

Module 2

C Programming Constructs

Contents

Introduction to C

- Basic structure of a C program and execution process.
- Pre-processor directives
- Constants and Variables
- Operators,
- Primitive data types
- Type casting
- I/O statements and format specifications.

Control statements

- Decision making and Loop control structure
- Unconditional control transfer statements

Overview C program

- C is a general-purpose, procedural, structured language.
- Computer programming language developed by **Dennis MacAlistair Ritchie** was an American computer scientist. He created the C programming language.
- C language was developed on UNIX and was invented to write UNIX system software.
- It is portable and It can extend itself.
- C is easy to learn.

Basic Structure of C program

Basic Structure of C Programs	
Documentation Section	
Link Section	
Definition Section	
Global Declaration Section	
main() Function Section	
{	
Declaration Part	
Executable Part	
}	
Subprogram Section	
Function 1	
Function 2	
Function 3	
-	
-	
-	
Function n	

Basic Structure of C program

Documentation section :consists of a set of comment line giving the **name of the program, the author name, and other details**. Compiler ignores these comments when it translates the program into executable code.

C uses 2 different formats

1. **Block comments** /*this is multi line comments*/
2. **Line comments** //this is single line comments

The Link section: provides instruction to the compiler to link functions from system library. This Section is also called as pre-processor Statements.

Basic Structure of C program

The definition section: defines all symbolic constants

eg: `#define PI 3.1415.`

Global Declaration section: there are some variables that are used in more than one function, such variables are called global variable and are declared in the global declaration section that is outside of all functions. This section also defines user defined functions.

Basic Structure of C program

Every C program must have one **main () function section**. This section contains two parts declaration part and executable part

Declaration part : declares all the variables

Executable part: There is at least one statement in an executable part.

These two part must appear at the beginning of the brace and ends at the closing brace. All statement in the declaration and executable part ends with semicolon (;).

The sub program section: contains all the user defined functions that are called in the main function although they appear in any order.

Basic Structure of C program

```
/* Display my details */
```

```
#include <stdio.h>    // PREPROCESSOR DIRECTIVES
```

```
void main()           //MAIN FUNCTION/ENTRY POINT
```

```
{
```

```
printf("Name: Amrutha\n"); //BODY OF MAIN FUNCTION
```

```
printf("Dept: CSE\n");
```

```
printf("Collge: MITE\n");
```

```
}
```



```
/* C program to display Area of a circle */ // DOCUMENTATION SECTION

#include<stdio.h> // PREPROCESSOR DIRECTIVES

#define pi 3.142 // DEFINITION SECTION

void hello(); // GLOBAL DECLARATION

void main( ) //MAIN FUNCTION/ENTRY POINT
{

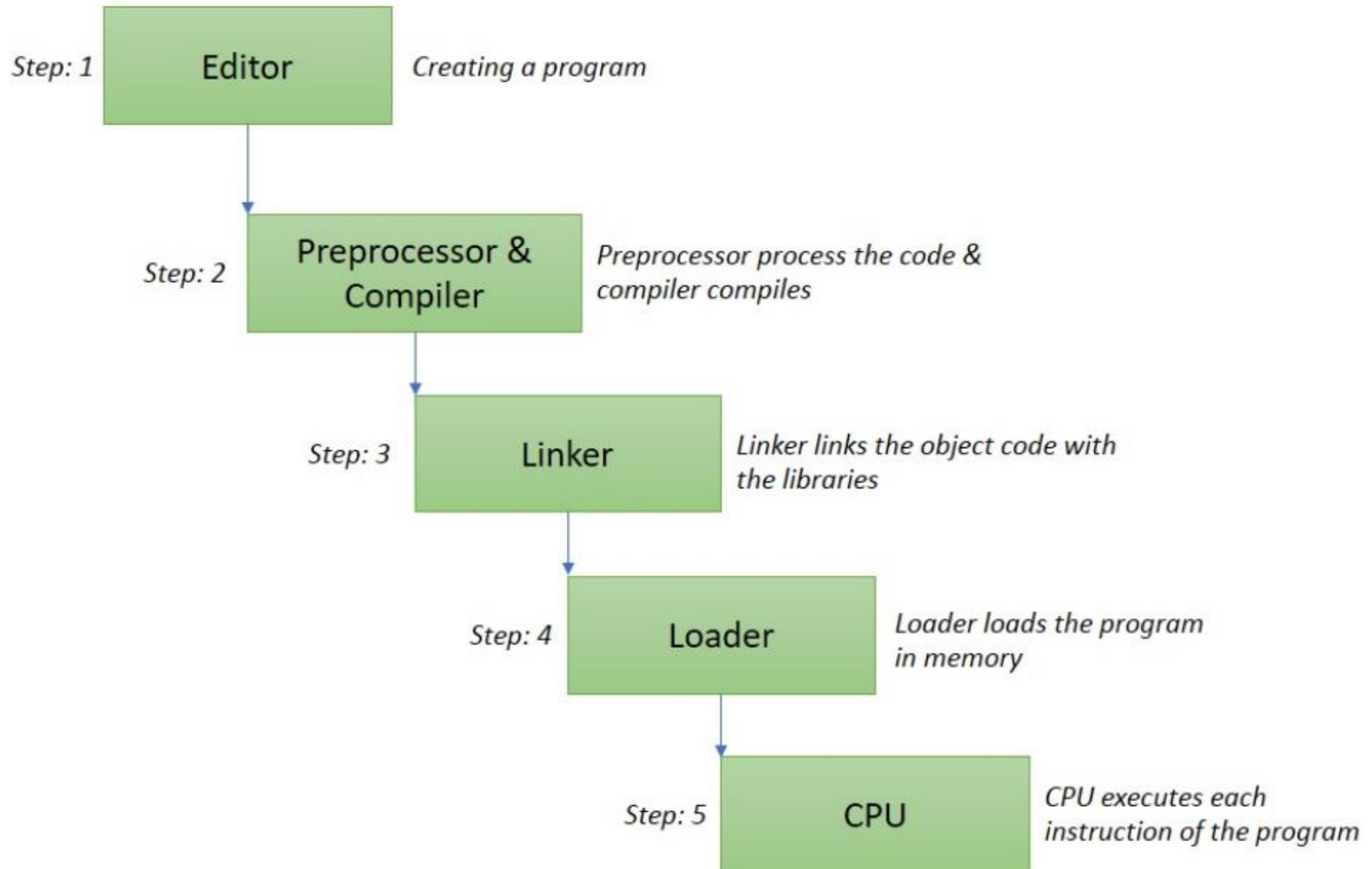
int r =5, Area; //VARIABLE DECLARATION

Area= pi*r*r;

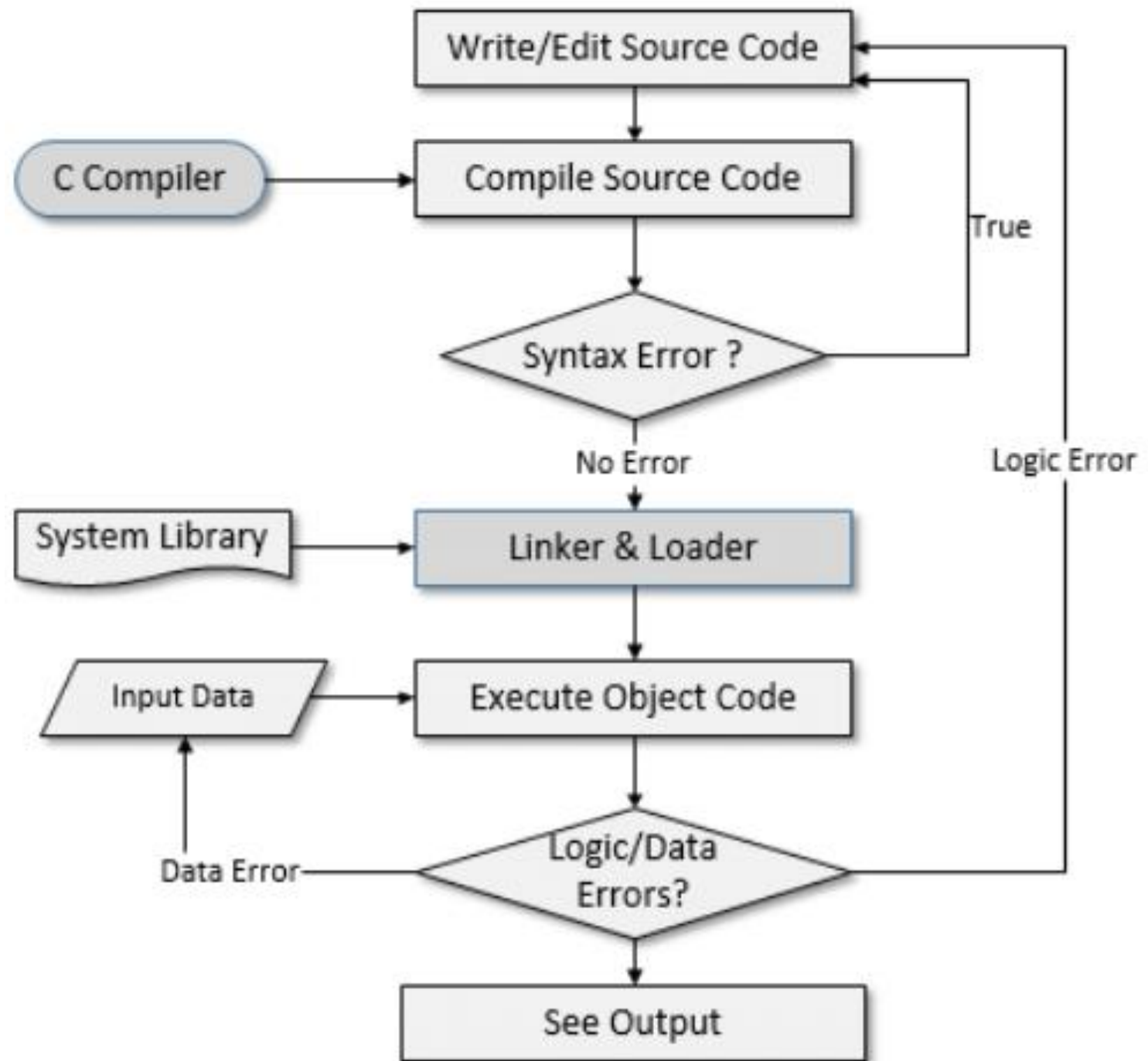
printf("Area of a Circle= %d\n", Area); //BODY OF MAIN FUNCTION
}

void hello() // USER DEFINED FUNCTION
{ printf("Hello");
}
```

Executing a C program

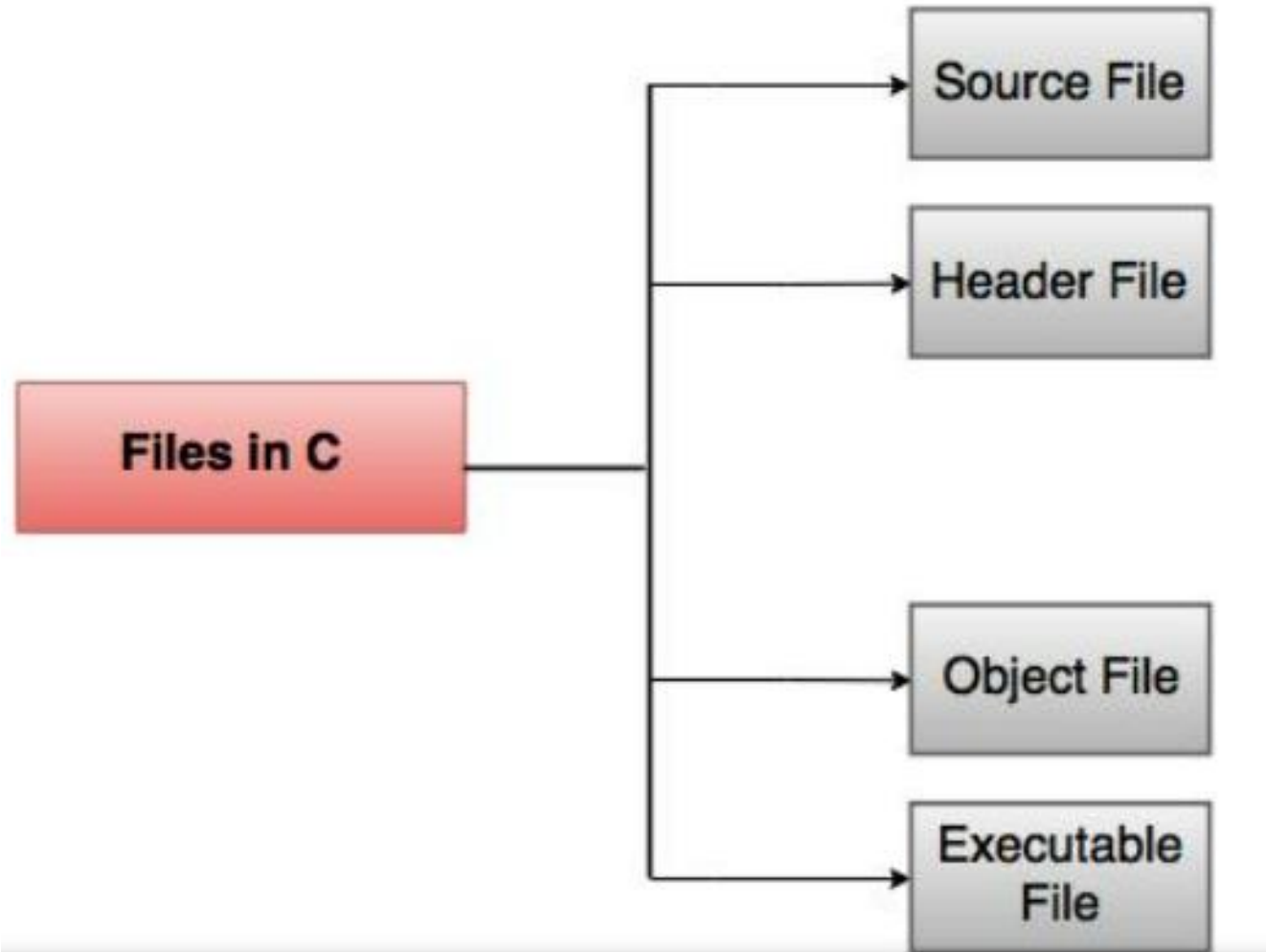


Executing a C program



Files used in C program

A C program uses four types of files as follows:



Files used in C program

A C program uses four types of files as follows:

1. Source Code File

- This file includes the source code of the program.
- The extension for these kind of files are '.c'.
- It defines the main and many more functions written in C.
- `main()` is the starting point of the program. It may also contain other source code files.

