MLR Institute of Technology

Laxma Reddy Avenue, Dundigal, Quthbullapur (M), Hyderabad – 500 043

Faculty Name	:	N.SANDHYA RANI
Subject	:	PPS
Topic	:	**Problem Solving and algorithmic thinking & introduction to c language**

UNIT –I CLASSNOTES

What is algorithm?

An algorithm is a procedure or step-by-step instruction for solving a problem. They form the foundation of writing a program.

For writing any programs, the following has to be known:

- Input
- Tasks to be preformed
- Output expected

Write an algorithm to add two numbers and display the result.

Input — Take two numbers

Processing — adding of two numbers

Output — display result

Algorithm

Step-1 Start

Step-2 Input first numbers say A

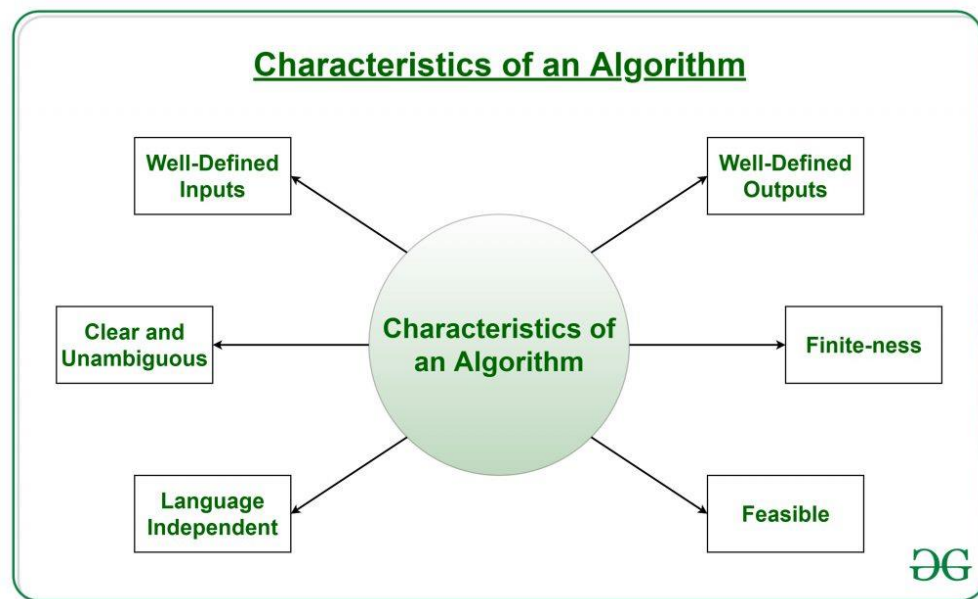**Step-3** Input second number say B

Step-4 SUM = A + B

Step-5 Display SUM

Step-6 Stop

Algorithm

Properties of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the below mentioned characteristics −

- **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their input/outputs should be clear and must lead to only one meaning.

- **Input** − An algorithm should have 0 or more well defined inputs.

- **Output** − An algorithm should have 1 or more well defined outputs, and should match the desired output.

- **Finiteness** − Algorithms must terminate after a finite number of steps.

- **Feasibility** − Should be feasible with the available resources.

- **Independent** − An algorithm should have step-by-step directions which should be independent of any programming code.

Flowchart

Definition:

A **flowchart** is a type of <u>diagram</u> that represents a <u>workflow</u> or <u>process</u>. A flowchart can also be defined as a diagrammatic representation of an <u>algorithm</u>, a step-by-step approach to solving a task.
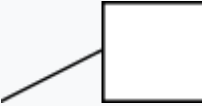
Common Symbols used in flowchart

ANSI/ISO Shape	Name	Description
→	Flowline (Arrowhead)	Shows the process's order of operation. A line coming from one symbol and pointing at another. Arrowheads are added if the flow is not the standard top-to-bottom, left-to right
(stadium)	Terminal	Indicates the beginning and ending of a program or sub-process. Represented as a <u>stadium</u>, oval or rounded (fillet) rectangle. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit inquiry" or "receive product".
(rectangle)	Process	Represents a set of operations that changes value, form, or location of data. Represented as a <u>rectangle</u>.
(diamond)	Decision	Shows a conditional operation that determines which one of the two paths the program will take. The operation is commonly a yes/no question or true/false test. Represented as a diamond (<u>rhombus</u>).
(rhomboid)	Input/Output	Indicates the process of inputting and outputting data as in entering data or displaying results. Represented as a <u>rhomboid</u>.

	Annotation (Comment)	Indicating additional information about a step in the program. Represented as an open rectangle with a dashed or solid line connecting it to the corresponding symbol in the flowchart.
	Predefined Process	Shows named process which is defined elsewhere. Represented as a rectangle with double-struck vertical edges.
	On-page Connector	Pairs of labeled connectors replace long or confusing lines on a flowchart page. Represented by a small circle with a letter inside.
	Off-page Connector	A labeled connector for use when the target is on another page. Represented as a home plate-shaped pentagon.

Example 1: Calculate sum of two number.
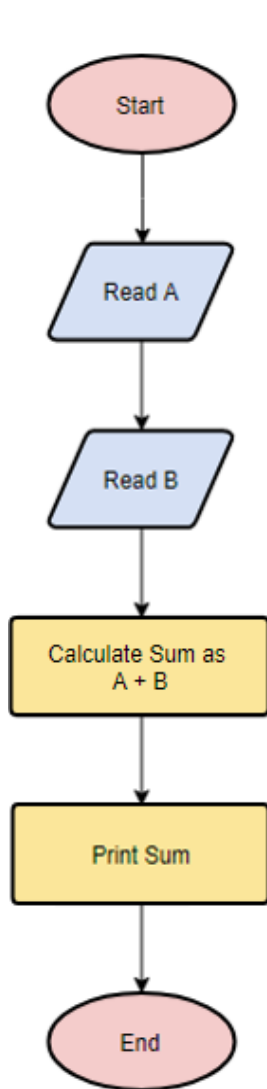
Flowgorithm

Introduction

Flowgorithm is a graphical authoring tool which allows users to write and execute programs using flow charts.

Flowgorithm is a beginner-friendly programming language in which a user would have to create a flowchart for carrying forward a task, which means in a programming language you would have to code it thoroughly to create and start a task.

Graphical Shapes of Flowgorithm

- **Flowgorithm combines the classic flowchart symbols**
- **Flowgorithm supports a wide-variable of color schemes**
- **The application comes with a selection of built-in schemes.**

Input	Declare	If	While
Output	Assign	Call	For
			Do
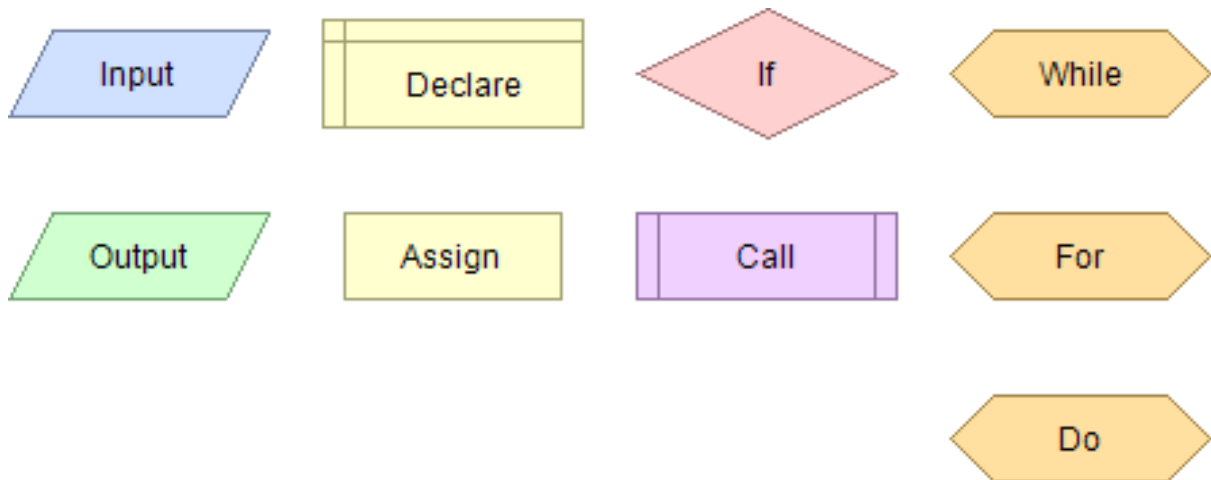
Examples of Flowgorithm

Example 1: Simple example

```
        ┌──────────┐
        │   Main   │
        └──────────┘
             │
             ▼
   ╱────────────────────╲
  ╱ Output "Hello, world" ╲
  ╲────────────────────────╱
             │
             ▼
        ┌──────────┐
        │   End    │
        └──────────┘
```

