Module 2: Introduction to C Programming & Control Structures	No. of Hrs: 09L+08P

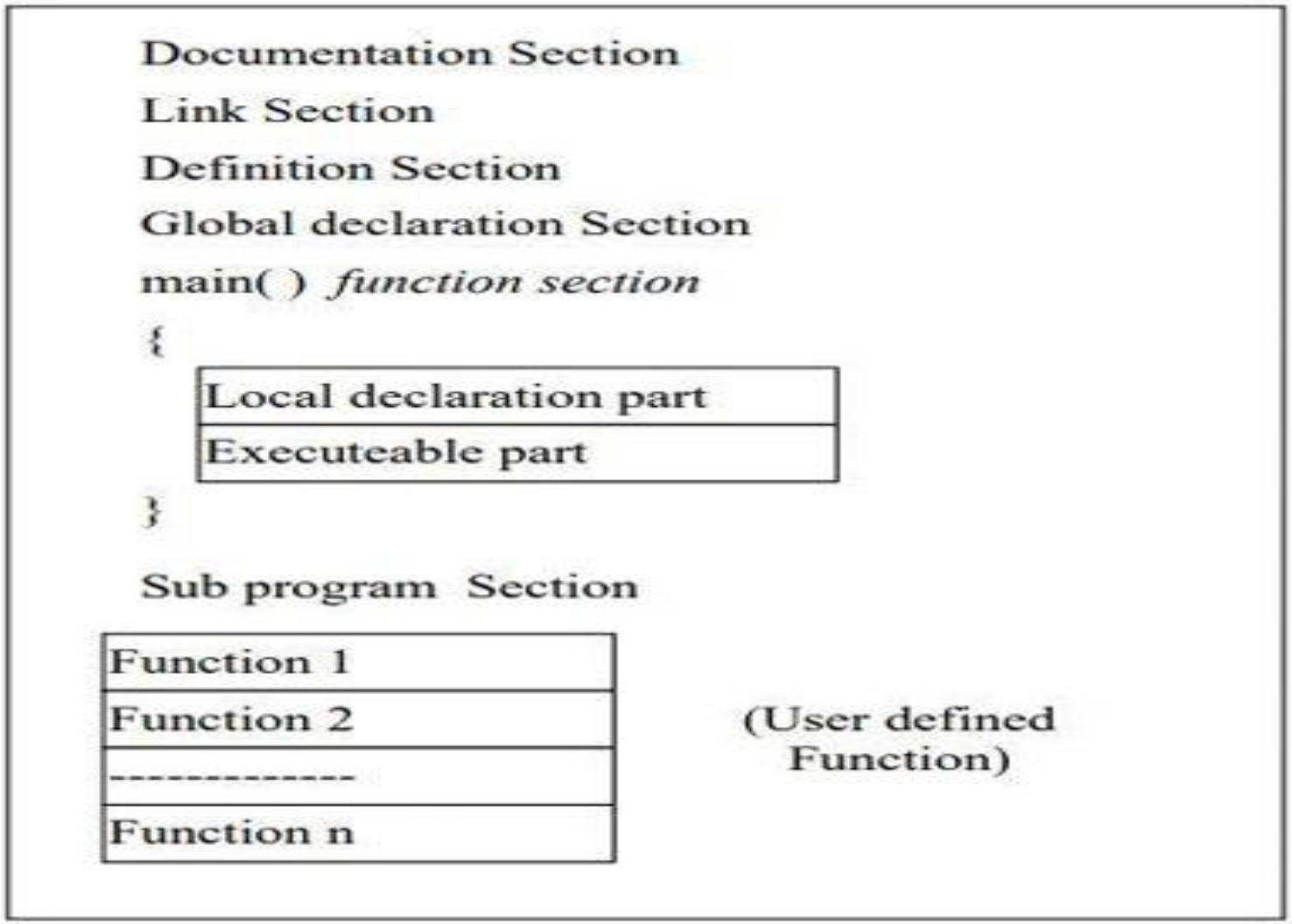
Introduction to C: Structure of C-Program, Header Files, Comments, Character Set, Tokens, Basic data types, Constants, Declarations and Input/output Statements.

Operators: Arithmetic, Relational, Logical, Increment & Decrement, Bitwise, Assignment and Conditional, Type Conversions, Expressions, Precedence and Order of Evaluation.

Decision Making: if, if-else, nested if-else, else-if ladder and switch statements,

Looping: while loop, do-while loop, for loop, Nested Loops, break, continue, and goto statements.

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 used in the 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 (;).
- **sub program section:** contains all the user defined functions that are called in the main function although they appear in any order.

- **Documentation section** :consists of a set of comment line giving the name of the program, the author, 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. 2. Line comments //this is single line comments
- **Link section:** provides instruction to the compiler to link functions from system library. This Section is also called as pre-processor Statements. Every program begins with the preprocessor Statement **#include<stdio.h>**
- **Definition section:** defines all symbolic constants
- **Global Declaration section:** There are some variables that are used in more than on function, such a variable 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.

/* C program to display a welcome message */

```
#include<stdio.h>
void main( )
{
        printf(“Welcome to C Programming for Problem solving”);
}
```

C Tokens:

- In C program the smallest logically meaning full individual units are known as c tokens.
- These are also called as the basic building blocks of C program which cannot be further broken into subparts.
- C has 6 Different types of tokens.
- C programs are written using these tokens and syntax of the language.

C Tokens:

i. Keywords
ii. Identifiers
iii. Constants
iv. Strings
v. Operators
vi. Special symbols

Keywords:

- These are predefined words in C compiler which are ment for specific purpose. These words are also called as reserved words.
- These words cannot be used as variable names.
- These words are usually case sensitive and are usually written in lower case letters only. There are 32 keywords in C.

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

Identifiers:

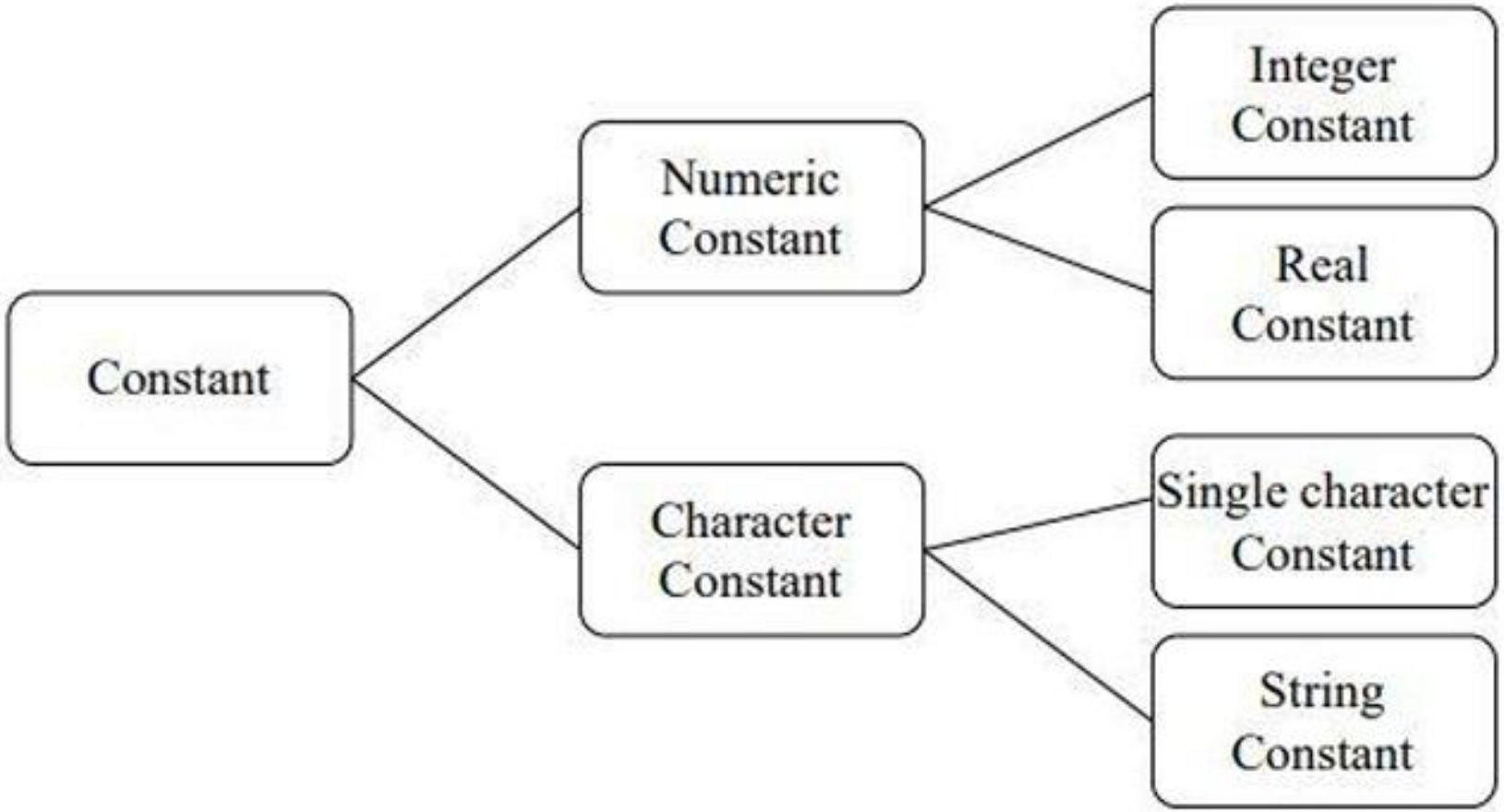
- These are the names given to various elements of the C program like variables, functions, arrays, etc.
- These are user defined names and consist of sequence letters, digits or underscore.
- Rules to define an Identifiers
  - The first character of the identifier must always be a letter or an underscore followed by any number of letters digits or underscore.
  - Keywords cannot be used as identifiers or variables.
  - An Identifier or a variable should not contain two consecutive underscores
  - Whitespaces and special symbols cannot be used to name the identifiers.
  - Identifiers are case sensitive (A same variable name declared in uppercase letters and lower case letters are two different variables in C program).