Files used in C program

A C program uses four types of files as follows:

2. Header Files

They have an extension '.h'. They contain the C function declarations and macro definitions that are shared between various source files.

C provides us with some standard header files which are available easily.

Common standard header files are:

- i) string.h** – used for handling string functions.
- ii) stdlib.h** – used for some miscellaneous functions.
- iii) stdio.h** – used for giving standardized input and output.
- iv) math.h** – used for mathematical functions.
- v) conio.h** – used for clearing the screen.

Files used in C program

A C program uses four types of files as follows:

3. Object files

- They are the files that are generated by the compiler as the source code file is processed.
- These files generally contain the **binary code** of the function definitions.
- The object file is used by the linker for producing an executable file for combining the object files together. It has a '**.o**' **extension**.

4. Executable file

- This file is generated by the linker.
- Various object files are linked by the linker for producing a binary file which will be executed directly.
- They have an '.exe' extension.

Pre Processor Directives

Preprocessor Directives:-Preprocessor is a program which is invoked by the compiler before the compilation of the user written program.

- The declaration of preprocessor statements always begin with **# symbol**
- These statements are usually **placed before the main() function.**

Types of Preprocessors:-

- Macros
- File Inclusion
 - Conditional compilation
- Other directives

Pre Processor Directives

Macros:-These are the piece of code in the program which is given some name.

- Whenever the name is encountered by the compiler in the program , the compiler replaces the name with the actual piece of code.
- The “**#define**” directive is used to define a macro.

Pre Processor Directives

Macros:-

```
/* C program to demonstrate macros */
#include<stdio.h>
#define pi 3.142
void main()
{
    float r,area;
    printf("Enter the value of radius\n");
    scanf("%f",&r);
    area=pi*r*r;
    printf("Area of circle = %f\n",area);
}
```

Pre Processor Directives

File Inclusion:-This type of preprocessor directive tells the compiler to include a file in the source code program

- These files contains the definition of the predefined functions like `printf()`, `scanf()`, etc. Different functions are declared in different header files.

Pre Processor Directives

File Inclusion:-

/ C program to demonstrate file inclusion */*

```
#include<stdio.h>
#include<math.h>
void main()
{
    int num, root;
    printf("Enter a number\n");
    scanf("%d",&num);
    root=sqrt(num);
    printf("squareroot= %d\n", root);
}
```

```
Enter a number
5
squareroot= 2
```

Pre Processor Directives

Conditional Compilation:-these are the type of directives that helps to compile a specific portion of the program or to skip a specific part of the program based on some conditions.

Some of the conditional compilation directive are: — #ifdef, #ifndef, #if, #else

Pre Processor Directives

Conditional Compilation:-

```
/* C program to demonstrate use of #ifdef */
```

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

```
#define age 18
```

```
void main()
```

```
{
```

```
#ifdef age
```

```
printf("The person is eligible to vote");
```

```
#endif
```

```
}
```

Pre Processor Directives

Other Directives:- Apart from the above directive there are two more directives which are not commonly used

#undef directive:- this directive is used to undefined a value which was declare using the #define directive

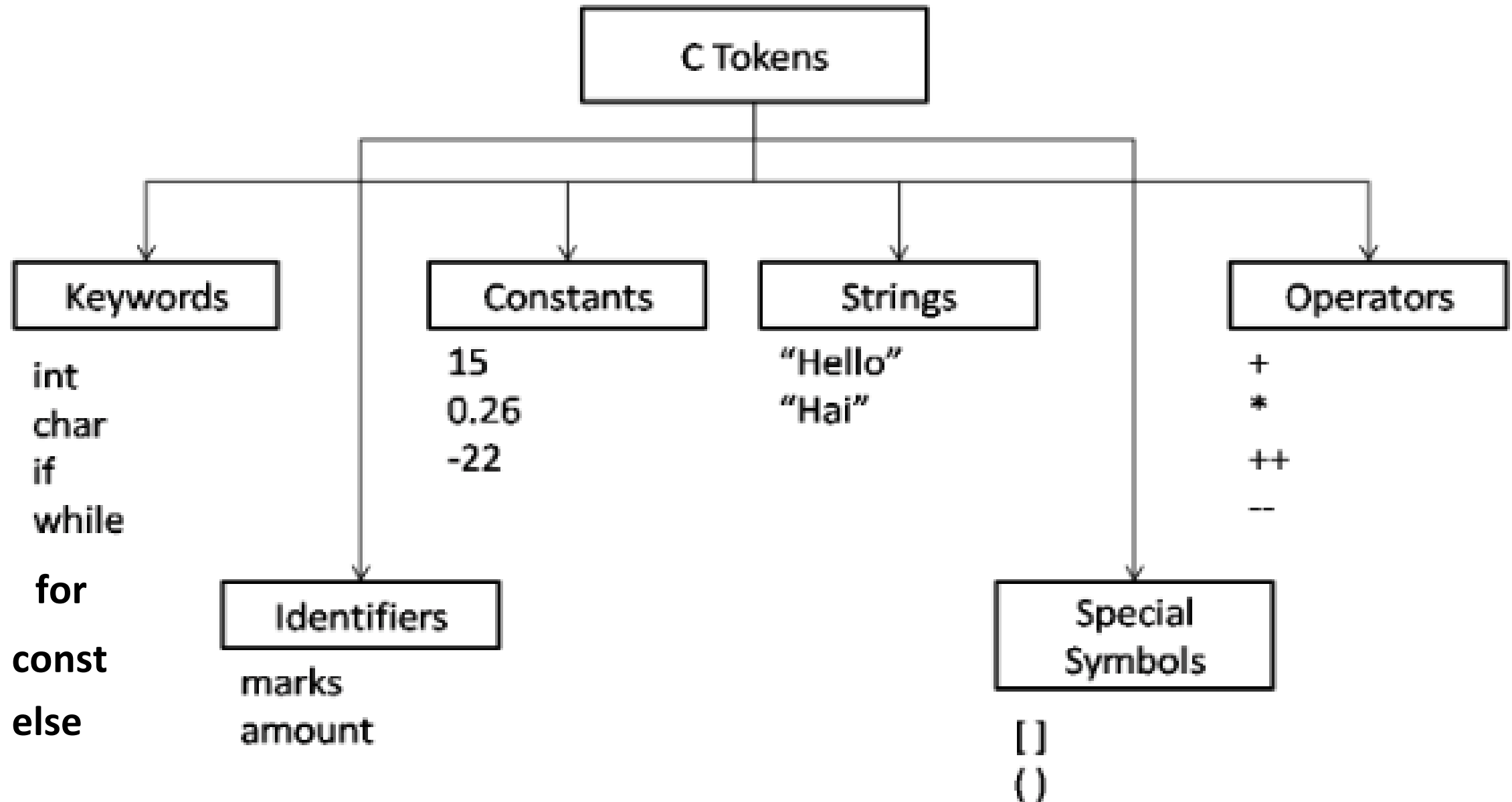
#pragma directive:- this directive is used in parallel programming (thread programming) which is a major concept of operating system.

C Tokens

C has 6 Different types of tokens. C programs are written using these tokens.

- i. **Keywords** : Keywords are reserved words that have special meaning in compiler
- ii. **Identifiers**: the names supplied for variables, types, functions, and labels in the program
- iii. **Constants**: is a name given to the variable whose values can't be altered or changed.
- iv. **Strings**: is a sequence of characters
- v. **Operators**: Its is used to perform some operations
- vi. **Special symbols**: the special symbols have some special meaning and they cannot be used for other purposes.

C Tokens



C Tokens

- **Identifiers:** These are the names given to various elements of the C program like variables, functions, arrays, etc. These are user defined names and consist of sequence letters, digits or underscore.

C Tokens

- **Rules to define an Identifiers**

1. The first character of the identifier must always be a letter or an underscore followed by any number of letters digits or underscore.
2. Keywords cannot be used as identifiers or variables.
3. An Identifier or a variable should not contain two consecutive underscores
4. Whitespaces and special symbols cannot be used to name the identifiers.
5. Identifiers are case sensitive

Constants in C

- Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.
- for example: 10, 20, 'a', 3.4, "c programming" etc.

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

Constants in C

2 ways to define constant in C

1. const keyword
2. #define preprocessor

```
#include<stdio.h>

int main(){

    const float PI=3.14;

    printf("The value of PI is: %f",PI);

    return 0;

}
```

The value of PI is: 3.140000

```
#include <stdio.h>
#define PI 3.14
int main()
{
    printf("%f",PI);
    return 0;
}
```

3.140000

Variables

- A variable is nothing but a name given to **a storage area** that our programs can manipulate.
- Each variable in C has a specific type, which determines the size and layout of the variable's memory.
- A variable can be composed of letters, digits, and the underscore character.

Variable Declaration

- Variables should be declared in the C program before it is used

data_type variable_name;

int age = 20; —————→ **value**
 ↗ ↖
Data type **Variable name**

Void main()

{

int a,b;

}

Types of Variables

1. Local Variable

A variable that is declared and used inside the function or block is called local variable.

It cannot be used outside the block. Local variables need to be initialized before use.

Types of Variables

2. Global Variable

A variable that is declared outside the function or block is called a global variable.

It is declared at the starting of program. It is available to all the functions.

Rules To Declare Variables

- Name your variables based on the terms of the subject area, so that the variable name clearly describes its purpose.
- Every variable name should start with alphabets or underscore (_).
- No spaces are allowed in variable declaration.
- Except underscore (_) no other special symbol are allowed in the middle of the variable declaration (not allowed -> roll-no, allowed -> roll_no).

Rules To Declare Variables

- Maximum length of variable is 8 characters depend on compiler.
- Every variable name always should exist in the left hand side of assignment operator (invalid -> 10=a; valid -> a=10;).
- No keyword should access variable name (int for <- invalid because for is keyword).

Types of Variables

```
#include <stdio.h>

/* global variable declaration */
int g;

int main () {

    /* local variable declaration */
    int a, b;

    /* actual initialization */
    a = 10;
    b = 20;
    g = a + b;

    printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

    return 0;
}
```

value of a = 10, b = 20 and g = 30

Types of Variables

3. Static Variable

Static variables have a property of preserving their value even after they are out of their scope.

A static variable is the one allocated “statically,” which means **its lifetime is throughout the program run.**

It is declared with the 'static' keyword and persists its value across the function calls.

Types of Variables

3. Static Variable By default

- which means the variable remains in memory throughout the life of the program

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

```
void main()
```

```
{
```

```
static int a=0;
```

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

```
}
```

Output:

a= 0

Data Types used in C

- String (or **str** or **text**). Used for a **combination of any characters that appear on a keyboard**, such as letters, numbers and symbols.
- Character (or **char**). Used for **single letters**.
- Integer (or **int**). Used for **whole numbers**.
- Float (or **Real**). Used for **numbers that contain decimal points, or for fractions**.
- Boolean (or **bool**). Used where data is restricted to **True/False or yes/no** options.

Data Type used in Variables

Integer

Character

Float

Double

Bool

String

```
int i, j, k;
```

```
char c, ch;
```

```
float f, salary;
```

```
double d;
```

```
bool a=true, b= false;
```

```
char a[10]="MITE";
```


Data Types used in C

- **Integer (or int)**. Used for **whole numbers**.

```
#include <stdio.h>
void main()
{
    int i = 5;
    printf("The integer value is: %d \n", i);
}
```

Output:
The integer value is: 5

Data Types used in C

- **Float (or Real).** Used for numbers that contain decimal points, or for fractions.

```
#include <stdio.h>
void main()
{
float f = 7.2357;
printf("The float value is: %f \n", f);
}
```

Output:

The float value is: 7.2357

Data Types used in C

- **Character (or char).** Used for **single letters**.

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

```
void main()
```

```
{
```

```
char c;
```

```
c = 'b';
```

```
printf("The character value is: %c \n", c);
```

```
}
```

Output:

The character value is: b

Data Types used in C

- **String** (or **str** or **text**). Used for a **combination of any characters that appear on a keyboard**, such as letters, numbers, alpha numeric and symbols.

```
#include<stdio.h>
int main()
{
char str[] = "Greeks";
printf("The string value is %s",str);
}
```

Output:

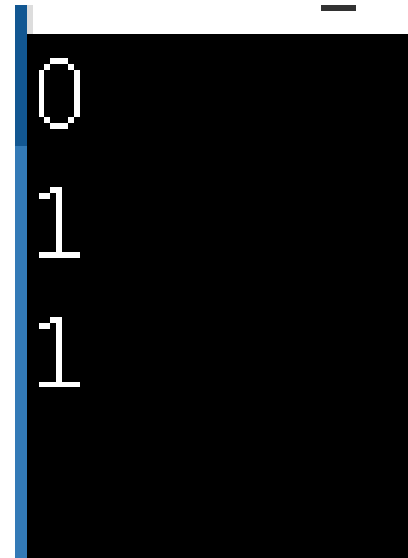
The string value
is :**Greeks**

Data Types used in C

- **Boolean data type:** Used to store true or false values.

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

int main()
{
    bool a=true, b=false;
    printf("%d\n",a&&b);
    printf("%d\n",a||b);
    printf("%d\n",!b);
}
```



A terminal window with a black background and a blue vertical bar on the left. It displays the output of the C program: 0, 1, and 1, each on a new line.

