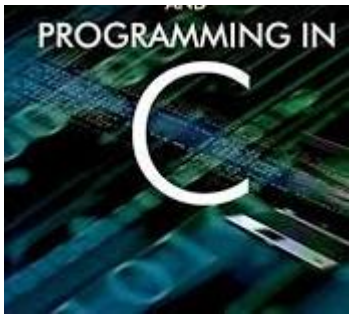
DONBOSCO COLLEGE

Kottiyam,Kollam,Kerala

(Affiliated to the University of Kerala)



CP1141: INTRODUCTION TO PROGRAMMING

S1 BCA

Name : ……

Reg No: ……

SYLLABUS

Module I: Introduction to programming: Algorithm & Flow charts: Definitions, Symbols used to draw flowcharts, Program Writing – Structure of the Program, top down design, Source code, Object code, Executable file, Variables and Constants, Rules for naming the Variables/Identifiers; Basic data types of C, int, char, float, double; storage capacity – range of all the data types;

Module II: Basic Elements: Operators and Expressions: Expression Evaluation (Precedence of Operators); simple I/O statements, Control structures, if, if else, switch-case, for, while, do-while, break, continue. Arrays: Defining simple arrays, Multi-dimensional arrays, declaration, initialization and processing.

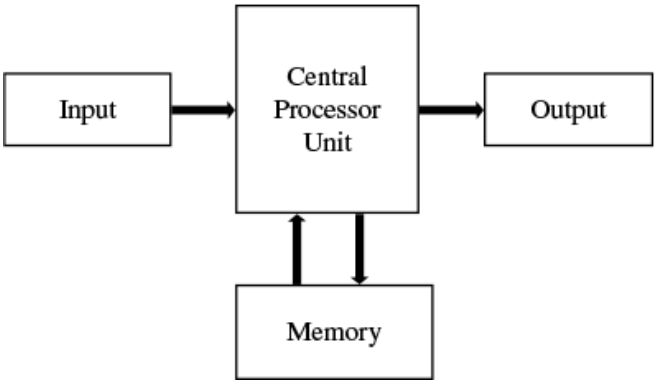
Module III: Functions & Pointers: concept of modular programming, Library, User defined functions, declaration, definition & scope, recursion, Pointers: The & and * Operators, pointer declaration, assignment and arithmetic, visualizing pointers, call by value; call by reference, dynamic memory allocation. Storage classes.

Module IV: Advanced features: Array & pointer relationship, pointer to arrays, array of pointers. Strings: String handling functions; Structures and unions; File handling: text and binary files, file operations, Functions for file handling, Modes of files

Module I

Introduction to programming: Algorithm & Flow charts: Definitions, Symbols used to draw flowcharts, Program Writing – Structure of the Program, top down design, Source code, Object code, Executable file, Variables and Constants, Rules for naming the Variables/Identifiers; Basic data types of C, int, char, float, double; storage capacity – range of all the data types;

A computer is a programmable electronic machine that accepts instructions and data through input devices, manipulating data according to instructions and finally providing result to the output device. The result can be stored in memory or sent to the output device. The conventional block diagram of a conventional computer.



Input Device: Input device is used to accept the data and instructions in to the computer. Through the input device, *i.e. keyboard*, instructions and data are stored in a computer's memory. Stored instructions are further read by the computer from its memory and thus a computer manipulates the data according to the instructions. The various input devices that can be used in a computer are keyboards, mouse, light pen, track ball, optical character reader and scanner.

Central Processor Unit: CPU is the abbreviation for central processor unit. It is the heart of the computer system. It is a hardware device in the computer and quite often it called as microprocessor chip. Since it is a tiny chip hence called as microprocessor chip. This chip is produced from silicon vapor over which millions of transistors are mounted with modern fabrication techniques.

The brain of the computer is CPU. This chip is responsible to interpret and execute the instructions. It comprises arithmetic and logical unit, registers and control unit. The arithmetic and logical unit performs various arithmetic and logical operations on the data based upon the instructions. The control unit generates the timing and control signals for carrying out operations within the processor. Registers are used for holding the instructions and storing the results temporarily. Instructions are stored in the memory and they are fetched one by one and executed by the processor.

Output Device: The output device is used to display the results on the screen or to send the data to an output device. The processed data is ultimately sent to the output device by the computer. The output device can be a monitor, a printer, and so on.

Memory: Memory is used to store the program. There are two types of semi-conductor memories. They are as follows:

i. RAM (Random access memory)
ii. ROM (Read only memory)

semi-conductor memory is used for storing the instructions and data in the form of ones and zeros. The memory can be called user's memory or read–write memory. Processor first reads the instructions and data from the primary memory (semi-conductor memory). Then, it executes the instructions.

One more memory device used by a computer is called read only memory (ROM). This contains a fixed software program for providing certain operations. This is non-volatile memory. Its contents cannot be eliminated when power supply goes off. The basic input–output system (BIOS) is a software used to control various peripheral devices such as a keyboard, a monitor, a printer, a mouse, ports including serial and parallel ports. In fact, this is an operating system. It is possible to access the users' written programs, i.e. by loading a file, saving it and doing modifications to it in the later stage. As soon as a personal computer is switched on, the software gets booted from ROM. Thus, various functions are assigned to all supporting peripherals of a central processing unit (CPU) and easy interactions are provided to the user by BIOS while booting the system.

INTRODUCTION TO C

C is one of the most popular general-purpose programming languages. C language has been designed and developed by Dennis Ritchie at Bell Laboratories, USA, in 1972. Several important concepts of C are drawn from 'Basic combined programming language' and 'B' language. Martin Richards developed BCPL in 1967. The impact of BCPL on C is observed indirectly through the language B, which was developed by Ken Thompson in 1970. C is also called an offspring of the BCPL.

Sr. No.	*Language*	*Inventor*	*Year*
1.	BCPL	Martin Richards	1967
2.	B	Ken Thompson	1970
3.	C	Dennis Ritchie	1972

The C language is closely associated with the UNIX operating system. The source code for the UNIX operating system is in C. The C programs are efficient, fast and

highly portable, i.e. C programs written on one computer can be run on another with mere or almost no modification.

The C programming language contains modules called functions. The C functions are the basic building blocks and the most programmers take its benefit. Programmers include these functions in their program from the C standard library.

C language is a middle-level computer language. It combines the features of a high-level language and functionality like assembly languages. In C, one can develop a program fast and execute fast. It reduces the gap between high-and low-level languages; that is why it is known as a middle level language. It is well suited for writing both application and system softwares.

C is a structural language. Structured language facilitates the development of a variety of programs in small modules or blocks. Lengthy programs can be divided into shorter programs. The user requires thinking of a problem in terms of functional blocks. With appropriate collection of different modules, the programmer can make a complete program.

It is easy for writing, testing, debugging and maintenance with structured programming. Programming with non-structured languages is tough in comparison to structured languages.

C is also called a system-programming language because it is greatly helpful for writing operating systems, interpreters, editors, compilers, database programs and network drivers.

C also provides control-flow statements such as decision-making statements (if-else) and (switch-case) multi-choice statement. C supports for, while and do-while looping statements.

C is not a strongly typed language. But typed statements are checked thoroughly by C compilers. The compiler will issue errors and warning messages when syntax rules are violated. There is no automatic conversion of incompatible data types. A programmer has to perform explicit type conversion.

MACHINE, ASSEMBLY AND HIGH-LEVEL LANGUAGE

Machine languages

It is a computer's natural language, which can be directly understood by the system. This language is machine dependent, i.e. it is not portable. A program written in 1's and 0's is called a machine language. A binary code is used in a machine language for a specific operation. A set of instructions in binary pattern is associated with each computer. It is difficult to communicate with a computer in

terms of 1s and 0s. Hence, writing a program with a machine language is very difficult. Moreover, speed of writing, testing and debugging is very slow in machine language. Chances of making careless errors are bound to be there in this language. The machine language is defined by the hardware design of that hardware platform. Machine languages are tedious and time consuming.

Assembly Language

Instead of using a string of binary bits in a machine language, programmers started using English-like words as commands that can be easily interpreted by programmers. In other words, English-like words abbreviated as mnemonics that are similar to binary instructions in machine languages. The program is in alphanumeric symbols. The designer chooses easy symbols that are to be remembered by the programmer, so that the programmer can easily develop the program in assembly language. The alphanumeric symbols are called mnemonics in the assembly language. The ADD, SUB, MUL, DIV, RLC and RAL are some symbols called mnemonics.

The programs written in other than the machine language need to be converted to the machine language. Translators are needed for conversion from one language to another. Assemblers are used to convert assembly language program to machine language. Every processor has its own assembly language. For example, 8085 CPU has its own assembly language. CPUs such as 8086, 80186, 80286 have their own assembly languages.

Following disadvantages are observed with the assembly languages:

1. It is time consuming for an assembler to write and then test the program.
2. Assembly language programs are not portable.
3. It is necessary to remember the registers of CPU and mnemonic instructions by the programmer.
4. Several mnemonic instructions are needed to write in assembly language than a single line in high-level language. Thus, assembly language programs are longer than the high language programs.

High-Level Language

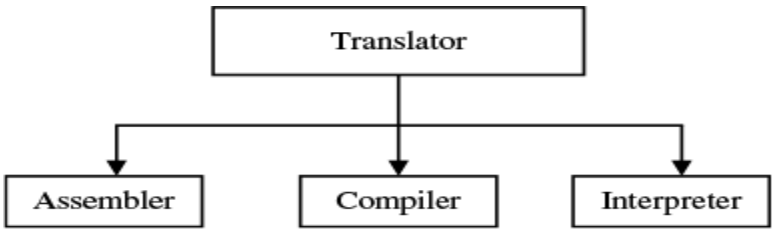
Procedure-oriented languages are high-level languages. These languages are employed for easy and speedy development of a program. The disadvantages observed with assembly languages are overcome by high-level languages. The programmer does not need to remember the architecture and registers of a CPU for developing a program. The compilers are used to translate high-level language program to machine language. Examples of HLL are COBOL, FORTRAN, BASIC, C and C++. The following advantages are observed with HLL languages:

1. Fast program development.
2. Testing and debugging a program is easier than in the assembly language.
3. Portability of a program from one machine to other.

ASSEMBLER, COMPILER AND INTERPRETER

A program is a set of instructions for performing a particular task. These instructions are just like English words. The computer interprets the instructions as 1's and 0's. A program can be written in assembly language as well as in high-level language. This written program is called the source program. The source program is to be converted to the machine language, which is called an object program. A translator is required for such a translation.

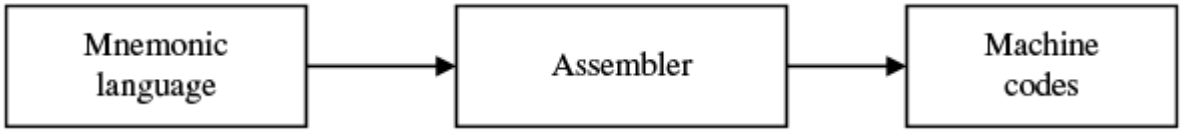
Program translator translates source code of programming language into machine language-instruction code. Generally, computer programs are written in languages like COBOL, C, BASIC and ASSEMBLY LANGUAGE, which should be translated into machine language before execution. Programming language translators are classified as follows.



Translators are as follows:

1. Assembler
2. Compiler
3. Interpreter

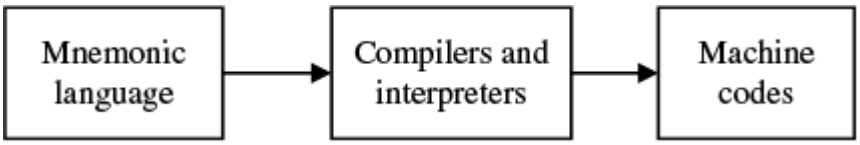
Assembler: An assembler translates the symbolic codes of programs of an assembly language into machine language instructions . The assembler programs translate the low-level language to the machine code. The translation job is performed either manually or with a program called assembler.



Assembler

Compiler: Compilers are the translators, which translate all the instructions of the program into machine codes, which can be used again and again . The program, which is to be translated, is called the source program and after translation the object code is generated. The source program is input to the compiler. The object code is output for the secondary storage device. The entire program will be read by

the compiler first and generates the object code. However, in interpreter each line is executed and object code is provided.

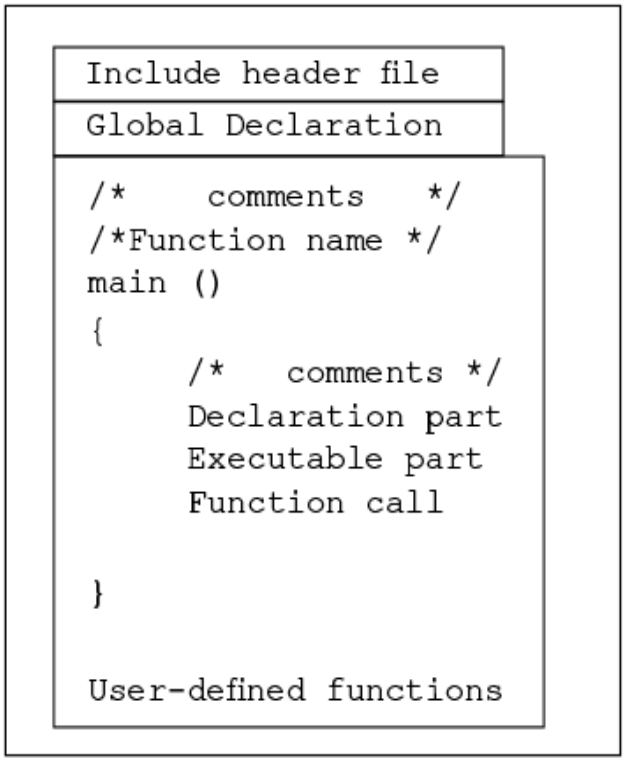


Interpreter: Interpreters also come in the group of translators. It helps the user to execute the source program with a few differences as compared to compilers. The source program is just like English statements in both interpreters and compilers. The interpreter generates object codes for the source program. Interpreter reads the program line by line, whereas in compiler the entire program is read by the compiler, which then generates the object codes. Interpreter directly executes the program from its source code. Due to this, every time the source code should be inputted to the interpreter. In other words, each line is converted into the object codes. It takes very less time for execution because no intermediate object code is generated.

Linking: C language provides a very large library, which contains numerous functions. In some applications of C the library may be a very large file. Linker is a program that combines source code and codes from the library. Linking is the process of bringing together source program and library code.

STRUCTURE OF A C PROGRAM

Every C program contains a number of building blocks known as functions. Each function of it performs a specific task independently.



Include Header File Section: C program depends upon some header files for function definition that are used in the program. Each header file has extension '.h'. The header files are included at the beginning of the program in the C language. These files should be included using # include directive as given below.

Example:

include <stdio.h> or

include "stdio.h"

1. In this example, <stdio.h> file is included, i.e. all the definitions and prototypes of function defined in this file are available in the current program. This file is also compiled with the original program.
2. **Global Declaration:**This section declares some variables that are used in more than one function. These variables are known as global variables. This section must be declared outside of all the functions.
3. **Function main():**Every program written in **C** must contain main() and its execution starts at the beginning of this function. In ASCII C standard, first line of C program from where program execution begins is written as follows.

int main(void)

This is the function definition for main(). Parenthesis followed to main is to tell the user again that main() is a function. The int main(void) is a function that takes no arguments and returns a value of type int. Here in this line, int and void are keywords and they have special meanings assigned by the compiler. In case int is not mentioned in the above statement, by default the function returns an integer. Alternately, one can also write the first line of C program from where program execution begins is as follows.

void main(void)

Here, this function takes no arguments and returns nothing. Alternately, one can also write the same function as follows.

void main(): This functions returns nothing and takes no arguments.

The programmer can either write the function main with int main(void) or void main(). Only in the formal case, return statement should be used before the end of function for terminating the execution of the main function.

The program contains statements that are enclosed within the braces. The opening brace ({) and closing brace (}) are used in C. Between these two braces, the program should declare declaration and executable part. The opening curly brace specifies the start of the definition of the main function. The closing curly brace specifies the end of the code for the main function.

4. **Declaration Part:**The declaration part declares the entire local variables that are used in executable part. Local variable scope is limited to that function where the

local variables are declared. The initializations of variables can also be done in this section. The initialization means providing initial value to the variables.

5. **Executable Part:**This part contains the statements following the declaration of the variables. This part contains a set of statements or a single statement.
6. **Function Call**:From the main() a user defined function can be invoked by the user as per need/application.
7. **User-defined Function:**The functions defined by the user are called user-defined functions. These functions are defined outside the main() function.
8. **Body of the Function:** The statements enclosed within the body of the function (between opening and closing brace) are called body of the function.
9. **Comments:**Comments are not necessary in a program. However, to understand the flow of programs a programmer can insert comments in the program. Comments are to be inserted by the programmer. It is useful for documentation. The clarity of the program can be followed if it is properly documented.

Comments are statements

Comments are statements that give us information about the program which are to be placed between the delimiters /* and */. The programmers in the programs for enhancing the lucidity frequently use comments. The compiler does not execute comments.

Example:

```
/* This is single comment */
/* This is an example of /* nested comments */*/
/* This is an example of
of comments in
multiple lines */ /* It can be nested */
```

PROGRAMMING RULES

A programmer while writing a program should follow the following rules:

1. Every program should have main() function.
2. C statements should be terminated by a semi-colon. At some places, a comma operator is permitted. If only a semi-colon is placed it is treated as a statement.
3. An unessential semi-colon if placed by the programmer is treated as an empty statement.
4. All statements should be written in lowercase letters. Generally, uppercase letters are used only for symbolic constants.
5. Blank spaces may be inserted between the words. This leads to improvement in the readability of the statements. However, this is not applicable while declaring a variable, keyword, constant and function.

6. It is not necessary to fix the position of statement in the program; i.e. a programmer can write the statement anywhere between the two braces following

the declaration part. The user can also write one or more statements in one line separating them with a semi-colon (;). Hence, it is often called a free-form language. The following statements are valid:

```
a=b+c;
d=b*c;
or
a=b+c; d=b*c;
```

7. The opening and closing braces should be balanced, i.e. if opening braces are four; closing braces should also be four.

EXECUTING THE C PROGRAM

A C program must go through various phases such as creating a program with editor, execution of preprocessor program, compilation, linking, loading and executing the program. The following steps are essential in C when a program is to be executed in MS-DOS mode:

1. **Creation of a Program:**The program should be written in C editor. The file name does not necessarily include extension '.C'. The default extension is '.C'. The user can also specify his/her own extension. The C program includes preprocessor directives.
2. **Execution of a Preprocessor Program:**After writing a program with C editor the programmer has to compile the program with the command (Alt-C). The preprocessor program executes first automatically before the compilation of the program. A programmer can include other files in the current file. Inclusion of other files is done initially in the preprocessor section.
3. **Compilation and Linking of a Program:** The source program contains statements that are to be translated into object codes. These object codes are suitable for execution by the computer. If a program contains errors the programmer should correct them. If there is no error in the program, compilation proceeds and translated program is stored in another file with the same file name with extension '.obj'. This object file is stored on the secondary storage device such as a disc.

   Linking is also an essential process. It puts together all other program files and functions that are required by the program. For example, if the programmer is using pow() function, then the object code of this function should be brought from math.h library of the system and linked to the main() program. After linking, the program is stored on the disc.
4. **Executing the Program:**After the compilation the executable object code will be loaded in the computer's main memory and the program is executed. The loader performs this function. All the above steps/phases of C program can be performed using menu options of the editor.

pre-processor directories/program is executed before compilation of the main program. The compiler checks the program and if any syntax error is found, the

same is displayed. The user is again forced to go to edit window. After removing an error, the compiler compiles the program. Here, at this stage object code is generated. During the program execution, if user makes mistakes in inputting data, the result would not be appropriate. Therefore, the user again has to enter the data. The output is generated when a program is error free.

ADVANTAGES OF C

1. It contains a powerful data definition. The data type supported are characters, alphanumeric, integers, long integer, floats and double. It also supports string manipulation in the form of character array.
2. C supports a powerful set of operators.
3. It also supports powerful graphics programming and directly operates with hardware. Execution of program is faster.
4. An assembly code is also inserted into C programs.
5. C programs are highly portable on any type of OS platforms.
6. System programs such as compilers, operating systems can be developed in C. For example, the popular operating system UNIX is developed in C.
7. The C language has 32 keywords and about 145 library functions and near about 30 header files.
8. C works closely with machines and matches assembly language in many ways.

HEADER FILES

stdio.h: Standard input and output files. All formatted and unformatted functions include file operation functions defined in this file. The most useful formatted printf() and scanf() are defined in this file. This file must be included at the top of the program. Most useful functions from this header files are printf(), scanf(), getchar(), gets(), putc() and putchar().

conio.h: Console input and output. This file contains input and output functions along with a few graphic-supporting functions. The getch(), getche() and clrscr() functions are defined in this file.

math.h: This file contains all mathematical and other useful functions. The commonly useful functions from this files are floor(), abs(), ceil(), pow(), sin(), cos() and tan().

Commonly useful header files

Sr. No.	Header File	Functions	Function Examples
1.	stdio.h	Input, output and file operation functions	printf(), scanf()
2.	conio.h	Console input and output functions	clrscr(), getche()
3.	alloc.h	Memory allocation-related functions	malloc(), realloc()
4.	graphics.h	All graphic-related functions	circle(), bar3d()
5.	math.h	Mathematical functions	abs(), sqrt()
6.	string.h	String manipulation functions	strcpy(), strcat()
7.	bios.h	BIOS accessing functions	biosdisk()
8.	assert.h	Contains macros and definitions	void assert(int)
9.	ctype.h	Contains prototype of functions which test characters	isalpha(char)
10.	time.h	Contains date- and time-related functions	asctime()

There are different ways of representing the logical steps for finding a solution of a given problem. They are as follows:

1. Algorithm
2. Flowchart
3. Pseudo-code

In the algorithm, a description of the steps for solving a given problem is provided. Here, stress is given on the text. Flowchart represents the solution of a given problem graphically. Pictorial representation of the logical steps is a flowchart. Another way to express the solution of a given problem is by means of a pseudo-code.

