Unit - 2
Operators & Expressions

Definition

"An operator is a symbol (+,-,*,/) that directs the computer to perform certain mathematical or logical manipulations and is usually used to manipulate data and variables"

Ex: a+b

Operators in C

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

Arithmetic operators

Operator	**example**	**Meaning**
+	a + b	Addition –unary
-	a – b	Subtraction- unary
*	a * b	Multiplication
/	a / b	Division
%	a % b	Modulo division-remainder

The modulo division produces the remainder of an integer division.

The modulo division operator cannot be used on floating point data.

Note: C does not have any operator for *exponentiation*.

Integer Arithmetic

When both the operands in a single arithmetic expression are integers, the expression is called an *integer expression* , and the operation is called *integer arithmetic.*

During modulo division the sign of the result is always the sign of the first operand.

That is

-14 % 3 = -2

-14 % -3 = -2

14 % -3 = 2

Real Arithmetic

An arithmetic operation involving only real operands is called *real arithmetic.* If **x and y** are floats then we will have:

1) x = 6.0 / 7.0 = 0.857143

2) y = 1.0 / 3.0 = 0.333333

The operator % cannot be used with real operands.

Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called *a mixed-mode arithmetic* expression and its result is always a real number.

Eg: 1) 15 / 10.0 = 1.5

Arithmetic operator

```
main()
{
   int a = 21;       int b = 10;       int c ;
    printf("Line 1 - Value of c is %d\n", c );      c = a + b;
   printf("Line 2 - Value of c is %d\n", c );      c = a - b;
   printf("Line 3 - Value of c is %d\n", c );      c = a * b;
   printf("Line 4 - Value of c is %d\n", c );      c = a / b;
   printf("Line 5 - Value of c is %d\n", c );      c = a % b;
   printf("Line 6 - Value of c is %d\n", c );      c=a++;
   printf("Line 7 - Value of c is %d\n", c );      c = a--;
 }
```

OUTPUT :

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 7 - Value of c is 22
```

Relational Operators

Comparisons can be done with the help of *relational operators*. The expression containing a relational operator is termed as a *relational expression*. The value of a relational expression is either *one* or *zero*.

Operator	**Meaning**
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Equal to
!=	Not equal to

Relational operator

```
main()
{
  int a = 21;      int b = 10;      int c ;
   if( a == b )
  {
printf("Line 1 - a is equal to b\n" );
  }
  else
  {
    printf("Line 1 - a is not equal to b\n" );
  }
if ( a < b )
```

```
   {
      printf("Line 2 - a is less than b\n" );
   }
   else
   {
      printf("Line 2 - a is not less than b\n" );
   }
   if ( a > b )
   {
      printf("Line 3 - a is greater than b\n" );
   }
   else
   {
      printf("Line 3 - a is not greater than b\n" );
   }
/* Lets change value of a and b */
   a = 5;
   b = 20;
if ( a <= b )
   {
      printf("Line 4 - a is either less than or euqal to  b\n" );
   }
   if ( b >= a )
   {
      printf("Line 5 - b is either greater than  or equal to b\n" );
   }
  }
```

OUTPUT :

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to  b
```