Example 2: Flowgorithm of a number which is greater or less than zero

```
                    ┌──────────┐
                    │   Main   │
                    └──────────┘
                         │
                         ▼
                 ┌─────────────────┐
                 │ Integer number  │
                 └─────────────────┘
                         │
                         ▼
                ╱───────────────────╲
               ╱  Output "What is your ╲
               ╲    number?" ...       ╱
                ╲─────────────────────╱
                         │
                         ▼
                ╱───────────────────╲
                ╲  Input number      ╱
                 ╲─────────────────╱
                         │
                         ▼
      False          ◇ number > 0 ◇          True
    ┌──────────────────             ──────────────────┐
    │                                                 │
    ▼                                                 ▼
╱─────────────────────╲                  ╱─────────────────────╲
╱ Output "Less than or  ╲                ╱  Output "More than 0." ╲
╲   equal to 0."         ╱               ╲──────────────────────╱
 ╲─────────────────────╱                            │
    │                                               │
    └──────────────────► ● ◄──────────────────────┘
                         │
                         ▼
                ╱───────────────────────╲
                ╱ Output "End of program." ╲
                ╲─────────────────────────╱
                         │
                         ▼
                    ┌──────────┐
                    │   End    │
                    └──────────┘
```
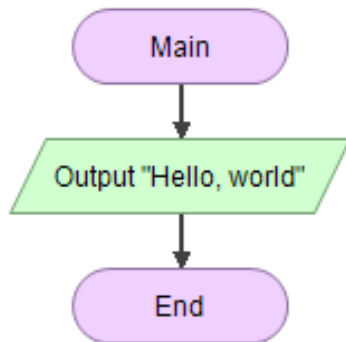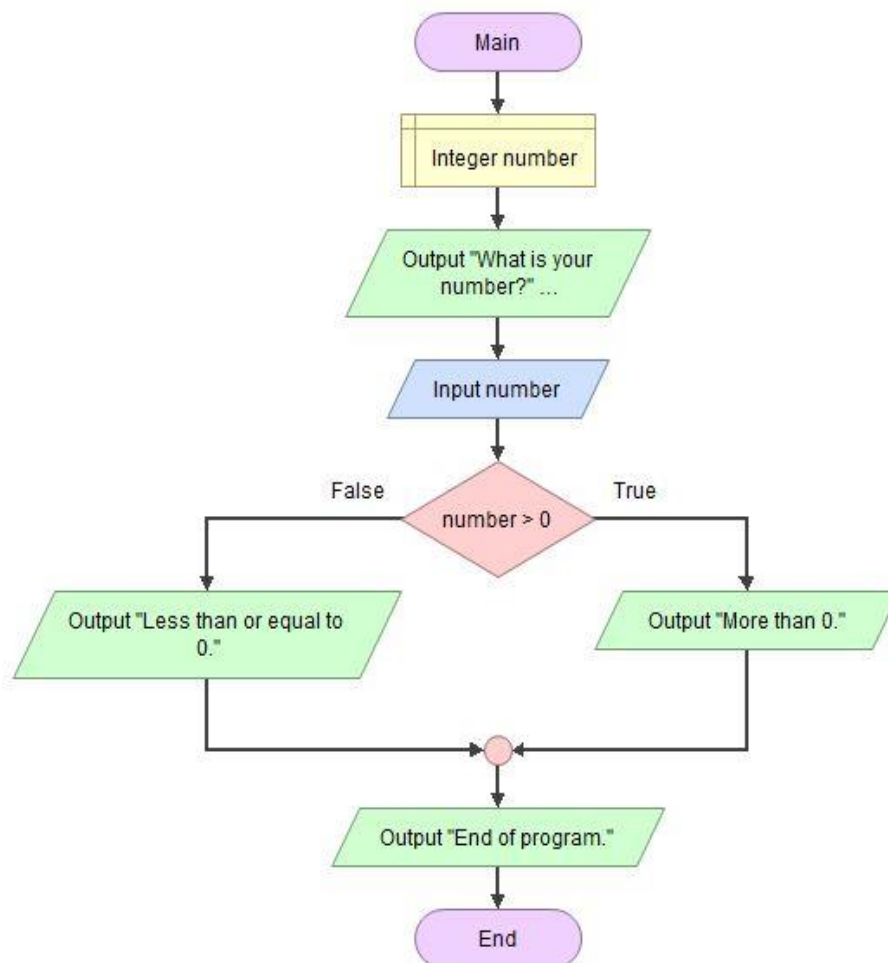
Example 3: Flowagorithm of swapping of two numbers
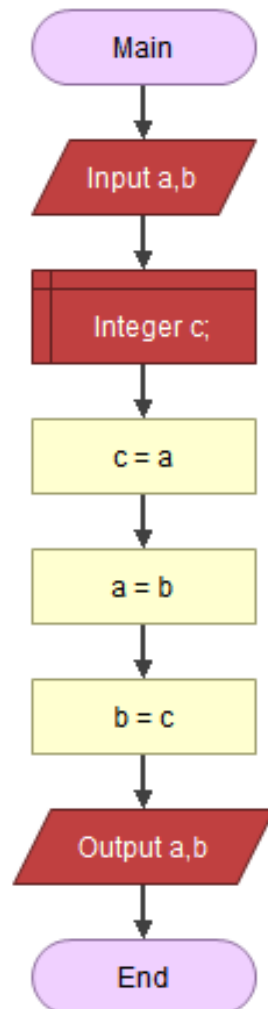
Constituents of algorithms – (Sequence, Selection and Repetition)
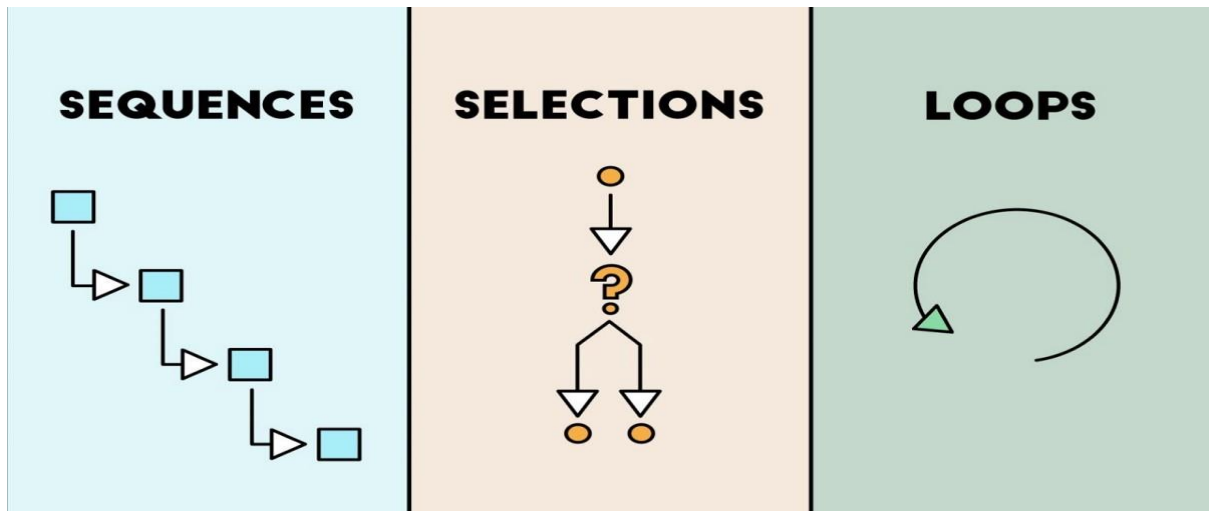
- The algorithm and flowchart, classification to the three types of control structures. **They are:**

  1. Sequence

  2. Branching (Selection)

  3. Loop (Repetition /iteration)

- These are three basic building blocks (constructs) to use when designing algorithms.
- These three control structures are sufficient for all purposes

SEQUENCE:-

- The sequence is exemplified by sequence of statements place one after the other – the one above or before another gets executed first. In flowcharts, sequence of statements is usually contained in the rectangular process box.

EX: FIND THE AREA OF CIRCLE

Problem 1: Find the area of a Circle of radius r.
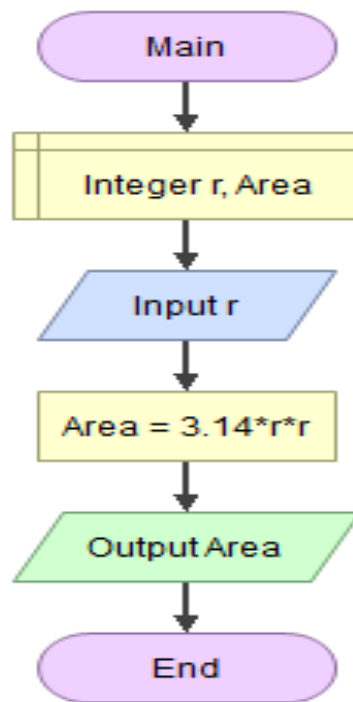
Inputs to the algorithm: Radius r of the Circle.

Task :calculate radius of circle

Expected output: Area of the Circle

Algorithm:

- Step1:Start
- Step 2: Read\input the Radius r of the Circle
- Step 3: Area PI*r*r

// calculation of area

- Step 4: Print Area
- Step 5:Stop

Selection:-

- When designing algorithms, there are many steps where decisions must be made.
- Selection is a decision or question.
- At some point in an algorithm there may need to be a question because the algorithm has reached a step where one or more options are available.
- Depending on the answer given, the algorithm will follow certain steps and ignore others.
- The branch refers to a binary decision based on some condition.
- If the condition is true, one of the two branches is explored;
- if the condition is false, the other alternative is taken.
- This is usually represented by the 'if-then' construct in pseudo-codes and programs.
- In flowcharts, this is represented by the diamond-shaped decision box.
- This structure is also known as the Selection/Branching structure.

- **Problem1: write algorithm to find the greater number between two numbers**

  - Inputs to the algorithm: First num1. Second num2.
  - Task: find greatest of two numbers
  - Expected output: print the greatest number.

    Algorithm:
    Step1: Start
    Step2: Read/input A and B
    Step3: if A greater than B then
    　　　display "A is greatest"
    Step4: otherwise
    　　　display "B is greatest"
    Step6: End
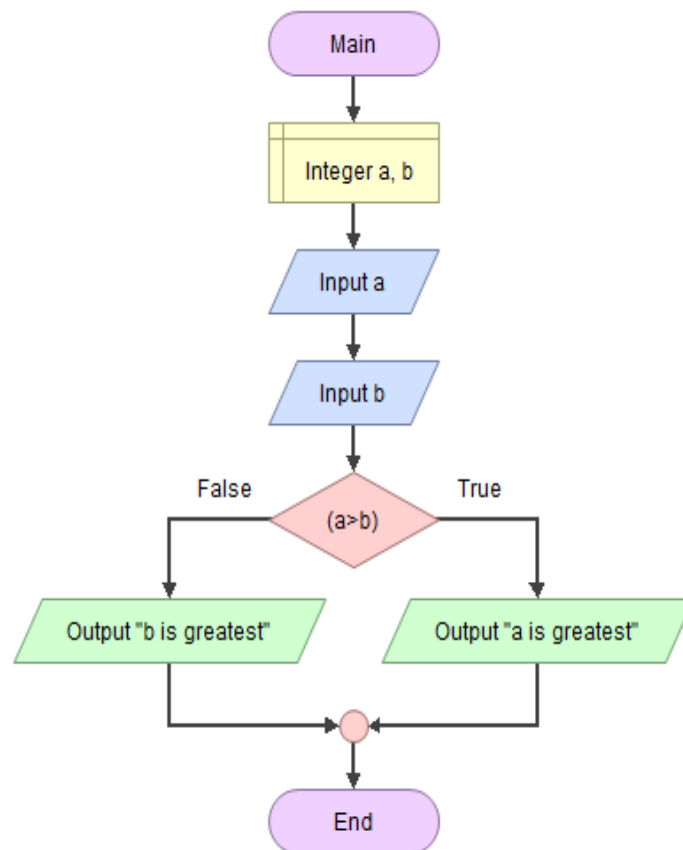
REPETITION:-

When designing algorithms, there may be some steps that need repeating. This is known as iteration and can be displayed with algorithms and flowcharts.

Iteration in programming means repeating steps, or instructions, over and over again. This is often called a 'loop'.

Algorithms consist of instructions that are carried out (performed) one after another. Sometimes an algorithm needs to repeat certain steps until told to stop or until a particular condition has been met.

Iteration is the process of repeating steps.

The loop allows a statement or a sequence of statements to be repeatedly executed based on some loop condition.

It is represented by the 'while' and 'for' constructs in most programming languages.

In the flowcharts, a back arrow hints the presence of a loop.

A trip around the loop is known as iteration.

You must ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers.

The loop is also known as the repetition structure.

Problem 1: A algorithm to find sum of individual digits of a given number

- Inputs to the algorithm: number
- Task: calculate sum of individual digits
- Expected output: print result

Algorithm:

Step 1:Start

Step 2: input num
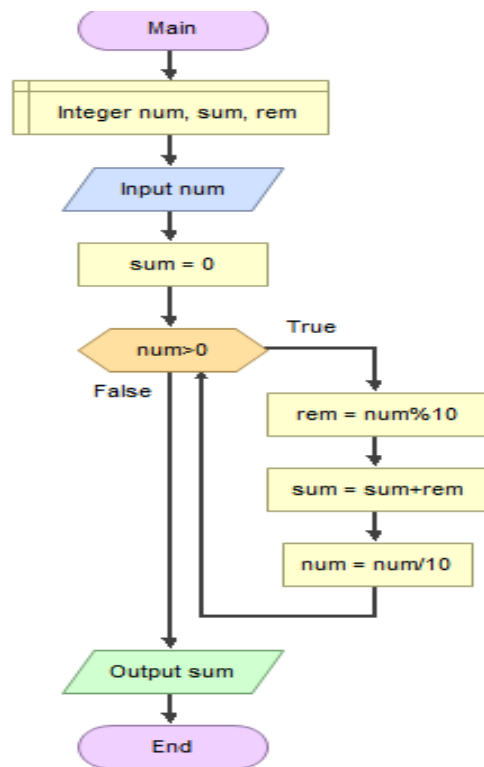
Step 3: assign sum=0

Step 4:Repeat the steps a to c until the condition   n>0 is false

- step a:rem=n%10
- step b:sum=sum + rem
- step c:n=n/10

Step 5:print sum

Step 6:stop

```
                    ┌──────────────┐
                    │     Main     │
                    └──────┬───────┘
                           │
                           ▼
               ┌────────────────────────┐
               │ Integer num, sum, rem  │
               └───────────┬────────────┘
                           │
                           ▼
                  ╱────────────────╲
                 ╱   Input num      ╲
                 ╲                  ╱
                  ╲────────────────╱
                           │
                           ▼
               ┌────────────────────────┐
               │       sum = 0          │
               └───────────┬────────────┘
                           │
                           ▼                    True
                     ╱──────────╲──────────────────────┐
                     ╲  num>0   ╱                       │
                      ╲────────╱                        ▼
                           │                  ┌──────────────────┐
                     False │                  │   rem = num%10   │
                           │                  └────────┬─────────┘
                           │                           │
                           │                           ▼
                           │                  ┌──────────────────┐
                           │                  │  sum = sum+rem   │
                           │                  └────────┬─────────┘
                           │                           │
                           │                           ▼
                           │                  ┌──────────────────┐
                           │                  │   num = num/10   │
                           │                  └────────┬─────────┘
                           │                           │
                           │◄──────────────────────────┘
                           ▼
                  ╱────────────────╲
                 ╱   Output sum     ╲
                 ╲                  ╱
                  ╲────────────────╱
                           │
                           ▼
                    ┌──────────────┐
                    │     End      │
                    └──────────────┘
```