```
0
1
1
```

Operators used in C

- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.
- C language has built-in operators & provides the following types of operators
 1. Arithmetic Operators
 2. Relational Operators
 3. Logical Operators
 4. Bitwise Operators
 5. Assignment Operators
 6. Misc Operators

1. Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc. on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

1. Arithmetic Operators

```
#include <stdio.h> // Working of arithmetic operators
int main()
{
    int a = 9,b = 4, c,d,e,f,g;
        c = a+b; //Addition
        printf("a+b = %d \n",c);
    d= a-b; // Subtraction
        printf("a-b = %d \n",d);
    e= a*b; //Mutiplication
        printf("a*b = %d \n",e);
    f = a/b; //Division
        printf("a/b = %d \n",f);
    g = a%b; //Modulus
        printf("Remainder when a divided by b = %d \n",g);
}
```

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```


2. Increment and Decrement Operators

C programming has two operators

- | | | |
|-----------------|---|---|
| 1) Increment ++ | } | To change the value of an operand
of an operand (constant or variable) by 1. |
| 2) Decrement -- | | |

Increment ++ increases the value by 1

Decrement -- decreases the value by 1.

These two operators are unary operators, meaning they only operate on a single operand.

Increment Operators

- Increment Operators are the unary operators used to increment or add 1 to the operand value.
- The Increment operand is denoted by the double plus symbol (++). It has two types:

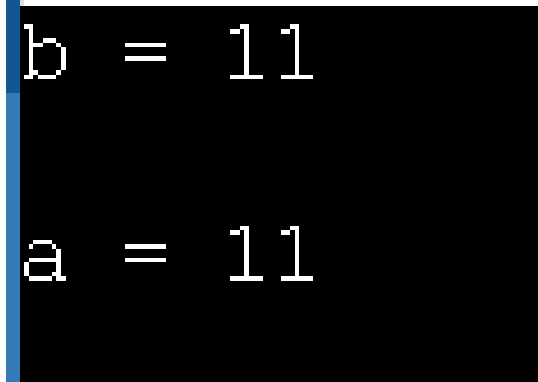
Pre Increment and Post Increment Operators.

Increment Operators -Pre-increment Operator

The pre-increment operator is used to increase the original value of the operand by 1 before assigning it to the expression.

```
#include<stdio.h>

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



```
b = 11
a = 11
```

Here first the value of a increments and then is assigned to variable b. **So both a and b value will be 11.**

Increment Operators –Post-increment Operator

The post-increment operator is used to increment the original value of the operand by 1 after assigning it to the expression.

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

```
b = 10
a = 11
```

Here first value of a(i.e., 10) is assigned to b and then value of a is incremented. **So b = 10 and a = 11 is printed.**

Decrement Operator

Pre-increment

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

```
void main()
```

```
{
```

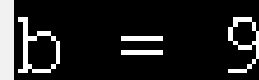
```
    int a = 10, b;
```

```
    b = --a;
```

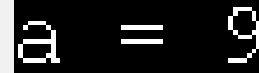
```
    printf("b = %d\n\n", b);
```

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

```
}
```



```
b = 9
```



```
a = 9
```

Here first the value of a decrements and then is assigned to variable b.

So both a and b value will be 9.

Decrement Operator

Post Decrement

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

```
void main()
```

```
{
```

```
    int a = 10, b;
```

```
    b = a--;
```

```
    printf("b = %d\n\n", b);
```

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

```
}
```

```
b = 10
a = 9
```

Here first value of a(i.e., 10) is assigned to b and then value of a is decremented **So b = 10 and a = 9 is printed.**

3. Relational Operators

Relational operators in C are commonly used to check the relationship between the two variables

```
#include <stdio.h>

int main()
{
    int a = 9;
    int b = 4;

    printf(" a > b: %d \n", a > b);
    printf("a >= b: %d \n", a >= b);
    printf("a <= b: %d \n", a <= b);
    printf("a < b: %d \n", a < b);
    printf("a == b: %d \n", a == b);
    printf("a != b: %d \n", a != b);

    return 0;
}
```

```
a > b: 1
a >= b: 1
a <= b: 0
a < b: 0
a == b: 0
a != b: 1
```

4. Logical Operators

These operators are used to perform logical operations on the given expressions.

Operators	Example/Description
&& (logical AND)	<code>(x>5)&&(y<5)</code> It returns true when both conditions are true
(logical OR)	<code>(x>=10) (y>=10)</code> It returns true when at-least one of the condition is true
! (logical NOT)	<code>!((x>5)&&(y<5))</code> It reverses the state of the operand “((x>5) && (y<5))” If “((x>5) && (y<5))” is true, logical NOT operator makes it false

4. Logical Operators

```
#include <stdio.h>
void main()
{
    int a = 10, b = 4, c = 10, d = 20;
    if (a > b && c == d)
        printf("a is greater than b AND c is equal to d\n");
    else
        printf("AND condition not satisfied\n");
    if (a > b || c == d)
        printf("a is greater than b OR c is equal to d\n");
    else
        printf("Neither a is greater than b nor c is equal d\n ");
}
```

AND condition not satisfied

a is greater than b OR c is equal to d

5. Bitwise Operators

- Bitwise operator works on bits and perform bit-by-bit operation.
- The truth tables for $\&$, $|$, and \wedge .
-

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

5. Bitwise Operators

Bitwise AND

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Output = 8

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

```
    00001100
& 00011001
  _____
    00001000 = 8 (In decimal)
```

5. Bitwise Operators

Bitwise OR

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

Output = 29

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

```
  00001100
| 00011001
  _____
  00011101 = 29 (In decimal)
```

5. Bitwise Operators

Bitwise Left Shift (<<): Shift specified number of bits to left side

X	0	1	0	0	0	1	1	0
X<<2	0	0	0	1	1	0	0	0

Empty boxes will be marked as zero

Bitwise Right Shift (>>): Shift specified number of bits to right side.

X	0	1	0	0	0	1	1	0
X>>2	0	0	0	1	0	0	0	1

6. Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	$C = A + B$ will assign the value of $A + B$ to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand	$C /= A$ is equivalent to $C = C / A$

6. Assignment Operators

<code>%=</code>	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	<code>C %= A</code> is equivalent to <code>C = C % A</code>
<code><<=</code>	Left shift AND assignment operator.	<code>C <<= 2</code> is same as <code>C = C << 2</code>
<code>>>=</code>	Right shift AND assignment operator.	<code>C >>= 2</code> is same as <code>C = C >> 2</code>
<code>&=</code>	Bitwise AND assignment operator.	<code>C &= 2</code> is same as <code>C = C & 2</code>
<code>^=</code>	Bitwise exclusive OR and assignment operator.	<code>C ^= 2</code> is same as <code>C = C ^ 2</code>
<code> =</code>	Bitwise inclusive OR and assignment operator.	<code>C = 2</code> is same as <code>C = C 2</code>

7. Misc Operators

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Expressions in C

An expression is a formula in which operands are linked to each other by the use of operators to compute a value.

example: $A * B$

There are 4 types of expressions:

1. Arithmetic expressions
2. Relational expressions
3. Logical expressions
4. Conditional expressions

Expressions in C

1. Arithmetic Expressions

Addition (+), Subtraction(-), Multiplication(*), Division(/), Modulus(%), Increment(++ and Decrement(–) operators are said to “Arithmetic expressions”.

This operator works in between operands. like $A+B$, $A-B$, $A-$, $A++$ etc.

2. Relational Expressions

$==$ (equal to), $!=$ (not equal to), $!=$ (not equal to), $>$ (greater than), $<$ (less than), $>=$ (greater than or equal to), $<=$ (less than or equal to) operators are said to “Relational expressions”.

This operators works in between operands. Used for comparing purpose. Like $A==B$, $A!=B$, $A>B$, $A<B$ etc.

Expressions in C

3. Logical Expressions

&&(Logical and), ||(Logical or) and !(Logical not) operators are said to “Logical expressions”. Used to perform a logical operation.

This operator works in between operands. Like A&&B, A||B,A!B etc.

4. Conditional Expressions

?(Question mark) and :(colon) are said to “Conditional expressions”. Used to perform a conditional check. It has 3 expressions first expression is condition.

If it is true then execute expression2 and if it is false then execute expression3. Like (A>B)?”A is Big”:”B is Big”.

Evaluation of Expressions in C

Operator	Description	Associativity	Precedence(Rank)
() []	Function call Array element reference	Left to right	1
+ - ++ -- ! ~ * & Sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer to reference Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ -	Addition Subtraction	Left to right	4
<< >>	Left shift Right Shift	Left to right	5

Evaluation of Expressions in C

<	Less than	Left to right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equality	Left to right	7
!=	Inequality		
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
+= *= -=	Assignment	Right to left	14

Evaluation of Expressions in C

10 - 3 % 8 + 6 / 4

17 - 8 / 4 * 2 + 3 - ++a

a=5

Evaluation of Expressions in C

10 - 3 % 8 + 6 / 4



10 - 3 + 6 / 4



10 - 3 + 1



7 + 1



8

Evaluation of Expressions in C

`17 - 8 / 4 * 2 + 3 - ++a`

`a=5`

`17 - 8 / 4 * 2 + 3 - 6`

`++a` Is Incrementor so
Executed First

`17 - 2 * 2 + 3 - 6`

`17 - 4 + 3 - 6`

`13 + 3 - 6`

`16 - 6`

`10`

`10 + 4 * 3 / 2`

`1+2*5+3`

`4-2+6*3`

`a+b*a/b-a%b -> a=10,b=2`

Evaluation of Expressions in C

1. If a=8, b=15 and c=4 calculate the expression

$$2 * ((a \% 5) * (4 + (b - 3) / (c + 2)))$$

2. Evaluate the expression

$$a += b *= c -= 5, \text{ Given } a=3, b=5, c=8.$$

3. Evaluate the expression

$$100 / 20 <= 10 - 5 + 100 \% 10 - 20 == 5 >= 1 != 20$$

Evaluation of Expressions in C

1. If a=8, b=15 and c=4 calculate the expression

$$2 * ((a \% 5) * (4 + (b - 3) / (c + 2)))$$

$$= 2 * ((\underline{8 \% 5}) * (4 + (15 - 3) / (4 + 2))) \text{ //Substitution of values}$$

$$= 2 * (3 * (4 + (\underline{15 - 3}) / (4 + 2))) \text{ //Brackets having the highest priority}$$

$$= 2 * (3 * (4 + 12 / (\underline{4 + 2}))) \text{ //inner most brackets are evaluated first}$$

$$= 2 * (3 * (4 + \underline{12 / 6})) \text{ //Brackets having the highest priority}$$

$$= 2 * (3 * (\underline{4 + 2})) \text{ //within the brackets '/' has the highest priority}$$

$$= 2 * (\underline{3 * 6}) \text{ // inner most brackets are evaluated}$$

$$= \underline{2 * 18} \text{ // Brackets having the highest priority}$$

$$= 36 \text{ //Final Result}$$

Evaluation of Expressions in C

$a += b * c -= 5$, Given $a=3$, $b=5$, $c=8$.

$a += b * \underline{c -= 5}$ //Apply Associativity i.e. evaluate from right to left

$a += b * (c = \underline{c - 5})$ //Deduce the short hand Expression

$a += b * (c = \underline{8 - 5})$ //Substitute the given value of c

$a += b * (\underline{c = 3})$ //Reduce the equation to simplified form

$a += \underline{b * 3}$ //Apply Associativity i.e. evaluate from right to left

$a += (b = \underline{b * 3})$ //Deduce the short hand Expression

$a += (b = \underline{5 * 3})$ //Substitute the given value of b

$a += (\underline{b = 15})$ //Reduce the equation to simplified form

$\underline{a + = 15}$ //Apply Associativity i.e. evaluate from right to left

$a = \underline{a + 15}$ // Deduce the short hand Expression

$a = \underline{3 + 15}$ //Substitute the given value of a

$a = 18$ // **Final Result**

Evaluation of Expressions in C

$100 / 20 \leq 10 - 5 + 100 \% 10 - 20 == 5 \geq 1 != 20$

→ **100 / 20** $\leq 10 - 5 + 100 \% 10 - 20 == 5 \geq 1 != 20$

→ $5 \leq 10 - 5 +$ **100 % 10** $- 20 == 5 \geq 1 != 20$

→ $5 \leq$ **10 - 5** $+ 0 - 20 == 5 \geq 1 != 20$

→ $5 \leq$ **5 + 0** $- 20 == 5 \geq 1 != 20$

→ $5 \leq$ **5 - 20** $== 5 \geq 1 != 20$

→ **5 <= -15** $== 5 \geq 1 != 20$ // Simplify the relational operators

→ $0 ==$ **5 >= 1** $!= 20$ // True is given by 1 and false is given by 0

→ **0 == 1** $!= 20$

→ **0 != 20**

→ 1

Writing C expressions for Mathematical Expressions

Basic Conversions

$$\frac{x}{y} \rightarrow x/y$$

$$\sqrt{v} \rightarrow \text{sqrt}(v)$$

$$|h| \rightarrow \text{abs}(h)$$

$$g^t \rightarrow \text{pow}(g,t)$$

$$e^x \rightarrow \text{exp}(x)$$

$$\sin x \rightarrow \sin(x)$$

$$\sin 45^\circ \rightarrow \sin((45 * 3.142) / 180) /*\text{converting degrees to radians}*/$$

Write the C equivalent expressions for the following mathematical Expressions

$$1. A = \frac{5x+3y}{a+b} \rightarrow A = ((5 * x) + (3 * y)) / (a + b)$$

$$2. C = e^{|x+y-10|} \rightarrow C = \exp(\text{abs}(x + y - 10))$$

$$3. P = \frac{e^{\sqrt{x}} + e^{\sqrt{y}}}{x \sin \sqrt{y}} \rightarrow P = (\exp(\text{sqrt}(x)) + \exp(\text{sqrt}(y))) / (x * \sin(\text{sqrt}(y)))$$

$$4. X = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \rightarrow X = ((-b) + \text{sqrt}(b * b - 4 * a * c)) / (2 * a)$$

Input/output statements in C

- Reading, processing, and writing of data are the three essential functional functions of a computer program input and output operations.
- There are two methods of providing data to the program variables.
 - ✓ **One method** is to **assign values to variables(variable declaration)**
 - ✓ **Another method** for outputting results extensively(**printf**)

Input/output statements in C

- All input/output operations are carried out through function calls such as **printf** and **scanf**.
- These functions are predefined in the respective header files.
- The input and output functions are used in the program whose functionality are predefined in the header file “**#include<stdio.h>**”

MANAGING INPUT AND OUTPUT

Input and output functions are broadly classified into

A)Formatted Input

`scanf ()`

Formatted Output

`printf()`

B) Unformatted Input

1) `getch()`

2) `getche()`

3) `getchar()`

Unformatted Output

1) `putch()`

2) `putchar()`

MANAGING INPUT AND OUTPUT

A) Formatted Output:

1. **printf()**: It is a predefined function from the header <stdio.h>

It is used to write the formatted output.

printf : syntax

printf(*“format specifier”, var1, var 2*);

- The number of format specifier must match the number of variables in the variable list.