ALGORITHM

The algorithm is defined as 'the finite set of steps, which provide a chain of actions for solving a problem'. Each step in the algorithm should be well defined.

An algorithm is a well-organized, pre-arranged and defined textual computational module that receives some value or set of values as input and provides a single value or a set of values as output. These well-defined computational steps are arranged in a sequence, which processes the given input into an output. Writing precise description of the algorithm using an easy language is most essential for understanding the algorithm. Every step is known as an instruction.

Write a program to swap two numbers using a third variable.

Algorithm for swapping two numbers:

STEP 1: Start.

STEP 2: Declare variables a, b, & c.

STEP 3: Read values of a & b.

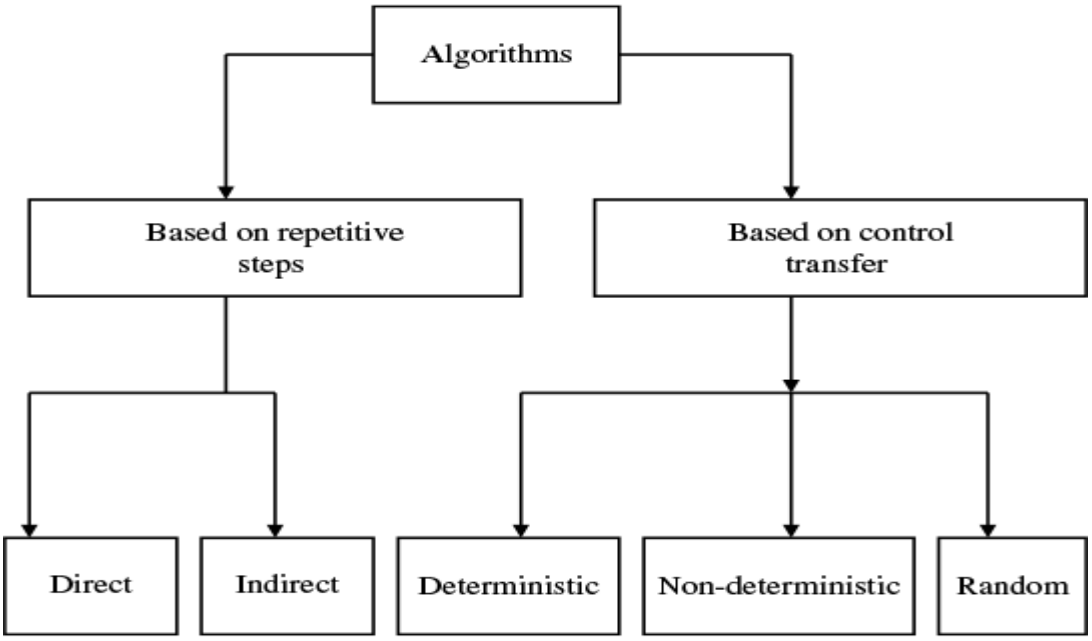
STEP 4: Copy value of a to c; b to a; & c to b.

STEP 5: Print swapped values of a & b.

STEP 6: Exit.

CLASSIFICATION OF ALGORITHMS

The classification of algorithm is based on repetitive steps and on control transfer from one statement to another. the classification of Algorithms.



On the basis of repetitive steps, an algorithm can further be classified into two types.

1. **Direct Algorithm:** In this type of algorithm, the number of iterations is known in advance.For example, for displaying numerical numbers from 1 to 10, the loop variable should be initialized from 1 to 10. The statement would be as follows:
    i. for (j=1;j<=10;j++)

ii. In the above statement, it is predicted that the loop will iterate 10 times.

2. **Indirect Algorithm:** In this type of algorithm, repetitively steps are executed. Exactly how many repetitions are to be made is unknown.

for (j=1;j<=n;j++)

For example, the repetitive steps are as follows:

1. To find the first five Armstrong numbers from 1 to n, where n is the fifth Armstrong number.
2. To find the first three palindrome numbers.

Based on the control transfer, the algorithms are categorized in the following three types.

1. **Deterministic:** Deterministic algorithm is based on either to follow a 'yes' path or 'no' path based on the condition. In this type of algorithm when control comes across a decision logic, two paths 'yes and 'no' are shown. Program control follows one of the routes depending upon the condition.

   Example:
   Testing whether a number is even or odd. Testing whether a number is positive or negative.
2. **Non-deterministic:**In this type of algorithm to reach to the solution, we have one of the multiple paths.

   Example:
   To find a day of a week.
3. **Random algorithm:**After executing a few steps, the control of the program transfers to another step randomly, which is known as a random algorithm.

A random search

Another kind of an algorithm is the infinite algorithm.

Infinite algorithms: This algorithm is based on better estimates of the results. The number of steps required would not be known in advance. The process will be continued until the best results emerged. For final convergence more iterations would be required.

Example:
To find shortest paths from a given source to all destinations in the network.

FLOWCHARTS

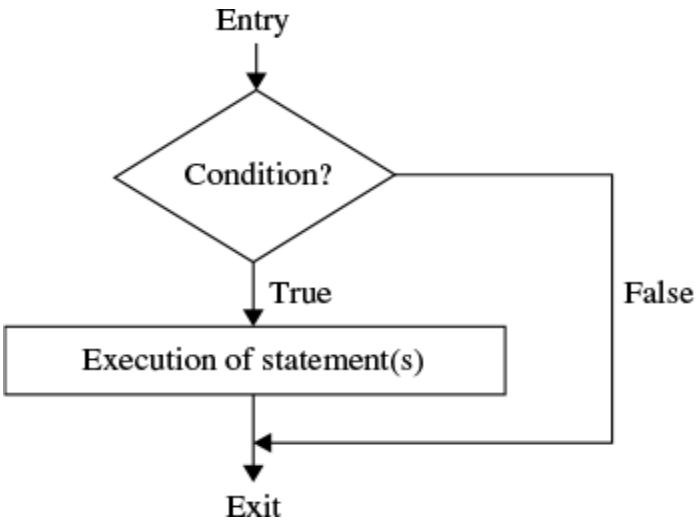
A flowchart is a visual representation of the sequence of steps for solving a problem. This is an easy way to solve the complex designing problems.

A flowchart is an alternative technique for solving a problem. Instead of descriptive steps, it use pictorial representation for every step. It shows a sequence of operations. A flowchart is a set of symbols, which indicates various operations in the program. For every process, there is a corresponding symbol in the flowchart. Once an algorithm is written, its pictorial representation can be done using flowchart symbols. In other words, a pictorial representation of a textual algorithm is a flowchart.

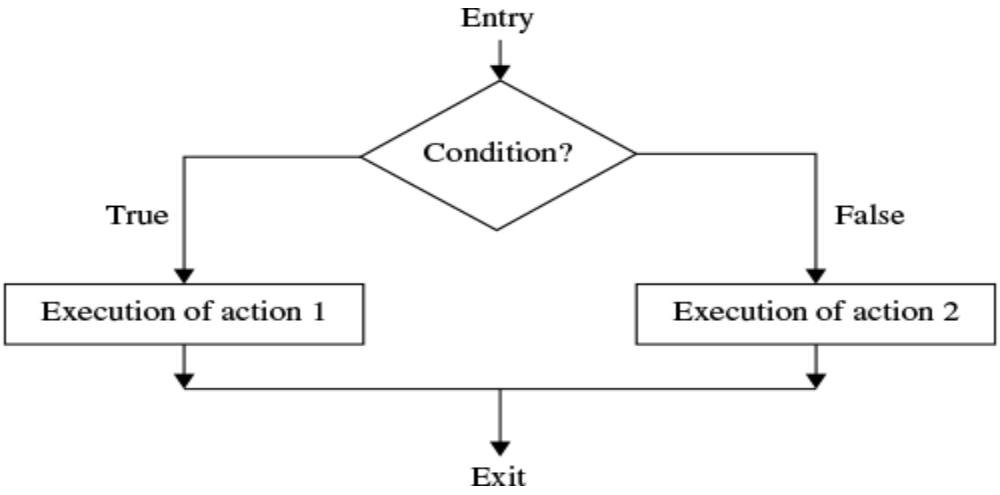
Start and end: The start and end symbols indicate both the beginning and the end of the flowchart. This symbol looks like a flat oval shaped. Usually this symbol is used twice in a flowchart, that is, at the beginning and at the end.

Decision or test symbol: The decision symbol is diamond shaped. This symbol is used to take one of the decisions. Depending on the condition the decision block selects one of the alternatives. While solving a problem, one can take a single, two or multiple alternatives depending upon the situation. In case the condition is satisfied /TRUE a set of statement(s) will be executed otherwise for false the control transfers to exit.

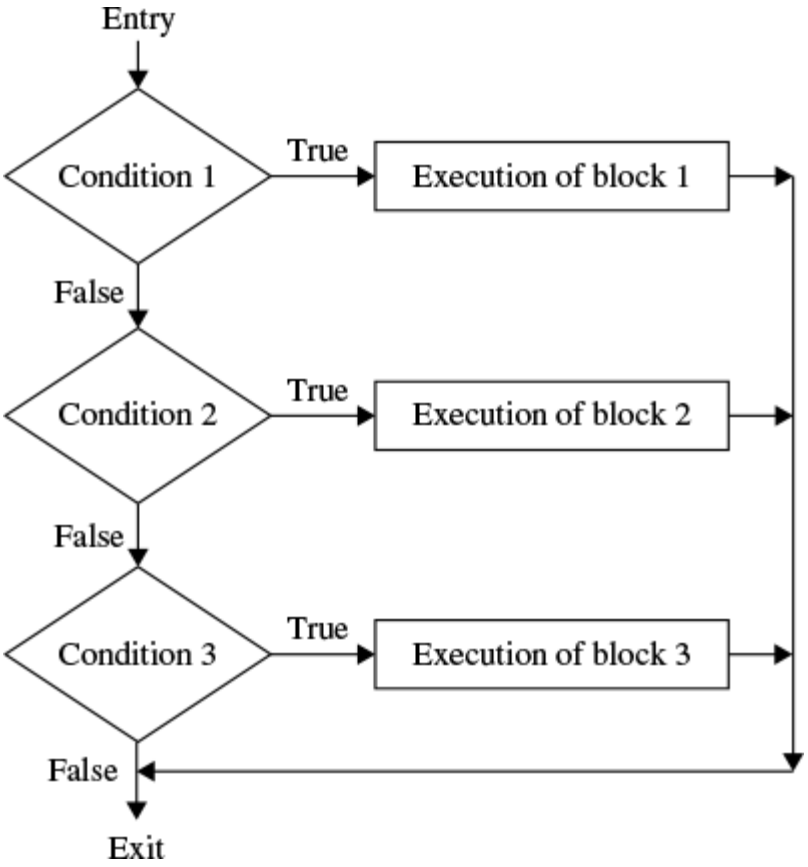
Single alternative decision: Here more than one flow line can be used depending upon the condition. It is usually in the form of a 'yes' or 'no' question, with branching flow line depending upon the answer. With a single alternative, the flow diagram will be as



Two alternative decisions: On satisfying the condition statement(s) pertaining to 1 action will be executed, otherwise the other statement(s) for action 2 will be executed.

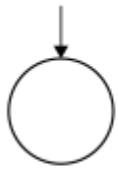


Multiple alternative decisions: Every decision block has two branches. In case the condition is satisfied, execution of statements of appropriate blocks take place, otherwise next condition will be verified. If condition 1 is satisfied then block 1 statements are executed. In the same way, other decision blocks are executed.

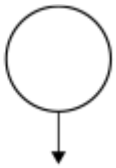


Connector symbol: A connector symbol has to be shown in the form of a circle. It is used to establish the connection, whenever it is impossible to directly join two

parts in a flowchart. Quite often, two parts of the flowcharts may be on two separate pages. In such a case, connector can be used for joining the two parts. Only one flow line is shown with the symbol. Only connector names are written inside the symbol, that is, alphabets or numbers.

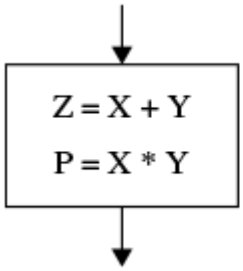


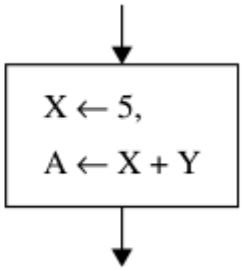
Connector for connecting to the next block



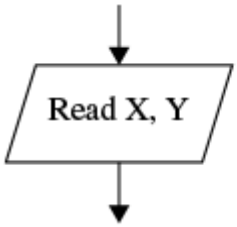
Connector that comes from the previous block

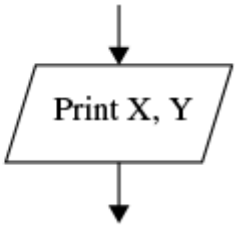
Process symbol: The symbol of process block should be shown by a rectangle. It is usually used for data handling, and values are assigned to the variables in this symbol. The operations mentioned within the rectangular block will be executed when this kind of block is entered in the flowchart. Sometimes an arrow can be used to assign the value of a variable to another. The value indicated at its head is replaced by the tail values. There are two flow lines connected with the process symbol. One line is incoming and the other line goes out.

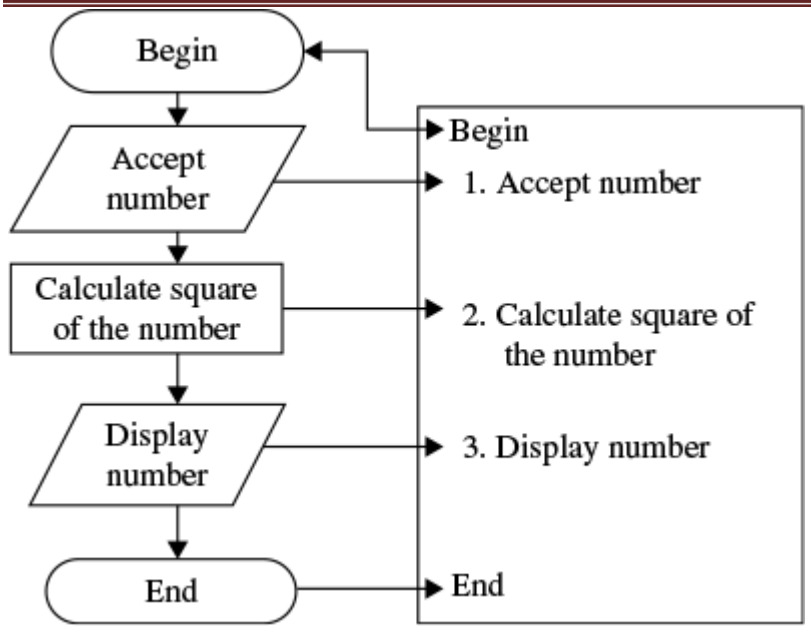




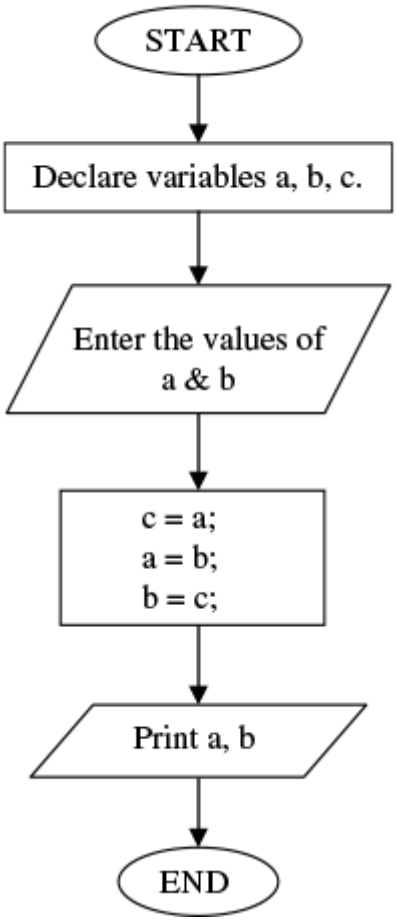
input/output symbol: Input/output symbol looks like a parallelogram. The input/output symbol is used to input and output the data. When the data is provided to the program for processing, then this symbol is used. There are two flow lines connected with the input/output symbol. One line comes to this symbol and the other line goes from this symbol.







Flowchart for swapping two numbers:



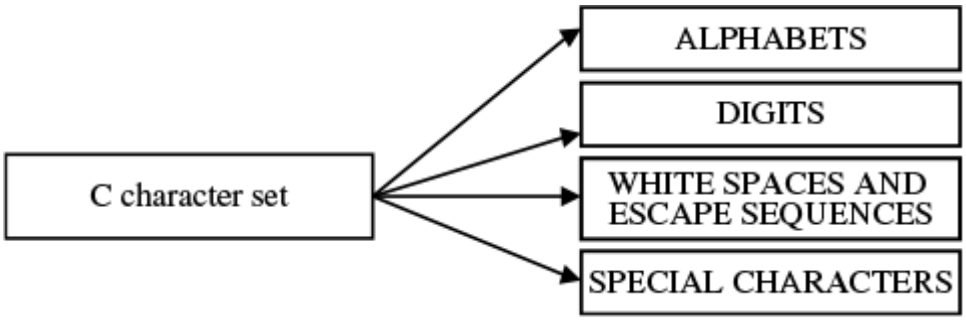
A program is a set of statements, which performs a specific task, executed by users in a sequential form. These statements/instructions are formed using certain words and symbols according to the rules known as syntax rules or grammar of the language. Every program must accurately follow the syntax rules supported by the language.

THE C CHARACTER SET

A character is a part of a word, sentence or paragraph. By using different characters, words, expressions and statements can be created on the basis of the requirement.

A character is represented by any alphabets in lowercase or uppercase, digits or special characters.

valid C character set which are as follows: (i) Letters (ii)Digits (iii) White Spaces & Escape Sequences and (iv) Special Characters. As C character set consists of escape sequences so each escape sequence begins with back slash (\) and it represents a single character. Any character can be represented as character constant using escape sequence.



Character set

Letters	***Digits***	***White Spaces and Escape Sequences***
Uppercase **A** to **Z**	All decimal digits **0** to **9**	Back space \b
Lowercase **a** to **z**		Horizontal tab \t
		Vertical tab \v
		New line \n

Letters	***Digits***	***White Spaces and Escape Sequences***
		Form feed \f
		Backslash \\
		Alert bell \a
		Carriage return \r
		Question mark \?
		Single quote \’
		Double quote \”

The C characters are assigned unique codes. There are many character codes used in the computer system. The widely and standard codes used in computer are ASCII and EBCDIC (Extended Binary Coded Decimal Interchange Code).

List of special characters

,	Comma	&	Ampersand
.	Period or dot	^	Caret
;	Semi-colon	*	Asterisk
:	Colon	–	Minus sign
'	Apostrophe	+	Plus sign
"	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	()	Parenthesis (left/right)
/	Slash	[]	Brackets (left/right)
\	Back slash	{}	Curly braces (left/right)
~	Tilde	%	Percent sign
_	Underscore	#	Number sign or hash
$	Dollar	=	Equal to
?	Question mark	@	At the rate

DELIMITERS

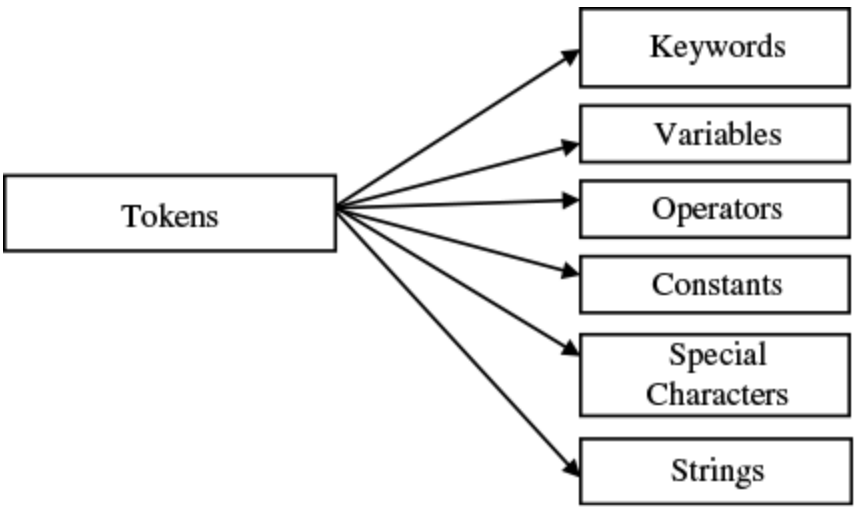
The language pattern of C uses special kind of symbols, which are called delimiters.

Delimiters

Delimiters	***Symbols***	***Uses***
Colon	:	Useful for label
Semi-colon	;	Terminates statements
Parenthesis	()	Used in expression and function
Square brackets	[]	Used for array declaration
Curly braces	{}	Scope of statement
Hash	#	Preprocessor directive
Comma	,	Variable separator
Angle brackets	<>	Header file

TYPES OF TOKENS

The smallest unit in a program/statement is called a token. The compiler identifies them as tokens. Tokens are classified in the following types.



The tokens are as follows:

1. **Keywords:** Key words are reserved by the compiler. There are 32 keywords (ANSI Standard).
2. **Variables:** These are user defined. Any number of variables can be defined.
3. **Constants:** Constants are assigned to variables.
4. O**perators:** Operators are of different types and are used in expressions.
5. **Special Characters:** These characters are used in different declarations in C.
6. **Srings:** A sequence of characters.

THE C KEYWORDS

The C keywords are reserved words by the compiler. All the C keywords have been assigned fixed meanings and they cannot be used as variable names.

