RNS INSTITUTE OF TECHNOLOGY

Affiliated to VTU, Recognized by GOK, Approved by AICTE, New Delhi
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph: (080) 28611880, 28611881 URL: www.rnsit.ac.in

Department of Computer Science and Engineering

PRINCIPLES OF PROGRAMMING USING C
(BPOPS103/203)

MODULE 2
Operators, Decision Control & Looping Statements

Mrs. Vidya Y
Assistant Professor
Department of CSE
RNSIT

Operators And Expressions

Introduction

C supports rich set of built-in Operators such as +, -, *, & and <

- An Operator is a symbol that tells the computer to perform certain mathematical and Logical manipulation
- Operators are used in programs to manipulate data and Variables.
- C Operators can be classified into following categories.

Operator Category	Description	Operators
Arithematic Operators	perform arithematic Operations namely addition (+), Subtraction (-) multiplication (*), division (/), modulo arithematic (%), increment (++), decrement (--), negation (~) unary plus (+)	+ - * / % ++ --
Relational Operators	Perform comparission Operations namely, less than (<), greater than (>), less than or equal to (<=) and greater than equal to (>=)	< > <= >=
Equality Operators	perform equality (==) and inequality (!=) tests	== !=

Commonly used Operators in C Language

Logical Operators	Join simple conditional expressions using AND (&&), OR (||) and NOT (!) Operations to form compound expressions [Boolean expressions]	&& || !
Assignment Operators	Assign a Value (of a Constant, Variable or expression) to a Variable	= += -= *= /= %=

- Depending on number of Operands on which an Operator Operates, Operators can be grouped into three categories

1) Unary Operators → Takes only one Operand Eg -a
2) Binary Operators → Operates on 2 Operands. Eg a+b
3) Ternary Operators → takes three Operands.

Eg: (a>b) ? a : b

→ Depending on how the Operators are used with the Operands, there are three forms

1) Infix Operator: It is used between the Operands

Eg: a+d , c*b

2) prefix Operator: It is used before an Operand

Eg: -a, ++d (++ → Increment operator)

3) postfix Operator: It is used after an Operand

Eg: ++y y-- , x++

-- → decrement Operator

Arithematic Operators:

→ Arithematic Operators are used to perform arithemal Operations on arithematic Operands i.e., Operands of integral as well as floating type.

→ They are grouped into three categories

1) Unary Operators
2) Multiplicative Operators
3) Aditive Operators

Unary

Operator group	Operator	Description	Example	Associativ
Unary Operators	+	unary plus	+a	Right to Left
	-	unary Minus	-a	
	++	Prefix Increment / postfix Increment	++x / x++	
	--	Prefix & postfix Decrement	--x / x--	
Multiplicative Operators	*	Multiplication	a * b	Left to Rigr
	/	Division	a / b	
	%	Modulo arithematic	a % b	
Additive Operators	+	Addition	a + b	Left to Righ
	-	Subtraction	a - b	

Arithematic Operators in the C Language

Unary arithematic Operators

→ They include unary plus (+), unary minus (-), increment (++) and decrement (--).

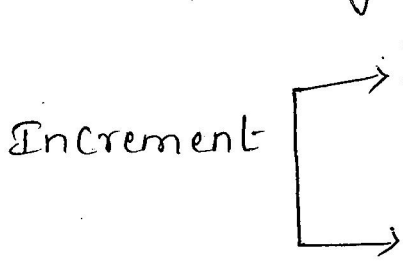
→ These Operators Operate on arithematic Operands.

Eg: -a , +10.25 , +12k etc

→ unary minus Operator negates the value of its Operand where as unary plus Operator has no effect on the value of its Operand.

Increment and Decrement Operators:

→ ++ is an Increment Operator. It increments the value of a Variable by one. Increment Operator is classified into 2 categories.



Example
a++
++a

Post Increment: If the increment Operator ++ is placed immediately after the Operand, then the Operator is post Increment

→ Here Operand Value is used first & then Operand is incremented by 1

Example:

```
#include<stdio.h>
Void main()
{
    int a=20;
    a++;
    printf("%d", a);
}
```

Output = 21

Pre Increment

- If the increment Operator ++ is placed before the Operand, then the Operator is called pre increment.
- As the name indicates, pre increment means increment before Operand Value is used. So the operand value is incremented by 1 & this incremented value is used.

Eg:
```
Void main()
{
    int a=20;
    ++a;
    printf("%d", a)
}
```
Output = 21

Note: In the above program, there is no different between pre increment and post increment.
- If post increment & pre increment Operator is not part of any expression, then there is no difference betwe pre increment & post increment.

Example:

Let a = 20

post increment

b = a++;

- current value of a used — b = a — b = 20
- a is incremented by 1 — a = a+1 — a = 21

Pre Increment

Let a = 20

b = ++a;

- a is incremented by 1 — a = a + 1 — a = 21
- incremented value of a is used — b = a — b = 21

Note: Observe from above example that even though the final values of a's remain same, the final values of b's are different.

Decrement Operator:

* -- is a decrement Operator. This is a unary Operator. It decrements value of variable by one. It is classified into 2 categories

Decrement Operator
- post decrement
- pre decrement

Post Decrement:

- If the decrement Operator -- is placed after the Operand, then the Operator is called post decrement.
- As the name indicates post decrement means decrement after the Operand value is used. So the Operand value is used first & then the Operand value is decremented by

Example

```
Void main()
{
    int a=10;
    a--;
    printf("%d", a);
}
```

output

a = 9

Pre decrement :

- If the decrement Operator -- is placed before the Operand, then the Operator is called pre decrement.
- As the name indicates pre decrement means decreme before the Operand Value is used. So the Value is decremented by 1 first & this decremented Value is used

Example :

```
Void main()
{
   int a=10;
   --a;
   printf("%d", a);
}
```

Output = 9

Note : There is no difference between post decrement or pre decrement since they are not part of any expressio

- If they are part of expression or assignment statement, there is a difference between post increment decremen & pre decrement

post decrement

Let a = 10

b = a--;

- current value of a used — b = a — b = 10
- a is decremented by 1 — a = a - 1 — a = 9

Pre decrement

Let a = 10

b = --a;

- a is decremented by 1 — a = a - 1 — a = 9
- decremented Value of a is used — b = a — b = 9

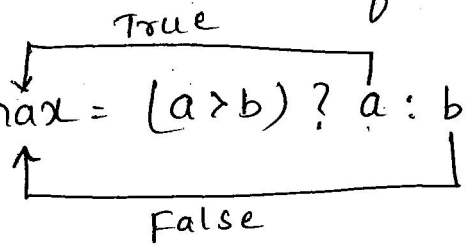
Conditional Operator

- Conditional Operator is called ternary Operator. As the name indicates, an operator that Operates on three Operands is called ternary Operator. Ternary Operators are ? and :.