MANAGING INPUT AND OUTPUT

A) Formatted Output:

1. printf():

```
printf(“%d%c”,var1,var 2);
```

- **format specifier** indicates the type of data to be displayed
- **variable list** indicates the value present in the variable

MANAGING INPUT AND OUTPUT

A) Formatted Input:

1. scanf():

- scanf() function reads all type of data value from input device or from a file.
- The address operator “&” is used to indicate the memory location of the variable.
- This memory location is used to store the data which is read through the keyboard

MANAGING INPUT AND OUTPUT

A) Formatted Input:

1. scanf():

- **syntax:**

scanf("format specifier", addresslist);

- **format specifier** indicates the type of data to be stored in the variable.

- **address list** indicates the location of the variable where the value of the data is to be stored

scanf("%d %d",&value1,&value 2);

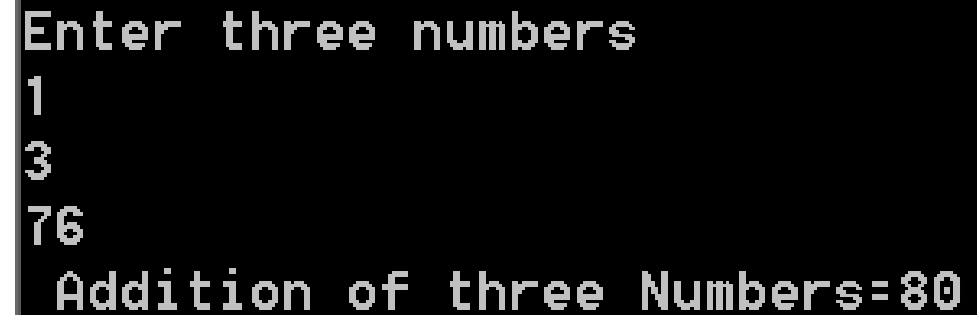
Format specifications

Data type		Format specifier
Integer	short signed	%d or %I
	short unsigned	%u
	long signed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%X
	unsigned octal	%O
Real	float	%f
	double	%lf
Character	signed character	%c
	unsigned character	%c
String		%s

C program to demonstrate Formatted Input and Output Statements

```
#include<stdio.h>

void main()
{
int a, b, c, sum;
printf("Enter three numbers\n");
scanf("%d%d%d",&a,&b&c);
sum=a+b+c;
printf(" Addition of three Numbers=%d\n",sum);
}
```



```
Enter three numbers
1
3
76
Addition of three Numbers=80
```

C program to find the area of circle by reading the input from keyboard

```
#include<stdio.h>
#define pi 3.14
int main()
{
    int r;
    float area;
    printf("enter the radius of a circle\n");
    scanf("%d",&r);
    area=3.14*r*r;
    printf("area of a circle is:%f",area);
    return 0;
}
```

```
enter the radius of a circle
5
area of a circle is:78.500000|
```


C program to find the area of circle by reading the input from keyboard

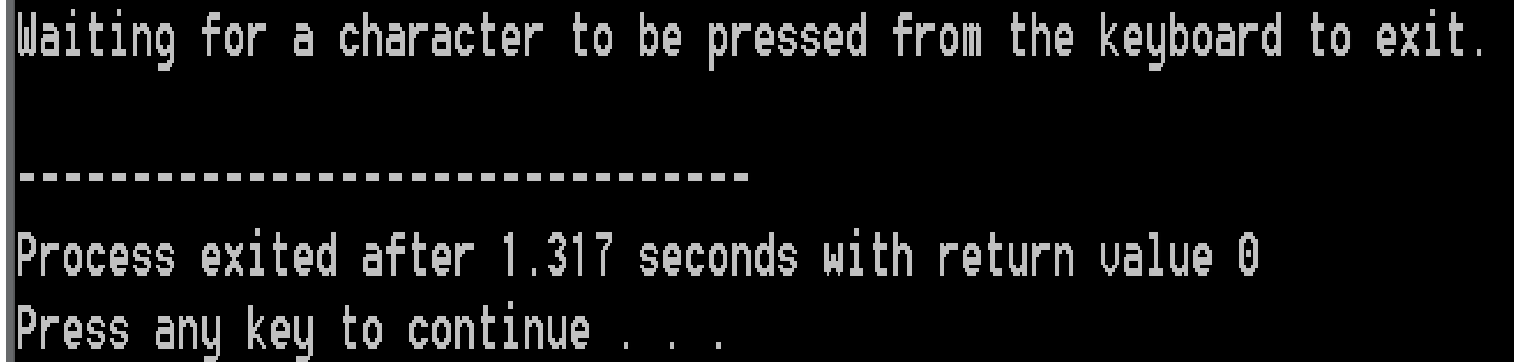
```
#include<stdio.h>
int main()
{
    int r;
    float pi,area;
    printf("enter the radius of a circle\n");
    scanf("%d",&r);
    printf("enter the value of pi\n");
    scanf("%f",&pi);
    area=pi*r*r;
    printf("area of a circle is:%f",area);
    return 0;
}
```

```
enter the radius of a circle
5
enter the value of pi
3.14
area of a circle is:78.500000|
```

MANAGING INPUT AND OUTPUT

B) UnFormatted Input: **b) getch():** pauses the Output Console until a key is pressed

```
#include<stdio.h>
#include<conio.h>
void main()
{
printf("Waiting for a character to be pressed from the keyboard to
exit.\n");
getch();
}
```

A screenshot of a terminal window with a black background and white text. The text shows the program's execution: it first displays "Waiting for a character to be pressed from the keyboard to exit." followed by a dashed line separator. Then, it shows "Process exited after 1.317 seconds with return value 0" and "Press any key to continue . . .".

```
Waiting for a character to be pressed from the keyboard to exit.
-----
Process exited after 1.317 seconds with return value 0
Press any key to continue . . .
```

MANAGING INPUT AND OUTPUT

B) UnFormatted Input: a) `getche()`: to read a single character from the keyboard which displays immediately on screen without waiting for the enter key

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

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

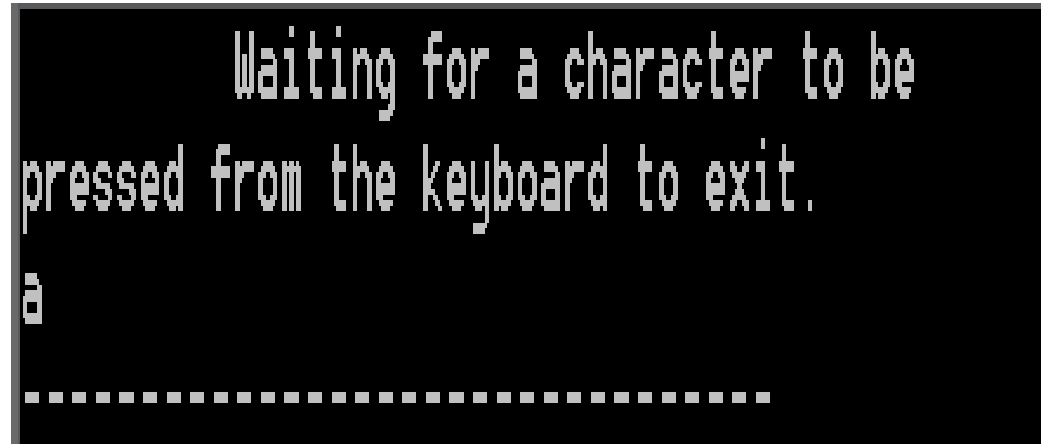
```
void main()
```

```
{
```

```
    printf("Waiting for a character to be pressed from the keyboard to exit.\n");
```

```
    getche();
```

```
}
```

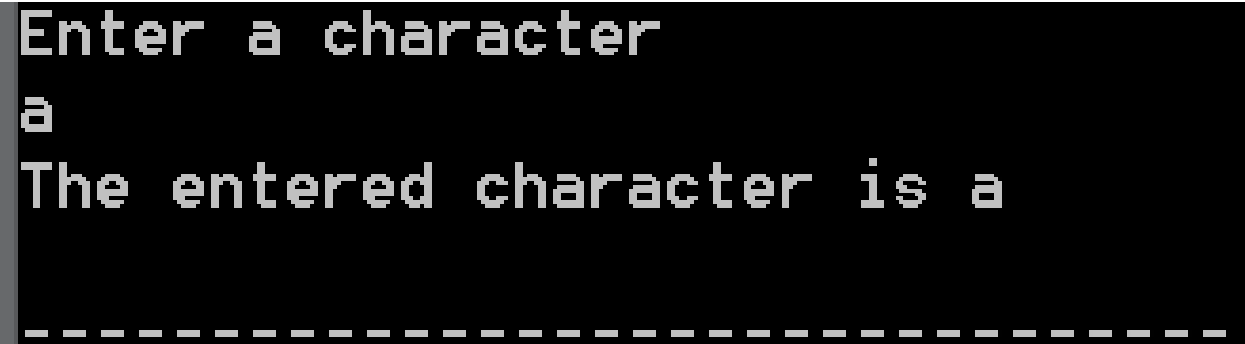


```
Waiting for a character to be  
pressed from the keyboard to exit.  
a  
-----
```

MANAGING INPUT AND OUTPUT

B) UnFormatted Input: c) `getchar()`: reads a single character from the standard input stream `stdin`, regardless of what it is, and returns it to the program.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter a character\n");
    ch=getchar();
    printf("The entered character is %c\n",ch);
}
```



```
Enter a character
a
The entered character is a
-----
```

MANAGING INPUT AND OUTPUT

B) UnFormatted Output: putchar(), puts()

putchar() function is used to write a character on standard output/screen.

putchar()

In a C program, we can use putchar function as below.
putchar(char); where, char is a character variable/value.

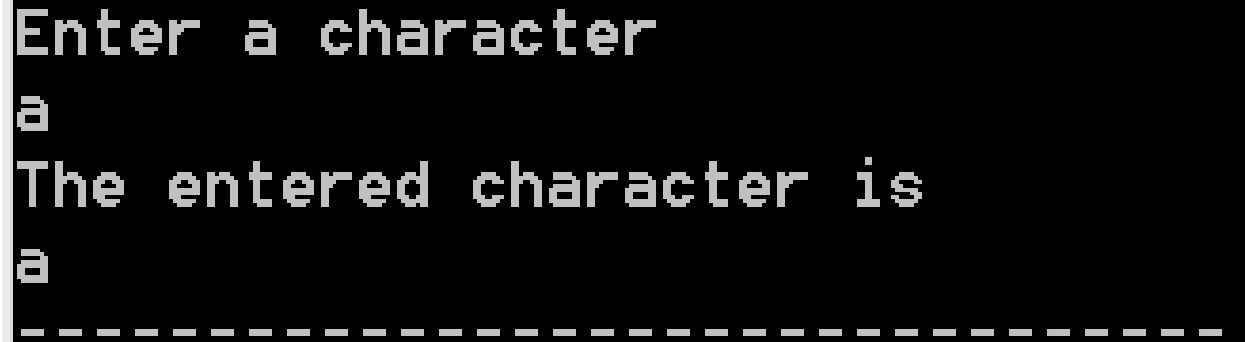
puts()

Used to print the string variable/value.

MANAGING INPUT AND OUTPUT

B) UnFormatted Output: getchar() and putchar()

```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter a character\n");
    ch=getchar();
    printf("The entered character is \n");
    putchar(ch);
    return 0;
}
```

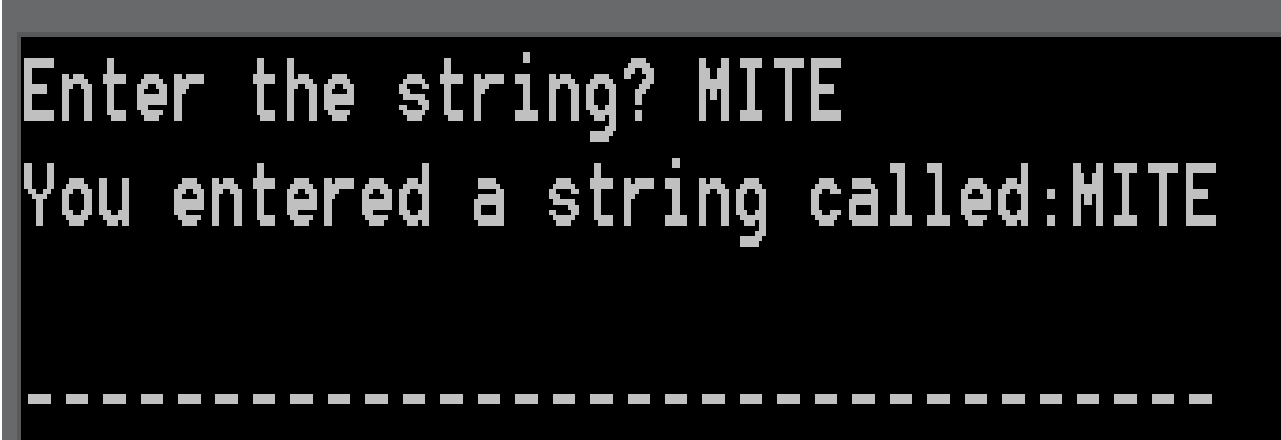
A terminal window with a black background and white text. The first line is the prompt "Enter a character". The second line is the user input "a". The third line is the output "The entered character is". The fourth line is the user input "a". The fifth line is a dashed line representing the continuation of the output.