C keywords

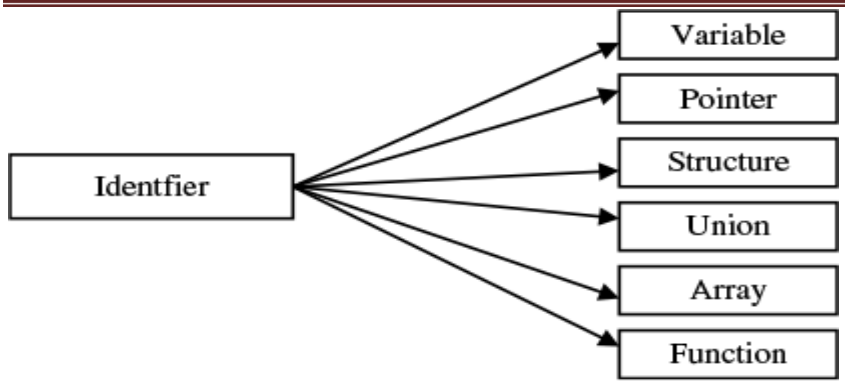
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Additional Keywords for Borland C

asm	cdecl	far	huge	interrupt	near	pascal

IDENTIFIERS

A symbolic name is used to refer to a variable, constant, function, structure, union etc. This process is done with identifiers. In other words, identifiers are the names of variables, functions, arrays, etc. They refer to a variety of entities such as structures, unions, enumerations, constants, typedef names, functions and objects . An identifier always starts with an alphabet, and it is a plain sequence of alphabets and/or digits. C identifier does not allow blank spaces, punctuations and signs.



Identifiers

Identifiers are user-defined names. They are generally defined in lowercase letters. However, the uppercase letters are also permitted. The underscore (_) symbol can be used as an identifier. In general, an underscore is used by a programmer as a link between two words for the long identifiers.

Valid identifiers are as follows:
length, area, volume, sUM, Average
Invalid identifiers are as follows:
Length of line, S+um, year's

Example:

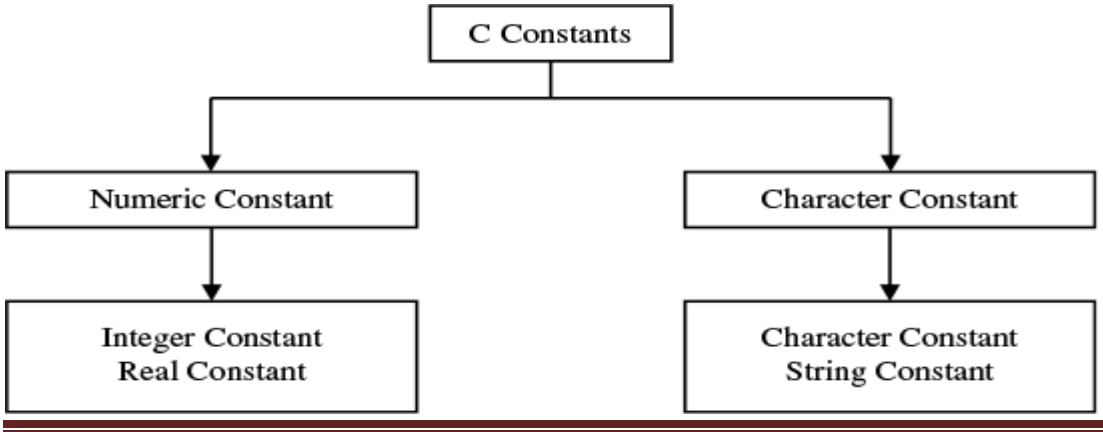
User-defined identifiers are as follows:

1. #define N 10
2. #define a 15

Here, 'N' and 'a' are user-defined identifiers.

CONSTANTS

The constants in C are applicable to the values, which do not change during the execution of a program. There are several types of constants in C. They are classified into following groups



Numerical Constants

1. **Integer Constants:** These constants are represented with whole numbers. They require a minimum of 2 bytes and a maximum of 4 bytes of memory.

   The following concepts are essential to follow the numerical constants:
   a. Numerical constants are represented with numbers. At least one digit is needed for representing the number.
   b. The decimal point, fractional part, or symbols are not permitted. Neither blank spaces nor commas are permitted.
   c. Integer constant could be either positive or negative or may be zero.
   d. A number without a sign is assumed as positive.

Some valid examples: 10, 20, +30, –15, etc.
Some invalid integer constants: 2.3, .235, $76, 3*^6, etc.

- Besides representing the integers in decimal, they can also be represented in octal or hexadecimal number system based on the requirement.
- Octal number system has base 8 and the hexadecimal number system has base 16. The octal numbers are 0, 1, 2, 3, 4, 5, 6, and 7 and the hexadecimal numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
- The representation of octal numbers in C would be done with leading digit 0 and for hex representation, with leading OX or O_X.
- Examples of octal and hexadecimal numbers:
- Octal numbers – 027, 037, 072
- Hexadecimal numbers – 0X9, 0Xab, 0X4

2. **Real Constants:** Real constants are often known as floating point constants. Real constants can be represented in exponential or fractional form. Integer constants are unfit to represent many quantities. Many parameters or quantities are defined not only in integers but also in real numbers. For example, length, height, price, distance, etc. are also measured in real numbers.

The following concepts are essential to follow the real numbers:
1. The decimal point is permitted.
2. Neither blank spaces nor commas are permitted.
3. Real numbers could be either positive or negative.
4. The number without a sign is assumed as positive.

Examples of real numbers are ***2.5, 5.521, 3.14, etc.***

The real constants can be written in exponential notation, which contains fractional and exponential parts. For example, the value 2456.123 can be written as 2.4561 X $e^{+}3$.

The part that precedes e is called mantissa and the part that follows it is an exponent. In this example, 2.4561 is the mantissa and +3 is the exponent.

The following points must be noted while constructing a real number in exponential form:

1. The real number should contain a mantissa and an exponent.
2. The letter 'e' separates the mantissa and the exponent and it can be written in lower or upper case.
3. The mantissa should be either a real number represented in decimal or integer form.
4. The mantissa may be either positive or negative.
5. The exponent should be an integer that may be positive or negative.

Some valid examples are 5.2e2,–2, 5.0e–5, 0.5e–3, etc.
In double type also the real numbers can be expressed with mantissa and exponent parts.

Character Constant

1. **Single Character Constants:** A character constant is a single character. It can also be represented with single digit or a single special symbol or white space enclosed within a pair of single quote marks or character constants are enclosed within single quotation marks.

Example:
'a', '8', '–'
Length of a character, at the most, is one character.

Character constants have integer values known as ASCII values. For example the statement printf ("%c %d", 65, 'B') will display character 'A' and 66.

2. **String Constants:** String constants are a sequence of characters enclosed within double quote marks. The string may be a combination of all kinds of symbols.

Example:
"Hello", "India", "444", "a".

VARIABLES

A variable is used to store values. It has memory location and can store single value at a time. When a program is executed, many operations are carried out on the data. The data types are integers, real or character constants. The data is stored in the memory, and at the time of execution it is fetched for carrying out different operations on it.

A variable is a data name used for storing a data value. Its value may be changed during the program execution. The variable value keeps on changing during the

execution of the program. In other words, a variable can be assigned different values at different times during the program execution. A variable name may be declared based on the meaning of the operation. Variable names are made up of letters and digits. Some meaningful variable names are as follows.

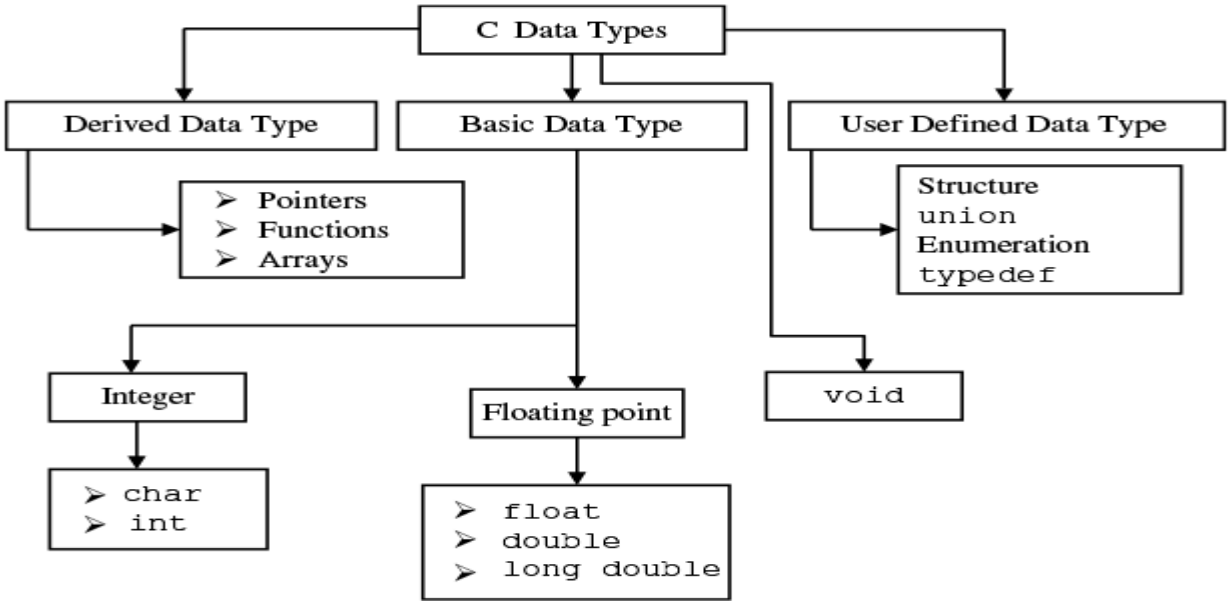
RULES FOR DEFINING VARIABLES

1. A variable must begin with a character or an underscore without spaces. The underscore is treated as one type of character. It is very useful to increase readability of variables having long names. It is advised that the variable names should not start with underscore because library routines mostly use such variable names.
2. The length of the variable varies from compiler to compiler. Generally, most of the compilers support eight characters excluding extension. However, the ANSI standard recognizes the maximum length of a variable up to 31 characters.
3. The variable should not be a C keyword.
4. The variable names may be a combination of uppercase and lowercase characters. For example, suM and sum are not the same. The traditional practice is to use lowercase characters for variable names and uppercase letters for symbolic constants.
5. The variable name should not start with a digit.
6. Blanks and commas are not permitted within a variable name.

DATA TYPES

The basic data types are char, int, float and double.

The C supports a number of qualifiers that can be applied to these basic data type. The short and long qualifiers provide different length to int. The short is always 16 bits and long is 32 bits, i.e. int is either 16 or 32 bits.



C data type can be classified as follows:

1. **Basic Data Type:**
   (a) Integer (int), (b) character (char), (c) floating point (float), (d) double floating point (double).
2. **Derived Data Type:**
   (a) Pointer: the variable that stores the address of another variable.

   Eg: int *p;

   Declares a pointer variable p. This variable can be assigned by a normal variable.

   Eg: int a;

   p = &a;

   Now the pointer gets the address of variable a,

   (b) Functions: Dividing the program into different modules

```
void sum();

void main()
{
sum();

 }

void sum()

 {

Statements;

 }
```

   The above sample program has two functions sum() and main(). Where sum()is a user defined function.

   (c) Arrays: Homogeneous memory allocation. The elements are stored in continues memory location.

int a[10]; the variable a can store values upto 10. The variable start from

a[0],a[1],....a[9].

3. User defined data types:

i. Structure: To declare all the variables for the program under a common name. It uses separate memory location for the variables.

```
struct struct name
    {
    int rollno;
    char   name[20];
    char grade;
    };
```

ii. Union: same as structure. It uses same memory location for the variables.

```
union union name
    {
    int rollno;
    char name[20];
    char grade;
    };
```

iii. Enumeration: It is used for declaring enumeration type.
enum tagname {var1, var2, …varn};
Eg: enum month {jan,feb,march….};

Here jan will be assigned a constant value 0, feb as 1 and so on. The identifiers are not to be enclosed in quotation mark. Integer constants are not permitted.

iv.Typedef:

Used to create new datatype.

typedef type dataname;

typedef char string[50]; // create a new data type string

String name; //creates a string variable name of size 50Derived data types are pointers, functions and arrays.

C DATA TYPES

1. **Integer Data Type**
   int, short and long

   All C compilers offer different integer data types. They are short and long. Short integer requires half the space in memory than the long. The short integer requires 2 bytes and long integers 4 bytes

Difference between short and long integers

Short Integer	**Long Integer**
Occupies 2 bytes in memory	Occupies 4 bytes in memory
Range: −32,768 to 32,767	Range: −2147483648 to 2147483647
Program runs faster	Program runs slower
Format specifier is %d or %i	Format specifer is %ld
Example:	*Example:*
int a=2;	long b=123456;
short int b=2;	long int c=1234567;
When variable is declared without short or long keyword, the default is short-signed int.	

a. Integers Signed and Unsigned

The difference between the signed integers and unsigned

Difference between signed and unsigned integers

Signed Integer	***Unsigned Integer***
Occupies 2 bytes in memory	Occupies 2 bytes in memory
Range: −32,768 to 32,767	Range: 0 to 65535
Format specifier is %d or %i	Format specifier is %u

Signed Integer	***Unsigned Integer***
By default signed int is short signed int	By default unsigned int is short unsigned int
There are also long signed integers having Range from −2147483648 to 2147483647.	There are also long unsigned int with range 0 to
Example:	4294967295
int a=2;	Example:
long int b=2;	unsigned long b=567898;
	unsigned short int c=223;
	When a variable is declared as unsigned the negative range of the data type is transferred to positive, i.e. doubles the largest size of the possible value. This is due to declaring unsigned int; the 16th bit is free and not used to store the sign of the number.

2. char, signed and unsigned:

Difference between signed and unsigned char

Signed Character	***Unsigned Character***
Occupies 1 bytes in memory	Occupies 1 bytes in memory
Range: −128 to 127	Range: 0 to 255
Format specifier: %c	Format specifier: %c

Signed Character	***Unsigned Character***
When printed using %d format specifier, prints ASCII character.	When printed using %d format specifier, prints ASCII character
char ch='b';	unsigned char ='b';

3. **Floats and Doubles:**

Difference between floating and double floating

Floating	***Double Floating***
Occupies 4 bytes in memory	Occupies 8 bytes in memory
Range: 3.4e−38 to +3.4e+38	Range: 1.7 e−308 to +1.7e+308
Format string: %f	Format string: %lf
Example:	Example:
float a;	double y;
	There also exist long double having ranged 3.4e − 4932 to 1.1e + 4932 and occupies 10 bytes in memory.
	Example:
	long double k;

4. Entire data types in C: The entire data types supported by the C

Table 2.10 *Data types and their control strings*

Data Type	Size (bytes)	Range	Format String
char	1	–128 to 127	%c
unsigned char	1	0 to 255	%c
short or int	2	–32,768 to 32,767	%i or %d
unsigned int	2	0 to 65535	%u
long	4	–2147483648 to 2147483647	%ld
unsigned long	4	0 to 4294967295	%lu
float	4	3.4 e–38 to 3.4 e + 38	%f or %g
double	8	1.7 e–308 to 1.7 e + 308	%lf
long double	10	3.4 e–4932 to 1.1 e + 4932	%lf
enum	2*	–32768 to 32767	%d

DECLARING VARIABLES

The declaration of variables should be done in declaration part of the program. The variable must be declared before they are used in the program. Declaration ensures the following two things: (i) compiler obtains the variable name and (ii) it tells to the compiler data type of the variable being declared and helps in allocating the memory. The variable can also be declared before main() such variables are called external variables. The syntax of declaring a variable is as follows.

Syntax:

Data_type variable_name;

Example:

```
int   age;
char m;
 float s;
 double k;
 int a,b,c;
```

The int, char, float and double are keywords to represent data types. Commas separate the variables, in case variables are more than one.

Data types and keywords

Data Types	***Keyword***
Character	char
Signed character	signed char

Data Types	***Keyword***
Unsigned character	unsigned char
Integer	int
Signed integer	signed int
Unsigned integer	unsigned int
Unsigned short integer	unsigned short int
Signed long integer	signed long int
Unsigned long integer	unsigned long int
Floating point	float
Double floating point	double
Extended double point	long double

INITIALIZING VARIABLES

Variables declared can be assigned or initialized using assignment operator '='. The declaration and initialization can also be done in the same line.

Syntax:

variable_name = constant;

or

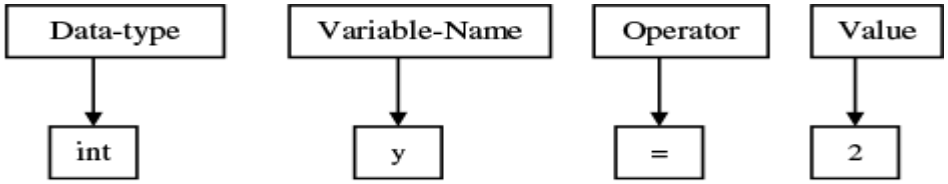
data_type variable_name= constant;

Example:
x=5; where x is an integer variable.
Example:
int y=4;

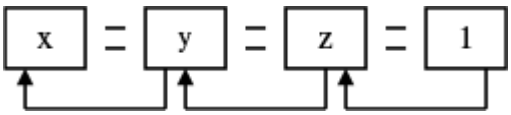
Example:

int x,y,z; /* Declaration of variables */

The third example as cited above for declaration of variables is also a valid statement.



Value assignment



Multiple assignment

In Figure , the variable z is assigned a value 1, and z then assigns its value to y and again y to x. Thus, initialization in chain system is done. However, note that following declarations are invalid:

int x=y=z=3; /* invalid statement */

DYNAMIC INITIALIZATION

The initialization of variable at run time is called dynamic initialization. Dynamic refers to the process during execution. In C initialization can be done at any place in the program.

TYPE MODIFIERS

The keywords signed, unsigned, short and long are type modifiers. A type modifier changes the meaning of basic data type and produces a new data type. Each of these type modifiers is applicable to the basic data type int. The modifiers signed and unsigned are also applicable to the type char. The long type modifier can be applicable to double type.

Example:

long l; /* int data type is applied */

int s; /* signed is default */

unsigned long int; /* int keyword is optional */

TYPE CONVERSION

In C, it is possible to convert one data type to another. This process can be done either explicitly or implicitly. The following section describes explicit type conversion.

1. Explicit type conversion
   Sometimes a programmer needs the result in certain data type, for example division of 5 with 2 should return float value. Practically the compiler always returns integer values because both the arguments are of integer data type.
2. Implicit type conversion

   In C automatically 'lower' data type is promoted to 'upper' data type. For example, data type char or short is converted to int. This type of conversion is called as automatic unary conversion. Even when binary operator has different data types, 'lower' data type is converted to 'upper' data type.
   In case an expression contains one of the operands as unsigned operands and another non-unsigned, the latter operand is converted to unsigned.

Similarly, in case an expression contains one of the operands as float operands and another non-float, the latter operand is converted to float.

Similarly, in case an expression contains one of the operands as double operand and another non-double, the latter operand is converted to double.

Similarly, in case an expression contains one of the operands as long int and the other unsigned int operands, the latter operand is converted to long int.

the following table describes automatic data type conversion from one data type to another while evaluating an expression.

Expression	*Operator*	*Operand 1*	*Operand 2*	*Outcome*
a	unary	short	–	int
a/b	binary	int	float	float
a/b-c	binary	float	long int	float
(a/b-c)*d	binary	float	double	double

WRAPPING AROUND

When the value of variable goes beyond its limit, the compiler would not flag any error message. It just wraps around the value of a variable. For example, the range of unsigned integer is 0 to 65535. In this type, negative values and values greater than 65535 are compatible.

unsigned int x=65536;

The value assigned is greater by 1 than the range. Here, the compiler starts again from beginning after reaching the end of the range.

Constant Variable

If we want the value of a certain variable to remain the same or unchanged during the program execution, it can be done by declaring the variable as a constant. The keyword const is then prefixed before the declaration. It tells the compiler that the variable is a constant. Thus, constant declared variables are protected from modification.

Example:

```
const int m=10;
```

Where const is a keyword, m is variable name and 10 is a constant.

The compiler protects the value of 'm' from modification. The user cannot assign any value to 'm', but using pointer the value can be changed. When user attempts to modify the value of constant variable, the message 'cannot modify the constant object' will be displayed.

Volatile Variable

The volatile variables are those variables that can be changed at any time by other external program or the same program. The syntax is as follows:

```
volatile int d;
```

Module II: Basic Elements: Operators and Expressions: Expression Evaluation (Precedence of Operators); simple I/O statements, Control structures, if, if else, switch-case, for, while, do-while, break, continue. Arrays: Defining simple arrays, Multi-dimensional arrays, declaration, initialization and processing.

An operator indicates an operation to be performed on data that yields a new value. An operand is a data item on which operators perform operations. C is rich in use of different operators. C provides four classes of operators which are: (i) arithmetic, (ii) relational, (iii) logical and (iv) bitwise. Apart from these basic operators, C also supports additional operators.

Types of operators

Type of Operator	***Symbolic Representation***
Arithmetic operators	+, −, *, / and %
Relational operators	>,<,= =,>=,<= and !=
Logical operators	&&,|| and !
Increment and Decrement operators	++ and −−
Assignment operator	=, *=, /=, %=,+=, −=, &=, ^=,|=, <<=, >>=
Bitwise operators	&,|,^,>>,<< and ˜
Comma operator	,
Conditional operators	?:

OPERATOR PRECEDENCE

Precedence means priority. Every operator in C has assigned precedence (priority). An expression may contain a lot of operators. The operations on the operands are carried out according to the priority of the operators. The operators having higher priority are evaluated first and then lower priority.

For example, in arithmetic operators, the operators *, / and % have assigned highest priority and of similar precedence. The operators + and − have the lowest priority as compared to the above operators.

Prioritywise list of operators

Operators	*Operation*	*Associativity or Clubbing*	*Priority*
()	Function call	Left to right	1st
[]	Array expression or square bracket		
->	Structure operator		
.	Structure operator		
+	Unary plus	Right to left	2nd
-	Unary minus		
++	Increment		
--	Decrement		
!	Not operator		
~	One's complement		
*	Pointer operator		
&	Address operator		
sizeof	Size of an object		
Type	Type cast		
*	Multiplication	Left to right	3rd
/	Division		
%	Modular division		
+	Addition (Binary plus)	Left to right	4th
-	Subtraction (Binary minus)		

ASSOCIATIVITY

When an expression contains operators with equal precedence then the associativity property decides which operation to be performed first. Associativity means the direction of execution. Associativity is of two types.