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

- exp1 is an expression evaluated to true or false
- If exp1 is evaluated to true, exp2 is executed
- If exp1 is evaluated to false, exp3 is executed

Example:



Program: Demonstrate the use of conditional Operator

```
#include<stdio.h>
Void main()
{
    int a, b, big;
    printf(" Enter value of a & b\n");
    Scanf("%d%d", &a, &b);
    big = (a>b) ? a : b;
    printf(" Big = %d\n", big);
}
```

Output

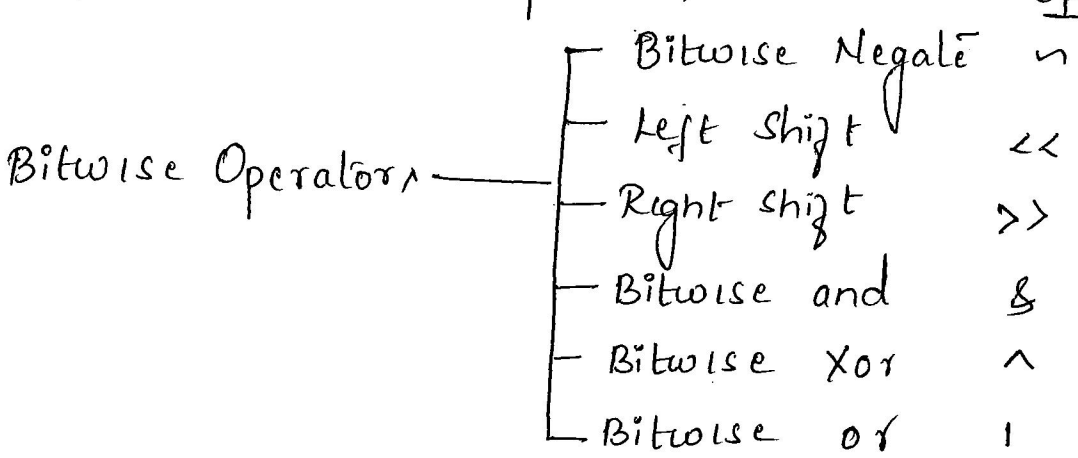
Enter the value of a and b

10 20

Big=20

Bitwise Operators:

- Operators that are used to manipulate the bits of given data are bitwise operators.



Bitwise Negate (~) [One's complement]

- The Operator is used to change every bit from 0 to 1 and 1 to 0 in specified Operand
- It is denoted by the symbol ~ (tilde)

Program to demonstrate bitwise negation Operation

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

Left shift Operator (<<)

- The Operator that is used to shift the data by a specified number of bit positions towards left
- It is denoted by <<

Syntax : b = a << num

- num: Constant or Variable
- <<: Left shift Operator
- a: Constant or Variable

- Contents of first Operand are shifted towards left by the number of bit positions specified in second Operand

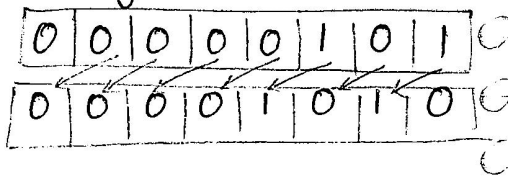
program showing Bitwise Left shift Operation.

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

Output

5 <<1 = 10

Binary representation.



Note: MSB is discarded and zero is appended at LSB

Leftshift of 1-bit is multiply by 2

Right Shift Operator (>>)

- Operator that is used to shift the data by a specified no of bit positions towards right.

→ It is denoted by symbol >>

Syntax: `b = a >> num`

- num → Constant or Variable
- >> → right shift Operator
- a → Constant or Variable

- Contents of first Operand are shifted towards right by the number of bit positions specified in second Operand.

Note: LSB bit discarded & a zero is appended at MSB

program to demonstrate the use of Right Shift Operator

```
#include<stdio.h>
void main()
{
   int a,b
   a=10
   b= a>>1
   printf("%d>>1 = %d\n", a, b);
}
```

Output

10 >>1 = 5

Binary Representation

0	0	0	0	1	0	1	0
0	0	0	0	0	1	0	1

Note: Right Shift by 1 bit is divide by 2

Bit wise AND (&)

- If the corresponding bit positions in both the operands are 1 then AND operation result is 1. Otherwise, AND operation result is 0
- bit-wise AND operator is denoted by symbol &

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

program to demonstrate the use of bit-wise AND

```
#include<stdio.h>
void main()
{
    int a, b, c;
    a = 10
    b = 6
    c = a & b;
    printf("%d & %d = %d\n", a, b, c);
}
```

Output : 10 & 6 = 2

Binary Representation

0 0 0 0 1 0 1 0	a = 10
0 0 0 0 0 1 1 0	b = 6
0 0 0 0 0 0 1 0	c = 2

Bit-wise OR (|)

- If the corresponding bit positions in both the Operands are 0, then OR Operation result is 0, Otherwise OR operation result is 1
- bit-wise OR operator is denoted by pipe symbol '|'

```
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1
```

Program showing bit-wise OR Operation

```
#include<stdio.h>
Void main()
{
    int a, b, c;
    a=10;
    b=6;
    c=a|b;
    printf("%d|%d=%d\n", a,b,c);
}
```

Output : 10|6=4

Binary Representation

0	0	0	0	1	0	1	0	a=10
0	0	0	0	0	1	1	0	b=6
0	0	0	0	1	1	1	0	c=14

Bit-wise X-OR (^)

- If the corresponding bit positions in both the Operands are different, the ex-OR operation result is 1 Otherwise result is 0

- Bit-wise X-OR Operator is denoted by symbol ^ (caret)

```
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0
```

Program showing bit-wise X-OR operation

```
#include<stdio.h>
Void main()
{
    int a, b, c;
    a=10;
    b=6;
    c=a^b;
    printf("%d ^ %d = %d\n", a,b,c);
}
```

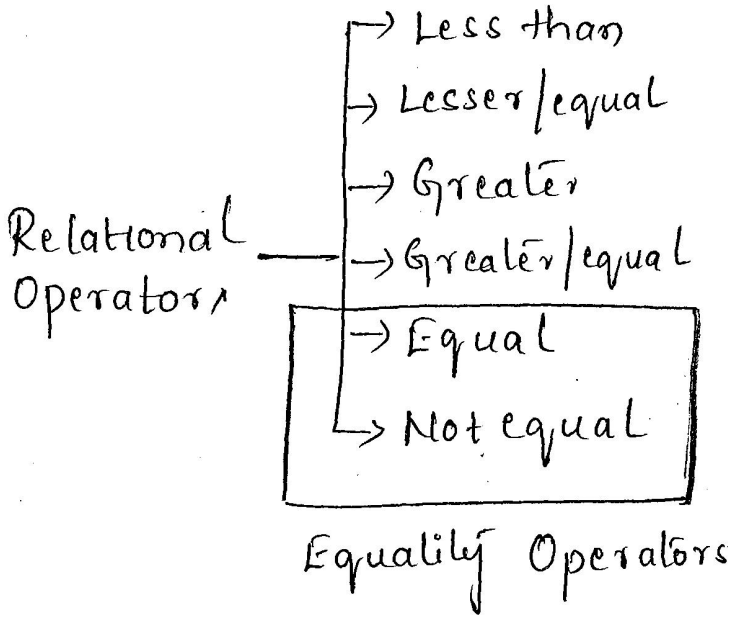
Output = 10 ^ 6 = 12

Binary Representation

0	0	0	0	1	0	1	0	a=10
0	0	0	0	0	1	1	0	b=6
0	0	0	0	1	1	0	0	c=12

Relational Operators:

- Operators that are used to find the relationship between 2 Operands are called Relational Operators.
- The relationship between 2 operand values result in true(1) or false(0).
- Operands may be Constants, Variables, expressions



- Following table gives the relation between the two operands & the result obtained

Relational expressions	Results TRUE/FALSE	Output
3 > 2	TRUE	1
2 > 3	FALSE	0
2 == 2	TRUE	1
2 != 2	FALSE	0
2 >= 2	TRUE	1
3+2 < 6	TRUE	1

Relational Expressions

- An expression that contains relational operators is called relational expression. Consider the following expressions:

```
a+b > c                 /* valid */
b*b-4*a*c == 0          /* valid */
b*b-4*a*c > 0           /* valid */
a =< b                  /* Invalid */
a <= b                  /* Invalid */
  ↓
 space
```

Precedence of Operators:

Rules to be followed while evaluating the relational expressions are below

1. Evaluate the expression within parentheses
2. Evaluate unary Operators
3. Evaluate arithmetic expressions
4. Evaluate relational expressions

Evaluate the following expression.