Line 5 - b is either greater than or equal to b

Logical Operators

C has the following three *logical operators.*

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

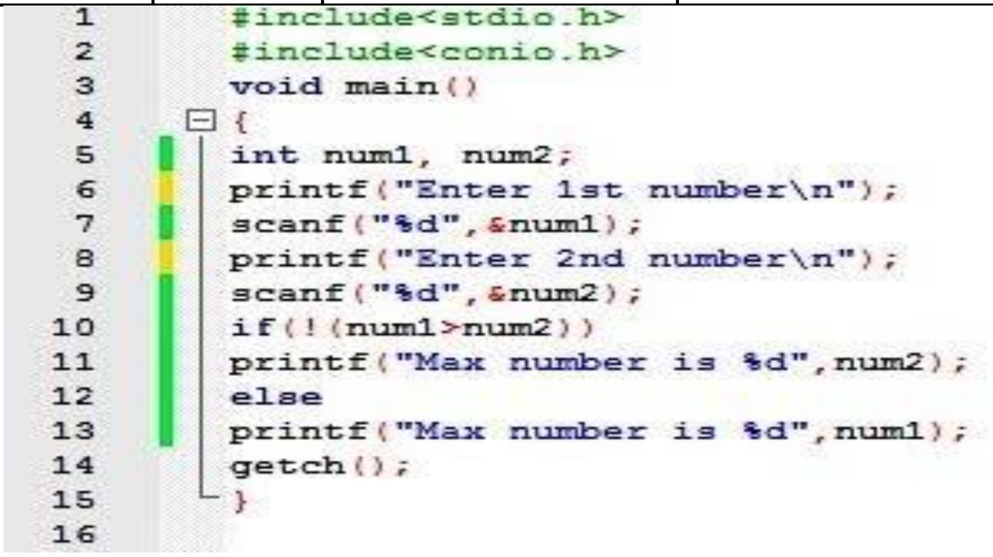
Logical expression or a compound relational expression-

An expression that combines two or more relational expressions

Ex: if (a==b && b==c)

Truth Table

A	**B**	**Value of the expression**	
		a && b	**a || b**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



```
#include<stdio.h>
#include<conio.h>
void main()
{
int num1, num2;
printf("Enter 1st number\n");
scanf("%d",&num1);
printf("Enter 2nd number\n");
scanf("%d",&num2);
if(!(num1>num2))
printf("Max number is %d",num2);
else
printf("Max number is %d",num1);
getch();
}
```

Assignment operators

The usual assignment operator is ‘=’.In addition, C has a set of‘shorthand’ assignment operators of the form, v **op** = exp;

Syntax:

v op = exp;

Where v = variable,

op = shorthand assignment operator

exp = expression

Ex: x=x+3

x+=3

Shorthand Assignment operators

Simple assignment operator	**Shorthand operator**
a = a+1	a + =1
a = a-1	a - =1
a = a* (m+n)	a * = m+n
a = a / (m+n)	a / = m+n
a = a %b	a %=b

Assignment Operators

```
main()
{
  int a = 21;       int c ;
  c =  a;
  printf("Line 1 - =  Operator Example, Value of c = %d\n", c );
 c +=  a;
printf("Line 2 - += Operator Example, Value of c = %d\n", c );
  c -=  a;
  printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
  c *=  a;
   printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
   c /=  a;
 printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
c  = 200;
 c %=  a;
printf("Line 6 - %= Operator Example, Value of c = %d\n", c
c <<=  2;
   printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );
   c >>=  2;
   printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );
   c &=  2;
   printf("Line 9 - &= Operator Example, Value of c = %d\n", c );
   c ^=  2;
   printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );
   c |=  2;
   printf("Line 11 - |= Operator Example, Value of c = %d\n", c );
}
```

OUTPUT :

```
Line 1 - =  Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

Increment & Decrement Operators

C supports 2 useful operators namely

1. Increment ++
2. Decrement – operators

The ++ operator adds a value 1 to the operand

The –operator subtracts 1 from the operand

++a or a++

--a or a--

Rules for ++ & -- operators

1. These require variables as their operands
2. When postfix either ++ or – is used with the variable in a given expression, the expression is evaluated first and then it is incremented or decremented by one
3. When prefix either ++ or – is used with the variable in a given expression, it is incremented or decremented by one first and then the expression is evaluated with the new value

Examples for ++ & -- operators

Let the value of a =5 and b=++a then

a = b =6

Let the value of a = 5 and b=a++ then

a =5 but b=6

i.e.:

1. a prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left

2. a postfix operator first assigns the value to the variable on left and then increments the operand.

Conditional operators

A ternary operator pair "?:" is available in C to construct conditional expression of the

form:

Syntax:

exp1 ? exp2 : exp3

Where exp1,exp2 and exp3 are expressions

Working of the ? Operator:

Exp1 is evaluated first, if it is nonzero(1/true) then the expression2 is evaluated and this becomes the value of the expression,

If exp1 is false(0/zero) exp3 is evaluated and its value becomes the value of the expression

Ex: m=2;