1. **Left to right:** In this type, expression, evaluation starts from left to right direction.

   Example:
   12 * 4 / 8 % 2
   In the above expression, all operators are having the same precedence, and so associativity rule is followed (i.e. direction of execution is from left to right).
   = 48 / 8 % 2
   = 6 % 2
   = 0 (The above modulus operation provides remainder 0)
2. **Right to left:** In this type, expression, evaluation starts from right to left direction.

Example:

a=b=c

In the above example, assignment operators are used. The value of c is assigned to b and then to a. Thus, evaluation, is from right to left.

COMMA AND CONDITIONAL OPERATOR

1. **Comma Operator (,):** The comma operator is used to separate two or more expressions. The comma operator has the lowest priority among all the operators. It is not essential to enclose the expressions with comma operators within the parentheses. For example, the following statements are valid.

   ***Example*:**

   a. a=2,b=4,c=a+b;
   b. (a=2,b=4,c=a+b;)

2. **Conditional Operator (?):** The conditional operator contains condition followed by two statements or values. The condition operator is also called the ternary operator. If the condition is true, the first statement is executed, otherwise the second statement is executed.

   Conditional operators (?) and (:) are sometimes called ternary operators because they take three arguments. The syntax of the conditional operator is as follows:

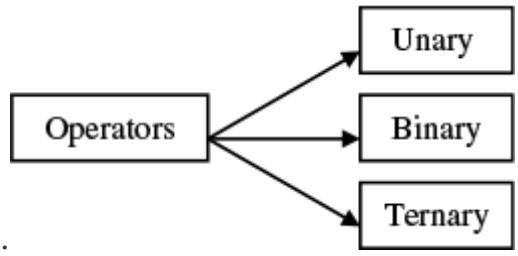
   ***Syntax*:**

   Condition ? (expression1) : (expression2);

   Two expressions are separated by a colon. If the condition is true expression 1 gets evaluated, otherwise expression 2 gets evaluated. The condition is always written before the ? mark.

ARITHMETIC OPERATORS

There are three types of arithmetic operators. These are (i) binary operator and (ii) unary operator and (iii) ternary operator



.

1.

Binary Operator: These operators are commonly used in most of the computer languages. These arithmetic operators are used for numerical calculations between

the two constant values. They are also called Binary Arithmetic Operators. Binary operators are those operators which require two operands. C evaluates arithmetic operations as follows.

Division, multiplication and remainder operations are solved first. When an expression contains many operators such as multiplication, division and modular division operations, expression evaluation starts from left to right. Multiplication, division and modular division operators are having equal level of precedence.
The addition and subtraction operations are solved after division, multiplication and modular division operations. Evaluation starts from left to right. Addition and subtraction have equal level of precedence.
the modular division operator cannot be applied to float and double data types.

Arithmetic operators

Arithmetic Operators	***Operator Explanation***	***Examples***
+	Addition	2+2=4
–	Subtraction	5–3=2
*	Multiplication	2*5=10
/	Division	10/2=5
%	Modular division	11%3=2 (Remainder 2)

2.
Unary Operators: The operators which require only one operand are called unary operators. Unary operators are increment operator (++), decrement (––) and minus (–).

Unary arithmetic operators

Operator	***Description or Action***
–	Minus
++	Increment
––	Decrement
&	Address operator
sizeof	Gives the size of an operator

1.
Minus (−): Unary minus is used for indicating or changing the algebraic sign of a value.

***Example*:**
int x=−50;
int y=−x;
assign the value of −50 to x and the value of −50 to y through x. The minus (−) sign used in this way is called the unary operator because it takes just one operand. There is no unary plus (+) in C. Even though, a value assigned with plus sign is valid, for example int x=+50, here, + is valid, but in practice this sign should not be attached in C.
2.
Increment (++) and Decrement (−−) Operators: The C compilers produce very fast and efficient object codes for increment and decrement operations. This code is better than generated by using the equivalent assignment statement. So, increment and decrement operators should be used whenever possible.
The operator ++ adds one to its operand, whereas the operator −− subtracts one from its operand. For justification, x=x+1 can be written as x++; and x=x−1; can be written as x−−;. Both these operators may either follow or precede the operand. That is x=x+1; can be represented as x++; or ++x;
If '++' or '−−' are used as a suffix to the variable name, then post-increment/decrement operations take place. Consider an example for understanding the '++' operator as a suffix to the variable.
x=20;
y=10;
z=x*y++;
In the above equation, the current value of y is used for the product. The result is 200, which is assigned to 'z'. After multiplication the value of y is incremented by one.
If '++' or '−−' are used as a prefix to the variable name, then pre increment/decrement operations take place. Consider an example for understanding '++' operator as a prefix to the variable.
x=20;
y=10;
z=x*++y;

```
z=x*y++;

a=x*y;

printf("\n%d %d",z,a);
```

} In the above equation, the value of y is incremented and then multiplication is carried out. The result is 220, which is assigned to 'z'. The following programs can be executed for verification of increment and decrement operations.

3.

sizeof and '&' Operator :

The sizeof operator gives the bytes occupied by a variable, i.e. the size in terms of bytes required in memory to store the value. The number of bytes occupied varies from variable to variable depending upon its data types.

The '&' operator prints address of the variable in the memory. The below mentioned example illustrates the use of these operators.

RELATIONAL OPERATORS

These operators are used to distinguish between two values depending on their relations. These operators provide the relationship between two expressions. If the relation is true then it returns a value 1 otherwise 0 for false. The relational operators together with their descriptions,

Operator	*Description or Action*	*Example*	*Return Value*
>	Greater than	5>4	1
<	Less than	10<9	0
<=	Less than or equal to	10<=10	1
>=	Greater than equal to	11>=5	1
==	Equal to	2==3	0
!=	Not equal to	3!=3	0

Assignment operators

=	*=	/=	%=
+=	-=	<<=	>>=
>>>=	&=	^=	!=

Consider the following example: k=k+3; in this expression, the variable on the left side of = is repeated on the right. The same can be written as follows:

k+=3;

The operators = and += are called assignment operators. The binary operators which need two operands to their either side are surely have a corresponding assignment operator **operand** = to the left side.

LOGICAL OPERATORS

The logical relationship between the two expressions is tested with logical operators. Using these operators, two expressions can be joined. After checking the conditions, it provides logical true (1) or false (0) status. The operands could be constants, variables and expressions the three logical operators

Operator	*Description or Action*	*Example*	*Return Value*
&&	Logical AND	5>3 && 5<10	1
||	Logical OR	8>5 || 8<2	1
!	Logical NOT	8!=8	0

1. The logical AND (&&) operator provides true result when both expressions are true, otherwise 0.
2. The logical OR (||) operator provides true result when one of the expressions is true, otherwise 0.
3. The logical NOT operator (!) provides 0 if the condition is true, otherwise 1.

BITWISE OPERATORS

C supports six bit operators. These operators can operate only on integer operands such as int, char, short, long.

Bitwise operators

Operators	***Meaning***
>>	Right shift
<<	Left shift
^	Bitwise XOR (exclusive OR)
~	One's complement
&	Bitwise AND
|	Bitwise OR

Shifting two bits right means the inputted number is to be divided by 2^s where s is the number of shifts i.e. in short $y=n/2^s$, where n = number and s = the number of position to be shift.

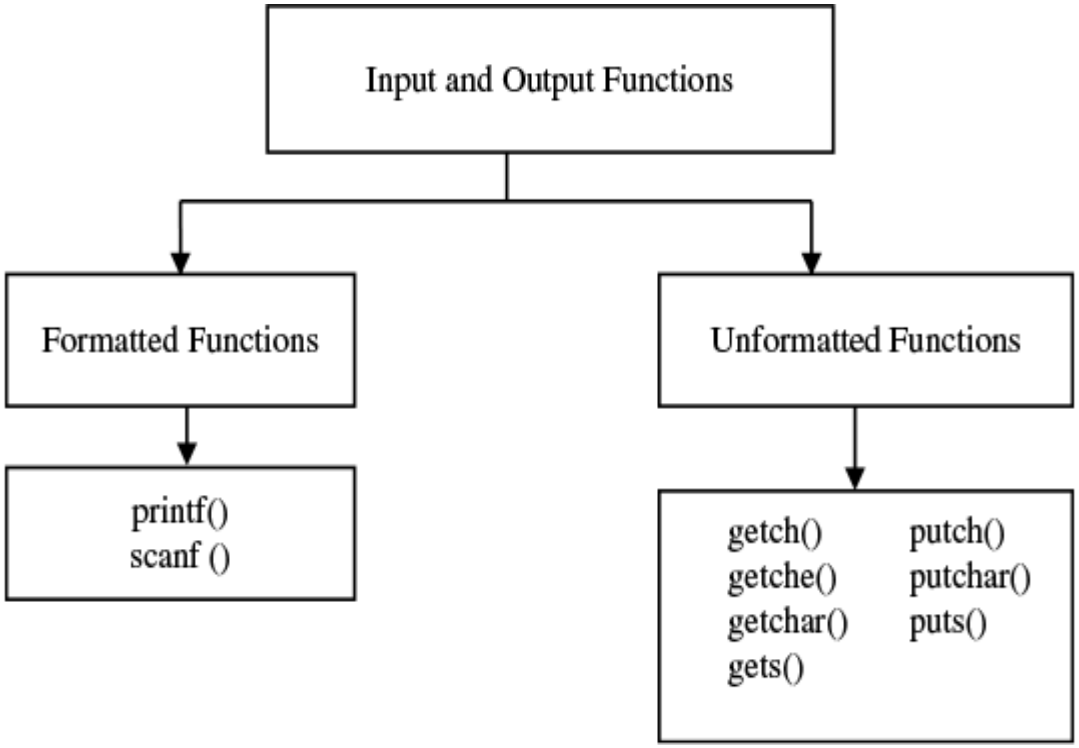
Shifting three bits left means the number is multiplied by 8; in short $y=n*2^s$ where n = number and s = the number of position to be shifted.

There are a number of I/O functions in C, based on the data types. The input/output functions are classified in two types:

1. Formatted functions
2. Unformatted functions

With formatted functions, the input or output is formatted as per our requirement. The readability in easy way is possible with formatted functions. For example, with formatted functions one can decide how the result should appear or display on the screen.

Formatted Functions: The formatted input/output functions read and write, respectively, all types of data values. They require format string to produce formatted results. Hence, they can be used for both reading and writing of all data values. The formatted functions return values after execution. The return value is equal to the number of variables successfully read/written.



1.
scanf("control string", argu1, argu2,. . .);
Precisely, if we write scanf() as scanf("%d",&x); where %d is a control string which is nothing but conversion specification and it is to be placed within double quote. The other part is the argument and a sign & (ampersand) must precede it. Arguments are the identifiers.
For displaying the result, prinf() formatted function is used.

2. **Unformatted Functions:** The unformatted input/output functions work only with character data type. They do not require format conversion symbol for formatting of data types because they work only with character data type. There is no need to convert data. In case, values of other data types are passed to these functions, they are treated as character data.

FORMATTED FUNCTIONS

1. The printf() statement: The formatted output as per the programmers requirement is displayed on the screen with printf(). The list of variables can be indicated in the printf().The syntax of printf() statement is as follows:

   printf("Control string",variable1,variable2,. . .variable n);

   The control string specifies the field format such as %d, %s, %f, and variables as taken by the programmer.

FLAGS, WIDTHS AND PRECISION WITH FORMAT STRING

Flags: Flags are used for output justification, numeric signs, decimal points, trailing zeros. The flag (−) left justifies the result. If it is not given, the default is right justification. The plus (+) signed conversion result always starts with a plus (+) or a minus (−) sign.

Width specifier : It sets the minimum field width for an output value. Width can be specified through a decimal point or using an asterisk '*'.

Formats for various outputs

Sr. No.	Format	Meaning	Explanation
1	%wd	Format for integer output	w is width in integer and d is conversion specification
2	%w.cf	Format for float numbers	w is width in integer, c specifies the number of digits after decimal point and f specifies the conversion specification
3	%w.cs	Format for string output	w is width for total characters, c are used for displaying leading blanks and s specifies conversion specification

2. The scanf() statement: The scanf() statement reads all types of data values. It is used for runtime assignment of variables. The scanf() statement also requires conversion symbol to identify the data to be read during the execution of the program. The scanf() statement stops functioning when some input entered does not match with format string. The syntax of the scanf() statement is the same as printf() except they work exactly opposite of each other.

***Syntax*:**

The syntax of the input function for inputting the data is scanf().

***Example*:**

scanf("control string", address of variable 1,address of variable 2,----);

The control string has to be enclosed within double quotes. It specifies the format specifier, such as %d for integer, %f for float, %c for character, etc. and the data are to be invoked by arguments, such as address of variable1, address of variable 2, etc.

scanf("%d",&x);

Here, " %d" is the format specifier in the control string, which is nothing but the conversion specification and it is to be placed within double quotes. The other part is the variable and & (ampersand) must precede it.

The format specifiers and their meanings are given below.
%d: The data is taken as integer.
%c: The data is taken as character.
%s: The data string.
%f: The data is taken as float.

scanf("%d %f %c",&a,&b,&c);

The scanf() statement requires '&' operator called address operator. The address operator prints the memory location of the variable. Here, in the scanf() statement the role of '&' operator is to indicate the memory location of the variable, so that the value read would be placed at that location.

The scanf() statement also returns values. The return value is exactly equal to the number of values correctly read. In case of any mismatch, error will be thrown.

Formats for the various inputs

Sr. No.	*Format*	*Meaning*	*Explanation*
1.	`%wd`	Format for integer input	w is width in integer and d conversion specification
2.	`%w.cf`	Format for float point input	w is width in integer, c specifies the number of digits after decimal point and f is conversion specification
3.	`%w.cs`	Format for string input	w is the width for total characters, c are used for inserting blanks and s is conversion specification

The prinf() and scanf() statements follow different data types It can be seen from this table that format string is initialized with %sign as a special character, which indicates the format of the data to be displayed on the screen.

Data types with conversion symbols

Data Type	***Format String***	
Integer	Short integer	%d or %i
	Short unsigned	%u
	Long signed	%ld
	Long unsigned	%lu
	Unsigned hexadecimal	%x
	Unsigned octal	%o
Real	Floating	%f or %g
	Double floating	%lf
Character	Signed character	%c
	Unsigned character	%c
	String	%s

Data Type	***Format String***	
Octal number		%o
Displays Hexa deci mal number in lowe rcase		%hx
Displays Hexa deci mal number in upp ercase		%p
Aborts program wit h error		%n

The printf() and scanf() statements follow the combination of characters called escape sequences. In order to come out, computers from routine sequence escape sequences are used. These are nothing but special characters starting with '\'.

Escape sequences with their ASCII values

Escape Sequence	*Use*	*ASCII Value*
\n	New line	10
\b	Backspace	8
\f	Form feed	12
\'	Single quote	39
\\	Backslash	92
\o	Null	0
\t	Horizontal tab	9
\r	Carriage return	13
\a	Alert	7
\"	Double quote	34

\v	Vertical tab	11
\?	Question mark	63

UNFORMATTED FUNCTIONS

C has three types of I/O functions.

1. Character I/O
2. String I/O
3. File I/O

4. Character I/O

1.getchar() -
This function reads a character-type data from standard input. It reads one character at a time till the user presses the enter key. The syntax of the getchar() is as follows:

Variable name=getchar();

***Example*:**

```
char c;
c=getchar();
```

A program is supported for the following getchar() function.

Write a program to accept characters through keyboard using getchar() function.

```
 void  main()
        {
        char c;
        clrscr();
printf("\nEnter a char :");
c=getchar();
printf("a=%c",c);
}
```

OUTPUT:

```
Enter a char :g
a=g
```

2.

putchar();
This function prints one character on the screen at a time, read by the standard input.

The syntax is as follows:

putchar(variable name);

***Example*:**
char c='C';
putchar (c);

A program is provided on putchar().

➢ 4.20 Write a program to use putchar() in work.

```
 void main()
{
        char c='C';
        clrscr();
putchar(c);
}
```

OUTPUT:
C

3.

getch() & getche()

These functions read any alphanumeric character from the standard input device. The character entered is not displayed by the gectch() function.
Syntax of getche() is as follows:

getche();
The getche() accepts and displays the character whereas getch() accepts but does not display the character.

4.

putch();
This function prints any alphanumeric character taken by the standard input device.

1.
String I/O

1.

gets ();
This function is used for accepting any string through stdin (keyboard) until enter key is pressed. The header file stdio.h is needed for implementing the above function. Format of gets() is as follows:

char str[length of string in number];
gets(str);

A program is given on gets().

➢ 4.23 Write a program to accept string through the keyboard using the gets() function.

```
void main()

{

char ch[30];

clrscr();

printf("Enter the String :");

n Entered String : %s", ch);

}
```

OUTPUT:

Enter the String : USE OF GETS()

2.
puts();
This function prints the string or character array. It is opposite to gets().

char str[length of string in number];
gets(str);
puts(str);

A program is given on puts().

➢ 4.24 Write a program to print the accepted character using puts() function.

Explanation:

This program is the same as the previous one. Here, to display the string puts() function is used.

```
void main()

{

char ch[30];

clrscr();

printf("Enter the String :");

gets(ch);

puts("Entered String :");

puts(ch);

}
```

OUTPUT:

Enter the String: puts is in use.

Entered String:

puts is in use.

3.

cgets();

This function reads string from the console. The syntax is as follows.

***Syntax*:**

```
cgets(char *st);
```

It requires character pointer as an argument. The string begins from st[2].

4.

cputs() :

This function displays string on the console. The syntax is as follows.

***Syntax*:**

```
cputs(char *st);
clrscr();
```

COMMONLY USED LIBRARY FUNCTIONS

1.

clrscr();

This function is used to clear the screen. It clears previous output from the screen and displays the output of the current program from the first line of the screen. It is defined in conio.h header file. The syntax is as follows.

***Syntax*:**

clrscr();

2.

exit();

This function terminates the program. It is defined in process.h header file. The syntax is as follows.

***Syntax*:**

exit();

3.

sleep();

This function pauses the execution of the program for a given number of seconds. The number of seconds is to be enclosed between parentheses. It is defined in dos.h header file. The syntax is as follows.

***Syntax*:**

sleep(1);

An example on sleep() is given below.

➢ 4.26 Write a program to show the effect of the sleep() function.

```
void main()
{
static char t[10];
clrscr();
```

```
printf("\n Enter Text Here :");

gets(t);

printf("\n Your Entered Text :");

sleep(5);

puts(t);

getche();

}
```

OUTPUT:

Enter Text Here: ashok

Your Entered Text: ashok

Explanation:

The explanation is straightforward and self-explanatory. See the effect of sleep(5). The display appears after taking a pause.

3.

system ();

This function is helpful in executing different DOS commands. It returns 0 on success and –1 on failure. The syntax is as follows.

Syntax:

```
system ("dir");
```

The command should be enclosed within double quotation marks. After we run this command using C, directory will be displayed. Programmer can verify this command.

Decision-making statements in a programming language help the programmer to transfer the control from one part to other parts of the program. Thus, these decision-making statements facilitate the programmer in determining the flow of control. This involves a decision-making condition to see whether a particular condition is satisfied or not. On the basis of real time applications it is essential:

1. to alter the flow of a program.
2. to test the logical conditions.
3. to control the flow of execution as per the selection.

These conditions can be placed in the program using decision-making statements. C language supports the control statements as listed below:

1. The if statement
2. The if–else statement
3. The if–else–if ladder statement
4. The switch case statement
5. The goto unconditional jump
6. The loop statement

Besides, the C also supports other control statements such as continue, break.

The decision-making statement checks the given condition and then executes its sub-block. The decision statement decides which statement to execute after the success or failure of a given condition.

The relational operators are useful for comparing the two values. They can be used to check whether they are equal to each other, unequal or one is smaller/greater than the other.

The reader or the programmer is supposed to understand the concepts as cited above. Following points are expected to be known to the programmer related to the decision-making statements.

Sequential execution: The statements in the program are executed one after another in a sequential manner. This is called the sequential execution.

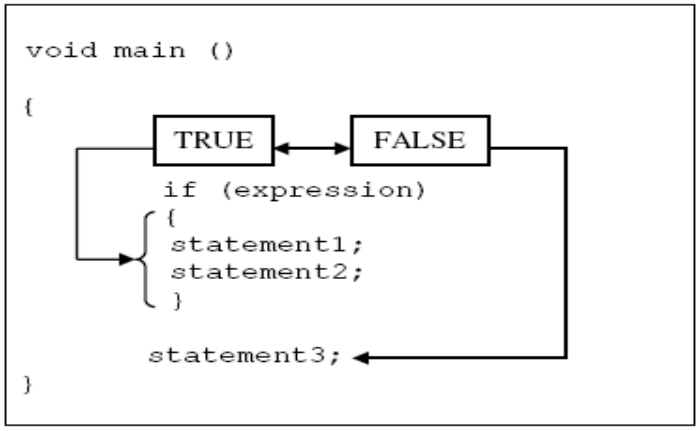
Transfer of control: The C statements such as if, goto, break, continue, switch allow a program to transfer the control at different places in the program. This is accomplished by skipping one or more statements appearing in a sequential flow. This jumping of control is called the transfer of control.

THE IF STATEMENT

C uses the keyword if to execute a set of command lines or one command line when the logical condition is true. It has only one option. The sets of command lines are executed only when the logical condition is true .

