MODULE 2

OPERATORS, DECISION CONTROL AND LOOPING STATEMENTS IN C

OPERATORS IN C

An operator is a symbol which acts on operands to produce certain result as output.

- For example in the expression a+b, + is an operator. a and b are operands.
- The operators are fundamental to any mathematical computations.

OPERATOR: An operator is a symbol (or a token) that specifies the operation to be performed on various types of operands. Operators connect operands i.e constants and variables to form expressions. **Example :** + (addition) , - (subtraction) , * (multiplication) ,etc

Classification based on the number of operands

The operators are classified into three major categories based on the number of operands as shown below:

1. Unary operator: acts on a single operand. For example: unary minus(-5, - 20, etc), address operator (&a)

2. Binary operator: acts on two operands. Ex: +, -, %, /, *, etc

3. Ternary operator: acts on three operands. The symbol ?: is called ternary operator in C language. Example: big= a>b?a:b; i.e if a>b, then big=a else big=b.

Classification Based On The Type Of The Operation

The operators are classified into following eight types based on the operation they perform on operands :

1. Arithmetic operators

2. Assignment operators

3. Relational operators

4. Logical operators

5. Bit wise operators

6. Conditional operators (ternary operators)

7. Increment/decrement operators

8. Special operators

1. **Arithmetic Operators**: Used to perform arithmetic operations.

S.no	Arithmetic Operators	Operation	Example
1	+	Addition	A+B, 4+2,6+5,-2+2 etc
2	-	Subtraction	A-B, 4-2, 2-4, 5-3 etc
3	*	multiplication	A*B, 4*2, 2*4, 3*5 etc
4	/	Division	A/B , 1/5, 3/5, 34/-5 etc
5	%	Modulus	A%B , 1%5, 5%2 etc

2. Assignment Operators

- The = operator is called as the assignment operator.
- It is used to assign a constant or expression or another variable to the variable.

Syntax : variable = constant ; OR variable = expression; OR variable = another_variable;

Examples for Assignment Statements x = 10; x += 10; y=x; sum = a+b;

3. Relational Operators

- Used to compare two quantities in a relational expression.
- The result of comparison is either "true" or "false".

S.no	Operators	Example	Description
1	>	x > y	x is greater than y
2	<	x < y	x is less than y
3	>=	x >= y	x is greater than or equal to y
4	<=	x <= y	x is less than or equal to y
5	==	x == y	x is equal to y
6	!=	x != y	x is not equal to y

4. Logical Operators

- Used to connect two relational expressions.
- The result of logical expression is also either "true" or "false".

The operation of Logical operators is shown in the following tables:

Operator	Meaning	Example
!	Logical NOT	!(a<0)
&&	Logical AND	(a>=0) && (a<=100)
||	Logical OR	(a<0) || (a>100)

Logical AND Operation

a	b	a&&b
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR Operation

a	b	A||b
0	0	0
0	1	1
1	0	1
1	1	1

Logical NOT Operation

a	!a
0	1
1	0

5. Bit Wise Operators

- Bit wise operators perform operation on individual bits.
- The operation of Bitwise AND, OR is similar to the tables of Logical AND and OR operators.
- The operation of Bitwise complement is same as Logical NOT.

Bitwise Operators	Symbols Used	Usage
Bitwise AND	&	a &b
BITWISE OR	|	a|b
BITWISE XOR	^	a^b
Left Shift Operator	<<	a<<2
Right Shift Operator	>>	a>>2
One's Complement	~	~ a

Logical XOR Operatic n

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

- **Left Shift operator (<<)** is used to shift the bits by specified number of positions to the left

For example, if we have x = 0001 1101, then x << 1 produces 0011 1010

- **Right Shift operator (>>)** is used to shift the bits by specified number of positions to the right

For example, if we have x = 0001 1101, then x >> 1 produces = 0000 1110 Similarly, if we have x = 0001 1101, then x >> 4 produces 0000 0001.

- **Ones complement operator (~)** is used to change the bits from 0 to 1 and from 1 to 0.

6. Conditional operators (ternary operators)

- Conditional operator which is also known as ternary operator, operates on three operands.
- Conditional operators return one value if condition is true and return another value if condition is false.

Syntax : (exp1)? (exp2):(exp3);

Example : (a> 100) ? 0 : 1;

In above example, if a is greater than 100, 0 is returned else 1 is returned. This is equivalent to if else conditional statement.

7. Increment/decrement operators

- C provides two unusual and powerful operators for incrementing and decrementing variables known as increment and decrement operators.
- The operator ++ adds one to its operand and the operator -- subtracts one from its operand. The increment and decrement operators may be post increment and pre increment or post decrement and pre decrement operators.

The syntax and description of the operators is illustrated in the table below:

Operator	Syntax	Equivalent to	Description

Pre-Increment Operator	**++variable;**	Variable=variable+1
		Variable+=1
			Value of i is incremented before assigning it to variable i.

Post-increment Operator	**variable++;**	Variable=variable+1
		Variable+=1
			Value of i is incremented after assigning it to variable i.

Pre-decrement Operator	**--variable;**	Variable= variable-1
		Variable-=1
			Value of i is decremented before assigning it to variable i.

Post-decrement Operator	**variable--;**	Variable=variable-1
		Variable-=1
			Value of i is decremented after assigning it to variable i.

8. Special operators

C supports some special operators of interest such as comma operator, sizeof( ) operator, pointer operators (& and *) and member selection operators (. and ->).

- **The Comma Operator:** The comma operator can be used to link related expressions together. It is used to combine many different statements into a single statement. A comma-linked list of expressions is evaluated left to right and value of right most expression is the value of the combined expression.

The multiple declarative statements can be written in single statement as illustrated in the examples below:

Example:

```
int a
int b        =   int a, b, c, d;
int c
int d
```

The sizeof( ) Operator :The sizeof( ) operator is an operator which gives the number of bytes allocated for a variable or a data type. Its general form is sizeof (data type/variable);

Example : int x = sizeof(x) ; results the value of x = 2 Bytes.