Faculty Name: N.SANDHYA
RANI   **Subject** :PPS
Topic: Introduction to C language:
Structure of C programs

Unit No   :2
Lecture No   :L17
Book Reference:T1

Basic structure of C programs:

C is a structured programming language. Every c program and its statements must be in a particular structure. Every c program has the following general structure...

```
/* comments */          It is opional. Generally used to provice description about the program
preprocessing commands  It is opional. Generally used to include header files, define constants and enum
gloabal declarations;   It is opional. Used to declare the variables that ae common for multiple functions
int main()
{                       main is a user defined function and it is compulsory statement.
                        It indicates the starting point of program execution.
    local declarations; Without main compiler does not understand from which statement execusion starts
    executable statements;
    .
    .                   Local declaration and executable statements are written according to our requirment
    .
    return 0;
}
userdefined function()  It is opional. Used to provide implementation for user defined functions that already
{                       declared either at global or local declaration part.
    function definition;
}
.
```

Line 1: Comments - They are ignored by the compiler

This section is used to provide small description of the program. The comment lines are simply ignored by the compiler, that means they are not executed. In C, there are two types of comments.

1.**Single Line Comments:** Single line comment begins with // symbol. We can write any number of single line comments.

2.**Multiple Lines Comments:** Multiple lines comment begins with /* symbol and ends with */. We can write any number of multiple lines comments in a program.

In a C program, the comment lines are optional. Based on the requirement, we write the comments. All the comment lines in a C program just provide the guidelines to understand the program and its code.

Line 2: Preprocessing Commands

Pre-processing commands are used to include header files and to define constants. We use **#include** statement to include header file into our program. We use **#define** statement to define a constant. The pre-processing statements are used according to the requirment. If we don't need any header file, then no need to write #include statement. If we don't need any constant, then no need to write #define statement.

Line 3: Global Declaration

Global declaration is used to define the global variables, which are common for all the functions after its declaration. We also use the global declaration to declare functions. This global declaration is used based on the requirement.

Line 4: int main()

Every C program must write this statement. This statement (main) specifies the starting point of the C program execution. Here, main is a user defined method which tells the compiler that this is the starting point of the program execution. Here, **int** is a datatype of a value that is going to return to the Operating System after completing the main method execution. If we don't want to return any value, we can use it as **void**.

Line 5: Open Brase ({)

The open brase indicates the begining of the block which belongs to the main method. In C program, every block begins with '{' symbol.

Line 6: Local Declaration

In this section, we declare the variables and functions that are local to the function or block in which they are declared. The variables which are declared in this section are valid only within the function or block in which they are declared.

Line 7: Executable statements

In this section, we write the statements which perform tasks like reading data, displaying result, calculations etc., All the statements in this section are written according to the requirment.

Line 9: Closing Brase (})

The close brase indicates the end of the block which belongs to the main method. In C program every block ends with '}' symbol.

Line 10, 11, 12, ...: Userdefined function()

This is the place where we implement the userdefined functions. The userdefined function implementation can also be performed before the main method. In this case, the user defined function need not to be declared. Directly it can be implemented, but it must be before the main method. In a program, we can define as many userdefined functions as we want. Every user defined function needs a function call to execute its statements.

1. Every executable statement must end with semicolon symbol (;).
2. Every C program must contain exactly one main method (Starting point of the program execution).
3. All the system defined words (keywords) must be used in lowercase letters.
4. Keywords can not be used as user defined names(identifiers).

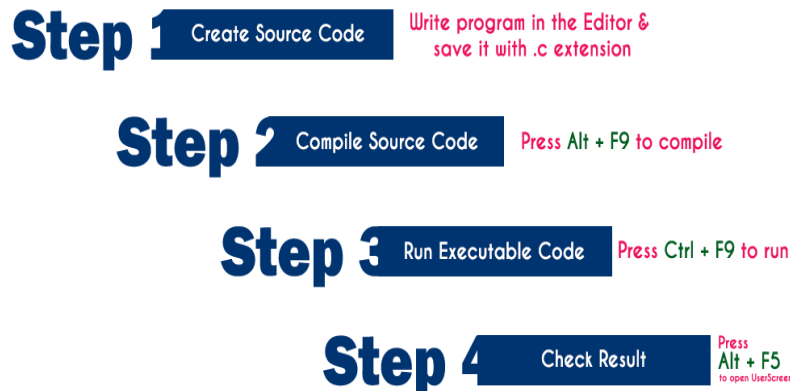5.For every open brase ({), there must be respective closing brase (}).

6.Every variable must be declared before it is used.

Process of compiling and running a C program:

Generally, the programs created using programming languages like C, C++, Java etc., are written using high level language like English. But, computer cannot understand the high level language. It can understand only low level language. So, the program written in high level language needs to be converted into low level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.

Popular programming languages like C, C++, Java etc., use compiler to convert high level language instructions into low level language instructions. Compiler is a program that converts high level language instructions into low level language instructions. Generally, compiler performs two things, first it verifies the program errors, if errors are found, it returns list of errors otherwise it converts the complete code into low level language.

To create and execute C programs in Windows Operating System, we need to install Turbo C software. We use the following steps to create and execute C programs in Windows OS…

Step 1 Create Source Code   Write program in the Editor & save it with .c extension

Step 2 Compile Source Code   Press Alt + F9 to compile

Step 3 Run Executable Code   Press Ctrl + F9 to run

Step 4 Check Result   Press Alt + F5 to open UserScreen

Step 1: Creating Source Code

Source code is a file with C programming instructions in high level language. To create source code, we use any text editor to write the program instructions. The instructions written in the source code must follow the C programming language rules. The following steps are used to create source code file in Windows OS…

- Click on Start button
- Select Run
- Type cmd and press Enter
- Type cd c:\TC\bin in the command prompt and press Enter
- Type TC press Enter
- Click on File -> New in C Editor window
- Type the program
- Save it as FileName.c (Use shortcut key F2 to save)

Step 2: Compile Source Code (Alt + F9)

Compilation is the process of converting high level language instructions into low level language instructions. We use the shortcut key Alt + F9 to compile a C program in Turbo C.

Compilation is the process of converting high level language instructions into low level language instructions.

Whenever we press Alt + F9, the source file is going to be submitted to the Compiler. On receiving a source file, the compiler first checks for the Errors. If there are any Errors then compiler returns List of Errors, if there are no errors then the source code is converted into object code and stores it as file with .obj extension. Then the object code is given to the Linker. The Linker combines both the object code and specified header file code and generates an Executable file with .exe extension.

Step 3: Executing / Running Executable File (Ctrl + F9)

After completing compilation successfully, an executable file is created with .exe extension. The processor can understand this .exe file content so that it can perform the task specified in the source file.

We use a shortcut key Ctrl + F9 to run a C program. Whenever we press Ctrl + F9, the .exe file is submitted to the CPU. On receiving .exefile, CPU performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called User Screen.

Step 4: Check Result (Alt + F5)

After running the program, the result is placed into User Screen. Just we need to open the User Screen to check the result of the program execution. We use the shortcut key Alt + F5 to open the User Screen and check the result.

Faculty Name: N.SANDHYA
RANI   **Subject** :PPS
Topic: Data types

Unit No   :2
Lecture No   :L18
**Book Reference**:T1

DataTypes:

Data used in c program is classified into different types based on its properties. In c programming langauge, datatype can be defined as a set of values with similar characteristics. All the values in a datatype have the same properties.

Datatypes in c programming language are used to specify what kind of value can be stored in a variable. The memory size and type of value of a variable are determined by varible datatype. In a c program, each variable or constant or array must have a datatype and this datatype specifies how much memory is to be allocated and what type of values are to be stored in that variable or constant or array. The formal definition of datatype is as follows...

Datatype is a set of value with predefined characteristics. Datatypes are used to declare variable, constants, arrays, pointers and functions.

In c programming language, datatypes are classified as follows...

1.Primary Datatypes (Basic Datatypes OR Predefined Datatypes)

2.Derived Datatypes (Secondary Datatypes OR Userdefined Datatypes)

3.Enumeration Datatypes

4.Void Datatype
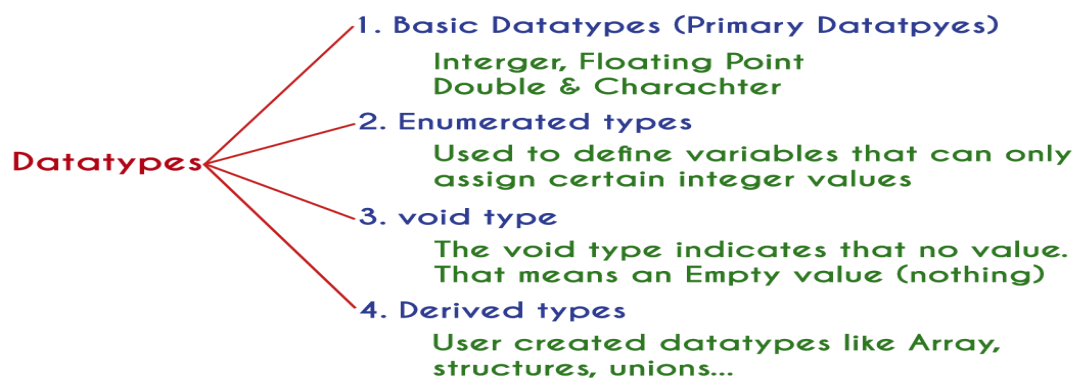
Primary Datatypes

The primary datatypes in C programming language are the basic datatypes. All the primary datatypes are already defined in the system. Primary datatypes are also called as Built-In datatypes. The following are the primary datatypes in c programming lanuage...

1.Integer Datatype

2.Floating Point Datatype

3.Double Datatype

4.Character Datatype

Basic Datatypes (Primary Datatpyes)

```
                    Basic Datatypes (Primary Datatpyes)
                                    |
        +---------------------------+---------------------------+
        |                           |                           |
     Interger                 Floating Point               Character
        |                           |                           |
   +----+----+                   — float                     — char
   |         |                   — double                    — signed char
Signed   Unsigned               — long double                — Unsigned Char
   |         |
 — int     — int
 — short int — short int
 — long int  — long int
```
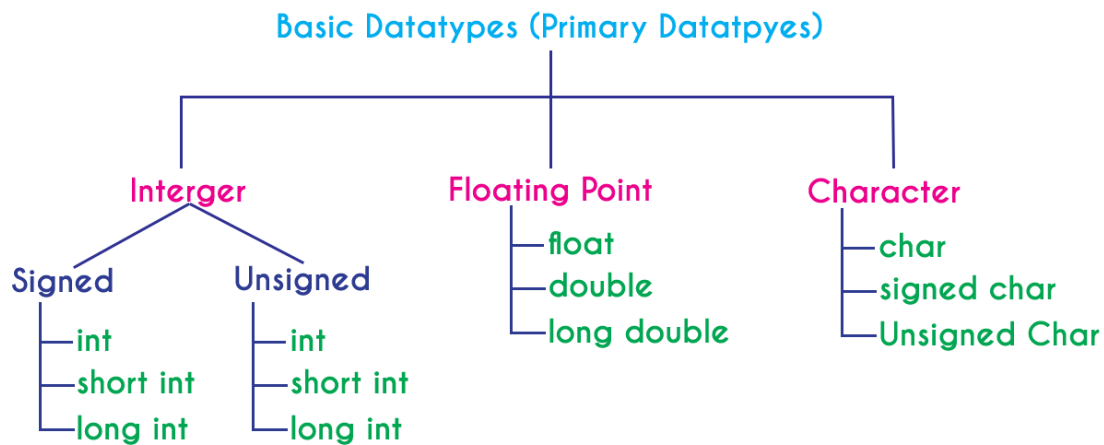
Integer Datatype

Integer datatype is a set of whole numbers. Every integer value does not have the decimal value. We use the keyword "**int**" to represent integer datatype in c. We use the keyword int to declare the variables and to specify return type of a function. The integer datatype is used with different type modifiers like short, long, signed and unsigned. The following table provides complete details about integer datatype.

