. A query can select a group of records that fulfill certain conditions. Forms can use queries so only certain information will appear on the screen. Reports can use queries to print only certain records. Queries can be based on tables or on other queries. Queries can be used to select, change, add, or delete records in your database.

Forms

Forms can be used in a variety of ways, but the most common way is for data entry and for display. Data entry forms are used to help to enter data into tables quickly, accurately, and easily. Forms display data in a more structured way than a normal table does. You can change, add, delete, or view records from a table using a form. Display forms are used for the selective display of certain information from a given table.

Reports

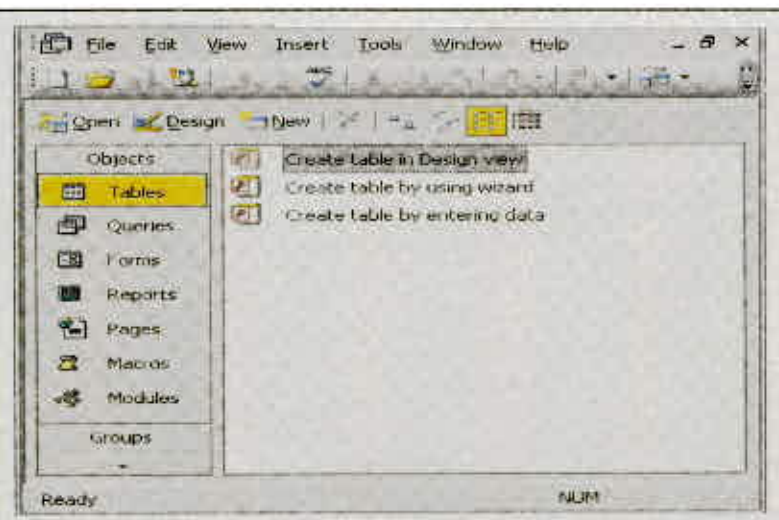
Reports present the data you select in a printed format. Reports can be based on tables to show all the data from the given table, or they can be based on queries to show only information that meets certain criteria. The reports can also be based on multiple tables and queries to show complex relationships that exist in your data. Access has many default reports that you can easily create to display your data in any way you might require.

Pages

Pages are new to Access 2000, and are more formally called Data Access Pages. Pages are HTML documents that can be bound directly to data in a database. These documents are very similar to Access forms, but they are designed to be viewed with Internet Explorer. One big difference between Data Access Pages and forms is that the pages are saved to a different file than the Access database, whereas forms are stored within your database file. This is because the Pages are designed to be used with an Internet browser (specially for Internet Explorer 5.0) and they make use of dynamic HTML.

Database Window

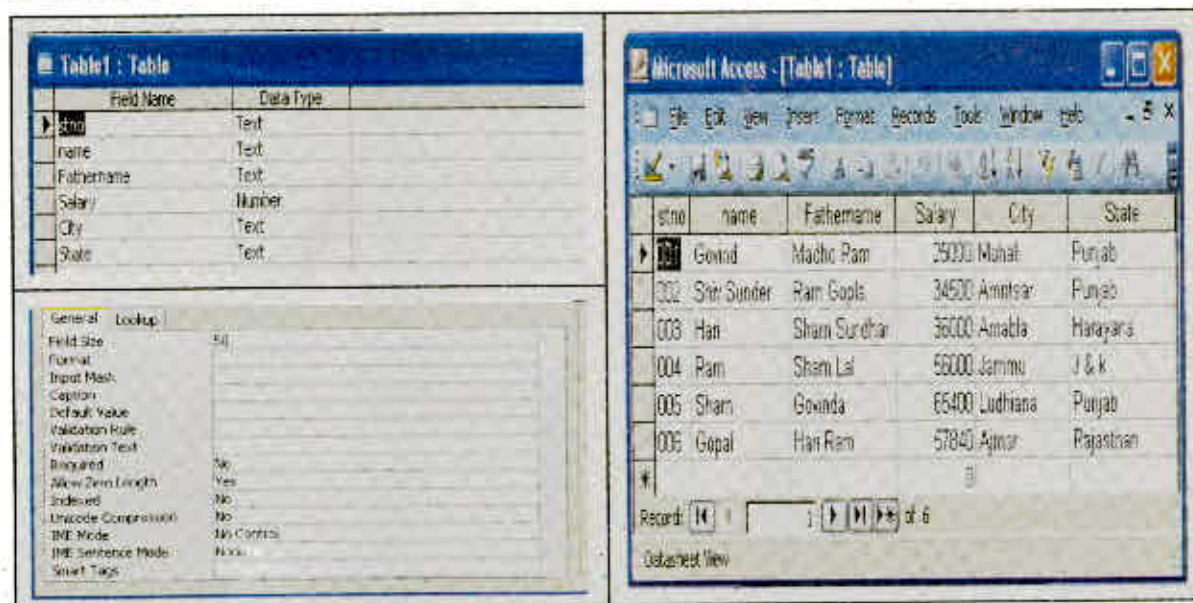
On the left appear the different **types of objects** that we can have in the database, (tables, queries, forms,...) and on the right, depending on the type of object selected on the left, Access shows us the objects of this type that are already been created and allows us to create new distinct objects.



Let us make a data entry screen to enter various fields using Access table and Access form. You can build a table using wizards or Design view. The following steps are used:

Step 1 : Create a database say db1, (this database acts like a folder where we can store the different components or objects of the access).

Step 2 : Build the structure of the table 1 and the corresponding records as shown



Now using this data, we will create a form for data entry screen.

Step 3 : In the database screen as shown below choose forms and then create form using wizards, and move the fields from the available fields list to selected fields List, then select the format of the form in which you want to display the data in the screen (columnar), after that select the style of the form with given option (here as shown, we have standard style)

Step 4: After that the form wizard asks you to give the title of the form, we have here **Data entry Screen** as the name of the form. Then click on the check button Open the form to view or enter information, then click Finish button the resultant screen will be :

The image shows two screenshots from a database application. The left screenshot is the 'Form Wizard' dialog box, which asks 'What do you want for your form?'. It has a 'Data Entry Screen' button and a 'Finish' button. The right screenshot is the 'Data Entry Screen' for a form titled 'Data Entry Screen'. It contains fields for 'idno', 'name', 'fathername', 'salary', 'city', and 'state'. The 'idno' field is highlighted, and the 'Records' bar at the bottom shows 14 records out of 6.

1.5.0 Review on Programming concepts

A computer indeed is a very useful machine. It makes life easier for us by helping us to perform a number of complex tasks quickly and accurately. With the help of computer, we can draw pictures, edit photographs or play games, etc. these are performed with the help of different types of programs.

Firstly, the task to be performed or the problem to be solved should be clearly defined and understood. This calls for determining what output is required and what input will be provided. Secondly, the methods to solve the problem must be ascertained. Then, a broad outline of the program should be developed. After this the actual programs must be written. Finally, the program should be tested to check whether it gives the desired results or not.

1.5.1 The various stages of a program development cycle are as follows.

- Analyzing the Problem
- Development of Solution
- Coding the Solution
- Testing the Program

1.5.1.1 Analyzing the Problem

It means to find out the requirements of the program. In other words, to determine exactly what you want the program to do. Besides deciding upon the required Output and given Input of the program, we have to work out on a logical process for achieving the desired results.

1.5.1.2 Development of Solution :

The most important task in developing a solution is the development of logic that solves the problem. This requires creation of a step-by-step procedure for solving the problem. Such a scheme is commonly referred to as an algorithm written in simple English. Then this algorithm is later converted into a flowchart, which is a pictorial

representation of the algorithm. Besides the algorithm pseudo code is also considered as a logic development technique, because the flowchart is a graphical representation of an algorithm. It makes use of the different symbols to indicate different operations. The various symbols and the operations they represent are explained below.

Input / Output Box

It is represented by parallelogram.

Data Flow Lines Processing

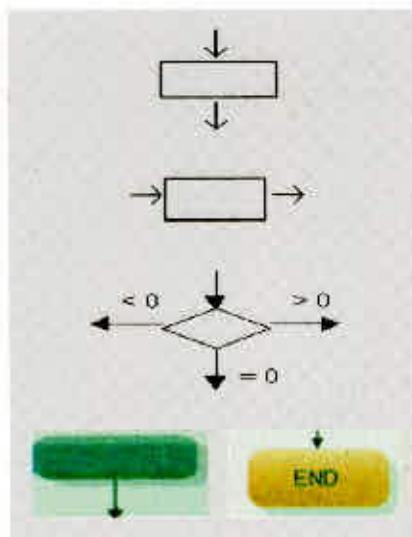
It is represented by the rectangle. Whenever data is to be processed, it is written inside the processing box. Only one flow line can enter or exit from this box. This is represented by the arrow (\rightarrow / \leftarrow) as shown in the figure on left side.

Decision Box

It is represented by a diamond. In this only one flow line can enter in the decision box and two or three flow lines can leave it to test some possible conditions as shown in the figure.

Terminal

With this symbol, only one flow line can be use as shown in the figure.



Now using the Flow chart let us find the largest of the three numbers.

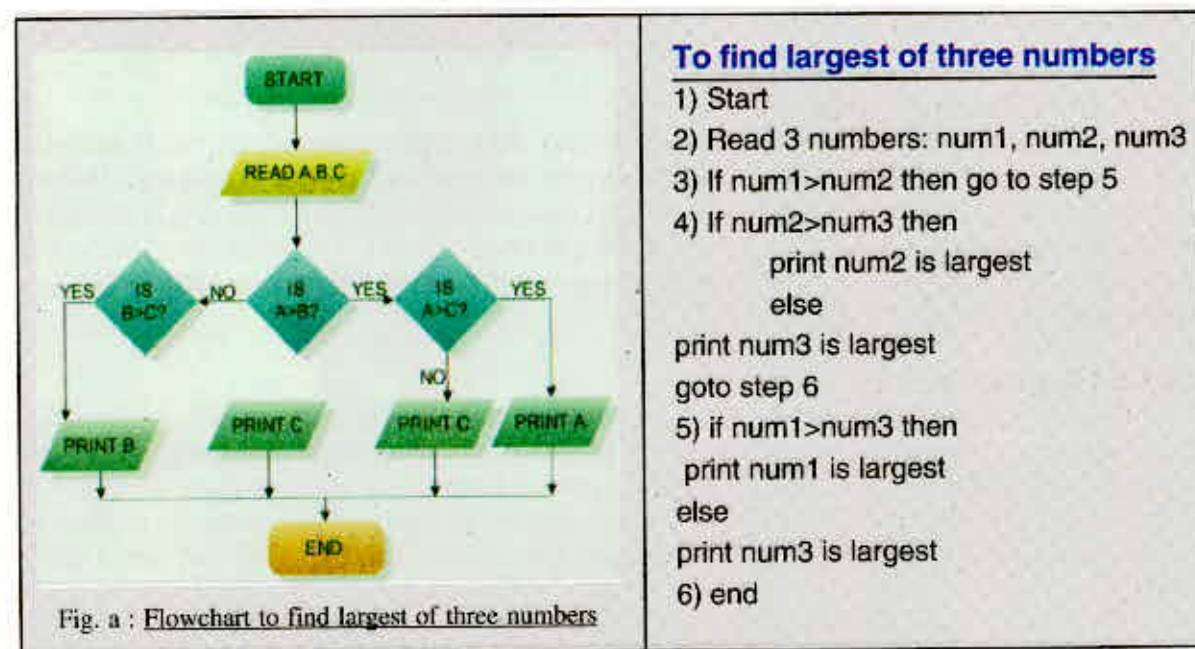


Fig. a : Flowchart to find largest of three numbers

Introductory Examples of Flowcharts and Pseudocode

Calculate Pay - sequence

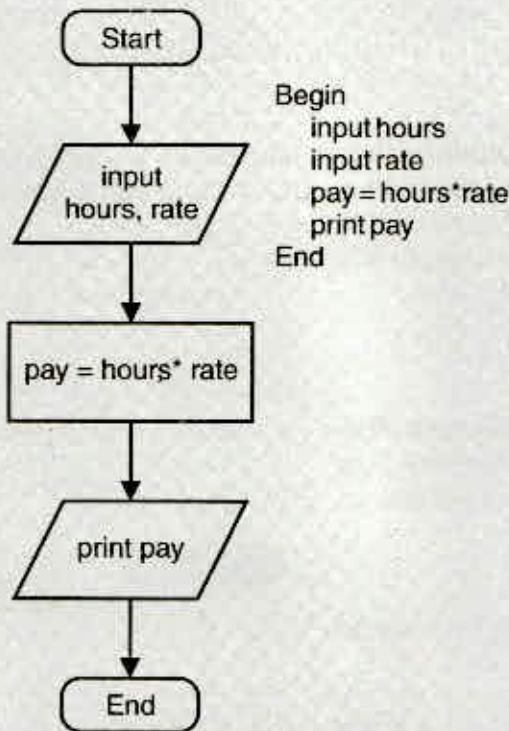


Fig. b

Sum of 2 Numbers - sequence

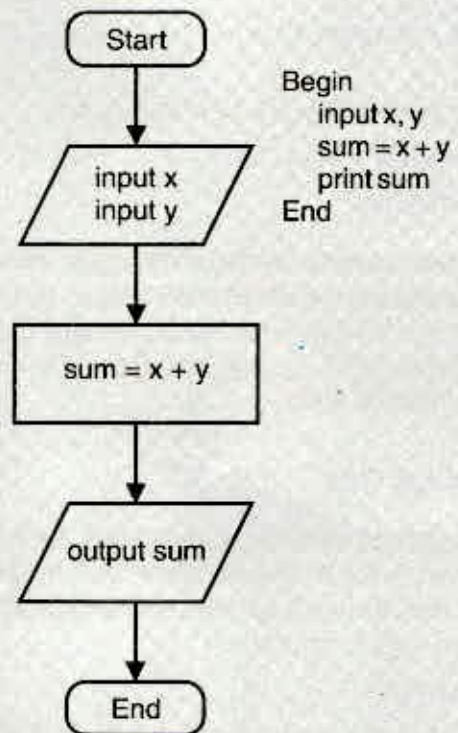


Fig. c

In the Fig. b, salary is calculated and in Fig. c, the sums of the two digits are shown.

1.5.1.3 Development cycle

The third stage is coding the solution or writing the program based on the flow chart and pseudo code. Coding starts with programmer working on the computer. Before proceeding to write a code, we must ensure that the appropriate software is installed on the computer. For example, before writing a program in C++, we must ensure the Borland C++ / Turbo C++ Software, which is required to develop program in C++, is installed in the computer system.

1.5.1.4 Program testing

This is the last state in the program development cycle. Once a program is developed. It is tested to ensure that it is free of bugs (errors) and is capable of solving the given problem. At times it is possible that a program does not show any compile- time or run - time errors but still fails to give the desired output. In such cases, through testing of the program and a review of all the stages should be done to find out what went wrong.

1.5.2 Elements of PROGRAMMING Language

Having understood what a programming language is and also the various stages of a program development cycle, let us move on to understand the various elements of a programming language.

Variables serve the purpose of string data and indentifying each item of data using a descriptive name. Once data are stored in variables, they can be manipulated easily. In order to manipulate the data contained in variables, a computer needs a list of instructions to perform a program.

The statements in a program describe how the data stored in variables can be manipulated. At times the manipulation of data involves performing some calculations on the variable - complex statements containing operators can be used for such purposes. Some programs need to make decision based on calculated results. This is accomplished by a list of statements called control statements. these statements control the sequence in which the instructions of the program are executed. Program contain many lines of code. Small groups of code that performs operations that you define are called functions or sub-routines. Arrays and structures are used to organize the variables by grouping them together as required. In fact many programmers consider the choice of data structures and data organization to be first steps in the design of any program.

Exercise

Q1 Answer the following questions.

- 1.1 What is the difference between the Application software and system software ?
- 1.2 Why Ms Access is called Relational Database ?
- 1.3 Differentiate between if() and countif() functions in excel.
- 1.4 What are the required steps to develop a good program ?
- 1.5 What is the use of tags in HTML ?

2. Fill in the blanks.

- 2.1 What is the important document of www _____ ?
- 2.2 The address of the 27th column and 30th row in the worksheet is _____ ?
- 2.3 The easiest way to design a form is through _____ ?
- 2.4 Flow charting belongs to which part of program development cycle _____ ?
- 2.5 Ms Office software belongs to _____ Category.

3. State whether True or False.

- 3.1 HTML tags are enclosed in curly braces.
- 3.2 If you enter 12+24 in a cell, excel will display 12 + 24 in a cell.
- 3.3 Before designing a form, data table must be present ?
- 3.4 While Analyzing the problem, output of the program should not be considered.
- 3.5 To make a formula in Ecel, 5 types of mathematical operators are used.

Answer to objective questions.

Q	.1	.2	.3	.4	.5
2	Web Page(s)	AA30	Wizards design	Developing a solution	Application S/w.
3	F	T	T	F	T

	Chapter
Programming in "C" Language	2

2.0 Introduction

C Language was designed by Dennis Rictchi at Bell laboratories in the early 1970s. The earlier version of C called B was written by Ken Thompson who adopted it from Martin Richard's BCPL (Basic Combined Program Language). Dennis Ritchie Studied BCPL, improved and named it C which is the second letter of BCPL.

2.1 Distinctive Features of C Language

Calling C as a language is a mistake as it is really a symbolic instruction code, a set of commands that perform actions on a computer. Writing text to display, adding two numbers, transferring data to disk-all of these actions and countless others can be programmed using C statements constructed according to C's rules and regulations-in other words, its syntax.

- **The C language is often described as a "middle-level" language**
- It permits programs to be written in much the same style as that of most modern high-level language, such as FORTRAN, COBOL, BASIC, PLII, and Pascal. It is possible in C to deal with the machine at a fairly low level.
- **Besides above C is a high-level procedural language.**
When using a procedural language, a programmer writes explicit directions for a computer about the steps it has to perform. The programmer must understand the meaning of the data, the step being performed and the result. A high level language frees the programmer from work of machine instructions.

2.2 The C Character Set

Similar to the natural language, computer language have well defined character set, grammar rules, which is often called as syntax etc. There are two sets of character set in C language.

Following shows the valid alphabets, numbers and special symbols allowed in C.

Alphabets	A, B,Y, Z OR a, b.....y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special Symbols	~ ` ! @ # % ^ & * () - _ += \ { } [] ; : " ' < > , . ? /

i) Execution or Escape character

The charcters on the key board can be printed or displayed by pressing the key. But some characters such as line feed, form feed, tab etc. cannot be printed or displayed directly. C provides the mechanism to get such characters that are invisible or difficult to get characters through execution characters. these are presented by a back slash (\) followed by character as shown in the table on next page:

Escape Character	Meaning	Escape Character	Meaning
"\n"	New line	"\r"	Carriage return
"\t"	Tab	"\""	Back slash
"\f"	form feed	"'"	Single quote
"\b"	Back space	"\""	Double quote

II) White Space Character

Any character that produces blank space when printed is called white space characters are spaces, tabs, new lines, and commnets.

2.3 Structure of a C Program

Different programming language have their own format of coding. The basic components of a C program are

The Basic structure of a C program

- void main ()
- pair of curly braces { , }
- declarations and statements
- user-defined functions

Preprocessor statements

Global declarations

void main()

{

declaration;

statement;

}

User defined functions

2.3.1 Head files

Files that are placed at the head before main() of a C program are called header files normally having .h extension.

Any line in a program that starts with # is an instruction to the compiler, not an actual statements in C language. The header files are entered into the sources program via a #include statement so the #include statement is a preprocessor statement and must be written at the beginning of the program. Each header file contains declaration for certain related library function.

2.3.2 Preprocessor statements/directives

At the time of compilation itself some processes can be done in C. The command which invoke such processes are called preprocessor directives. These can optionally be present in the program. The preprocessor directives are executed before the C program is compiled and are useful in peforming the following actions.

- Substitutions of values
- Including files and
- Conditional compilation.

There are three types of preprocessor dirctives available with C. They are inclusion (# include), macro substitution (#define) and conditional (#if) directive. These statements direct the preprocessors to include header files and symbols constants into a C program e.g.

#include<stdio.h>	: for the standard I/O functions.
#include<math.h>	: for certain mathematical functions.
#include<string.h>	: for string manipulation functions.
#include"test.h"	: for file inclusion of header file test.
#include NULL 0	: for defining symbolic constant, NULL = 0.

2.3.3 Global declaration

Variables or functions whose existence is known in the main function and other user defined functions, are called global variables, and their declaration are called global declarations.

Main () as the name itself indicates this is the main function of every C program. Execution of C program starts from main(). It should be written in lowercase letters and should not be terminated by a semicolon. It calls other library functions and user defined functions. There must be one and only one main () function in every C program.

Global variables are defined above main() in the following way :-

```

short number, sum = 0;
int begnumber, bigsum,; } Global Variables
char letter;

void main()
{
    .....
}

```

It is also possible to pre-initializes global variables using the = operator for assignment.

- **Brace**

Every C program uses a pair of curly braces { ,}. The left brace indicates the beginning and the right brace indicates the end of main or a user defined function.

- **Declarations**

It is a part of the C program where all the variables, arrays, functions etc. may be initialized with their basic data types.

- **Statements**

these are instructions to the computer to perform specific operations. they may be I/O statements, arithmetic statements and other statements. The statements to be commented on must be enclosed within /* and */ comment statements are not compiled and executed.

A sample C program is given below:

```

#include <stdio.h>
main( )
{
    printf ("Welcome to C\n");
}
Output : Welcome to C

```

- The first line tells the compiler to include standard I/O header file to perform reading and printing the data.
- The second line is a the main(), the main function of the C program. the body of the program contains.

```
printf ( "Welcome  
to C\n");
```

When the statement is taken for execution, main() called the print() statement so that <stdio.h> could be included in this. Print() "Welcome to C" prints on the display.

2.3.4 Compiling an execution of C Program:

Compiling a C program means translating it into machine language. C compilers are used for this purpose; The C program to be compiled must be typed by using an editor.

C compilers are available with and without editors. The environment where you find the compiler, editor, debugging tool, linking facilities, tracing and testing tools is called integrated Development Environment (IDE).

Example : Turbo C (TC), Borland, C etc.

The basic five steps in the successful execution of a program are:

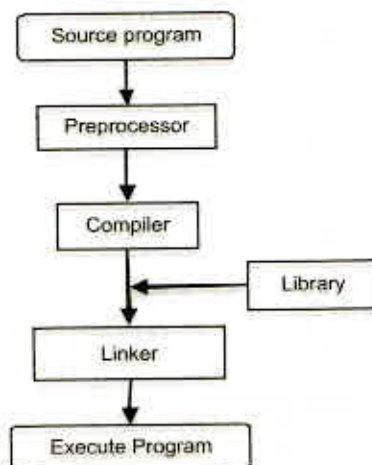
1. Creating a program file.
2. Saving the program.
3. Compilation.
4. Linking system library function.
5. Running (executing) Program.

The procedure used in compiling and executing a C program differs from one operating system to another. On DOS platform (under windows OS), if you are using the C/ C++ editor, the following steps help you in entering and compiling the C program.

2.4. Use of Editor

- To load and execute TC tutor editor type tc.
- Press F3 key and type filename (the program name say abc.c) & then press enter key.
- Now editor is open & type the C program. To save it press the key F2.
- For compilation of the program press ALT + F9 keys simultaneously and select build all options from the drop down menu from the compile option on the menu bar. Then press the ENTER key to create an obj (object) file of your program.
- To run the program Press ALT +R simultaneously & select run option from the drop down menu & press ENTER key for program execution.
- To display the output on Screen Press ALT+F5.
- To go back to the edit screen press any key.

Stages of Compilation & linking



2.5 Function

In C language functions are subprogram, that are written for performing specific tasks and they are complete in itself and independent, since these have their own declaration of variable statements whose beginning and ending is executed in the functions. In C there are two types of functions.

- Built in Functions / Library Functions.
- User defined Functions.

2.5.1 Built in Functions

These are always available in C language and can be called any time in the program. For the use of available functions, the function which is required to be used is called by writing its name followed by writing the arguments in the brackets.

2.5.2 User defined Functions

These are subprogram. Generally a subprogram is a function and they contain a set of statements to perform a specific task. These are written by the users; hence the name user defined functions. They may be written before or after main () functions.

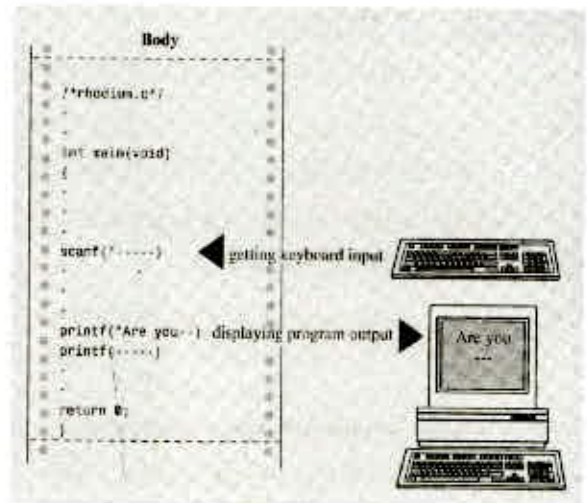
2.6 Formatted I/O Functions (scanf() & printf() functions)

In formatted I/O category, we have scanf() function for input and printf() function for output.