```
        n=3
        r=(m>n) ? m : n;
```

CONDITIONAL OPERATOR (? : Operator)

```
main()
{
  int a , b;
 a = 10;
b = (a == 1) ? 20: 30;
   printf( "Value of b is %d\n", b );
 b = (a == 10) ? 20: 30;
  printf( "Value of b is %d\n", b );
}
```

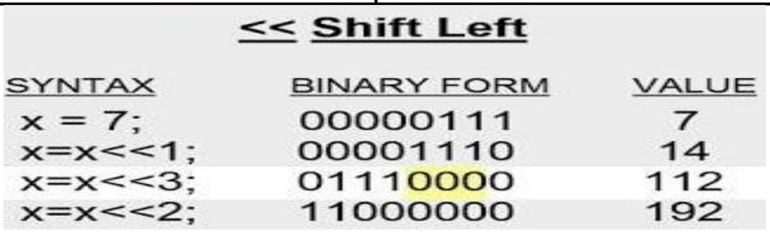
OUTPUT :

```
Value of b is 30
Value of b is 20
```

Bitwise operators

These operators allow manipulation of data at the bit level

Operator	**Meaning**
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right



<< Shift Left

SYNTAX	BINARY FORM	VALUE
x = 7;	00000111	7
x=x<<1;	00001110	14
x=x<<3;	01110000	112
x=x<<2;	11000000	192

Special operators

1. Comma operator (,)

2. sizeof operator – sizeof( )
3. Pointer operators – (& and *)
4. Member selection operators – (. and ->)

The Comma Operator

The comma operator can be used to link the related expressions together.

A comma-linked list of expressions are evaluated *left* to *right* and the value of *right-most* expression is the value of the combined expression.

Eg: value = (x = 10, y = 5, x + y);

This statement first assigns the value 10 to **x**, then assigns 5 to **y,** and finally assigns

15(i.e, 10+5) to **value**.

special operator (sizeof Operator)

- The size of is a compiler time operator and, when used with an operand, it returns the number of bytes the operand occupies.

```
main()
{
  int a;
  short b;
double double c;
  char d[10];
printf("Line 1 - Size of variable a = %d\n", sizeof(a) );   printf("Line 2 - Size of
variable b = %d\n", sizeof(b) );   printf("Line 3 - Size of variable c= %d\n", sizeof(c)
);   printf("Line 4 - Size of variable d= %d\n", sizeof(d) );
 /* For character string strlen should be used instead of sizeof*/
  printf("Line 5 - Size of variable d= %d\n", strlen(d) );
}
```

OUTPUT :

```
Line 1 - Size of variable a = 4
Line 2 - Size of variable b = 2
Line 3 - Size of variable c= 8
Line 4 - Size of variable d= 10
Line 5 - Size of variable d= 10
```

& and * Operators:

main()

```
{
int i=4;            /* variable declaration            */
int  *ptr;                  /* int pointer                 */
ptr = &i;                  /* 'ptr' now contains the address of 'i'*/ printf("  i  is  %d.\n", i);
printf("*ptr is %d.\n", *ptr);
}
```

OUTPUT :

