Module 2

OPERATORS IN C

An operator is a symbol that specifies the mathematical, logical or relational operation to be performed. C language supports different types of operators, which can be used with variables and constants to form expression. C divides the operators into the following groups:

- Arithmetic operator
- Relational operators
- Equality operators
- Logical operators
- Unary operators
- Conditional operators
- Bitwise operators.
- Comma operator
- Sizeof operator

Arithmetic variables

Consider three variables declared as

int a=9, b=3 , result;

table below shows the arithmetic operators, their syntax and usage in c

Operation	Operator	Syntax	Comment	Result
Multiply	*	a*b	result = a*b	27
Divide	/	a/b	result = a/b	3
Addition	+	a+b	result = a+b	12
Subtraction	-	a-b	result = a-b	6
Modulus	%	a%b	result = a%b	0

Arithmetic operators can be applied to any integer or floating-point numbers. The addition, subtraction, multiplication and division perform the usual arithmetic operations in C.

The modulus operator (%) finds the reminder of an integer division. This operator can be applied to only integers and not to floating point and double operands.

Relational operators

A relational operator also known as comparison operators, that compares two values. Expression with relational operators are called relational expression.

Relational operators returns a true(1) value if the condition holds , otherwise returns a false(0).

Relational operators can be used to determine the relationship between two operands. These are illustrated below.

Operator	Meaning	Example
<	Less than	3<5 gives 1
>	Greater than	7>9 gives false
<=	Less than or equal to	100<=100 gives 1
>=	Greater than equal to	50>=100 gives 0

The relational operators are evaluated from left to right. The operands of relational operator must evaluate to a number.

When arithmetic expressions are used on either side of the relational operator, then first the arithmetic expression will be evaluated and then the result will be compared. This is because arithmetic operators have higher precedence that the relational operators.

Relational operators should not be used for comparing string.

Equality operator

Used to compare the operands for strict equality or inequality. They are (==) and not equal to (!=) operators. The equality operators have lower precedence than the relational operators.

Operator	Meaning
==	Returns 1 if both operands are equal, 0 otherwise.
!=	Returns 1 if both operands do not have same value, 0 otherwise.

Logical operators

C language supports three logical operators- logical AND(&&), logical OR(||) and logical NOT(!). the logical expressions are evaluated from left to right.

Logical AND

It is a binary operator, which simultaneously evaluates two values or relational expression. If both the operands are true, then the whole expression evaluates to true. If both or one of the operands are false, then the whole expression evaluates to false.

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

Example :

(a<b) && (b>c)

The whole expression is true only if both expressions are ture, i.e. , if b is greater than both a and c.

Logical OR

It is binary operator that returns a false value if both the operands are false. Otherwise it returns a true. Value. The truth table of logical OR operator is given below .

A	B	A||B
0	0	0
0	1	1
1	0	1
1	1	1

Example :

(a<b) || (b>c)

The whole expression is true only if both expressions are ture, i.e. , if either b is greater than a or b is greater than c b is greater than both a and c.

Logical NOT

The logical not take a single expression and negates the value of the expression. It produces a 0 if the expression evaluates to a non-zero value and produces a 1 if the expression evaluates a 0.

It just reverses the value of the expression.

A	!A
0	1
1	0

Example

int a=1, b;

b= !a;

now the value of b=0. This is because value of a=1. !a=0. The value of !a is assigned to b , hence the result.

Unary operators

Unary operators are the operators that perform operations on a single operand to produce a new value.