2.6.1 printf Function

This is a C library function that allows us to output data in fixed format. The printf() function prints output to stdout, according to format and other arguments passed to printf() as shown in the fig.

Are You



String Format

The string format consists of two types of items (1) characters that will be printed to the screen and (2) format commands that define how the other arguments to printf() are displayed. Basically, you specify a format string that has text in it, as well as "special" characters that map to the other arguments of printf(). Its syntax is

```
printf("format string", arg1, arg2, arg3.....,
      argn);
```

Where the format string tells the function where and in what form the result is to be displayed and the expression that we wish to evaluate.

<format string> could be,
 %f for displaying floating point values
 %d for displaying integer values
 %c for displaying character values.

arg1, arg2, arg3....., argn may be constants, variables, expressions etc.

Example 1. Suppose you want to print on screen that the value 7 times 12 is 84 using printf()

```
printf("The value 7 times 12 is %d", 7 * 12);
```

This statement pass two arguments to printf(). the first is the message " the value 7 times 12 is %d" and second is the value of the expression 7*12. the first argument passed to printf() must be message. A message that also includes conversion control sequence, such that %d is termed as a format string. When the printf() sees conversion control sequence its string, it substitutes the value of the next arguments in place of the conversion control sequence. Since the next argument is the expression 7*12 wich is 84, it is this value that is displayed. So the output displayed is

```
The value of 7 times 12 is
84
```

To use this function, let us consider the following example1 program for calculating a simple interest with $si = p * r * t / 100$; for given values of principal, rate, time. Since p & t are in integer, while r, si may be in decimals, we write them as variables & before

using them, they must be declared in declaration section. Also if rate of interest is known say $r = 5.5$, then if we assume that $p = 1000$ & $t = 3$ years.

<p>Example 2.1</p> <pre>int p,t; p = 1000, t = 3; declaration are float r, si; r = 5.5; Statement 1 is $si = p*r*t/100$; Now to display Simple interest, we write</pre> <p>Statement 2 <code>printf("%f",si);</code></p>	<p>Example 2.2</p> <p>The structure of C program is</p> <pre>void main() { declaration; statement 1; statement 2;..... statement n; }</pre> <p>Every C statements ends with;</p>	<p>Example 2.3</p> <pre>#include<stdio.h> /*To print values on screen*/ void main() { int p,t; p = 1000, t = 3; float r, si; r = 5.5; si = (p*r*t)/100; printf("%f",si); }</pre> <p>Output will be : 165</p>
--	--	--

Let us consider the different forms of printf()

- To display the values of each variable/ constants

Statement	Output
<code>printf("%d %f %d %f", p,r,t,si);</code>	1000 5.5 3 165
<code>printf("Simple interes = Rs. %f",si);</code>	Simple interest = Rs 165
<pre>printf("Principal = %d\n Rate = %f",p,r);</pre> <p>Where \n inserts blank line between two output.</p>	<p>Principal = 1000</p> <p>Rate = 5.5</p>

Example 3

To print a value giving field width specification

```
#include <stdio.h>
Void main(0
{
  float a = 123.465;
  printf("%7f %7.3f %7.1f",a,a,a);
```

When the program is executed the output will be

123.465000 123.456 123.6

Example 4

```
#include<stdio.h>
void main()
{
    int a = 15;
    float b = 34.76;
    char c[2] = "Rs"
    printf("%d %f %s",a,b,c);
}
Output will be 15 34.76 Rs
```

2.6.2 Scanf ()

It allows the user to input the Numeric, Character and String type data in a fixed format. This function is available in stdio.h which is given at the start of the program.

Its syntax is : **scanf("format string", list of address of variables);**

Where the format string contains the format specifiers that begin with the character '%' and are separated by space or comma, it should be given within double quotes.

The list of addresses of variables are used so that scanf() function can place the data received from a standard input device say keyboard. The address of a variable is obtained by using address operator, character "&" and pronounced as address of the operator or pointer operator.

Example 5

Consider the scanf("%d %c", &bp, &city)

Here the scanf function allows four parameters, parameter %d symbol implies that one integer value should be read. The third parameter &bp tells that the integer value accepted will be stored in the variable name bp. Second %c to read, & city stored character type data in variable.

On execution, the input data must be supplied strictly according to the specified format string otherwise results can be very strange. Its syntax is:

Example 6

```
int x;
float y;
char c;
scanf("%d, %f, %f",&a, &b, &c);
Assume that the input value for a = 10
                        b = 32.65
                        Character c = Z
```

The data must be entered or typed in the following format 10, 32.65, Z but if it's entered as 10, 32.65 Z (without space), result will be very strange.

Example 7

- **Scanf("%d", &int_variable);**
If the input value is 55.75

The scanf() funtion will take it is 55 only

- scanf("%f",&float_variable);
if the input value is 45

The scanf() funtion will take it as 45.0

Now let us modify the **Example 2.3** given on previous page. Now it will ask the user to input value then it display the value of principal (p), time(t), rate(r) separated by new line.

Explanation : The character '\n' inserts a new line so that subsequent output can continue from the next line. So the string output up to this stage appears as:

```
/* calculating the simple interest*/
void main()
{
    int p, t;
    float r, si;
    printf("\n Enter value of principal p, time t and; rate r\n");
    scanf("%d %d %f", &p, &t, &r);
    si = p*r*t/100;
    printf("%f", si);
}
```

Enter value of principal p, time t, & rate r

1000

3

5.5

Output

165.000000

After that these input values are assigned to their respective variables in the equation $si = (p * r * t) / 100$, where si is calculated, it was then displayed through final printf(). The output will be as given in the previous example.

Example 8

```
//This program accsespts the temperature in Fahrenheit and
// converts into Celsius, formula is  $C = (f-32.0)/1.8$ 
```

```
#include<stdio.h>
void main()
{float ct, ft;
printf("enter the temperature in Faheinheit\n");
scanf("%f",&ft);
ct = (ft-32.0)/1.8;
printf("Fahrenheit temperature = %6.2f\n",ft);
printf("Celsius temperature = %6.2f\n",ct);
}
```

Output

Enter the temperature is farheinheit

34.20

Farheinheight temperature

34.20

Celsius temperature

0.12

2.7 Starting with C programming

Here we are going to use Windows Platform and key board strokes

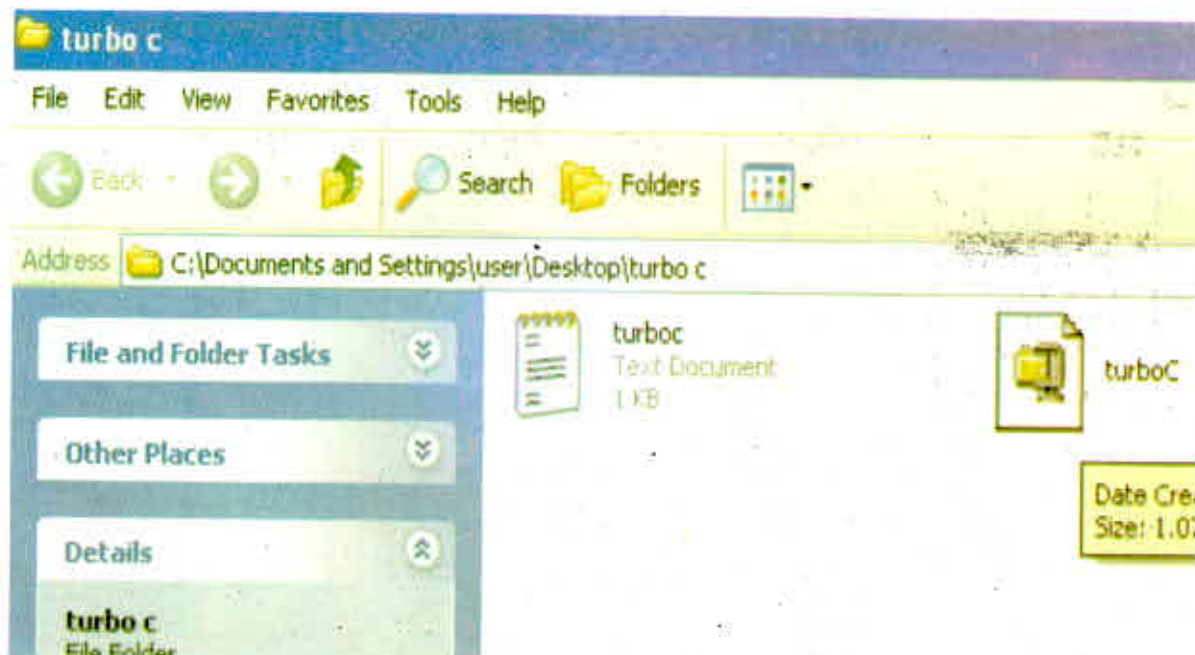
On Windows platform, we are using Turbo C; the following steps help you in entering and compiling your C program.

2.7.1 Installing Turbo C

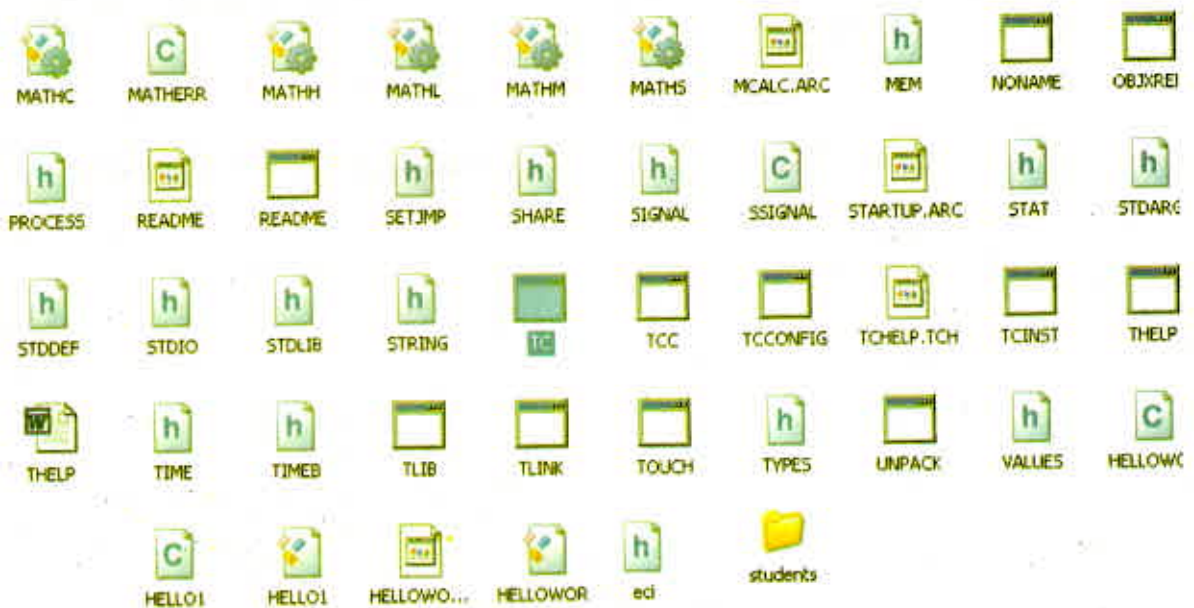
Figure showing Turbo C folder on desktop after downloading from CD/Internet.



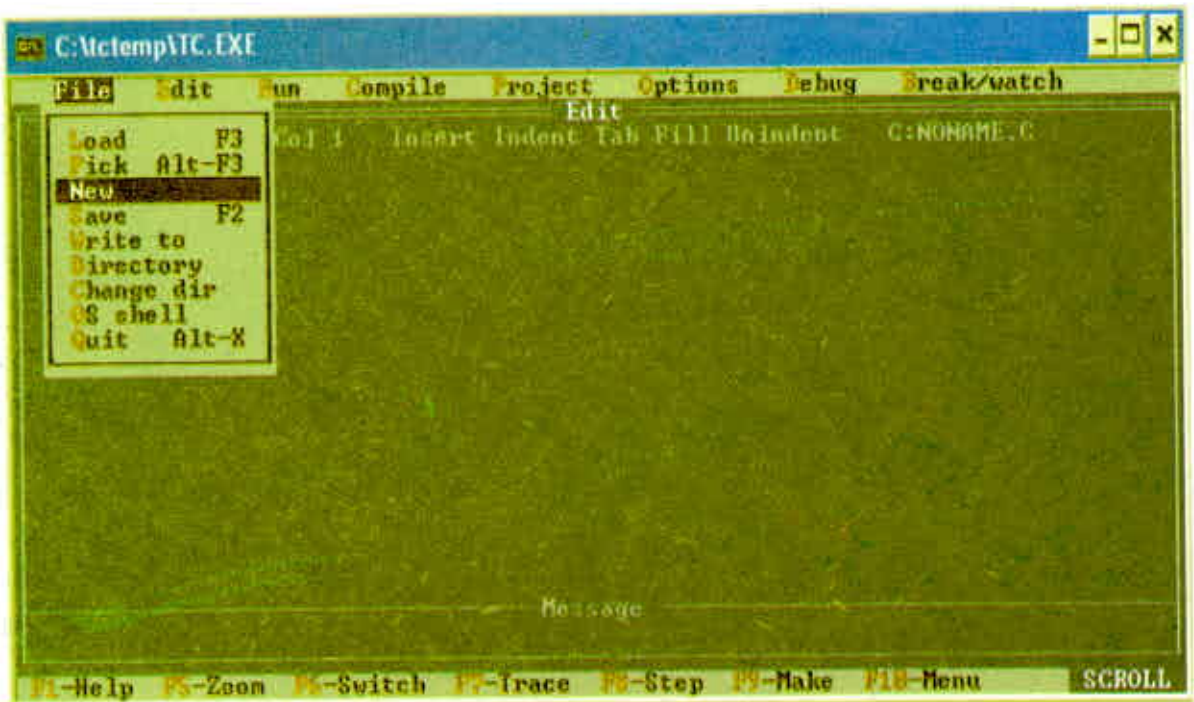
After opening the Turbo C folder, the following two icons appears as shown below:



Go to the drive (by default C :) where the folder named **tc-temp** was opened. Open the **tc-temp** folder and click on TC executive file (TC.exe) as shown in fig.



1. Double click on TC executive file (TC.exe). This will open window as shown
2. Now open the file menu by pressing Alt + F and select now to create a new program.

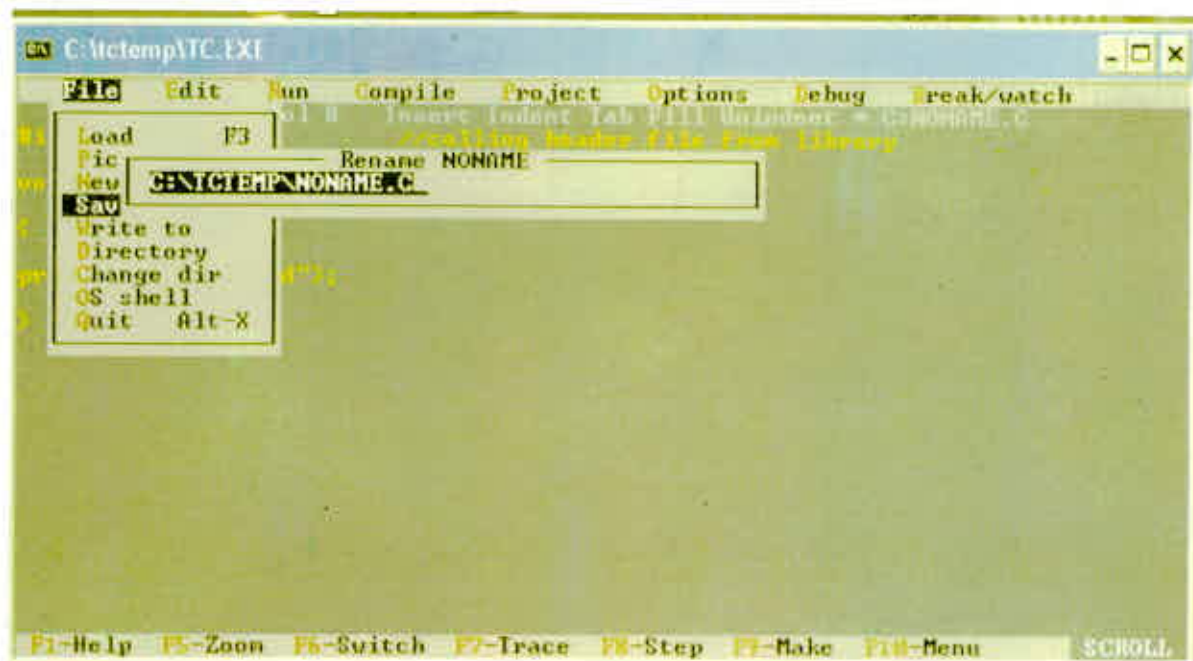


Now type the program as shown in the below figure.

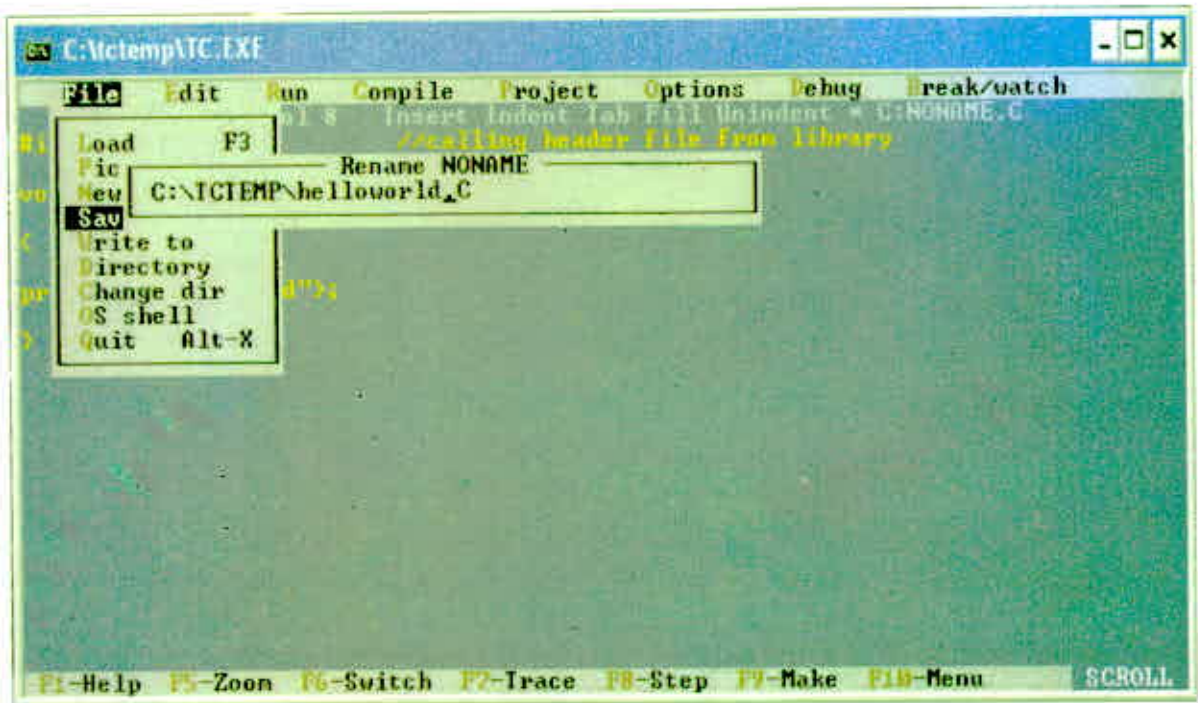


After the completion of typing program, you have to save the program by pressing ALT + S then select **Save** option from the menu and press enter.

Note : That extension C to every program comes automatically. The program can be saved in an individual folder as shown with folder namd student or in Turbo C folder itself.



Default Saving shown in these figure:



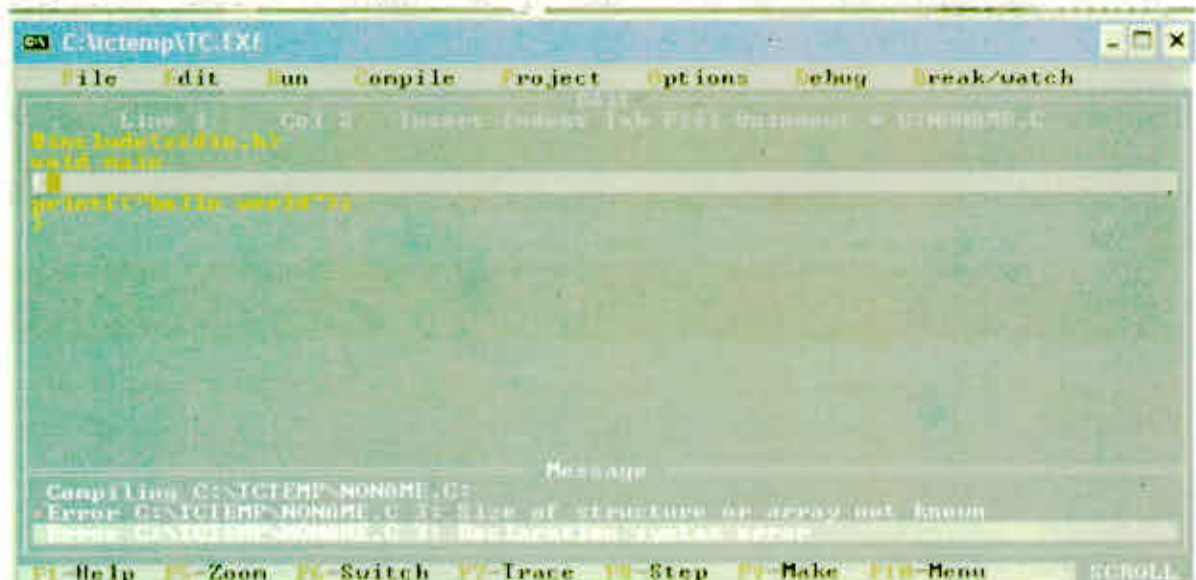
With user folder "student"



2.7.2 Compiling the program and executing the program

To complete the program follow the given instructions.

Press Alt + F9 keys simultaneously, if the compiler finds some errors in the program, then it will show you following errors message on the window screen as shown below:

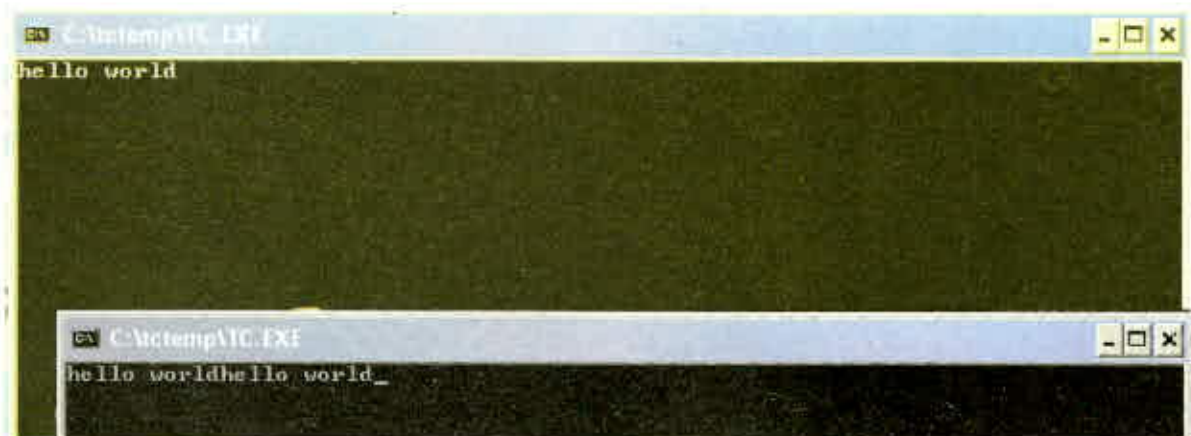


For example in the above picture two errors are shown. Since, in this program main is a function with which brackets are necessary, whereas in the above program these are missing.

For successful execution remove these errors & compile it again. If there are no errors, then it will show zero errors (shown below), that means that this program is ready for execution.



To see the output press Alt +F5, This will show you output on the screen as shown in the figure below:



So include the header file <conio.h> and clrscr() after initialization of variable along with stdio.h

Programming in C Language

C is general purpose structured programming language. It is overcome of two programming languages B & BCPL. All three programming language were developed at bell laboratory, USA. It is written by Dennis Ritchie & Brain Kerninghan in 1978.