Examples of valid & invalid identifiers

- food_court valid identifier
- $num Invalid identifier ($ is a special symbol)
- _mite2021 valid identifier
- mite mangalore Invalid identifier (white spaces are not allowed)
- continue Invalid identifier (continue is a keyword)

Constants:

- These are the fixed values assigned to the variables which cannot be cannot be changed or modified in the program.
- Constants are broadly classified as



Numeric Constants:

Integer Constant: These contain digits or whole numbers without decimal point which can be either positive or negative.

(i) Decimal: It is an integer constant consisting of numbers from 0-9. It can be preceded by + or –

(ii) Octal: It is an integer constant consisting of numbers from 0-7. It is preceded by o

(iii) Hexadecimal: It is an integer constant consisting of numbers from 0-9, A-F (A=10, B=11, C=12, D=13, E=14, F=15). It is preceded by 0x

Real Constant:

- These contain an decimal point or an exponent or both. It can be either positive or negative or both

Example: 21.5, 3.142, $6.6260X10^{-34}$, $2.15X10^{2} \rightarrow$ 2.15e2

Character Constants:

- **Single Character Constant:** can be single character enclosed within single quotes or a '\' (backslash) followed by any character.
- '\' is called escape character as it alters the meaning of character following it.
- Following are the complete list of escape sequence.

Character	Meaning	Character	Meaning
'\a'	alert (bell)character	'\ \'	backslash character
'\b'	backspace	'\?'	Question mark
'\f'	formfeed	'\' '	Single quote
'\n'	newline character	'\"'	double quotes
'\r'	carriage return	'\ooo'	octal number
'\t'	horizontal tab	'\xxh'	hexadecimal number
'\v'	vertical tab	'\0'	Null character

String Constant:

- String constants also termed as string literal are sequences of characters enclosed in double quotes.
- The character may be letters, numbers, special characters and blank space.
- A String literal always ends with a Null character (‘\0’)

Example:

M	I	T	E	‘\0’

Operators:

- An operator is a symbol that tells the compiler to perform specific mathematical and logical functions.
- The different operators supported in ‘C’ are:

(i) Arithmetic Operators

(ii) Relational Operators

(iii) Logical Operators

(iv) Assignment Operators

(v) Bitwise Operators

(vi) Unary Operators→ Increment and Decrement

(vii) Ternary/ Conditional Operator

(viii) Special Operators

(i) Arithmetic Operators:

- These operators are used to perform basic arithmetic operations

Operator	Name	Result	Syntax	Example (b=5, c=2)
+	Addition	Sum	a = b + c	a = 7
-	Subtraction	Difference	a = b - c	a = 3
*	Multiplication	Product	a = b * c	a = 10
/	Division	Quotient	a = b / c	a = 2
%	Modulus	Remainder	a = b % c	a = 1

(ii) Relational Operators:

- This operator compares two operands inorder to find out the relation between them. The output will be either 0 (False) or 1 (True).

Operator	Name	Syntax	Example (b=5, c=2)
<	Lesser than	a = b < c	a = 0 (False)
>	Greater than	a = b > c	a = 1 (True)
<=	Lesser than or Equal to	a = b <= c	a = 0 (False)
>=	Greater than or Equal to	a = b >= c	a = 1 (True)
= =	Equal to	a = b = = c	a = 0 (False)
!=	Not equal to	a = b!= c	a = 1 (True)

(iii) Logical Operators:

- These are used to test more than one condition and make decision. The different logical operators are:
  - Logical NOT
  - Logical AND
  - Logical OR
- **Logical NOT (!)** The *output is true* when *input is false* and vice versa. It accepts only one input

Input	Output
X	**!X**
0	1
1	0

(iii) Logical Operators:

- **Logical AND (&&)** The *output is true* only if *both inputs are true*. It accepts two or more inputs.

Input		Output
X	**Y**	**X && Y**
0	0	0
0	1	0
1	0	0
1	1	1

(iii) Logical Operators:

- **Logical OR (||)** The *output is true* only if *any of its input is true*. It accepts two or more inputs

Input		Output
X	**Y**	**X || Y**
0	0	0
0	1	1
1	0	1
1	1	1

(iv) Assignment Operators:

- The assignment operator is used to assign the values to the variables on the left hand side. The symbol "=" is used as an assignment operator.
- Example: **x = 10**, **c = a+b**
- **Shorthand Assignment:** An expression can be written in a compact manner i.e. if the operand on the left hand side of the assignment operator is same as the first operand of the right hand side expression it can be written using the shorthand assignment operator
  - Example: x = x+2 → **x+=2**
- **Multiple Assignment:** If more than one variable holds the same value we can use multiple assignment to avoid rewriting of the same values repeatedly.
  - Example: a=10,b=10,c=10 → **a=b=c=10**

(v) Bitwise Operators:

- These works on bits and performs bit by bit operations. The different types of bitwise operators are:
  - Bitwise NOT (~)
  - Bitwise AND (**&**)
  - Bitwise OR (|)
  - Bitwise XOR (^)
  - Bitwise left shift (<<)
  - Bitwise right shift (>>)
- **Bitwise NOT (~)**

X	~X
0	1
1	0

(v) Bitwise Operators:

Bitwise AND (&), Bitwise OR (|),Bitwise XOR (^)

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

(v) 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

- **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

(vi) Unary Operators:

- Unary Plus Operator & Unary Minus Operator
- Increment (++) & Decrement (--)
  - **Unary Plus and Unary Minus Operator :** These operators are used to determine the sign of the operand the only unary operators are + and –
- **Increment (++):** An increment operator adds one to the operand. The types of increment are Pre increment and Post increment
  - **Pre increment**: The increment operator followed by an operand is called Pre increment operator here the value on the right hand side is first incremented by one and then the value is assigned to the variable at the left.

(vi) Unary Operators:

- Example: If a=8, b= ++a what will be the value of a and b?
  - Since ++a is a pre increment operator first the value of a is incremented by 1 hence the new value of **a=9** now this new value is assigned to the variable b i.e. **b=9** Therefore **a=9** and **b=9**

- **Post increment:** An Operand followed by an increment operator is called post increment operator here the value is first assigned to the variable at the left and then the value of the variable in the right will be incremented by 1
- Example: If q=6, p= q++ what will be the value of p and q?
  - Since q++ is a post increment operator first the value of q is assigned to the variable p i.e. **p=6** now the value of q is incremented by 1 hence the new value of **q=7**, this new value of q will be used for the upcoming iteration Therefore **p=6** and **q=7**

(vi) Unary Operators:

- **Decrement (--):** A decrement operator subtracts 1 from the operand. The types of increment are Pre decrement and Post decrement
- **Pre decrement**: The decrement operator followed by an operand is called Pre decrement operator here the value on the right hand side is first decremented by one and then the value is assigned to the variable at the left.
- Example: If y=4, x= --y what will be the value of x and y?
  - Since --y is a pre decrement operator first the value of y is decremented by 1 hence the new value of **y=3** now this new value is assigned to the variable x i.e. **x=3** Therefore **x=3** and **y=3**
- **Post decrement:** An Operand followed by an decrement operator is called post decrement operator here the value is first assigned to the variable at the left and then the value of the variable in the right will be decremented by 1

(vi) Unary Operators:

- Example: If n=5, m= n-- what will be the value of m and n?
  - Since n-- is a post decrement operator first the value of n is assigned to the variable m i.e. **m=5** now the value of n is decremented by 1 hence the new value of **n=4**, this new value of n will be used for the upcoming iteration Therefore **m=5** and **n=4**

(vii) Ternary/ Conditional Operator:

It takes three arguments

Expression1 ? Expression2 : Expression3

Where,

Expression1→ Condition

Expression2→ Statement followed if condition is true

Expression3→ Statement followed if condition is false

Example:

large = (4 > 2) ? 4: 2 →large = 4

(viii) Special Operators:

- **Comma Operator:** It can be used as operator in expression and as separator in declaring variables.
- **sizeof() operator:** It is used to determine the size of variable or value in bytes.