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

0 == 5 >= 1 != 20

0 == 1 != 20

0 != 20

result = 1

Logical Operators

- The operators that are used to combine two or more relational expressions are called Logical Operators.
- Since the output of relational expressions is true or false, the output of logical expression is also true or false.

→ Logical operators is of 3 types

		Priority
Logical Operators	not (unary operator)	1
	and (binary operator)	2
	or (binary Operator)	3

Associativity is Left to Right for all the 3 Operators.

Logical NOT (!)

- Logical not operation is denoted by operator '!' can be true or false. The result is true if the Operand is false & the result is false if the Operand is true.

Operand	!operand
1	0
0	1

Logical AND (&&)

- Output of Logical and operation is denoted by Operator '&&' is true if and only if both the Operands are evaluated to true. If one of the Operand is evaluated

false, the result is false

Operand1	AND	Operand2	Result
1	&&	1	1
1	&&	0	0
0	&&	1	0
0	&&	0	0

Eg: If a, b and c are the 3 sides of a Δ^{le}, then if
a==b && b==c && c==a then the Δ^{le} is equilateral
Otherwise Δ^{le} is not equilateral.

Logical OR: (||)

- The output of Logical OR operation denoted by Operator '||' is true if and only if at least one of the operands is evaluated to true. If both the Operands are evaluated to false, the result is false.

Operand1	OR	Operand2	Result
1	||	1	1
1	||	0	1
0	||	1	1
0	||	0	0

Example: If a, b and c are 3 sides of a Δ^{le}, then if
a==b || b==c || c==a then the Δ^{le} is isosceles Δ,
Otherwise it is not an isosceles Δ^{le}.

Evaluating Logical Expressions:

- precedence rules to be followed while evaluating th expression Consisting of Logical expression is shown bela

  * parenthesis
  * unary expressions
  * Arithematic expressions
  * Relational expressions
  * Logical expressions

1) a+2>b && !c || a!=d && a-2<=e

a=11, b=6, c=0, d=7 and e=5

11+2>6 && !0 || 11!=7 && 11-2<=5

11+2 >6 && 1 || 11!=7 && 11-2<=5

13 >6 && 1 || 11!=7 && 11-2<=5

13>6 && 1 || 11!=7 && 9<=5

1 && 1 || 11!=7 && 9<=5

1 && 1 || 11!=7 && 0

1 && 1 || 1 && 0

1 || 1 && 0 1 || 1 && 0

1 || 0 1 && 0

Result = 1

Result = 0

2) a+2>b || !c && a==d || a-2<=e

a=11, b=6, c=0, d=7 and e=5

11+2 >6 || !0 && 11==7 || 11-2<=5

11+2 >6 || 1 && 11==7 || 11-2<=5

13 >6 || 1 && 11==7 || 11-2<=5

13 >6 || 1 && 11==7 || 9<=5

1 || 1 && 11==7 || 9<=5

1 || 1 && 11==7 || 0

1 || 1 && 0 || 0

1 || 0 || 0

1 || 0

Result = 1

3) 10!=10 || 5<4 && 8

10!=10 || 5<4 && 8

10!=10 || 0 && 8

0 || 0 && 8

0 || 0

Result = 0

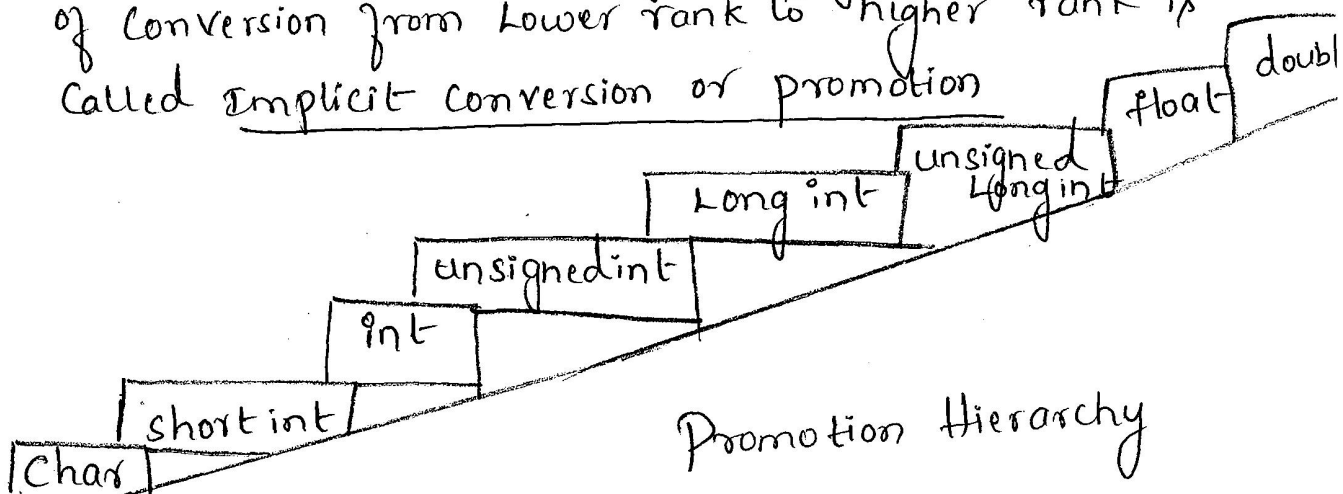
Type Conversion

- In C Language the programmer can instruct the compiler to convert the data from one data type to another an data type.
- process of converting the data from one data type to another data type is called Type Conversion
- Type conversion is done in 2 methods.

Type conversion
- Implicit Conversion
- Explicit Conversion

Implicit Conversion

- C can evaluate the expressions if only if the data types of 2 operands are same. So, if operands are of different data type, C cannot evaluate.
- In Such Situations, C Compiler converts data type with a Lower rank to the data type to higher rank. This process of Conversion from Lower rank to higher rank is called Implicit Conversion or promotion



Promotion Hierarchy

Note: when data is converted from Lower rank to higher rank data is not lost. But, if Conversion is done from higher rank to Lower rank, we may lose the data.