```
 i is  4.*ptr is 4.
```

EVALUATION OF EXPRESSIONS

Expressions are evaluated using an assignment statement of the form

variable = expression;

Eg:1) x = a * b – c;

Arithmetic Expressions

Algebraic expression	**C expression**
axb-c	a*b-c
(m+n)(x+y)	(m+n)*(x+y)
	a*b/c
$3x^2$+2x+1	3*x*x+2*x+1
	a/b
	S=(a+b+c)/2
area=	area=sqrt(s*(s-a)*(s-b)*(s-c))
	sin(b/sqrt(a*a+b*b))
	tow1=sqrt((rowx-rowy)/2+tow*x*y*y)
	tow1=sqrt(pow((rowx-rowy)/2,2)+tow*x*y*y)
	y=(alpha+beta)/sin(theta*3.1416/180)+abs(x)

Precedence of operators

BODMAS RULE-

Brackets of Division Multiplication Addition SubtractionBrackets will have the highest precedence and have to be evaluated first, then comes of , then comes division, multiplication, addition and finally subtraction.C language uses some rules in evaluating the expressions

and they are called as precedence rules or sometimes also referred to as hierarchy of operations, with some operators with highest precedence and some with least.

The 2 distinct priority levels of arithmetic operators in c are-

Highest priority : * / %

Lowest priority : + -

Rules for evaluation of expression

1. First parenthesized sub expression from left to right are evaluated.
2. If parentheses are nested, the evaluation begins with the innermost sub expression
3. The precedence rule is applied in determining the order of application of operators in evaluating sub expressions
4. The associatively rule is applied when 2 or more operators of the same precedence level appear in a sub expression.
5. Arithmetic expressions are evaluated from left to right using the rules of precedence
6. When parentheses are used, the expressions within parentheses assume highest priority

Hierarchy of operators

Operator	**Description**	**Associativity**
( ), [ ]	Function call, array element reference	Left to Right
+, -, ++, - - ,!,~,*,&	Unary plus, minus, increment, decrement, logical negation, 1's complement, pointer reference, address	Right to Left
*, / , %	Multiplication, division, modulus	Left to Right

Example

Evaluate the expression when a=4

b=a- ++a

=a – 5

=5-5

=0

MATHEMATICAL FUNCTIONS

Mathematical functions such as sqrt, cos, log etc., are the most frequently used ones.

To use the mathematical functions in a C program, we should include the line

#include<math.h>in the beginning of the program.

Function	**Meaning**
Trignometric	
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan2(x,y)	Arc tangent of x/y
cos(x)	cosine of x
sin(x)	sine of x
tan(x)	tangent of x
Hyperbolic	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
Other functions	
ceil(x)	rounded up to the nearest integer
exp(x)	e to the power x
fabs(x)	absolute value of x
floor(x)	rounded down to the nearest integer
fmod(x,y)	remainder of x/y
log(x)	natural log of x, x>0
log10(x)	base 10 log of x.x>0
pow(x,y)	x to the power y
sqrt(x)	square root of x,x>=0

Decision Making and Branching

'C' language processes decision making capabilities supports the flowing statements known as control or decision making statements

1. If statement
2. Switch statement
3. Goto statement

If Statement : The if statement is powerful decision making statement and is used to control the flow of execution of statements The If statement may be complexity of conditions to be tested

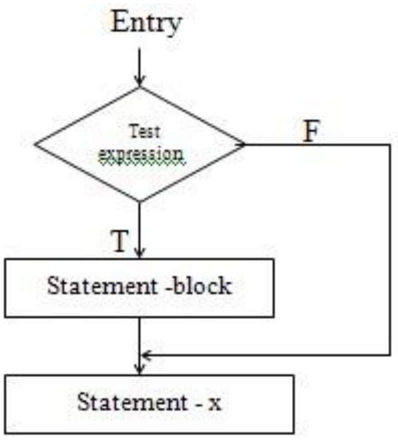
(a) Simple if statement
(b) If else statement
(c) Nested If-else statement
(d) Else –If ladder

Simple If Statement: The general form of simple if statement is