C is both system program and application program development tool. It lies between high and lower level languages so it is also called as Middle level language. The C character set consists of alphabets, digits, and special symbols. There are 91 character in C. The C tokens are the basic units of C program and they are categorized into keywords, identifiers, constants, string, operators and special symbols. All the keywords in C have one or more fixed meaning and they cannot be change. The keywords must be written in lowercase letters. the identifiers cannot be changed. The identifiers are the names given to the program elements such as variables, arrays and functions. All the variables must be declared for their types before they are used in the program. A constant is a quantity that does not change during the execution of a program. A variable is a quantity that changes during the execution of a program. Each instruction end with character semicolon";" & comments can be placed anywhere. All the words in a program line must be separated from each other by a least one space or tab. or punctuation. Every C program requires a main () function & its place is where the program execution begins.

Function

The execution of a function begins at the opening brace of he function & ends at the corresponding closng brace. We must make sure to include header files using # include directive when the program referes to special names & functions that it does not define.

Compiler directives

Compiler directives such as **define** & **include** are special instructions to the compiler to help it ti compile a program. They do not end with semicolon. The sign "#" of compiler directive must appear in the first column of the line.

When the braces are used to group statements, make sure that the opening brace has a corresponding closing brace.

To read data from the input device input statements are used and to print the processed data into the output device, output statements are used. These input/ output (I/O) statements can be formatted or unformatted. In unformatted I/O statements no specification of data types and the manner in which they are read in or written out is mentioned. But, Formatted I/O statements specify what type of data is being input or displayed and in what was it should be.

Scanf() and printf() are the two in-build functions use to read and print the data. They are used for all types of data like int, char, float, double, etc. In general, it is not possible to read multi-word string with scanf() function.

Using escape sequences, we can format the output as per our requirements.

Exercise

Q.1 Multiple Choice Questions.

1.1 Which of the following character is used to terminate an instruction in a C program ?

- a) , (comma) b) : (colon) c) ; (semicolon) d) . (period)

1.2 How many main functions can be used in C program ?

- a) 2 b) 3 c) 1 d) any number.

1.3 Which of the following is used to enclose the body of a function ?

- a) [] b) { } c) () d) None of these.

1.4 For building blocks of C language we require ?

- a) Character b) Data type c) names d) Expression & statements e) all of these.

1.5 Which is not a keyword in C ?

- a) const. b) main c) sizeof d) void

1.6 Which option is the best possible in case of C language that it ?

- a) has high efficiency b) is system independent c) has limited data types d) all of these.

1.7 The math library is set up for the user by the file ?

- a) float.h b) limits.h c) math.h d) time.h

1.8 Which of the following printf conversion character is used to display the data item as a signed decimal integer.

- a) d b) f c) i d) u.

1.9 The null character is represented by ?

- a) \n b) \0 c) \o d) \t

1.10 Which of the following is not used in C language.

- a) a digit b) an integer c) a character d) a word.

Q.2 Do the followings statements ?

2.1 What is the effect of executing the following statements.

```
printf("\nOne\n Two \n Three \n");
```

2.2 Write the printf() statement that will display the following on your screen ?

This is a test 1 2 3

123.33

These are the Characters a b c

2.3 Write the declaration statement that will declare the following variables as indicated ?

up, down

as integers

first, last

as single precision floating point

C

as a character.

2.4 Will the following program work ? If not state errors and correct them ?>

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

```
main () { printf("How many persons !");}
```

2.5 What is wrong with the following segment of C code ?

```
printf("c", "o", "m", "p", "t", "e", "r", "\n");
```

2.6 Point the errors in the program ?

```
name{}          \*This is one great program; stand buy ...../*
```

```
(
```

```
printf('I think I am getting hungry.\n');
```

2.7 Using the printf() function, write a C program that prints name on the one line, address on second line, city, state on the third line ?

2.8 Use the printf("\n sum = %d,\nagerage = %d\n", a+b+c, (a+b+c)/3 to print sum and average of three numbers ?

2.9 Correct the errors int he program to convert meter into kilometer

```
main( )
```

```
{ float mtrs, km;
```

```
printf("Enter the value in meters");
```

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

```
km = mtrs/1000.00
```

```
printf("\b The value of metre(s) %7.2f is converted into %7.2f kilometer (s)", mtr,km);}
```

2.10 Use $c = 5.0/90 * (f-32.0)$; to convert temperature to Fahrenheit to Celsius. Hint step ?

C and F are declared to be floats

Fahrenheit temperature is accepted

Computes the celcius value w.r.t. Fahrenheit

Prints the computed celcius.

Q.3 State whether True or False.

3.1 C Language is case sensitive ?

- 3.2 If a program contains logical errors, it will not be compiled ?
- 3.3 Blank space is called White space characters ?
- 3.4 Null string is represented by " " ?
- 3.5 C is a simple, versatile, more expressive general purpose language ?

Q. 4 Write an appropriate word for the following.

- 4.1 C language is of which level language ?
- 4.2 Each and every statement in C language end with ?
- 4.3 Character with value 0 is ?
- 4.4 \t represents a special character in C ?
- 4.5 Compile time directive ?

Q. 5 Answer the following questions.

- 5.1 What are the limitations of C language ?
- 5.2 Mention the application areas of C language ?
- 5.3 What are preprocessor directives or statements ?
- 5.4 What do you understand by global declaration ?
- 5.5 Give the general structure of C program ?
- 5.6 What are the basic steps in the execution of a C program ?
- 5.7 What are the stages of compilation and linking ?
- 5.8 What is the difference between the printf() and scanf() ?
- 5.9 Why printf() and scanf() are called formatted input/output functions ?
- 5.10 What is the purpose of & in scanf() ?

Answer to objectives questions

Q	.1	.2	.3	.4	.5	.6	.7	.8	.9	.10
1	c	c	b	e	b	d	c	c	b	c
3	True	False	True	True	False					
4	Middle	semicolon	'\0'	tab	#define					

3.0 Introduction

In this chapter will learn about Constant, Variables and Data types, in which we will learn to know about the rules to define names of variables and constant with the help of variable types such as short int, long int, double, long double.

3.1 Constants / Literals

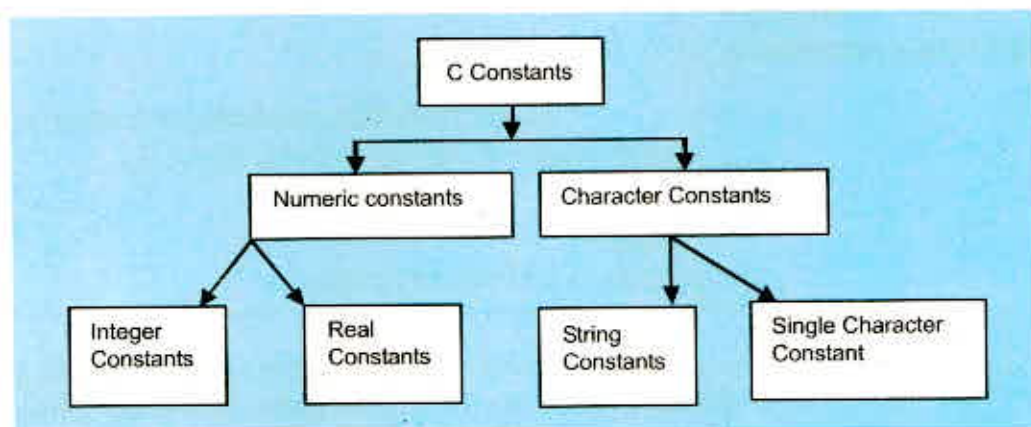
A constant in c is a quantity that does not change. The alphabets, numbers and special characters when properly combined in expression than it changes its forms to constants, variables and keywords. This quantity can be stored at a location in the memory of the computer. A variable can be considered as a name given to the location in the memory where a constant is stored.

Example

$y = 3.6 * x + 8.3$; Here 3.6 & 8.3 cannot change, they are called constant while quantities X & Y can vary or change hence are called variables.

3.1.1 The types of C constants

The C constants are divided into the following two categories.



3.1.1.1 Numeric Constant

It is made of a sequence of numeric digits with optional presence of decimal point (.)

- **Integer Numeric Constant**

This is a sequence of numeric digits without decimal point. The general form of integer constant is **Sign, Digit**.

Where **Sign** : is optional

“+” for positive nos, “-” for negative nos.

Digit : a sequence of digits

Example

+23, - 3421, 9999 etc.

- **Real / Floating point constant**

A sequence of numeric digits with numeric digits

Example

19 is a integer Constant
14.78 is a Float Constant

3.1.1.2 Character Constants

Character constant are integer values and are given in single quotes. The values of character constant are available in ASCII character set.

Example

Character Constant	ASCII Value
'A'	65
'B'	66
'C'	67

For more character constant values refer ASCII table.

The integer and character data conversion is shown in the below given program.

```

#include<stdio.h>
void main()
{
    Char c; int i;
    i = 1;
    c = 1; /*this is allowed*/
        /*c holds the character whose ASCII value is 1 */
    c = 'A'
    i = c;
        /* i holds the ASCII equivalent of A which is 65*/
    printf("value of c is %c and i is ^d\n",c,i);
}

```

Output
Value of C is A and i is 65.

3.1.1.3 String Constants

These are ASCII character in a sequence and are given within / surrounded by double quotes. This can also be defined as an array of character constants whose last character is \0 which is automatically placed at the end of the string by C compilers.

Example

"Is a Null string"
"This is C"
"x + 2"
"2006"

3.2 Types of C Variables

Variables

The quantity that changes its values during the execution of a program is called variable. They represent a particular memory location where the data can be stored.

Example

area, name, city, age, etc.

Invalid Variables	Correction	Valid Variables
RUN;	Include a punctuation character	marks
1st term	Start with a number	Total_marks
-	Underscore alone	Gross_Salary_2007
Net-pay	includes a punctuation character	area_of_circle()
Net pay	Include space	num[20]
Switch	C Keywords	
\n	Starts with punctuation character	
There are other types of variables that are used in declaration statement in C		int si, m_hra; float basic_sal; char code;

3.2.1 Delimiters

A delimiter is a symbols that specify the limits of basic elements of the program, hence these are called delimiters or separators. These constants separate the variable and statements in a program but they do not specify any operation to yield a value.

C language sentences, labels arrays, etc. are separated by special characters and are called as delimiters. Some of them are given below;

Hash	#	Pre-Processor directive
Comma	,	Variable delimiter in variable list
Curly braces	{ }	Used to block C sentences
Square brackets	[]	used with arrays
Parenthesis	()	used in expressions
Colon	:	Label delimiter

3.2.2 Declaration / initialization of Variables

All the variables must be declared before they are used in program. The basic purpose of declaring the variable is to reserve the amount of memory required for these variables.

The declaration is made in the declaration part of C program.

Its syntax is:

Data_Type
Variable list;

3.2.2.1 Data type

These are the basic data type such as int, float, char, or double

3.2.2.2 Varlist

One or more variables of the type data_type and, these variables must be separated by commas.

3.2.2.3 Semicolon

It is a delimiter of the declaration.

3.2.2.4 Assigning values to the variables.

The processing giving values to the variables are called assignment of values and are done via assignment operator "=".

Its syntax is : **Variable_name = value;**

Variable_name

The represents the memory location where "value" should be stored.

Example

to $\left. \begin{array}{l} \text{int i = 0;} \\ \text{char ch = 'a';} \end{array} \right\}$ are equivalent $\left\{ \begin{array}{l} \text{int i;} \\ \text{char ch;} \\ \text{i = 0;} \\ \text{ch = 'a';} \end{array} \right.$

This is called *initialization* of the variables. C always allows the programmer to write declarations/ initializers in this way, but it is not always desirable to do so

Example

```
int x = 1;
float total = 0.0;
char ch = "Y"
double r = 0.123e-3;
```

Example

```
x = 10;
total = 3000.00;
name = "Ram";
Ch = "y";
```

Assignment within an executable part does not include data type.

3.2.3 Storing constant in a variable.

This can be done via using const modifier that precedes the variable name is the declaration statement as

const type variablename = value;

Where type is one of the simple data types, value is a constant that will be stored in variable name.

Example

const float pie = 3.1428;

3.3 Data Types

In programming Language there is a set of data type values. Integer, float, character, string, pointer etc. are the examples of data types.

Data types indicate the type of data that a variable can hold. The data may be numeric or non-numeric in nature.

In C, the data type are categorized into

1. Built in data types
2. User-defined data types

3.3.1 Built in data types

They are the basic C data types and designate one single value. The below shown table gives the basic built-in data types.

Types	Keywords	size
Integer	int	2
Character	char	1

Types	Keywords	Size
real(floating-point)	float	4
Double precision	double	8
non-specific	void	---

There is one and only one non-specific data types called void in C. It does not specify any thing.

3.3.1.1 Ingeres (int)

Any integer number is a sequence of digits without a decimal point

Integer type	Bytes	Range	Valid integer	Invalid integer
Short int or in	2	-32768 to +32767	0300	.123 (decimal not allowed)
Long int	4	-2147483648 to +2147483647	-3578, -0210 123, 0	45324 (non permissible range) 3,234 comma not allowed

The **floating point** data type provides the means to store and manipulate numbers with fractional parts and a very large range of sizes.

Real/ floating-point	bytes	Range	Valid float	invalid float
float	4	3.4×10^{-38} to 3.4×10^{38}	-123.543, 1.E-2	1,5 (decimal missing, comma not allowed)
double	8	1.7×10^{-308} to 1.7×10^{308}	.0001E+2, 1.5E + 10	-12.65E+40 (Exponent too large)
long double	10	3.4×10^{-4932} to 3.4×10^{4932}	1234. , 10.00E0	+132.80E no digit for exponent
*E or e is the exponent part which indicate that the value preceding it is multiplied by ten to the value of the exponent part's integer.				

3.3.1.2 Character data type (Char)

Char is a keyword that displays the character type data. The data may be character constant or string constant. A character constant can be defined as any single character enclosed within a pair of apostrophes. It takes 8 bit for storage. The character constant may be signed or unsigned whereas signed values ranges from 128 to + 127 and unsigned character ranges from 0 to 255.

Examples of char data type:

```
char ch = 'y'
```

3.3.1.3 Double

This is a keyword used to indicate a double precision floating-point number. The precision is associated with the accuracy of data and is used whenever more accuracy is required. Normally, it is equivalent to float, but the number of significant digits stored after the decimal point is double than that of the float.

Float usually stores a maximum of 6 digits after the decimal point. But double stores 16 significant digits after the decimal point.

Example:

```
234.0000000000000000 or -0.0000001023999001
```

3.3.1.4 The void data type

Every C function has a return data type associated with it. Where returned type is one of the data type that determine the type of the value returned by the function. If a function is not supposed to return any value, its return type is specified as void. Since the main() is the function from where execution begins. It will usually be declared with type void. A pair of parentheses follows the word main. The matter, if any, contained in these parentheses specifies the formal arguments of the function. If there are not arguments, we write void

3.3.1.5 The main function header

Void main (void) can also be used as main().

3.4 Tokens (Identifiers, keywords, constants, operators)

The basic and the smallest units of C program are called tokens. There are six types of tokens in C. Each of these tokens is formed with one or more characters of C mentioned in above tables.

They are

1.	2.	3.	4.	5.	6.
Keywords	Identifiers	Constants	Strings	Operators	Special Symbols

3.4.1 Keywords and Identifiers

Every word in a C program is either a keyword or an identifier. All keywords (all reserved words) are basically the sequence of characters that have one or more fixed meaning and these meaning, in any circumstances cannot be changed. All C keywords must be written in lowercase letters; because, in C both uppercase and lowercase letters are significant. The list of C keywords is as shown

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

3.4.2 Identifiers

Identifier is used to describe the name of the variables, constants, and functions in C. It is basically a sequence of alphabets and digits. Variables in C are a quantity which may change during the program execution.

The rule that governs the formulation of identifiers names are given below:

- The first character must be an alphabet or underscore.
- All succeeding characters must be either letters or digits.
- Uppercase and lowercase identifiers are different in C.
- No special Character or punctuation symbols are allowed except the underscore "_".
- No two successive underscores are allowed.
- Keywords should not be used as identifiers.

3.4.3 Type modifier or Qualifiers

It offers three adjective keywords to modify the basic integer type : short, long, and unsigned. Here are some points to keep in mind :

Short int <= int <= long int
float <= double <= long double

3.4.4. Unsigned int

This qualifier is used for variables that takes positive values only, by prefixing the int declaration (or long int or short int) with the word unsigned, the range of positive numbers can be double. Unsigned numbers are always positive or zero and obey 2^n where n is the number of bits. So, for instance, if chars are 8 bits, unsigned char variables have values between 0 and 255.

3.4.5 Double long

The type long double specifies extended-precision floating point.

Summary

Constant, Variables and Data Types

Data type is a set of value along with set of permissible operations. Simple data types consists of integers(short, int, long), real number (float, double) and characters(char)

Every data item to be used in program needs to be declared / initialized.

The backslash (\) character constant performs a specific task. They begin with a backslash (\). Both the backslash and an associated character are enclosed within single quotes.

There are four data types: modifiers like signed, unsigned, short, long. These are used to provide the accuracy and precision to the integer and character type data.

Symbolic constants are used to substitutes values to the identifiers defined with the preprocessor directives # define.

Exercise

Q. 1 Multiple Choice Questions

- 1.1 Which of the following is not a valid data type in C language ?
a) Char b) float c) long d) double
- 1.2 Which of the following is not an arithmetic operator ?
a) + b) & c) % d) *
- 1.3 The operator % can be applied to ?
a) Float Values b) Double values c) Integral values d) All of these.
- 1.4 Which of the following is not a valid integer constant of the type int ?
a) 3750 b) 32800 c) -32767 d) 0
- 1.5 A variable of the type int in C can take a value in the range ?
a) 0 to 32767 b) 0 to 65535 c) -32768 to 32767 d) -32767 to 32768
- 1.6 Which of the following is not a reserve word in C language ?
a) for b) goto c) doo d) switch
- 1.7 What will be the value of the expression $5/6/3 + 8/3$?
a) 4 b) 4 c) 2.333 (approx) d) None of these
- 1.8 Identify which of the following are the C token(s)
a) Keyword b) Constants c) operators d) All of these.
- 1.9 Which of the following is not a keyword in C ?
a) const b) main c) size of d) void.
- 1.10 The number of binary arithmetic operator in C is ?
a) 5 b) 4 c) 6 d) 7

Q. 2 Do the following statements

- 2.1 Which of the following are invalid variable names and why ?
a) roll-no or noll_no b) interest_paid c) SUM d) none of these
- 2.2 What wil the program segment results ?

```
int x;  
x = 11;  
x = 12;  
x = 13;  
printf("%d %d %d\n",x, x, x);
```

Ans 13 13 13

- 2.3 What is printed by the following program ?

```
main()  
{ int a, b, c;  
b = 4;  
c = a +b;  
}
```

Ans Garbage Output because a is not assigned a value.

- 2.4 What will be the output of the following program ?

```
void main (void)  
{  
printf("%d", 'B');
```

}

Ans. 66

2.5 What will be the output of the following ?

```
void main (void)
{
float x = 1/2.0 - 1/ 2;
printf("%.2f", x);
}
```

Ans 0.50

**2.6 if x = 12.4568, what value will be printed, if the printf() function is used as ?
printf("%.3f", x);**

Ans 12.457

2.7 Find the error, if any, in the program ?

```
void main (void)
{
float a,b;
printf("\n Enter value of a:");
scanf("%f", a);
b = a * 3;
printf("\n Value of b = %f\n",b);
}
```

2.8 What will be the output of the following ?

```
int x = 3; n = 4;
x = + +n;
x = x++;
Ans 5 6
```

Q. 3 Solve the given below problems

3.1 Correct the statement

x = (float) 5/2

3.2 What would be the value of C ?

```
main(0
int c;
float a, b;
a = 245.05;
b = 40.02;
c = a + b; )
```

Ans 285

3.3 Find the value of a and c ?

```
main ()
{ int a, b, c;
b = 2;
a = 2 * (b++);
c = 2*(++b);
}
```

Ans a = 4, c = 8.

3.4 Find the value of a & b ?

```
main()
int a,b;
a = 2;
b = ++a+2;
printf("Value of a is %d and b is %d\n", a, b);
```

Q.4 State whether True or False

- 4.1 Variable is such quantity that changes its values during the execution of a program?
- 4.2 A delimiter is a symbol that has a syntactic meaning and significance ?
- 4.3 The size of an operator is data_type ?
- 4.4 A char data type always occupies one byte ?
- 4.5 Semicolon is a delimiter of one declaration ?

Q.5 Write an appropriate word or output for the following.

- 5.1 The computer splits a program into a number of parts ?
- 5.2 Modifier is used to declare a variable as constant ?
- 5.3 if a = -11 and b = -3. What is the value of a % b ?
- 5.4 The number of relational operators in C language is ?
- 5.5 If we have *, /, (), % operators then which of these have the highest precedence ?

Q.6 Answer the following Questions.

- 6.1 What is the procedure for storing constant in a variable ? Give example.
- 6.2 What does the statement indicate ? **big = a > b ? a: b;**
- 6.3 Write the logical statements for the following if:
 - a) Marks obtained is more than 95 for all students whose category is not T
 - b) Total marks is more than 550 for all students whose class is 3A or 3B

6.4 Suppose the expression $y = y^3 * (x + y^*z)$; is rewritten as:

$y^* = 3;$

$y^* = x + y^*z$

Will the above statement produce the same result as the earlier one ?

6.5 Identify unnecessary parenthesis in the following statements ?

a) $((x - 9y/5) + z) \% 8 + 25$

b) $((z - k) * y) + a$

c) $(a * b) + (-y/z)$

Check, what will be printed in the following below given programs:

6.6 `# include <stdio.h>`
 `main()`
 `{ int x, y;`
 `y = 2;`
 `x = y + 1;`
 `printf("%d %d\n", x, y);`
 `y = y + 2;`
 `x = 5;`
 `printf("%d %d\n" x, y);`
 `}`

6.8 `# include <stdio.h>`
 `main()`
 `{`
 `int a, c;`
 `a = 4;`
 `c = a * a * a;`
 `printf("%d %d\n", a, c, a*a);`
 `}`

6.7 `# include <stdio.h>`
 `main()`
 `{ int a, b;`
 `a = 6;`
 `a = a + 3;`
 `b = 4 % a;`
 `printf("%d %d\n", a, b);`
 `}`

6.9 `#include<stdio.h>`
 `main()`
 `{ int rate, time, dist, junk;`
 `rate = 7, time = 3;`
 `junk = rate + time;`
 `dist = rate * time;`
 `printf("%d %d %d %d\n", rate,`
 `time, junk, dist);`
 `}`

Q.7 Short answer type questions.

- 7.1** **What are delimiters ?**
7.2 **What is identifier ?**
7.3 **What is data type ?**

Answer to objective questions

Q	.1	.2	.3	.4	.5	.6	.7	.8	.9	.10
1	a	b	d	b	c	c	d	b	a	a
4	T	T	T	T	T					
5	token s	const	-2	6	()					

4.0 Introduction

Operators are the verbs of a language that let the user perform computations on values. You can think of operators as verbs and of operands as the subjects and objects of those verbs. C language is rich set of operators is one of its distinguished features. An expression is a formulat consisting of one or more operands and zero or more operators linked together to compute a value. The computed values are to be stored in a variable for future use. This is done using an assignment statement. In order to carry out basic arithmetic operations and logical operations, operators are needed. Operators act as connectors and they indicate what type of operation is being carried out. The values that can be operated by these operators are called operands.

4.1 Operators and Expressions.

In order to carry out basic arithmetic operations and logical operations, operators are needed. Operators act as connectors and they indicate what type of operation s being carried out. The values that can be operated by these operators are called **operands**. The format is

Operand1 Operator Operand2

(The value of operand2 is applied on operand1 as directed by operator.)

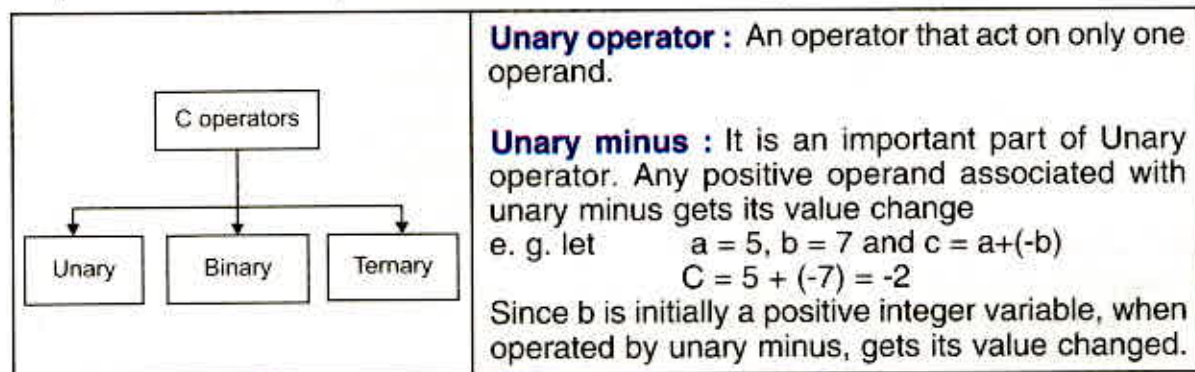
4.1.1 Expression

An expression is something which returns a result. We can then use the result as we like in our program. Expressions can be as simple as a single value and as complex as a large calculation. They ar made up of two things, operators and operands.