Type	Size (Bytes)	Range	Specifier
int (signed short int)	2	-32768 to +32767	%d
short int (signed short int)	2	-32768 to +32767	%d
long int (signed long int)	4	-2,147,483,648 to +2,147,483,647	%d
unsigned int (unsigned short int)	2	0 to 65535	%u
unsigned long int	4	0 to 4,294,967,295	%u

Floating Point Datatypes

Floating point datatypes are set of numbers with decimal value. Every floating point value must contain the decimal value. The floating point datatype has two variants...

- float
- double

We use the keyword **"float"** to represent floating point datatype and **"double"** to represent double datatype in c. Both float and double are similar but they differ in number of decimal places. The float value contains 6 decimal places whereas double value contains 15 or 19 decimal places. The following table provides complete details about floating point datatypes.

Type	Size (Bytes)	Range	Specifier
float	4	1.2E – 38 to 3.4E + 38	%f
double	8	2.3E–308 to 1.7E+308	%ld
long double	10	3.4E–4932 to 1.1E+4932	%ld

Character Datatype

Character datatype is a set of characters enclosed in single quotations. The following table provides complete details about character datatype.

Type	Size (Bytes)	Range	Specifier
char (signed char)	1	-128 to +127	%c
unsigned char	1	0 to 255	%c

The following table provides complete information about all the datatypes in c programming language...

	Integer	Floating Point	Double	Character
What is it?	Numbers without decimal value	Numbers with decimal value	Numbers with decimal value	Any symbol enclosed in single quotation
Keyword	int	float	double	char
Memory Size	2 or 4 Bytes	4 Bytes	8 or 10 Bytes	1 Byte
Range	-32768 to +32767 (or) 0 to 65535 (Incase of 2 bytes only)	1.2E - 38 to 3.4E + 38	2.3E-308 to 1.7E+308	-128 to + 127 (or) 0 to 255
Type Specifier	%d or %i or %u	%f	%ld	%c or %s
Type Modifier	short, long signed, unsigned	No modifiers	long	signed, unsigned
Type Qualifier	const, volatile	const, volatile	const, volatil	const, volatile

void Datatype

The void datatype means nothing or no value. Generally, void is used to specify a function which does not return any value. We also use the void datatype to specify empty parameters of a function.

Enumerated Datatype

An enumerated datatype is a user-defined data type that consists of integer constants and each integer constant is given a name. The keyword "**enum**" is used to define enumerated datatype.

Derived Datatypes

Derived datatypes are user-defined data types. The derived datatypes are also called as user defined datatypes or secondary datatypes. In c programming language, the derived datatypes are created using the following concepts...

- Arrays
- Structures
- Unions
- Enumeration

Faculty Name: N.SANDHYA RANI **Subject** :PPS **Topic:** data inputs, output statements	**Unit No** :2 **Lecture No** :L19 **Book Reference**:T1

Input Functions in C:

C programming language provides built-in functions to perform input operations. The input opeartions are used to read user values (input) from keyboard. C programming language provides the following built-in input functions...

1. **scanf()**
2. **getchar()**
3. **getch()**

4. **gets()**
5. **fscanf()**

scanf() function:

The scanf() function is used to read multiple data values of different data types from the keyboard. The scanf() function is built-in function defined in a header file called "**stdio.h**". When we want to use scanf() function in our program, we need to include the respective header file (stdio.h) using **#include** statement. The scanf() function has the following syntax...

Syntax:

```
scanf("format strings",&variableNames);
```

Example Program

```c
#include <stdio.h>

void main(){

 int i;

  printf("\nEnter any integer value: ");

scanf("%d",&i);

 printf("\nYou have entered %d number",i);

}
```

Output:

Enter any integer value: 55
You have entered 55 number

In the above example program, we used the scanf() function to read an integer value from the keyboard and store it into variable 'i'.

The scanf function also used to read multiple data values of different or same data types. Consider the following example program...

```c
#include <stdio.h>

void main(){

int i;

float x;
```

```
printf("\nEnter one integer followed by one float value : ");

scanf("%d%f",&i, &x);

printf("\ninteger = %d, float = %f",i, x);   }
```

Output:

Enter one integer followed by one float value : 20 30.5
integer = 20, float = 30.5

In the above example program, we used the scanf() function to read one integer value and one float value from the keyboard. Here **'i'** is an integer variable so we have used format string %d, and **'x'** is a float variable so we have used format string %f.

The scanf() function returns an integer value equal to the total number of input values read using scanf function.

Example Program

```
#include <stdio.h>

void main(){

  int i,a,b;

  float x;

  printf("\nEnter two integers and one float : ");

  i = scanf("%d%d%f",&a, &b, &x);

  printf("\nTotal inputs read : %d",i);

}
```

Output:

Enter two integers and one float : 10 20 55.5
Total inputs read : 3

getchar() function

The getchar() function is used to read a character from the keyboard and return it to the program. This function is used to read only single character. To read multiple characters we need to write multiple times or use a looping statement. Consider the following example program...

```c
#include <stdio.h>

void main(){

  char ch;

  printf("\nEnter any character : ");

  ch = getchar();

  printf("\nYou have entered : %c",ch);

}
```

Output:

```
Enter any character : A
You have entered : A
```

getch() function

The getch() function is similar to getchar function. The getch() function is used to read a character from the keyboard and return it to the program. This function is used to read only single character. To read multiple characters we need to write multiple times or use a looping statement. Consider the following example program...

```c
#include <stdio.h>

void main(){

  char ch;

  printf("\nEnter any character : ");

  ch = getch();

  printf("\nYou have entered : %c",ch);

}
```

Output:

```
Enter any character :
You have entered : A
```

gets() function

The gets() function is used to read a line of string and stores it into character array. The gets() function reads a line of string or sequence of characters till a newline symbol enters. Consider the following example program...

```
#include <stdio.h>

void main(){

  char name[30];

  printf("\nEnter your favourite website: ");

  gets(name);

  printf("%s",name);

}
```

Output:

Enter your favourite website: www.btechsmartclass.com

fscanf() function

The fscanf() function is used with the concept of files. The fscanf() function is used to read data values from a file. When you want to use fscanf() function the file must be opened in reading mode.

Output Functions in C:

C programming language provides built-in functions to perform output operation. The output opearations are used to display data on user screen (output screen) or printer or any file. C programming language provides the following built-in output functions...

printf()

putchar()

puts()

fprintf()

printf() function

The printf() function is used to print string or data values or combination of string and data values on the output screen (User screen). The printf() function is built-in function defined in a header file called "**stdio.h**". When we want to use printf() function in our program we need to include the respective header file (stdio.h) using **#include** statement. The printf() function has the following syntax...

Syntax:

printf("message to be display!!!");

Example Program

#include <stdio.h>

void main(){

   printf("Hello! Welcome to btechsmartclass!!!");

}

Output:

Hello! Welcome to btechsmartclass!!!

In the above example program, we used the printf() function to print a string on to the output screen.

The printf() function is also used to display data values. When we want to display data values we use **format string** of the data value to be display.

Syntax:

printf("format string",variableName);

Example Program

#include <stdio.h>

void main(){

   int i = 10;

   float x = 5.5;

```
printf("%d %f",i, x);   }
```

Output:

10 5.5

In the above example program, we used the printf() function to print data values of variables i and x on to the output screen. Here i is a integer variable so we have used format string %d and x is a float variable so we have used format string %f.

The printf() function can also used to display string along with data values.

Syntax:

printf("String format string",variableName);

Example Program

```
#include <stdio.h>

void main(){

   int i = 10;

   float x = 5.5;

   printf("Integer value = %d, float value = %f",i, x);


}
```

Output:

Integer value = 10, float value = 5.5

In the above program we are displaying string along with data values.

Every function in C programming language must have a return value. The printf() function also have integer as return value. The printf() function returns an integer value equalent to the total number of characters it has printed.

Example Program

#include <stdio.h>

```
void main(){

    int i;

    i = printf("btechsmartclass");

    printf(" is %d number of characters.",i);   }
```

Output:

btechsmartclass is 15 number of characters.

In the above program, first printf() function printing "btechsmartclass" which is of 15 characters. So it returns integer value 15 to variable "i". The value of "i" is printed in the second printf() function.

Formatted printf() function

Generally, when we write multiple printf() statements the result is displayed in single line because the printf() function displays the output in a single line. Consider the following example program...

```
#include <stdio.h>

void main(){

    printf("Welcome to ");

    printf("btechsmartclass ");

    printf("the perfect website for learning");

}
```

Output:

Welcome to btechsmartclass the perfect website for learning

In the above program, there are 3 printf() statements written in different lines but the output is displayed in single line only.

To display the output in different lines or as we wish, we use some special characters called escape sequences. Escape sequences are special characters with special functionality used in printf() function to format the output according to the user requirement. In C programming language, we have the following escape sequences...

Escape Sequence	Meaning
\n	New line
\t	Horizontal Tab
\v	Vertical Tab
\a	Beep sound
\b	Backspace
\\	Backward slash
\?	Question mark
\'	Single quotation mark
\"	Double quotation mark

Consider the following example program...

```
#include <stdio.h>
void main(){
  printf("Welcome to\n");
  printf("btechsmartclass\n");
  printf("the perfect website for learning");
}
```

Output:

Welcome to
btechsmartclass
the perfect website for learning

putchar() function

The putchar() function is used to display single character on the output screen. The putchar() functions prints the character which is passed as parameter to it and returns the same character as return value. This function is used to print only single charater. To print multiple characters we need to write multiple times or use a looping statement. Consiider the following example program...

```
#include <stdio.h>

void main(){

    char ch = 'A';

    putchar(ch);
```

     }

Output:

A

puts() function

The puts() function is used to display string on the output screen. The puts() functions prints a string or sequence of characters till the newline. Consiider the following example program...

#include <stdio.h>

void main(){

 char name[30];

 printf("\nEnter your favourite website: ");

 gets(name);

 puts(name);

}

Output:

Enter your favourite website: www.btechsmartclass.com
www.btechsmartclass.com

fprintf() function

The fprintf() function is used with the concept of files. The fprintf() function is used to print a line into the file. When you want to use fprintf() function the file must be opened in writting mode.

Operators:

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.

C operators can be classified into a number of categories, they are
1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise Operators
8. Special operators

1. Arithmetic Operators:

The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.

Operator	Meaning	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5
*	Multiplication	5 * 10 = 50
/	Division	10 / 5 = 2
%	Remainder of Division	5 % 2 = 1

The addition operator can be used with numerical data types and character data type. When it is used with numerical values, it performs mathematical addition and when it is used with character data type values, it performs concatenation (appending).

The remainder of division operator is used with integer data type only.