```
Enter a character
a
The entered character is
a
-----
```

MANAGING INPUT AND OUTPUT

B) UnFormatted Output: gets() and puts()

```
#include<stdio.h>
int main()
{
    char s[30];
    printf("Enter the string? ");
    gets(s);
    printf("You entered a string called:");
    puts(s);
    return 0;
}
```

A screenshot of a terminal window with a black background and white text. The first line shows the prompt 'Enter the string?' followed by the input 'MITE'. The second line shows the output 'You entered a string called:MITE'. Below the output is a dashed line representing a continuation of the output.

Enter the string? MITE
You entered a string called:MITE

Type Conversion:

Implicit Type Conversion: This type of conversion is done by the compiler, so it is called as implicit type conversion. Without user intervention this process is carried out.

- Whenever we are converting narrow operand (lower data type variable) into wide operand (higher data type variable) then compiler will do it implicitly.

Type Conversion:

Implicit Type Conversion

```
#include<stdio.h>
void main( )
{
char b= 'A';
int a;
a=b;
printf(“%d”,a);
}
```

OUTPUT: 65

Type Conversion:

Explicit Type Conversion/Type casting: This type of conversion is done by the user, Instead of being done automatically according to the rules of the language for implicit type conversion so it is called explicit type conversion.

Type casting is a mechanism in which one data type is converted to another data type using a **casting () operator by a programmer.**

Type conversion allows a **compiler to convert one data type to another data type** at the compile time of a program or code.

Type Conversion:

It is a process of converting an expression from one data type to another data type

There are two types:

Implicit Type conversion

Explicit Type Conversion

Type Conversion:

Implicit Type Conversion

```
#include<stdio.h>
void main( )
{
char b= 'A';
int a;
a=b;
printf(“%d”,a);
}
```

OUTPUT: 65

Type Conversion:

Explicit Type Conversion:

```
#include<stdio.h>
void main(){
    int x=7, y=5;
    float z;
    z = x/y;
    printf("z:%f",z);
}
```

Explicit type conversion

```
#include<stdio.h>
void main(){
    int x=7, y=5;
    float z;
    z = (float)x/(float)y;
    printf("z:%f",z);
}
```

z:1.000000

z:1.400000

Conditional Branching and Loops.

Conditional statements

- Conditional statements are used to execute A set of statements on some conditions.
- It provides A unit of block in which we can either execute one statement or more than one statements.
- If the given condition is true then the set of statements are executed otherwise body is skipped and next statement will be executed..

Conditional statements

- There are different types of Conditional statements
 - IF Condition
 - IF ELSE Condition
 - Nested IF ELSE condition
 - Cascaded if-else or else if ladder
 - Switch Case

Conditional statements

1. IF CONDITION

- It is conditional statement, which is used to execute a set of statement on some conditions.
- The condition must be of Boolean type expression.
- An expression, which returns only two value either TRUE or FALSE, is known as Boolean type expression.

Conditional statements

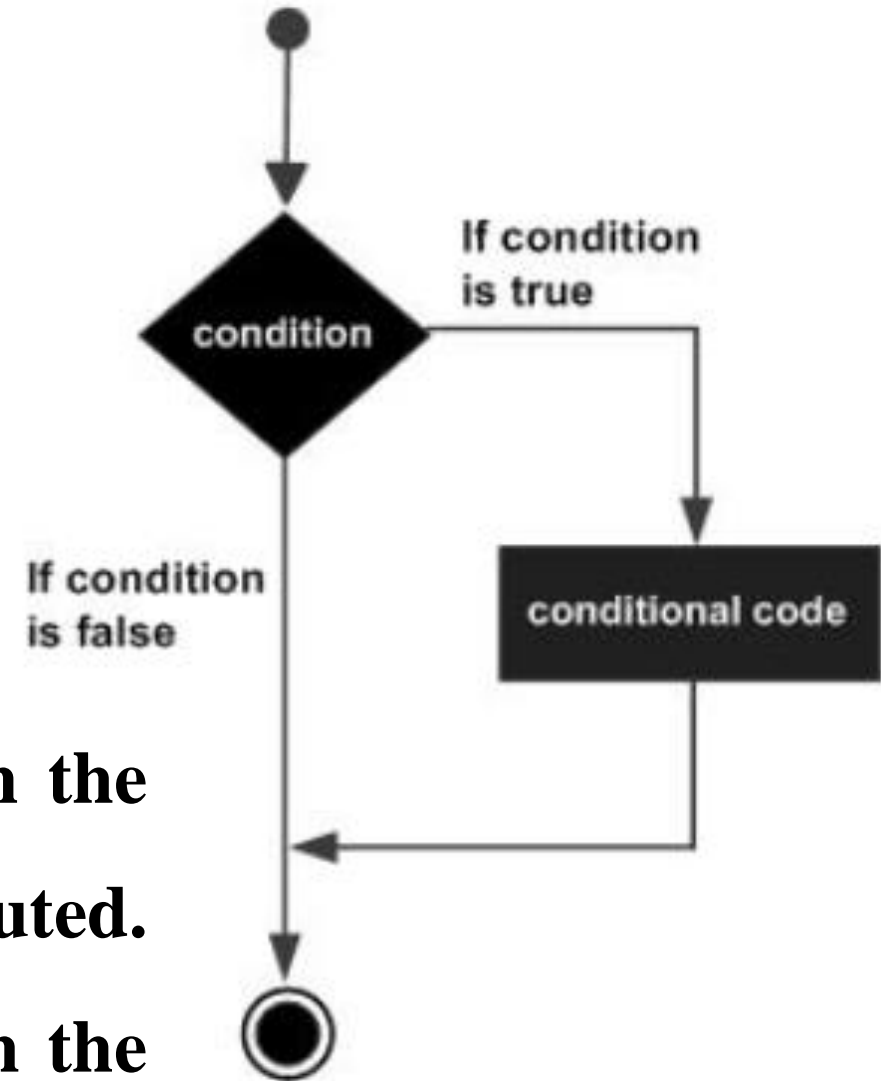
1. IF CONDITION

Syntax: if (condition)

```
{ .....  
..... }
```

If the Boolean expression evaluates to true, then the block of code inside the 'if' statement will be executed.

If the Boolean expression evaluates to false, then the first set of code after the end of the 'if' statement will be executed



Conditional statements

1. IF CONDITION

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

```
int main()
```

```
{
```

```
int number;
```

```
printf("enter a number:");
```

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

```
if(number==10)
```

```
printf("number is equals to 10");
```

```
return 0;
```

```
}
```



```
enter a number:6
```



```
-----
```



```
enter a number:10
```



```
number is equals to 10
```



```
-----
```

Conditional statements

2. IF ELSE CONDITION

- It is known as double blocked conditional statements.
- It means, it has TRUE parts as well as FALSE part.
- If the given condition is true then the true part is executed otherwise false part is executed.

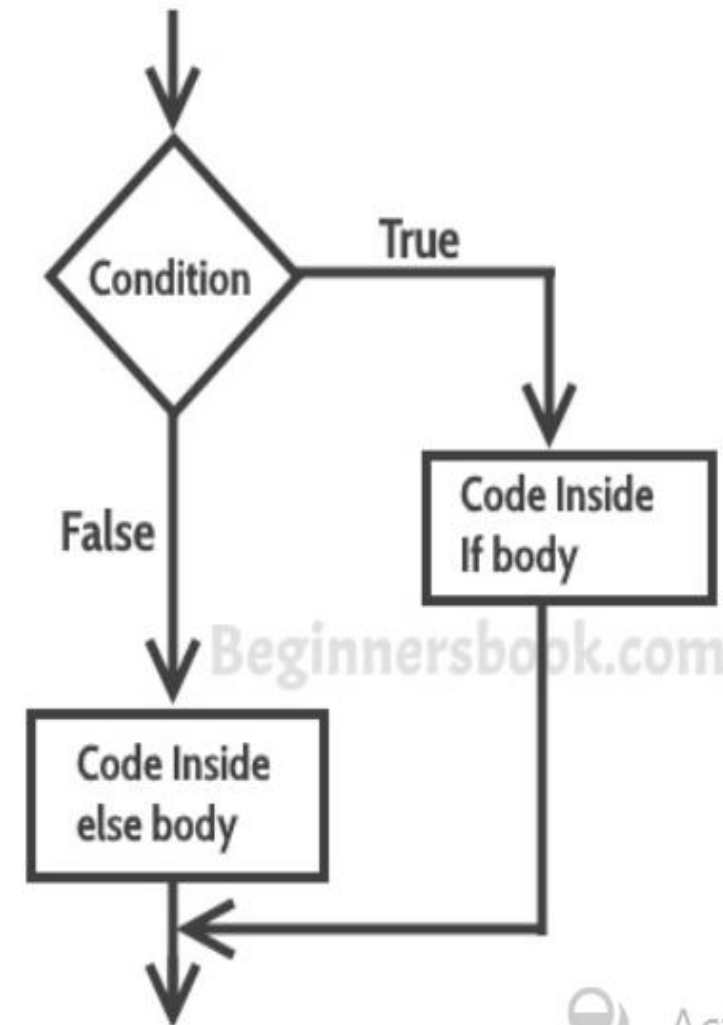
Conditional statements

2. IF ELSE CONDITION

Syntax: - **if (CONDITION)**

```
{ .....  
}  
else  
{ .....  
}
```

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped. If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.



Conditional statements

2. IF ELSE CONDITION

```
#include <stdio.h>
int main()
{
    int age;
    printf("Enter your age?");
    scanf("%d",&age);
    if(age>=18)
    {
        printf("You are eligible to vote...");
    }
    else
    {
        printf("Sorry ... you can't vote");
    }
}
```

```
Enter your age?19
You are eligible to vote...
```

```
Enter your age?15
Sorry ... you can't vote
```

Conditional statements

3. NESTED IF ELSE

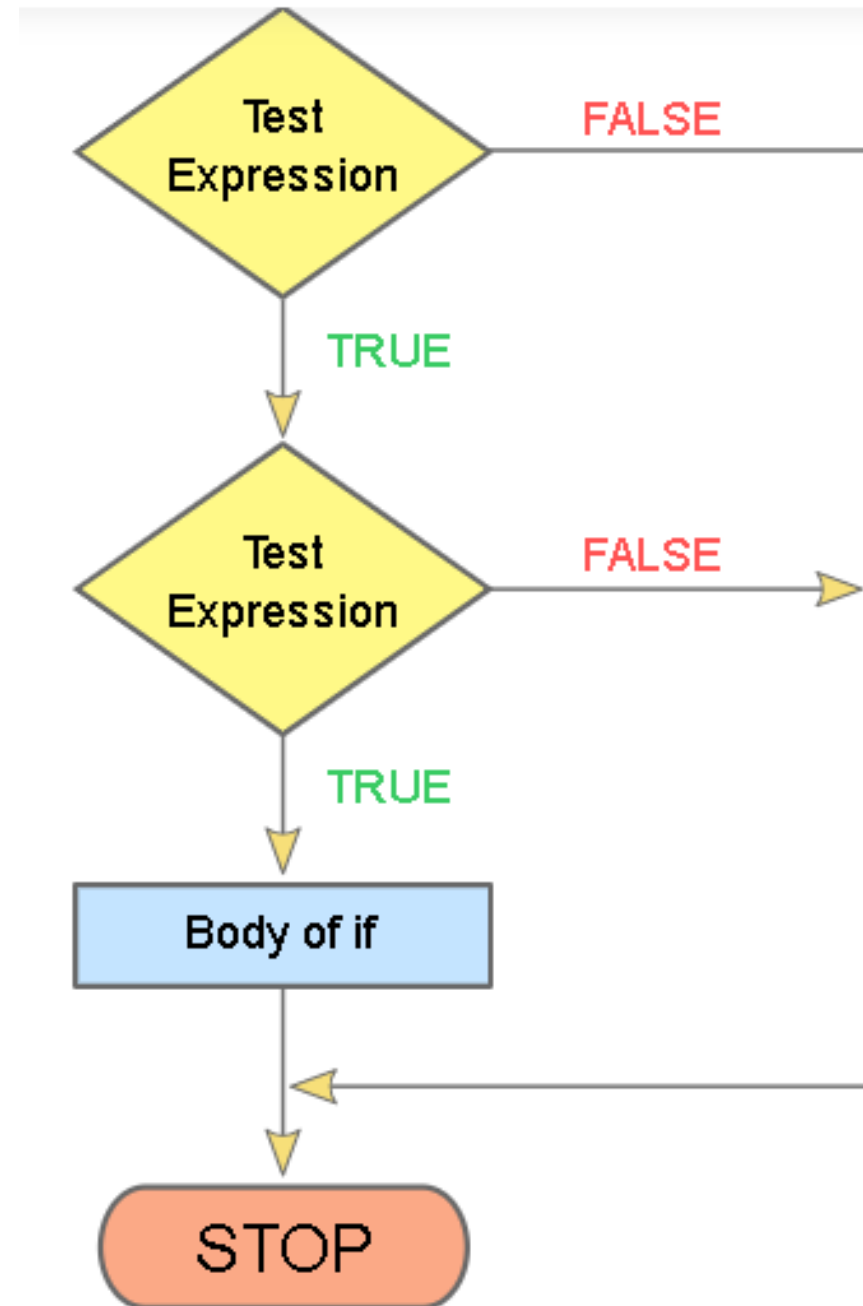
- Using One If Statement Within Another If Statement Is Known As Nested If else Statement.
- Nested if statements are often used when you must test a combination of conditions before deciding on the proper action.

Conditional statements

3. NESTED IF ELSE

syntax-

```
if(conditional_expression1)
{
    if(conditional_expression2)
    {
        statement1;
    }
    else
    {
        statement2;
    }
}
else
{
    statement 3;
}
```



Conditional statements

3. NESTED IF ELSE