Example of expressions are: $2 + 3 * 4$, $-1 + 3$, $(2 + 3) * 4$

These expressions are worked out (evaluated) by C moving from left to right.

C operators are broadly classified into three main categories as

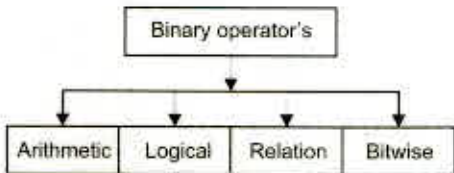


Example : $a = 5$, $b = 7$ and $c = a + (-b)$
 $C = 5 + (-7) = -2$

Since b is initially a positive integer variable, when operated by unary minus, gets its value changed.

4.1.2 Binary operators

These operators act on two operands. They are further classified into following



4.1.3 Arithmetic Operators

There are five arithmetic operators in C language. They are:

Operator	Meaning
+	Addition or Unary Plus
-	Substraction or Unary Minus
*	Multiplication
/	Division
%	Modules Operator

The operator % (modules operator) is used to get remainder after integer division. This operator cannot be used with floating point numbers.

Examples : The expression x%y produce the remainder, when x is divided by y & thus is zero when y divides exact integer division truncates and fractional part as shown below:

Suppose a = 12.5 and b = 2.0 are floating point variables arithmetic expression and their results are shown below:	Suppose a = 10 and b = 3 are integer variable the arithmetic expression and their result are shown:																						
<table> <tr> <th>Expression</th><th>Value</th></tr> <tr> <td>a + b</td><td>14.5</td></tr> <tr> <td>a - b</td><td>10.5</td></tr> <tr> <td>a * b</td><td>25.5</td></tr> <tr> <td>a/ b</td><td>6.25</td></tr> </table>	Expression	Value	a + b	14.5	a - b	10.5	a * b	25.5	a/ b	6.25	<table> <tr> <th>Expression</th><th>Value</th></tr> <tr> <td>a + b</td><td>13</td></tr> <tr> <td>a - b</td><td>7</td></tr> <tr> <td>a * b</td><td>30</td></tr> <tr> <td>a/ b</td><td>3</td></tr> <tr> <td>a % b</td><td>1</td></tr> </table>	Expression	Value	a + b	13	a - b	7	a * b	30	a/ b	3	a % b	1
Expression	Value																						
a + b	14.5																						
a - b	10.5																						
a * b	25.5																						
a/ b	6.25																						
Expression	Value																						
a + b	13																						
a - b	7																						
a * b	30																						
a/ b	3																						
a % b	1																						

4.1.3 Operations & Hierarchical order

Though the expression are presented by a sequence of operators and variables, the C compilers follows a hierarchy to execute them in order. The hierarchy for an arithmetic expression is as:

1.	* / %	evaluated from left to right
2.	+ -	evaluated from left to right

Hierarchy of operator's or the Precedence of the Operators.

Operator precedence describes the order in which C reads expressions.

For example, the expression $a = 4 + b * 2$ contains two operations, an addition and a multiplication.

Does the C compiler evaluate $4 + b$ first, then multiply the result by 2 or does it evaluate $b * 2$ first, then add 4 to the result.

The operator precedence chart contains the answers. Operators higher in the chart have a higher precedence, meaning that the C compiler evaluates them first.

In the following table, x represents an expression. The operator with the highest precedence (the parentheses) is listed first; the one with the lowest(=), is listed last

Operator classification	Operators
parentheses	()
postfix operators	[], x++, x--
unary / prefix operators	++x, --x, +x, -x, !
multiplicative	&, /, %
additive	+, -
relational	<, >, >=, <=
equality (boolean)	=, !=
logical AND	
assignment	=

Example : Now let us consider the following determine the hierarchy of operations and evaluate the expression: $a = 3 * 3 / 7 + 3 / 3 + 8 - 2 + 5/8$

$a = 9 / 7 + 3 / 3 + 8 - 2 + 5/8$	operation : *
$a = 1 + 3 / 3 + 8 - 2 + 5/8$	operation : / as 9/7 gives integer 1 not 1.3
$a = 1 + 1 + 8 - 2 + 5/8$	operation : / as 3/3 gives integer 1
$a = 1 + 1 + 8 - 2 + 0$	operation : / as 5/8 gives 0, 5 & 8 are integer constants and returns integer value
$a = 2 + 8 - 2 + 0$	operation : +
$a = 10 - 2 + 0$	operation : +
$a = 8 + 0$	operation : -
$a = 8$	operation : +

Example 1: To illustrate the basic arithmetic operations here the variables are defined with constant values and each arithmetic operation is executed through program.

```
#include<stdio.h>
void main()
{
    int n1, n2;
    int sum, diff, prod, quotient, remainder;
    n1 = 5;
    n2 = 3;
    sum = n1 + n2;
    diff = n1 - n2;
    prod = n1 * n2;
    quotient = n1 / n2;
    remainder = n1 % n2;
    printf("SUM = %d\n",sum);
    printf("DIFFERENCE= %d\n",diff);
    printf("PRODUCT = %d\n",prod);
    printf("QUOTIENT = %d\n",quotient);
    printf("REMAINDER= %d\n",remainder);
};
```

Output

```
SUM          = 8
DIFFERENCE= 2
PRODUCT      = 15
QUOTIENT     = 1
REMAINDER    = 2
```

Example 2: Write a c program calculate the gross payment for Rate per hour = 80.75, working hours = 45.75, where gross pay is working hours*rate per hour.

```
#include<stdio.h>
void main()
{
    //declaring float variables
    float hourRate;
    float hourwrkd;
    float grosspay;
    // assigning values to variable
    hourRate = 80.75;
    hourWrkd = 45.75;
    grosspay = hourRate * hourwrkd;
    // displaying the variables on the screen
    printf("Hourly Rate = Rs. %f\n Hour worked = %f hours\n Gross Pay = Rs. %f",hourRate,hourwrkd* : grosspay);
}
```

Output

```
Hourly Rate = Rs. 80.750000
Hourly worked = 45.75 0000
```

```
Gross pay = Rs. 36943.312500
```

Modify the above output & write the program to display the output as:

```
SUM( 5 + 3)          = 8
DIFFERENCE (5 -3)    = 2
PRODUCT (5 * 3)      = 15
QUOTIENT (5/3)       = 1
REMAINDER (5 % 3)    = 2
```

What will be the output of the following C program.

```
#include<stdio.h>
void main()
{
    int months, days;
    printf("enter days\n") // input 3 values 45, 265, 364
    scanf("%d", & days);
    months = days / 30;
    days = days % 30;
    printf("Months = %d Days = %d,", months, days);
}
```

4.2 Relational & Logical Operator

4.2.1 Relational Operator

These are used to compare the operands of operators and display the relations between the two constant or variables. The result of these is either true or false. Where value of true is non zero and false is zero.

<p>The marks of two students of salary of two persons needs to be compared. We can compare these with relational operators. There is only one relational operator in this simple relational expression and takes the following form</p> <p>exp1 relational operator exp2</p> <p>Where exp1 and exp2 are expressions, which can either be their simple constant variables or combination.</p>	Operator	Meaning
	<	Less than
	<=	Less than or equal to
	>	More than
	>=	More than or equal to
	==	equal to
	!=	Not equal to _

In C Program, to use these operators, we use if statements. This is a conditional control statement that determines the value of condition is whether true or false.

<p>Example : To determine whether both digits are true or false:</p> <pre>#include <stdio.h> main() int x = 2, y = 2; if(x == y) printf("x and y are equal\n"); }</pre> <p>Output: x and y are equal</p>	<p>Example: To determines the unequality of the operators.</p> <pre>#include<stdio.h> main() { float a = 3.533, b = 0.0232; if(a>b) printf("a is greater than b\n"); }</pre> <p>Output: a is greater than b</p>
--	--

4.2.2 Logical Operators

C has the following logical operators; they compare or evaluate logical and relational expressions.

Following are the logical operator of C:

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

Logical AND (&&) Operator

This operator is used to evaluate two conditions or expressions with relational operators simultaneously. If both the expressions to the left and to the right of the logical operators is true then the whole compound expression is true.

Example : Consider the statement `a > b && x == 10`

The expression to the left is `a > b` and that on the right is `x == 10` the whole expression is true only if both expressions are true i.e., if `a` is greater than `b` and `x` is equal to 10.

Logical OR (||)

The logical OR is used to combine two expression or the conditions evaluates to true if any one of the two expressions is true.

Example: Consider the statement `a < m || a < n`

The expression evaluates to true if any one of them is true or if both of them are true. It evaluates to true if `a` is less than either `m` or `n` and when `a` is less than both `m` and `n`.

Logical NOT (!)

The logical not operators takes single expression and evaluates to true if the expression is false and evaluates to false if the expression is true. In other words it just reverses the value of the expression.

Example : Consider the statement ! `(x >= y)`

The NOT expression evaluates to true only if the value of `x` is neither greater than nor equal to `y`. The Not operator is often used to reverse the logical value of the string variable as in expression.

If (! a) the another way of saying this is **if (a ==0)**

Based on the above said examples let us consider the following two examples:

If we need to check two conditions simultaneously like the grade of the student in exams should be equal to 1 and his total marks should be more than 550 then we could write the conditions as `if (grade ==1 && tmarks > 550)`

In this case both conditions should be true for the complete expression to be returning a true value. If even one of the two conditions is false, the complete returned value is false.

Problem : What will be the explanation for following statement ?

```
if(grade ==1 ||  
tmarks>550)
```

4.2.2 Assignment Operator

This Assignment operator is to "set equal to". The special character used for the assignment operator is the equal sign (=).

This:	is identical to this:	Operation Name
$x += y;$	$x = x + y;$	Addition assignment
$x -= y;$	$x = x - y;$	Subtraction assignment
$x *= y$	$x = x * y$	Multiplication assignment
$x /= y$	$x = x / y;$	Division assignment
$x \% = y$	$x = x \% y;$	Remainder assignment (int only)

It is used as **operand1 = operand 2**

The assignment operator specifies that the first operand is assigned the value of the second operand. Operand2 is evaluated and the result is assigned to operand1.

Example : $x = y;$ $/*$ is set equal to the value of $y*$

Consider the statement

$i = i + 5;$ this can be written as $i += 5;$

$i = i * 5;$ this can be written as $i *= 5;$

Multiple Assignment Statement

Whenever an initialize different variable in a single statement. This type of initialization would need more number of assignment operators. Hence, the name multiple assignment

Example: i $\text{int } x = y = z = 10;$

ii **Float**

$a = b = c = d = 5.75;$

First statement initializes the three int variables x, y and z to 10. The second statement initializes the four float variables a, b, c, d to 5.75

Problem : if $x = 11$, $y = 6$, check the resultant values given the column value

Expression	value
$x > 9 \ \&\& \ y != 3$	1
$x == 5 \ \&\& \ y != 3$	1
$5 \ \&\& \ y != 8 \ \&\& \ 0$	0
$!(x > 9 \ \&\& \ y != 23)$	0

4.2.3 Increment & Decrement Operators

These operators specify that the operand is either incremented or decremented, either before (pre) or after (post) the value of the operand is used, in the context of C statement.

Operator	Usage
++	increment the variable
--	decrement the variable

like ++i or --i
 1++ or i --

Incrementing and Decrementing are of two types:

1. First increment or decrement then takes the value for processing (**prefix**)
2. Take the value for preprocessing & then increment or decrement the variable (**postfix**)

Example 1: Post increment

a = 10;

b = a ++;

Here the value assigned to **b** would be 10 & the new value of **a** would be 11.

b = a;

a = a + 1;

Example 2 : Post Decrement

a = b--;

This equivalent to two assignment statements

a = b; and b = b - 1;

Example 3: Pre Increment

a = 10;

b = ++a;

Here the value assigned to **b** would be 11 & the new value of **a** would be 11.

a = a + 1;

b = a;

Example 4: Pre Increment

a = ++b;

This equivalent to two assignment statements.

b = b + 1; and a = b;

Example 5: Pre Decrement

a = --b;

This is equivalent to assignment statements

b = b - 1; and a = b;

Increment and Decrement (prefix) :**Example 1:**

k = 2; /* k is assigned with 2 */
 i = ++k; /* Increment k by 1, then
 assigns to i therefore i = 3 and k = 3 */
 i = --k; /* Decrement k by 1, then assigns to i
 therefore i = 2 and k = 2. */

Example 3 :

/* Program to explain the prefix increment
 operator */

```
void main()
{
    int i, j;
    i = 5;
    j = ++i + 5; /* i is incremented
                  then processed */
    printf("Value of i is %d and j is %d\n", i, j);
}
```

Output : Value of i is 6 and j is 11

Explanation:

J is assigned with the result of ++5. First i is incremented the result is added with 5 and the net result is assigned to j.

Postfix Increment/Decrement :**Example 2:**

k = 2; k is assigned with 2
 i = k++; /* assigns i with k, then
 increments k therefore i = 2 and k = 3.*/
 i = k--; /* Assigns i with k, then decrements
 k therefore i = 3 and k = 2.*/

Example 4 :

/* Program to explain the postfix increment
 operator */

```
void main()
{
    int i, j;
    i = 5;
    j = 5 + i++; /* i is added to 5, assigned
                  to j and i is incremented */
    printf("Value of i is %d and j is %d\n", i, j);
}
```

Output : Value of i is 6 and j is 10

Explanation:

j is assigned with the result of 5 + i++ first i is added with 5, i is incremented and the net result is assigned to j.

Example 5:

/* Program to explain the prefix decrement
 operator */

```
void main()
{
    int i, j;
    i = 5;
    j = --i + 5; /* i is decremented & then
                  processed */
    printf("Value of i is %d and j is %d\n", i, j);
}
```

Output : value of i is 4 and j is 9

Explanation:

j is assigned with the result of -i + 5 First i is decremented, the result is added with 5 and the net result is assigned to j.

Example 6 :

/* Program to explain postfix
 decrement operator */

```
void main()
{
    int i, j;
    i = 5; /* i is added to 5, assigned to j and
            then decremented */
    j = 5 + i--;
    printf("value of i is %d and j is %d\n", i, j);
}
```

Output : Value if i is 4 and j is 10

Explanation:

j is assigned with the result of 5 + i -, First i is added with 5, i is decremented and the net result is assigned to j.

Problem increment Operator :

What would be the value of x & y so that the output should be.

Problem Decrement Operator :

What would be the value of x & y so that the output should be

<pre> x = 3 y = 7 #include<stdio.h> void main() { int x = ?, y = ?; printf("x = %d\n", x++); printf("y = %d\n", ++y); } </pre>	<pre> x = 5 y = 8 #include<stdio.h> void main() { int x = ?, y = ?; printf("x = %d\n", --x); printf("y = %d\n", y--); } </pre>
--	--

4.2.4 Ternary Operator

It is a conditional operator in C. It takes three operands. The symbol "?" is used as ternary operator in C. It is used as

<expression> ? <value1> : <value2>;

Where

expression → relational operator

Value1 → value to be assigned on true

Value2 → value to be assigned when the result of the expression is false.

Example : **x = y < z ? y : z**

Here X will be assigned the value of y if y is less than z otherwise it will be assigned the value of z.

4.2.5 Command Operator

It is used to link two/more related expressions together.

Example : **n = (i = 5, j = 3, ++j);**

(First assigns the value 5 to i, then assign 3 to j, and finally assign 15 to n)

Example:

```

#include<stdio.h>
void main()
{
int a = 4, b = 5, result1, result2;
result1 = a > b ? a : b;
printf("The result1 = %d\n", result1);
result2 = a < b ? a : b;
printf("The result2 = %d\n", result2);
}

```

Output :

The result 1 = 5

The result 2 = 4

4.2.6 Size of () of Operator

This operator returns the size (i.e the number of bytes) of the operand. It should be written in the lower case. It must precede its operand. This may be constant, a variable or data type. It is used as.

Sizeof(operand);

4.2.7 Bitwise Operator

All data items are stored in the memory as a sequence of bits (0's or 1's), and some applications need the manipulation of these bits. To perform, the bitwise operations, C provides six operators. These operators work with int or char type data.

Example :

```
#include<stdio.h>
void main()
{
    int x;
    float y;
    char ch = 'y';
    x = 10;
    y = 100.00;
    printf("Size of x = %d\n",sizeof(x));
    printf("Size of y = %d\n",sizeof(y));
    printf("Size of ch = %d\n",sizeof(ch));
    printf("Size of double = %d\n",sizeof(double));
}
```

Output:

```
Size of x = 2
Size of y = 4
Size of ch = 1
Size of double = 8
```

Summary

Operators and Expression

C operators are basically classified into unary, binary and ternary operators. There are five unary, four types of binary operators and one ternary operator. Where unary minus, !, ++, -- and - are unary operators. The +, -, *, / are basic arithmetic operators. Modules operators is another arithmetic operators which results in the remainder after an integer division.

The <, <=, >, >=, ==, != are relational operators. These are used to define the relation between constants and variables.

C language supports variety of assignment statements.

There is precedence of operators that determines the evaluation of expressions.

Exercise

Q. 1 Fill in the blanks

- 1.1 Operator of the exponentiation is _____
- 1.2 _____ Operator is used to combine the different conditions.
- 1.3 `int i = 5` is called _____ assignment.
- 1.4 `i = 5` is called _____ assignment.
- 1.5 There is no Hierarchical order for _____ operators.

Q. 2 State whether True or False

- 2.1 The expression `++ (a+b)` is valid one, (if `a = y, b = 5`).
- 2.2 The expression `a && = b` is valid one, (if `a = y, b = 5`).
- 2.3 The expression `-- 10` is valid one.
- 2.4 The assignment statement "`a + b = c;`" is acceptable in C language.
- 2.5 An identifier in C can have A – Z as its first letter.

Q. 3 Short Answer type Questions

- 3.1 Where does the Modules Operator can be only applied.
- 3.2 Find the difference between logical and relational operator.
- 3.3 What is Unary operator.
- 3.4 Given one example each for Increment and decrement operator.
- 3.5 In C, for which purpose expression is used.
- 3.6 Write different types of assignment operator.
- 3.7 What is operand.

Q. 4 What will be result of the execution of program that is given below ?

```
a = 7;  
b = ++a+5;  
c = b--=10;  
printf("%d, %d, %d\n",a, b, c);
```

Answer to objective questions

Q.	.1	.2	.3	.4	.5
1	**	relational	Declaration	General	relational
2	F	F	T	F	T

	Chapter
Control Flow (Part - I)	5

5.0 Introduction

A computer program is a set of instructions given to a computer to do specific work. These instructions or statements are executed one after the other as they appear in the program. This order of executing the statement works well for simple problems where no decision making process is involved. But in actual practice, it is often required to change the order of execution of statements or repeat a group of statements for known number of times or until certain specific conditions are met. In such cases, the order in which these statements will be executed is to be controlled. Changing the order of execution of the statements is the changing of the flow of control, might involve one of these situations.

5.0.1 Branching

Branching means depending on the outcome of a decision, executing one group of instructions.

5.0.1.1 Conditional control statements or decision making statements

They appear in a program along with the sequential structures. They are responsible to check for the validity of the given condition and initiate execution of specific program statements if the condition is true and omit specific statements if false or vice versa. They are:

- If statement
- If-else statement
- Else if statement
- Switch statement

5.0.2 Looping

Looping means executing a group of instructions repetitively either for a given number of times or until the required conditions is met. **Looping statements:** it is like a conditional statement evaluating a specific condition. If the condition evaluates to be true a given set of program statements are executed repeatedly until the condition (which is re-evaluated during each loop) evaluates to false. They are:

- For statement
- While statement
- Do-while statement

5.0.3 Jumping

Jumping means transferring control from one point to another point in a same program unit. These include the followings.

- break statement
- continue statement
- goto statement

C language provides facilities for controlling the order of execution of the statements, which are referred to as **flow control** statements. C provides the following sets of control statements.

5.1 Decision making statements

5.1.1 If statements

This statement is used to check for a condition before executing a statement it has two formats.

if(expressions)	[if(expression)];	Example:
statement The statement can also be a block of statements.	<pre>{ statement1; statement2; statement3; }</pre> The expression is evaluated. If the result is true then the statement following it is executed otherwise it is skipped.	<pre>int a,b,c; a = 10, b = 5, c = 10; if(a>b) c = c + 5; printf("%d",c);</pre> Output: 15

Here if the value of a to begin with is less 5, say 4, then the value printed is 10 since the expression (a>b) is evaluated to be false so that the statement c = c + 5 is bypassed.

5.1.2 If-else

The expression in the if statement is evaluated first. If it is true then statement-1 is executed otherwise (i.e. if it is false) statement-2 is executed.

<pre>If expression { statement1; else statement2; }</pre> <pre>int a,b,c; a = 10; Example 1 b = 5; if(a>b) c = 5; else c = 7; printf("Value of c = %d\n",c);</pre> <p>Output Value of c = 5</p> <p>however, if the initial value of a is 4 (less than the value of b) then the result printed will be 7.</p>	<pre>graph TD Entry[Entry] --> Condition{Condition} Condition -- True --> Statement1[Statement1] Condition -- False --> Statement2[Statement2] Statement1 --> Exit[Exit] Statement2 --> Exit</pre>
--	---

Example 2 This program accepts a number and prints if it is an odd number otherwise exit.

```
#include<stdio.h>  
void main(0
```

```

{
int numb;
printf("enter the number\n");
scanf("%d", & numb);
if((numb%2) != 0)
printf("%d, is an odd numbers\n", numb);
} /* end of main()*/

```

Output

Enter the number
13
13, is an odd number

Example 3

To find the largest of three numbers

```

void main()
{
float a, b, c;
printf("Enter the three numbers ->");
scanf("%f %f %f",&a, &b, &c);
printf("\n\n The largest among %7.3f, %7.3f, %7.3f is", a, b, c);
if(a>b && a>c)
printf("%7.3f\n",a);
else if ((b>a && b > c)
printf("%7.3f\n",b);
else
printf("%7.3f\n",c);
}

```

Output

Enter the three numbers -> 23 76 45
The biggest among 23.000, 76.000, 45.000 is 76.000

Example 4 This program accepts two floating point numbers and computes their ratio. If the ratio is greater than zero then it exchange the contents of these numbers

```
#include<stdio.h>
void main()
{
float x, y, ratio, temp;
Printf("Enter the X and Y \n");
scanf("%f %f", &x, &y);
ratio = x/y;
printf("Ratio = %f\n",ratio);
if (ratio>0)
{
temp = x;                                /*exchange variables */
x = y;
y = temp;
}
printf("X = %f, Y = %f\n",x, y);
}
```

Output :

Enter X and Y

34.0

55.0

Ratio = 0.6181818

X = 55.000000, Y = 34.000000

Example 5: Use nested if as in example 2 in previous page, to complete the program, if the marks obtained by a student in different subjects are input through keyboard. The student gets a grade as per the following rules:

marks

75 or above

- grade = A

above 60 but below 75

- grade = B

above 50 but below 60

- grade = C

less than 50

- grade = D

Hint

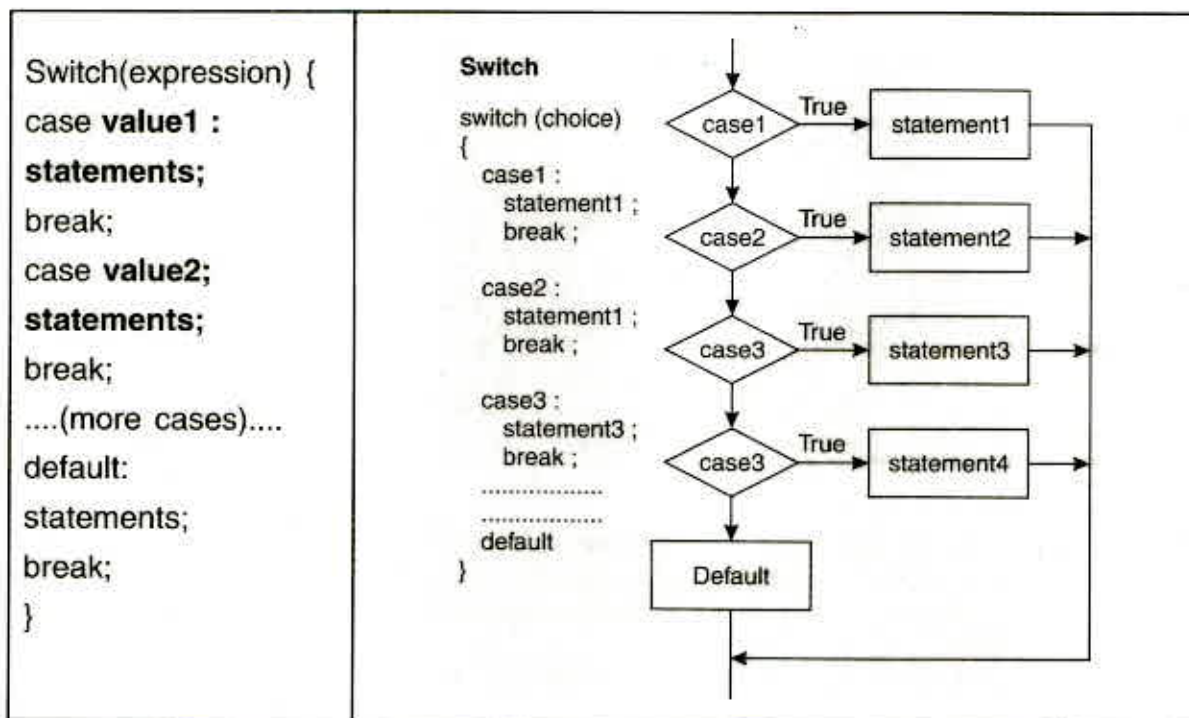
```
if(marks>=75)
    grade = "A";
else
    if(marks>=60)
        grade = "B";
    else
        if(marks>=50)
            grade = "C";
        else
            printf("grade = D");
```

5.2 Switch statement

We have seen that the if-else statement chooses one of two statements, based on the value of expression. The switch statement chooses one of the several statements, based on the value on an integer (int, byte, short, or long) or a char expression. The keyword case is followed by an integer or a character constant. Each constant in each case must be different from all the other.