2.Relational Operators :

The relational operators are the symbols that are used to compare two values. That means, the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE

3. Logical Operators:

The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.

Operator	Meaning	Example
&&	**Logical AND** - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
||	**Logical OR** - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 || 12 > 10 is TRUE
!	**Logical NOT** - Returns TRUE if condition is FLASE and returns FALSE if it is TRUE	!(10 < 5 && 12 > 10) is TRUE

Logical AND - Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.
Logical OR - Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

4. Assignment Operators :

The assignment operators are used to assign right hand side value (Rvalue) to the left hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The following table describes all the assignment operators in C programming language.

Operator	Meaning	Example
=	Assign the right hand side value to left hand side variable	A = 15
+=	Add both left and right hand side values and store the result into left hand side variable	A += 10 ⇒ A=A+10
-=	Subtract right hand side value from left hand side variable value and store the result into left hand side variable	A -= B ⇒ A=A-B
*=	Multiply right hand side value with left hand side variable value and store the result into left hand side variable	A *= B ⇒ A=A*B
/=	Divide left hand side variable value with right hand side variable value and store the result into left hand side variable	A /= B ⇒ A=A/B
%=	Divide left hand side variable value with right hand side variable value and store the remainder into left hand side variable	A %= B ⇒ A=A%B

5.Increment & Decrement Operators:

The increment and decrement operators are called as unary operators because, both needs only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.

Operator	Meaning	Example
++	**Increment** - Adds one to existing value	int a = 5; a++; ⇒ a = 6
--	**Decrement** - Subtracts one from existing value	int a = 5; a--; ⇒ a = 4

The increment and decrement operators are used infront of the operand (++a) or after the operand (a++). If it is used infront of the operand, we call it as pre-increment or pre-decrement and if it is used after the operand, we call it as post-increment or post-decrement.

Pre-Increment or Pre-Decrement

In case of pre-increment, the value of the variable is increased by one before the expression evaluation. In case of pre-decrement, the value of the variable is decreased by one before the expression evaluation. That means, when we use pre increment or pre decrement, first

the value of the variable is incremented or decremented by one, then modified value is used in the expression evaluation.

Example:
```
#include <stdio.h>
void main()
{
    int i = 5,j;
    j = ++i; // Pre-Increment
    printf("i = %d, j = %d",i,j);
}
```

Output:

```
i = 6, j = 6
```

Post-Increment or Post-Decrement

In case of post-increment, the value of the variable is increased by one after the expression evaluation. In case of post-decrement, the value of the variable is decreased by one after the expression evaluation. That means, when we use post-increment or post-decrement, first the expression is evaluated with existing value, then the value of the variable is incremented or decremented by one.

Example
```
#include <stdio.h>
void main()
{
  int i = 5,j;
  j = i++;  // Post-Increment
   printf("i = %d, j = %d",i,j);
}
```

Output:

```
i = 6, j = 5
```

6.Conditional Operator (?:)

The conditional operator is also called as ternary operator because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax...

Condition ? TRUE Part : FALSE Part ;

Example:

A = (10<15) ? 100 : 200 ; ⇒ A value is 100

7.Bitwise Operators:

The bitwise operators are used to perform bit level operations in c programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in C programming language.

Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100)

Operator	Meaning	Example
&	the result of **Bitwise AND** is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
|	the result of **Bitwise OR** is 0 if all the bits are 0 otherwise it is 1	A | B ⇒ 29 (11101)
^	the result of **Bitwise XOR** is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of **Bitwise once complement** is nagation of the bit (Flipping)	~A ⇒ 6 (00110)
<<	the **Bitwise left shift** operator shifts all the bits to the left by specified number of positions	A<<2 ⇒ 100 (1100100)
>>	the **Bitwise right shift** operator shifts all the bits to the right by specified number of positions	A>>2 ⇒ 6 (00110)

8.Special Operators (sizeof, pointer, comma, dot etc.)

The following are the special operators in c programming language.

sizeof operator

This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator is used with the following syntax...

sizeof(variableName);

Example

sizeof(A); ⇒ result is 2 if A is an integer

Pointer operator (*)

This operator is used to define pointer variables in c programming language.

Comma operator (,)

This operator is used to separate variables while they are declaring, separate the expressions in function calls etc..

Dot operator (.)

This operator is used to access members of structure or union.

Operator Precedence and Associativity

Explain Operator Precedence

Operator precedence is used to determine the order of operators evaluated in an expression. In c programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with least precedence is evaluated last.

Discuss about Operator Associativity

Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression. In c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.

In c programming language the operator precedence and associativity is as shown in the following table...

Precedence	Operator	Operator Meaning	Associativity
1	()	function call	Left to Right
	[]	array reference	
	->	structure member access	
	.	structure member access	
2	!	negation	Right to Left
	~	1's complement	
	+	Unary plus	
	-	Unary minus	
	++	increment operator	
	--	decrement operator	
	&	address of operator	
	*	pointer	
	sizeof	returns size of a variable	
	(type)	type conversion	

3	* / %	multiplication division remainder	Left to Right
4	+ -	addition subtraction	Left to Right
5	<< >>	left shift right shift	Left to Right
6	< <= > >=	less than less than or equal to greater than greater than or equal to	Left to Right
7	== !=	equal to not equal to	Left to Right
8	&	bitwise AND	Left to Right
9	^	bitwise EXCLUSIVE OR	Left to Right
10	|	bitwise OR	Left to Right
11	&&	logical AND	Left to Right
12	||	logical OR	Left to Right
13	?:	conditional operator	Left to Right
14	= *= /= %= += -= &= ^= |= <<= >>=	assignment assign multiplication assign division assign remainder assign additon assign subtraction assign bitwise AND assign bitwise XOR assign bitwise OR assign left shift assign right shift	Right to Left
15	,	separator	Left to Right

In the above table, the operator precedence decrease from top to bottom and increase from bottom to top.

Faculty Name:N.SANDHYA
RANI   **Subject** :PPS
Topic: evaluation of expressions

Unit No   :2
Lecture No   :L22
Book Reference:T1

Expressions:

Define an Expression

In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.

In C programming language, an expression is defined as follows...

An expression is a collection of operators and operands that represents a specific value.

In the above definition, operator is a symbol which performs tasks like arithmetic operations, logical operations and conditional operations etc.,
Operands are the values on which the operators perform the task. Here operand can be a direct value or variable or address of memory location.

Expression Types in C

In C programming language, expressions are divided into THREE types. They are as follows...

1.Infix Expression
2.Postfix Expression
3.Prefix Expression

The above classification is based on the operator position in the expression.
1.Infix Expression

The expression in which operator is used between operands is called as infix expression.
The infix expression has the following general structure...

Operand1 Operator Operand2
Example

Operand1   Operator   Operand2

$(a+b)$
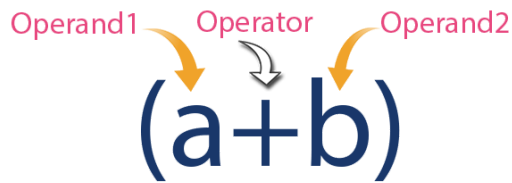
2.Postfix Expression

The expression in which operator is used after operands is called as postfix expression.

The postfix expression has the following general structure...

Operand1 Operand2 Operator

Example:

Operand1   Operand2   Operator

ab+

3.Prefix Expression

The expression in which operator is used before operands is called as prefix expression.

The prefix expression has the following general structure...

Operator Operand1 Operand2

Example:

Operator   Operand1   Operand2

+ab

Expression Evaluation in C

In C programming language, expression is evaluated based on the operator precedence and associativity. When there are multiple operators in an expression, they are evaluated according to their pr

ecedence and associativity. The operator with higher precedence is evaluated first and the operator with least precedence is evaluated last.

An expression is evaluated based on the precedence and associativity of the operators in that expression.

To understand expression evaluation in c, let us consider the following simple example expression...

$$10 + 4 * 3 / 2$$

In the above expression there are three operators +, * and /. Among these three operators, both multiplication and division have same higher precedence and addition has lower precedence. So, according to the operator precedence both multiplication and division are evaluated first and then addition is evaluated. As multiplication and division have same precedence they are evaluated based on the associativity. Here, the associativity of multiplication and division is left to right. So, multiplication is performed first, then division and finally addition. So, the above expression is evaluated in the order of * / and +. It is evaluated as follows...

4 * 3 ====> 12
12 / 2 ===> 6
10 + 6 ===> 16
The expression is evaluated to **16**.

Faculty Name: N.SANDHYA
RANI   **Subject** :PPS
Topic: Type Conversions In
Exprssions

Unit No   :2
Lecture No   :L23
Book Reference:T1

Type Conversions

In a programming language, the expression contains data values of same datatype or different datatypes. When the expression contains similar datatype values then it is evaluated without any problem. But if the expression contains two or more different datatype values then they must be converted to single datatype of destination datatype. Here, destination is the location where the final result of that expression is stored. For example,the multiplication of an integer data value with float data value and storing the result into a float variable. In this case, the integer value must be converted to float value so that the final result is a float datatype value.

In c programming language, the data conversion is performed in two different methods as follows.
Type Conversion
Type Casting

Type Conversion

The type conversion is the process of converting a data value from one datatype to another datatype automatically by the compiler. Sometimes type conversion is also called as **implicit type conversion**. The implicit type conversion is automatically performed by the compiler.
For example, in c programming language, when we assign an integer value to a float variable the integer value automically gets converted to float value by adding decimal value 0. And when a float value is assigned to an integer variable the float value automatically gets converted to integer value by removing the decimal value. To understand more about type conversion observe the following...
int i = 10 ;
float x = 15.5 ;
char ch = 'A' ;

i = x ; =======> x value 15.5 is converted as 15 and assigned to variable i

x = i ; =======> Here i value 10 is converted as 10.000000 and assigned to variable x

i = ch ; =======> Here the ASCII value of A (65) is assigned to i
Example Program
#include <stdio.h>
void main()
{
  int i = 95 ;
  float x = 90.99 ;

```
  char ch = 'A' ;
   i = x ;
  printf("i value is %d\n",i);
  x = i ;
  printf("x value is %f\n",x);
  i = ch ;
  printf("i value is %d\n",i);
}
```
Output:
i value is 90
x value is 90.000000
i value is 65

In the above program, we assign **i = x**, i.e., float variable value is assigned to integer variable. Here, the compiler automatically converts the float value (90.99) into integer value (90) by removing the decimal part of the float value (90.99) and then it is assigned to variable **i**. Similarly when we assign **x = i**, the integer value (90) gets converted to float value (90.000000) by adding zero as decimal part.

Type Casting

Type casting is also called as **explicit type conversion**. Compiler converts data from one datatype to another datatype implicitly. When compiler converts implicitly, there may be a data loss.In such case, we convert the data from one datatype to another datatype using explicit type conversion. To perform this we use the **unary cast operator**. To convert data from one type to another type we specify the target datatype in paranthesis as a prefix to the data value that has to be converted. The general syntax of type casting is as follows...

(TargetDatatype) DataValue

Example

```
int totalMarks = 450, maxMarks = 600 ;
float average ;
```

average = (float) totalMarks / maxMarks * 100 ;

In the above example code, both totalMarks and maxMarks are integer data values. When we perform totalMarks / maxMarks the result is a float value, but the destination (average) datatype is float. So we use type casting to convert totalMarks and maxMarks into float datatype.

Example Program

```
#include <stdio.h>
void main(){
  int a, b, c ;
  float avg ;
  printf("Enter any three integer values : ") ;
```

```
    scanf("%d%d%d", &a, &b, &c) ;
    avg = (a + b + c) / 3 ;
    printf("avg before casting = %f\n",avg);
    avg = (float)(a + b + c) / 3 ;
    printf("avg after casting = %f\n",avg);
}
```

Output:

Enter any three integer values : 5 3 2
avg before casting = 3
avg after casting = 3.333333

Control structures: Decision statements; if and switch statement; Loop control statements: while, for and do while loops, jump statements, break, continue, goto statements.