MODES OF EXPRESSION:

There are three different modes of expressions.

1. Integer Arithmetic Expression
2. Real Arithmetic Expression
3. Mixed-mode Arithmetic Expression

1. Integer Arithmetic Expression

- When both the operands such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic.
- This mode of expression always gives an integer value as result.
- Example: If a and b are integers i.e. a=14 and b=4, We have the following results:
  a - b=10   a + b = 18   a * b = 56   a/ b = 3   a% b = 2

2. Real Arithmetic Expression

- An arithmetic operation involving only real operands is called real arithmetic.
- A real operand may assume values either in decimal or exponential notation.
- If x, y, and z are floats, then we will have: x =6.0/7.0 = 0.857143
  y= 1.0/3.0 =0.333333   z= -2.0/3.0= - 0.666667.
  The operator % cannot be used with real operands.

3. Mixed- mode Arithmetic

When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression. Thus, 15 / 10.0 = 1.5 whereas, 15 / 10 = 1

Arithmetic operator's precedence and Associativity

Precedence of Operators.

- In a program, the value of any expression is calculated by executing one arithmetic operation at a time.
- The order in which the arithmetic operations are executed in an expression is based on the rules of precedence of operators.
- The precedence of operators is :

Unary (-)	FIRST
Multiplication(*), Division (/) and (%)	SECOND
Addition (+) and Subtraction (-)	LAST

For example, in the integer expression –a*b/c+d the unary - is done first, the result –a is multiplied by b, the product is divided by c (integer division) and d is added to it.

The answer is thus: -ab/c+d

- All the expressions are evaluated from left to right.
- All the unary negations are done first.
- After completing this, the expression is scanned from left to right;
- now all *, / and % operations are executed in the order of their appearance.
- Finally all the additions and subtractions are done starting from the left of the expression.

For example, in the expression: z = a + b* c

Initially b*c is evaluated and then the resultant is added with a. Suppose, if want to add a with b first, then it should be enclosed within parentheses, as shown below z = (a + b) * c

Use of parentheses:

- Parentheses are used if the order of operations governed by the precedence rules are to be overridden.
- In the expression with a single pair of parentheses, the expression inside the parentheses is evaluated FIRST.
- Within the parentheses, the evaluation is governed by the precedence rules.

  For example, in the expression: a*b/(c+d*k/m+k)+a the expression within the parentheses is evaluated first giving:c+dk/m+k

- After this, the expression is evaluated from left to right using again the rules of precedence giving ab/c+dk/m+k +a
- If an expression has many pairs of parentheses, then the expression in the innermost pair is evaluated first, then, the next innermost and so on till all parentheses are removed.
- After this, the operator precedence rules are used in evaluating the rest of the expression.

Example: In the expression,

((x * y)+z/(n*p+j)+x)/y+z xy, np+j will be evaluated first.

In the next scan,

xy+z/np+j +x will be evaluated.

In the final scan the expression evaluated would be: (xy+ z/np+j+x)/y +z

Associativity of operators.

Whenever two or more operators in the expression have the same precedence, then the direction (***Left to Right*** or ***Right to Left***) chosen to evaluate the expression is called Associativity.

Two Types:

1. **Left to Right**: Expression is evaluated from left to right
   EX: Arithmetic Operators, Logical Operators and Relational Operators follow L to Rassociativity.
2. **Right to Left**: Expression is evaluated from right to left
   EX: Assignment Operators, Unary Operators and Conditional Operator follow R to Lassociativity.

Consider an expression **y=12/4*5**

Here, since both / and * have same precedence. They are evaluated based on ***Left toRight*** Associativity.

C Operators Precedence Table

Operator	Description	Precedence or Priority	Associativity
( ) [ ]	Parentheses (function call) Brackets (array subscript)	1	left-to-right
+	Unary Plus	2	right-to-left
-	Unary Minus		
++	Increment		
--	Decrement		
!	Logical Negation		
&	Address		
sizeof	Sizeof an object		
(type)	Type Cast (conversion)		
* / %	Multiplication/division/modulus	3	left-to-right
+ -	Addition/subtraction	4	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	5	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	6	left-to-right
== !=	Relational is equal to/is not equal to	7	left-to-right
&	Bitwise AND	8	left-to-right
^	Bitwise exclusive OR	9	left-to-right
|	Bitwise inclusive OR	10	left-to-right
&&	Logical AND	11	left-to-right
| |	Logical OR	12	left-to-right
? :	Ternary conditional	13	right-to-left
=+= -=*= /= %= &=^= |= <<= >>=	Assignment Operators	14	right-to-left
,	Comma (separate expressions)	15	left-to-right

TYPE CONVERSION AND TYPE CASTING

Type conversion: The process of converting data from one data type to another data type is called typeconversion. It is possible to convert integer data to floating point data or vice versa.