```
#include <stdio.h>
void main()
{
    int num1, num2, num3, small;
    printf("\nEnter Three numbers = ");
    scanf("%d%d%d", &num1, &num2, &num3);
    if(num1 < num2)
    {
        if(num1 < num3)
            small = num1;
        else
            small = num3;
    }
    else
    {
        if(num2 < num3)
            small = num2;
        else
            small = num3;
    }
    printf("\nsmallest number = %d", small);
}
```

Conditional statements

4. Cascaded if-else or else if ladder

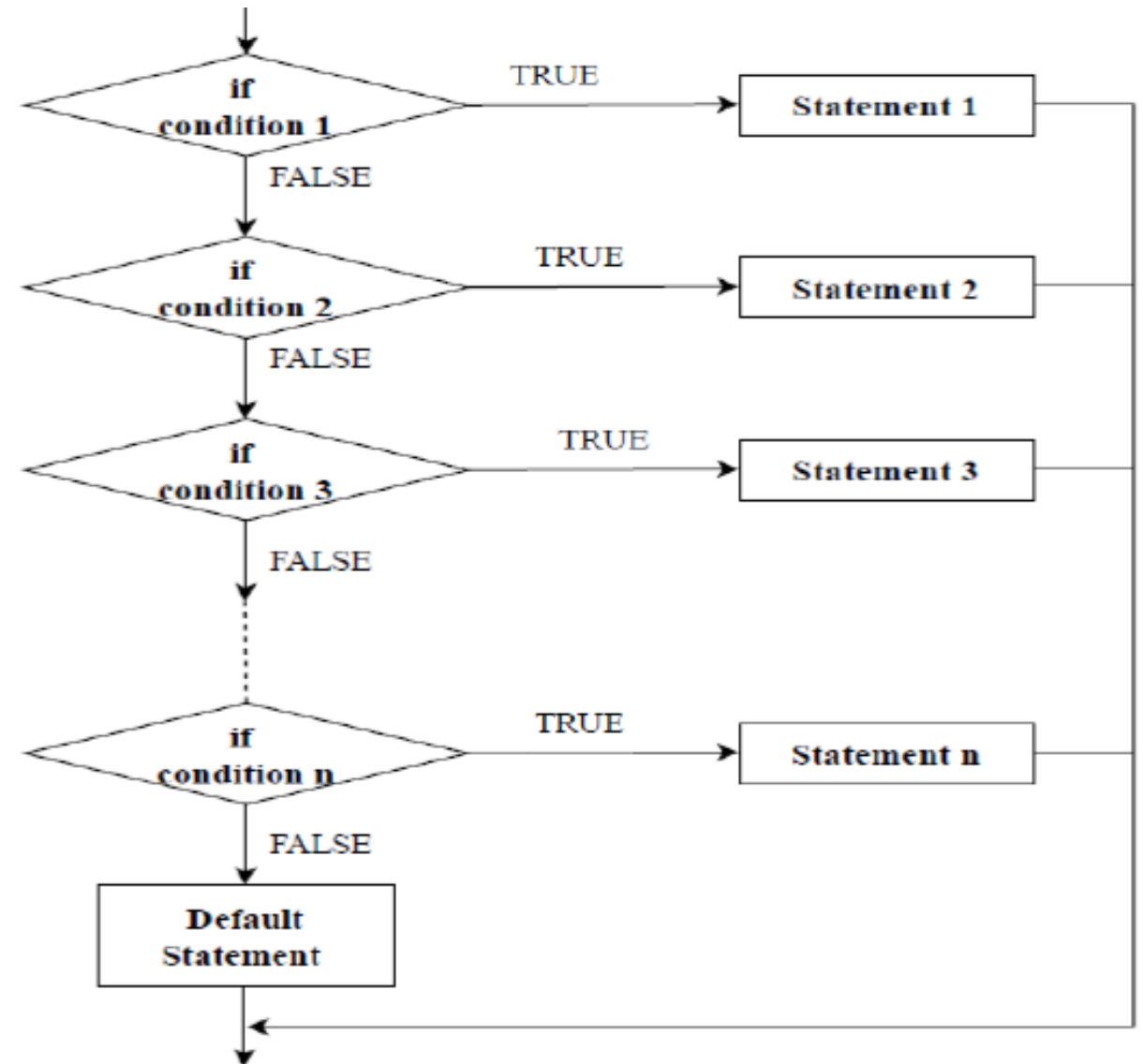
- With an if or if/else statement we evaluate a single true/false condition. A cascaded if statement, on the other hand, makes it possible to evaluate several conditions in a row. This type of if statement has several if code blocks placed below each other, with optional else code at the end.
- Here the conditions are evaluated from top to bottom. As soon as the true condition is found the statement associated with it is executed and the control is transferred to statement X, skipping the rest of the ladder.

Conditional statements

4. Cascaded if-else or else if ladder

Syntax: `if(condition 1)`
 `statement 1;`
 `else if(condition 2)`
 `statement 2;`
 `else if(condition 3)`
 `statement 3;`

 `else if(condition n)`
 `statement n;`
 `else`
 `default statement;`
 `statement X;`



Conditional statements

4. Cascaded if-else or else if ladder

```
#include<stdio.h>
void main()
{
    float avg;
    printf("Enter the Average marks\n");
    scanf("%f",&avg);
    if(avg>=80)
        printf("Distinction\n");
    else if(avg>=60)
        printf("First Division\n");
    else if(avg>=50)
        printf("Second Division\n");
    else if(avg>=40)
        printf("Third Division\n");
    else
        printf("Fail\n");
}
```

```
Enter the Average marks
90
Distinction
```

```
Enter the Average marks
79
First Division
```

```
Enter the Average marks
59
Second Division
```

```
Enter the Average marks
41
Third Division
```

```
Enter the Average marks
30
Fail
```

Conditional statements

5. SWITCH CASE CONDITION

- It is multiple conditioned checking statements, which is generally used for menu- driven program.
- Where we have to select one option out of several options at a time. The number of —case within switch – statement is same as the number of options present in menu.
- Each —case is used to do only one work at a time.

Conditional statements

5. SWITCH CASE CONDITION

Syntax:

```
switch(expression)
{
case    constant1:    statement
sequence
break;
case    constant2:    statement
sequence
break;
...
default: statement sequence
break;
}
```

```
// Program to create a simple calculator
#include <stdio.h>
int main() {
    char operation;
    double n1, n2;
    printf("Enter an operator (+, -, *): ");
    scanf("%c", &operation);
    printf("Enter two operands: ");
    scanf("%lf %lf", &n1, &n2);
    switch(operation)
    {
        case '+':
            printf("%.11f + %.11f = %.11f", n1, n2, n1+n2);
            break;
        case '-':
            printf("%.11f - %.11f = %.11f", n1, n2, n1-n2);
            break;
        case '*':
            printf("%.11f * %.11f = %.11f", n1, n2, n1*n2);
            break;
        // operator doesn't match any case constant +,
        default:
            printf("Error! operator is not correct");
    }
    return 0;
}
```

```
Enter an operator (+, -, *): +
Enter two operands:
4
5
4.0 + 5.0 = 9.0
```

```
Enter an operator (+, -, *): *
Enter two operands: 5
0 * 5.0 = 25.0
```

```
Enter an operator (+, -, *): ?
Enter two operands: 6
7
Error! operator is not correct
```

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

```
void main()
```

```
{
```

```
int age;
```

```
printf("Please Enter Your Age Here:\n");
```

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

```
if (age >= 18)
```

```
    if(age <= 60 )
```

```
    {
```

```
        printf("You are Eligible to Work \n");
```

```
    }
```

```
else
```

```
{
```

```
    printf("You are too old to work \n");
```

```
}
```

```
else
```

```
{
```

```
    printf("Not Eligible to Work");
```

```
}
```

```
}
```

NESTED IF ELSE

// Write a C program to check whether the person is eligible for work using nested if

- 1) If age is in between 18 – 60 eligible to work
- 2) If age >60 too old to work
- 3) If age < 18 not eligible to work

```
Please Enter Your Age Here:
21
You are Eligible to Work
```

```
Please Enter Your Age Here:
65
You are too old to work
```

```
Please Enter Your Age Here:
16
Not Eligible to Work
```



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

```
void main()
```

```
{    int num;  
    printf("Enter a number:\n");  
    scanf("%d", &num);  
    if (num > 0)  
    {  
        printf("Positive");  
    }  
    else if(num < 0)  
    {  
        printf("Negative");  
    }  
    else  
    {  
        printf("Zero");  
    }  
}
```

Cascaded if-else or else if ladder

// C Program to check whether
a number is positive,
negative or zero using if else
if ladder

```
Enter a number:  
5  
Positive
```

```
Enter a number:  
0  
Zero
```

```
Enter a number:  
-3  
Negative
```

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

```
void main()
```

```
{
```

```
    int Semester;
```

```
    printf("Enter the Semester number:\n ");
```

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

```
    printf("The semester you selected is : ");
```

```
    switch (Semester) {
```

```
        case 1:
```

```
            printf("First");
```

```
            break;
```

```
        case 2:
```

```
            printf("Second");
```

```
            break;
```

```
        case 3:
```

```
            printf("Third");
```

```
            break;
```

```
        default:
```

```
            printf("Invalid Input");
```

```
            break;
```

```
    }
```

```
}
```

Switch Statement

// C Program to display the semester in which student is studying using switch statement

```
enter the Semester number:
```

```
3
```

```
The semester you selected is : Third
```

```
enter the Semester number:
```

```
1
```

```
The semester you selected is : First
```

Introduction to Conditional looping statements.

- A set of **statements have to be repeatedly executed** for a **specified number** of times until a condition is satisfied.
- The statements that help us to execute the **set of statements repeatedly** are called as looping condition.

Introduction to Conditional looping statements.

- The various looping constructs in C are:
 - (i) while Loop
 - (ii) do-while Loop
 - (iii) for Loop structs or loop control statements.

Introduction to Conditional looping statements.

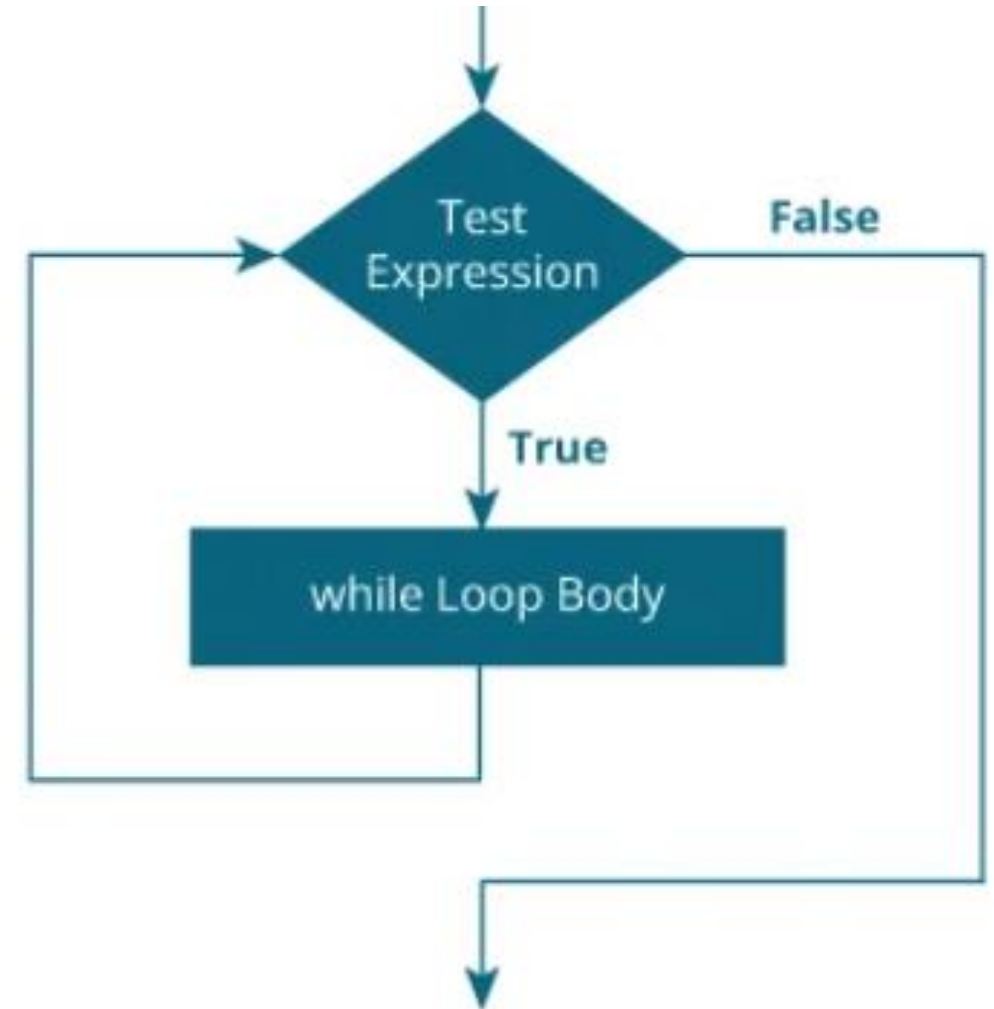
(i) while Loop

- The while loop evaluates the **test Expression** inside the parentheses ().
- If **test Expression** is **true**, statements inside the body of while loop are executed. Then, **test Expression** is evaluated again.
- The process goes on until **test Expression** is evaluated to **false**.
- If **test Expression** is **false**, the loop terminates (ends).

Introduction to Conditional looping statements.

(i) while Loop

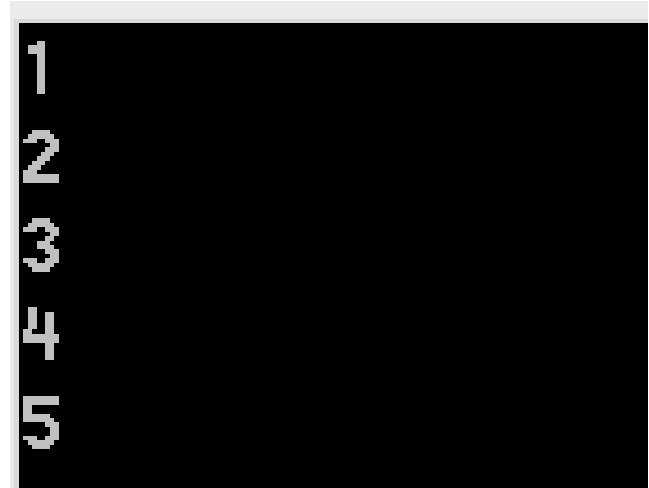
```
initialization;  
while(test condition)  
{  
    set of statements to be executed  
    including increment/decrement  
    opetator  
}
```



Introduction to Conditional looping statements.

(i) while Loop

```
#include<stdio.h>  // Print numbers from 1 to 5
int main()
{
    int i = 1;
    while (i <= 5)
    {
        printf("%d\n", i);
        ++i;
    }
    return 0;
}
```