Decision Making statements:

In c programming language, the program execution flow is, line by line from top to bottom. That means the c program is executed line by line from the main method. But this type of execution flow may not be suitable for all the program solutions. Sometimes, we make some decisions or we may skip the execution of one or more lines of code. Consider a situation, where we write a program to check whether a student has passed or failed in a particular subject. *Here,* we need to check whether the marks are greater than the pass marks or not. If marks are greater, then we take the decision that the student has passed otherwise failed. To solve such kind of problems in c we use the statements called decision making statements.

Decision making statements are the statements that are used to verify a given condition and decides whether a block of statements gets executed or not based on the condition result.

In c programming language, there are two decision making statements they are as follows...
1. if statement
2. switch statement

1. if statement in c

In c, if statement is used to make decisions based on a condition. The if statement verifies the given condition and decides whether a block of statements are executed or not based on the condition result. In c, if statement is classified into four types as follows...

1. Simple if statement
2. if - else statement
3. Nested if statement
4. if-else-if statement (if-else ladder)
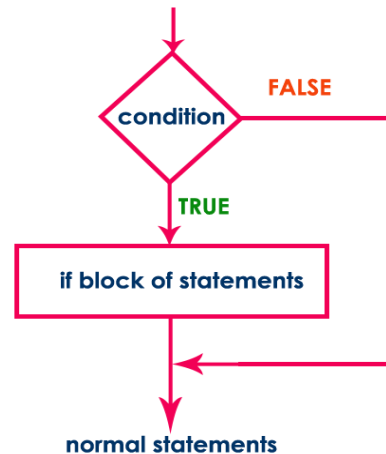
1. Simple if statement

Simple if statement is used to verify the given condition and executes the block of statements based on the condition result. The simple if statement evaluates specified condition. If it is TRUE, it executes the next statement or block of statements. If the condition is FALSE, it

skips the execution of the next statement or block of statements. The general syntax and execution flow of the simple if statement is as follows...

Syntax

Execution flow diagram

```
if ( condition )
{
    ....
    block of statements;
    ....
}
```


Simple if statement is used when we have only one option that is executed or skipped based on a condition.

Example Program | Test whether given number is divisible by 5.

```c
#include <stdio.h>
#include<conio.h>
void main(){
  int n ;
  clrscr() ;
  printf("Enter any integer number: ") ;
  scanf("%d", &n) ;
  if ( n%5 == 0 )
     printf("Given number is divisible by 5\n") ;
  printf("statement does not belong to if!!!") ;
}
```

Output 1:
Enter any integer number: 100
Given number is divisible by 5
statement does not belong to if!!!

Output 2:
Enter any integer number: 99
statement does not belong to if!!!

2. if - else statement

The if - else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result. The if-else statement evaluates the specified condition. If it is TRUE, it executes a block of statements (True block). If the condition is FALSE, it executes another block of statements (False block). The general syntax and execution flow of the if-else statement is as follows...

Syntax

```
if ( condition )
{
   ....
   True block of statements;
   ....
}
else
{
   ....
   False block of statements;
   ....
}
```

Execution flow diagram
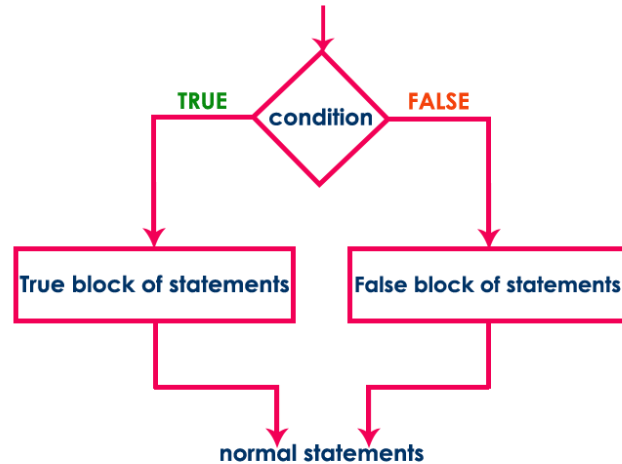
The if-else statement is used when we have two options and only one option has to be executed based on a condition result (TRUE or FALSE).

Example Program | Test whether given number is even or odd.

```c
#include <stdio.h>
#include<conio.h>
void main(){
  int n ;
  clrscr() ;
  printf("Enter any integer number: ") ;
  scanf("%d", &n) ;
  if ( n%2 == 0 )
    printf("Given number is EVEN\n") ;
  else
    printf("Given number is ODD\n") ;
}
```

Output 1:
Enter any integer number: 100
Given number is EVEN

Output 2:
Enter any integer number: 99
Given number is ODD

3. Nested if statement
Writing a if statement inside another if statement is called nested if statement. The general syntax of the nested if statement is as follows...

Syntax

```
if ( condition1 )
{
    if ( condition2 )
    {
        ....
        True block of statements 1;
    }
    ....
}
else
{
    False block of condition1;
}
```

The nested if statement can be defined using any combination of simple if & if-else statements. The flow of control using Nested if...else statement is determined as follows
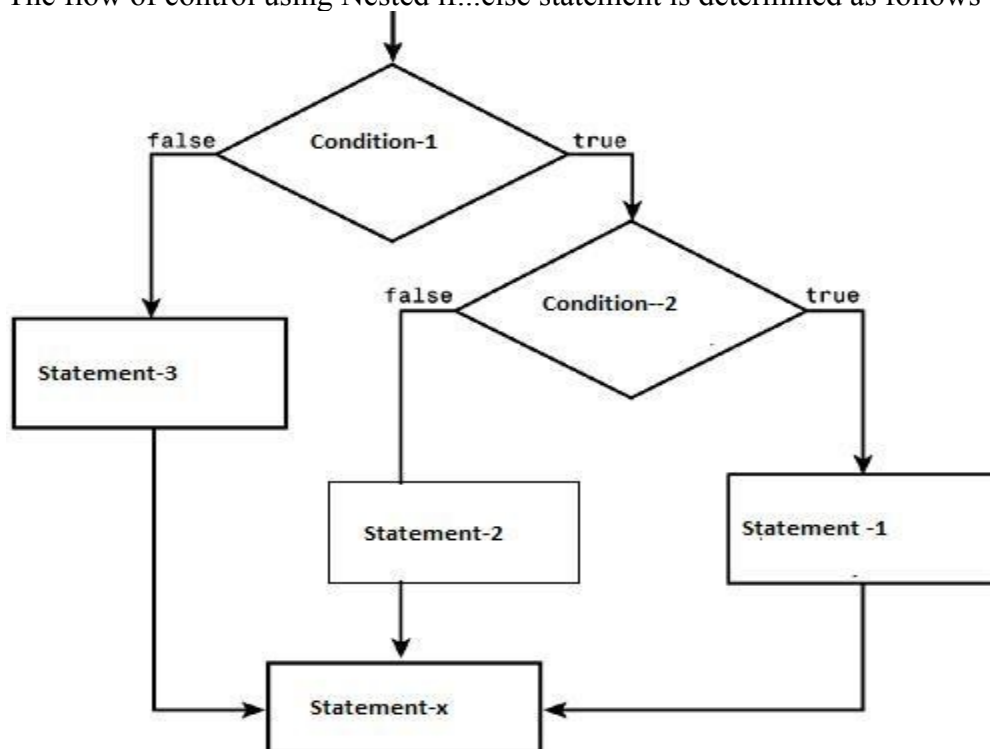
Whenever nested if…else statement is encountered, first <condition1> is tested. It returns either true or false.

If condition1 (or outer condition) is false, then the control transfers to else-body (if exists) by skipping if-body.

If condition1 (or outer condition) is true, then condition2 (or inner condition) is tested. If the condition2 is true, if-body gets executed. Otherwise, the else-body that is inside of if statement gets executed.

Example Program | Test whether given number is even or odd if it is below 100.

```c
#include <stdio.h>
#include<conio.h>
void main(){
  int n ;
  clrscr() ;
  printf("Enter any integer number: ") ;
  scanf("%d", &n) ;
  if ( n < 100 )
  {
    printf("Given number is below 100\n") ;
    if( n%2 == 0)
      printf("And it is EVEN") ;
    else
      printf("And it is ODD") ;
  }
  else
    printf("Given number is not below 100") ;
}
```

Output 1:

Enter any integer number: 55
Given number is below 100
And it is ODD

Output 2:

Enter any integer number: 999
Given number is not below 100

4. if - else - if statement (if-else ladder)

Writing a if statement inside else of a if statement is called if - else - if statement. The general syntax of the if-else-if statement is as follows...

Syntax

```
if ( condition1 )
{
    ....
    True block of statements1;
    ....
}
else if ( condition2 )
{
    False block of condition1;
    &
    True block of condition2
}
```

The if-else-if statement can be defined using any combination of simple if & if-else statements. The flow of control using else--- if ladder statement is determined as follows:
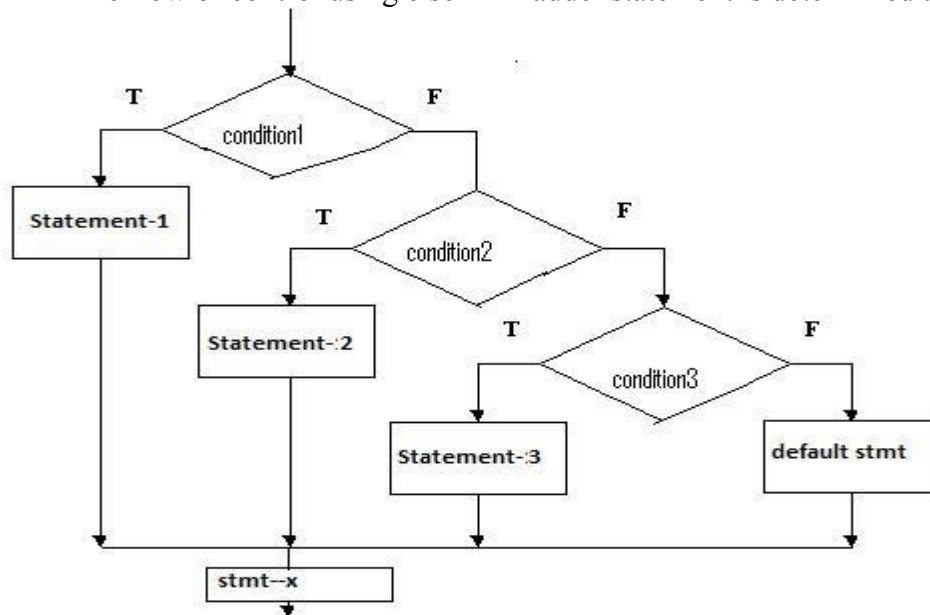
Whenever else if ladder is encountered, condition1 is tested first. If it is true, the statement 1 gets executed. After then the control transfers to *stmt-x*.

If *condition1* is false, then *condition2* is tested. If *condition2* is false, the other conditions are tested. If all are false, the default stmt at the end gets executed. After then the control transfers to *stmt-x*.

If any one of all conditions is true, then the body associated with it gets executed. After then the control transfers to *stmt-x*.

Example Program | Find the largest of three numbers.

```c
#include <stdio.h>
#include<conio.h>
void main(){
  int a, b, c ;
  clrscr() ;

  printf("Enter any three integer numbers: ") ;
  scanf("%d%d%d", &a, &b, &c) ;

  if( a>=b && a>=c)
     printf("%d is the largest number", a) ;

  else if (b>=a && b>=c)
     printf("%d is the largest number", b) ;

  else
     printf("%d is the largest number", c) ;
}
```

Output 1:
Enter any three integer numbers: 55 60 20
60 is the largest number
MOST IMPORTANT POINTS TO BE REMEMBERED
When we use conditional control statement like if statement, condition might be an expression evaluated to a numerical value, a variable or a direct numerical value. If the expression value or direct value is zero the conditon becomes FALSE otherwise becomes TRUE.