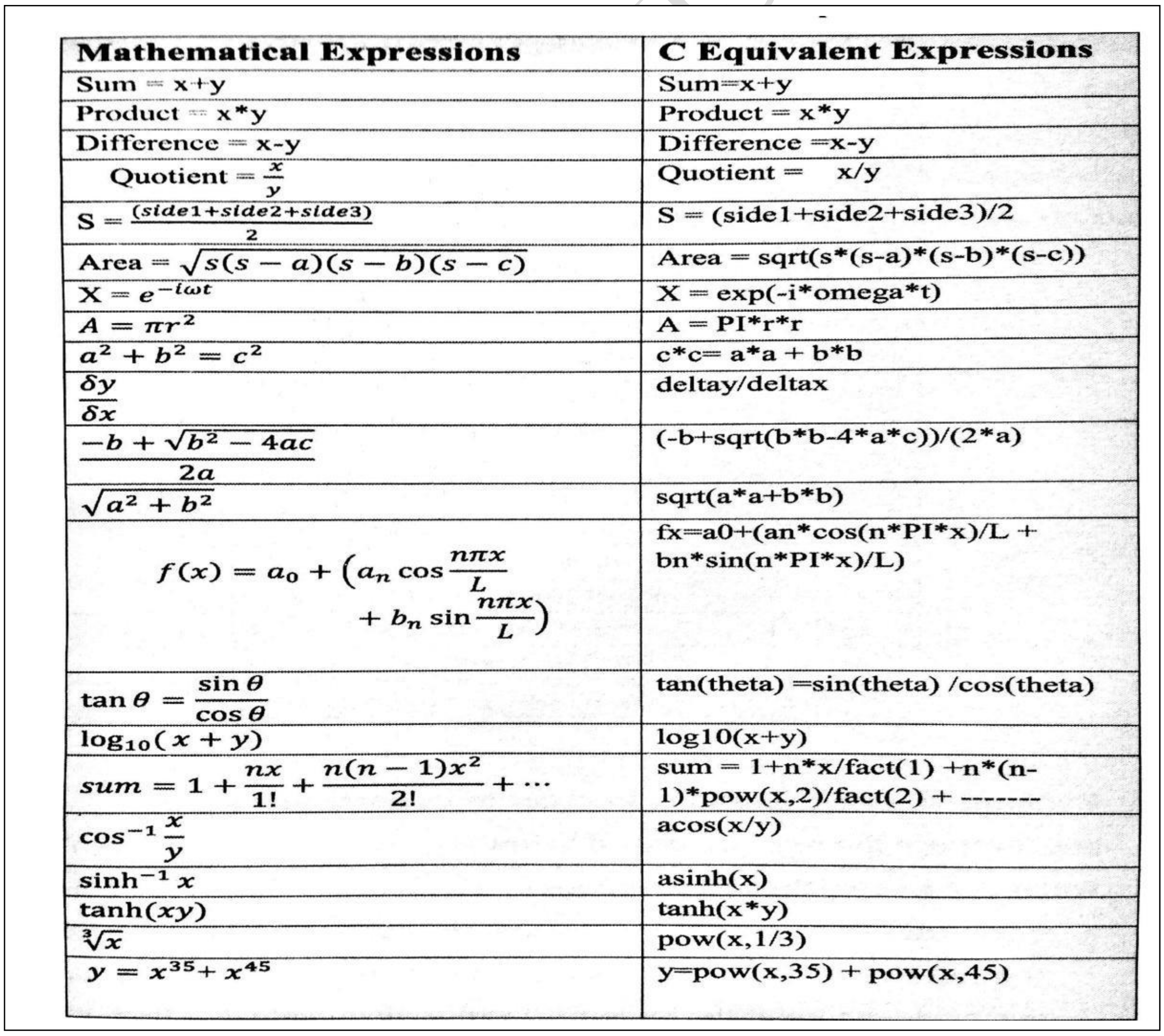
1. Implicit type conversion: the process of converting lower data type to higher data type automatically by the compiler is called implicit type conversion.
   Ex: float x; If x=3; The value of x in memory will be 3.000000.

2. Explicit type conversion: Forcibly conversion from one data type to another data type is called explicit type conversion.
   Ex: int p=7, q=2; r = p / q; 7/2=3.000000;
   Suppose if we write, r = (float)p / q ; then it will be (float)7 / 2 = 3.500000

Conversion of Mathematical Expressions into C equivalent expression

Mathematical Expressions	C Equivalent
ExpressionsSum = x+y	Sum = x + y;
Product = x*y	Product = x * y;
Difference = x-y	Difference = x - y;

The following table shows the conversion of mathematical expressions into C equivalents.



Mathematical Expressions	C Equivalent Expressions
Sum = x+y	Sum=x+y
Product = x*y	Product = x*y
Difference = x-y	Difference =x-y
Quotient = $\frac{x}{y}$	Quotient = x/y
$S = \frac{(side1+side2+side3)}{2}$	S = (side1+side2+side3)/2
Area = $\sqrt{s(s-a)(s-b)(s-c)}$	Area = sqrt(s*(s-a)*(s-b)*(s-c))
$X = e^{-i\omega t}$	X = exp(-i*omega*t)
$A = \pi r^2$	A = PI*r*r
$a^2 + b^2 = c^2$	c*c= a*a + b*b
$\frac{\delta y}{\delta x}$	deltay/deltax
$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$	(-b+sqrt(b*b-4*a*c))/(2*a)
$\sqrt{a^2 + b^2}$	sqrt(a*a+b*b)
$f(x) = a_0 + \left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right)$	fx=a0+(an*cos(n*PI*x)/L + bn*sin(n*PI*x)/L)
$\tan\theta = \frac{\sin\theta}{\cos\theta}$	tan(theta) =sin(theta) /cos(theta)
$\log_{10}(x+y)$	log10(x+y)
$sum = 1 + \frac{nx}{1!} + \frac{n(n-1)x^2}{2!} + \cdots$	sum = 1+n*x/fact(1) +n*(n-1)*pow(x,2)/fact(2) + -------
$\cos^{-1}\frac{x}{y}$	acos(x/y)
$\sinh^{-1} x$	asinh(x)
$\tanh(xy)$	tanh(x*y)
$\sqrt[3]{x}$	pow(x,1/3)
$y = x^{35} + x^{45}$	y=pow(x,35) + pow(x,45)

DECISION CONTROL AND LOOPING STATEMENTS:

BRANCHING AND LOOPING: Two way branching (if, if-else, nested if- else), Multi-way Branching (cascaded if-else, switch statement), ternary operator(?:), goto,Loops (while, do-while, for) in C, break and continue statements.

BRANCHING AND LOOPING

- Many a time, we want a set of instructions to be executed in one situation and anentirely different set of instructions in another situation.
- This kind of situation is dealt with, in C programs, using a decision controlinstruction.
- There are two types of decision making statements or branching statements in C. They are

1) **Conditional Branching statements:**

Here, the control is transferred to a remote statement based on somecondition which is a relational expression whose value is true or false.

a) Two way branching statements

- Simple if statement
- if-else statement
- Nested if statement

b) Multiway branching statements

- else –if ladder also known as cascaded if
- switch statement

2) **Unconditional Branching statements**

Here the control is transferred without any condition.