Example: Evaluate 4.0/3

float/int → float

4.0 / 3.0

= 1.333333

So the output of 4.0/3 = 1.333333

2) What is the output of following program Segment

```
int p=7;
float q=2;
float r;
```

r = p/q

= 7/2.0

= 7.0/2.0

r = 3.5

3) What is the output of following program Segment

```
int p=7;
int q=2;
int r;
```

r = p/q

= 7/2

r = 3

Explicit Type Conversion [Type Casting]

- If the Operands are of same data type, no Conversion takes place by the compiler
- Sometimes, type conversion is required to get desi results. In such cases, programmer can instruct th Compiler to get change the type of dat Operand fro one data type to another. This forcible conversion from one data type to another data type is called explicit type conversion

Syntax: (type) expression

Ex: (int) 9.43

Example1: Evaluate A|3 to get the answer 1.333333 using explicit conversion

A|3

(float) 4 | 3 → programmer's Instruction

4.0 | 3 → Explicit Conversion

4.0 | 3.0 → Implicit Conversion

= 1.333333

Program:

```
#include <stdio.h>
Void main()
{
    int a,b; float c;
    a=4, b=3;
    c = (float) a|b;
    printf("%f", c);
    return 0;
}
```

2) int p=7
int q=2
float r;
r=p/q

output

r=3.500000

```
#include<stdio.h>
int main()
{
    int p=7, q=2;
    float r;
    r=(float) p/q
    printf("%f", r);
    return 0;
}
```

3) Assuming a is integer with value 3, evaluate the following expressions

a) (float) (a/10)
b) (float) a/10

a) a=3
(float) (a/10)
(float) (3/10)
(float) (0)
=0.0

b) (float) a/10
a=3
(float) 3/10
3.0/10
3.0/10.0
=0.3

program to read the marks of student & print find the average.

```
#include<stdio.h>
int main()
{
    int m1, m2, m3, average;
```

```
    printf("Enter the marks of 3 subjects\n");
    scanf("%d %d %d", &m1 &m2 &m3);
    average = (float) (m1+m2+m3) /3;
    printf("%d %d %d %f", m1, m2, m3, average);
    return 0;
}
```

Assignment Operator

- Assignment expression evaluates the operand on the right side of Operator (=) and places its value in the left variable
- There are 2 forms of assignment
  1. simple assignment
  2. compound "

1. Simple assignment: Eg: a = 5;
   - The left variable will receive value 5.

2. compound assignment (shorthand notation)
   - Here left Operand is repeated as a part of the right expression

compound expression	Eg: x *= y+3 x=10 y=5 x = x * y+3 = 70	equivalent simple expression
1. a *= b		1. a = a * b
2. a += b		2. a = a + b
3. a -= b		3. a = a - b
4. a %= b		4. a = a % b
5. a /= b		5 a = a / b

Conversion of expressions :

Following table shows the rules that have to be followed while converting mathematical expressions to C expressions

Mathematical Expression	C equivalent expression
1. $\frac{a}{b}$	1. a/b
2. $s = \frac{a+b+c}{2}$	2. s = (a+b+c)/2
3. $area = \sqrt{s(s-a)(s-b)(s-c)}$	3. area = sqrt(s*(s-a)*(s-b)*(s-c))
4. ax^2+bx+c	4. a*x+b*x+c
5. $\sin\left[\frac{b}{\sqrt{a^2+b^2}}\right]$	5. sin(b/sqrt(a*a+b*b))
6. $x1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$	6. x1 = (-b+sqrt(b*b-4*a*c))/(2*a)

PRECEDENCE & ASSOCIATIVITY

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Evaluation of expressions:

1. Replace the variables by their values
2. Evaluate the expressions inside the parenthesis
3. evaluate each of expression based on precedence & associativity

1) $b = a - ++a$ given $a = 4$

$b = 5 - 5$

$= 0$

$a = 5 \quad b = 0$

2) $i++ - j--$ $\quad i=3 \; j=4 \; K=2$

$3 - 4$

3) $++K \;\%\; --j$ $\quad i=3 \; j=4 \; K=2$

$3 \;\%\; 3$

$= 0$

$i=3, \; j=3, \; K=3$

4) $j+1 \;/\; j-1$ $\quad i=3 \; j=4 \; K=2$

$1/3$

5. j++ / i-- i=3, j=4, K=2

3

j++ | 3

4 | 3

1

i=2, j=5, K=2

a
5 6 7

6. a+= (a++) + (++a) a=5

a+= 5 + (++a) // a=6

a+= 5 + 7 // a=7

a = a + 12

a = 7 + 12

a=19

a+= a++ + 6

a+= 6 + 6

a = a + 12

a = 7 + 12

a=19

7. a = (--a) - (a--) a=5

a = 4 - (a--) // a=4

a = 4 - 4 // ~~a=3~~

a = 0

a-- a=-1

8. b = (--a) - (a--) a=5

b = 4 - (a--) a=4

b = 4 - 4

b = 0

a-- a=3

Comma Operator :-

- Denoted by ',' used in declaration to separate the variables. Eg: int a, b, c;
- It has least precedence among all the operators & left associative

1. Evaluate the expression.

a = 12, 345, 678

soln **a = 12**

Since assignment operator has higher precedence over comma operator 12; copied into a & other integers are discarded.

2. printf("%d", 12, 345, 678);

o/p : 12

3. printf("%d %d", 12, 345, 678);

o/p = 12 345

Sizeof() operator:

This operator is used to determine the no. of bytes occupied by a variable or a constant in the memory.

Syntax: sizeof(operand);

Eg: sizeof(char) - 1 byte
sizeof(int) - 2 bytes
sizeof(float) - 4 bytes

Introduction to Selection / Branching / Decision Control Statements

→ Code in the C program is executed sequentially from the first line of the program to its last line. i.e

- Although this is true, in some cases we want only selected statements to be executed. These are called as Selecting or Branching or decision control statements.
- C supports 2 types of decision control statements
  1. Conditional Branching statements
  2. unconditional Branching statements

Conditional Branching Statements

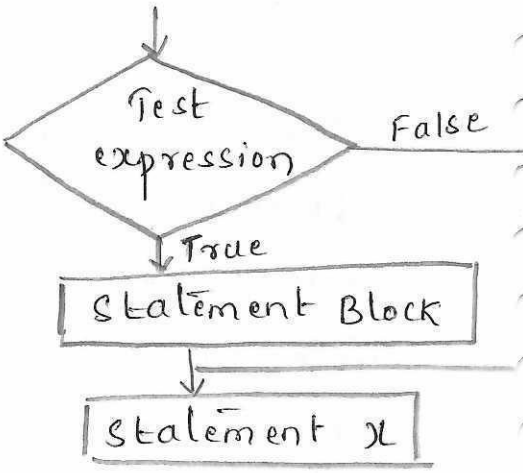
- These statements help to jump from one part of the program to another depending on whether particular condition is satisfied or not
- The statements are
  1. if statement
  2. if-else statement
  3. if-else-if statement nested if-else statement
  4. switch statement
  5. else-if ladder

The if statement :-

- The if structure may include one or more statements enclosed within curly brackets
- First the test expression is evaluated, if test expression is true, the statements of if block are executed

Otherwise these statements will be skipped & the execution will jump to statement x