Data Types:

- These are the keywords that are used to assign the type of a variable based on the type of data stored in it.
- Data types are used to
  - Identify the type of variable when it is used
  - Identify the type of return value of the function
  - Identify the type of parameter expected by the function

Data Types:

- **Primary or built-in or primitive data type**
- These are the data types which are already predefined by the compiler.
  - Integer
  - Floating Point
  - Double
  - Character
  - Void

Data Types:

- **(i) Integer data type**
  - It is used to store whole numbers and its range depends on the word length defined for a computer.
  - It usually occupies 2 or 4 bytes of memory
  - Keyword **int** is used to declare variables of integer data type
  - Format Specifier %d is used to represent integer Variables.

Data Types:

- **(ii) Floating point data type:**
  - It is used to store decimal numbers that have single precision floating point value.
  - It provides 6 digits after the decimal point and occupies 4 bytes of memory
  - Keyword **float** is used to declare variables of floating point data type
  - Format Specifier %f is used to represent floating point variables.

Data Types:

- **(iii) Double data type:**
  - These are used to store real numbers that have double precision floating point value. .
  - It provides 16 digits after the decimal point. this data type is used when performing complex calculations to get accurate results It occupies 8 bytes of memory.
  - Keyword **double** is used to declare variables of floating point data type
  - Format Specifier %lf is used to represent floating point variables.

Data Types:

- **(iv) Character Datatype:**
  - This data type basically stores character type of data.
  - The character data can be an Alphabet [a to z or A to Z] , digits [0 to 9] and all special characters or symbols[@,$,&,#,...]which is enclosed with in single quotes. It occupies 1 byte of memory
  - Keyword **char** is used to declare variables of floating point data type
  - Format Specifier %c is used to represent floating point variables.

Data Types:

- **(v) Void Datatype:**
  - It does not store any value hence we cannot store any operation on the variable declared as void.
  - Void datatype occupies no memory
  - Keyword **void** is used to declare non return type
  - No format Specifiers are used for void.

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:**
- 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.

Example:

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

OUTPUT: 65

Type Conversion:

- **Explicit Type Conversion:**
- This type of conversion is done by the user so it is called explicit type conversion.
- Whenever we are converting wider operand (higher data type variable) into a narrower operand (lower data type variable) then its called explicit conversion

```
#include<stdio.h>
void main()
{
        int a=4;
        float b;
        b=1/(float)a;
        printf("%f",b);
}
```

OUTPUT: 0.250000

Precedence & Associativity:

- **Precedence:** The order in which operators are evaluated is based on the priority value

- **Associativity:** It is the parsing direction used to evaluate an expression. It can be left to right or right to left.

Precedence & Associativity:

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

Precedence & Associativity:

Operator	Description	Associativity	Precedence(Rank)
<<	Left shift	Left to right	5
>>	Right Shift		
<	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 operators	Right to left	14
,	Comma operator	Left to right	15

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 * (**(8 % 5)** * (4 + (15 – 3) / (4 + 2))) //Substitution of values

= 2 * (3 * (4 + **(15 – 3)** / (4 + 2))) //Brackets having the highest priority

= 2 * (3 * (4 +12 /**(4 + 2)**)) //inner most brackets are evaluated first

= 2 * (3 * (4 +**12 / 6**)) //Brackets having the highest priority

= 2 * (3 * **(4 + 2)**) //within the brackets **'/'** has the highest priority

= 2 * **(3 * 6)** // inner most brackets are evaluated

= **2 * 18** // Brackets having the highest priority

= 36 //**Final Result**

100 / 20 <= 10 – 5 + 100 % 10 – 20 == 5 >= 1 != 20

→ **100 / 20** <= 10 – 5 + 100 % 10 – 20 == 5 >= 1 != 20

→ 5 <= 10 – 5 + **100 % 10** – 20 == 5 >= 1 != 20

→ 5 <= **10 – 5** + 0 – 20 == 5 >= 1 != 20

→ 5 <= **5 + 0** – 20 == 5 >= 1 != 20

→ 5 <= **5 – 20** == 5 >= 1 != 20

→ **5 <= -15** == 5 >= 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

Evaluation of Expressions in C

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

a += b *= **c -=5** //Apply Associativity i.e. evaluate from right to left

a += b *= (c = **c – 5**) //Deduce the short hand Expression

a += b *= (c = **8 – 5**) //Substitute the given value of c

a += b *= **(c = 3)** //Reduce the equation to simplified form

a += **b *= 3** //Apply Associativity i.e. evaluate from right to left

a += (b = **b * 3**) //Deduce the short hand Expression

a += (b = **5 * 3**) //Substitute the given value of b

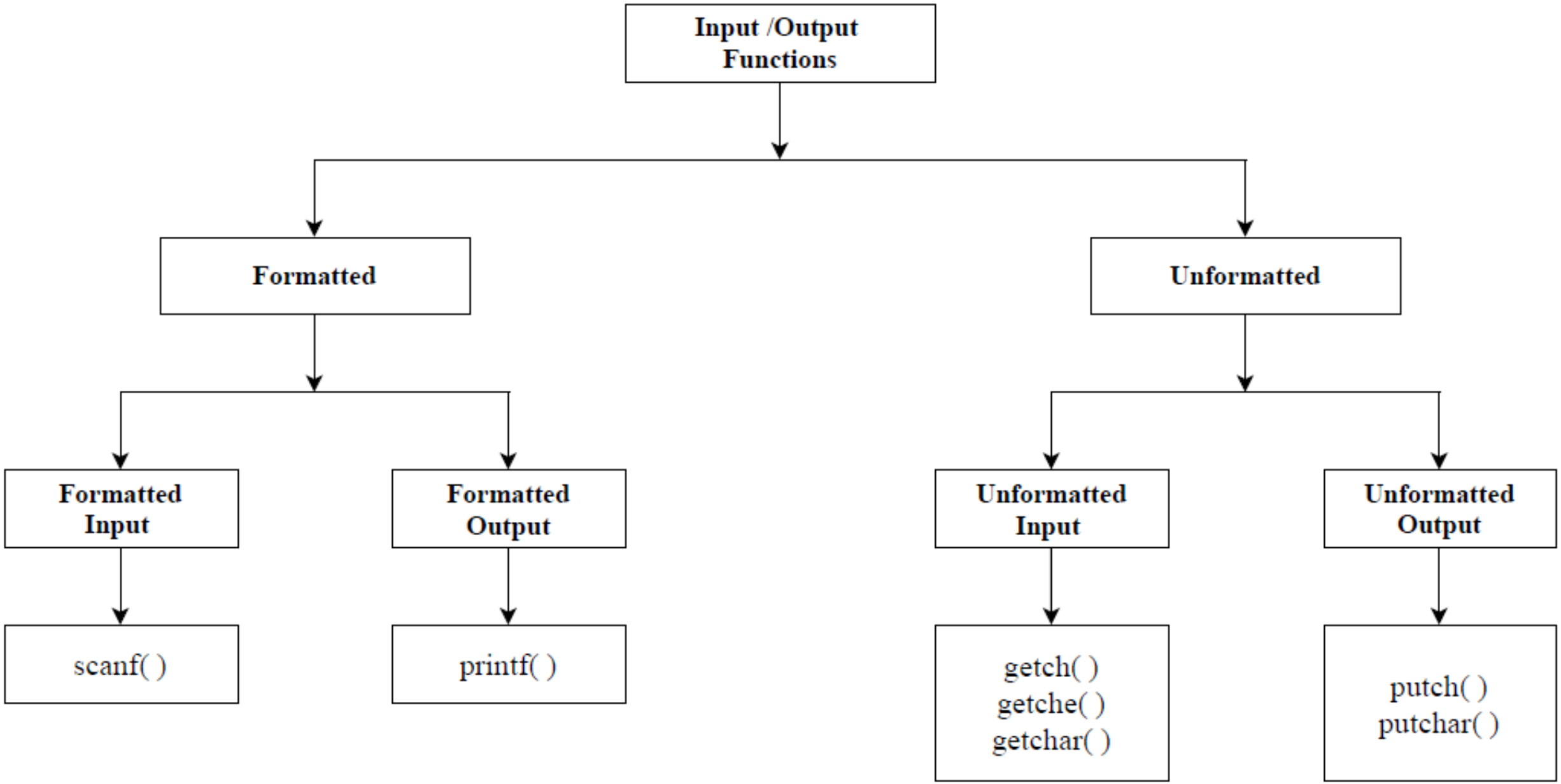
a += **(b = 15)** //Reduce the equation to simplified form

a + = 15 //Apply Associativity i.e. evaluate from right to left

a = **a + 15** // Deduce the short hand Expression

a = **3 + 15** //Substitute the given value of a

a =18 // **Final Result**


Input /Output Functions
Formatted
Unformatted
Formatted Input
Formatted Output
Unformatted Input
Unformatted Output
scanf( )
printf( )
getch( )
getche( )
getchar( )
putch( )
putchar( )

Formatted Input and Output statements

- **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.
- Syntax:
  - *scanf(“format specifier”,addresslist);*

    where:
    - 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.
    - The address list is usually prefixed with an ”&”(ampersand) operator for each variable.