- goto statement
- break statement
- continue statement
- return statement

1. SIMPLE IF STATEMENT:

- The "if" statement is a powerful decision making statement and is used to control the flow of execution of statements. OR This is used to execute a statement or a set of statements conditionally.
- The syntax and flowchart of if statement are shown in figure 2.1 below.
- If the condition is true, the statements S1, S2, S3….. are executed and as usual the next statement in sequence i.e. Statement X is executed.
- If condition is false, the statements S1, S2, S3…. are skipped and as usual the next statement in sequence i.e. Statement X is executed.

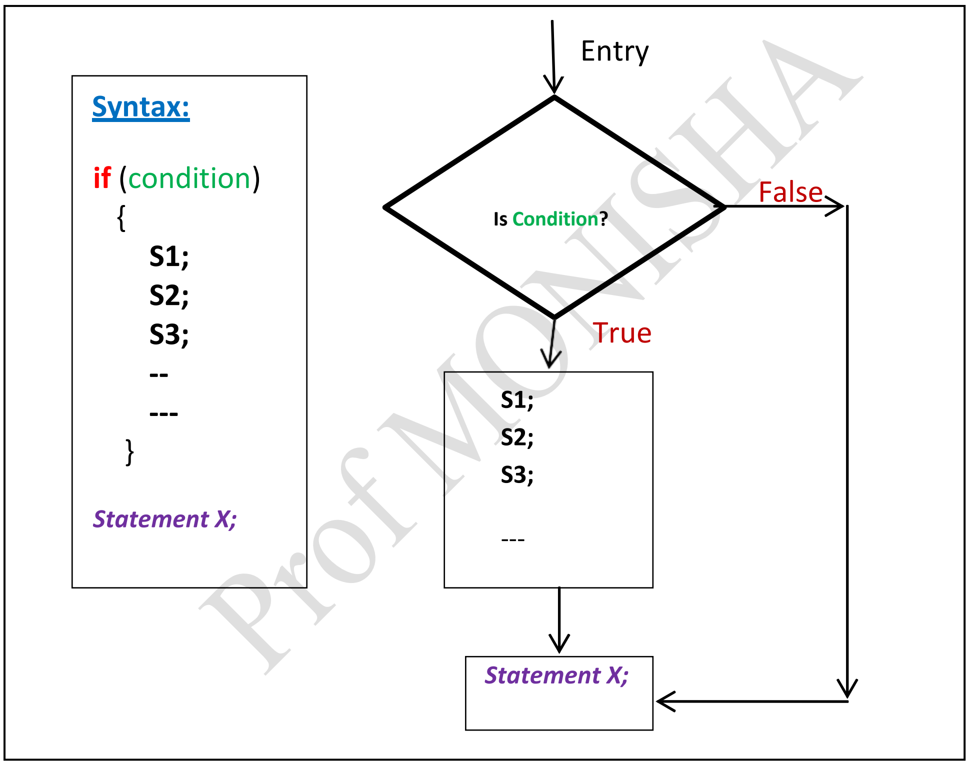


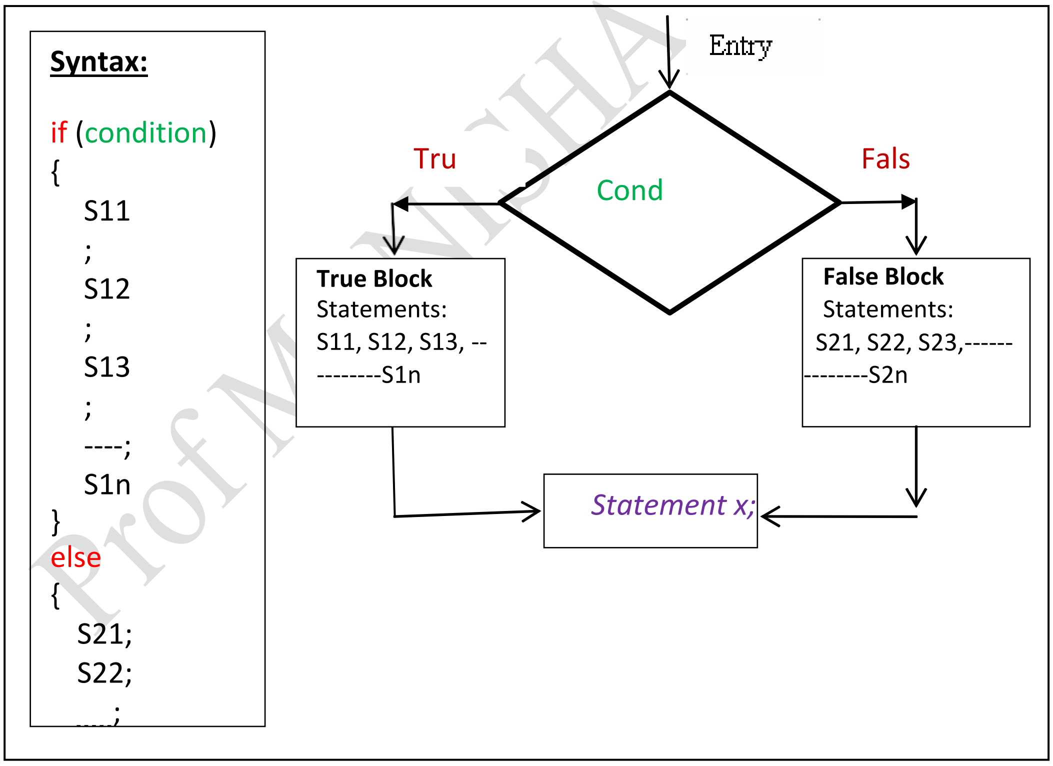
Fig 2.1: Syntax and Flowchart of if statement

Example1: C program to illustrate the use of if statement.

```
#include<stdio.h>void main( )
{
        int x, y, large;clrscr( );
        printf("\n Enter the value of x and y: \n");scanf("%d %d", &x, &y);
        large = x; if (y>large)
           large=y;
        printf("Large =%d\n", large);getch( );
}
```

2. if-else STATEMENT

- The if-else statement is used to execute only one the two statements to be executedalternatively.
- The syntax and flow diagram of if-else is shown in the figure 2.2 below.
- As illustrated in the figure, if the condition is true, the set of statements {S_{11}, S_{12},------S_{1n}} gets executed and statements{S_{21},S_{22},S_{23}------S_{2n}} are skipped and next Statement X is executed.
- On the other hand, if the condition is false, the set of statements {S_{11}, S_{12}, ---------------------S_{1n}} are skipped and the statements {S_{21}, S_{22}, S_{23}------S_{2n}} gets executed and then Statement X is executed.