Under certain conditions during loop operations, it may be necessary to skip a part of the body of the loop and besides this we have to run the next part. To do this in C, continue statement is used.

The syntax is:



5.3 The Break Statement

Sometimes while executing a loop it becomes desirable to skip a part of the loop or quit the loop as soon as certain condition occurs.

For example consider searching a particular number in a set of 100 numbers as soon as the search number is found it is desirable to terminate the loop. C language permits a jump from one statement to another within a loop as well as to jump out of the loop. The break statement allows us to accomplish this task. A break statement provides an early exit from for, while, do and switch constructs. A break causes the innermost enclosing loop or switch to be exited immediately.

Example 6: Program to illustrate the use of break statement:

Complete this program to find the average.

/* A program to find the average of the marks*/

```

#include<stdio.h>
void main(0)
{
    int a, num = 0;
    float sum = 0, average;
    printf("Input the marks, -1 to end\n");
    while(1)                                /*While loop starts*/
    {
        scanf("%d", &a);
        if(a == -1)
            break;
        sum = sum + a;
        num++;
    }
}

```

5.4 Continue statement

During loop operations it may be necessary to skip a part of the body of the loop under certain conditions. like the break statement C supports similar statement called continue statement. The continue statement causes the loop to be continued with the next iteration after skipping any statement in between. To continue with the next iteration the format of the continue statement is simply; **continue**;

Consider the following program that finds the sum of five positive integers. If a negative number is entered, the sum is not performed since the remaining part of the loop is skipped using continue statement

```

#include<stdio.h>
void main()
{
    int a = 1, num, sum = 0;                /*declare and initialize the variables */
    for (a = 0; a < 5; a++)                 /* for loop */
    {
        printf("Enter the Integer");        /*Message to the user */
        scanf("%d", &num);                 /*read and store the number */
        if(num < 0)                         /*check whether the number is less than zero */
        {
            printf("You have entered a negative number"); /*message to the user */
            continue;                       /*starts with the beginning of the loop */
        }                                  /*end of for loop */
        sum += num;                         /*add and store sum to num */
    }
    printf("The sum of positive numbers entered = %d", sum); /*print the sum */
}

```

Example 7 :

Input the number to get the day of the week (Using switch statement)

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

Example 8 :

Program to evaluate simple expression of the form

```

//value operator value
# include <stdio.h>
void main ( )

```

```
#include<conio.h>
void main (void)
{
int day;
clrscr();
printf("\nEnter the day of the week as number : ");
scanf("%d", &day);
switch(day)
{
case 0: printf("\n Day of the week is Sunday\n");
break;
case 1: printf("\n Day of the week is Munday\n");
break;
case 2: printf("\n Day of the week is Tuesday\n");
break;
case 3: printf("\n Day of the week is Wednesday\n");
break;
case 4: printf("\n Day of the week is Thursday\n");
break;
case 5: printf("\n Day of the week is Friday\n");
break;
case 6: printf("\n Day of the week is Saturday\n");
break;
default : printf("\n Wrong input\n");
}
}
```

Output ;

Enter the day of week as number : 4

Day of the week is Thursday

```
{
float value1, value2;
char operator1;
printf("Type in your expression.\n");
scanf("%f %c %f", &value1, &operator1, &value2);
switch(operator1)
{
case '+':
printf("%2f\n", value1+value2);
break;
case '-':
printf("%2f\n", value1-value2);
break;
case '*':
printf("%2f\n", value1*value2);
break;
case '/':
if(value2 == 0)
printf("Division by zero.\n");
else
printf("%2f\n",value1/value2);
break;
default;
printf("Unknown operator.\n");
break;
}
}
```

Output

Type in your expression.

178.99 - 326.8

147.81

Example 9

Program to calculate the area of the geometrical figures such as circle, square, triangle, rectangle. It should display the menu of the figure codes of different figures. On choosing particular fig_code, the corresponding parameters required by that figure are accepted, processed and area is displayed.

```
#include<stdio.h>
main()
```

```

{
    int fig_code;
    float side, base, length, breadth, height, area, radius;
    printf("=====\n");
    printf("1 Circle\n");
    printf("2 Rectangle\n");
    printf("3 Triangle\n");
    printf("=====\n");
    printf("enter the figure code");
    scanf("%d",&fig_code);
    switch(fig_code) {
    case 1:      printf("Enter radius \n");
                 scanf ("%f" & radius);
                 area = 3.142 * radius * radius;
                 printf("area of the circle = %f\n",area);
                 break;
    case 2:      printf("Enter breadth and length \n");
                 scanf("%f %f", &breadth, &length);
                 area = breadth * length;
                 printf("area of the rectangle = %f\n",area);
                 break;
    case 3:      printf("Enter base and height \n");
                 scanf("%f %f", &base, &height);
                 area = 0.5*base*height;
                 printf("area of the triangle = %f\n",area);
                 break;
    default:     printf("error in the figure code\n");
                 break;
    }
}

```

Output :

```

=====
1      Circle
2      Rectangle
3      Triangle
=====

```

```

Enter the figure code 3
Enter base and height 10 and 12
Area of triangle is    0.6

```

Summary

We have seen in this chapter, there are three types of situations that demand the changes in the flow of the control. These are:

- Skipping some segments of the programs or going back to execute the same segment again.
- Executing one segment out of two or more segments depending on the outcome of some condition.
- Executing a segment repeatedly either for a given number of times or till some conditions is met.

For these situations, C has four types of statements; declaration statement, input-output statements, arithmetic, logical statements and control statements.

The control statement defines; the order or execution of the program statements provides. Three types of control statements are:

- Conditional
- Looping and unconditional
- Control Statement

The four basic conditional control statements are :

- if
- if-else
- if-else-if
- switch statement

These are called selective or decision marking control statements.

The goto is unconditional control statement which transfers the control to the specified statement in a program.

Looping is also called an iterative or repetitive control mechanism, where in, set of statement are repeatedly executed either for a fixed number of times or as long as a certain logical condition is true.

Exercise

Q. 1 Multi Choice Questions

1.1 C is an example of ?

- a) object oriented language
- b) object based language
- c) Structural programming language
- d) None of these.

1.2 Structural programming approach makes use of ?

- a) Modules
- b) Control structures
- c) both a and b
- d) None of these.

1.3 Coma is used as ?

- a) A separator in C
- b) an operator in C
- c) terminator in C
- d) a delimiter in C.

1.4 A null statement can be represented by a ?

- a) New line
- b) blank space
- c) semicolon
- d) colon.

1.5 Infinite loop is ?

- a) Useful for time delay
- b) useless
- c) used to terminate execution
- d) not possible

1.6 Break statement is used to..... ?

- a) Selective control structures only
- b) Loop control structures only
- c) both option a and b
- d) switch-case control structures only.

1.7 Continue statement is used to ?

- a) Continue the next iteration of the loop statement.
- b) Exit the block of loop statement.
- c) Continue execution of the program even errors occurs
- d) Exit from the outmost block even it is used in the innermost block.

Q. 2 State whether True or False

2.1 Every control statement must be terminated by semicolon ?

2.2 The else keyword always matches to the nearest if statement ?

2.3 Break statement must be used for each case, except the last, in a switch statement ?

2.4 In switch statement, case values must be in ascending order ?

2.5 The continue statement cannot be used with decision making statements?

Q. 3 Short answer type questions:

- 3.1 What is Loop ?
- 3.2 What do you mean by forward and backward jump ?
- 3.3 When do you prefer if-else statements ?
- 3.4 What are the four basic conditional continue statements ?

Q. 4 Answer the following questions.

- 4.1 What is Conditional execution ? Where it is useful ?
- 4.2 In the process of grouping the people into child, young and old categories, depending on the age group, which control statement would you like to choose any why ?
- 4.3 When do we choose for loop ? How is it different from a while loop ?
- 4.4 What are Case labels ? Where they are used ?

Q. 5 Complete the following program to test the output ?

```
void main (void)
{
    int a,b;
    for (a = 0; a<10; a++)
    {
        switch (a%5)
        {
            case 0 : b = 0;
            case 1 : b++;
            case 2 : b+ = 2;
            case 3 : b + = 3;
            case 4 : b + = 4;
            default : b + = 10;
        }
    }
}
```

```
printf("%d", b);}
```

Output

90

Q. 6 Complete the following program and find the output.

6.1 if(x > 10), then
 x + = 1;
 else
 x - = 1;

6.2 if(a<>b)
 printf("a and b are unequal");
 else
 printf("a and b are equal");

6.3 if(value = ! 1100)
 value - -;
 printf("value = %d\n", value)

6.4 if (x>0)
 y = 1
 else if (x==0)
 y = 0
 else if (x<0)
 y = 1
 printf("y = %d\n", y);}

6.5 main()
 {
 int a = 0, b = 0: if (a==0)
 if (b == 0)
 printf("Hello Mr");
 else/
 printf("Hello Mrs.);}

6.6 main()
 float p = 3.2, q = 6.0;
 int r = 25;
 if(r)
 { p = p + 2.3;
 q = p + q; }
 printf("%f %f\n",p,f);}

6.7 main ()
 { char ch = "**";
 switch (ch)
 {
 case '*' : printf("%c",ch);
 case ' ?' : printf("%c", ch);
 default : printf("%c",*);}}

Answer to objective questions

Q.	.1	.2	.3	.4	.5	.6	.7
1	c	c	d	b	a	c	a
2	F	T	T	F	F		

	Chapter
Control Flow (Part - 2)	6

6.0 Introduction

The order of the execution of any program is based upon the control statements. It decides that which part of the program is to be executed and how many times. In the previous part-1 we studied that how a part of the program can be executed no. of times (again and again) with the help of counters whose condition can be tested with the help of if statements, however it has limitations that it can be performed on certain specific conditions. For a loop to be correctly executed a counter must be initialized, incremented & every value of the counter should be in accordance with the condition specified. Usually, the if-statement does not indicate the number of times the loop is to be executed.

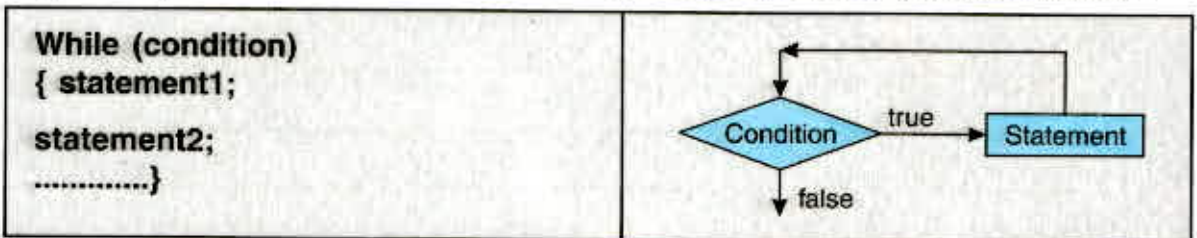
In this chapter, we will make use of the “**for**” and “**while**” loops to execute the part of the program again and again (until required) and will also apply it in some common programs.

6.1 Control loop structures

Many programs require a group of instructions to be executed repeatedly, until some logical conditions has been satisfied. These loop structures are:

6.1.1 The while statement

The while statement, also called the while loop, executes a block of statements as long as a specified conditions is true. The while statement has the following form:



Condition is any C expression, and statement is a single or compound C statement. When program execution reaches while statement, the following events occur:

1. The expression condition is evaluated.
2. If conditions evaluates to false (that is, zero), the while statment terminates, and execution passes to the first statement of following statement.
3. If condition evaluates is true (that is, nonzero), the C statement(s) in statement block are executed.
4. Execution returns to step 1.

Example : Program to find the sum of first n integers (use of while loop)

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

```

{ int n, a, sum = 0;
printf("Enter the value of N\n");
scanf("%d", &n);
a = 1;
while(a<n)
{
sum+ = a;
a++;
}
printf("Sum = %4d\n", sum);
}
/*end of while*

```

Output:

```

Enter the value of N
10
Sum = 55

```

Example : A simple while statement

/*Demonstrate a simple while statement to display first ten numbers */

```

#include<stdio.h>
int count;
int main()
{
count = 1;
while(count <=10)
{
printf("%d\t", count);
count++;
}
return 0;
}
/* global declaration 8/
/* print the numbers 1 through 10
/* or count = count +1 */

```

Output :

```

1 2 3 4 5 6 7 8 9 10

```

6. 1. 2 Do while

The syntax for the do while is:

```
do {  
    statement1;  
    statement2;  
} while (expression);
```

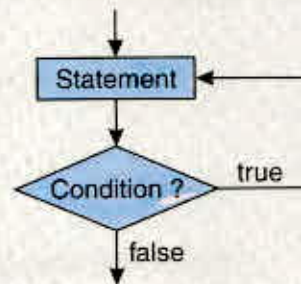
The expression is evaluated. The block of the statements are executed first. Then statements are executed repeatedly until the value expression is false.

Example:

In this program the loop is executed nine times & when the value of x becomes 10, the control comes out of the loop.

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

The block of statements after do is executed at least once. Then the condition is tested. So the loop is executed nine times.



Example:

In example, if we write While (x>10) see the difference in the output as:

x is 1

- x is assigned with 1.
- prints as x is 1.
- x is incremented by 1. so x is 2 now.
- Tests for condition x>10
- since it is false, it comes out of the loop

So, this program prints once as x is 1

Example : Calculate the average of n numbers

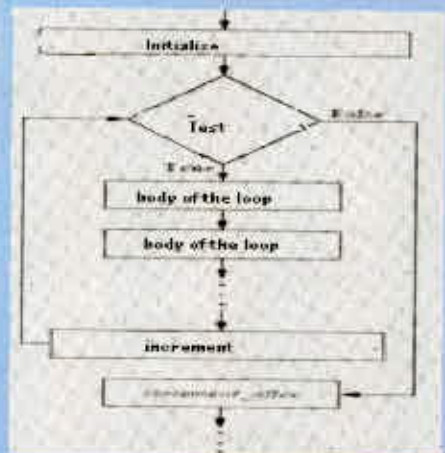
```
#include<stdio.h>  
void main()  
{  
    int n, count = 1;  
    float x, average, sum = 0;          /* initialize & read in a value for n */  
    printf("How many numbers ?");  
    scanf("%d", &n);                    /* read the numbers */  
    do  
    {  
        printf("x =");  
        scanf("%f", &x);  
        sum+ = x;  
    }  
    while(count <= n);                  /* calculate the average & write the answer */  
    average = sum / n;  
    printf("\n the average is %f\n", average);  
}
```

6. 1. 3 For Statement (loop)

for statement is suited for the problem where the number of times a statement to statement block will be executed is known in advance.

```
for(expression1 ; expression; expression3)
{
    action-statements
}
```

The expression1 is evaluated first and then the expression2 is evaluated to check if the loop has to be repeated. At the end of executing the statement, expression3 is evaluated. Thus expression1 may normally be initialization of the loop counter and expression3 the increment/decrement of the loop counter.



Example : `for(count = 1; count<=10; count = count +1)`

- When the for statement is executed for the first time, the value of the count is set to an initial value 1
- Now the condition `count <= 10` is tested. Since count is 1 the condition is satisfied and the body of the loop is executed for the first time.
- Upon reaching the closing brace of **for**, the computer sends the control back to the for statement, where the value of the count gets incremented by 1.
- Again the test is performed to check whether the new value of count exceeds 10.
- If the value of the count is still within the range 1 to 10, then statement within the braces of for are executed again.
- The body of **for** loop continues to get executed till count doesn't exceed the final value 10.
- When the count reached the value 11 the control exits from the loop and is transferred to the statement if any immediately after the body of for.

Example : Program to prints sum of the first ten natural numbers(use of for loop)

```
#include<stdio.h>

void main()
```

```

{
int sum, number;
sum = 0;    /*initialize sum to 0*/
for (number = 1; number<=10;number++)
sum += number;
printf("sum = %d\n",sum);
}

```

Output:

Sum = 55

Explanation

The first time for statement executes it initializes the value of the sum to 0 and the value of the number to 1. C then tests to see if it has to execute the body of for by evaluating the expression `number <= 10`, since the value of number is 1, it executes the body of for. C then evaluates the third expression which increments number by one to prepare for the next iteration of the loop. C again tests to see if it has to execute the body of **for** by evaluating expression2, `number <= 10`. Since the value of the number is now 2. C executes the body of **for** again. When it has executed the body of **for** ten items and prepared for the next iteration, the value of the number is 11. This time the test in expression2 is false and control is passed to the `printf` statement.

Example : Program to print

```

/* for loop header */
#include<stdio.h>
void main()
{
int i;
for(i = 1; i<= 5; i = i+1)    /* for loop braces not required as body contains only
one statement */
printf(" i = %d", i);
}

```

In this example, the initial value for `i` is 1. The test to determine whether to continue is to see if `i` is less than or equal to 5. The statement `i = i + 1` means that at the end of each pass through the body of the loop, the control variable `i` will increase by 1 unit it reaches 6 (and this value causes the test controlling the loop to become false.)

The body of the loop comes after for loop header

- The variable *i* start at 1, which is immediately compared to the final value of 5. Since 1 is less than or equal to 5, the program enters the body of the loop and executes the `printf()` instruction. Naturally, the print instruction prints the value 1. This completes the first pass through the body of **for** loop.
- Then we increment *i* by 1 to 2 and compare this new value to 5. Since 2 is less than or equal to 5, we execute the body of the loop. This time the `printf()` prints 2 on a new line. This completes the second pass.
- Then *i* increases to 3, which is less than or equal to the limiting value of the control variable (5), so we print 3.
- Similarly, *i* increases to 4, and we print 4.
- Then *i* increases to 5, which is also allowed since we test whether the current value is less than or equal to the final one. The print prints 5.
- Then *i* increases to 6. At this point the condition $i \leq 5$ is no longer true. Therefore, we do not enter the body of the loop with *i* have the value 6. Instead, **for** loop is completed, and we continue with the next statement. In our example, since the body of the loop consists of a statement to print the value of *i*, the entire for loop prints the values 1, 2, 3, 4 and 5, each on a separate line.

Example : Print the numbers from 4 to 9 and their squares.

```
#include<stdio.h>
void main()
{
    int number, sqnumber;
    for(number = 4; number <=9; number = number +1)
    {
        sqnumber = number * number;
        printf("%d %d\n",number, sqnumber);
    }
}
```

Output

4	16
5	25
6	36
7	49
8	64
9	81

Explanation

- When **for** loop statement is executed, number starts with the value 4 (what is in sqnumber at this point?).
- Since this value is less than or equal to the limiting or final value of number (in this case 9), we can execute the body of the loop.
- Inside the opening brackets for the body of the loop, sqnumber is set to number * number, which in this case is $4 * 4 = 16$. The printf function prints the values 4 & 16.
- When we reach the closing bracket, the program automatically goes back to the loop header, where number increases by 1, making it equal to 5.
- Since 5 is less than or equal to the limiting value of 9, we execute the body of the loop.
- This time, sqnumber is set to $5 * 5 = 25$. The printf prints 5 and 25 on a new line.
- Then number increases by 1 again, so that it now has a value of 6.
- The body of the loop is executed, sqnumber is set to $6 * 6 = 36$, and 6 and 36 are printed.
- When number is 7 and 8, we print 7 49 and 8 64.
- After printing, we go back to the header, where number increases to 9.
- Since 9 is less than or equal to 9, we execute the body of the loop once again.
- This time sqnumber is set to $9 * 9 = 81$, and the values 9 81 are printed.
- When number increases to 10, the condition controlling the loop is false.
- Therefore, we do not enter the body of the loop again. Instead we continue with the next statement in the program.

Example : To print number from 1 to 5 without expression1 and expression3

```
#include<stdio.h>

void main()
{ int i = 0; limit = 5;
  printf("Value of i\n");
  for (;i<limit ;)
  {
    i++;
    [printf("%d\n",i);
  }          /* end of for loop */
}
```

Output

value of i

1
2
3
4
5

Here after every pass, the value of i will be incremented within the body of for loop & the first value of i is initialized to zero outside the **for** loop.

Summary

Control Flow

The while, do-while and for statements are the three powerful loop control structures available in C. The while statement is a pre-test loop statement (it is executed if only the expression is true) and the do-while is a post test loop statement (it is executed at least once. Then the expression valuated for repeating the loop.) The for loop is used when the user knows the number of interation to be made.

Sometimes, it become desirable to exit a loop or skip a part of the loop in an iterative process. To do so, the break and continue statements are used.

Exercise

Q. 1 Fill in the blanks

1.1	While loop can be executed for minimum _____ no. of time ?
1.2	General form of do while statement is _____ ?
1.3	Do while loop can also be called _____ ?

Q. 2 State True or False

2.1	In while do loop, expression is executed first ?
2.2	In do while loop, block of statements executed first ?
2.3	Whether while (i=20) {---} statement can be converted into an infinite loop ?

Q. 3 Short Answer type question

3.1	What are the three different types of expression for for statement ?
3.2	When the loop is executed then what will be the minimum count in the do while body ?
3.3	In loop how the null statement can be displayed ?
3.4	How the execution of while loop can be terminated ?

Q. 5 Complete the following programs (1 to 5) for following output

<p>5.1) <code>for (a = 1; a <= 6; a++)</code> <code>{ if(a %2)</code> <code>continue;</code> <code>else</code> <code>printf("%d\n", a);</code> <code>printf("end of loop\n");</code> <code>}</code></p> <p>Output: 2 end of loop 4 end of loop 6 end of loop</p> <p>5.3) <code>a = 0;</code> <code>do {</code> <code>if(a<3) {</code></p>	<p>5.2) <code>a = 0;</code> <code>while (a<5)</code> <code>{ if (a<2) {</code> <code>a += 2;</code> <code>continue;</code> <code>}</code> <code>else</code> <code>printf("%d\n", ++a)</code> <code>printf("end of loop \n");}</code></p> <p>Output 3 end of loop 4 end of loop 5 end of loop</p> <p>5.4) <code>inc x;</code> <code>x = 7;</code> <code>while (x>=0)</code></p>
---	--

```

a += 2;
printf("%d\n",a);
continue; }
else {
printf("%d\n", ++a);
break; }
} while (a<5);

```

Output

2
4
6

```

{ x = x -2;
printf("%d\n",x); }

```

Output

7
5
3
1

5.5)

```

int x;
x = 7;
while (x >=0)
x = x -2;
printf("%d\n",x);

```

Output

-1

Q. 6 Complete the following program and find the output ?

6.1)

```
main()
{
int a;
for(a = 0; a<10;a++)
{
:
}
}
```

6.2)

```
int x;
x = 7;
While (x >= 0)
x = x - 2 ;
printf ("%d/n", x);
```

Output
-1

6.3)

```
main()
{
int j;
j = 0;
while (++j <10)
printf("GOOD\n");
printf("j = %d\n",j);
}
```

6.4)

```
main()
{
int a, b, c;
b = 10;
while (b<100)
{ a = b++;
c = ++b }
printf("a = %d\n",a);
printf("b = %d\n",b);
printf("c = %d\n",c);}
```

6.5) **Using For Loop:**
To increment or decrement a variable by more than one number (correct the program so that the output is as shown)

6.6) **Use of commas in the for loop**
(multiple initialization with command expressions), correct the program for given output

#include<stdio.h>

<pre>#include<stdio.h> main() { int num; for (num = 1; num<=100;num+=20) /* increment */ { printf("%d\n",num); printf("\n");} for(num = 100; num>=1;num-=20) /* decrement */ printf("%d\n",num);} Output : 1 21 41 61 81 100 80 60 40 20</pre>	<pre>main() { int num, total, square, cube; for(total = 0, square = 0,cube =0, num = 1; num<=5;++num) { total = total + num; square = square + num*num; Cube = cube + num*num*num; } printf("Total = %d\n",total); printf("Square =%d\n",square); printf("Cube = %d\n",cube); } Output Total = 15 Square = 55 Cube = 225</pre>
--	---

Answer to objective questions

Questins	.1	.2	.3
1.	D	do statement while(expression);	post test loop
2.	T	T	T

Arrays (Part - 1)

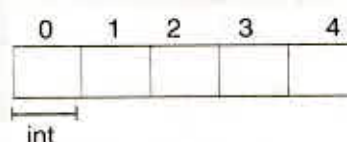
7.0 Introduction

Array is a set of data type elements that shares the same name.