- Example:

  if we want to store the values 50 and 31from the keyboard in variables num1 and num2 then the input function is read as

  scanf(“%d%d”,&num1,&num2);

the value 50 will be assigned to num1 and value 31 will be assigned to num2

- **printf( ):** In C programming language, printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen. The features of printf() can be effectively exploited to control the alignment and spacing of printouts on terminals
- Syntax:

  printf("Text Message");

  OR

  printf("format specifier",variablelist);

  where:

  format specifier indicates the type of data to be displayed

  variable list indicates the value present in the variable.

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

Example:

if we want to display the values stored in variables num1 and num2 then the printf statement can be written as

```
printf("The Value of num1 = %d and The value of num2 =
%d\n",num1,num2);
```

This statement will display the values stored in the respective variables. The output will be of the form:

The Value of num1 = 50 and The value of num2 = 31

Example:

if we want to display the values stored in variables num1 and num2 then the printf statement can be written as

```
printf("The Value of num1 = %d and The value of num2 =
%d\n",num1,num2);
```

This statement will display the values stored in the respective variables. The output will be of the form:

The Value of num1 = 50 and The value of num2 = 31

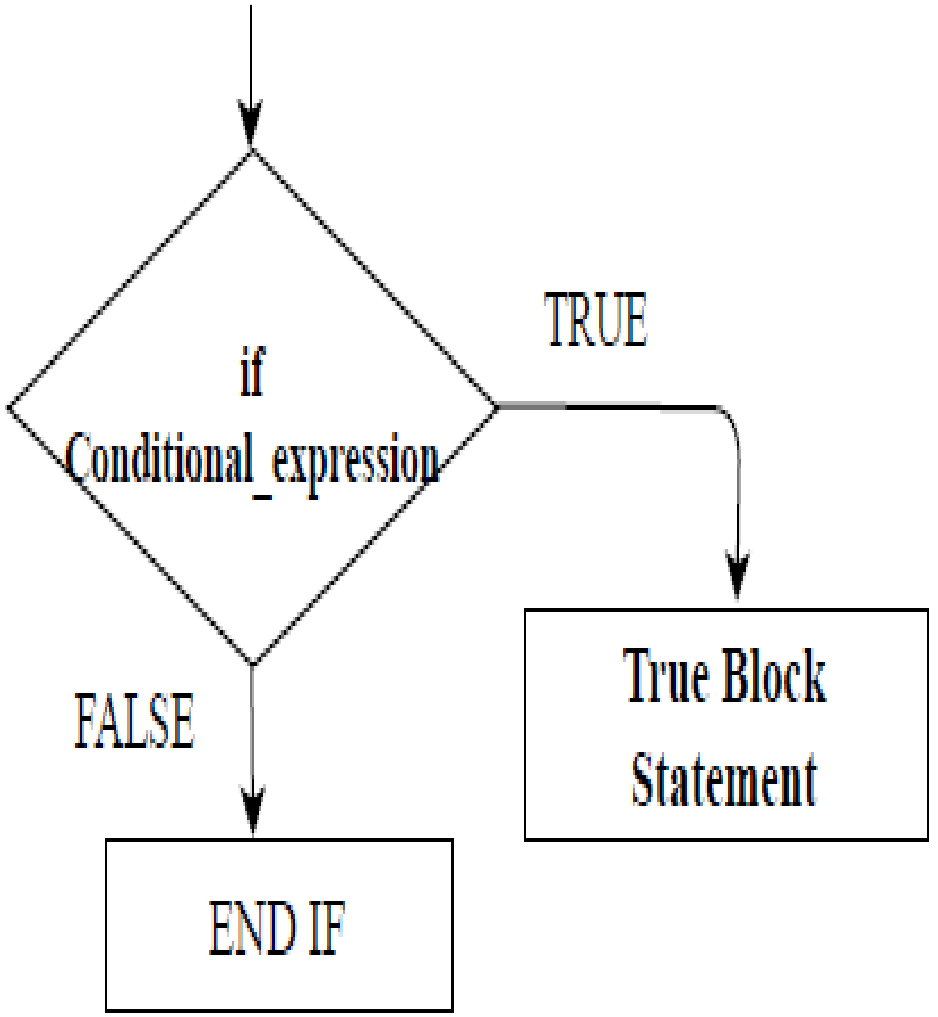
(i) simple if:

- This is a one way selection statement which helps the programmer to execute or skip certain block of statements based on the particular condition.

Syntax:

```
if(conditional_expression)
{
        True block statements;
}
```

- FlowChart



Example:

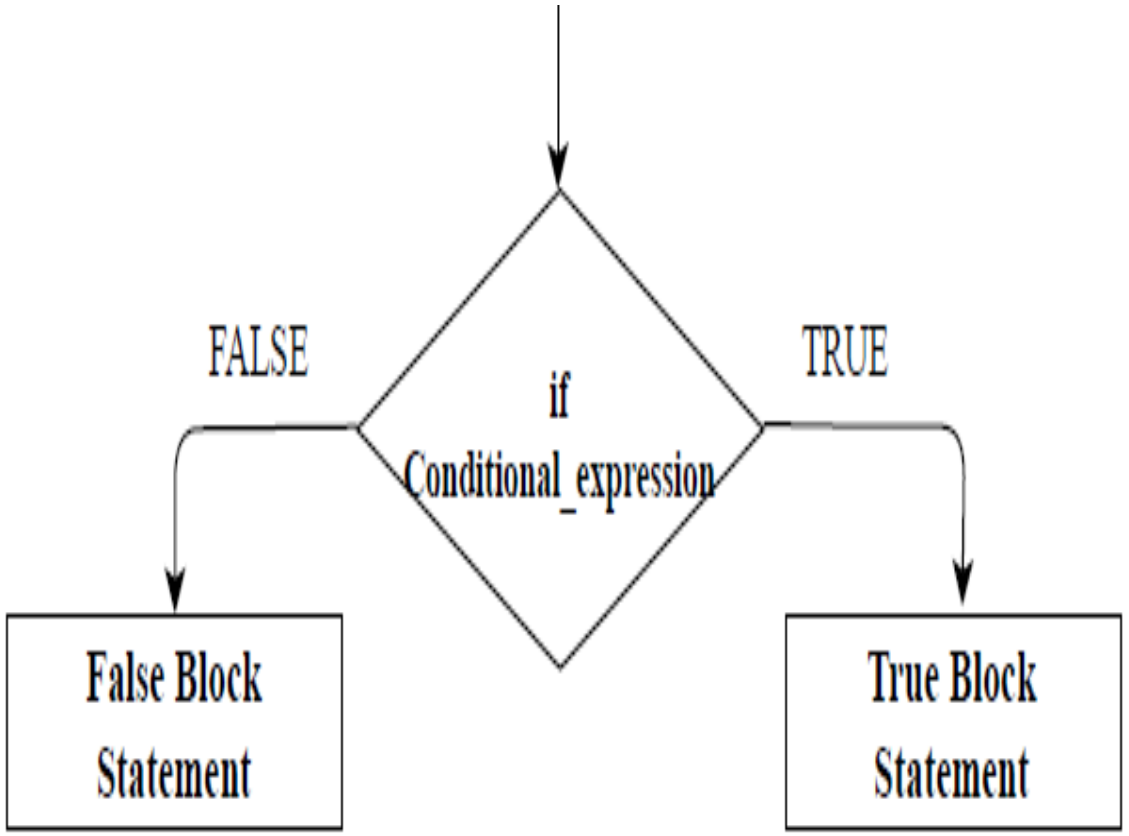
```
/* C program to check the voting eligibility of the person*/
 #include<stdio.h>
void main()
{
        int age;
        printf(" Enter the age of the person\n");
        scanf("%d",&age);
        if(age>=18)
        {
                printf("The person is eligible to vote\n");
        }
        if(age<18)
        {
                printf("The person is not eligible to vote\n");
        }
}
```