Syntax for the simplest if statement :-

```
if (condition) /* no semi-colon
*/ statement;
```



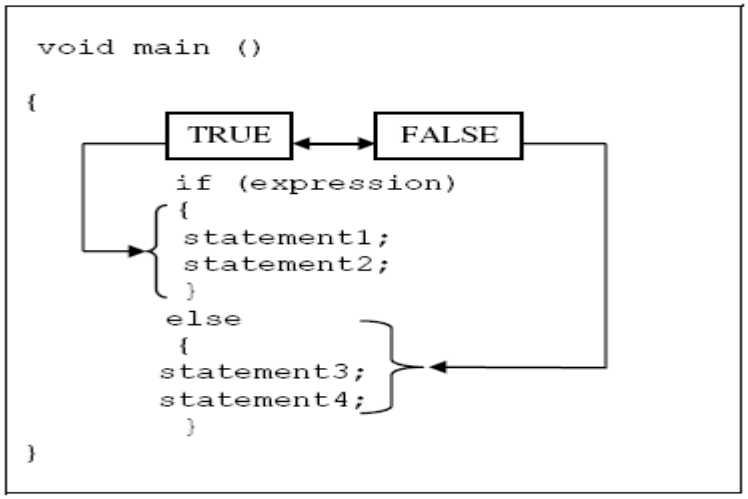
The **if** *statement*

The if statement contains an expression which is evaluated. If the expression is true it returns 1, otherwise 0. The statement is executed when the condition is true. In case the condition is false, the compiler skips the lines within the if block. The condition is always enclosed within a pair of parentheses. The conditional statements should not be terminated with semi-colons (;). The statements following the if statement are normally enclosed within curly braces. The curly braces indicate the scope of the if statement. The default scope is one statement. But it is good practice to use curly braces even if a single statement is used following the if condition.

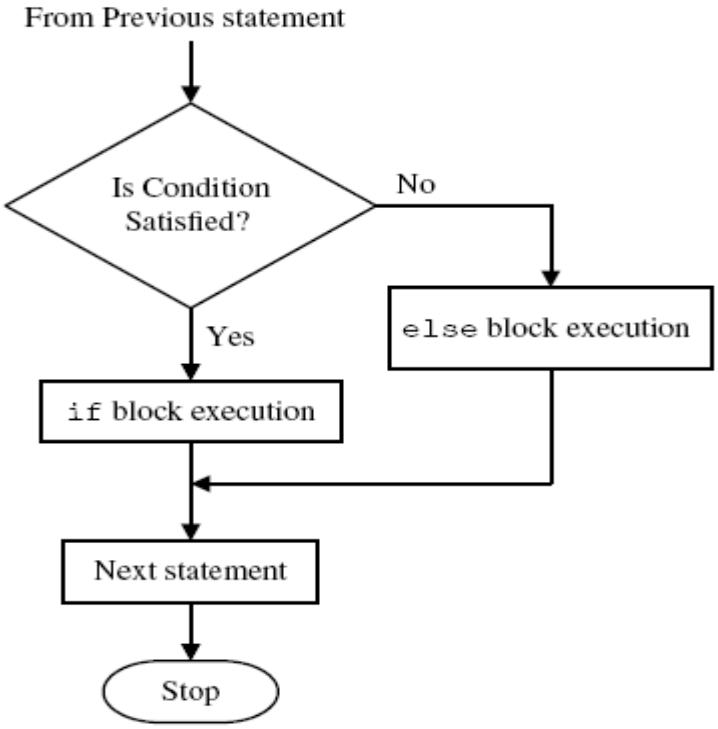
THE IF–ELSE STATEMENT

the if block statements execute only when the condition in if is true. When the condition is false, program control executes the next statement which appears after the if statement.

The if–else statement takes care of the true and false conditions. It has two blocks. One block is for if and it is executed when the condition is true. The other block is of else and it is executed when the condition is false. The else statement cannot be used without if. No multiple else statements are allowed with one if



The **if–else** *statements*

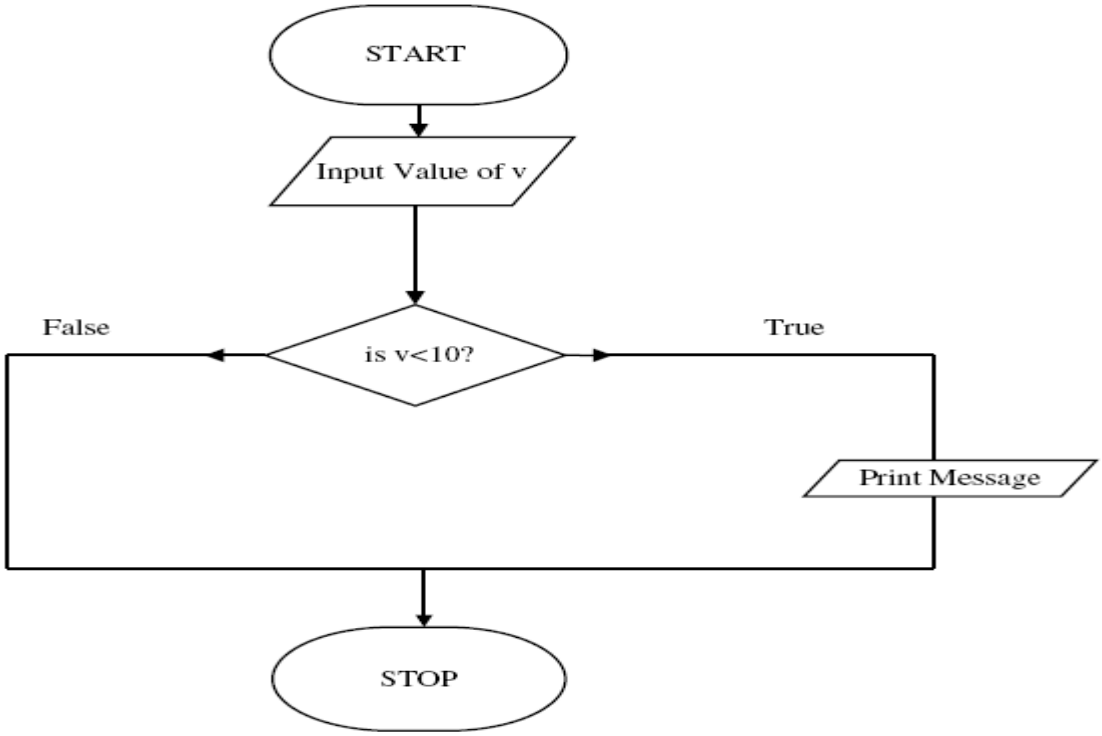


The **if–else** *statement*

The syntax of if–else statement is as follows:
if(the condition is true)
execute the Statement1;
else
execute the Statement2;
OR

Syntax of if–else statement can be given as follows:

```
if ( expression is true)
{
statement1;     /* if block */
statement2;
}
else
{
statement3;     /* else block */
statement4;
;
}
```



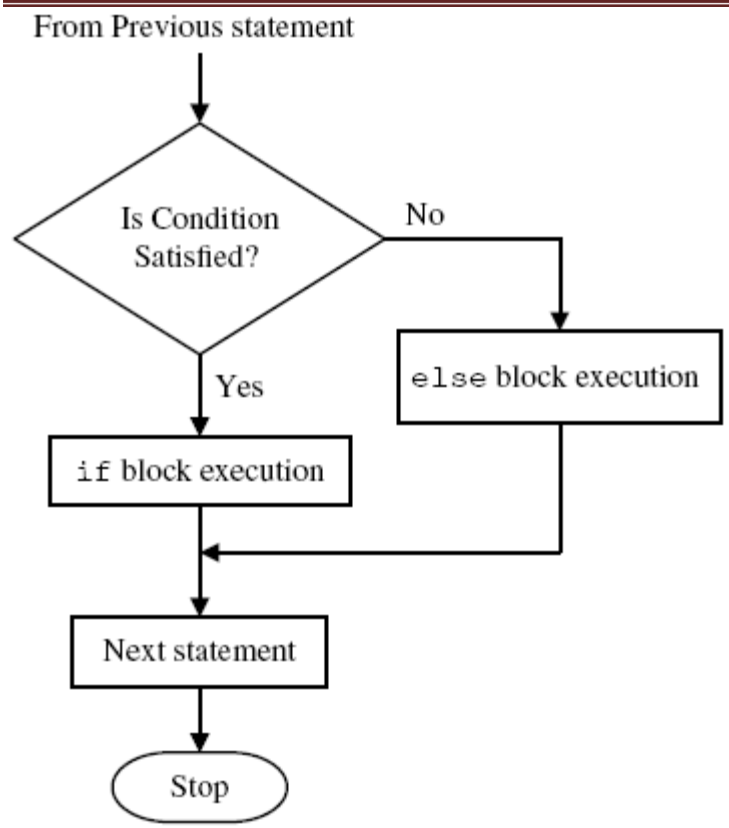
The **if** *statement (flow of control)*

5.

```
void main ()

{
        if (expression)
        {
        statement1;
        statement2;
        }
        else
        {
        statement3;
        statement4;
        }
}
```

TRUE FALSE

The **if–else** *statements*



The **if–else** *statement*

The syntax of if–else statement is as follows:

if(the condition is true)

execute the Statement1;

else

execute the Statement2;

OR

Syntax of if–else statement can be given as follows:

if (expression is true)

{

statement1; /* if block */

statement2;

}

else

```
{
statement3;     /* else block */
statement4;
}
```

The if–else statement is demonstrated in the following programs.

- Write a program to calculate the square of those numbers only whose least significant digit is 5.

```
 void main()
{
int s,d;
clrscr();
printf("\n Enter a Number :");
scanf("%d",&s);
d=s%10;
if(d==5)
{
  s=s/10;
  printf("\n Square = %d%d",s*s++,d*d);
}
else
  printf("\n Invalid Number");
}
```

OUTPUT:

Enter a Number : 25

Square = 625

Explanation:

In the above program, a number whose square is to be computed is entered. With the modular division, operation the last digit is separated to confirm whether it is 5 or not. If yes, the body of the if loop is executed where 10 divides the entered number and a quotient is obtained. The quotient and its consecutive number are multiplied and displayed. Followed by this, a square of 5 is calculated and displayed. Care is taken in the printf() statement to display the two results: (i) multiplication of quotient and its consecutive number and (ii) square of 5 without space. Thus, the square of a number is displayed.

➢ Write a program to calculate the salary of a medical representative based on the sales. Bonus and incentive to be offered to him will be based on total sales. If the sale exceeds or equals to Rs.1,00,000, follow the particulars of Table 1, otherwise follow Table 2.

TABLE

Basic=Rs. 3000.

Hra=20% of basic.

Da=110% of basic.

Conveyance=Rs.500.

Incentive=10% of sales.

Bonus=Rs. 500.

2. TABLE

Basic=Rs. 3000.

Hra=20% of basic.

Da=110% of basic.

Conveyance=Rs.500.

Incentive=5% of sales.

Bonus=Rs. 200.

```
void main()
{
float bs,hra,da,cv,incentive,bonus,sale,ts;
clrscr();
printf("\n Enter Total Sales in Rs.:");
scanf("%f", &sale);
if(sale>=100000)
{
    bs=3000;
    hra=20 * bs/100;
    da=110 * bs/100;
    cv=500;
    incentive=sale*10/100;
    bonus=500;
}
    else
    {
       bs=3000;
       hra=20 * bs/100;
       da=110 * bs/100;
       cv=500;
       incentive=sale*5/100;
       bonus=200;
    }
```

```
ts=bs+hra+da+cv+incentive+bonus;

printf("\nTotal Sales : %.2f",sale);

printf("\nBasic Salary : %.2f"',bs)

printf("\nHra : %.2f",hra);

printf("\nDa : %.2f",da);

printf("\nConveyance : %.2f",cv);

printf("\nIncentive : %.2f",incentive);

printf("\nBonus : %.2f",bonus);

printf("\nGross Salary : %.2f",ts);

getch();

}
```

OUTPUT:

Enter Total Sales in Rs. 100000

Total Sales: 100000.00

Basic Salary: 3000.00

Hra: 600.00

Da: 3300.00

Conveyance: 500.00

Incentive: 10000.00

Bonus: 500.00

Gross Salary: 17900.00

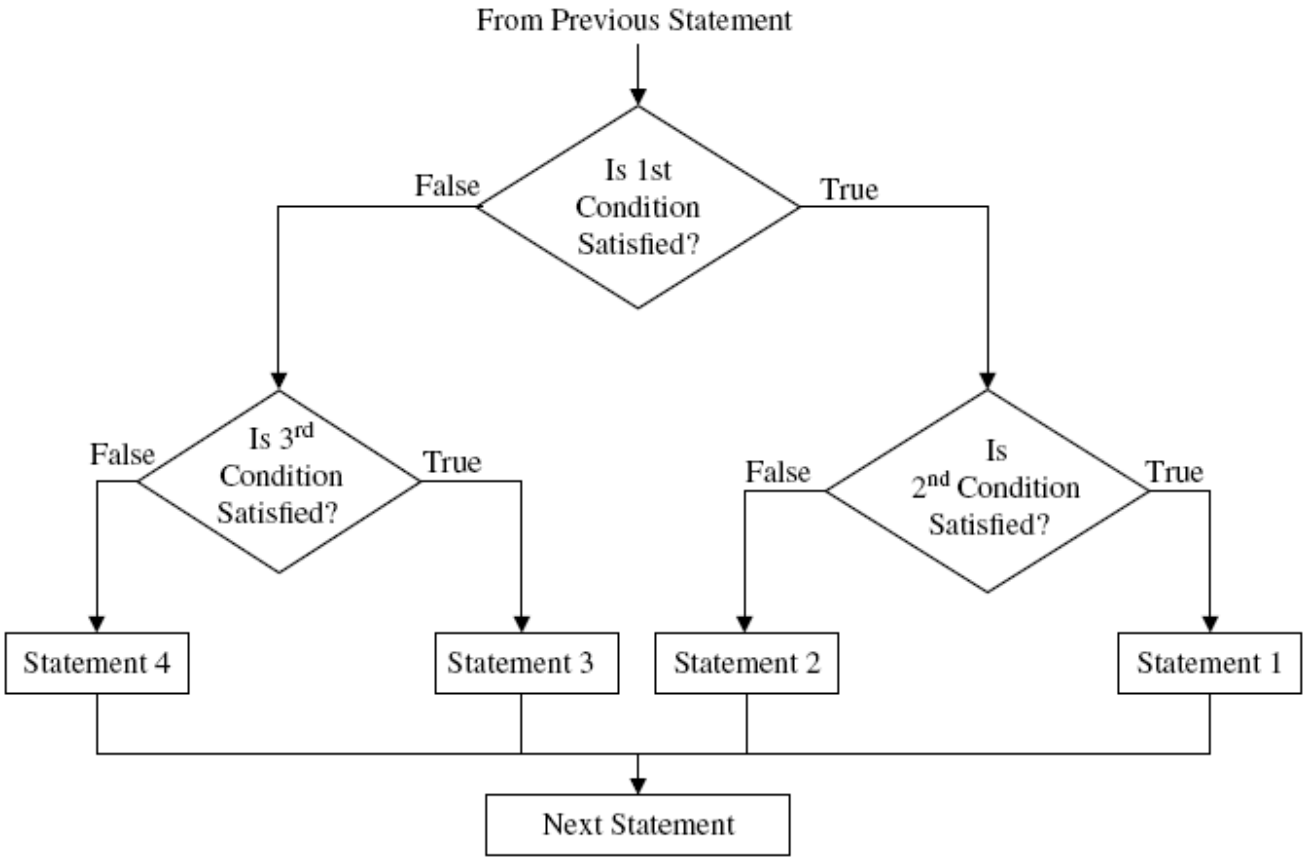
Explanation:

This program calculates the salary of a medical representative depending on sales. The basic salary is the same but other allowances and incentives change, depending on the sales. If the sale is more than Rs. 1,00,000, the rate of allowances and incentive is as per Table 1 otherwise it is as per Table 2.

The if condition checks the given figure of sales. If sale is more than Rs. 1,00,000, the first bock following the if statement is executed otherwise the else block is executed. In both the blocks, simple arithmetic operations are performed to calculate the allowances and total salary.

NESTED IF–ELSE STATEMENTS

In this kind of statement, a number of logical conditions are checked for executing various statements. Here, if any logical condition is true the compiler executes the block followed by if condition, other wise it skips and executes the else block. In the if–else statement, the else block is executed by default after failure of condition. In order to execute the else block depending upon certain condition we can add, repetitively, if statements in else block. This kind of nesting will be unlimited. Figure 5.5 describes the nested if–else–if blocks.



Nested **if–else** *statements*

Syntax of nested if-else statement can be given as follows.

```
 if(condition)
{
/*Inside first if block*/
if(condition)
{
```

```
  statement 1; /*if block*/
  statement 2;
}
else
{
  statement 3; /*else block*/
  statement 4;
}
}
else
{
  /*Inside else block*/
  if(condition)
  {
    statement 5; /*if block*/
    statement 6;
  }
else
  {
    statement 7; /*else block*/
    statement 8;
  }
}
```

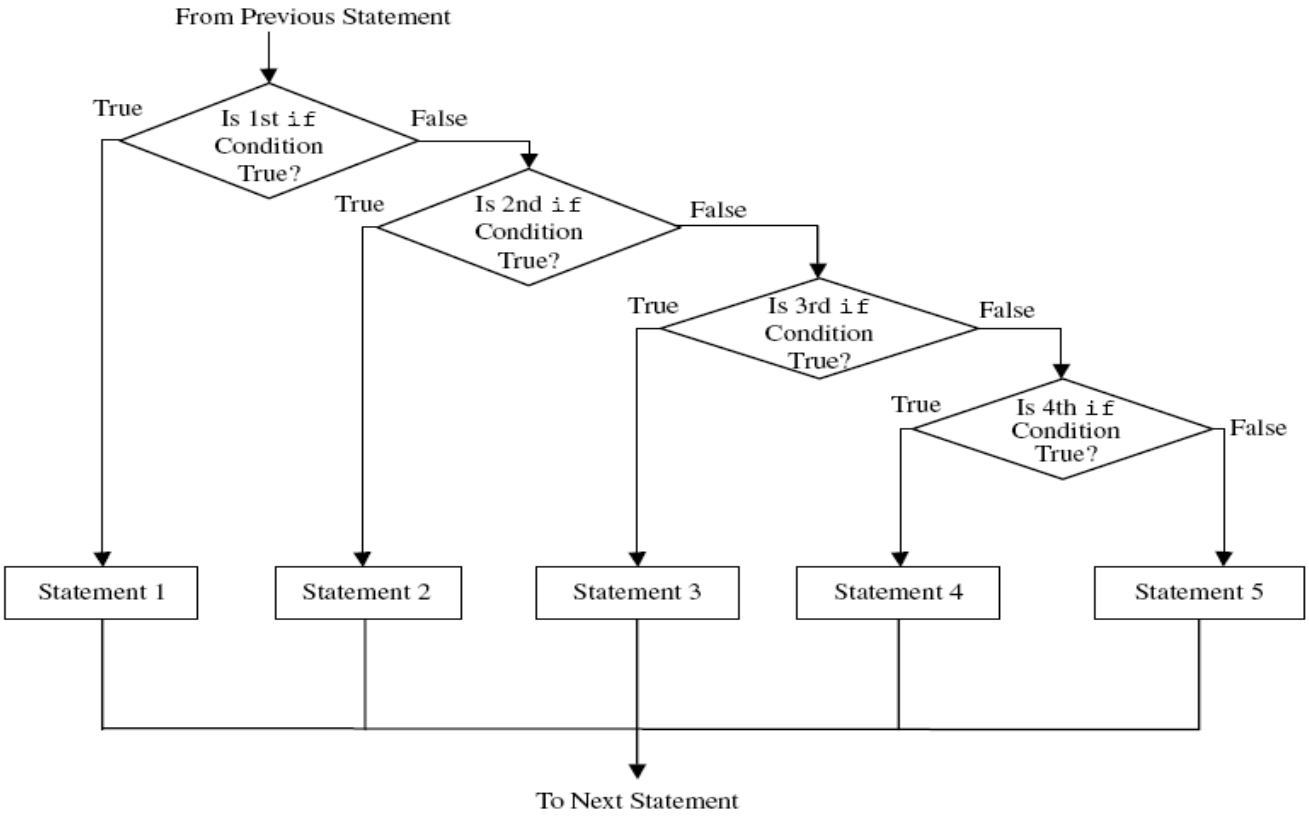
From the above block, following rules can be described for applying nested if-else-if statements.

1. Nested if-else can be chained with one another.
2. If the condition is true control passes to the block following first if. In that case, we may have one more if statement whose condition is again checked. This process continues till there is no if statement in the last if block.
3. If the condition is false control passes to else block. In that case, we may have one more if statement whose condition is again checked. This process continues till there is no if statement in the last else block.

THE IF-ELSE-IF LADDER STATEMENT

In this kind of statement, a number of logical conditions are checked for executing various statements. Here, if the first logical condition is true the compiler executes the block followed by first if condition, otherwise it skips that block and checks for next logical condition followed by else-if, if the condition is true the block of statements followed by that if condition is executed. The process is continued until a true condition is occurred or an else block is occurred. If all if conditions become false, it executes the else block. In the if-else-if ladder statement, the else block may or may not have the else block.

In if-else-if ladder statement we do not have to pair if statements with the else statements that is we do not have to remember the number of braces opened like nested if-else. So it is simpler to code than nested if-else and having same effect as nested if-else.



if–else–if *ladder statement*

Syntax of if-else-if statement can be given as follows.

```
 if(condition)
{
statement 1; /*if block*/
statement 2;
}
else if(condition)
{
statement 3; /*second if block*/
statement 4;
```

```
}

else if(condition)

{

statement 5; /*third if block*/

statement 6;

}

else

{

statement 7; /*else block*/

statement 8;

}
```

From the above block, following rules can be described for applying nested if-else-if statements:

1. Nested if-else can be chained with one another.
2. If the first if condition is false control passes to else-if block where condition is again checked with the next if statement. This process continues till there is no if statement in the last else block.
3. If one of the if statements satisfies the condition, other nested else-if statement will not be executed.

Given below programs are described on the bases on nested if-else and if-else-if ladder statements.