Syntax:

```
if (condition)
{
    S11
    ;
    S12
    ;
    S13
    ;
    ----;
    S1n
}
else
{
   S21;
   S22;
   .....;
```

Fig : Syntax and Flowchart of if-else

Example: C program to illustrate the use of if-else statement.

```
#include<stdio.h>void main( )
{
int num;clrscr( );
printf("\n Enter the number: \n");scanf("%d", &num);
if (num%2==0)
printf("The number is Even\n");else
printf("\nThe number is odd\n");getch( );
}
```

3. NESTED "if–else" STATEMENT

- Using of one if-else statement in another if-else statement is called as nested if- else statement.
- The syntax and flow diagram of nested if-else is shown in the figure 2.3 below.
- When a series of decisions are involved, we may have to use more than one if- else statements in nested form.
- If Test Condition1 is true then control enters into outer if block, and it checks Test Condition2. If it is true, then Statement- S1 is executed; if it is false, then else block i.e Statement- S2 is executed.
    - If Test Condition1 is false, then it skips the outer if block and it goes to else block and Test Condition 3 is checked. If it it is true, Statement-S3 executed and if it is false, then statement-S4 is executed.
- Finally, as usual STATEMENT x in sequence is executed.

Syntax:

```
if(cond1)
{
   if(cond2)
   {
     s1;
    }
   else
   {
    s2;
   }
}
else
{
 if(cond3)
  {
    s3;
  }
else
 {
    s4;
   }
}
Statement x;
```

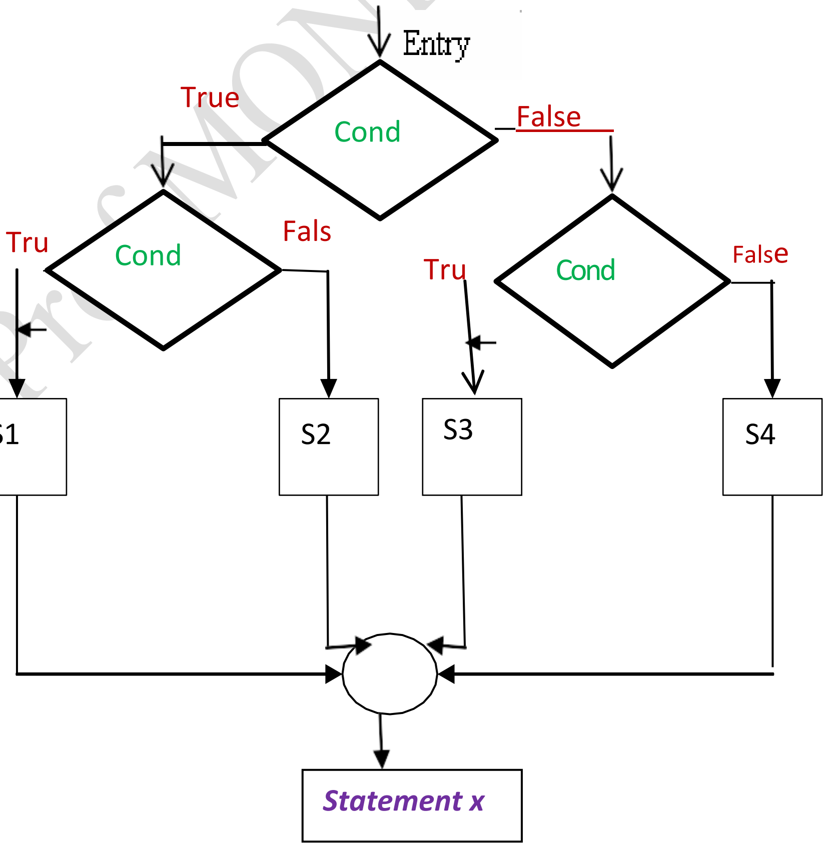


Fig : Syntax and Flowchart of nested if-else statement

Example: C program to illustrate the use of nested if-else statement

```
#include<stdio.h>
void main( )
{
            int  a, b, c ;
            clrscr( );
            printf("\n Enter the values of a, b and c: \n");scanf("%d
            %d %d", &a, &b, &c);
             if (a>b)
                    if (a>c)
                           printf("%d is big \n", a);else
                            printf("%d is big \n", c);
              else
                     if (b>c)
                            printf("%d is big \n", b);else
                                   printf("%d is big \n", c);
       } /*end of main*/
```

4. else-if LADDER STATEMENT (CASCADED IF)

- The cascaded if (or else if ladder) statement is a multi-way branching statement.
- The syntax and flow chart is shown figure below.