(ii) if-else statement:

- This is a two way selection statement which executes true block or false block of statements based on the given condition. The keyword “else” is used to shift the control when the condition is evaluated to false.
- **Syntax:**

```
if(conditional_expression)
{
        True block statements;
}
else
{
        False block statements;
}
```

Flow Chart



Example:

/* C program to check the entered num is even or odd*/

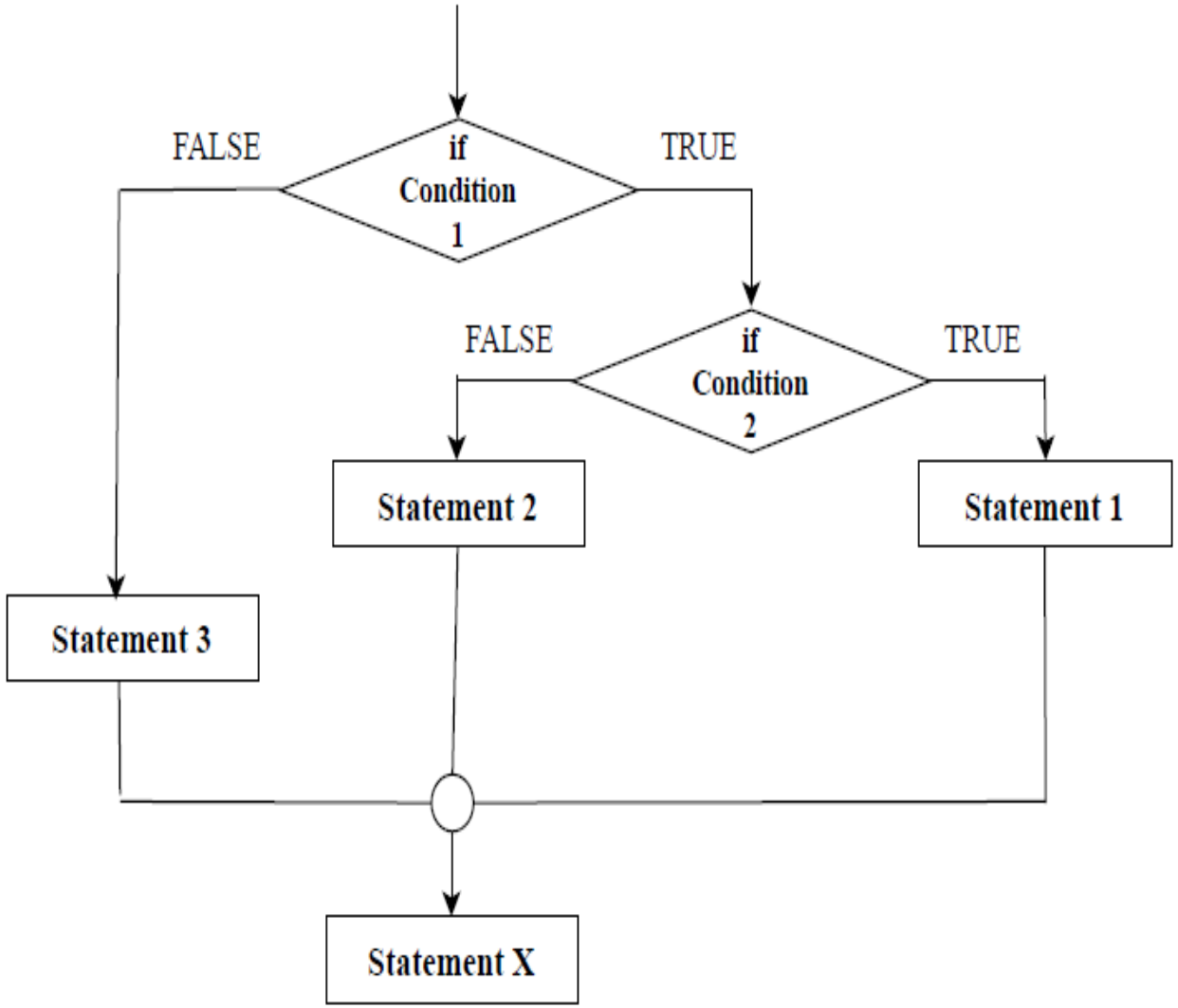
```
#include<stdio.h>
void main()
{
        int num;
        printf(“Enter  a number\n”);
        scanf(“%d”,&num);
        if(num%2==0)
                printf(“%d is a even number\n”,num);
        else
                printf(“%d is a odd number\n”,num);
}
```

(iii) Nested if Statement:

- An if statement within another if statement is called as a nested if statement. This helps the programmer to select one among many alternatives based on a given condition.
- **Syntax**:

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

- Flow chart:



Example:

```
/* C program to find  smallest of three numbers*/
#include<stdio.h>
void main()
{
        int a,b,c,small;
        printf("Enter three numbers\n");
        scanf("%d%d%d",&a,&b,&c);
        if(a<b)
        {
                if(a<c)
                        small=a;
                else
                        small=c;
        }
        else
        {
                if(b<c)
                        small=b;
                else
                        small=c;
        }
        printf("Smallest among three numbers=%d",small);
}
```

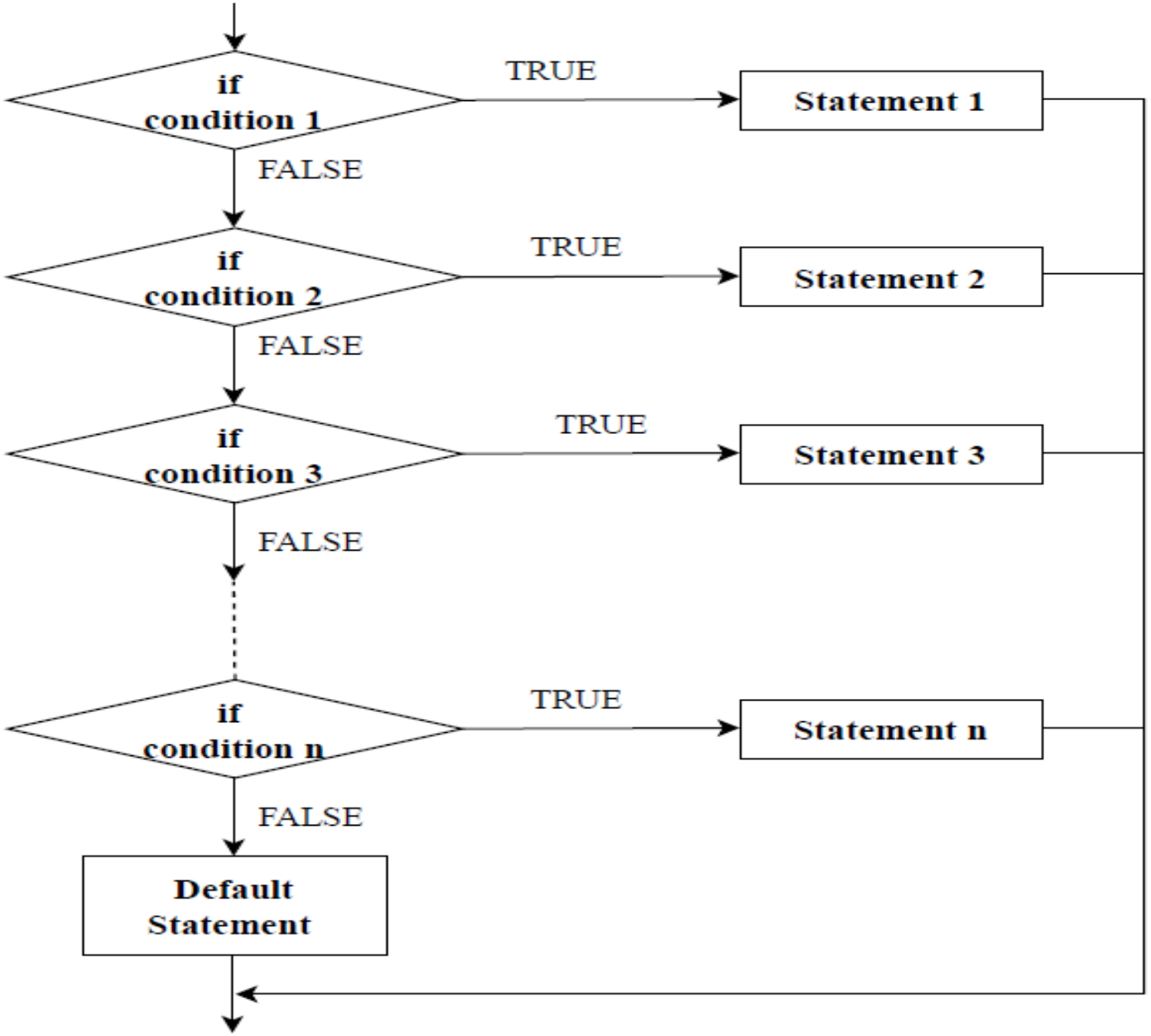
(iv) Cascaded if-else or else if ladder:

- This is another way of putting all if's together when multipath decision are involved The multipath decision is a chain of if statement in which the statement associated with each else is a if statement. 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 rest of the 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;
```

- Flowchart:


if
condition 1
TRUE
Statement 1
FALSE
if
condition 2
TRUE
Statement 2
FALSE
if
condition 3
TRUE
Statement 3
FALSE
if
condition n
TRUE
Statement n
FALSE
Default
Statement

Example:

```
/* C program to display  the grade of the student based on the average marks obtained */
#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");
}
```

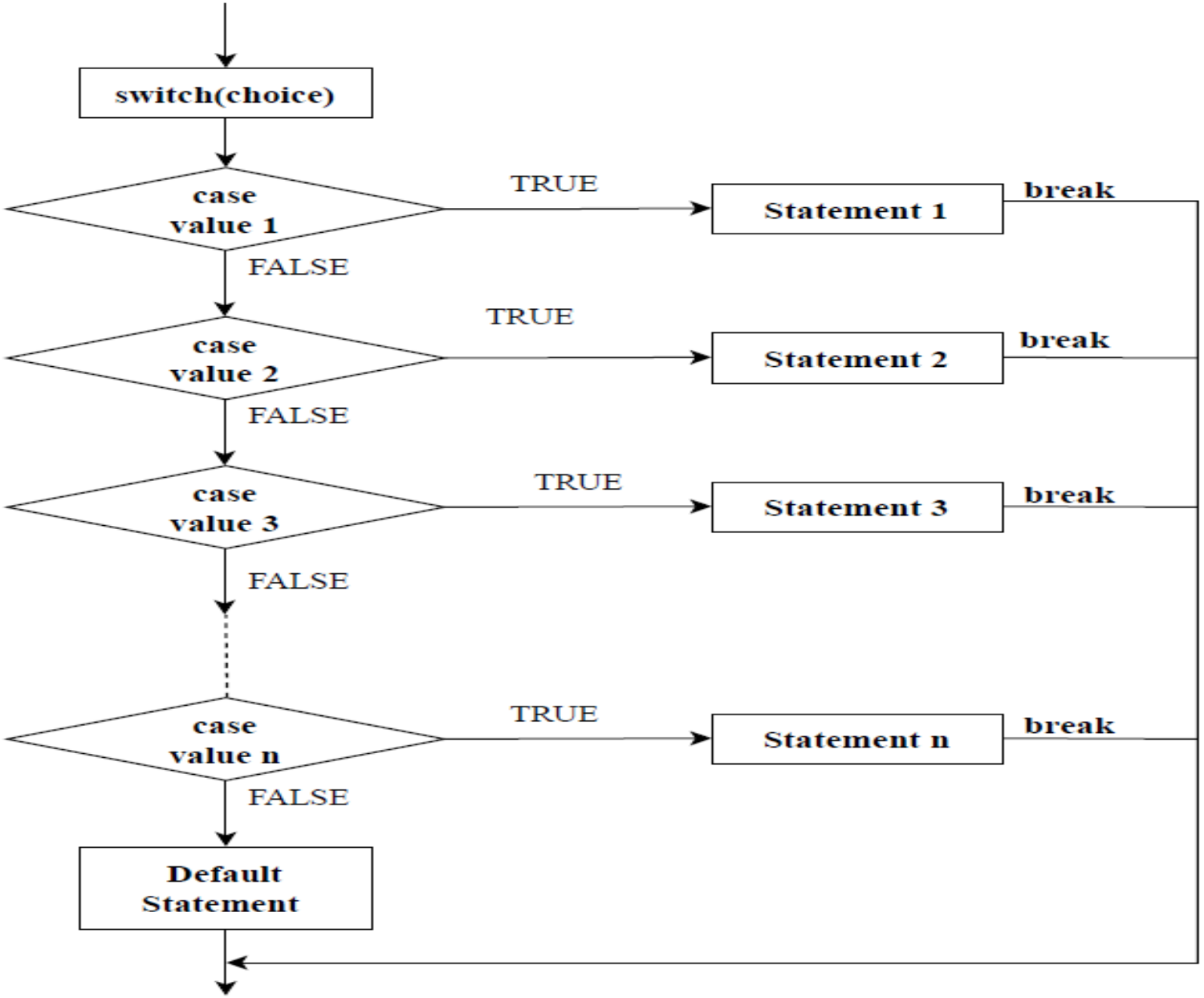
(v)Switch Statement:

- A switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed. Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found. If a case match is not found, then the default statement is executed, and the control goes out of the switch block. The break statement is used at the end of each case to come out of the switch block.

- Syntax:

```
switch( expression )
{
        case value-1:  Statement-1;
                       break;
        case value-2:  Statement-2;
                       break;
        case value-3:  Statement-3;
                       break;
        ----------------------
        ----------------------
        ----------------------
        case value-n:  Statement-n;
                       break;
        default:  default Statement
                       break;
}
Statement-x;
```

- Flowchart:


switch(choice)
case value 1
TRUE
Statement 1
break
FALSE
case value 2
TRUE
Statement 2
break
FALSE
case value 3
TRUE
Statement 3
break
FALSE
case value n
TRUE
Statement n
break
FALSE
Default Statement

Example:

```
/* C program to find Area of various geometric figures*/
#include<stdio.h>
void main()
{
float a,b,area;
int choice;
printf("\n MENU\n")
printf("1. Square\t 2. Circle\t 3. Rectangle\t 4. Triangle\n");
printf("Enter your Choice as 1 OR 2 OR 3 OR 4\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("\nSQUARE\n");
        printf("Enter the Side\n");
        scanf("%d",&a);
        area=a*a;
        break;
case 2: printf("\nCIRCLE\n");
        printf("Enter the Radius\n");
        scanf("%d",&a);
        area=3.142*a*a;
        break;
```

```
case 3: printf("\nRECTANGLE\n");
        printf("Enter the length and breadth\n");
        scanf("%d%d",&a,&b);
        area=a*b;
        break;
case 4: printf("\nTRIANGLE\n");
        printf("Enter the base and height\n");
        scanf("%d%d",&a,&b);
        area=0.5*a*b;
        break;
default: printf("you have entered a wrong choice\n");
exit(0);
        }
        Printf("Area=%f\n",area);
}
```

Looping Constructs

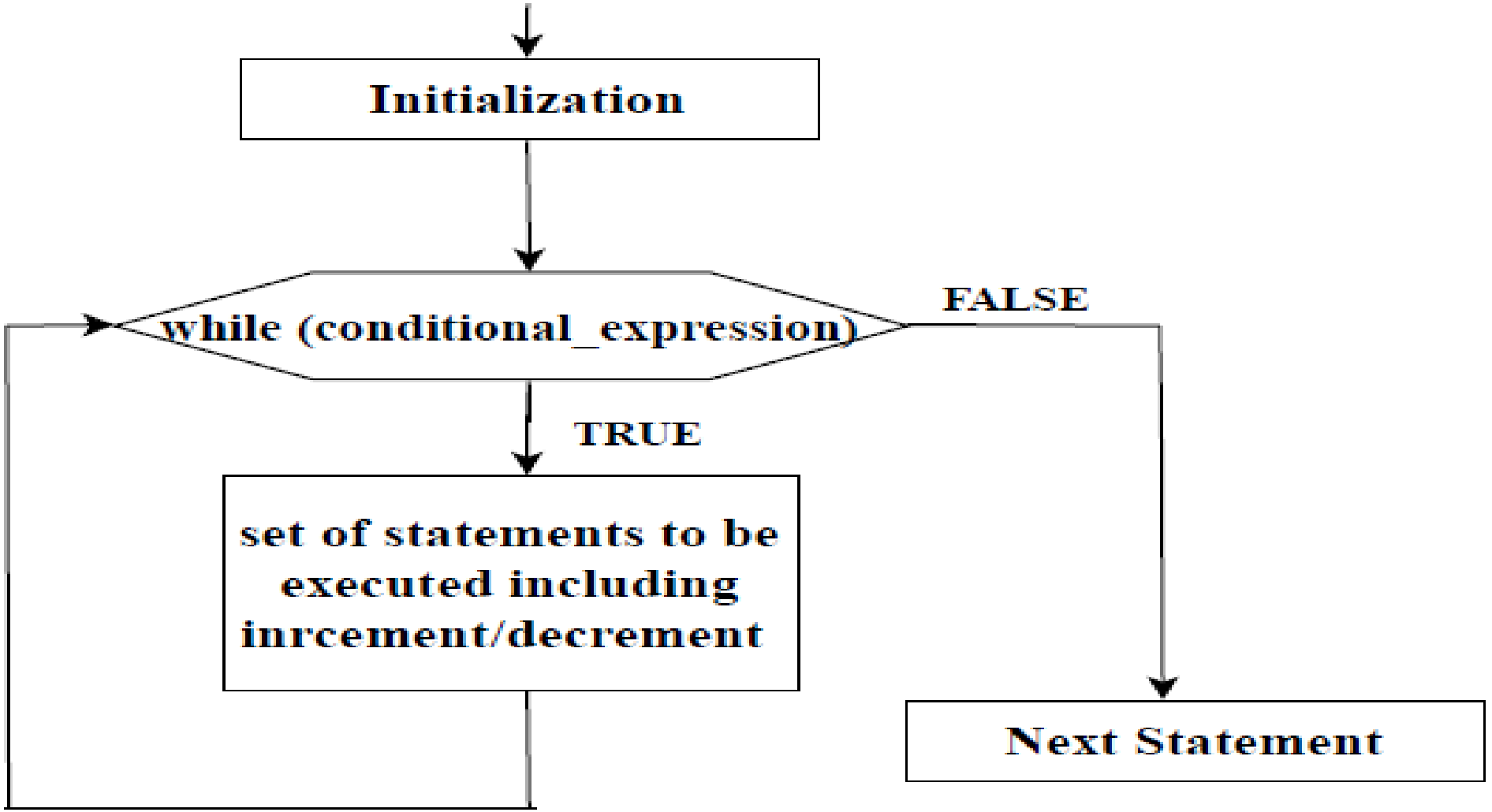
while Loop

- It is an entry-controlled loop.
- In while loop, a condition is evaluated before processing a body of the loop.
- If a condition is true then and only then the body of a loop is executed.
- After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false.
- Once the condition becomes false, the control goes out of the loop.
- After exiting the loop, the control goes to the statements which are immediately after the loop.

Syntax:

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

Flow chart:


Initialization
while (conditional_expression)
FALSE
TRUE
set of statements to be executed including inrcement/decrement
Next Statement