Arrays are the data type and allow us to use single variable to store a number of values with same data type. Each constitute value of the array is called element. Each element of an array has its own storage location, which can be referenced like any other variable. The elements of arrays are stored in consecutive storage locations in memory and identified by the name of the array followed by a numbers in brackets; these numbers starts from 0 and increases each time by 1. Say each number 0, 1, 2, 3, 4,..... is called *subscript or an index*, individual elements of the array are called subscript variables as shown below.

Example

An array contain 5 integer values of int called *age* could be represented like this. Where each blank panel represents an element of the array, that in this case are integer values of type int. These elements are numbered from 0 to 4 since in arrays the first index is always 0, independently of its length.



if age is of array type with n elements, the array elements are:

age[0], age[1], age[2], age[3], age[4],.....age[n].

7.1 Declaring and initialization of array

like a regular variable, an array must be declared before is used.

Data_type array_name [number of elements in array]

Where type is a valid type (like int, float...), name is a valid identifier and the elements field (which is always enclosed in square brackets[]), specifies how many of these elements the array has to contain.

Let's start by looking at a single variable used to store a person's age.

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

```
int main()
{
    short age;
    age=23;
    printf("%d\n", age);
    return 0;
}
```



The Variable age is created as a short datatype and a value is assigned to age. Finally, age is printed to the screen.

Now let's keep track of 4 ages instead of just one. We could create 4 separate variables, but 4 separate variables have limited appeal. Rather than using 4 separate

variables, we'll use an array. Here's how to create an array and one way to initialize an array:

```

1: #include <stdio.h>
2:
3: int main( )
4: {
5:     short age[4];
6:     age [0] =23;
7:     age [1]=34;
8:     age [2]=65;
9:     age [3]=74;
10:    printf("%d\n", age[0]);
11: }
```

On line (5), an array of 4 short datatype is created. Values are assigned to each variable in the array online (6) through line (9). The program above shows that the first element of an array is accessed with the number 0 rather than 1. The elements field within brackets [] which represents the number of elements the array is going to hold, must be a constant value.

The following assignment statements are valid in arrays

- a) x[0]= 15; c.) x[5]+ = 1; e) char c[2] g) b = age[a+2];
b) x[6]= x[6]+6+ x[5]; d.) x[5] = x[12]+ 6 ; f) float b[4]; h) age[age[a]] = age[2] + 5

7.1.1. Initializing arrays

Initialization is nothing but assigning some values to the variable that undergoes processing. Like the initialization to ordinary variable, the individual elements of an array can be initialized. All the initial values must be constants. Initialization can be made to all array elements during the time of declaration.

Date type array_name[size] = {element1, element2, element3, element4,.....element n}

Example

int age [5] = { 16,2, 77, 40, 71}; This declaration would have created an array like this:

	0	1	2	3	4
age	16	2	77	40	71

7.2 Some special rules

1. Number of initializer can be less than the number of array elements

```
int ary[5] = {2, 5, 8} ;
```

/ here only 1 first 3 elements of array are initialized*/*

2. Number of initializers larger than number of elements results in error.

3. **int limit [] = {0,1,2,3,8};** is same as **int limit{4} = {0,1,2,8};**

Here no number is supplied for the size of the array. In this case the compiler counts the number of elements in initialization list and fixes that as the array size.

Example: This program illustrates the initialization of the five integers to the corresponding elements of a array using for statement.

```

#include <stdio.h>
void main()
{
    int marks [ ] = {10, 25, 20, 35};
    int x ;
    printf("Element of the array are\n");
    for (x = 0 ; x < 4 ; x++)
        printf("marks [%d] =%d\n " , x, marks[x]) ;
}    /* End of main */
```

Output : Element of the array are

marks [0] = 10

marks [0] = 25

marks [0] = 20

marks [0] = 35

Problem **What does this loop do ?**

```
for(x = 0 ; x <3 ; x++)
```

```
num[x] = x * 3;
```

it assigns each element of array num a value which is three times its subscript: that is,

num [0] gets the value 0

num [1] gets the value 3

num [2] gets the value 6

7.2.1 Entering Data into an Array

Since array represents a collection of values, so to input data into the individual elements of the array, we make use of loop statements. If the actual size of the array is pre defined, then for loop, is preferred to while loop or do - while loop.

Here is the section of code that places data into an array:

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    clrscr();
    int i;
    printf ("Enter marks for 5 students:");
    for (i=0; i<5; i++)
    {
        printf("\nMarks of student %d:", i+1);
        scanf("%d", &marks[i]);
    }
}
```

Output:

```
Enter marks for 5 students
Marks of student 1: 45 (press enter)
Marks of student 2: 49 (press enter)
Marks of student 3: 67 (press enter)
Marks of student 4: 98 (press enter)
Marks of student 5: 76 (press enter)
```

The "**for loop**" causes the process of asking for and receiving a student's marks from the user to be repeated 5 times. The first time through the loop, i has a value 0, so the scanf() statement will cause the value type to be stored in the array element marks[0], the first element of the array. The loop will be continue to be executed untill i becomes 4 (i<5).

In the **scanf ()** statement, we have used the "address of" operator (&) on the element marks[i] of the array. In so doing, we are passing the address of this particular array element to the scanf() function, rather than its value,

clrscr()

It is used to clear the data that is displayed on the screen. For this function **conio.h** is used before main().

How about printing out each of the values separately ? Try this:

```
1: #include <stdio.h>
2:
3: int main()
4: {
5:     short age[4];
6:     age[0]=23;
7:     age[1]=34;
8:     age[2]=65;
9:     age[3]=74;
10:    printf("%d\n", age[0]);
11:    printf("%d\n", age[1]);
12:    printf("%d\n", age[2]);
13:    printf("%d\n", age[3]);
14:    return 0;
15: }
```

This program stores ages through the array age with given values 23,34,65,74. these are displayed on the scree as

age = 23

age = 34

age = 65

age = 74

Note: There is no single statement in the language that says "print an entire array to the screen". Each element in the array must be printed to the screen individually.

7.3 Copying arrays

Let's try copying arrays using a technique similar to the technique used to print arrays (that is, one element at a time):

```
1: #include <stdio.h>
2:
3: int main()
4: {
5:     short age[4];
6:     short same_age[4];
7:
8:     age[0]=23;
9:     age[1]=34;
10:    age[2]=65;
11:    age[3]=74;
12:
13:    same_age[0]=age[0];
14:    same_age[1]=age[1];
15:    same_age[2]=age[2];
16:    same_age[3]=age[3];
17:
18:    printf("%d\n", same_age[0]);
19:    printf("%d\n", same_age[1]);
20:    printf("%d\n", same_age[2]);
21:    printf("%d\n", same_age[3]);
22:    return 0;
23: }
```

This technique for copying arrays works fine. Two arrays are created: age and same_age. Each element of the age array is assigned a value. Then, in order to copy of the four elements in age into the same_age array, we must do it element by element.

copy first element of the array



copy second element of the array

Note: Like printing arrays, there is no single statement in the language that says "copy an entire array into another array". The array elements must be copied individually.

The technique used to copy one array into another is exactly the same as the technique used to copy 4 separate variables into 4 other variables. So what is the advantage of using arrays over separate variables ?

One significant advantage of an array over separate variables is the name. In our examples, using four separate variables requires 4 unique names. The 4 short variables in our array have the same name, age. The 4 short's in the array are identical except for an index number used to access them. This distinction allows us to shorten our code in a way that would be impossible with 4 variables, each with unique names:

Since the only difference between each of the short's in the arrays is their index, a loop and a counter can be used to more easily copy all of the elements. The same technique is used to shorten the code that prints the array to the screen.

```

1:  #include <stdio.h>
2:
3:  int main()
4:  {
5:      short age[4];
6:      short same_age[4];
7:      int i, j;
8:      age[0]=23;
9:      age[1]=34;
10:     age[2]=65;
11:     age[3]=74;
12:
13:     for(i=0; i<4; i++)
14:         same_age[i]=age[i];
15:
16:     for(i=0; i<4; i++)
17:         printf("%d\n", same_age[i]);
18:     return 0;
19: }
```

7.4 Accessing the values of an array

In any point of a program in which an array is visible, we can access the value of any of its elements individually as if it was a normal variable, thus being able to both read and modify its value. Ther format is as simple as:

name[index];

Following the previous examples in which age had 5 elements and each of those elements was of type int, the name which we can use refer to each element is the following:

Array name	age[0]	age[1]	age[2]	age[3]	age[4]
age					

Example To store the value 75 in the third element of age, we could write the following statement;

age[2] = 75;

and, for example, to pass the value of the third element of age to a variable called a, we could write:

a = age[2];

Therefore, the expression age[2] is for all purposes like a variable of type int.

Notice That the third element of age is specified age[2], since the first one is age[0], the second one is age[1], and therefor, the third one is age[2]. By this same reason, its last element is age[4]. Therefore, if we write age[5], we would be accessing the sixth element of age and therefor exceeding the size of the array. Array elements can be accessed by writing the name of the array and the position of the index of the item in array.

This expression is equivalent to the value of the array at the index.

```

int data[2];
data[0] = 30;
```

```
data[1] = 20;
```

```
printf("value at data[1] is %d\n",data[1]);
```

At this point it is important to be able to clearly distinguish between the two uses that brackets [] have related to arrays. They perform two different tasks: one is to specify the size of arrays when they are declared; and the second one is to specify indices for concrete array elements. Do not confuse these two possible uses of brackets [] with arrays.

int age[5];	// declaration of a new array
age[2] = 75;	//access to an element of the array

Some other valid operations with arrays:

```
age[0] = a;
```

```
age[a] = 75;
```

```
b = age [a+2];
```

```
age [age[a]] = age[2] + 5;
```

7.5 Manipulation of array elements:

C does not support performing any operation on the entire array. This is, the whole array cannot process as a single element. But, it allows the programmer to perform certain operations on a element by element basis.

Array elements can be used for mathematical operations such as adding the different values of array element, finding the product of array elements. There are two ways to do these operation one by using the values given for their initialization, secondly by supplying values through the user (keyboard).

Here an array age has been initialized by five values and these are added to give sum of ages for loop will proceed for 5 times to accept the assigned values. The variable result give the output as the sum of the ages given in initialization.

```
// arrays example
```

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

```
int age[ ] = {16, 2, 77, 40 10};
```

```
int n, result=0;
```

```
{
```

```
    for ( n=0 ; n<5 ; n++)
```

```
    {
```

```
        result += age [n];
```

```
    }
```

```
    printf("\n the sum of ages is = %d",  
    result); return 0;
```

```
}
```

Output: the sum of ages is = 145

7.5.1 Sum of Elements

let's write a program to add the array element supply via user through keyboard.

Program 1 : Sum of the elements in an array

```
#include<stdio.h>

void main()
{
    int a[5],i, sum=0;
    for (i=0;i<5;i++)
    {
        printf("Enter the value of element%d, , i+1);
        scanf("%d", &a[i];
    }
    for (i=0;i<5;i++)
    {
        sum=sum+a[i];
    }
    printf( "The sum of all the elements of the array is %d",sum);
}
```

Output:

Let's say the following numbers are inputted in the array a

Enter the value of element1 : 10

Enter the value of element2 : 12

Enter the value of element3 : 14

Enter the value of element4 : 12

Enter the value of element5 : 30

The sum of all the element of the array is 78

Program 2 : Product of all the elements in an array

```
#include<stdio.h>

void main()
{
    int a[10], i, prod=1;
    for (i=0;i<=9;i++)
    {
        printf("enter the element value");
        scanf("%d", &a[i];
    }
    for (i=0;i<=9;i++)
    {
        prod=prod * a[i];
    }
    printf( "The product of all the elements of the array is %d",prod);
}
```

Output

Let's say the following numbers are inputted in the array a

i.e 2 2 2 2 2 2 3 3 3 3 3

Finally the product of all the elements is stored in the variable prod which will be equal to 7776.

So the output seen in the screen will be

The product of all the elements of the array is 7776

Program 3 : Product of the respective elements in two one dimensional arrays

```
#include<stdio.h>
void main ()
{
    int a[10], b[10],c[10],i, prod=1;    /* input the a array*/
    for (i=0;i<=9;i++)
    {
        printf("enter the element value");
        scanf("%d",&a[i];
    }                                     /* input the b array */
    for (i=0;i<=9;i++)
    {
        printf("enter the element value");
        scanf("%d",&b[i];
    }                                     /* product of all the elements in both the arrays
    */
    for (i=0;i<=9;i++)
    c[i]= a[i] *b[i]                    /* point the array where product of both a & b arrays is stored
    i.e. array c*/
    printf("The product of both the arrays a and b are stored in c whose values are\n");
    for (i=0;i<=9;i++)
    {
        printf("%d\t",c[i];
    }
}
```

Output

Let's say the following numbers are inputted in the array a; i.e 22222 33333

& if the following numbers are inputted in the array b: 33333 22222

Finally the product of all the elements is stored in the array C. So the output seen in the screen will be:

The product of both the arrays a and b are stored in c whose values are: 6666666666

Program 5 : Linear Search: Searching for a value in the array

```
#include<stdio.h>
void main()
{
    int a[10],i, sum=0,val,c=0;           /* input the array*/
    for (i=0;i<=9,i++)
    {
        printf("enter the element value");
        scanf("%d",&a[i]);
    }                                     /* input the value to be searched*/
    printf("enter the element value to be searched");
    scanf("%d",&val);                   /* search in all the elements in the array */
    i=0;
    while (i<=9)
    {
        if (val == a[i])
        {
            c=1;
            break;
        }
        else
            c=0;
        i++;
    }                                     /* print */
    if (c==1)
        printf("The Value found is %d at location a[%d]:,val,i+1)
    else
        printf("The Value does not exist in the array");
}
```

Output:

Let's say the following numbers are inputted in the array a i.e 1 2 3 4 5 6 7 8 9 10
when it asks for Enter the value to be searched you enter 5, so the output seen in the
screen will be:

The value found is 5 at location a[5]

When it asks for Enter the value to be searched you enter 90, so the output seen in
the screen will be:

The value does not exist in the array

Program 6: To find the maximum number in the array

```
#include<stdio.h>
void main ()
{
    int a [10],i, max;           /* input the array */
    for (i=0;i<=9;i++)
    {
        printf("enter the element value");
        scanf("%d",&a[i]);
    }
    max=a[0];                   /* assume that the first element is the maximum */
    for (i=1;i<=9;i++)          /* check max with the other elements and if some element
                                is found greater than the assumed value stored in max,
                                max is changed to the new value */
    {
        if (max<a[i])
            max=a[i];
    }
    /* print the maximum number */
    printf("The maximum of all the elements of the array is %d",max);
}
```

Output:

Let's say the following numbers are inputted in the array i.e 1 2 3 4 5 6 7 8 9 10

So the output seen in the screen will be

The maximum of all the elements of the array is 10

Program 7: To find the minimum number in the array

```
#include<stdio.h>
void main ()
{
    int a [10],i, min;           /* input the array*/
    for (i=0;i<=9;i++)
    {
        printf("enter the element value");
        scanf("%d",&a[i];
    }

    min=a[0];                   /*assume that the first element is the minimum */
    for (i=0;i<=9;i++)          /* check min with the other elements and if some element
                                is found lesser than the assumed value stored in min,
                                min is changed to the new */

    {
        if (min>a[i];
        min=a[i];
    }                            /* print the minimum number */

    printf("The minimum of all the elements of the array is %d:",min);
}
```

Output:

Let's say the following numbers are inputted in the array a: 7 8 9 10 1 2 3 4 5 6

So the output seen in the screen will be:

The minimum of all the elements of the array is 1

Let us consider an example of arrays where 10 values of age are input by user. The sum and average of ages is evaluated, maximum and minimum ages are displayed

```

#include<stdio.h>

int main ()
{
    int age [10];
    int i, sum=0, avg=0;
    int max=0,min=100;
    for(i=0;i<10;i++)
    {
        printf("Enter the age of student%d:",i+1);
        scanf("%d",&age[i]);
    }
    for(i=0;i<10;i++)
    {
        sum=sum+age[i];
        if (age[i]>max)
        {
            max=age[i];
        }
        if(age[i]<min)
        {
            min=age[i];
        }
    }
    avg=sum/10;
    printf("Average age of the students of the class : "
    ,avg"); printf("Maximum age of the student of the
    class : ","max");
    printf("Minimum age of the student of the class : "
    ,min");
    return(0)
}

```

The result of the program is:-

```

Enter the age of the student 1
13
Enter the age of the student 2
14
Enter the age of the student 3
15
Enter the age of the student 4
12
Enter the age of the student 5
11
Enter the age of the student 6
14
Enter the age of the student 7
14
Enter the age of the student 8
17
Enter the age of the student 9
18
Enter the age of the student 10
13
Average age of the students of the class : 14
Maximum age of the student of the class : 18
Minimum age of the student of the class : 11
Press any key to continue_

```

In the program the array is declared by the statement

int age[10];

age is the identifier of the array which is of type integer and whose size is 10. In the for loop user is asked to enter the age of the student, the age is stored in the element of the array by the statement

scanf("%d",&age[i]);

Here i is the index of the array. The index starts from 0. In the next for loop, average age, minimum age and maximum age of the students is calculated.

Each element is accessed by index i . The sum of the elements of the array is calculated by the statement

```
sum=sum+age[i];
```

age[i] is the $(i+1)$ th element of the array.

The maximum age is calculated by the if statement

```
if(max>age[i];  
{  
    max>age[i];  
}
```

The minimum age is calculated by the another if statement

```
if (min<age[i]  
{  
    min=age[i];  
}
```

The average is calculated by the statement

```
avg=sum/10;
```

The total n. of bytes used to store an array is equal to

Total bytes=size of(base type)×size of array.

Summary

Arrays

an array is a data structure used to store collections of similar values under a given name. They are also called subscripted variables.

Elements of array stored in contiguous memory locations

These elements are accessed by their name of the array followed by the subscript number within brackets.

Exercise

Q.1 Multiple Choice Questions

1.1 Array name is ?

- (a) an array variable
- (b) A keyword
- (c) Common name shared by all elements
- (d) None of these

1.2 In C, array subscript always starts at ?

- (a) -1
- (b) 0
- (c) 1
- (d) any value

1.3 In c, square brackets [] are used in ?

- (a) functions
- (b) arrays
- (c) statements
- (d) all of these

1.4 maximum no. of elements in the declaration `int a [4] [5];` is ?

- a) 28
- b) 32
- c) 20
- d) 9

1.5 Given declaration `int a[10];` identify when of following is wrong ?

- a) `a[-1]`
- b) `a[0]`
- c) `a[10]`
- d) `++a`

Q.2 State whether True or False

2.1 An array is a collection of values of same type ?

2.2 Elements of the array are placed next to each other in the memory ?

2.3 Arrays can be initialized in the declaration statement itself ?

2.4 If the size of the array is less than the number of initializers, it is an error ?

Q.3 Write an appropriate word or output for the following

3.1 If we omit size of the array in `x [size]` from the declaration of array `x`, then what type of variable does the computer think `x` is ?

3.2 The value within [] in an array declaration specifies size of ?

3.3 When a loop is preferred in arrays, the size of array is always ?

Q.4 Short answer type questions

4.1 Show what is stored in the cells of the array when we write
`int x[5] = {8, -2, 20, 26, -1000};`

- 4.2 How a real array named data with 50 elements will be declared ? What will be the subscript of the last element ?
- 4.3 What does this statement imply ?
`a = 4; num [a]=7;`
- 4.4 Do the two definitions allocate the same number of cells ?
`char Characters[1000];`
`int numbers[10][1000];` justify your answer ?
- 4.5 What is the purpose of initializing an array ?

Q.5 Give the answers for the following Questions

- 5.1 Differentiate between an array and an ordinary variable ?
- 5.2 What are the basic rules for naming arrays ?
- 5.3 Answer the following about the declaration `int amount[25];`
 a) How many elements are in the array ?
 b) What are the upper and lower bounds of the array ?
 (Hint upper bound = 24, lower bound = 0 why ?)
- 5.4 What happens if the number of initializing values is greater than the dimension specified for the array ?
- 5.5 Given this declaration, answer the following
`int item [25], amount [25], a;`
 a) if `a = 8`, what array element is referred to by the item `[a-3]` ?
 b) if `a` is 24, what element is amount `[a+1]` ?

Ans i) item [5] ii) amount [25]

Answer to objective questions

Q.	.1	.2	.3	.4	.5	.6	.7	.8	.9	.10
1	c	b	b	c	a					
2	T	T	T	F						
3	Simple variable	array	known							

	Chapter
Arrays (Part - 2)	8

8.0 Introduction

Ordinary arrays are indexed by a single integer. In the concept of a *multi-dimensional* array, in which, we index the array using an ordered list of integers, such as in `a[3], [1], [5]`. The number of integers in the list used to index into the multi-dimensional array is always the same and is referred to as the array's dimensionality, and the bounds on each of these are called the array's dimensions. An array with dimensionality *k* is often called *k-dimensional*. One-dimensional arrays correspond to the simple arrays discussed thus far; two-dimensional arrays are particularly common representation for matrices. In practice, the dimensionality of an array rarely exceeds three. Mapping a one-dimensional array into memory is obvious, since memory is logically itself a (very large) one-dimensional array.

If the number of subscripts is more than one, then such arrays are called multi dimensional arrays. Thus, we can say 2 dimensional (2 D) array, 3 dimensional (3 D) array and so on. The dimensionality is understood from the number of pairs of square brackets placed after the array named.

Example

1. `array 1 []` → one-dimensional array
2. `array 2 [][]` → two-dimensional array
3. `array 3 [][][]` → three-dimensional array

Programmer writes the numbers of elements between the square brackets that will be used for the function of the program. In the below part, two dimensional array has been discussed in detail.

8.1 Two dimensional array

It is an ordered table of homogeneous elements. It is generally, referred to as a matrix, of sum rows and sum columns. It is also called as two subscripted variables.

8.1.1 Declaration of Two dimensional array(s)

In C language, the syntax of the declaration of two dimensional array(s) is:

`data type array_name [row] [columns];`

where row --> number of elements that will be processed under subscript 1.

columns --> number of elements that will be processed under subscript 2.

Example

1. `int marks [5] [13];`
2. `float matrix [3] [3];`
3. `char page [25] [80];`

The first example (`int marks [5] [13];`) indicates that `int marks` are an two dimensional arrays having 5 rows and 3 columns.

In the second example (`float matrix [3] [3];`) `float matrix` is a 2 dimensional array having 3 rows and 3 cloumns.

In the third example (`char page [25] [80];`), `page` is character type 2 dimensional array with 25 rows and 80 columns

8.1.2 Let us see representaion of 2 dimensional array

The array contains three rows and four columns, so it is said to be a 3-by-4 (3×4) arrays. In general, an array with *m* rows and *n* columns is called an *m*-by-*n* (*m*×*n*) array.

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>A[0][0]</code>	<code>A[0][1]</code>	<code>A[0][2]</code>	<code>A[0][3]</code>
Row 1	<code>A[1][0]</code>	<code>A[1][1]</code>	<code>A[1][2]</code>	<code>A[1][3]</code>
Row 2	<code>A[2][0]</code>	<code>A[2][1]</code>	<code>A[2][2]</code>	<code>A[2][3]</code>

Say `[1][2]` means that `[1]` is row subscript where as `[2]` is column subscript. Also every element of the arrays is identified by `A[i][j]`; *A* is the name of the array, & *i* & *j* are the subscript that uniquely identifying each element in *A*.

8.1.3. Initialization of 2-dimensional array

Like, initialization of one-dimensional array elements. two-dimensional array elements can also be initialized at the time of declaration. The syntax for a two-dimensional array initialization is as follows:

Example 1 let us see the below given declaration

```
int matrix [3] [3] = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Then, the first 9 elements of the matrix will be,

```
matrix [0] [0] = 1;    matrix [0] [1] = 2;    matrix [0] [2] = 3;
```

```
matrix [1] [0] = 4;    matrix [1] [1] = 5;    matrix [1] [2] = 6;
```

```
matrix [2] [0] = 7;    matrix [2] [1] = 8;    matrix [2] [2] = 9;
```

If the number of elements to be assigned are less than the total number of elements that two-dimensional array contained, then all the remaining elements are assigned zeros.