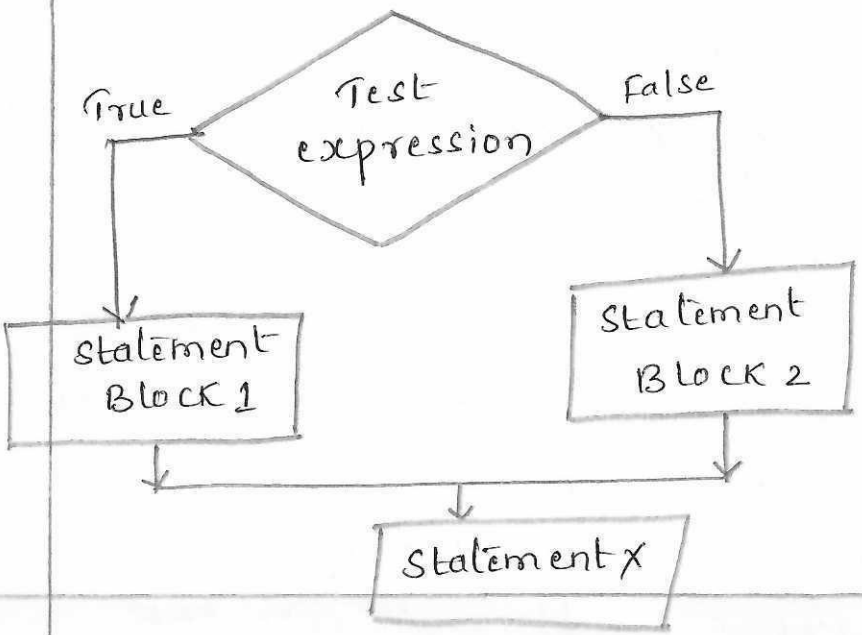
Syntax:
```
if (test expression)
{
    statement 1
    ------
    statement n;
}
statement x;
```



Note: 1. Test expression is any valid C language expression that may include logical operators.
2. No semicolon after the test expression
3. In case, statement block contains only one statement, putting curly brackets becomes optional.

The if-else statement :-

- C implements 2 way selection with if-else statement
- It is a composite statement used to make decision between 2 alternatives.




```
if (test expression)
{
    statement Block 1
}
else
{
    statement Block 2
}
statement x;
```

- Statement Block may include one or more statements.
- First test expression is evaluated, if the expression is true, statement block 1 is executed & statement Block 2 is skipped.
- Otherwise, if the expression is false, statement Block 2 is executed & statement Block 1 is skipped.
- Now in any case after statement Block 1 or 2 gets executed the control will pass to statement X. This statement is executed in every case.

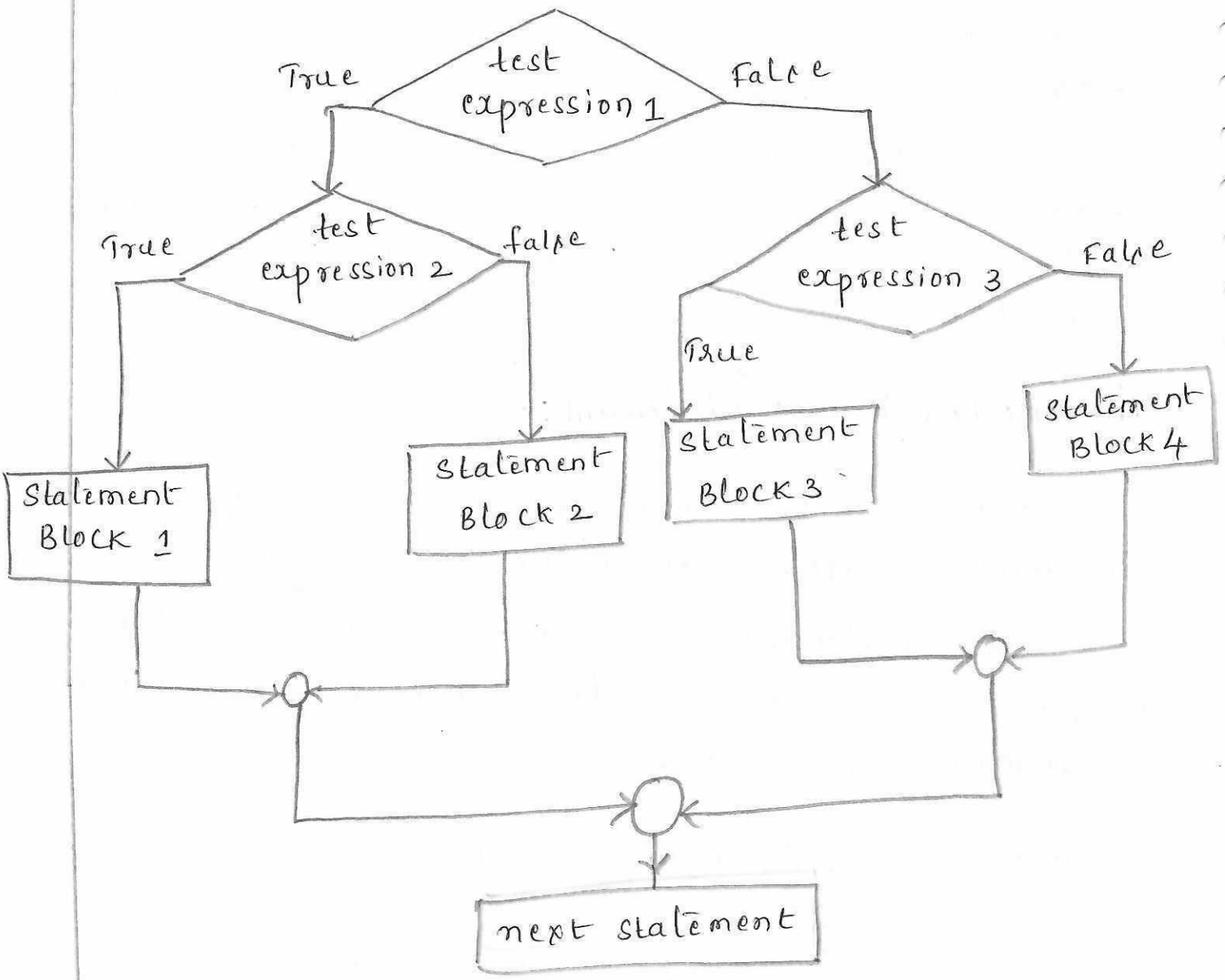
The nested if-else statement:-

- An if-else statement within another if-else statement is called nested if-else statement.
- This statement is used when an action has to be performed based on many decisions. Hence it is called a multi-way decision statement.

Syntax:

```
if (test expression 1)
{
    if (test expression 2)
    {
        Statement Block 1
    }
    else
    {
        Statement Block 2
    }
else
{
    if (test expression 3)
    {
        statement Block 3
    }
    else
    {
        statement Block 4
```

- Here Flowchart



+) Here, firstly expression 1 is evaluated to true or false

* If the expression 1 is true, then statement Block 1 is executed expression 2 is evaluated to true or false

* If the expression 2 is true, then statement Block 1 is executed

- If the expression 2 is false, then statement Block 2 is executed

2) If the expression1 is false, then expression 3 is evaluated to true or false

- If the expression 3 is true, Statement Block 3 is executed
- If the expression 3 is false, Statement Block 4 is executed