```
/*  c program to find factorial of a number using while
loop*/
#include<stdio.h>
void main()
{
        int n,i ,fact=1;
        printf(" Enter an integer number\n");
        scanf("%d",&n);
        i=1;
        while(i<=n)
        {
                fact=fact*i;
                i=i+1;
        }
        printf("Factorial of the Number= %d\n",fact);
}
```

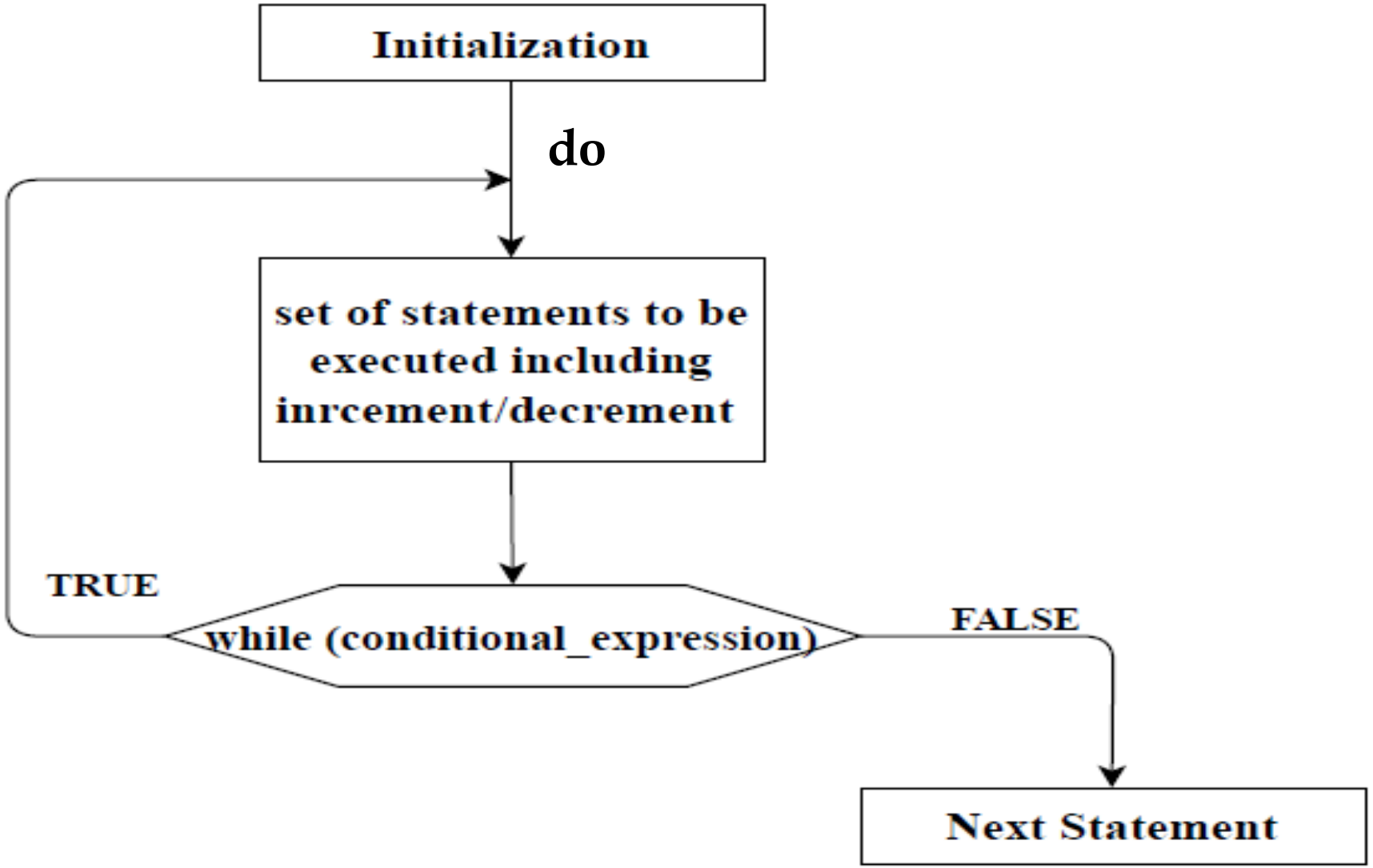
do-while loop

- 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
- In the do-while loop, the body of a loop is always executed at least once.
- After the body is executed, then it checks the condition.
- If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Syntax:

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

Flowchart:


Initialization
do
set of statements to be executed including inrcement/decrement
TRUE
while (conditional_expression)
FALSE
Next Statement