To understand more consider the following statements...
if(10) - is TRUE
if(x) - is FALSE if x value is zero otherwise TRUE
if(a+b) - is FALSE if a+b value is zero otherwise TRUE
if(a = 99) - is TRUE because a value is non-zero
if(10, 5, 0) - is FALSE because it considers last value
if(0) - is FALSE
if(a=10, b=15, c=0) - is FALSE because last value is zero

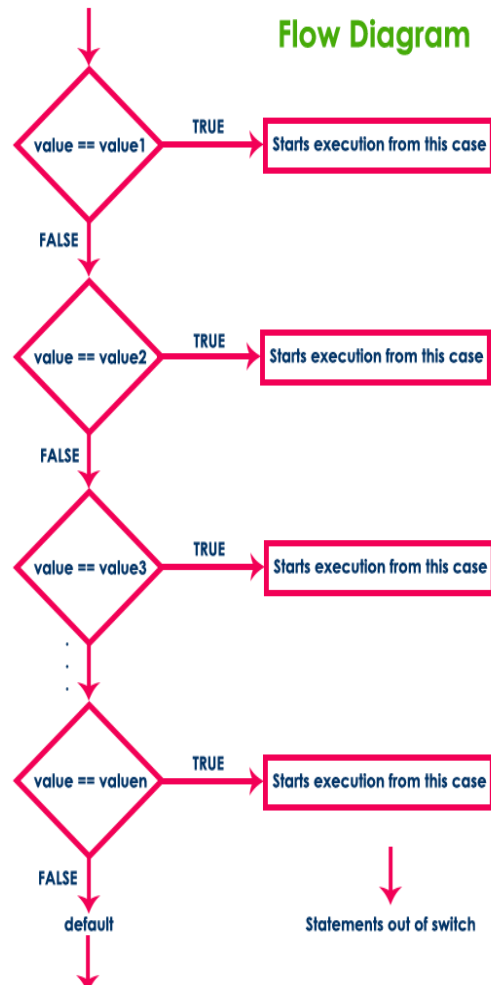|	
Faculty Name: N.SANDHYA RANI **Subject** :PPS **Topic:** Switch Statement	**Unit No** :2 **Lecture No** :L25 **Book Reference**:T1

Switch statement:

Consider a situation in which we have more number of options out of which we need to select only one option that is to be executed. Such kind of problems can be solved using **nested if** statement. But as the number of options increases, the complexity of the program also gets increased. This type of problems can be solved very easily using **switch** statement. Using switch statement, one can select only one option from more number of options very easily. In switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches with the value associated with an option, the execution starts from that option. In switch statement every option is defined as a **case**.

The switch statement has the following syntax and execution flow diagram...

Flow Diagram

Syntax

```
switch ( expression or value )
{
    case  value1:  set of statements;
                    ....
    case  value2:  set of statements;
                    ....
    case  value3:  set of statements;
                    ....
    case  value4:  set of statements;
                    ....
    case  value5:  set of statements;
                    ....
    .
    .
    default: set of statements;
}
```

value == value1 — TRUE → Starts execution from this case

FALSE

value == value2 — TRUE → Starts execution from this case

FALSE

value == value3 — TRUE → Starts execution from this case

value == valuen — TRUE → Starts execution from this case

FALSE

default

Statements out of switch

The switch statement contains one or more number of cases and each case has a value associated with it. At first switch statement compares the first case value with the switchValue, if it gets matched the execution starts from the first case. If it doesn't match the switch statement compares the second case value with the switchValue and if it is matched the execution starts from the second case. This process continues until it finds a match. If no case value matches with the switchValue specified in the switch statement, then a special case called **default** is executed. When a case value matches with the switchValue, the execution starts from that particular case. This execution flow continues with next case statements also. To avoid this, we use "**break**" statement at the end of each case. That means the **break** statement is used to terminate the switch statement. However it is optional.

Example Program | Display pressed digit in words.

```c
#include <stdio.h>
#include<conio.h>
void main(){
  int n ;
  clrscr() ;

  printf("Enter any digit: ") ;
  scanf("%d", &n) ;

  switch( n )
  {
          case 0: printf("ZERO") ;
          break ;
          case 1: printf("ONE") ;
          break ;
          case 2: printf("TWO") ;
          break ;
          case 3: printf("THREE") ;
          break ;
          case 4: printf("FOUR") ;
          break ;
          case 5: printf("FIVE") ;
          break ;
          case 6: printf("SIX") ;
          break ;
          case 7: printf("SEVEN") ;
          break ;
          case 8: printf("EIGHT") ;
          break ;
          case 9: printf("NINE") ;
          break ;
          default: printf("Not a Digit") ;
  }
  getch() ;
}
```

Output 1:

Enter any digit: 5

FIVE
Output 2:
Enter any digit: 15
Not a Digit

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use switch statement, we must follow the following...
Both switch and case are keywords so they must be used only in lower case letters.
The data type of case value and the value specified in switch statement must be same.
switch and case values must be either integer or character but not float or string.
A switch statement can contain any number of cases.
The keyword case and its value must be separated with a white space.
The case values need not be defined in sequence, they can be in any order.
The default case is optional and it can be defined anywhere inside the switch statement.
The switch value might be a direct value, a variable or an expression.

<table>
<tr><td>

Faculty Name: N.SANDHYA
RANI   **Subject** :PPS
Topic: Loop Control Statements
:While

</td><td>

Unit No   :2
Lecture No   :L26
Book Reference: T1

</td></tr>
</table>

Loop control statements:
        Consider a situation in which we execute a single statement or block of statements repeatedly for required number of times. Such kind of problems can be solved using **looping** statements in C. For example, assume a situation where we print a message for 100 times. If we want to perform that task without using looping statements, we have to either write 100 printf statements or we have to write the same message for 100 times in a single printf statement. Both are complex methods. The same task can be performed very easily using looping statements.
The looping statements are used to execute a single statement or block of statements repeatedly until the given condition is FALSE.

C language provides three looping statements...
1.while statement
*2.do-while statement*
3.for statement

1. While statement:

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as **Entry control looping statement**. The while statement has the following syntax...

Syntax:

```
while( condition )
{
    ...
    block of statements;
    ...
}
```

The while statement has the following execution flow diagram...
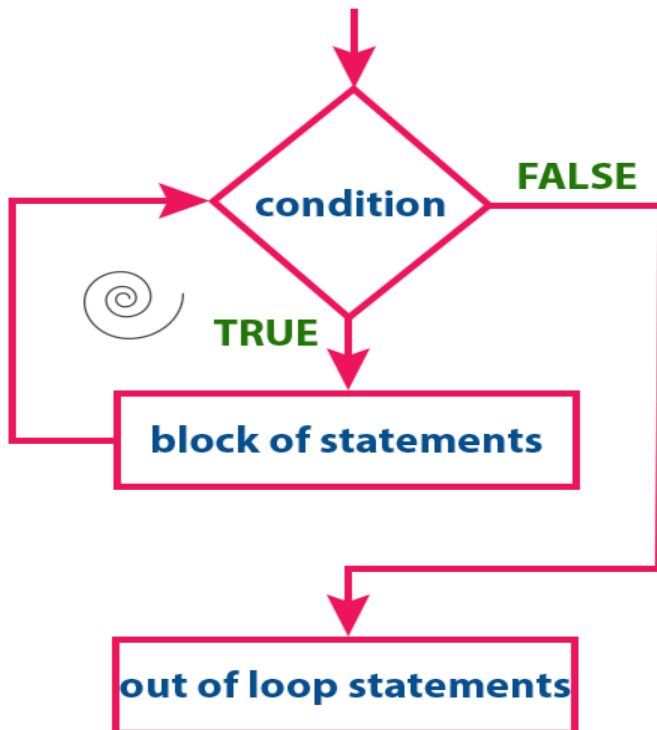
At first, the given condition is evaluated. If the condition is TRUE, the single statement or block of statements gets executed. Once the execution gets completed the condition is evaluated again. If it is TRUE, again the same statements gets executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the while block.

Example Program | Program to display even numbers upto 10.

```
#include <stdio.h>
#include<conio.h>
void main(){
  int n = 0;
  clrscr() ;
  printf("Even numbers upto 10\n");

  while( n <= 10 )
  {
    if( n%2 == 0)
      printf("%d\t", n) ;
    n++ ;
  }

  getch() ;
}
```

Output 1:
Even numbers upto 10
0 2 4 6 8 10

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use while statement, we must follow the following...
while is a keyword so it must be used only in lower case letters.
If the condition contains variable, it must be assigned a value before it is used.
The value of the variable used in condition must be modified according to the requirement inside the while block.
In while statement, the condition may be a direct integer value, a variable or a condition.
A while statement can be an empty statement.

Faculty Name: N.SANDHYA RANI   **Subject** :PPS
Topic: Loop Control Statements :
Do-While ,for

Unit No   :2
**Lecture No**   :L27
Book Reference: T1

2. do-While statement:
The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as **Exit control looping statement**. The do-while statement has the following syntax...

Syntax:

```
do
{
    ...
    block of statements;
    ...
} while( condition ) ;
```

The do-while statement has the following execution flow diagram.
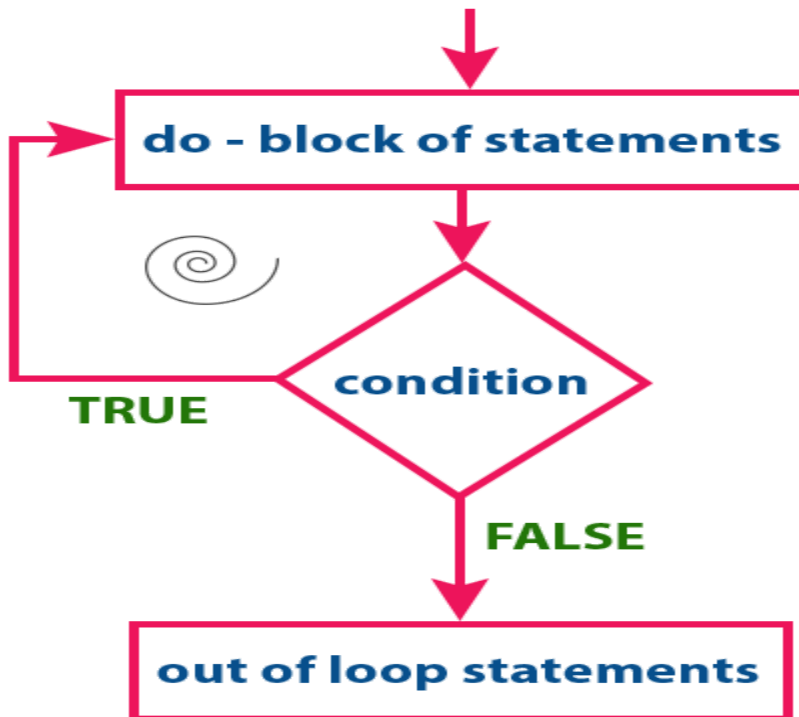
At first, the single statement or block of statements which are defined in **do** block are executed. After execution of do block, the given condition gets evaluated. If the condition is evaluated to TRUE, the single statement or block of statements of do block are executed again. Once the execution gets completed again the condition is evaluated. If it is TRUE, again the same statements are executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the while block.