```
                    If(test expression)
      {    statement block;
                    }    statement-x ;
```

The statement -block may be a single statement or a group of statement if the test expression is true the statement block will be executed. Otherwise the statement -block will be skipped and the execution will jump to the statement –X. If the condition is true both the statement –block sequence .

Flow chart :



Ex :
```
int main()
{
 int m=40,n=40;
 if (m == n)
  {
 printf("m and n are equal");
  }
}
```
OUTPUT:

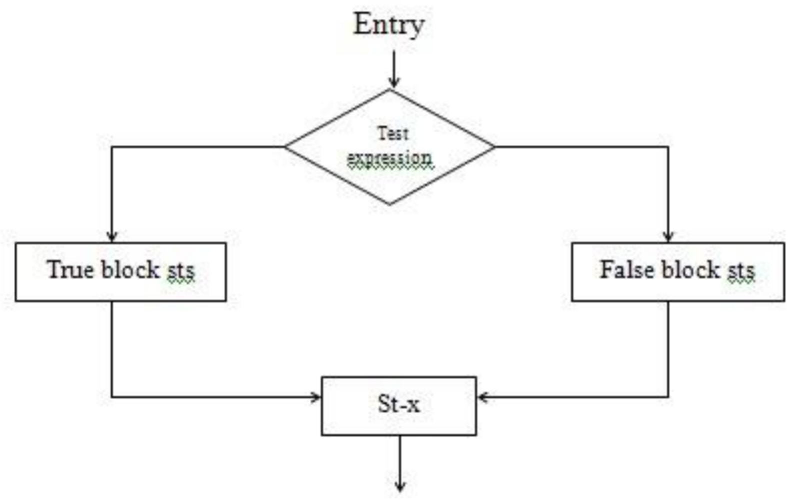
```
m and n are equal
```

If –Else Statement : The If statement is an extension of the simple If statement the general form is

```
            If (test expression)
            {
              true-block statements;
            }
      else
            {
             false-block statements;
            }
       statement – x;
```

If the test expression is true then block statement are executed, otherwise the false – block statement are executed. In both cases either true-block or false-block will be executed not both.

Flow chart :



Ex :

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m == n)
  {
  printf("m and n are equal");
  }
  else
  {
  printf("m and n are not equal");
  }
}
```

OUTPUT:

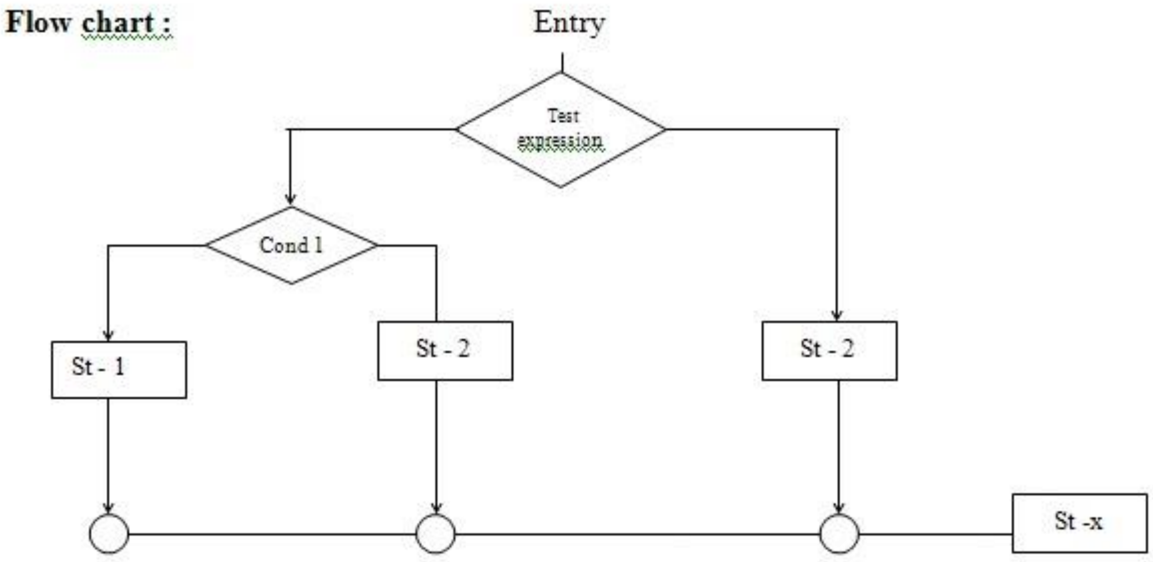
```
m and n are not equal
```

Nested If –else statement : When a series of decisions are involved we may have to use more than one if-else statement in nested form of follows .

```
            If(test expression)
      { if(test expression)
      {    st –1;
```

```
}
else
{   st – 2;
}else
{
   st – 3;
}
}st – x;
```