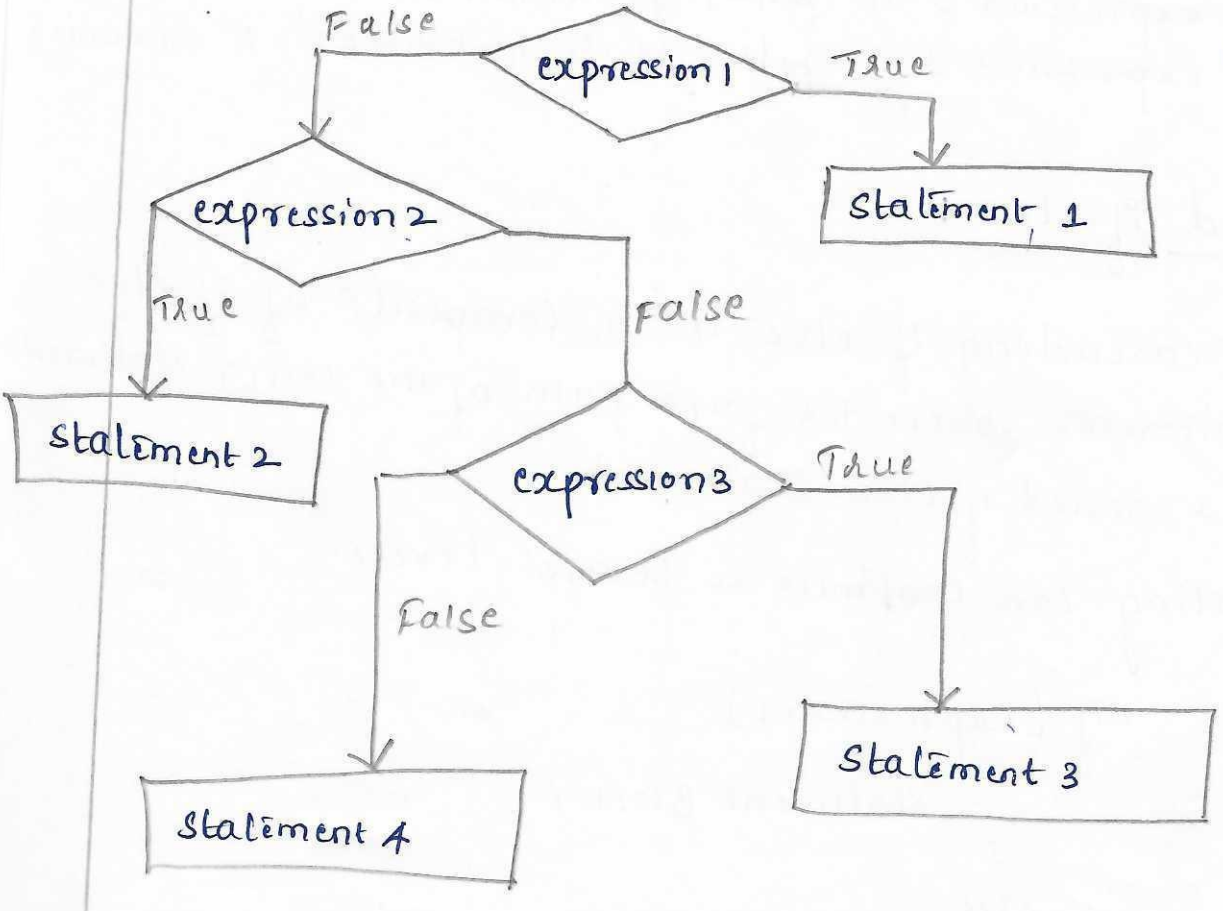
Cascaded if-else :

- A cascading if-else is a composite of if-else statements where the false path of the outer statement is a nested if-else statement.
- nesting can continue to several levels.

Syntax:

```
if (expression 1)
{
        Statement Block 1
}
else
{
    if (expression 2)
    {
        Statement Block 2
    }
    else
    {
        if (expression 3)
        {
            Statement Block 3
        }
        else
            Statement Block 4
    }
}
```

Flowchart



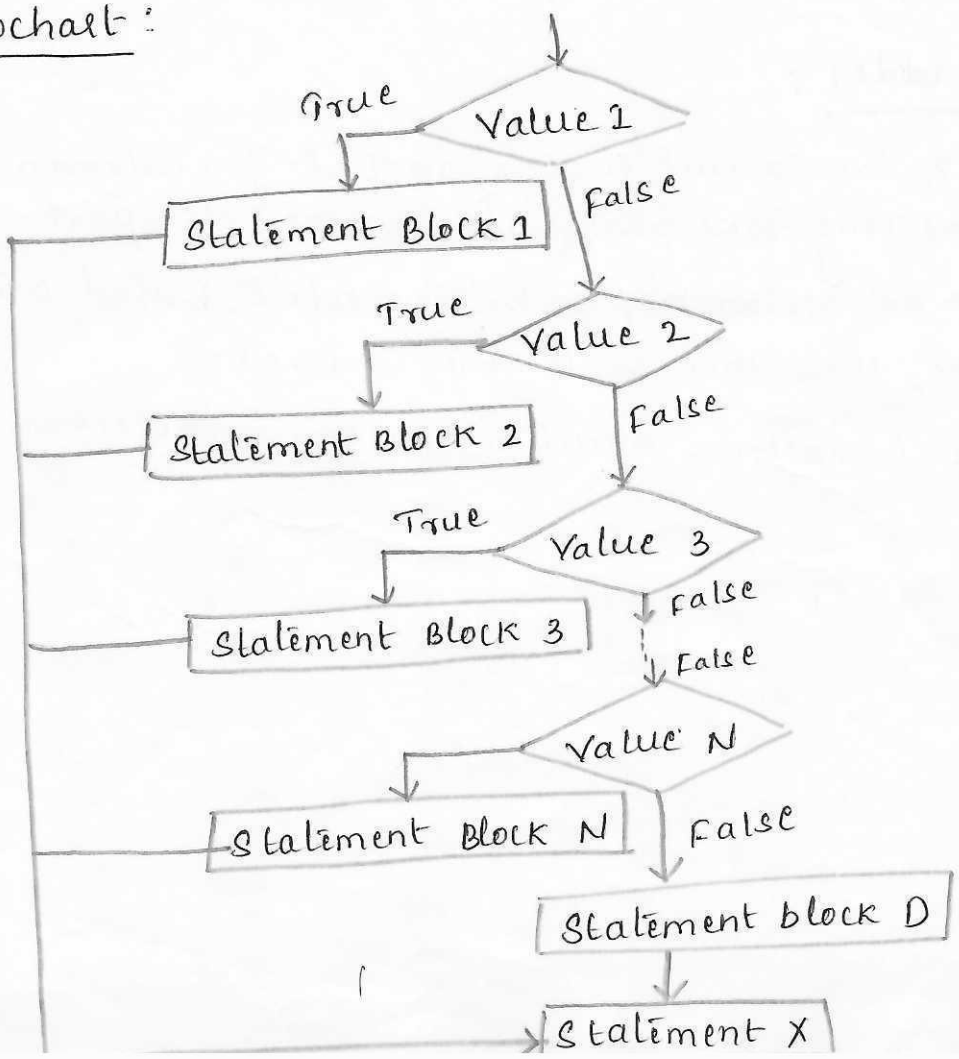
Switch case :-

- It is a multi-way decision statement.
- It is simplified version of an if-else block that evaluates only one variable
- Switch statements are used in 2 situations,
    1. Where there is only one variable to evaluate in the expression
    2. When many conditions are being tested.

Syntax :

```
switch (expression)
{
    Case Value1 : Statement Block1;
                  break;
    Case Value 2 : Statement Block 2;
                   break;
    Case Value 3 : Statement Block 3;
                   break;
    - - - - - - - - - -
    - - - - - - - - - -
    Case Value n : Statement Block n;
                   break;
    default : statement block D;
}
Statement X;
```

Flowchart :



- Expression that follows the keyword switch must be of an integral type (i.e either be an integer or any value that can be converted to an integer)
- Each case label is the keyword case followed by constant expression
- No two case labels can have same constant expression value
- But two case labels can be associated with same set of actions.
- Default label is optional & is executed only when the value of the expression does not match with any labelled constant expression.
- Switch statement can include atmost one default label. Default may be coded anywhere, but it is coded last.