THE BREAK STATEMENT

The keyword break allows the programmers to terminate the loop. The break skips from the loop or block in which it is defined. The control then automatically goes to the first statement after the loop or block. The break can be associated with all conditional statements.

We can also use the break statements in the nested loops. If we use the break statement in the innermost loop, then the control of the program is terminated only from the innermost loop.

The difference between the break and exit() is provided in Table 5.1.

Difference between **break** *and* **exit()**

Sr. No	**break**	**exit()**
1.	It is a keyword.	It is a function.
2.	No header file is needed.	Header file process.h must be included.
3.	It stops the execution of the loop.	It terminates the program.

5.7 THE CONTINUE STATEMENT

The continue statement is exactly opposite of the break statement.
The continue statement is used for continuing the next iteration of the loop statements. When it occurs in the loop, it dose not terminate, but it skips the statements after this statement. It is useful when we want to continue the program without executing any part of the program. Table 5.2 gives the differences between break and continue.

Difference between **break** *and* **continue**

break	**continue**
Exits from current block or loop.	Loop takes the next iteration.
Control passes to the next statement.	Control passes at the beginning of the loop.
Terminates the loop.	Never terminates the program.

THE GOTO STATEMENT

This statement does not require any condition. This is an unconditional control jump. This statement passes control anywhere in the program, i.e. control is transferred to another part of the program without testing any condition. User has to define the goto statement as follows:

goto label;

where, the label name must start with any character.

Here, the label is the position where the control is to be transferred. A few examples are described for the sake of understanding.

THE SWITCH STATEMENT

The switch statement is a multi-way branch statement. In the program, if there is a possibility to make a choice from a number of options, this structured selection is useful. The switch statement requires only one argument of any data type, which is checked with the number of case options. The switch statement evaluates expression and then looks for its value among the case constants. If the value matches with case constant, this particular case statement is executed. If not, default is executed. Here, switch, case and default are reserved keywords. Every case statement terminates with ':'. The break statement is used to exit from current case structure. The switch statement is useful for writing the menu-driven program.

The syntax of the switch case statement is as follows.

```
switch(variable or expression)
{
case constant A :
statement;
break;
case constant B :
statement;
break;
default :
statement ;
}
```

1. The switch expression

In the block, the variable or expression can be a character or an integer. The integer expression following the keyword switch will yield an integer value only. The integer may be any value 1, 2, 3, and so on. In case a character constant, the values may be given in the alphabetic order such as 'x', 'y', 'z'.

2. The switch organization

The switch expression should not be terminated with a semi-colon and/or with any other symbol. The entire case structure following switch should be enclosed with curly braces. The keyword case is followed by a constant. Every constant terminates with a colon. Each case statement must contain different constant values. Any number of case statements can be provided. If the case structure contains multiple statements, they need not be enclosed with curly braces. Here, the keywords case and break perform the job of opening and closing curly braces, respectively.

3. The switch execution

When one of the cases satisfies, the statements following it are executed. In case, there is no match, the default case is executed. The default can be put anywhere in the switch expression. The switch statement can also be written without the default statement. The break statement used in switch causes the control to go outside the switch block. By mistake, if no break statements are given all the cases following it are executed

NESTED SWITCH CASE

The C supports nested switch statements. The inner switch can be a part of an outer switch. The inner and outer switch case constants may be the same. No conflicts arise even if they are the same. A few examples are given below on the basis of nested switch statements.

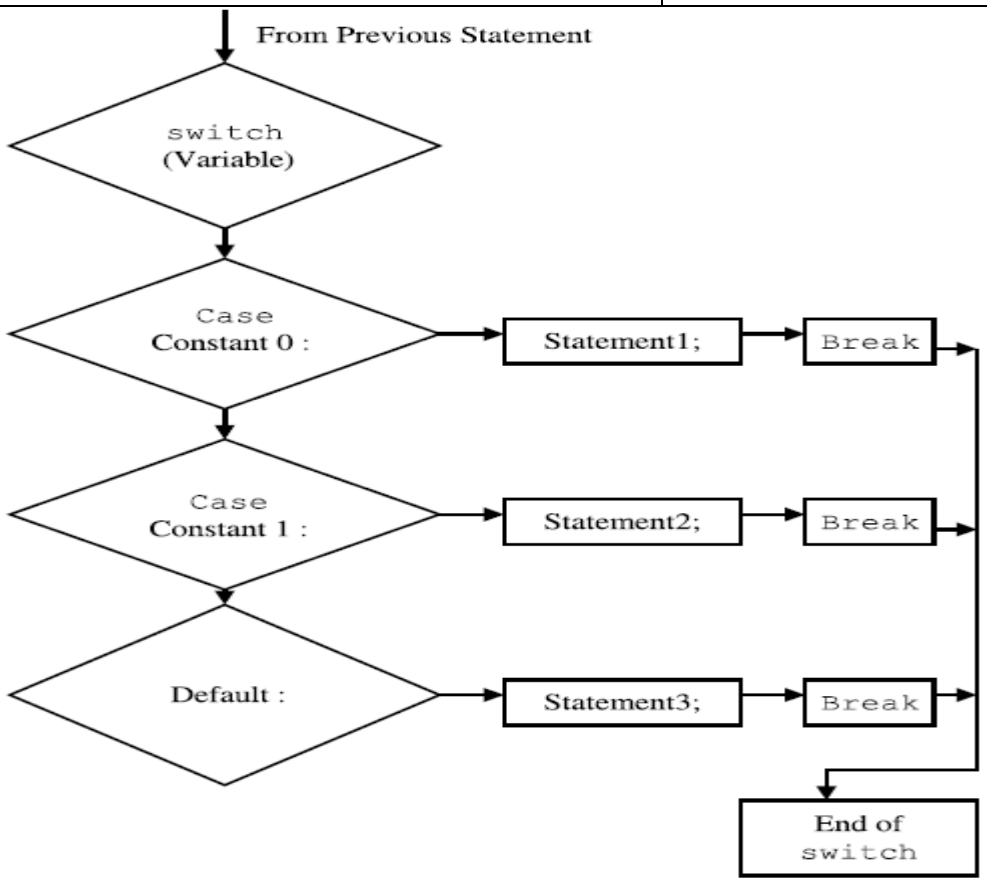
THE SWITCH CASE AND NESTED IFS

The distinction between the switch case and the nested ifs is narrated in Table 5.3.

Table 5.3 *Distinction between the* switch case *and* **nested** if**s**

switch case	**nested ifs**
The switch can only test for equality, i.e. only constant values are applicable.	The if can evaluate relational or logical expressions.

switch case	nested ifs
No two case statements have identical constants in the same switch.	Same conditions may be repeated for the number of times.
Character constants are automatically converted to integers.	Character constants are automatically converted to integers.
In switch case statement nested if can be used.	In nested if statement switch case can be used.



switch *statement*

Explanation:

In this program, a menu appears with five options and it requests the users to enter their choice. The choice entered by the user is then passed to switch statement. In the switch statement, the value is checked with all the case constants. The matched case statement is executed in which the line is printed of the user's choice. If the user enters a non-listed value, then no match occurs

and default is executed. The default warns the user with a mes-sage 'Invalid Choice'.

What is a Loop?

A loop is defined as a block of statements, which are repeatedly executed for a certain number of times.

The loops are of two types.

1. *Counter-controlled repetition*: This is also called the definite repetition action, because the number of iterations to be performed is defined in advance in the program itself. The steps for performing counter-controlled repetitions are as follows.

Steps in Loop

Loop variable: It is a variable used in the loop.

Initialization: It is the first step in which starting and final values are assigned to the loop variable. Each time the updated value is checked by the loop itself.

Incrimination/decrimination: It is the numerical value added or subtracted to the variable in each round of the loop. The updated value is compared with the final value and if it is found less than the final value the steps in the loop are executed.

The above steps are implemented in numerous programs in this chapter.

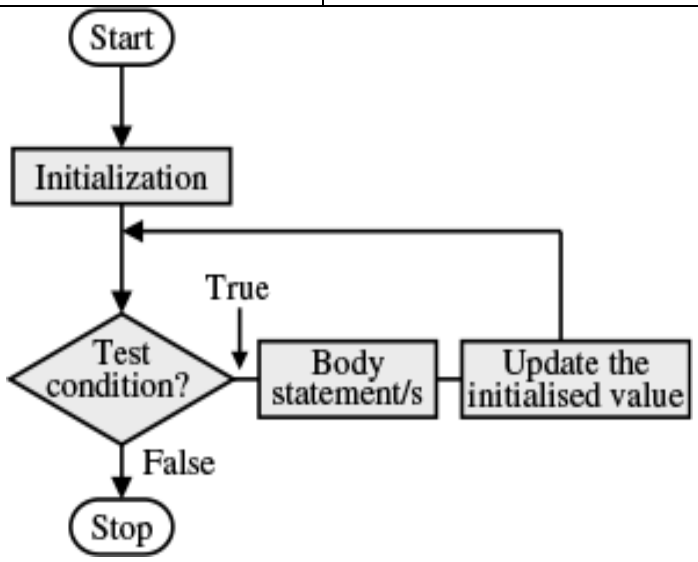
2. *Sentinel-controlled repetition* : This is also called the indefinite repetition. One cannot estimate how many iterations are to be performed. In this type, loop termination happens on the basis of certain conditions using the decision-making statement.

The C language supports three types of loop control statements and their syntaxes are described in Table 6.1 (also see Figures 6.1 and 6.2).

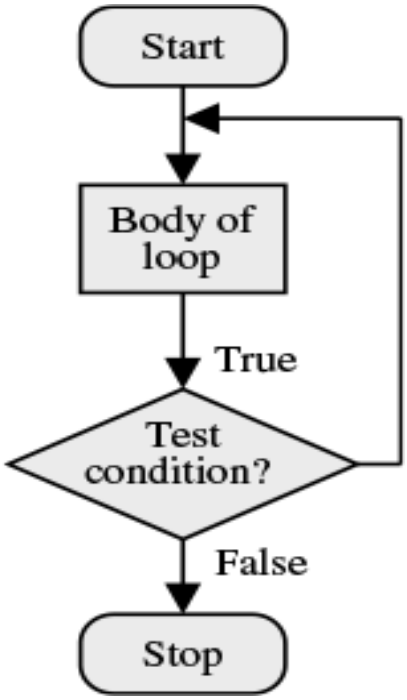
Loops in C

for	*while*	*do-while*

for(expression −1;
expression−2;
expression−3)
statement;
	expression −1;
	while(expression −2)
	{
	statement;
	expression −3;
	
	}
		Expression −1; do
		{
		statement;
		expression−3;
		}
		
		while(expression−2);



The while statement



The do-while statement

The for loop statement comprises three actions. These actions are placed in the for statement itself. The three actions are initialize counter, test condition and Re-evaluation parameters, which are included in one statement. The expressions are separated by semi-colons (;). This leads the programmer to visualize the parameters easily. The for statement is equivalent to while and do-while statements. The only difference between for and while is that in the latter case logical condition is tested and then the body of the loop gets executed. However, in the for statement, test is always performed at the beginning of the loop. The body of the loop may not be executed at all times if the condition fails at the beginning.

```
for (a=10;a<10;a−−)

printf(“%d”, a);
```

For example, in the above two-line program will never execute because the test condition is not proper at the beginning, hence statement following to for loop does not execute.

The do-while loop executes the body of the loop at least once regardless of the logical condition.

THE FOR LOOP

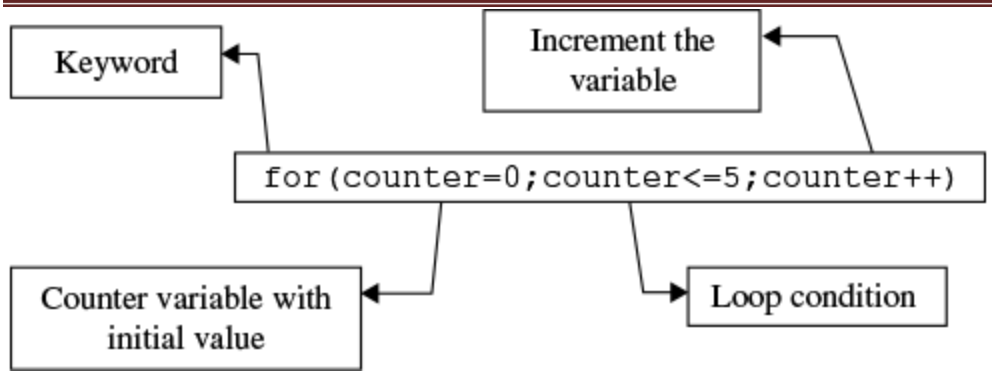
The for loop allows to execute a set of instructions until a certain condition is satisfied. Condition may be predefined or open-ended. The general syntax of the for loop will be as given in Table 6.2 (see also Figures 6.3 and 6.4).

Explanation:

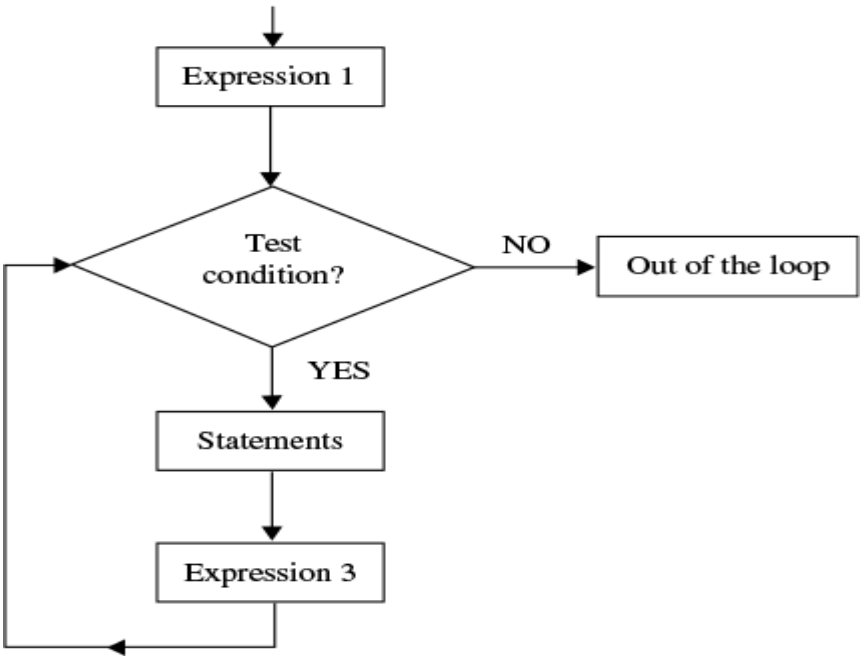
The for statement contains three expressions which are separated by semi-colons. Following actions are to be performed in the three expressions.

Syntax of for loop

```
for(initialize counter; test condition; re-evaluation parameter)
{
   statement1;
   statement2;
}
```



The for loop



Execution of the for loop

1. The initialize counter sets to an initial value. This statement is executed only once.
2. The test condition is a relational expression, that determines the number of iterations desired or determines when to exit from the loop. The 'for' loop continues to execute as long as conditional test is satisfied. When the condition becomes false the control of the program exits from the body of the 'for' loop and executes next statement after the body of the loop.
3. The re-evaluation parameter decides how to make changes in the loop (increment or decrement operations are to be used quite often). The body of the loop may contain either a single statement or multiple statements. In case, there is only one statement after the for loop, braces may not be necessary. In such a case, only one statement is executed till the condition is satisfied. It is good practice to use braces even for single statement following the for loop.

The syntax of the 'for' loop with only one statement is shown in the third row of Table 6.3.

The for loop can be specified by different ways and it is as per Table 6.3.

Various formats of 'for' Loop

Syntax	***Output***	***Remarks***
for(; ;)	Infinite loop	No arguments
for(a=0; a<=20;)	Infinite loop	'a' is neither incremented nor decremented.
for(a =0;a<=10; a++) printf("%d", a);	Displays value from 0 to 10	'a' is incremented from 0 to 10. Curly braces are not necessary. Default scope of the for loop is one statement after the for loop.
for(a =10;a>=0;a−−) printf("%d", a)	Displays value from 10 to 0	'a' is decremented from 10 to 0.

NESTED FOR LOOPS

We can also use loop within loops. In nested for loops one or more for statements are included in the body of the loop. In other words C allows multiple for loops in nested forms. The numbers of iterations in this type of structure will be equal to the number of iteration in the outer loop multiplied by the number of iterations in the inner loop.

THE WHILE LOOP

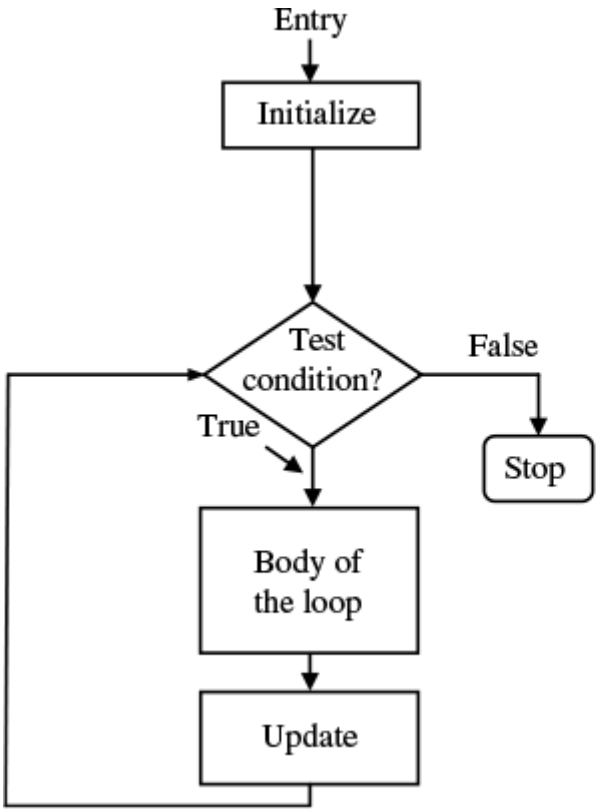
Another kind of loop structure in C is the while loop. The while loop is frequently used in programs for the repeated execution of statement/s in a loop. Until a certain condition is satisfied the loop statements are executed.

The syntax of while loop is

```
 while(test condition)

{

body of the loop

}
```

The test condition is indicated at the top and it tests the value of the expression before processing the body of the loop. The test condition may be any expression. The loop statements will be executed till the condition is true, i.e. the test condition is evaluated and if the condition is true, then the body of the loop is executed. When the condition becomes false the execution will be out of the loop.

The execution of the loop can be followed by the following flow chart given in Figure 6.5.



Steps of while loops areas follows.

(1) The test condition is evaluated and if it is true, the body of the loop is executed.

(2) On execution of the body, test condition is repetitively checked and if it is true the body is executed.

(3) The process of execution of the body will be continued till the test condition becomes true.

(4) The control is transferred out of the loop if test condition fails.

The flow chart showing the execution of the loop

Here, the block of the loop may contain either a single statement or a number of statements. The same block can be repeated.

The braces are needed only if, the body of the loop contains more than one statement. However, it is good practice to use braces even if the body of the loop contains only one statement.

THE DO-WHILE LOOP

The syntax of do-while loop in C is as follows.

```
do
{
```

statement/s;

}

while(condition);

The difference between the while and do-while loop is the place where the condition is to be tested. In the while loops the condition is tested following the while statement, and then the body gets executed, whereas in the do-while the condition is checked at the end of the loop. The do-while loop will execute at least one time even if the condition is false initially. The do-while loop executes until the condition becomes false. The comparison between the while and do-while loop is given in Table

Some programs are given on do-while loop.

Comparison of the while and do-while loop

Sr. No.	***while Loop***	***do-while Loop***
1	Condition is specified at the top.	Condition is mentioned at the bottom.
2	Body statement/s is/are executed when condition is satisfied.	Body statement/s executes even when the condition is false.
3	No brackets for a single statement.	Brackets are essential even when a single statement exits.
4	It is an entry-controlled loop.	It is an exit-controlled loop.