Flow chart :



If the condition is false the st-3 will be executed otherwise it continues to perform the nested If –else structure (inner part). If the condition 2 is true the st-1 will be executed otherwise the st-2 will be evaluated and then the control is transferred to the st-x

Some other forms of nesting If-else

```
If ( test condition1)
{  if (test condition2)
st –1 ;
} else
if (condition 3)
{ if (condition 4)
st – 2;
}st – x;
```

Ex:

```
 #include <stdio.h>
int main()
{
  int m=40,n=20;
        if (m>n) {
       printf("m is greater than n");
        }
        else if(m<n) {
        printf("m is less than n");
        }
        else {
        printf("m is equal to n");
        }
        }
```

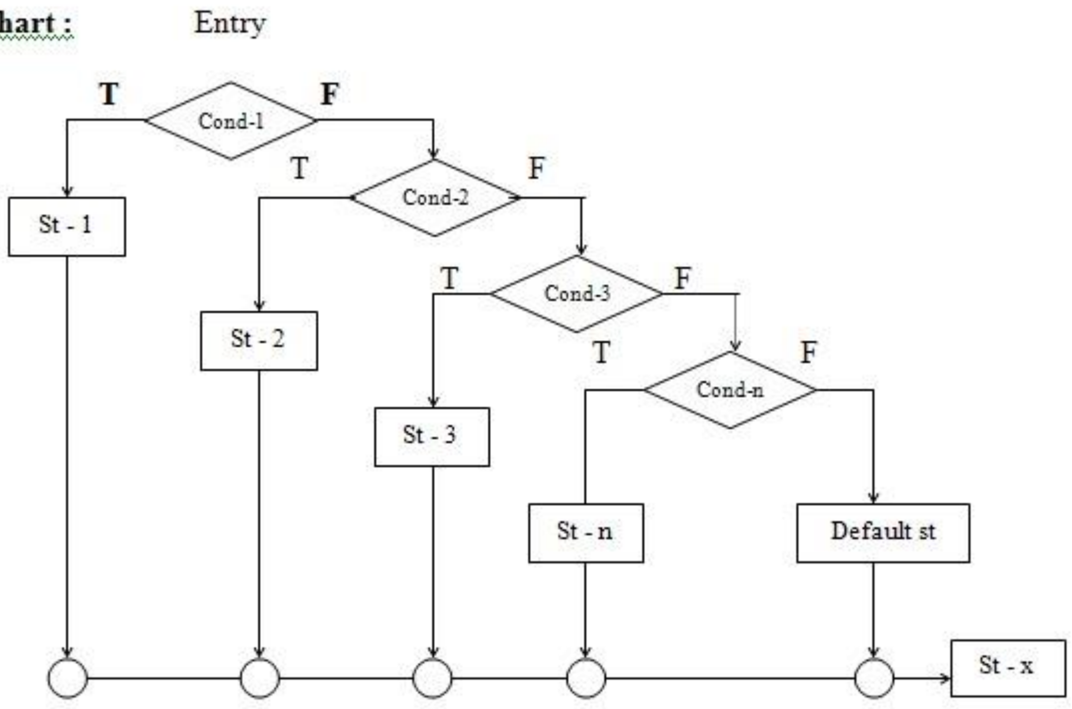
OUTPUT:

```
m is greater than n
```

Else-If ladder : A multi path decision is charm of its in which the statement associated with each else is an If. It takes the following general form.

```
                If (condition1)
        St –1;
        Else  If (condition2)
        St –2;
        Else if (condition 3)
        St –3;
        Else
        Default – st;
        St –x;
```

Flow chart :



This construct is known as the wise-If ladder. The conditions are evaluated from the top of the ladder to down wards. As soon as a true condition is found the statement associated with it is executed and the control the is transferred to the st-X (i.e.., skipping the rest of the ladder). when all the n-conditions become false then the final else containing the default – st will be executed.