Example Program | Program to display even numbers upto 10.

```
#include <stdio.h>
#include<conio.h>
void main(){
   int n = 0;
   clrscr() ;
   printf("Even numbers upto 10\n");

   do
   {
      if( n%2 == 0)
         printf("%d\t", n) ;
      n++ ;
   }while( n <= 10 ) ;

   getch() ;
}
```

Output 1:

Even numbers upto 10
0 2 4 6 8 10

MOST IMPORTANT POINTS TO BE REMEMBERED
When we use do-while statement, we must follow the following...
Both do and while are keywords so they must be used only in lower case letters.
If the condition contains variable, it must be assigned a value before it is used.
The value of the variable used in condition must be modified according to the requirement inside the do block.
In do-while statement the condition may be, a direct integer value, a variable or a condition.
A do-while statement can be an empty statement.
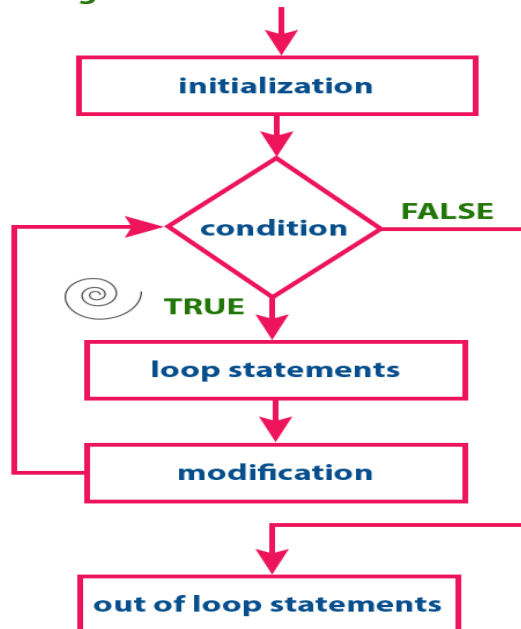In do-while, the block of statements are executed at least once.

3.for statement:
The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE. The for statement has the following syntax and execution flow diagram...

Syntax:
```
for( initialization ; condition ; modification )
{
    ...
    block of statements;
    ...
}
```

Execution flow diagram:

At first, the for statement executes initialization followed by condition evaluation. If the condition is evaluated to TRUE, the single statement or block of statements of for statement are executed. Once the execution gets completed, the modification statement is executed and again the condition is evaluated. If it is TRUE, again the same statements are executed. The same process is repeated until the condition is evaluated to FALSE. Whenever the condition is evaluated to FALSE, the execution control moves out of the for block.

Example Program | Program to display even numbers upto 10.

```c
#include <stdio.h>
#include<conio.h>
void main(){
   int n ;
   clrscr() ;
   printf("Even numbers upto 10\n");

   for( n = 0 ; n <= 10 ; n++ )
   {
      if( n%2 == 0)
         printf("%d\t", n) ;
   }

   getch() ;
}
```

Output 1:
Even numbers upto 10
0 2 4 6 8 10

MOST IMPORTANT POINTS TO BE REMEMBERED
When we use for statement, we must follow the following.
for is a keyword so it must be used only in lower case letters.
Every for statement must be provided with initialization, condition and modification (They can be empty but must be separated with ";")
Ex: for (; ;) or for (; condition ; modification) or for (; condition ;)
In for statement, the condition may be a direct integer value, a variable or a condition.
The for statement can be an empty statement.

Faculty Name: N.SANDHYA RANI **Subject** : PPS **Topic:** Braeak, Jump , Continue , goto statement	**Unit No** :2 **Lecture No** :L28 **Book Reference**: T1

Break,continue and goto statements:
In c, there are control statements which does not need any condition to control the program execution flow. These control statements are called as **unconditional control statements**. C programming language provides the following unconditional control statements...
1.break
2.continue
3.goto
The above three statements does not need any condition to control the program execution flow.

1.break statement:
In C, the **break statement** is used to perform the following two things.
break statement is used to terminate switch case statement
break statement is also used to terminate looping statements like while, do-while and for.
When a **break** statement is encountered inside the switch case statement, the execution control moves out of the switch statement directly. For example consider the following program.
Example Program | Program to perform all arithmetic operations using switch statement.

```c
#include <stdio.h>
#include<conio.h>
void main(){
  int number1, number2, result ;
  char operator;
  clrscr() ;
  printf("Enter any two integer numbers: ") ;
  scanf("%d%d", &number1, &number2) ;
  printf("Please enter any arithmetic operator: ");
  operator = getchar();
  switch(operator)
  {
     case '+': result = number1 + number2 ;
            printf("Addition = %d", result) ;
            break;
     case '-': result = number1 - number2 ;
            printf("Subtraction = %d", result) ;
            break;
     case '*': result = number1 * number2 ;
            printf("Multiplication = %d", result) ;
            break;
```

```
      case '/': result = number1 / number2 ;
              printf("Division = %d", result) ;
              break;
      case '%': result = number1 % number2 ;
              printf("Remainder = %d", result) ;
              break;
      default: printf("\nWrong selection!!!") ;
   }
   getch() ;
}
```
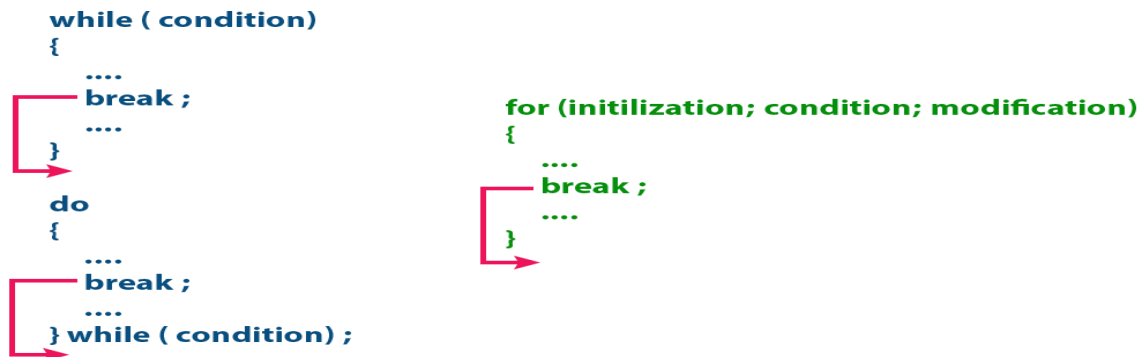
Output

Enter any two integer numbers: 50 30
Please enter any arithmetic operator: *
Multiplication = 1500

When the **break** statement is encountered inside the looping statement, the execution control moves out of the looping statements. The **break** statement execution is as shown in the following figure.

For example, consider the following example program...

Example Program for *break statement*.

```
#include <stdio.h>
#include<conio.h>
void main(){
  char ch ;
  clrscr() ;
  do
  {
    printf("Enter Y / N : ") ;
    scanf("%c", &ch) ;
    if(ch == 'Y')
    {
      printf("Okay!!! Repeat again !!!\n") ;
    }
    else
```

```
        {
            printf("Okay !!! Breaking the loop !!!") ;
            break ;
        }
    } while( 1 ) ;
    getch() ;
}
```
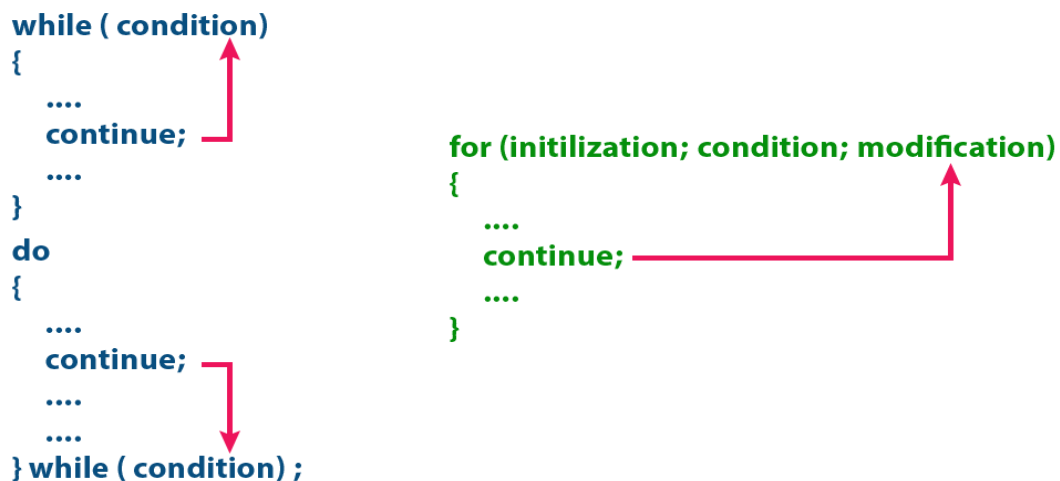
Output:
Enter Y / N : Y
Okay!!! Repeat again !!!
Enter Y / N : Y
Okay!!! Repeat again !!!
Enter Y / N : N
Okay !!! Breaking the loop !!!

2.Continue statement:

The **continue** statement is used to move the program execution control to the beginning of looping statement. When **continue** statement is encountered in a looping statement, the execution control skips the rest of the statements in the looping block and directly jumps to the beginning of the loop. The **continue** statement can be used with looping statements like while, do-while and for.

When we use **continue** statement with **while** and **do-while** statements the execution control directly jumps to the condition. When we use **continue** statement with **for** statement the execution control directly jumps to the modification portion (increment / decrement / any modification) of the for loop. The **continue** statement execution is as shown in the following figure.

```
while ( condition)
{
   ....
   continue;
   ....
}
do
{
   ....
   continue;
   ....
   ....
} while ( condition) ;
```

```
for (initilization; condition; modification)
{
   ....
   continue;
   ....
}
```

Example Program | Program to illustrate continue statement.
```
#include <stdio.h>
```

```
#include<conio.h>
void main(){
   int number ;
   clrscr() ;
   while( 1 )
   {
       printf("Enter any integer number: ") ;
       scanf("%d", &number) ;
       if(number%2 == 0)
       {
           printf("Entered number is EVEN!!! Try another number!!!\n") ;
           continue ;
       }
       else
       {
           printf("You have entered ODD number!!! Bye!!!") ;
           exit(0) ;
       }
   }
   getch() ;
}
```

Output

Enter any integer numbers: 50

Entered number is EVEN!!! Try another number!!!

Enter any integer numbers: 100

Entered number is EVEN!!! Try another number!!!

Enter any integer numbers: 10

Entered number is EVEN!!! Try another number!!!

Enter any integer number: 15

You have entered ODD number!!! Bye!!!

3.goto statement:

The **goto** statement is used to jump from one line to another line in the program.

Using **goto** statement we can jump from top to bottom or bottom to top. To jump from one line to another line, the goto statement requires a **lable**. Lable is a name given to the instruction or line in the program. When we use **goto** statement in the program, the execution control directly jumps to the line with specified lable.

Example Program for *goto statement*.

```
#include <stdio.h>
#include<conio.h>
void main(){
   clrscr() ;
   printf("We are at first printf statement!!!\n") ;
   goto last ;
   printf("We are at second printf statement!!!\n") ;
   printf("We are at third printf statement!!!\n") ;
   last: printf("We are at last printf statement!!!\n") ;
   getch() ;
}
```

Output

We are at first printf statement!!!
We are at last printf statement!!!

MOST IMPORTANT POINTS TO BE REMEMBERED

When we use break, continue and goto statements, we must follow the following...

- The **break** is a keyword so it must be used only in lower case letters.
- The **break** statement can not be used with **if** statement.
- The **break** statement can be used only in switch case and looping statements.
- The **break** statement can be used with **if** statement, only if that **if statement** is written inside the switch case or looping statements.
- The **continue** is a keyword so it must be used only in lower case letters.
- The **continue** statement is used only within **looping statements**.
- The **continue** statement can be used with **if** statement, only if that **if statement** is written inside the looping statements.
- The **goto** is a keyword so it must be used only in lower case letters.
- The **goto** statement must requires a **lable**.
- The **goto** statement can be used with any statement like if, switch, while, do-while and for etc,.