```
1
2
3
4
5
```

Introduction to Conditional looping statements.

(i) while Loop

Step 1: initialized i to 1.

Step 2: When $i = 1$, the test expression $i \leq 5$ is true. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of i is increased to 2.

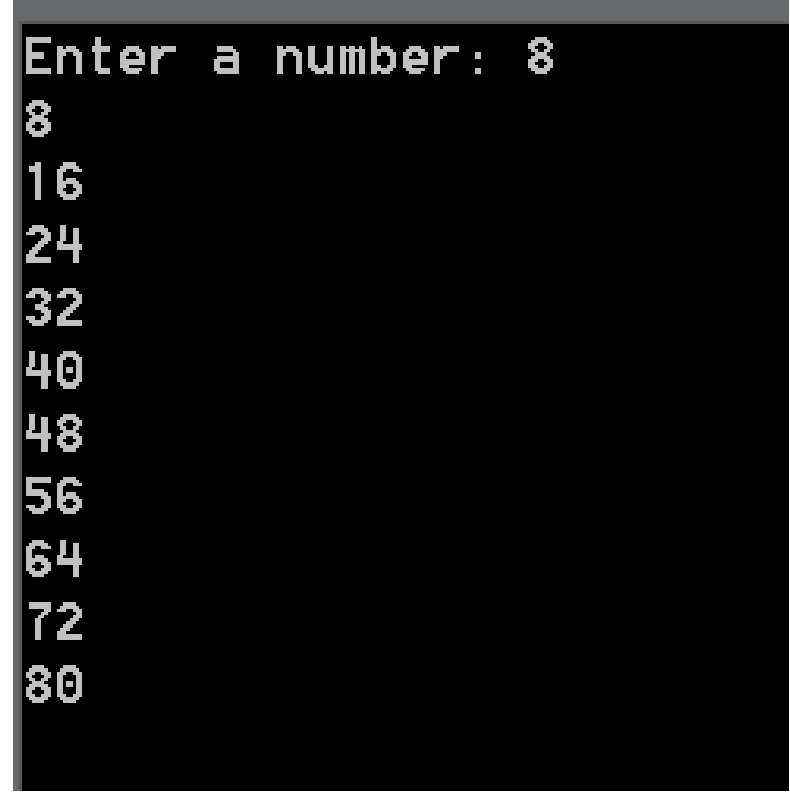
Step 3: Now, $i = 2$, the test expression $i \leq 5$ is again true. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.

Step 4: This process goes on until i becomes 6. Then, the test expression $i \leq 5$ will be false and the loop terminates.

Introduction to Conditional looping statements.

(i) while Loop

```
#include<stdio.h>
int main(){
int i=1,number;
printf("Enter a number: ");
scanf("%d",&number);
while(i<=10){
printf("%d \n",(number*i));
i++;
}
return 0;
}
```



```
Enter a number: 8
8
16
24
32
40
48
56
64
72
80
```

Introduction to Conditional looping statements.

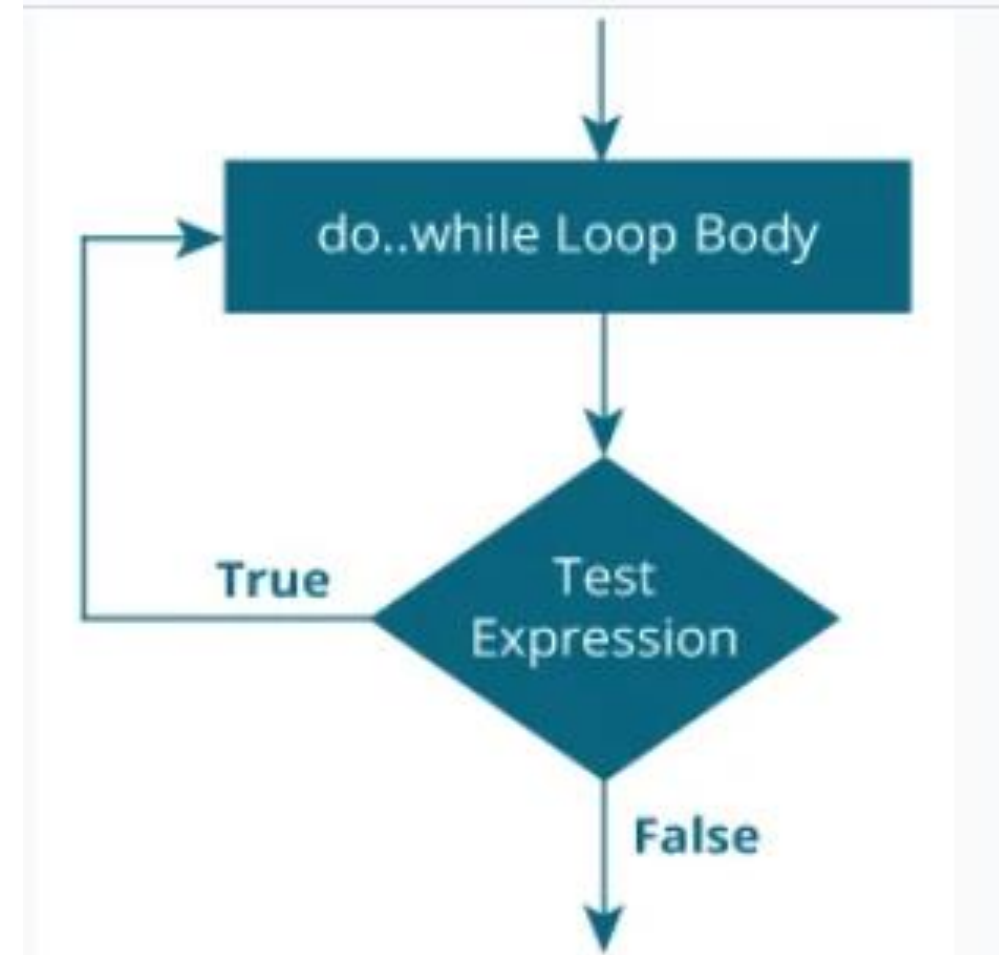
(ii) do- while Loop

- A do-while loop is similar to the while loop except that the **condition is always executed after the body of a loop**. It is also called an exit-controlled loop.
- The body is executed if and only if the condition is true. In some cases, we **have to execute a body of the loop at least once even if the condition is false**.
- This type of operation can be achieved by using a do-while loop. In the do-while loop, the body of a loop is always executed at least once.

Introduction to Conditional looping statements.

(ii) do- while Loop

```
do
{
    // the body of the loop
}
while (testExpression);
```



Introduction to Conditional looping statements.

(ii) do- while Loop

- The body of do...while loop is executed once. Only then, the testExpression is evaluated.
- If testExpression is **true**, the body of the loop is executed again and testExpression is evaluated once more.
- This process goes on until testExpression becomes **false**.
- If testExpression is **false**, the loop ends.

Introduction to Conditional looping statements.

(ii) do- while Loop

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

```
int main()
```

```
{
```

```
    int j=0;
```

```
    do
```

```
    {
```

```
        printf("Value of variable j is: %d\n", j);
```

```
        j++;
```

```
    }
```

```
    while (j<=3);
```

```
return 0;
```

```
}
```

```
Value of variable j is: 0
```

```
Value of variable j is: 1
```

```
Value of variable j is: 2
```

```
Value of variable j is: 3
```

```
int j=4;
```

```
Value of variable j is: 4
```

Introduction to Conditional looping statements.

(ii) do- while Loop

```
#include<stdio.h> // Program to add numbers until the user enters zero
```

```
int main()
{
    int number, sum = 0;
    do
    {
        / the body of the loop is executed at least once
        printf("Enter a number: ");
        scanf("%d", &number);
        sum += number;
    }
    while(number != 0);
    printf("Sum = %d",sum);
    return 0;
}
```

```
Enter a number: 5
Enter a number: 6
Enter a number: 7
Enter a number: 8
Enter a number: 9
Enter a number: 0
Sum = 35
```

```
Enter a number: 0
Sum = 0
-----
```

Introduction to Conditional looping statements.

Difference between While and do- while Loop

While loop	Do while loop
Syntax initialization; while(test condition) { set of statements to be executed including increment/decrement opetator }	Syntax: initialization; do { set of statements to be executed including increment/decrement opetator }while(test condition);
Condition is checked first.	Condition is checked later.
Since condition is checked first, statements may or may not get executed.	Since condition is checked later, the body statements will execute at least once.
The main feature of the while loop is,its an entry controlled loop.	The main feature of the do while loops is it is an exit controlled loop

Introduction to Conditional looping statements.

Difference between While and do- while Loop

```
#include<stdio.h>
void main()
{
    int i;
    i=1;
    while(i<=5)
    {
        printf("%d\t",i);
        i++;
    }
}
```

```
#include<stdio.h>
void main()
{
    int i;
    i=1;
    do
    {
        printf("%d\t",i);
        i++;
    } while(i<=5);
}
```


Introduction to Conditional looping statements.

(iii) for loop

A **for loop** is a more efficient loop structure in 'C' programming which is used when the loop has to be traversed for a fixed number of times. The for loop basically works on three major aspects

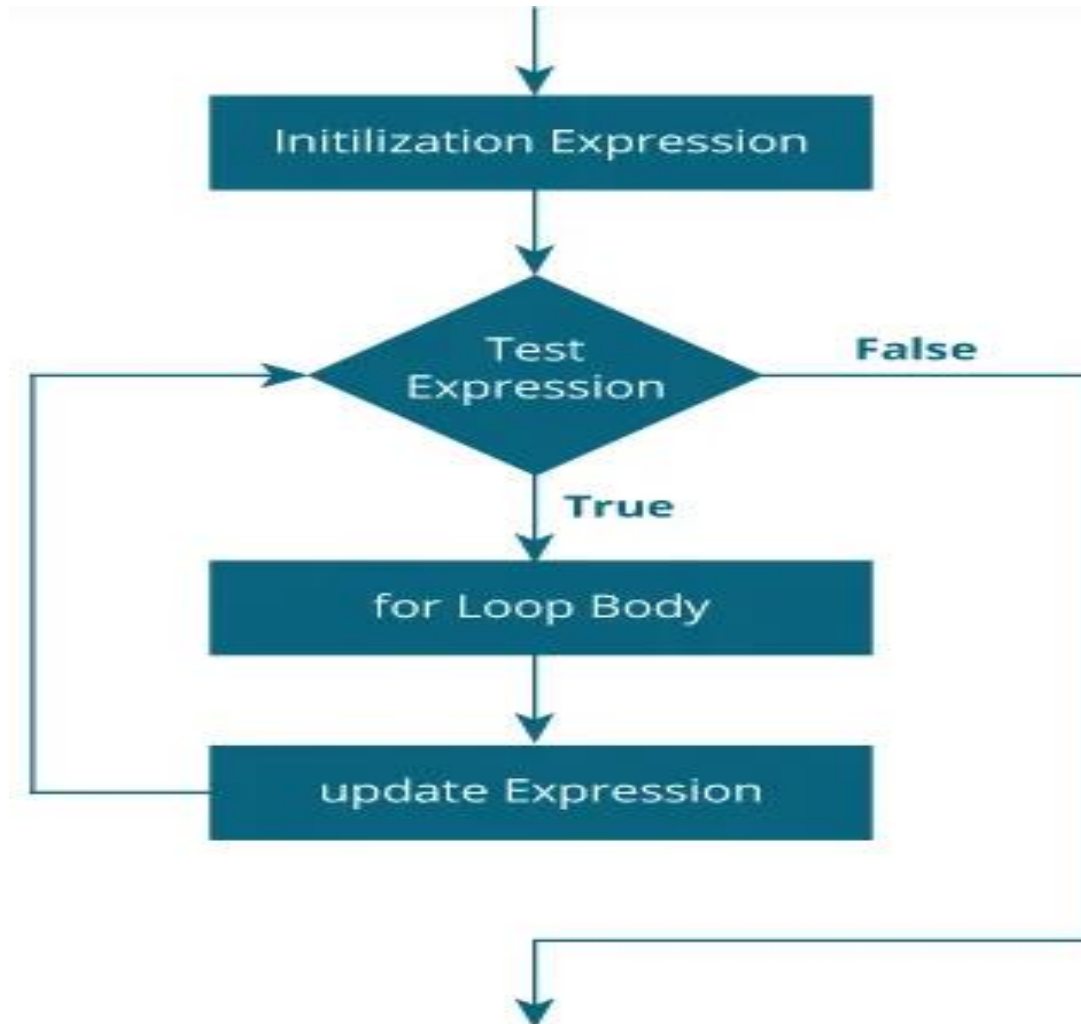
- (i) The **initial value** of the **for loop** is performed only once.
- (ii) The **condition** is a **Boolean expression** that tests and compares the counter to a **fixed value after each iteration**, stopping the for loop when false is returned.
- (iii) The **incrementation /decrementation** increases (or decreases) the counter by a set value.

Introduction to Conditional looping statements.

(iii) for loop

Syntax: **for (initial value; condition; incrementation or decrementation)**

```
{  
    statements;  
}
```



Introduction to Conditional looping statements.

(iii) for loop

```
// Print numbers from 1 to 10
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i < 11; ++i)
    {
        printf("%d ", i);
    }
    return 0;
}
```



```
1 2 3 4 5 6 7 8 9 10
-----
```

3. Write a C program to print sum of first n natural numbers using for loop

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

```
void main()
```

```
{
```

```
int n,i sum=0;
```

```
printf("Enter the value of n\n");
```

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