Example 2 **Float xy[2] [2] = {1.0, 1.5, 2.0}**

Here, `xy` is a two-dimensional array of 2 × 2 elements of type `float`. But, only the first 3 elements are initialized. The fourth elements is then automatically set to zero as shown below:

```
xy [0] [0] = 1.0      xy [0] [1] = 1.5
```

```
xy [1] [0] = 2.0      xy [1] [1] = 0.0
```

Example 3

```
int num [2] [2] = {{1,2}, {3,4}} ;
```

Here, the inner set {1, 2} is taken for assigning values 1 and 2, respectively, to the first row element of an array num. Thus,

```
num [0] [0] = 1    num [0] [1] = 2
```

Similarly, the other set {3, 4} is taken for assigning its values 3 and 4, respectively, to the second row elements of an array num. Thus,

```
num [1] [0] = 3    num [1] [1] = 4
```

8.1.4 Initialization of the two-dimensional array elements

Remember the following points while dealing with the initialization of the two-dimensional array elements.

The number of sets of initial values must be equal to the number of rows in the array. One-to-one mapping is preserved i.e. the first set of initial values is assigned to the first row elements and the second set of initial values is assigned to the second row element and so on.

If the number of initial values in each initializing set is less than the number of corresponding row elements, then all the elements of that row are automatically assigned to zero.

If the number of initial values in each initializing set exceeds the number of the corresponding row elements then there will be a compilation error.

Before discussing the processing of the two dimensional arrays or multi-dimensional array, let us see the use of nested loop.

A loop may contain another loop in its body. This form of a loop is called nested loop. But in a nested loop, the inner loop must terminate before the outer loop. The following is an example of a nested loop:

<pre>For (i= 1;i<=5;++i) // outer loop { printf("\n") ; For (j=i;j<=1;--j) // inner loop printf("***") ; }</pre>	<p>The sample output will be:</p> <pre> * ** *** ****</pre>
--	--

The inner for loop is executed for each value of i. The variable i takes values 1,2,3 and 4. The inner loop is executed once for i= 1 according to condition j = 1 (i.e, 1<=1 means once), twice for i = 2, three times for i = 3 and four times i = 4.

While working with nested loops, you need to understand one thing and that is, the value of outer loop variable will change only after the inner loop is completely finished or interrupted. To understand this, consider the following code lines.

```
for (outer =1 ; outer <10 ; outer +=4)
{
    for (inner =1 ; inner <=outer ; inner +=2)
        printf("%d%4d ", outer, inner);
}
```

The above code will produce the output as:

	1	1	← Inner loop got over after value 1's iteration
Change in outer's value	5	1	← Inner loop got over after value 5's iteration
	5	3	
	5	5	
Change in outer's value	9	1	← Inner loop got over after value 1's iteration
	9	3	
	9	5	
	9	7	
	9	9	

The two dimension array is also called as matrix array. Let us consider a sample program that stores roll numbers and marks obtained by a student:

The program on the right side has two parts, in the first part for loop is used to read the values of roll no. & marks, whereas, in second part another for loop is used to print out these values.

	Col no. 0	Col no. 1
Row no 1	5555	88
Row no 2	6666	76
Row no 3	7777	78
Row no 4	8888	72

The scanf () statement :

scanf("%d%d", &stud[i][0], &stud[i][1]);
 having the subscript &stud[i][0], here the first subscript of variable stud [i] is for row number which changes for every student &stud[0], the second subscript tell about zero column which contains the rollno, similarly &stud[1] tells the first column which contains the marks.

So stud[0][0] = 5555 and stud[0][1] = 88

```
main ( )
{
    int stud[4][2];
    int i, j;

    for ( i = 0 ; i <=3; i++)    /*inputting data
    {
        printf("\n Enter roll no. and marks") ;
        scanf("%d %d", &stud[i][0], &stud[i][1]) ;
    }

    for (i = 0 ; i <=3; i++)    /* printing data
        printf("\n %d %d", stud[i][0], stud[i][1];
    }
```

So we can say that array elements are stored row wise and accessed row wise.

8.2 Memory occupation in two dimensional arrays elements

Conceptually elements of the array are stored in a continuous chain. In two dimensional arrays it is as shown:

stud[0][0]	stud[0][1]	stud[1][0]	stud[1][1]	stud[2][0]	stud[2][1]	stud[3][0]	stud[3][1]
5555	88	6666	33	7777	78	8888	72
1002	1004	1006	1008	1010	1012	1014	1016

Example `printf("marks of second student = %d,stud[1][1];`

Output: marks of second student = 33

8.3 Multi dimensional Arrays Character type

Any array of Char with more than one subscript or size specifiers is defined as multidimensional array. The syntax is

storage – class character data – name [{size} --- {size} --- {size}] ;

As each size specifier is enclosed within a set of beackets. Storage-class is optional.

Example `char words [30][15]; or char tens[10][15];`

Each case the 15th character string is being the string terminator '\0'.

8.3.1 Accessing elements of Multidimensional Arrays

The value of the elements of multidimensional array of char are accessed by specifying the name of the variable, row and column of the element. But, in case of array of char, there is a difference between accessing a string and accessing elements of the string which is apart of two dim array of char. to clarify the point, take the following example.

Example

`char input [2][3];`

<code>input[0]</code>	will access all the characters in the string at row 0.
<code>input[1]</code>	will access all the characters in the string at row 0.
<code>input[0][0]</code>	will access first characters in the first string
<code>input[0][1]</code>	will access second characters in the first string
<code>input[0][2]</code>	will access string terminator of the first string
<code>input[1][0]</code>	will access first characters in the second string
<code>input[1][1]</code>	will access second characters in the second string
<code>input[1][2]</code>	will access string terminator of the second string

8.3.2 Initialization of char type multi dimensional array

It is similar to the Initialization of 2-dim array of numbers.

Example

`char words [5][21] = {"programming", "computer", "debug", "syster", "about"};`

If the number of characters read in is less than the specified number which is 20 in the array words. The unfilled places will be filled in by 'x0', the hexadecimal equivalent of the string terminator '\0' as is shown below.

p	r	o	g	r	a	m	m	i	n	g	\x0	\x0	\x0	\x0	\x0	\x0	\x0	\x0
---	---	---	---	---	---	---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----

The `printf("%s,words[1])` will print the string "programming", only 11 characters as the 12th character happened to be a string terminator which will not let `printf` function to proceed further. However the following "printf statement with C-conversation" will print all the characters including string terminator (which will be converted into spaces) of the array `words`.

Processing of text word by word requires a slight amendment to the technique of inputting character. A word may be defined a sequence of non- white consecution characters terminated by a space or punctuation mark. We will deal with those words which are terminated by space to keep the program simple.

C language

Example Program : This program reads and prints the elements of a matrix.

<pre>#include<stdio.h> void main () { int n ,a ,b, row, col; int mat [10][10] ; printf("Enter the order of the matrix\n"); scanf("%d%d", &row , &col); printf("Enter the elements of the matrix\n"); for(a= 0; a<row; a++) { for(b =0; b<col; b++) { scanf("%d" ,&mat[a][b]); } } printf("Matrix is \n") ; for(a= 0 ; a<row; a++) { for(b =0 ; b<col ; b++) { printf("%3d" ,mat[a][b] ; } printf("\n") ; } }</pre>	<p>Output</p> <p>Enter the order of the matrix</p> <p>3 3</p> <p>Enter the elements of the matrix</p> <p>1</p> <p>2</p> <p>3</p> <p>4</p> <p>5</p> <p>6</p> <p>7</p> <p>8</p> <p>9</p> <p>Matrix is</p> <table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9
1	2	3								
4	5	6								
7	8	9								

Example program : To calculate the average of elements of two in 2-dimensional array

<pre>#include<stdio.h> int main () { int int age[2][5]= { {12, 13, 14, 15, 15}, { 12, 16, 17, 13, 12}}; int i,j; int sum=0,avg=0; for(i=0;<2;i++) { for(j =0;j< 5;j++) { sum=sum+age[i][j]; } avg=sum/5; printf("Average of the elements of the row ; %d is %d \n" , i+1 , avg); sum=0; } return(0); }</pre>	<p>Explanation:</p> <p>Output of the program is:</p> <div style="background-color: black; color: white; padding: 10px;"> <p>Average of the elements of the row 1 is 13</p> <p>Average of the elements of the row 2 is 14</p> <p>Press any key to continue</p> </div>
---	--

8.4 Single Character input output

The basic operation done in input output is to read characters from the standard input device such as the keyboard and to output or writing it to the output unit usually the screen. The getchar function can be used to read a character from the standard input device. The scanf can also be used to achieve the function. The getchar has the following form.

variable name = getchar

Variable name is a valid 'C' variable, that has been declared already & that possess the type char.

Example :

```
#include <stdio.h>           // assigns stdio-h header file to your program
void main ( )                // Indicates the starting point of the program.
{
    char C;                   // variable declaration
    printf ("Type one character."); // message to user
    C = getchar ( ) ;         // get a character from keyboard and stores it in variable C.
    printf (" The character you typed is = %c" , C); // output
}                             // Statement which displays value of C on
                             // Standard screen.
```

The putchar function which is analogous to getchar function used for writing character one at a time to the output terminal. The general form is:

putchar (variable name);

(Where variable is a valid C type variable that has already been declared).

putchar (); Displays the value stored in variable C to the standard screen.

Program shows the use of getchar function in an interactive environment.

```
#include<stdio.h>           //insert stdio.h header file into the Pgm
void main ( )                // Beginning of main function.
{
    char in;                  // character declaration of variable in.
    printf (" please enter one character"); //message to user
    in = getchar ( ) ;        // assign the keyboard input value to in.
    putchar (in);             // out put 'in' value to standard screen.
}
```

This program read any character that you have typed on keyboard until it finds a '.' character in input characters

```
#include<stdio.h>
#include<conio.h>
main()
{
    char ch;
    while(ch!='.')
    {
        ch=getchar();
        putchar(ch);
    }
}
```

Each key press will be both shown on console and captured to a buffer. This buffer will be delivered to 'ch' variable after pressing enter key (not before this).

So input characters are buffered until we press enter key and at that moment program execution continues running statements that follow getchar() function (These are loop statements). If there is a '.' character in buffered characters, loop execution continues sending characters to console with putchar() function until it reaches '.' and after that stops the loop and comes out of while loop.

If it does not encounter '.' in characters it returns to the start of loop, where it starts putchar() function again and waits for user input.

First 'test' string in output is the result of our key presses and second 'test' is printed by putchar statement after pressing enter key.

8.5 # define directive:

This belongs to C pre-processor category. As the name itself indicates, (pre-processor) means that it is always placed before its compilation. Hence placed before the main () function. In C programming, we know that the value of the variable can change as a program is executing. However, the value of the constant is fixed by a definition. The general form of the definition for the constant is:

```
# define name value
```

The compiler replaces every occurrence of the identifier name with the value. For example, say in a program using these constant definitions we want every occurrence NUMB is replaced by value 7, then we write.

```
# define numb 7
```

In other view # define directive is also called as Macro substitution (process of replacing an identifier of a C program by a constant or symbolic constant.)

Let us see how it is used.

Example program 1

```
#include <stdio.h>
#define PI 3.142
void main ( )
{
    float rad , area ;
    printf("Enter the radius \n");
    scanf("%f" , &rad) ;
    area = PI * rad * rad ;
    printf("area of a circle = %f\n" , area);
}
```

Example program 2

```
#include <stdio.h>
# define CONDITION if (a>b)
# define PRINT printf("a is greater than b\n") ;
main ( )
{
    int a, b ;
    printf("Enter the value of a and b /n");
    scanf("%d %d " , &a, &b);
    CONDITION
    PRINT
}
```

Note: Any Macro substitution inside the string will not be replaced.

Example

printf("PI = %f\n" ,PI);

in this statement first occurrence of PI will not be replaced by 3.142 while the second is replaced by 3.142.

Now let s # define directive in a program to print Multiplication table

Example program

```
/*Program to display a Multiplication table
#include <stdio.h>
#define ROWS 5
#define COLUMNS 5
void main ( )
{
    int row, column, product [ROWS][COLUMNS] ;
    int i,j;
    printf(" MULTIPLICATION TABLE \n\n") ;
    printf(" ") ;
    for ( j = 1; j <=COLUMNS ;j++)
        printf("%4d",j);
        printf("\n") ;
        printf("_____ \n") ;
    for(i = 0 ; i<ROWS ; i++)
    {
        row = i+1;
        printf("%2d" ,row) ;
        for(j = 1 ;j<=COLUMNS ; j++)
        {
            columns = j;
            product[i][j] = row * columns ;
            printf("%4d",poroduct[i][j]);
        }
        printf("\n") ;
    }
}
```

Explanation:

1. Step 1, 2 defines the declaration of two variables rows and columns.
2. Step 4 we declare a two - dimensional array of size 5 × 5.
3. Step 5 we print Multiplication table. Here, we use two for loops to access the array. The loop for(i = 0 ; i<Rows ; i++) prints the elements row-wise i.e. for each value of i, all the columns are printed
4. In the next step multiplication of row and column take place.
5. number of iteration done by the for loop is equal to the array

Output:

Multiplication table

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

Another way of initialization of multi dimensional array:

Example

```
int age[2][5]= { {12, 13, 14, 15, 15}, {12, 16, 17, 13, 12}};
```

in the above statement, the declaration and initialization of two dimensional is shown. Also here, the initialization of elements of the first row {12, 13, 14, 15, 15} and second row {12, 16, 17, 13, 12} is shown.

Example

```
sum=sum+age[i][j];
```

Array age[i][j] is shown with row i and column j and when sum of the elements of row is calculated the the average is calculated by the statement as shown below:

```
avg=sum/5;
```

Here value of sum starts from zero and is repeated for the second row.

Summary

Arrays

There are two types of arrays - one dimension and multidimensional arrays.

One a pair of square brackets is used for each dimension. For example one dimensional array has only one subscript, and it is viewed as row matrix or column matrix.

But, multidimensional arrays have more than one subscript, hence the name two dimensional array three dimensional arrays and so on/

C does not support strings data type but strings are handled as an array of characters.

There is rich library of functions for manipulating strings.

Exercise

Q.1 Answer the following questions

- 1.1 Define a Multi-dimensional array ?
- 1.2 What are different ways to declare two-dimensional arrays ?
- 1.3 State the condition when all the elements of a row are automatically assigned a ZERO value ?
- 1.4 What is the significance of # define Row 10 ?
- 1.5 A declaration for a 3 dim array sales which holds sales data for five shops, for 12 months, for three years is given as double sales [5][12][3]; state why double data type is used ?

Q.2 Fill in the blanks

- 2.1 Process of replacing an identifier of a C program by a constant_____
- 2.2 The_____function is used to read a character from the standard input device.
- 2.3 The_____function is used for writing characters one at a time to the output terminal.
- 2.4 If the number of characters read in is less than the specified number, The unfilled places will be filled in by_____.
- 2.5 The two dimensional array is generally called as_____.

Q.3 State whether True or False

- 3.1 Elements of two-dimensional array are stored row-wise.
- 3.2 The elements of the array can be accessed in order of their subscript.
- 3.3 A list of strings can be stored within a two dimensional array.
- 3.4 Number of elements in a two-dimensional array having 5 rows and 3 columns is 8.
- 3.5 If the size of an array is less than the number of initializers, the size is increased automatically.

Q.4 Multi Choice questions

4.1 Int marks [10] [5] gives the defination of _____ array ?

- a) 5 column, 10 rows
- b) 10 columns 5 rows
- c) both a and b are correct
- d) none of these

4.2 Statement `int num[2][3] = {{3, 8, 6}}, {9, 4, 7}};`

- a) assigns a value 4 to `num[1][2]`
- b) assigns a value 7 to `num[1][2]`
- c) assigns a value 8 to `num[1][2]`
- d) assigns a value 9 to `num[1][2]`

4.3 Statement in `num[2][3] = {{1, 2}, {3, 4}, {5, 6}};`

- a) Assigns a value 2 to `num[1][2]`
- b) Assigns a value 4 to `num[1][2]`
- c) Assigns a value 3 to `num[1][2]`
- d) gives an error message.

Answer to objective questions

Q.	.1	.2	.3	.4	.5
2	Marco Subsitution	getch()	putch()	'\x0'	matix
3	T	F	F	F	F
4	b	a	d		

DESKTOP PUBLISHING

At the end of this chapter, we will learn about these topics :

1. What is Desktop Publishing (DTP) ?
2. Printing Methods.
3. Fonts.
4. Scaling, Tracking and Leading.
5. Frames, Page layout.
6. WYSIWYG
7. Advantage of DTP over word Processing.
8. Document learning.

9.0 Introduction to DTP (Desktop Publishing) :

An application of computer that enables small companies and individuals to produce reports, Visiting cards, Calendar, Advertising, Magazines etc to near typeset quality. Modern Systems, which simulate many of the professional typesetting functions, consist of a personal computer using DTP software. The software is designed to format text into pages using a wide range of different typefaces. A common feature is the ability to preview each page on the computer's screen before it is printed. The term Desktop Publishing is commonly used to describe all the page layout skills.

Desktop Publishing begins in 1985 with introduction of Mac Publisher. Mac Publisher is the first WYSIWYG (What you see is what you get) layout program. The point to be noted, before the invention of Desktop Publishing software (Corel Draw, Page Maker, Photoshop, MS-Word etc.), the task involved in desktop publishing were done by many peoples (Manually). Different types of peoples were involved in both graphic and text design. Now Desktop Publishing software becomes more and more powerful and gives you more control over text and graphics. Desktop Publishing is generally & three types (i) Page Layout (ii) Editing (III) Illustration.

- (i) **Page Layout**– Adode Page Maker etc.
- (II) **Editing** – Adode Photoshop, Corel Photo Paint etc.
- (III) **Illustration** – Coral Draw M.S. Publisher etc.

To prepare a document our first step is to decide what type of document is going to be prepared. Do you want to create a wedding card, front cover of the book, Brochure or business card. If you need to create a front cover of the book, the document size should be known. If you know about the all previous steps, you can make your document using any desktop publishing software.

9.1 Print Documents

It is the ver essential part of DTP. We can print the document in many ways and many forms. Printing is the process of reproducing text, images as well as graphic, typically with ink on paper in a printing press, can be through laser printer or inkjet printer. It may be large scale industrial process or may be a individual process.

9.2 Printing Methods :

Offset Printing (Lithography) : Ink sit on the surface of the paper, Today all modern printing is offset. Most short-run-jobs are now being done digitally. offset printing is better and cheaper for business use as compared for personal use..

Laser Printing : A Laser Printer is a common type of printer that produce high quality text and graphics Laser Printer uses non-impact photo copier technology. For printing a document a laser beam draws the document on selenium coated drum using electrical charges. After the drum charges, it is rolled. Colour Laser printer is up to ten times mor expensive than a Black/White Printer. The fastest Black/White model can print over 200 pages per minute and the fastest colour laser printer can print over 100 pages per minute. The standard resolution of laser printer is 600 dots per inches (dpi). It also depends upon graphics to be printed.

9.3 Fonts

A font is the combination of typeface and other qualities, such as size, point, width. For example Arial is a typeface that defines the shape of each character.

In simple words, "A font is the combination of all characters (including all the special characters i.e. comma(,), semicolon(;), hyphen(-), @ etc)". Font family is the collection of all the variations, for example Arial, Arial Bold, Arial Narrow are all the part of font families. The height of the font is measured in points. Low point means small characters, high point means large characters. For example :

View of Content	Font Size
School	8
School	10
School	12
School	14
School	16

9.4 Scaling, Tracking and Leading :

Adjusting font in document : There are a number of ways for adjusting text in a document. This term refers to that, without changing the points of any font we can increase its width or can decrease its width. The term known as scaling.

Tracking : Tracking simple means to increase or decrease the character space in a word or multiple words. It can also measure in points.

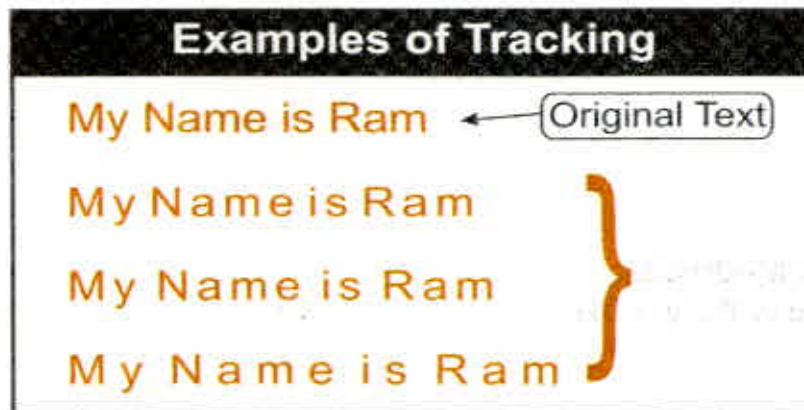


Fig. 9.1 Tracking

Leading : It simply shows the line space between two or more lines. It can also measure in points or depends upon the desktop publishing software. For example.

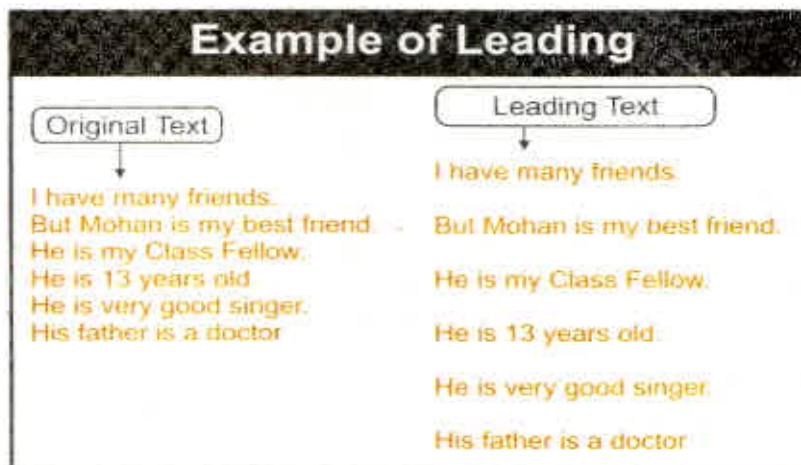


Fig. 9.2 Leading

9.5 Frames

Frames group related information or graphics for example.

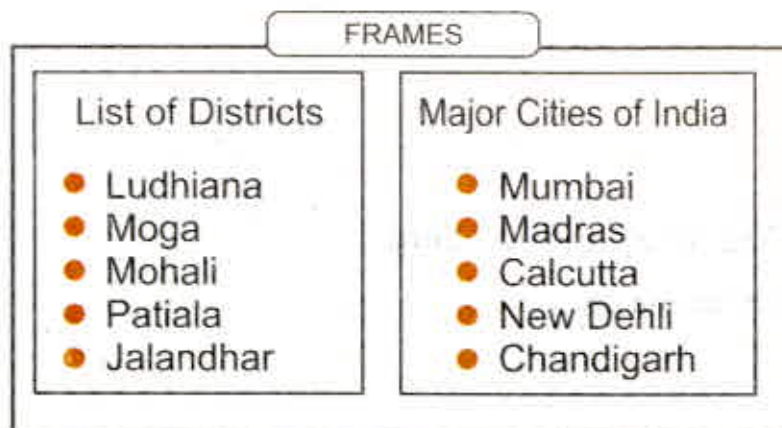


Fig. 9.3 Frames

In each frame, each object is individual entity. To use every frame, you must know the purpose of every frame, box or border. Frames form a barrier. Frames are used to draw attention to key points or useful tips. It indicates, the information is more important than others.

9.6 Page Layouts

Page layout is the task of placing and arranging all objects (i.e. text or graphics) on the page to produce meaningful documents such as books, brochures or posters. A page or document layout can be designed on a rough paper with pencil, pen or sketch. But the electronic page is better known as graphical user interface (GUI), Page layout refers, to just like the actual page and its composition. Page layout also includes occasionally place holders (i.e. grids, Lines, Boxes) that are not printed with ink.

9.7 WYSIWYG - (What-You-See-Is-What-You-Get):

This word used in computing is to describe a system, in which electronic page is displayed in very similar manner to the final output. That page can be a printed document, web page and can be a slide. Now Various Modern Desktop Publishing Software are doing a great job of optimizing the screen display for a certain type of output. For example MS-word is optimized for output to a typical colored laser printer, if we print that document then the document will be as close as to WYSIWYG.

In some situation, WYSIWYG is unimportant because:

- Additional Information is useful in composition made i.e. non-printing characters.
- In layout made we often use grids or guide lines for aligning objects and for margin lines in a document.

9.8 Advantages of DTP over Word Processor

From one point of view "Word Processor" is little more than a fancy electric typewriter, A simple "text editor" has three basic features (spell characters, formatting of blocks of text into heading, paragraphs etc) in "Word Processor" primary jobs are typing article, change the contents as necessary, spell checkers and simple margins and some page layout features. If you need more powerful layout control (e.g. to prepare brochures, Newsletter, Magazines, or a business cards), then you need Desktop Publishing type of software, instead of word processor. In simple means word processing is a strictly text based editor and DTP is object based software. In Desktop Publisher Softwares, documents are easier to layout and the text need to be more than on one place. In a DTP Software ever item or object also occurred in a flexible manner.