Syntax

```
if(c1)
   S1;
else if (c2)
   S2;
else if (c3)
   S3;
else if (c4);
   S4;
-------
else if (Cn)
   Sn;
else
default statement;

Statement x;
```

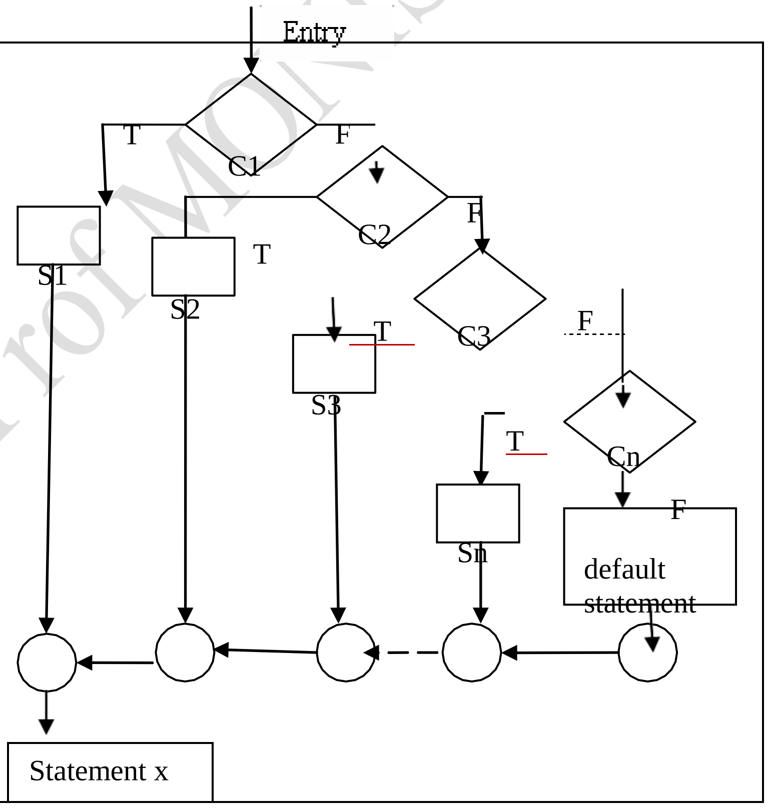


Fig : Syntax and Flowchart of else if ladder statement

- If condition 1 is true, statement s1 is executed else if condition 2 is true, statement S2 is executed and so on if condition n is true, statement Sn is executed. If none of the conditions is true, then default statement is executed. Finally, as usual, Statement x is executed next.

Example 1: C program to illustrate the use of else if ladder statement

```
#include<stdio.h>
void main( )
{
        int  a,b,c ;
        clrscr();
        printf("\n Enter the values of a, b and c\n");scanf("%d
        %d %d", &a, &b, &c);
         if (a>b && a>c)
             printf("%d is big \n", a);else if
             (b>a && b>c )
                  printf("%d is big \n", b);else if
                (c>a && c>b)
                         printf("%d is big \n", c);
 }/*end of main*/
```

Example2:Program to check the grade based on the marks as follows.

Marks	Grades
0 to 39	F (FAIL)
40 to 49	E
50 to 59	D
60 to 69	C
70 to 79	B
80 to 89	A
90 to 100	O(OUT STANDING)

```
/* Program to display Grade*/
#include<stdio.h>
void main( )
{
   int marks;


   clrscr( );
   printf("Enter the Marks\n");
   scanf("%d", &marks);
   if (marks<=39) printf("Grade
        = F");else if (marks<=49)
              printf("Grade = E");
```

```
                    else if (marks<=59)
                                printf("Grade = D");else
                         if (marks<=69)
                                      printf("Grade = C");else
                               if (marks<=79)
                                         printf("Grade = B");else
                                    if (marks<=89)
                                                  printf("Grade = A");
                                          else
                                                  printf("Grade = OUT STANDING");
      }/*end of main*/
```

Dangling else Problem:

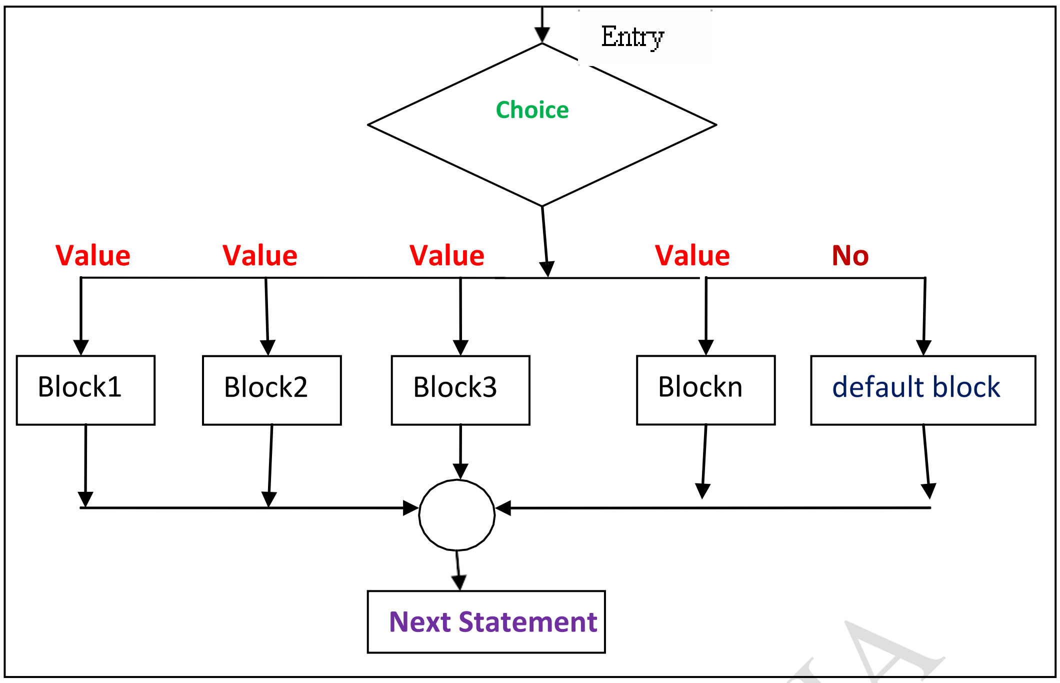
- One of the classic problems encountered when we start using nested if … else statements is the dangling else.
- This occurs when a matching else is not available for an if.
- The answer to this problem is very simple. Always match an else to the mostunmatched if in the current block.
- In some cases, it is possible that the false condition is not required. In suchsituations, else statement may be omitted.
- "else is always paired with the most recent paired if".