Ex :

```
 #include<stdio.h>
#include<string.h>
void main()
{
    int n;
printf(" Enter 1 to 4 to select random color");
 scanf("%d",&n);
 if(n==1)
```

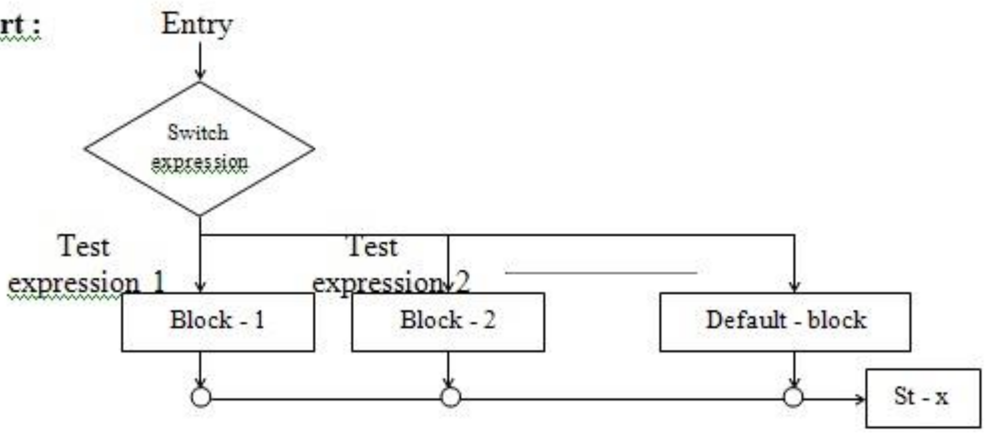
```
  {
          printf("You selected Red color");
               }
                    else if(n==2)
                    {
                         printf("You selected Green color");
                    }
                         else if(n==3)
                         {
                              printf("You selected yellow color");
                         }
                              else if(n==4)
                              {
                                   printf("You selected Blue color");
                              }
                                   else
                                   {
                                       printf("No color selected");
                                   }
                    getch();
          }
```

Switch Statement : Instead of else –if ladder, 'C' has a built-in multi-way decision statement known as a switch. The general form of the switch statement is as follows.

```
                         Switch (expression)
{
case value1   : block1;
                break;
case value 2  : block 2;
                break;

default       :  default block;
                break;
}
st – x;
```

Flow Chart :



The expression is an integer expression or character value1, value-2---- are constants or constant expressions and also known as case lables. Each of the values should be a unit within a switch and may contain zero or more statements.

When the switch is executed the value of the expression is successively compared against the values value-1,value-2------- If a case is found whose value matches with the of the expression then the block of statements that follows the case are executed .

The break statement at the end of each block signals the end a particular case and causes an exist from the switch statement transfering the control to the st-x following the switch. The default is an optional case . If will be executed if the value of the expression doesn't match with any Of the case values then control goes to the St-x.

Ex :

```
 #include <stdio.h>

int main ()

{

  int value = 3;

  switch(value)

   {

    case 1:

    printf("Value is 1 \n" );

    break;

    case 2:

     printf("Value is 2 \n" );

     break;

     case 3:

     printf("Value is 3 \n" );

     break;
case 4:
    printf("Value is 4 \n" );
   break;
    default :
   printf("Value is other than 1,2,3,4 \n" );  }
  return 0;
}
```

Output:

```
Value is 3
```

Goto Statement : The goto statement is used to transfer the control of the program from one point to another. It is something reffered to as unconditionally branching. The goto is used in the form

Goto label;

Label statement : The label is a valid 'C' identifier followed by a colon. we can precede any statement by a label in the form

Label : statement ;

This statement immediately transfers execution to the statement labeled with the label identifier.

Ex :

```
  #include <stdio.h>
int main()
{
   int i;
   for(i=0;i<10;i++)
     {
   if(i==5)
   {
     printf("\nWe are using goto statement when i = 5");
     goto HAI;
   }
   printf("%d ",i);
}

HAI : printf("\nNow, we are inside label name \"hai\" \n");
}
```

OUTPUT:

```
0 1 2 3 4
We are using goto statement when i = 5
Now, we are inside label name “hai”
```