HE WHILE LOOP WITHIN THE DO-WHILE LOOP

The syntax of do-while with multiple while statement loop is as follows:

do-while(condition)

{

```
statement/s;

}

while(condition);
```

C continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
1. //loop statements
2. continue;
3. //some lines of the code which is to be skipped
```

C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

```
1. //loop or switch case
2. break;
```

Flowchart of break in c

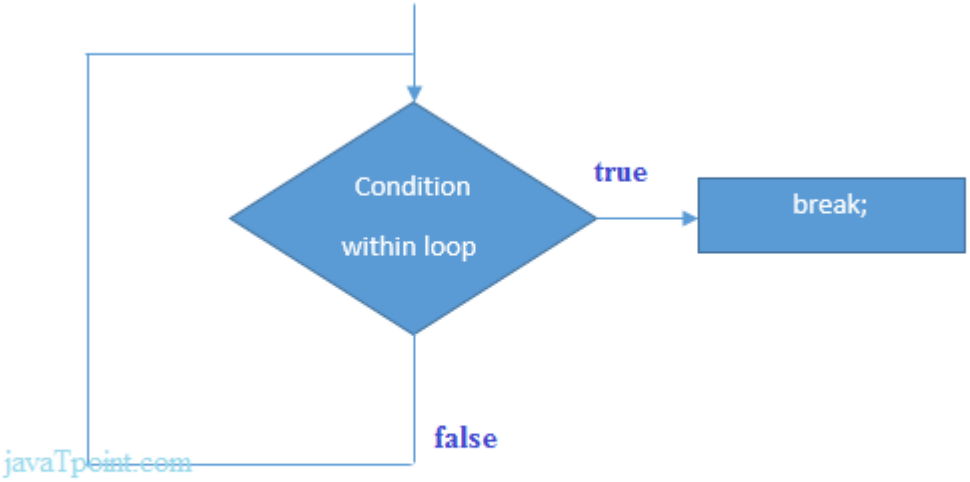


Figure: Flowchart of break statement

C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

```
1. //loop or switch case
2. break;
```

Flowchart of break in c

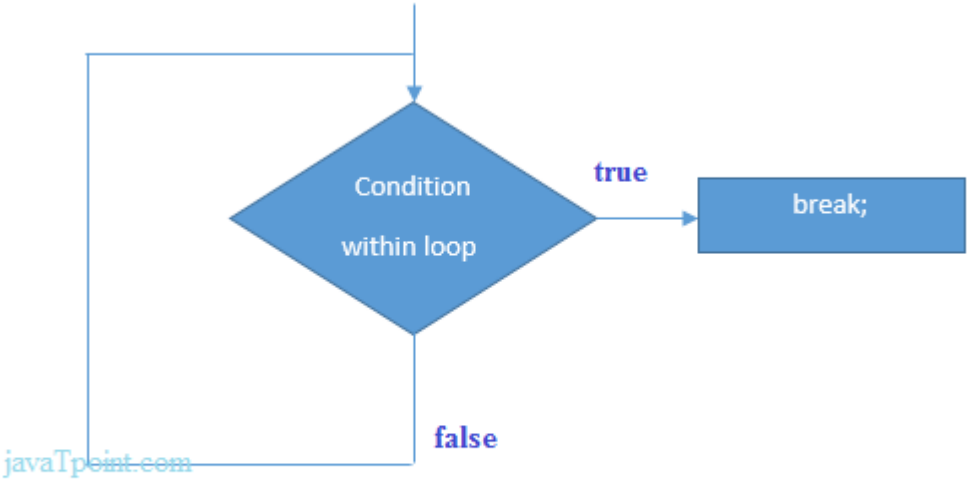


Figure: Flowchart of break statement

4. DATA STRUCTURE : ARRAY

An array is a linear and homogeneous data structure. Similar type of elements are stored continuously in memory and that too under one variable name.

Thus for example: int num[6]; This is the declaration of an integer array of

six elements. The array could be initialized as:

int num[6] = {2, 4, 6, 8, 10, 12}; The individual elements could be initialized:

Example: num[0] = 2; num[1] = 4; etc.

Array Terminology

Size: The number of elements or the capacity to store elements in an array is called its size. It is always written in brackets '[]'.

Type: Type refer to data type. It decides which type of element is stored in the array. It instructs the compiler to reserve memory according to the data

type.

Base: The address of the first element (0th) is a base address. The array

name itself stores address of the first element.

Index: The array name is used to refer to the array element. For example, in num[x], num is array name and x is the index. The value of x begins at 0

and goes up to size - 1.

Range: Index of an array varies from lower bound to upper bound while

writing or reading elements from an array. For example in num[100] the range of index is 0 to 99.

1) One-Dimensional Array

Array elements are stored in sequence one after another. The elements of an array are aranged in one-dimension. They can be shown in a row or column. Single subscript is used in one-dimensional arrays to represent its elements.

Example: int num[5]; An array is initialized. The type of variable is integer. Its variable name is num and the size of the array is 5. The elements of the integer array num[5] are stored in contiguous memory locations. Each integer element requires 2 bytes. Similarly, the elements of arrays of any data type are stored in contiguous memory location. The only difference is that the number of memory locations is different for different data types.

Character arrays are called strings. There is a slight difference between integer arrays and character arrays. In character array a NULL character ('\0') is automatically added at the end. The NULL character acts as the end of the string. Other type of arrays do not have the NULL character at the end.

Initialization of One-Dimensional Arrays

After an array is declared, its elements must be initialized. Otherwise, they will contain "garbage". An array can be initialized at compile time or at run time.

a. Compile Time Initialization

We can initialize the elements of arrays in the same way as the ordinary variables, when they are declared. The general form of initialization of arrays is:

type array-name[size] = {list of values };

The values in the list are separated by commas. For example, the statement

int num[3] = { 0, 0, 0 };

will declare the variable num as an array of size 3 and will assign zero to each element. If the number of values in the list is less than the number of elements, then only that many elements are initialized.

The remaining elements will be set to zero automatically.

For example,

Int total[5] = {100,150, 75}; will initialize the first three elements to 100, 150, 75 and the remaining two elements to zero.

The size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. For example,

int counter[] = {1, 1, 1, 1}; will declare the counter array to contain 4 elements

with initial values 1. This approach works fine as long as we initialize every

element in the array.

Character arrays may be initialized in a similar manner.

Thus, char name[ ] = {'J', 'o', 'h', 'n','\0'}; declares the name to be an array of five characters, initialized with the string "John" ending with the null character. Alternatively, we can assign the string literal directly as: char name [ ] = "John"; Compile time initialization may be partial. That is, the number of initializers may be less than the declared size. In such cases, the remaining elements are initialized to zero, if the array type is numeric and NULL if the type is char.

b. Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually

applied for initializing large arrays.

One-Dimensional Array Operations

Traversing is the operation of accessing or going across the elements of an array.
Program to read and display the elements of an array.

```
void main()
{
int i, num[10];
printf("\n\t Enter ten numbers : ");
for(i=0; i&lt;10; i++)
scanf("%d", &amp;num[i]);
printf("\n\t Elements Address");
for (i=0; i&lt;10; i++)
printf("\n\t%d %u", num[i], &amp;num[i] );
}
```

1. Operations with Arrays

The operations performed with arrays are: Deletion, Insertion, Searching,

Merging and Sorting.

Searching: The process of seeking specific elements in an array is called

searching.

Program to search an element in an array.

```
void main()
{ int i, n, num[12] = {11, 22, 33, 44, 22, 55, 66, 33, 22, 77, 88, 55};
printf("\n\t The Array Elements are:\n\t");
for(i=0; i&lt;12; i++)
printf("%d, ", num[i]);
printf("\n\n\t Enter the element to be searched : ");
scanf("%d", &amp;n);
for(i=0; i&lt;12; i++)
if(num[i]==n)
printf("\n\t%d is found at position %d.", n, i+1);
}
```

2. Deletion: This operation is the deletion of specified elements from an array.

Program to delete an element from an array.

```
void main()
{
int num[20] = {0}, i, n, p;
printf("\n\n\t Enter the number of elements : ");
scanf("%d", &amp;n);
printf("\n\t Enter the Elements : ");
for(i=0; i&lt;n; i++)
scanf("%d", &amp;num[i]);
printf("\n\t The Array Elements are:\n");
for(i=0; i&lt;n; i++)
printf("\n\n\t%d. %d %u", i+1, num[i], &amp;num[i]);
printf("\n\t Enter the position of the element to be deleted : ");
scanf("%d", &amp;p);
```

```
p--;
for(i=0; i&lt;n; i++)
if(i&gt;=p)
num[i] = num[i+1];

printf("\n\t The Elements after Deletion:");
for(i=0; i&lt;n; i++)
if(num[i]!=0)
printf("\n\n\t%d %u", num[i], &amp;num[i]);

}
```

1. Program to delete an element in an array by entering the element.
2. Program to delete the first element.
3. Program to delete the element.

3. Insertion: This operation is used to insert an element at a specified position in an array.

Program to insert an element in an array.

```
void main()
{
int i, n, el, p, num[20] = {0};
printf("\n\n\t Enter the number of elements : ");
scanf("%d", &amp;n);
printf("\n\t Enter the Elements : ");
for(i=0; i&lt;n; i++)
scanf("%d", &amp;num[i]);
printf("\n\t The Elements and their locations:");
for(i=0; i&lt;n; i++)
printf("\n\n\t%d. %d %u", i+1, num[i], &amp;num[i]);
```

```
printf("\n\t Enter element and the position to insert at : ");
scarf ("%d %d", &amp;amp;el, &amp;amp;p);
p--;
for(i=n; i!=p; i--)
num[i] = num[i-1];

num[i] = el;
printf("\n\t The Elements after Insertion:");
for (i=0; i&amp;lt;=n; i++)
printf("\n\n\t%d. %d %u", i+1, num[i], &amp;amp;num[i] );

}
```

1. Program to insert an element at the first position.
2. Program to insert an element at the last position.

3. Merging: The elements of two arrays can be merged into a single one. One method of merging two arrays is to copy all elements of one array into a third array. Then copy the elements of the second array into the third one. Alternatively, we can append elements of one array to end of the second array. The elements of two arrays can also be merged in alternate order.

Program to merge two arrays into a third array.

```
void main()
{
int i, j = 0, k = 0;
int num1[5] = {11, 22, 33, 44, 55 };
int num2[5] = {66, 77, 88, 99, 111};
int num3[10];
```

```
printf("\n\t The Elements of Array 1:\n\t");
for(i=0; i&lt;5; i++)
printf("%d, ", num1[i]);
printf("\n\n\t The Elements of Array 2:\n\t");
for(i=0; i&lt;5; i++)
printf("%d, ", num2[i]);
for(i=0; i&lt;10; i++) /* Merging the arrays */
if(i%2 == 0)
num3[i] = num1[j++];
else
num3[i] = num2[k++];

printf("\n\n\t The Elements of Array 3:\n\t");
for (i=0; i&lt;10; i++)
printf("%d, ", num3[i]);
}
```

Program to display the elements of an array using pointer.

```
void main()
{
int i, *ptr, num[8] = {11, 22, 33, 44, 55, 66, 77, 88};
ptr = num;
printf("\n\tThe Elements are:");
for(i=0; i&lt;8; i++)
printf("\n\n\t%d. %d at location %u", i+1, *(ptr+i), (ptr+i));
}
```

5. Sorting: Arranging the elements of an array either in ascending or in descending order is sorting.

Program to sort an array of integers in ascending order.

```
void main()
```

```
{
int num[20] = {0}, i, j, n, temp;
printf("\n\n\t Enter the number of values : ");
scanf("%d", &amp;n);
for(i=0; i&lt;n; i++)
{ printf("\n\t Enter numbrs # %d : ", i+1);
scanf("%d", &amp;num[i]);
}
printf("\n\tThe Numbers are:\n\t");
for(i=0; i&lt;n; i++)
printf("%5d", num[i]);
/* Sorting */
for (i=0; i&lt;n-1; i++)
{
for (j=i+1; j&lt;n; j++)
{
if(num[i] &gt; num[j])
{
temp = num[i];
num[i] = num[j];
num[j] = temp;
}
}
}
printf("\n\t The Numbers in ascending order:\n\t");
for (i=0; i&lt;n; i++)
printf("%5d", num[i]);
}
```

3) Two-Dimensional Array and Operations

Two-dimensional arrays can be thought of as a rectangular display of

elements with rows and columns. Consider the example, int x[3][3]; The two-dimensional array can be arranged into rows and columns. This arrangement of array elements is only for the sake of understanding. Actually the elements are stored in the contigious memory locations. The two-dimensional array is a collection of three one-dimensional arrays. The first argument in x[3][3] means the number of rows, i.e. the number of one-dimensional array and the second argument shows the number of elements. x[0][0] means the first element of the first row and column. In one row, the row number remains the same and the column numbers change. The number of rows and columns iscalled the range of an array. The computer memory is linear and all types ofarrays, one, two or multi-dimensional arrays are stored in the continuousmemory locations.

Like the one-dimensional arrays, two-dimensional arrays may be initialized

following their declaration with a list of initial values enclosed in braces .Theymay also be initialized at run time.

int table[2][3] = {0, 0, 0, 2, 2, 2};

int table[2][3] = {{0, 0, 0}, {2, 2, 2}};

Program to demonstrate the use of two-diamentional array.

```
void main()
{
int i, j, a[3][3] = {11, 22, 33, 44, 55, 66, 77, 88, 99};
printf("\n\t Array elements and address\n");
printf("\n\t\t Col-0 Col-1 Col-2");
printf ("\n\t\t ====== ====== ======");
for(i=0; i&lt;3; i++)
{
printf("\n\tRow%d ", i);
for(j=0; j&lt;3; j++)
printf("%d [%u]", a[i][j], &amp;a[i][j]);
}
}
```

Program to display the elements of a two-dimensional using a pointer.

```
void main()
{
```

```
int i, *p, num[3][3]={11, 22, 33, 44, 55, 66, 77, 88, 99);
p = &amp;num[0][0];
for(i=0; i&lt;9; i++)
printf("\n\t %d %u", *(p+i), unsigned(p+i));
}
```

argument shows the number of elements. x[0][0] means the first element of the first row and column. In one row, the row number remains the same andthe column numbers change. The number of rows and columns is called therange of an array. The computer memory is linear and all types of arrays, one,two or multi-dimensional arrays are stored in the continuous memorylocations.

Like the one-dimensional arrays, two-dimensional arrays may be initialized following their declaration with a list of initial values enclosed in braces .They may also be initialized at run time.

```
int table[2][3] = {0, 0, 0, 2, 2, 2};
int table[2][3] = {{0, 0, 0}, {2, 2, 2}};
```

Program to demonstrate the use of two-diamentional array.

```
void main()
{
int i, j, a[3][3] = {11, 22, 33, 44, 55, 66, 77, 88, 99};
printf("\n\t Array elements and address\n");
printf("\n\t\t Col-0 Col-1 Col-2");
printf ("\n\t\t ====== ====== ======");
for(i=0; i&lt;3; i++)
{
printf("\n\tRow%d ", i);
for(j=0; j&lt;3; j++)
printf("%d [%u]", a[i][j], &amp;a[i][j]);
}
}
```

Program to display the elements of a two-dimensional using a pointer.

```
void main()
{
int i, *p, num[3][3]={11, 22, 33, 44, 55, 66, 77, 88, 99);
p = &amp;num[0][0];
for(i=0; i&lt;9; i++)
printf("\n\t %d %u", *(p+i), unsigned(p+i));
}
```

1. Transpose of a Matrix:

The transpose of matrix interchanges rows and columns, i.e. the row elements become column elements and vice versa.

Program to read the elements of the matrix and transpose its elements.

```
void main ()
{
int i, j, row1, row2, col1, col2, a[10][10], b[10][10];
printf("\n\tEnter the order of matrix A : ");
scanf("%d %d", &amp;row1, &amp;col1);
printf("\n\tEnter Elements of matrix A :\n\t");
for (i=0; i&lt;row1; i++)
for(j=0; j&lt;col1; j++)
scanf ("%d", &amp;a[i][j]) ;
row2 = col1; col2 = row1;
for(i=0; i&lt;row1; i++)
for(j=0; j&lt;col1; j++)
b[j][i] = a[i][j];
printf("\n\tThe Transpose Matrix:\n\t");
for(i=0; i&lt;row2; i++)
{
```

```
for(j=0; j&lt;col2; j++)
printf("%4d", b[i][j]);
printf("\n\t");
}
}
```

Program to perform addition and subtraction of two matrices.

```
void main()
{
int i, j, r1, c1, a[10][10], b[10][10], c[10][10];
printf("\n\tEnter Order of Matrices A &amp; B : ");
scanf("%d %d", &amp;r1, &amp;c1);
printf("\n\tEnter Elements of Matrix of A :\n\t\t");
for(i=0; i&lt;r1; i++)
for(j=0; j&lt;c1; j++)
scanf("%d ", &amp;a[i][j]) ;
printf("\n\tEnter Elements of Matrix of B :\n\t\t");
for(i=0; i&lt;r1; i++)
for(j=0; j&lt;c1; j++)
scanf("%d ", &amp;b[i][j]);
printf("\n\tMatrix Addition \n\t");
for(i=0; i&lt;r1; i++)
for(j=0; j&lt;c1; j++)
c[i][j] = a[i][j]+b[i][j]);
printf ("\n\tMatrix Subtraction \n\t") ;
for(i=0; i&lt;r1; i++)
{

for(j=0; j&lt;c1; j++)
c[i][j] = a[i][j]-b[i][j]);
```

```
}
}
/* display the new matrix after addition and subtraction */
Program to perform multiplication of two matrices.
void main()
{
int i, j, k, r1, c1, a[10][10], b[10][10], c[10][10];
printf("\n\tEnter Order of Matrices A &amp; B : ");
scanf("%d %d", &amp;r1, &amp;c1);
printf("\n\tEnter Elements of Matrix of A :\n\t\t");
for(i=0; i&lt;r1; i++)
for(j=0; j&lt;c1; j++)
scanf("%d ", &amp;a[i][j]) ;
printf("\n\tEnter Elements of Matrix of B :\n\t\t");
for(i=0; i&lt;r2; i++)
for(j=0; j&lt;c2; j++)
scanf("%d ", &amp;b[i][j]);
printf("\n\tMatrix Multiplication \n\t");
for(i=0; i&lt;r1; i++)
for(j=0; j&lt;c2; j++)
for(k=0; k&lt;c2; k++)
c[i][j] = c[i][j] + a[i][k] * b[k][j]);

for(i=0; i&lt;r1; i++)
{
for(j=0; j&lt;c2; j++)
printf("%5d", c[i][j]);
printf ("\n\t");
}
```

}

Points to Remember (Arrays)

* We need to specify three things, namely, name, type and size, when we declare an array.

* Always remember that subscripts begin at 0 (not 1) and end at (size -1).

* Defining the size of an array as a symbolic constant makes a program more scalable.

* Be aware of the difference between the “kth element” and the “element k”. The kth element has a subscript k-1, whereas the element k has a subscript of k itself.

* Do not forget to initialize the elements; otherwise they will contain “garbage”.

* Supplying more initializes in the initializer list is a compile time error.

* Use of invalid subscript is one of the common errors. An incorrect or invalid index may cause unexpected results.

* When using expressions for subscripts, make sure that their results do not go outsid permissible range of 0 to size -1. Referring to an element outside the array bounds is an error.

* When using control structures for looping through an array, use proper relational expressic eliminate “off-by-one” errors.

* Referring a two-dimensional array element like x[i, j] instead of x[i][j] is a compile time error.

* When initializing character arrays, provide enough space for the terminating null character.

* Make sure that the subscript variables have been properly initialized before they are used.

* Leaving out the subscript reference operator [ ] in an assignment operation is compile time error.

* During initialization of multi-dimensional arrays, it is an error to omit the

array size for any dimension other than the first.

STRINGS

A sequence of characters, digits, symbols enclosed within double quotation mark is known as string. A string is always declared as character array. Every sting is terminated with a null character '\0'.

Declaration

To declare a string the variable should be declared as character array.

char var name[size];

Eg:char name[20];

Initialization

The compiler automatically inserts the null character at the end of the string .so initialization of null character is not essential.but an extra memory space should be provided to include a null character at the end . If extra memory space is not provided then the output will contain garbage value along with the string.

char str[]="india";

char str[]="india \0";

char name [8]={'n','i','m','i','s','h','a','\0'};

Operation with srting

To read and print strings use printf and scanf along with loop statements or else use puts and gets functions.

q:1. Write a program to print welcome

Program to print your name using string initialization

String standard function

c compiler supports a large no: of string handling library functions. The header file string .h is used whenever the standard library function are used

1. strlen()

This function is used to determine the length of a string.

1. Write a program to count the no: of characters in a string

2. Program to read a string through keyboard and determine the length of string and find its equivalent ascii character

General syntax

int var = strlen(charvar);

2.strcpy()

The function is used to copy a string from a source to destination .the syntax is

strcpy(destination var,source var);

1.

Program to copy contents of one string to another by using strcpy

3.strncpy()

The function copies character of a string to another string up to a specified length .

The general syntax is

strncpy(destination var ,source var,int var);

Or numerical no:

The statement copies 1st 5 characters of str1 to str 2

Eg:int n=6;

strncpy(str2,str1,n);

Eg:strncpy(str2,str1,5);

Program to copy source string to destination string up to a specified length .the length should be entered by the keyboard

4. strcmp()

The function compares the characters of 2 strings .the function discriminates b/w small and capital letters

Eg: str1=Hello;

Str2=hello;

the general syntax is

strcmp(source var,destination var);

5.stricmp()

The function compares 2 strings and does not discriminate b/w the lower and upper case characters .

Eg:str1=hello

str2= hello

As per the function the 2 strings are same.

Syntax

stricmp(source var,destination var);

6.strcmp()

The function compare 2 strings upto a specified length.the syntax of the function

isstrcmp(source var,destination var,int var);

Pgm to compare 2 string upto a specified length.

7.strncmp()

The function can be used instead of strncmp().the only difference between strncmp() and strnicmp() is strnicmp does not discriminate between small and capital letters.

8.strlwr()

The function is used to convert string to lower case.

Syntax

strlwr(char var);

9.strupr()

The function converts lower case string to upper case strings.

Syntax

strupr(char var);

10.strdup()

The function is used for duplicating a given string at the allocated memory location which is pointed by the pointer variable.

Syntax

ptr var=strdup(char var);

11.strchr();

The function returns the pointer position to the 1st occurrence of the character in the gien string.

Syntax

ptr var=strchr(char array,char var);

12.strstr();

The function finds the second string in the 1st string .it returns the pointer location from where the second string starts.

Syntax

strstr (char array,char array);

13.strcat();

The function is used to append the target string to the source string ie; the function concatenate 2 strings.

Syntax

strcat(char array,char array);

Program to append the 2nd string at the end of 1st string

14.strncat()

The function concatenates 2 strings up to the specified length.

Syntax

strcat(car array1,char array2,length);

15.strrev()

The function reverses the given string

Syntax

str rev(char array);

16.strset()

The function replaces every character of the string with the given symbol.

strset(text,symbol);

17. strnset()

The function replaces every character of a string with the given symbol upto a length.

strset(text,symbol, n);

18. strspn()

This function returns the position of the string from where the source array is not matching the target one.

19.strpbrk()

The function searches for the 1st occurrence of the character in a given string and then displays the string starting from th specified character.

Syntax

ptr var=strbrk(txt1,txt2[5]);

Functoins	Description
strncmp()	compares characters of two strings up to the specified length
strnicmp()	compares characters of two strings up to the specified length, ignores case
strlwr()	converts uppercase of a string to lower case

strupr()	converts lower case of a string to uppercase
strdup()	duplicating string
strchr()	determines the first occurrence of a given character in a string
strrchr()	determines the last occurrence of a given character in a string
strstr()	determines the first occurrence of a given string in another string
strcat()	appends source string to the destination string
strncat()	appends source string to the destination string up to a specified length
strrev()	reverse all characters of a string
strset()	sets all the characters of a string with a given symbol
strnset()	sets specified number of characters of a string with a given symbol
strspn()	finds up to what length two strings are identical
strpbrk()	searches the first occurrence of the character in a given string and then displays the starting from that character

Module III: Functions & Pointers: concept of modular programming, Library, User defined functions, declaration, definition & scope, recursion, Pointers: The & and * Operators, pointer declaration, assignment and arithmetic, visualizing pointers, call by value; call by reference, dynamic memory allocation. Storage classes.

FUNCTIONS

A function is a self contained block or a sub program of one or more statements that perform a special task when called. A big program is divided into small modules and the same module can be repeatedly call from the main function when required .this method is known as divide and conquer method. The small modules are called functions

There are two types of functions

1. **Library functions**

   They are predefined set of functions .their task is limited.they cannot be changed or modified by the user. Eg: pow(),sqrt(),clrscr()

2. **User defined function**

   The functions defined by the user according to their requirements are called user defined functions. A user can modify the functions according to a requirement.

Use of functions:

1. To perform the particular block of statements any number of time there is no need to rewrite the statement again and again.
2. Large programs can be reduced to smaller ones.

Working of a function

1. Once a function is defined and called, it takes some data from the calling function and returns a value to the called function.

2. The inner working of a function is unknown to the rest of the program.

3. The values of actual arguments passed by the calling function are received by the formal arguments of the called function.

*the no: of actual arguments should be the same.

4. The function operates on the formal arguments and send back the result to the calling function. The statement 'return' is used to send back the result to the calling function.

Return type function name (datatype1,datatype2……..); // **function prototype**

```
void main()
{
function name(var1,var2….);// Function call
}
returntype function name(datatype1,datatype2,….) // Function definition
{
return(); // return statement
}
```

1.function prototype

It is the function name and follows the same rule for naming a variable.

Eg:int sum(int,int);

2. Function call

A function can be called by using its name and it is terminated by a semicolon

(a)if there is no arguments the n the function can be called by using only the function name.

Eg:sum();

(b) If there is arguments then the function can be called by using the actual arguments

Eg:sum(a,b);

3. Function definition

A function provides the actual body of the function. a function definition contains return type of the function, function name ,parameter list(data type and formal arguments)&the function body.

Eg: int sum(int a,int b)

```
{
int add;
add=a+b;
printf("%d",add);
}
```

The arguments of the called function are called formal arguments.

4. return statement

It is used to return a value to the calling function. The return statement can be used in a no: of ways.

1. return(); the statement returns 0 to the operating system if the value entered by the user is 1 or –ve.

2.
return;it is used to exit from a function

return(exp);

eg: return (a*a*a);

3.
return (&var); it returns value of avariable using pointer.
5.
return (sqrt(var)); library functions can be used along with the return statement.

6.
return float(square(2.8));

TYPES OF FUNCTIONS

Depending on arguments and return value the functions are divided into 4 types

1.without arguments & return value

In this type data is neither passed through the calling function nor send to back to called function. There no data transfer is there. The functions are used perform any operation and act independently .they read data & print result in the same block.