5. switch STATEMENT

- The switch statement is a multi-way decision that tests whether an expressionmatches one of a number of constant integer values and branches accordingly.
- It is a multi-way decision making control statement used to make a selectionbetween many alternatives.
- It is also known as switch case break and default statement.
- The syntax and flow diagram of switch statement are illustrated below:

Syntax :

```
switch (choice/expression)
    {
                  case value1 :  block1;
                                 break;
                  case value2 :  block2;
                                 break;
                  case value3 :  block3;
                                 break ;
            - - - - - - - - - - - - -  -
            case valuen :  blockn;
                               break;
            default          : default_block  ;
    }
```



- Here, switch and case are the key words.
- Choice/expression is either integer or character variable or expression.
- Value1, value2 etc are the labels.
- The choice/expression value is compared with the case values and whichevercase value matches, the corresponding block of statements is executed.
- The break statement of the block causes the control to come out of switchstatement.
- If the choice/expression value does not match with any of the case values, thenthe default block of statements will be executed.
- The case values need not be in order.
- If the case values are characters, then they should be enclosed in single quotes.
- Here is the sample program with a switch statement:

```
/* Program to simulate simple calculator*/#include
<stdio.h>
#include<process.h>

void main( )
{
     int a, b, choice;
    float result; clrscr(
    );
     printf(“Enter values of a and b:\n”);
     scanf(“%d %d”, &a, &b);
```

```
        printf(“\n 1: addition\n 2: subtraction\n  3: Multiplication\n
                    4: division\n”);
        printf(“Enter your choice: “);
        scanf(“%d”, &choice);

        switch (choice)
              {
                    case 1 : result=a+b;
                                    break;

                    case 2 : result=a-b;
                                   break;

                    case 3 : result=a*b;
                                   break;

                    case 4 : if (b!=0)
                                            result=a/b;
                                   else
                                   {
                                         printf(“Divide by zero Error!\n”);exit(0);

                                    }
                                    break;

                    default : printf(“ Illegal Operator !\n”);exit(0);
            } /* End of switch */

          printf(“result = %f’, result);
} /* End of main */
```

UNCONDITIONAL BRANCHING STATEMENTS

goto statement

goto statement transfers control to the statement specified by a label.

Syantax: - goto label;

Example:

```
        ………
        if (a>10)
          goto BOTTOM;else
          printf("/n%d", a);
     a=a+2;
BOTTOM: printf("The number is %d", a);
```

- Here, if value of a is greater than 10, the statement labeled BOTTOM (i.e printf ("The number is %d",a);) is executed otherwise statement printf("/n%d",a); is executed.

break statement

- It causes control to come out of loop or switch statement.

Syantax: - break;

Example:

```
………
for(i=1; i<10;i++)
{
        if (i%5==0)
              break; printf("
                     %d", i);
  }
```

```
OUTPUT: 1  2  3  4
```

- Here, for loop is executed and it goes on printing the value of i as long as it is notequal to 5.
- Once i value becomes 5, the break statement causes the control to come out ofloop.

continue statement

- Program control stops current iteration of loop and continues with next iteration.
- Used in loops but not used in switch statement like break statement.

Syntax : continue;

Example :

```
 for(i=1; i<10;i++)
{
   if (i%5==0)
         continue;
   printf("      %d", i);
 }
```

```
OUTPUT: 1  2  3  4      6  7  8  9
```

- The continue statement causes value of i=5 not to display (i.e stops that iteration after printing 1,2,3,and 4 and continues with the next iteration thus by printing 6, 7, 8 and 9).

return statement

- Used to return from the function (sub program).
- It is the last statement in the function.

Syantax: return(computed-value);

Example:

```
/* Program to add two numbers using
function */
#include<stdio.h>
void main( )
{
  int a, b, result;
  clrscr( );
  printf(“Enter two numbers☺;
  scanf(“%d%d”, &a,&b);
  result=add(a,b);
  printf(“Sum=%d”,result);
}
```

```
/* Function to add two numbers*/
int add(int m, int n)
{
  int sum;

  sum = m + n;
  return(sum);
}
```

DIFFERENCE BETWEEN BREAK AND CONTINUE STATEMENT:

SNO	**break Statement**	**continue Statement**
1	When break statement is executed, the statements following break are skipped andcauses the loop to be terminated.	When continue statement is executed, the statements following continue are skipped and causes the loop to be continued with the next iteration.
2	It can be used in switch statement to transfer the control outside switch	It cannot be used inside a switch statement

3	for (i=1; i<=5; i++)
	{
	    if (i==3)
	        break ;
	    printf("%d\n", i);
	}
	
	output : 1 2
		for (i=1; i<=5; i++)
		{
		    if (i==3)
		        continue ; printf("
		    %d\n", i);
		}
		
		output : 1 2 4 5

LOOPS

LOOP: Same set of statements are repeatedly executed until some condition is true.

- This involves repeating some portion of the program either a specified number oftimes or until a particular condition is being satisfied.
- Types of Looping statements: Depending on the position of the control statementin the loop, the looping statements are classified into the following two types:
- Entry controlled Loop
- Exit Controlled Loop
- The loop in a program consists of two parts: 1) Body of the loop. 2) Condition.

TYPES OF LOOPS SUPPORTED IN C:

- **The C Language supports the following three types of loops.**
  1. **while loop**
  2. **do while loop**
  3. **for loop**
- Every loop must have three things.
  - ✓ Loop initialization – loop variable is first assigned with some value.
  - ✓ Loop condition – the Boolean condition which is either true or false
  - ✓ Loop updation - loop variable is incremented/decremented to makethe loop condition false (i.e. to come out of loop).

1. while LOOP:

- ➢ It is an entry controlled loop.
- ➢ It is also known as pre test or top testing loop
- ➢ If the condition is as long as true, the body of the loop is executed.
- ➢ Once the condition becomes false, then the control comes out of loop.
- ➢ If the condition is false, the body of the loop is not executed even once.