```
/*  c program to find factorial of a number using do-
    while*/
#include<stdio.h>
void main()
{
        int n,i ,fact=1;
        printf(" Enter an integer number\n");
        scanf("%d",&n);
        i=1;
        do
        {
                fact=fact*i;
                i=i+1;
        }while(i<=n);
        printf("Factorial of the Number= %d\n",fact);
}
```

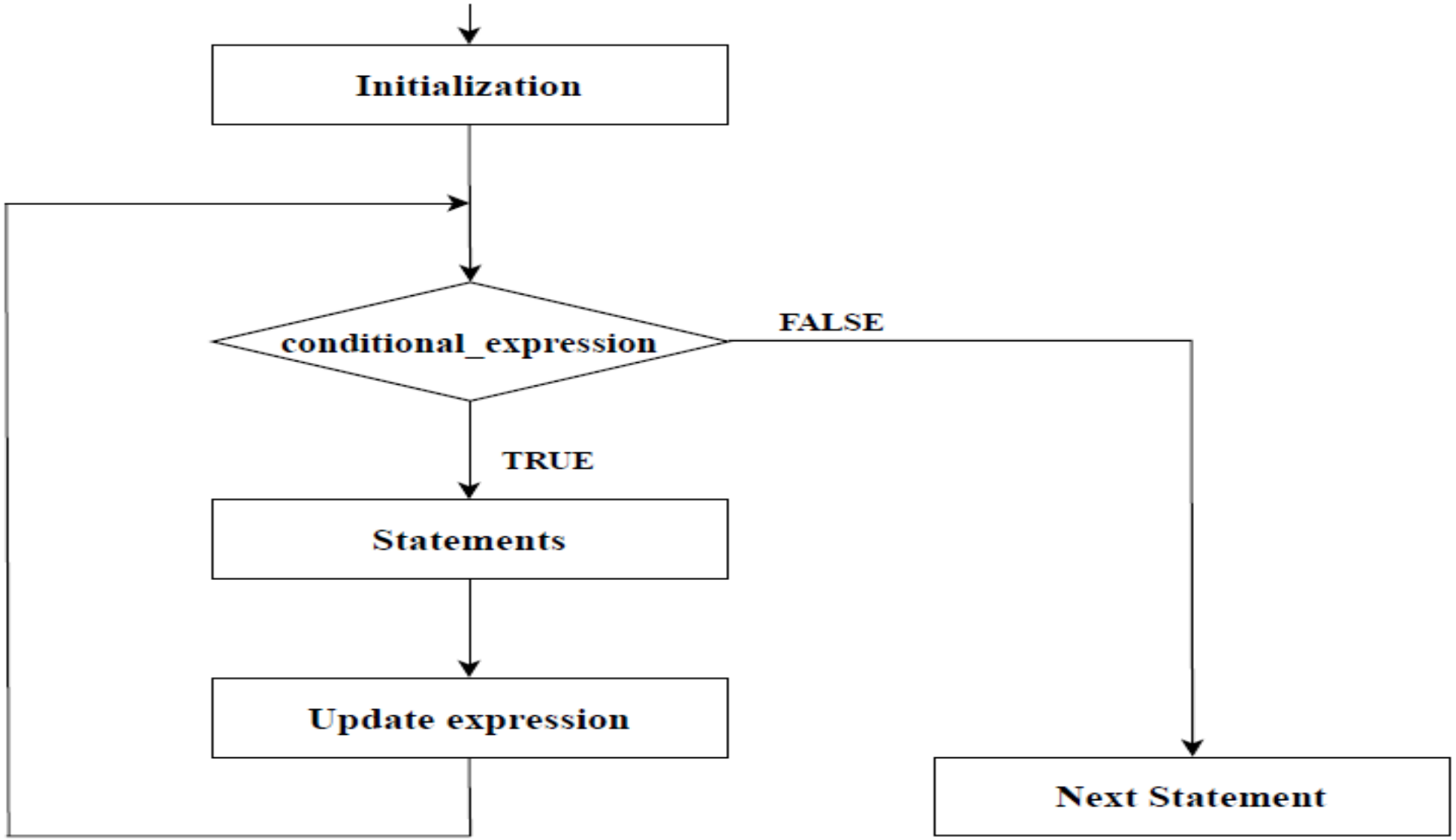
For loop

- A for loop is a more efficient loop structure in 'C' programming
- The initial value of the for loop is performed only once.
- 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.
- The incrementation/decrementation increases (or decreases) the counter by a set value

Syntax:

```
for (initial value; conditionalexpression; incrementation or decrementation )
{
        set of statements to be executed;
}
```

Flowchart:


Initialization
conditional_expression
FALSE
TRUE
Statements
Update expression
Next Statement

```
/*C program to find  sum of N  natural numbers using
for loop*/
#include<stdio.h>
void main()
{
        int n,i,sum=0;
        printf("Enter a positive number\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                sum=sum+i;
        }
        printf("Sum of N natural numbers=%d\n",sum);
}
```

/*C program to generate n Fibonacci series 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;
    }
}
```

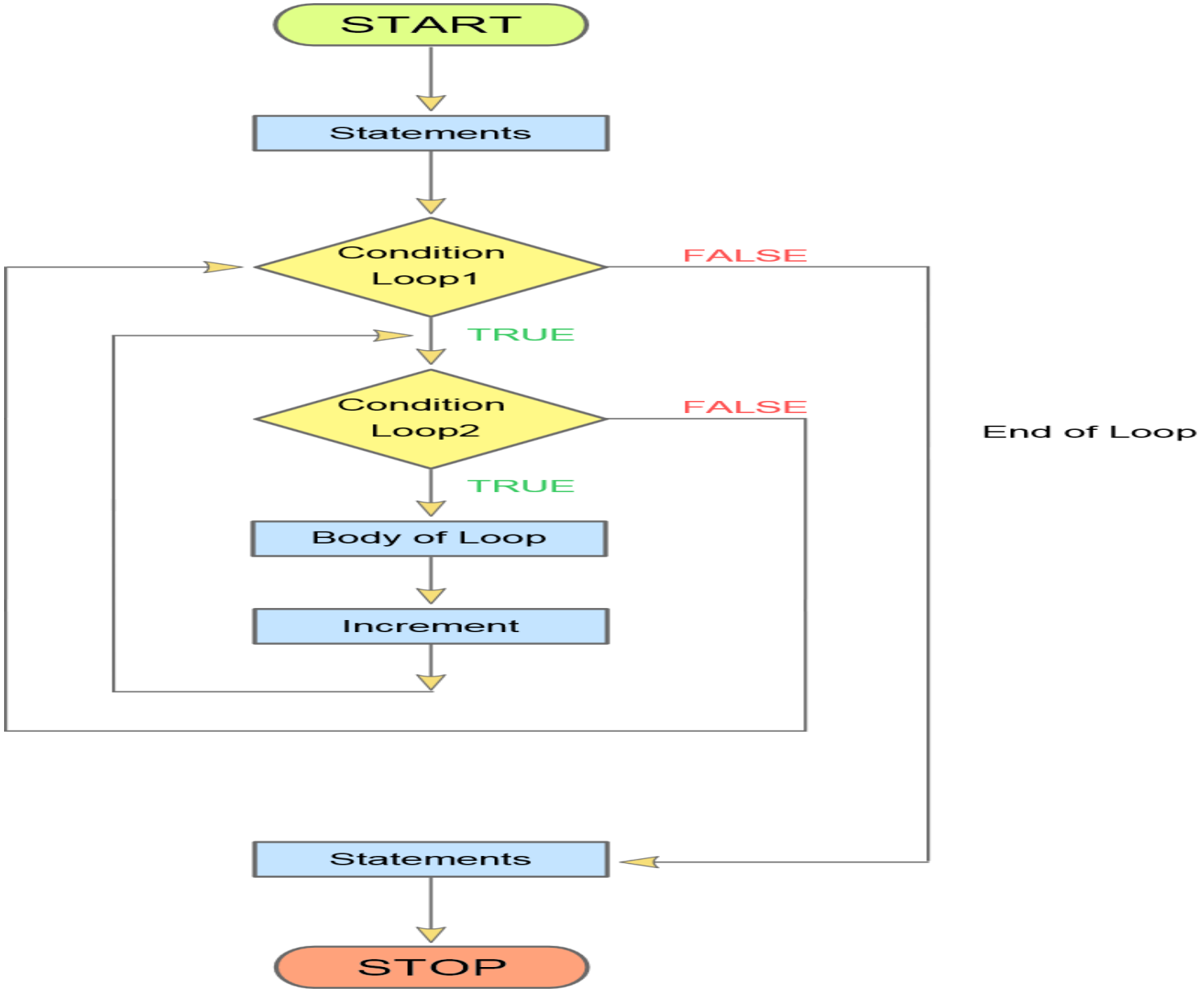
Nested for loop

- 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.
- 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

Syntax:

```
for ( initialization; condition; increment )
{
        for ( initialization; condition; increment )
        {
                statement of inside loop
        }
        statement of outer loop
}
```

Flow Chart:


START
Statements
Condition
Loop1
FALSE
TRUE
Condition
Loop2
FALSE
TRUE
Body of Loop
Increment
End of Loop
Statements
STOP

Example:

Write a C program to get the triangle of numbers as a result

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

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

Example:

Write a C program to get the triangle of numbers as a result

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

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

unconditional control statements

- Unconditional Control Transfer Statements is used to transfers the control to some other place in the program.
- In C Programming Language, There are four types of unconditional control transfer statements.

(i) break (ii)continue (iii) goto (iv)return

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;

Example:

```
#include<stdio.h>
void main()
{
        int i=0;
        while(i<=5)
        {
                i++;
                if(i==3)
                        break;
                printf("%d\t",i);
        }
}
```

OUTPUT:

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.
- 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;

Example:

```
#include<stdio.h>
void main()
{
        int i=0;
        while(i<5)
        {
                i++;
                if(i==3)
                        continue;
                printf(“ %d\t”,i);
        }
}
```

OUTPUT:

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.
- 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.

- **Syntax :**

goto label;
.....
.....
.....
label:
statement;

Program without using goto

```
#include<stdio.h>
void main()
{
printf("MITE \t");
printf("is \t in\t");
printf("Moodbidri\n");
}
```

OUTPUT

MITE is in Moodbidri

Program using goto

```
#include<stdio.h>
void main()
{
printf("MITE \t");
goto label1;
printf("is \t in\t");
label1: printf("Moodbidri\n");
}
```

OUTPUT

MITE Moodbidri

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;