```
Eg:    void abc();
       void main()
{
//function call

abc();
       }
       void abc()
       {
//input statement
// processing
```

```
    //output statement
    }
```

eg:Program to add 2 no:s

2.without arguments and with return value

In this type no arguments is passed through the main function, but the called function returns values. The called fn is independent. It reads value from the keyboard or the variables are initialized & return the value to the main function

```
Eg: void abc ()
void main()
{
int z;
z=abc();
// Output statements
}
void abc()
{
int y=5;
// Input statement, process statement and return.
return(y)
}
```

Program to add 2 no:s

3.with arguments &without return value

In this type of functions arguments are passed to the calling function & the called fn operates on the value but no result is send back .the result is utilized by the called function.

```
Eg: int abc (int,int);
void main()
{
int a,b;
// input statement
abc(a,b);
}
```

```
int abc(int x,int y)
{
// process statement and output statements
}
```

Program to add 2 no:s

4.with arguments & with return value

In this type of function copy of the actual arguments is passed to the formal arguments .here data is transferred to b/w calling and called functions. There should be a return statement in the called function and is collected by a variable in the main function.

```
Eg: int abc (int,int);
void main()
{
int a,b;
// input statement and output statements
abc(a,b);
}
int abc(int x,int y)
{
// process statement and return statement.
}
```

Program to add 2 no:s

Program to add 3no:s

Call by value & call by reference

1.call by value

In this type the value of actual arguments is passed to the formal arguments and the operation is done on the formal arguments .any change in the formal argument does not affect actual argument. The changes made in the actual arguments are local to the block of called function (copy of actual arguments are passed to the formal arguments

Program to swap 2 no:s

2. Call by reference

In this type the address of the values are passed. Functions operate on address rather than values. The formal arguments are pointers to actual arguments. The changes made in the arguments are permanent.

Program to swap 2 no:s

Recursion

A function calling repetitively by itself is called recursion. The recursion can be directly or indirectly

Program to find factorial of no

Types of recursion

The major application of recursion is game programming

1.direct recursion

When a function calls itself it is know n as direct recursion. In this type only one function is involved.

2. Indirect recursion

In this type of recursion a function calls to another function and then called function calls to the calling function. Here more than 2 functions are involved.

Advantages

- Almost all the problem can be solved without recursion, but there may be some situation where a programmer must use recursion.
- Eg:program to display all the file of a system.
- Recursion is very flexible in data structure.
- Length of program can be reduced.

Disadvantages

- It requires extra memory space.
- If programmer forgets to specify the exit condition then pgm will execute out of memory. In such situation user has to press ctrl+break.
- The recursion function is not efficient in execution speed and time
- If possible try to solve a problem with iteration instead of recursion.

POINTERS

poi

nter is a special variable that stores the address of another variable .pointer can have any name as like normal variables but is always denoted by an asterisk(*) symbol

Features.

1. It is a fundamental tool in developing code in c.
2. Saves memory space.
3. Memory is accessed efficiently.

Pointers are used to allocate memory dynamically.

4. Pointers are used with data structures.
5. used for file handling.
6. Execution time with pointer is faster.

Pointer declaration

Genral syntax:

data type * variable name;

The data type specifies the type of variable the pointer variable holds.

The indirection operator is also called dereferencing operator (*). When a pointer is dereferenced, the value at that address is retrieved. The indirection operator is used for declaration of dereference & symbol is the address operator and represents address of the variable .the operator immediately preceding the variable will return the address of the variable

Eg: int *x;
float *y;

The statement int *x ; tells the compiler that it holds the address of an integer value .

Dynamic memory allocation

The allocation of memory during a program runtime is called dynamic memory allocation. It is essential for data structure.

Types of pointers

1.Void pointers

Pointers can also be declared as void. The void pointers cannot be dereferenced

because the compiler cannot determine the size of the object that the pointer points to. Even though void pointer declaration is possible it is not allowed.

2.Wild pointers.

when a pointer points to an unallocated memory location or data value whose memory is deallocated such a pointer id called a wild pointer. The wild pointer generates garbage memory location .when a pointer is declared and is not initialized then it holds an unauthorized address. It is very difficult to manipulate such pointers.

3.Constant pointers

the pointers that are declared as constant cannot be modified .the variable 'const' is sed to declare a variable as pointer

Eg: int k=10;

int const * p=&k;

char * const str =”constant”;

Program to compare 2 pointer address.

Module IV: Advanced features: Array & pointer relationship, pointer to arrays, array of pointers. Strings: String handling functions; Structures and unions; File handling: text and binary files, file operations, Functions for file handling, Modes of files

POINTERS & ARRAYS

Array name itself is an address or pointer .it points to the address of 1st element .the elements of the array together with their address can be displayed by using array name itself. Array elements are always stored in continuous memory location.

q:Write a program to display array elements and their address using array name as a pointer?

Points to remember
1.arr[5]={10,20,30,12,34},p;
arr[p];
(arr+p),(p+ar),p[arr] will give the value of the array
2.&arr[p],arr+p will give the address of the array.
Array of pointers

Array of pointers is a collection of addresses .it stores the address of variables as a pointer.

Declaration

General syntax

datatype*var name[size];

q:Write a program to display array elements using array of pointers?

Structure

A structure is a collection of one or more variables of different data types of grouped under common name. It is a derived data type to be arranged in a group of related data items of different data types.

Declaration &initialization

The general syntax to declare a structure is

struct structure name

{

datatype1 var1;

datatype2 var2;

.

.

}**obj name;**

Here struct is a keyword and struct name is the structure type which is also known as a tag. The different variables of different data type are enclosed within a pair of curly braces .the closing brace is terminated with a semi colon.

Objects

After defining the structure the objects of the structure has to be created .it is similar to declaring variables of any data type. The memory allocation takes place only when the objects are declared.

Initialization

The variables of the structure can be initialized in the following ways.

1.struct book

{

char name[20];

float price;

}

struct book b1;

b1.bname=”pgmg in c”;

b1.price=350;

Or

struct book b1={“pgmg in c”,350}

q:Program to read the values and assign them to structure variables(the structure should include book name,no of pages&price)

Program to initialize structure variables.the structure contain company name,type,price.

Write aprogram to read and display the car no ,starting and reaching time.

Structure with in a structure

We can create object of one structure as a member of another structure. a structure within a structure can be used to create complex data applications.

Eg: struct stud

```
{
char name[20];
int age;
};
struct stud mark
{
int roono;
float mark;
struct stud s1;
};
struct stud mark sm1;
sm1.
```

Array of structures

Array is a collection of similar datatype .we can define an array of structure where the every element is of structure type .array of structure can be declared as

```
struct time
{
int sec;
int min;
int hr;
}t[3];
```

Here t[3]; is an array of 3 elements containing 3 objects of time structure .each element of t[3] has 3 members sec ,min & hr.

q:Write a pgm to create an array of structure objects .the structure details can be information about car no,start time and reaching time of 10 cars.

q:Write a pgm to display name rolllno grade of 5 students .declare the structure student with variables name,rollno,garde.create an array of structure objects.

q:Wite a pgm to display name age and height of 3 students using array of structure.

Pointer to structure

Programmer can define pointer to structure .here starting address of the member variable can be accessed .such pointers are called structure pointers.

```
struct stud
{
char name[20];
int rollno;
float avg;
}*ptr;
```

in the eg: * ptr is pointer to structure stud (when the object is a pointer .create a normal object and assign it to the pointer by symbol '&'.

The variables of the structure can be accessed using an -> operator along with pointer variable.

Eg:

```
void main()
{
struct stud
{
char name[20];
int rollno;
}*s1;
struct stud n1;
printf("enter values");
gets(n1.name);
scanf(%d",&n1.rollno);
s1=&n1;
printf("the values are ");
printf("%s%d",s1->name,s1->rollno);
```

Q:write a pgm to declare the pointer to structure and display contents of structure .the structure book contains variable name,author and pages.

2.declaring variables as structure pointers.

Syntax:

```
struct stud
{
char name[20];
int * roll no;
}
struct stud s1;
char name[20];
int rollno;
 s1.name=&name1;
s1.rolno=&rollno1;
```

in this type structure variables are accessed without->operator.

Eg:

Q:write a pgm to declare pointer as member of structure and display the content without using ->operator. The structure contain name, age and height.

Q:write a pgm to declare a pointer as member of structure and display content without using ->operator. The structure book contains details name ,author pages and price.

3.variables as well as objects as pointers.

In this type the member of structure as well as object are pointers. Here declare normal /ordinary variables for the structure members. Assign ordinary variables to the structure variables while using->operator .to display the contents of structure variables use*operator

Along with ->operator.

Eg:

```
struct stud
{
```

```
char* name[20];
int*rollno;
}*s1;
char nm[20];
int rn;
printf("enter name and rollno");
gets(nm);
sacnf("%d",rn);
s1->name=&nm;
s1->rollno=&rn;
printf("the name is%s",*s1->name):
print("rollnois%d",*s1->rollno);
```

Write a pgm to declare pointers as members of structure and object of structure as pointer.display the content of structure .the structure stud containsname,rollno,height and weight.

Type definition

Typedef

It is used to create a new datatype .

Syntax

typedef datatype variable name;

Eg: typedef int hours;

From the above stmt hours can be used to create integer variables instead of int.

hours hr;

The stmt declaes 'hr' as integer variable.

Write a pgm to create a user defined datatype hours on int datatype and use int in the pgm.

Enumerated data type.

It is used for declaring enumeration types.
The enum is the keyword.
For eg:enum month{jan,feb,…….dec};

The programmer can create heir own data type and define what values the variables of these data types can hold.
Syntax: tag name{identifiers};
enum month {jan=1,feb,…dec};

The statement creates a user defined data type and keyword enum is followed by the tagname month. The enumerators are jan,…dec.
The values of the identifiers are constant unsigned integers starting from 0.by default the compiler assign values from 0 onwards. The programmer can initialize their own constants to each identifier.

Q:write a pgm to create enumerated datatype for days in a week and display their integer constants.
Q:wirite a program to create enumerated data for 12 months initialize the 1st identifier with 1 and display integer constants of jan feb june and dec.
Q:write a program to declare enum datatype &display values(true means 1 ,false means0).

UNION

Union is a variable similar to structure. In structure each member has its own memory location. The union requires bytes that are equal to the number of bytes required for the largest member.
Syntax:
union unionname
{
datatype 1 varname1;
datatype2 varname2;

.
}object;
Here union is the keyword and union name is defined by the programmer. The object along with dot(.)Operator is used to access the union variables.

q:write a program to define a union result with variables rollno,mark and grade;

FILES

A file is a accumulation of data stored on the disk created by the user ie;a file is a collection of records. A record is a group of related data item. A file can be collection of numbers, symbols and text. The library functions of file are available in<stdio.h>
Types of files
Sequential file

In this type data are kept sequentially it takes more time for accessing the records .eg:to access the 10^{th} record, the first 9 records should be accessed.
Random access file

In this type data can be read and modified randomly. The access time is less compared to the sequential file. The user can access the desired record without accessing the other records.
Opening a file

The opening of a file creates a link between the operating system and the file functions. This task is carried out by the file that is defined in <stdio.h>
Syntax:
file*fopen (“filename”,”mode”);
Eg:file*fp;
fp=fopen(“data.txt”, “r”);
Here fp is a pointer variable data.txt is the filename and r specifies that the file is open in read mode.
Closing a file

The file that is open using fopen should be closed after the work is over. The closing enables to washout all the contents from the ram buffer.

Syntax:

fclose(ptr var name);

Or

fcloseall();

TEXT MODES

text mode	binary mode	explanation
w(write)	wb	this mode opens file for writing .if the file is already exist then its contents will be over written.if the file is not found , a new file is created
r(read)	rb	this mode opens file for reading . if the file is not found then the compiler returns 'null' to the file pointer.
a(append)	ab	this mode opens pre existing file for appending .if the file is not found then 'null' is returned.
w+(write+read)	r+b	this mode opens file in read and write mode
a+		this mode pens file in read mode and records can be added at the end of the file
r+(read+write)	w+b	this mode opens file in read and write mode.
	a+b	this mode opens file in append mode ie;data can be written at the end of the file.

FILE INPUT AND OUTPUT.

1.fprintf()

The function is used for writing values of different datatype to the file.

Syntax:

fprintf(ptrvar,"ctrlstrg",text);

2.fscanf()

The function reads values of different datatype from file pointed by the file pointer.

Syntax:

fscanf(ptrvar,"ctrlstrg",&text);

Q:write a program to enter name rollno and mark into the text file and read the same.

3.fgetc and fputc

This function is similar to getc() function. It also reads a character and increments the file pointer position.

Syntax:

var=fgetc(file ptr);

fputc()

This function writes a character to the file. It also incerements the file pointer.

Syntax:

fputc(var,filepointer);

4.fgets()

The function is used too read a string from a file. It also copies string to a memory location referred by an array

Syntax;

fgets(var,num,fileptr);

variable is a string variable, number represent no: of characters and file pointer is a variable created by using file.

5.fputs()

The function is used to write a string into the open file .

Syntax

fputs(var,fileptr);

READ AND WRITE FUNCTION.

6.**fwrite()**

This function is used for writing an entire structure to a given file.

Syntax:

fwrite(&objnmae,sizeof(objname),num of file,fileptr);

7.**fread()**

The function is used for reading an entire file from a given file.

Syntax:

fread(&objname,sizeof(objname),num of file,fileptr);

1. *write a program to read and write a character from file using fgetc and fputc.*
2. *write a program to read an write text from given file using fgets and fputs.*
3. *write a program to write and read the information about student to the given file using fread()and fwrite() functions.*
4. *write a program to read and write information of players containing name age and runs using fread() & fwrite().*

1.EOF: the keyword is used to determine the end of file.

syntax: **filepointer!=EOF**;

2.feof():this function is to determine the end of a file.

program to display data in a file.

write a program to write text to a file and read the data till the end of the data and display it

3.NULL: to find whether the file is empty or not

the statement ,**if (file ptr==NULL)** checks whether the file is empty or not.

other file functions.

1.fseek(): the function is used to move the file pointer to any position in a file. The format is:

fseek(file ptr,offset,position);

1.file pointer

it is the pointer variable to open a file.

2.offset

it is the value for moving the file pointer forward from current position or backward from current position.

3.position

it is the current position of the file pointer.

integer value	constant	location in the file
0	SEEK-SET	beginning of the file
1	SEEK-CUR	current position of file pointer
2	SEEK-END	end of file

2.ftell()

this function is used to find the current position of the file pointer. it returns the pointer from the beginning of the file.

syntax

variable=ftell(fileptr);

write a pgm to print the current position of the file pointer if the file is not empty.

write a pgm to write to a file from the 10th position.

preprocessor directives

preprocessor is a program that process the source program before it is passed to the compiler. the program written in any language and type in the editor is the source code to the preprocessor. it can reduce the execution time of a program. the processor directives are always initialized at the beginning of program before the main().it begins with symbol#.

#define:

the syntax is:

#define macroname char sequence

Macro is a name given to a block of C statements as per the pre processor directive

Eg:

#define identifier substitute or #define identifier(arg1,arg2….arg n)

Eg : **#define pi=3.14**

Here pi is the macro name.

In this statement pi is the macro template and 3.14 is the macro substitute. During preprocessing the preprocessor replaces every occurrence of pi with 3.14. the macro template are generally declared with capital letters &is not terminated with semicolon.

#include directive

the #include directive loads specified file in the current program. the syntax is:

#include "filename"

or

#include<filename>

the file name included in " " indicate that the search for the file is made in current directory and in the standard directory. when the file name is included in the angle brackets<> then the search for the file is made only in the standard directory.

Eg: #include<stdio.h>

conditional compilation.

programmer can include conditional compilation directives which allow the programmer to include the portions of code based on the conditions . the most frequently used conditional compilation directives are: #ifdef, #else, #endif.

eg:

```
#define LINE1
void main()
{
int x=5;
#ifdef LINE
printf("this line is no:1") ;
#else
```

```
printf("this is line no:2");
#endif
getch();
}
```

Undefining a macro

A macro defined with #define can be undefined with #undef directive.

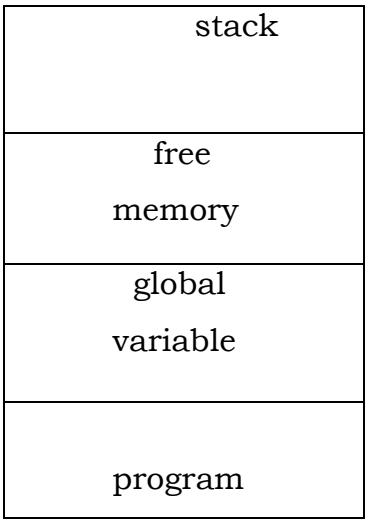
syntax:

#undef macrotemplate macrosubstitute

Dynamic memory allocation

There are 2 techniques for storing the information of c programs in the main memory.

1. Storing global and local variables in the fixed memory area during compilation and remain constant throughout run time program.
2. The program can obtain memory space in the main memory at the time of execution of the program. The method of allocation of memory at the time of execution of the program is called dynamic memory allocation.



Space for local variables is allocated from stack .each time when the variable comes into existence; the free memory is called the 'heap'. The size of the heap does not remain constant all the time for all programs. Its size keeps on changing during the execution of the program. This is due to the creation and destruction of variables that are local to the functions and blocks.

MEMEORY MODELS

1. Tiny:

All segment registers are initialized with identical value and the addressing is accomplished 16 bits. The programs are executed quickly in this case.

2. Small:

All codes should fit in a single 64 kb segment and all data should fit in a 2nd 64 kb bit segment. Execution speed is same as that of tiny model.

3. Medium:

All data should be fit in single 64 kb segment and the code is allowed to use multiple segments fast access of data but slower program execution.

4. Compact:

All codes should fit in 64 kb but the data can use multiple segments. There is slow access to data and quick code execution.

5. Large:

Both code and data are allowed to use multiple segments. There is slower code execution.

6. Huge:

Both code and data are allowed to use multiple segments and there is slowest code execution.

MEMORY ALLOCATION FUNCTIONS

1. **malloc():** it is used to allocate the memory block with the function .ie; it is used to allocate memory space to the variables of different data type.

2. **calloc():**

it is used for allocating multiple blocks of memory for the variables.

3. **Free ()**

It is used to release the memory if it is not required.

(Header file used for these three are <stdio.h>or<alloc.h>)

ALGORITHM EFFICIENCY ANALYSIS

It is essential to analyze the algorithm for calculating or guessing the resources needed for an algorithm. Resources mean computer memory, processing time, logic gates. The most important factor is time because the program developed should be faster in processing.

The analysis can also be made for logical accuracy, tracing the algorithm implementing it and checking with some data and with mathematical techniques to confirm its accuracy.

GRAPHIC FUNCTIONS

Aheader file graphics.h is used to include the graphic functions to the program. To switch to the graphic mode, a function name init graph() is used.

FEW GRAPHIC FUNCTIONS

1. initgraph(): this function initializes the graphic system.
2. Close graph (): it is used to shut down the graphic system.
3. line (): it is used to draw a line between 2 specified points.
4. circle (): this is used to draw a circle at specified points of the given radius.