Syntax:

```
while (Condition)
{
        S1;
        S2;
        S3;
        ------
        Sn;
        updation;
}
```

Example:

```
sum=0;
i=1;
while(i<=10)
{
     sum=sum+i;
      i =i+1;
}
```

Here is a simple example of the use of the while loop to count from 1 to 100.

```
/* counting from 1 to 100 */
#include<stdio.h>
void main ( )
{
   int count = 1;                              //initialization
   while (count <= 100)                       // condition
   {
            printf("%d\n",count);
            count = count+1;                 //updation
    }
}
```

2. do while LOOP

- It is also an exit controlled loop.
- It is also known as post-test or bottom-testing loop.
- Since the condition is tested at the bottom of the loop, the statements within thebody of the loop will be executed at least once even if the condition is false.

Syntax: -

```
do
{
        S1;
        S2;
        S3;
        ------
        Sn;
        updation;
} while (Condition);
```

Example:

```
sum=0;
i =1;
do
{
     sum=sum+i;
     i =i+1;
}
while (i<=10);
```

Here is a simple example of the use of the do while loop to count from 1 to 100.

```
/* counting from 1 to 100 */
#include<stdio.h>
void main ( )
{
   int count = 1;                                   //initializationdo
   {
            printf("%d\n",count); count
            = count+1;
      //updation
    }
    while (count <= 100); }                          //condition
```

Example: Program to check whether the given number is palindrome or not

```
/* Program to check whether the given number is a palindrome or not */
#include<stdio.h>
void main( )
{

    int  num, rev=0, digit;

    printf("Enter a number :\n");
    scanf("%d", &num);
    temp = num;


    do
    {
              digit = num % 10; rev =
              rev * 10 + digit;num =
              num / 10;
     }
     while (num != 0);

    printf("The reverse of %d = %d", temp, rev);if

    (rev==temp)
              printf(" \nThe number %d is palindrome!", temp);
     else
              printf(" \nThe number %d is not a palindrome!", temp);
}
```

DIFFERENCE BETWEEN WHILE LOOP AND DO WHILE LOOP

While loop	**Do while loop**
It is a pre-test loop where the condition is checked before executing the body of the loop	It is a post-test loop where the condition is checked after executing the body ofthe loop.

Syntax:
while (Condition)
{
    S1;
    S2;
    S3;
    ------ 	
    Sn;
    updation;
}
	Syntax :
	do
	{
	    S1;
	    S2;
	    S3;
	    ------ 	
	    Sn;
	    updation;
	} while (Condition);

Initialization is done before the while loop, testing is done in the beginning of the while loop and updation is done in the body of the loop normally as the last statement.	Initialization is done before the do while loop, testing is done at the end of the while loop and updation is done in the body of the loop normally as the laststatement.

Example :
sum=0;
i=1;
while(i<=10)
{
  sum=sum+i;i
  =i+1;
}
	Example :
	sum =0;
	i=1;
	do
	{
	  sum=sum+i;i
	  =i+1;
	} while(i<=10);

If the condition is false , the body of the loop is not at all executed even once.	If the condition is false , the body of the loop is executed at least once.

Similarities between while and do-while loops

while loop	**do while loop**
SIMILARITIES	
Both the loops are used whenever we don't know how many times body of the loop has to be executed	
Both loops will have loop variable initialization before body of the loop., loop condition and loop variable updation	
Both loops will have the loop condition	
Both loops will have loop updation within body of the loop.,	
The body of the loops is a compound statement	

3. for LOOP

for loop is used when the number of times the loop body to be executed is knownwell in advance.

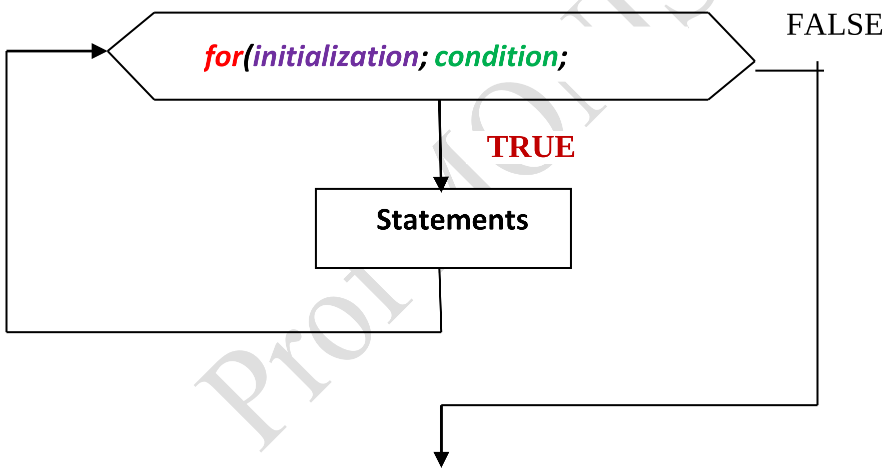
Syntax:

```
for (initialization; condition; updation)
{
        Body of loop;
}
```

Here

- initialization : The loop variable is initialized.
- condition : It is the test condition which is true or false. Body of the loop isexecuted as long as the condition is true. Once condition becomes false, the control comes out of loop.
- updation: is incrementation / decrementation to make the condition false to come out of loop.

Flow chart: -



Here is a simple example of the use of the for loop to count from 1 to 100.

```
/* Program to count from 1 to 100 */
#include<stdio.h>
void main ( )
{
   int count;
   for (count=1; count<=100; count++)
                      printf("%d\n", count);
}
```

4.Nested for loops

If one for loop is used within another for loop, it is called nested for loops.

Example:

```
for(i=0; i<n; i++)          /* Outer for loop */
{
        for(j=0; j<m; j++)          /* Inner for loop */
        {
                C[i][j]=a[i] + b[j];
        }
}
```

- ✓ Loop variable must be different for both loops.

Write a program to find the factorial of a given number.

```
/* Program to find the factorial of a given number*/
#include<stdio.h>
void main( )
{
        int n, fact=1, i;
        printf("Enter the number:\n");
        scanf("%d", &n);
        for(i=1;  i<n;  i++)
                fact=fact*i;
        printf("\nFactorial of %d = %d", n, fact);
}
```