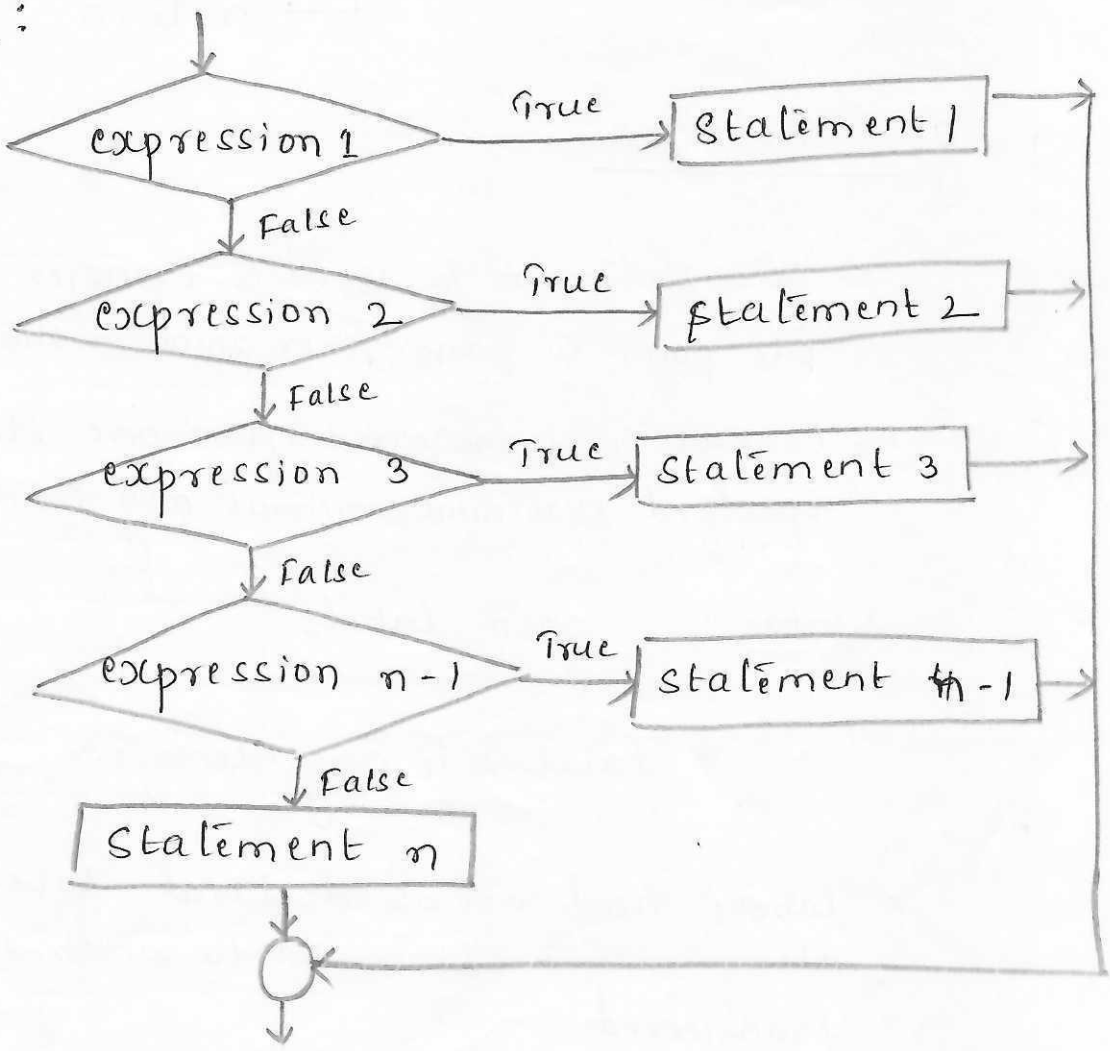
else-if Ladder:

- else-if is the special case of nested-if statements, where nesting take place only in the else part.
- etc when an action has to be selected based on ranges of values, then this statement is used.
- Hence it is called multi-way decision/selection statement

Syntax:

```
if (expression 1)
        Statement 1;
else if (expression 2)
        statement 2;
else if (expression 3)
        Statement 3;
    - - - - - -
    - - - - - -
else if (expression n-1)
        Statement n-1;
else
        Statement n;
```

Flowchart:

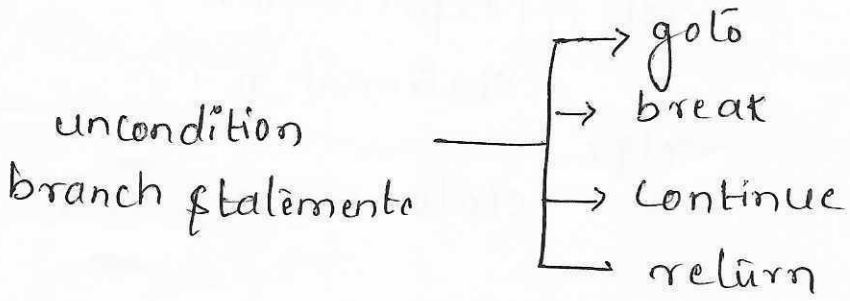


Unconditional Branch Statements:

- Statements that transfer the control from one statement to other statement in the program without any condition are called un-conditional branching statements

- unconditional branch statements are classified as

1.



goto statement:

* goto statement is used to transfer the control from one point to some other point in the program
* control is transferred from one statement to the specified statement without any condition.

Syntax: goto label;

label → is any identifier

* labels need not be declared. label should be used along with a statement to which the control is transferred

Example :

```
void main()
{
    int sum=0, i=0;
top: if (i>10) goto end;
    sum = sum+i;
    i++;
    goto top;
end: printf("sum=%d\n", sum);
}
```

Op : 55

Break statement :-

* Break statement is Jump statement used in switch statement and Loops.
* Break statement in switch statement causes control to terminate switch statement & the statement following switch statement will be executed
* If Break executed in a loop, the control comes out of the loop & the statement following the loop will be executed.
* Break statement is used mainly to terminate the loop when a specific condition is reached.

Example:

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

Olp: 1 2 3 4

```
for (----)
{
-----
    for (---)
    {
    ------
        if (error1)
            break;
    }  -----
    ::::::
}
```

The Continue Statement

* Continue statement is a Jump statement. The continue statement can be used inside the loops
* Execution of these statement does not cause an exit from the loop but it suspend the execution of the loop

for that iteration and transfer control back to the loop for the next iteration

Syntax: continue;

Example:

```
int main()
{
    int i;
    for(i=1; i<=5; i++)
    {
        if(i==3)
            continue;
        Printf("%d\n", i);
    }
    return 0;
}
```

o/p:
1
2
4
5

Return statement

* 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.

* It is also used to return a value to the calling function.