9.9 Document Planning

Let us discuss some tips, where desktop publishing is used. We also discuss some general guidelines for designing common documents.

Page Layout : After you have assembled your written text, graphics or special type of symbols (box frames and borders) then you are ready to layout your page. Layout is the process by which all the elements are laid on the page orderly. Subject matter should be highlighted.

Style : In Desktop Publishing software, you will be able to choose the best of several types of fonts, Graphics design style, its color and some type of filters and effects may also be applied to layout elements. Whatever font you decide, use this font in all the documents. Matter should be impressed in a stylish way it may be digest, understand and memorable.

Applying a style in a document is a way to automate the formatting. Without using a style you would have to highlight each paragraph, line individually. it become very tedious job. A styling includes heading style, header and footer style, bulle style etc.

Margin : Margins are the distance between the text from left and right edges, top and bottom edges.

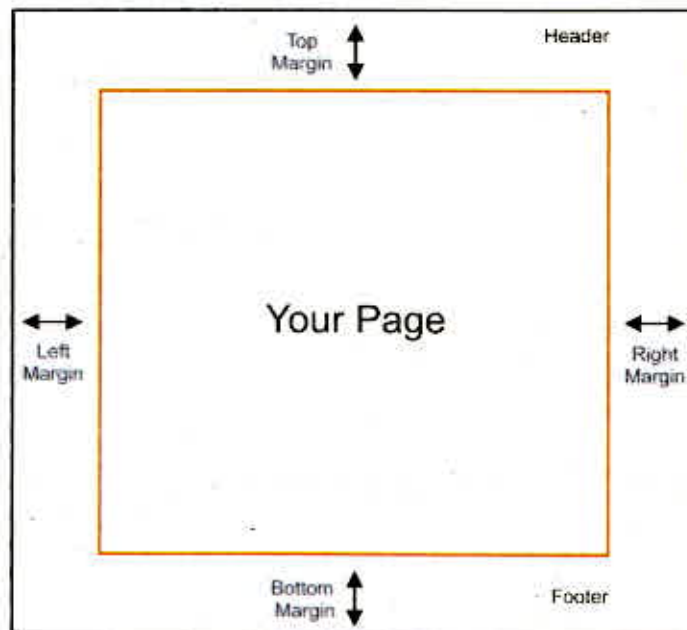


Fig. 9.4 Margin

Margins are normally 1 inch, but you can adjust it according to your need.

Header : A header is the text which is printed at the top of each page. It may be page no., chapter name or may be special information.

Footer : Footer is a text which is printed at the bottom of each page. It may also be page no. chapter name or may be special information. Almost every DTP software can automatically generate page no. and other information on every page on a document.

Font : It is the most important part of the document design, Generally readers like Arial, Times New Roman font because they are easy to read and more format in most of the text books, magazines and news papers are using this type of font.

These are the basic needs of document planning, but the other points are page size, number of pages, columns, title and sub title, captions etc.

POINTS TO REMEMBER

1. DTP softwares are used to create Reports, Visiting Cards, Calendar, Advertisement, Magazines etc.
2. To prepare a document, our first step is to decide what type of document is going to be prepared.
3. Printing is the process of reproducing text, image as well as graphics.
4. Laser Printing and offset printing are the two most common printing methods.
5. Font is the combination of typeface and other qualities, such as size, point and width.
6. Leading means line space between two or more lines.
7. Page layout is the task of placing and arranging all objects i.e. text, graphics or shapes.
8. WYSIWYG means What You See Is What You Get.
9. If you need more powerful layout control (e.g. for making brochures news letter, magazines or a business card) then you need desktop publishing software instead of word processor.
10. Margins are the distance between the text from left and right edges, top and bottom edges.

EXERCISE

Q.1. Short Answer Type Question :

1. What do you mean by Desktop Publishing ?
2. What are the various methods of printing ?
3. What do you mean by Scaling, Tracking & Leading ?
4. What do you mean by WYSIWIG ?
5. What are Margins ?

Q.2. Long Answer Type Questions

1. What do you mean by Page Layout ?
2. What are fonts ?
3. Write down the difference between DTP and Word Processor.
4. What do you mean by Document Planning ?
5. What do you mean by DTP ? Why we need to use DTP ? What are the various types of DTP software ?

Q.3. True/False

1. Header is written on the bottom of the page.
2. MS-Word is a DTP software.
3. We don't need planning for preparing a document.
4. Style includes bullet effects.
5. Spacing between two or more lines are called scaling.

Q.4. Fill in the Blanks

1. _____ means spacing between two or more lines.
2. Frame _____ related information or graphics.
3. WYSIW/G means _____ .
4. We can _____ document in many ways.
5. Fastest color laser printer can print _____ pages.

Common Programmings Errors

Let us examine some of the common errors that a programming learner/beginner can commit while do programming.

Case 1 Non-terminated comment

Consider the following statements with the comments given

```
a=b;           /* this is a bug
c=d;           /* c=d will never happen */
```

As the closing comment `*/` is missing in the first statement, all the statements that follows, until the closing comment `*/`, are ignored by the compiler i.e. the compiler will search for closing comment (`*/`) further down in the program and treat all the lines as comment, but if the same does not found, we may get an error message.

Case 2 a) Misuse of semicolon

Consider the following statements in which we try to sum the integer values from 1 to 5, if we write the statment

```
for (a=1;a <=5;a++);
sum=sum + a;
```

Only the exit value of the a=5 is added to sum.

b) Missing Semicolons

Every C statement must end with a semicolon. A missing semicolon may cause considerable confusion to the compiler and result in 'misleading' error messages. For example if we consider the following two statements.

```
c = a + b
d = a / b;
```

Here the compiler will treat the `d = a / b;` statement as a part of the `c = a + b` and takes `bd` as a variable name. Due to this "**undefined name**" error message will be displayed in the second line. Clearly the error was in the first line and message displayed is on the second line where there is no error, so in these situations we should check the preceding line for a missing of semicolon.

Case 3 Missing Braces / putting braces at proper place

Some times we forget to put braces specially closing braces while ending the program. It will be usually detected by the compiler because the number the opening braces should be equal to the closing beaces. However if we put the matching braces at a wrong place, this may not be detected by the compiler but the program may produce unexpected results. To see this let us consider the following example

Example Statements without proper braces

```
for(a=1;a <=5;a++)
```

```

sum1 = sum1 + a;
avg = sum1/5 ;
printf( "%d %d\n", sum1, avg);

```

This program code is to compute sum 1 & average for a varying from 1 to 5 (in steps of 1) & then to print their values, But, actually the **for loop** treats only the first statement namely **sum1 = sum1 + a:** as its body and therefore the statement **avg = sum1/5 ;** is evaluated only once when the loop is exited.

Example Statements with proper braces

```

for(a = 1; a <= 5; a++)
{
    sum1 = sum1 + a;
    avg = sum1/5 ;
}
printf( "%d %d\n", sum1, avg);

```

The proper braces are shown in above

Case 4 Missing Quotes

Every string must be enclosed in double quotes, while a single character constant in single quotes. On missing them, the string will be interpreted as a variable name

For example `if (married ==; No")` `/* No is a string */`
 `status = "N"` `/* N is a character constant */`

Here No & N are treated as variables & therefore, message **"undefined name"** may occur

There should be 'N' instead of "N" in above example

Case 5 Accidental assignment/Accidental Booleans

Consider the following case statements

a) `if (a=b) c;` `/* a always equals b, but c will be executed if b!=0*/`

Depending on your view point, the bug in the language is that the assignment operator is too easy to confuse with the equality operator; or may be the bug is that C doesn't much care what constitutes a boolean expression: **(a=b)** is not a boolean expression!

Closely related to this lack of rigor in booleans, consider this construction:

`if(0 < a < 5) c;` `/* this "boolean" is always true! */`

Always true because $(0 < a)$ generates either 0 or 1 depending on if $(0 < a)$, then compares the result to 5, which is always true, or consider this:

`if(a != b) c;` `/* this is compiled as (a = !b), an assignment, rather than (a != b) or (a == !b) */`

b) Use of = instead of ==

For example `if (a= 1)`
 `count++;`

Here the variable **a** is assigned 1 and then, because **a = 1** is true, count is incremented but this statement does not perform any test on **a** irrespective of the

previous values of **a**, **count ++**; is always executed. Same errors can occurs in other control statements like **for** and **while**.

Case 6 Undeclared Variables

We must declare the variables and its type, before we used it. Otherwise a message "**undefined variable**" will be displayed on screen.

Case 7 Forgetting the precedence of operators

Experssion should be evaluated according to the precedence of operators. Consider the statement

If (value = product () >= 5)

Fee = 0.05 * value;

The call **product** returns the product of two numbers, which is compared to 5. If it equal to or greater that 5, the relational test is true, and a 1 is assigned to **value** ; otherwise a 0 is assigned. In either case, the only values **value** can take on are 1 or 0. This certainly is not the programmer wanted.

The statement was actually expected to assign the value returned by the **product ()** to **value** and then compare **value** with 5. If **value** was equal to or greater than 5 , Free should have been computed, by statement **Fee = 0.05 * value;**

This error is due to the higher precedence of the relational operator compared to the assignment operator. So the correction is

If (value = product ()) >= 5)

Fee = 0.05 * value;

Case 8 Forgetting to declare Function

Every function that is called should be declared in the calling function for the **type** of the value it returns. For example the function below given returns a **double type** value but this fact is not known to the calling function and therefore it accepts to receive an **int** type value. So whenever this is used in a program, an error message such as "redefinition" or meaning less result is obtained.

```
main( )
{ float a = 12.05 ;
  float b =7.35 ;
  printf("%f\n " ,division(a,b));
}
double division(x,y)
float x, y;
```

In this example the x and y are float type and the result of x/y is

```
{
return (x,y);
}
```

The function division is like other variable for the main () and therefore it should be declared as double in the main () as

```
division (x,y)
float x, y;
{ return(x/y);
}
```

also float, the function returns only integer value because no type specifier is given in the function definition. This is wrong too. The function header should include the type specifier to force the function to return a particular type of value.

Case 9 Missing & operator in scanf parameter

All non-pointer variables in a scanf call should be preceded by an & operator, if the variable code is declared as an integer, then the statement **scanf("%d", a);** is wrong the correct one is **scanf("%d", &a);**

Case 10 Crossing the Bounds of an array:

The first subscript or index in the arrays is 0. A common mistake is to start the subscript from 1.

for Example the segment

```
int x[5],sum,a;
sum = 0;
for(a = 1;a <=5;a++)
sum = sum +x[a];
```

Would not find the correct sum of the element of array x. The for loop aexpression should be corrected as

```
for(a = 0;a <5;a++)
```

Glossary

A	
Analog Data	Data that can have any value in a range and that can change continuously; the time of day represented by clock hands, or the temperature represented by a liquid thermometer are examples of analog data.
Analog Signal	A type of signal that encodes data transmitted over wire or through the air, and is commonly represented as an oscillating wave. An analog signal can take any value in a range, and changes smoothly between values.
	An analog signal can transmit analog or digital data. For example, a radio station sends analog music data using analog signals, while a modern transmits digital data using analog signals.
Argument	Calling a function, refers to the actual values passed to the function.
Argument Matching	The process of determining which of a set of functions of a specified name matches given arguments in a function call.
Array	An ordered and indexable sequence of values. C++ supports arrays of a single dimension or of multiple dimensions.
ASCII	A numeric code standard for characters. Literally, ASCII stands for American Standard Code for Information Interchange. This is the most common character to integer translation code.
Assignment	The process of giving a value to a pre-existing variable /constant.
Assignment Operator	An operator for doing assignment.
B	
Backbone Network	A network that works as a backbone to connect several LANs.
Bandwidth	The capacity of a medium to transmit a signal.
Boolean	C keyword used to declare a Boolean data type.
Break	C keyword used to specify a statement that is used to break out of a for or while loop or out of a switch statement.
Bridge	Device that links two networks together.
C	
C Library	A system library that contains common C language subroutines for file access, string operators, character operations, memory allocation, and other functions.
Cable	Transmission medium of copper wire or optical fiber wrapped in a protective cover.
Call	To transfer control to a procedure, program, routine, or subroutine.
Call By Reference	Passing a copy of an argument to a function. The function can then change the argument value. C uses call by value argument passing. But also call by reference.
Caller	A routine that calls another routine
Carriage-Return Character	A character that in the output stream indicates that printing should start at the beginning of the same physical line in which the carriage-return character occurred. The carriage-return is the character designated by '\r' in the C and C++ languages. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the beginning of the line.

Case Clause	In switch statement, a case label followed by any number of statements.
Case Label	The word case followed by a constant expression and a colon. When the selector evaluates the value of the constant expression, the statements following the label are processed.
CD-Rom	High-capacity, read-only memory in the form of an optically read compact disc.
Character	(1) A letter, digit, or other symbol that is used to represent data. (2) A sequence of one or more bytes representing a single graphic symbol or control code.
Character Constant	A character or an escape sequence enclosed in apostrophes.
Character Set	1) A finite set of different characters that are complete for a given purpose; for example, the character set in ISO Standard 646, 7-bit Coded Character Set for Information Processing Interchange. T. (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print.
Character String	A contiguous sequence of characters terminated by and including the first null byte.
Character Variable	The name of a character data item whose value is assigned or changed during program execution.
Class	1) A group of objects that share a common definition and that therefore share common properties, operations, and behavior. (2) A C++ aggregate that may contain functions, types, and user-defined operations in addition to data. Classes can be defined hierarchically, allowing one class to be an expansion of another, and classes can restrict access to their members.
Client/Server	A networking system in which one or more file servers (Server) provide services; such as network management, application and centralized data storage for workstations (Clients).
Coaxial Cable	Cable consisting of a single copper conductor in the center surrounded by a plastic layer for insulation and a braided metal outer shield.
Code Page	1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, or assignment of characters and meanings to 128 code points for a 7-bit code. (2) A particular assignment of hexadecimal identifiers to graphic characters.
Code Point	1) A 1-byte code representing 1 of 256 possible characters. (2) An identifier in an alert description that represents a short unit of text. The code point is replaced with the text by an alert display program.
Coded Character Set	1) A set of graphic characters and their code point assignments. The set may contain fewer characters than the total numbers of possible characters; some code points may be unassigned. (2) A coded set whose elements are single characters; for example, all characters of an alphabet.
Comma Expression	An expression that contains two operands separated by a comma. Although the compiler evaluates both operands, the value of the expression is the value of the right operand. If the left operand produces a value, the compiler discards this value. Typically, the left operand of a comma expression is used to produce side-effects.

Command	An instruction to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.
Command Line Interface	A type of computer interface in which the input command is a string of text. Contrast with graphical user interface (GUI).
Compilation Unit	<p>(1) A portion of a computer program sufficiently complete to be compiled correctly.</p> <p>(2) A single compiled file and all its associated include files.</p> <p>(3) An independently compilable sequence of high-level language statements. Each high-level language product has its own rules for what make up a compilation unit.</p>
Compile	To translate from source code into an object form.
Compiler Options	Keywords that can be specified to control certain aspects of compilation.
Composite	The combination of two or more film, video, or electronic images into a single frame or display.
Concentrator	A device that provides a central connection point for cables from workstations, servers, and peripherals. Most concentrators contain the ability to amplify the electrical signal they receive.
Condition	<p>(1) A relational expression that can be evaluated to a value of either true or false.</p> <p>(2) An exception that has been enabled, or recognized, by the language environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware or operating system and result in an interrupt. They can also be detected by language-specific generated code or language library code.</p>
Conditional Compilation Directive	A preprocessor directive that causes the preprocessor to preprocess specified source code in the file depending on the evaluation of a specific condition.
Conditional Expression	A compound expression that contains a condition (the first expression), an expression to be evaluated if the condition has a nonzero value (the second expression), and an expression to be evaluated if the condition has the value zero (the third expression).
Const	<p>(1) An attribute of a data object that declares that the object cannot be changed.</p> <p>(2) An attribute of function that declares that the function will not modify data members of its class.</p>
Constant	<p>(1) In programming languages, a language object that takes only one specific value.</p> <p>(2) A data item with a value that does not change during the running of the program.</p>
Constant Expression	An expression having a value that can be determined during compilation and that does not change during the running of the program.
Control Character	<p>(1) A character whose occurrence in a particular context specifies a control function <i>T</i>.</p> <p>(2) A character, other than a graphic character, that effects the recording, processing, transmission, or interpretation of text.</p>

Control Statement	A language statement that changes the normal path of execution.
Conversion	<p>(1) In programming languages, the transformations between values that represent the same data item but belong to different data types. Information may be lost because of conversion since accuracy of data representation varies among different data types.</p> <p>(2) The process of changing from one method of data processing to another or from one data processing system to another. The process of changing from one form of representation to another; for example, changing from decimal representation to binary representation.</p> <p>(3) A change in the type of a value; for example, when you add values having different data types, the compiler converts both values to a common form before adding the values.</p>
Current Working Directory	<p>1) A directory, associated with a process that is used in path-name resolution for path names that do not begin with a slash.</p> <p>(2) in DOS, the directory that is searched when a file name is entered with no indication of the directory that lists the file name. DOS assumes that the current directory is the root directory unless a path to another directory is specified.</p>
Cursor	A reference to an element at a specific position in a data structure.
Cursor	The process of repeatedly moving the cursor to the next element in a collection until some condition is satisfied.
D	
Declaration	A C entity that introduces one or more names into a program.
Declaration	A declaration in the form of a statement that may be used in C where statements would normally be used.
Default Argument	An optional argument to a function. A value specified in the function declaration is used if the argument is not given.
Digital Data	Data that can have only a limited number of separate values. The time of day represented by a digital clock, or the temperature represented by a digital thermometer are examples of digital data; the digital values do not change continuously, but remain at one discrete value and then change to another, discrete value.
Dumb Terminal	Refers to devices that are designed to communicate exclusively with a host (main frame) computer. It receives all screen layouts from the host computer and sends all keyboard entry to the host. It cannot function without the host computer.
E	
Else	C keyword, part of the if statement.
Enum	C keyword used to declare an enumeration.
Expansion Slot	Area in a computer that accepts additional input/output boards to increase the capability of the computer.
Expression	A combination of constants, variables, and operators used to produce a value of some type.
Expression Statement	A statement that is an expression, such as a function call or assignment.
Extern	A C keyword used to declare an external name.

F	
False	C keyword used to specify a value for the boolean type.
Fiber Optic Cable	A cable, consisting of a center glass core surrounded by layers of plastic, that transmits data using light rather than electricity. It has the ability to carry more information over much longer distances.
File Server	A computer connected to the network that contains primary files/applications and shares them as requested with the other computers on the network. If the file server is dedicated for that purpose only, it is connected to a client/server network.
Firewall	A system to prevent unauthorized access to /from a network.
Function	A C entity that is a sequence of statements. It has its own scope, accepts a set of argument values, and returns a value on completion.
G	
Gigabyte (GB)	One billion bytes of information. One thousand megabytes.
Global Variable	A variable that is accessible throughout the whole program, whose lifetime is that of the program.
Goto	C keyword, used to transfer control within a C function.
H	
Header File	A file containing class declarations, preprocessor directives, and so on, and included in a translation unit. It is expanded by the preprocessor.
Host	Machines on a network that run user programs.
Hub	A hardware device that contains multiple independent but connected modules of network and internet work equipment. Hubs can be active (where they repeat signals sent through them) or passive (where they do not repeat but merely split signals sent through them).
I	
Infrared	Electromagnetic waves whose frequency range is above that of microwaves, but below that of the visible spectrum.
Initialization	To give an initial value to a variable or constant.
Initialize	The process of initialization.
Initializer	A value of expression used to initialize an object during initialization.
Int	A C keyword and fundamental type, used to declare an integral type.
Integral Conversion -	The process by which an integer is converted to signed or unsigned.
Internet	A global network of networks used to exchange information using the TCP/IP protocol. It allows for electronic mail and the accessing and retrieval of information from remote sources.
Intranet	Network internal to an organization that uses Internet protocols.
K	
Keyword	A reserved identifier in C++, used to denote data types, statements of the language, and so on.

L	
Label	A name that is the target of a goto statement.
LAN	A network connecting computers in a relatively small area such as a building.
Library	A set of files grouped together. A linker will search them repeatedly and use whatever object files are needed.
Literal	A constant like 1234
Local Variable	A variable declared local to a function.
Long	C keyword used to declare a long integer data type
Long Double	A floating point type in C
M	
Macro	A preprocessor feature that supports parameter substitution and expansion of commonly-used code sequences.
MAN	(Metropolitan Area Network) - A network connecting computers over a large geographical area, such as a city or school district.
Modem	Modulator/Demodulator) - Devices that convert digital and analog signals. Modems allow computer data (digital) to be transmitted over voice-grade telephone lines (analog).
Multiplexer	A device that allows multiple logical signals to be transmitted simultaneously across a single physical channel.
N	
Name Space	A grouping of names.
Network Interface Card	(NIC) a board that provides network communication capabilities to and from a computer.
Network Modem	A modem connected to a Local Area Network (LAN) that is accessible from any workstation on the network.
Network Operating System	NOS) - Operating system designed to pass information and communicate between more than one computer. Examples include Windows NT Server.
Node	End point of a network connection. Nodes include any device attached to a network such as file servers, printers, or workstations.
Node device	Any computer or peripheral that is connected to the network.
Null	A special constant value.
O	
Object	It has several meanings. In C++, often refers to an instance of a class. Also more loosely refers to any named declaration of a variable or other entity that involves storage.
P	
Parameter	It allows calling code to pass a value to the procedure when it calls it.
Peer-To-Peer Network	A network in which resources and files are shared without a centralized management source.
Physical Topology	The physical layout of the network; how the cables are arranged; and how the computers are connected.
Pointer	The variable that stores the reference to another variable.

Point-To-Point	A direct link between two objects in a network.
Ports	A connections point for a cable.
Programming Environment	A set of integrated tools used in developing software, includin a compiler, linker, debugger, and browser.
Protocol	A formal description of a set of rules and conventions that govern how devices on a network exchange information.
R	
Reference	Another name for an object. Access to an object via a reference is like manipulating the object itself. References are typically implemented as pointers in the underlying generated code.
Register	C keyword used as a hint to the compiler that a particular local variable should be placed in a machine register.
Return	C keyword used for returning values from a function.
Return Value	The value returned from a function.
S	
Segment	Refers to a section of cable on a network. In Ethernet networks, two types of segments are defined. A populated or trunk segment is a network cable that has one or more nodes attached to it. A link segment is a cable that connects a computer to an interconnecting device, such as a repeater or concentrator, or connects an interconnecting device to another interconnecting device.
Short	A C fundamental type used to declare small integers.
Signed	C keyword used to indicate a signed data type.
Sizeof	C keyword for taking the size of an object or type.
Speed of Data Transfer	The rate at which information travels through a network, usually measured in megabits per second.
Star-Wired Ring	Network topology that connects network devices (such as computers and printers) in a complete circle.
Statement	The parts of a program that actually do the work.
Static	See static member, static object, and static storage.
Static Member	A class member that is part of a class for purposes of access control but does not operate on particular object instances of the class.
Static Object	An object that is local to a function or to a translation unit and whose lifetime is the life of the program.
Storage Class	See auto and static.
Switch	C keyword denoting a statement type, used to dispatch to one of several sequences of statements based on the value of an expression.
Syntax	The rules that govern how C expressions, statements, declarations, and programs are constructed.
T	
Tag	A name given to a class, struct, or union
Terminator	A device that provides electrical resistance at the end of a transmission line. Its function is to absorb signals on the line, thereby keeping them from bouncing back and being received again by the network.

Token	A special packet that contains data and acts as a messenger or carrier between each computer and device on a ring topology. Each computer must wait for the messenger to stop at its node before it can send data over the network.
Token Ring	A network protocol developed by IBM in which computers access the network through token-passing. Usually uses a star-wired ring topology.
Topology	There are two types of topology; physical and logical. The physical topology of a network refers to the configuration of cables, computers and other peripherals. Logical topology is the method used to pass that information between workstations.
Transceiver	(Transmitter/Receiver) - A Device that receives and sends signals over a medium. In networks, it is generally used to allow for the connection between two different types of cable connectors, such as AUI & RJ-45.
True	C keyword used to specify a value for the bool type.
Twisted pair	Network cabling that consists of four pairs of wires that are manufactured with the wires twisted to certain specifications. Available in shielded and unshielded versions.
U	
Unsigned	A C keyword used to declare an integral unsigned fundamental type.
USB	(Universal Serial Bus) Port - A hardware interface for low-speed peripherals such as the keyboard, mouse, joystick, scanner, printer, and telephony devices.
W	
WAN	(Wide Area Network) - A network connecting computers within very large areas, such as states, countries, and the world.
While	A loop in C language, used to execute a set of statements repeatedly as long as the specified condition is true.
Workgroup	A collection of workstations and servers on a LAN that are designated to communicate and exchange data with one another.
Workstation	A computer connected to a network at which users interact with software stored on the network.