```
for(i=1;i<=n;i++)
```

```
{
```

```
sum=sum+i;
```

```
}
```

```
printf("Sum of natural numbers=%d\n",sum);
```

```
}
```

Introduction to Conditional looping statements.

For LOOP

While loop

Command	The structure of for loop is – for(initial condition; number of iterations){//body of the loop }	Structure of while loop is- While(condition){statements ;//body}
Iterations	Iterates for a preset number of times.	Iterates till a condition is met.
Initialization	Initialization in for loop is done only once when the program starts.	Initialization is done every time the loop is iterated.
Use	Used to obtain the result only when the number of iterations is known.	Used to satisfy the condition when the number of iterations is unknown.

Introduction to Conditional looping statements.

(iii) for loop

1. i is initialized to 1.

2. The test expression $i < 11$ is evaluated. Since 1 less than 11 is true, the body of for loop is executed. This will print the **1** (value of i) on the screen.

3. The update statement $++i$ is executed. Now, the value of i will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print **2** (value of i) on the screen.

4. Again, the update statement $++i$ is executed and the test expression $i < 11$ is evaluated. This process goes on until i becomes 11.

5. When i becomes 11, $i < 11$ will be false, and the for loop terminates.

Introduction to Conditional looping statements.

(iv) Nested for loop

- Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as “loop inside loop”.
- In nested for loop one or more statements can be included in the body of the loop.
- In nested for loop, The number of iterations will be equal to the number of iterations in the outer loop multiplies by the number of iterations in the inner loop.

Introduction to Conditional looping statements.

(iv) Nested for loop

- When the control moves from outer loop to inner loop the control remains in the inner loop until the inner loop condition fails, once the condition fails the control continues with the outer loop condition Again when the control comes to inner loop the inner loop is reset to the initial value.
- The Nested for loop stops execution when the outer for loop condition fails

Introduction to Conditional looping statements.

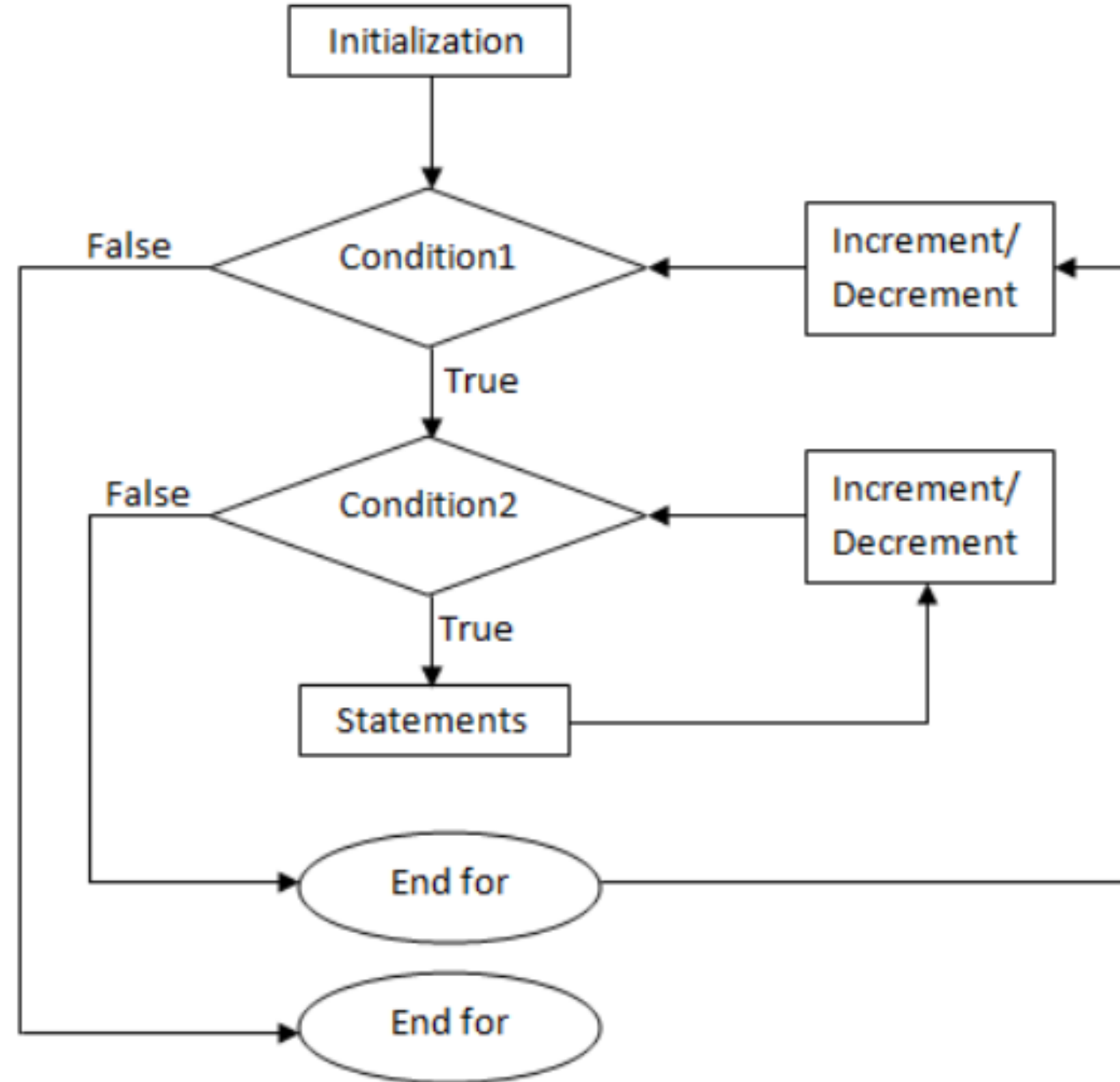
(iv) Nested for loop

Syntax:

```
for ( init; condition; increment ) {  
  
    for ( init; condition; increment ) {  
        statement(s);  
    }  
    statement(s);  
}
```

Introduction to Conditional looping statements.

(iv) Nested for loop



Introduction to Conditional looping statements.

(iv) Nested for loop

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

```
#include <stdio.h>
int main()
{
    int a, b;
    for(a = 1; a <= 5; a++)
    {
        for(b = 1; b <= 5; b++)
        {
            printf("%d ", b);
        }
        printf("\n");
    }
    return 0;
}
```

Output for
inner loop

```
1 2 3 4 5
```

Introduction to Conditional looping statements.

(iv) Nested for loop

```
Enter the number of rows: 6
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

```
#include <stdio.h>
int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    for (i = 1; i <= rows; ++i)
    {
        for (j = 1; j <= i; ++j)
        {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

Output for
inner loop

```
Enter the number of rows: 6
* * * * * *
```

UnConditional looping statements.

- An unconditional statements are the statements which transfer the control or flow of execution unconditionally to another block of statements. They are also called jump statements.
- There are four types of unconditional control transfer statements.
 - (i) break
 - (ii) continue
 - (iii) goto
 - (iv) return

UnConditional looping statements.

i) break Statement: A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop. i.e., the break statement is used to terminate loops or to exit from a switch.

Syntax :

Jump-statement;

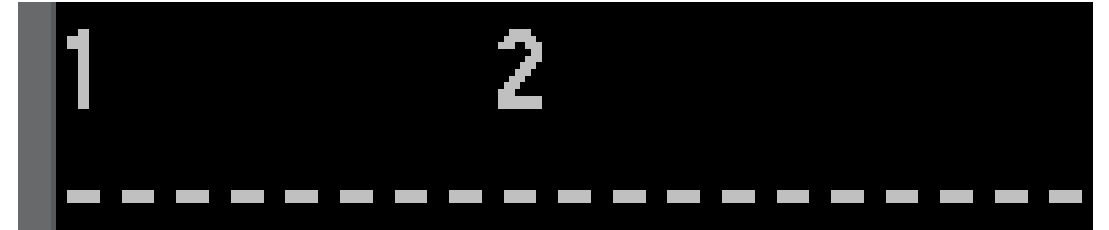
break;

UnConditional looping statements.

i) break Statement:

```
#include<stdio.h>
void main()
{
int i=1;
while(i<=5)
{
if(i==3)
break;
printf("%d\t",i);
i++;

}
}
```



UnConditional looping statements.

(ii) continue statement: The continue statement is used to bypass the remainder of the current pass through a loop.

The loop does not terminate when a continue statement is encountered.

Instead, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.

UnConditional looping statements.

(ii) continue statement:

It is simply written as “continue”. The continue statement tells the compiler “Skip the following Statements and continue with the next Iteration”.

Syntax :

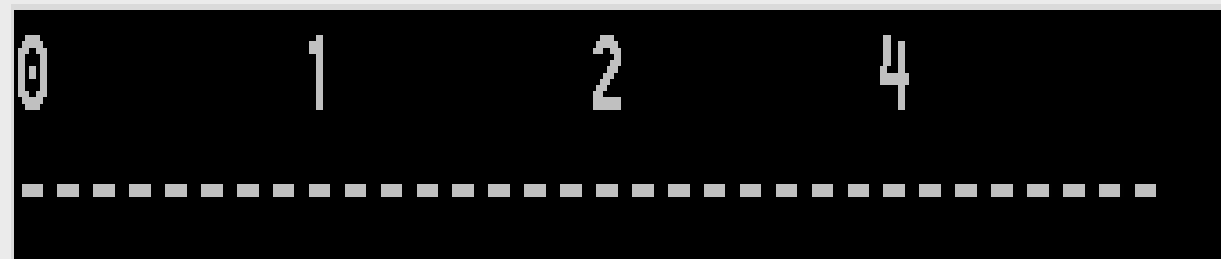
Jump-statement;

Continue;

UnConditional looping statements.

(ii) continue statement:

```
#include<stdio.h>
int main()
{
int i=0;
for(i=0;i<=4;i++)
{
if(i==3)
continue;
printf("%d\t",i);
}
return 0;
}
```



UnConditional looping statements.

(iii)goto statement :

C supports the “goto” statement to branch unconditionally from one point to another in the program.

Although it may not be essential to use the “goto” statement in a highly structured language like “C”, there may be occasions when the use of goto is necessary.

UnConditional looping statements.

(iii)goto statement :

The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name and must be followed by a colon (:).

The label is placed immediately before the statement where the control is to be transferred.

The label can be anywhere in the program either before or after the goto label statement.

UnConditional looping statements.

(iii)goto statement :

Syntax:

goto label;

.....

.....

.....

label: statement;

Forward jump

goto label;

.....

•

label: statement;

Backward jump

label: statement;

.....

.....

goto label;

If the label statement is below the goto statement then it is called **forward jump**. if the label statement is above the goto statement then it is called **backward jump**

UnConditional looping statements.

(iii)goto statement :

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

```
void main()
```

```
{
```

```
printf("MITE \t");
```

```
printf("is \t in\t");
```

```
printf("Moodbidri\n");
```

```
}
```

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

```
int main()
```

```
{
```

```
printf("MITE \t");
```

```
goto label1;
```

```
printf("is \t in\t");
```

```
label1:
```

```
printf("Moodbidri\n");
```

```
Return 0;
```

```
}
```

```
MITE      is      in      Moodbidri
```

```
MITE      Moodbidri
```

Write a C Program to check if the entered number is positive Negative or Zero using goto statement..

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int num;
    printf("Enter the number\n");
    scanf("%d",&num);
    if(num==0)
        goto zero;
    else if(num>0)
        goto pos;
    else
        goto neg;
    zero: printf("The entered number is Zero\n");
    exit(0);
    pos: printf("The entered number is Positive\n");
    exit(0);
    neg: printf("The entered number is Negative\n");
    exit(0);
}
```

```
Enter the number
0
The entered number is Zero
```

```
Enter the number
9
The entered number is Positive
```

```
Enter the number
-8
The entered number is Negative
```

UnConditional looping statements.

(iv) return statement :

- The **return** statement terminates the execution of a function and returns control to the calling function.
- Execution resumes in the calling function at the point immediately following the call. A **return** statement can also return a value to the calling function.
- **Syntax :**

Jump-statement:

return expression;

UnConditional looping statements.

(iv) return statement :

```
#include <stdio.h>
void print() // User defined Function
{
    printf("Welcome to C Programming");
}

int main()
{
    // Calling print
    print();

    return 0;
}
```

3. Write a C program to print sum of first n natural numbers using for loop

```
#include<stdio.h>
void main()
{
int n,i sum=0;
printf("Enter the value of n\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
sum=sum+i;
}
printf("Sum of natural numbers=%d\n",sum);
}
```

4. Write a C program to print fibonacci series up to n numbers using for loop

```
#include<stdio.h>
void main()
{
    int n,i,fib1,fib2,fib3=0;
    printf("Enter the number of series to to be genetared:");
    scanf("%d",&n);
    fib1=0;
    fib2=1;
    if(n==1)
        printf("%d\n",fib1);
    else if(n==2)
        printf("%d\n%d\n",fib1,fib2);
    else
        printf("%d\n%d\n",fib1,fib2);
    for(i=3;i<=n;i++)
    {
        fib3=fib1+fib2;
        printf("%d\n",fib3);
        fib1=fib2;
        fib2=fib3;
    }
}
```

4. Write a C program to print factorial of a given n numbers using for loop

```
#include<stdio.h>
Void main()
{
int i,fact=1,number;
    printf("Enter a number: ");
    scanf("%d",&number);
    for(i=1;i<=number;i++)
    {
        fact=fact*i;
    }
    printf("Factorial of %d is: %d", number, fact);
}
```

Write a C program to determine eligibility for admission to a professional course based on the following criteria:

Eligibility Criteria: Marks in Maths ≥ 65 and Marks in Phy ≥ 55 and Marks in Chem ≥ 50 and Total in all three subjects ≥ 190 or Total in Maths and Physics ≥ 140 .

Input the marks obtained in Physics:65

Input the marks obtained in Chemistry:51

Input the marks obtained in Mathematics:72

Total marks of Maths, Physics and Chemistry: 188

Total marks of Maths and Physics: 137 The candidate is not eligible.

```
#include <stdio.h>

void main()

{ int p,c,m,t,mp;

    printf("Eligibility Criteria :\n");

    printf("Marks in Maths >=65, Phy >=55, Chem>=50 \n");

    printf("and Total in all three subjects >=190 or Total in Maths and Physics >=140\n");

    printf("Input the marks obtained in Physics :");

    scanf("%d",&p);

    printf("Input the marks obtained in Chemistry :");

    scanf("%d",&c);

    printf("Input the marks obtained in Mathematics :");

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

```
printf("Input the marks obtained in Mathematics :");  
  
scanf("%d",&m);  
  
printf("Total marks of Maths, Physics and Chemistry : %d\n",m+p+c);  
  
printf("Total marks of Maths and Physics : %d\n",m+p);  
  
if (m>=65 && p>=55 && c>=50)  
    if((m+p+c)>=190 || (m+p)>=140)  
        printf("The candidate is eligible for admission.\n");  
    else  
        printf("The candidate is not eligible.\n");  
else  
    printf("The candidate is not eligible.\n");  
  
}
```