Syntax: return expression;

Note: Expression is optional

* If no return statement appears in a function definition, control automatically returns to the calling function after the last statement of the called function is executed.

```
void function1();
int function3();
void main()
{
    int i;
    function1();
    i=function2();
    Printf("Returned value = %d\n", i);
    getch();
}
void function1()
{
    Printf("First\n");
}
int function3()
{
    Printf("Third\n");
    return 10;
    Printf("second\n"); // unreachable code
}
```

Op:

First

Returned value = 10

PROGRAMMING EXAMPLES

1.Program to check whether entered number is positive negative or zero using nested-if construct.

```
#include<stdio.h>
int main()
{
      int num;
      clrscr();
      printf("enter a number: ");
      scanf("%d",&num);
      if(num<=0)
      {
          if(num==0)
               printf("number is zero\n");
          else
               printf("%d is negative\n",num);
      }
      else
          printf("%d is positive\n",num);
      return 0;
}
```

Output:

```
enter a number: 100
100 is positive
enter a number: -23
-23 is negative
enter a number: 0
number is zero
```

2. program to determine whether a person is eligible to vote using if construct.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int age;
     clrscr();
     printf("enter the age: ");
     scanf("%d",&age);
     if(age>=18)
          printf("you are eligible to vote");
     return 0;
}
```

Output:

```
enter the age: 18
you are eligible to vote
```

3. program to find largest of two numbers using if-else construct.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int a,b,large;
     clrscr();
     printf("enter the values of a and b:");
     scanf("%d%d",&a,&b);
     if(a>b)
          large=a;
     else
          large=b;
     printf("large=%d",large);
     return 0;
}
```

Output:

```
enter the values of a and b:23  64
large=64_
```

4. program to find whether the given number is odd or even using if-else construct.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int num;
     //   clrscr();
     printf("\nenter any number:");
     scanf("%d",&num);
     if(num%2==0)
          printf("%d is is an even number",num);
     else
          printf("%d is an odd number",num);
     return 0;
}
```

Output:

```
enter any number:44
44 is is an even number
enter any number:33
33 is an odd number
```

5. Program to enter any character. If the entered character is in lower case then convert it into upper case and if it is a lower case convert it into upper case using if-else construct.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     char ch;
     clrscr();
     printf("\nenter any charcter:");
     scanf("%c",&ch);
     if(ch>='A' && ch<='Z')
          printf("the entered character was in upper case. In lower
case it is: %c",(ch+32));
     else
          printf("the entered character was in lower case. In upper
case it is: %c",(ch-32));
     return 0;
}
```

Output:

```
enter any charcter:A
the entered character was in upper case. In lower case it is: a
enter any charcter:f
the entered character was in lower case. In upper case it is: F
```

6.Program to enter a character and then determine whether it is vowel or not using if-else

```
#include<stdio.h>
#include<conio.h>
void main()
{
     char ch;
     clrscr();
     printf("\nenter any charcter:");
     scanf("%c",&ch);
     if(ch=='A' ||ch=='E'||ch=='I'||ch=='O'||ch=='U'||ch=='a'||
          ch=='e'||ch=='i'||ch=='o'||ch=='u')
```

```
            printf(" %c is a vowel",ch);
      else
            printf(" %c is not a vowel",ch);
      getch();
 }
```

Output:

```
enter any charcter:E
 E is a vowel
enter any charcter:g
 g is not a vowel
```

7.program to find largest of three numbers using nested if-else construct

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int n1,n2,n3;
      clrscr();
      printf("enter the numbers\n");
      scanf("%d%d%d",&n1,&n2,&n3);
      if(n1>n2)
      {
            if(n1>n3)
                  printf("%dis greatest\n",n1);
            else
                  printf("%dis greatest\n",n3);
      }

      else
            if(n2>n3)
                  printf("%dis greatest\n",n2);
            else
                  printf("%dis greatest\n",n3);
      return 0;
}
```

```
enter the numbers
23
56
19
56is greatest
```

8. program to find largest of three numbers using logical && operator

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int n1,n2,n3;
        //  clrscr();
      printf("enter the numbers\n");
      scanf("%d%d%d",&n1,&n2,&n3);
      if(n1>n2 && n1>n3)
            printf("%dis greatest\n",n1);
      else if(n2>n1 && n2>n3)
            printf("%dis greatest\n",n2);
      else
            printf("%dis greatest\n",n3);
      return 0;
}
```

Output:

```
enter the numbers
89
56
23
89is greatest
```

9. program to enter marks of a student in four subjects.Then calculate the total, aggregate and display the grades obtained by the student.

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int m1,m2,m3,m4,tot;
      float avg;
      clrscr();
      printf("enter the marks in four subjects\n");
      scanf("%d%d%d%d",&m1,&m2,&m3,&m4);
      tot=m1+m2+m3+m4;
      avg=tot/4;
      printf("total=%d\n",tot);
      printf("aggregate=%f\n",avg);
```

```
        if(avg>=75)
              printf("\n distinction");
        else if(avg>=60  &&  avg<75)
              printf("\n first division");
        else if(avg>=50  &&  avg<60)
              printf("\n second division");
        else if(avg>=40  &&  avg<50)
              printf("\n third division");
        else
              printf("fail");

        return 0;
}
```

Output:

```
enter the marks in four subjects
23
56
89
21
total=189
aggregate=47.000000

 third division
```

10.program to perform arithmatic operations using switch statement

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

void main()
{
      int a,b,res,choice;
      clrscr();
      printf("enter the value of a and b\n");
      scanf("%d%d",&a,&b);
      printf("press
1:addition\n2:subtraction\n3:multiplication\n4:division\n5:modulus\n")
;
      printf("enter the choice\n");
      scanf("%d",&choice);
      switch(choice)
      {
            case 1:res=a+b;
                  break;
            case 2:res=a-b;
                  break;
            case 3:res=a*b;
```

```
                break;
            case 4:res=a/b;
                break;
            case 5:res=a%b;
                break;
            default :printf("invalid choice\n");
        }
        printf("res=%d",res);
}
```

Output:

```
enter the value of a and b
5
5
press 1:addition
2:subtraction
3:multiplication
4:division
5:modulus
enter the choice
1
res=10
```

11.program to find the area of square,rectangle,triangle,circle using switch

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

void main()
{
     int choice;
     int s,l,b,ba,h,r;
     float area;
     clrscr();
     printf("press 1:area of square\n2:area of rectangle\n3:area of
triangle\n4:area of circle\n");
     printf("enter the choice\n");
     scanf("%d",&choice);
     switch(choice)
     {
          case 1:printf("enter the side\n");
                 scanf("%d",&s);
                 area=s*s;
                 printf("area=%f",area);
                 break;
          case 2:printf("enter the value of length and breadth\n");
```

```
            scanf("%d%d",&l,&b);
            area=l*b;
            printf("area=%f",area);
            break;
case 3:printf("enter base and height\n");
            scanf("%d%d",&ba,&h);
            area=0.5*ba*h;
            printf("area=%f",area);
            break;
case 4:printf("enter the radius");
            scanf("%d",&r);
            area=3.142*r*r;
            printf("area=%f",area);
            break;
default : printf("invalid choice");
}
}
```

Output:

```
press 1:area of square
2:area of rectangle
3:area of triangle
4:area of circle
enter the choice
1
enter the side
4
area=16.000000
```

Leap Year

```
int main()
{
   int year;
   Sf("%d", &year);
if(y%4==0) && (year%100!=0) ||
                (year%400==0)
        Pf("Leap Year");
else
        Pf("Not leap year");
```

using switch

1. vowel or not

```
int main()
{
   char ch;,ch1;
   Sf("%c", &ch);
   ch1=tolower(ch);
   switch(ch1)
   {
     case 'a':
     case 'e':
          'i':
          'o':
          'u': printf("It is a vowel")
     default: printf(" consonant");
   }
}
```