- **Unary Minus-** The minus operator (–) changes the sign of its argument. A positive number becomes negative, and a negative number becomes positive.
  int a = 10
  int b = -a; //value of b = -10
- **Increment (++) and Decrement (- -) operator**
  The increment operator (++) is used to increment the value of the variable by 1. The increment can be done in two ways:

a) prefix increment- In this method, the operator precedes the operand (e.g., ++a). The value of the operand will be altered before it is used.
Example: int a = 1;
int b = ++a; // b = 2

b) postfix increment- In this method, the operator follows the operand (e.g., a++). The value operand will be altered after it is used.
Example: int a = 1;
int b = a++; // b = 1
int c = a; // c = 2

```
#include <stdio.h>

int main()
{
    int a = 5;
    int b = 5;
    printf("Pre-Incrementing a = %d\n", ++a);
    printf("Post-Incrementing b = %d", b++);
    return 0;
}
```

- The decrement operator (—) is used to decrement the value of the variable by 1. The decrement can be done in two ways:
  A. prefix decrement- In this method, the operator precedes the operand (e.g., – -a). The value of the operand will be altered before it is used.
  Example:
  int a = 1;
  int b = --a; // b = 0
  B. postfix decrement- In this method, the operator follows the operand (e.g., a- -). The value of the operand will be altered after it is used.
  Example: int a = 1;
  int b = a--; // b = 1
  int c = a; // c = 0

```
#include <stdio.h>

int main()
{
    int a = 5;
    int b = 5;
    printf("Pre-Decrementing a = %d\n", --a);
    printf("Post-Decrementing b = %d", b--);
    return 0;
}
```

Conditional operator

The conditional operator in C is kind of similar to the if-else statement. The conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the ternary operator in C as it operates on three operands.

The conditional operator can be in the form

variable = Expression1 ? Expression2 : Expression3;

Or

variable = (condition) ? Expression2 : Expression3;

Example

```
#include <stdio.h>
  int main()
{
    int m = 5, n = 4;

    (m > n) ? printf("m is greater than n that is %d > %d",
                     m, n)
            : printf("n is greater than m that is %d > %d",
                     n, m);

    return 0;
}
```

Bitwise operators

The bitwise operators are the operators used to perform the operations on the data at the bit-level. When we perform the bitwise operations, then it is also known as bit-level programming. It consists of two digits, either 0 or 1. It is mainly used in numerical computations to make the calculations faster.

We have different types of bitwise operators in the C programming language. The following is the list of the bitwise operators:

Operator	Meaning of operator
&	Bitwise AND operator
|	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator (unary operator)
<<	Left shift operator

- Bitwise AND operator- Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator. If the corresponding bits of both the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

For example, We have two variables a and b.

a =6;

b=4;

The binary representation of the above two variables are given below:

a = 0110

b = 0100

When we apply the bitwise AND operation in the above two variables, i.e., a&b, the output would be:

Result = 0100

- Bitwise OR operator- The bitwise OR operator is represented by a single vertical sign (|). Two integer operands are written on both sides of the (|) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.
  For example,
  We consider two variables,
  a = 23;
  b = 10;
  The binary representation of the above two variables would be:
  a = 0001 0111
  b = 0000 1010
  When we apply the bitwise OR operator in the above two variables, i.e., a|b , then the output would be:
  Result = 0001 1111
- Bitwise exclusive OR operator- Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.
  For example,
  We consider two variables a and b,
  a = 12;
  b = 10;
  The binary representation of the above two variables would be:
  a = 0000 1100
  b = 0000 1010
  When we apply the bitwise exclusive OR operator in the above two variables (a^b), then the result would be:
  Result = 0000 1110

  Comma operator

The comma operator in C takes two operands. It takes by evaluating the first and discarding its value, and then evaluates the second and returns the value as result of the expression. With multiple comma separated values the expression is evaluated from left to right and yields the rightmost result. The comma operator has the least precedence.
Example
int a=2, b=3, x=0
x=(++a, b=+a);
output: the value of x=6

sizeof() operator
It determines the size of the expression or the data type specified in the number of char-sized storage units. The sizeof() operator contains a single operand which can be either an expression or a data typecast where the cast is data type enclosed within parenthesis.
int a=2;
unsigned int result;
result =sizeof(a);
output: the value of result =2,
which is the space required to store the variable **a** in the memory. Since a in an integer, it requires 2 bytes of storage space.

Operator precedence chart

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

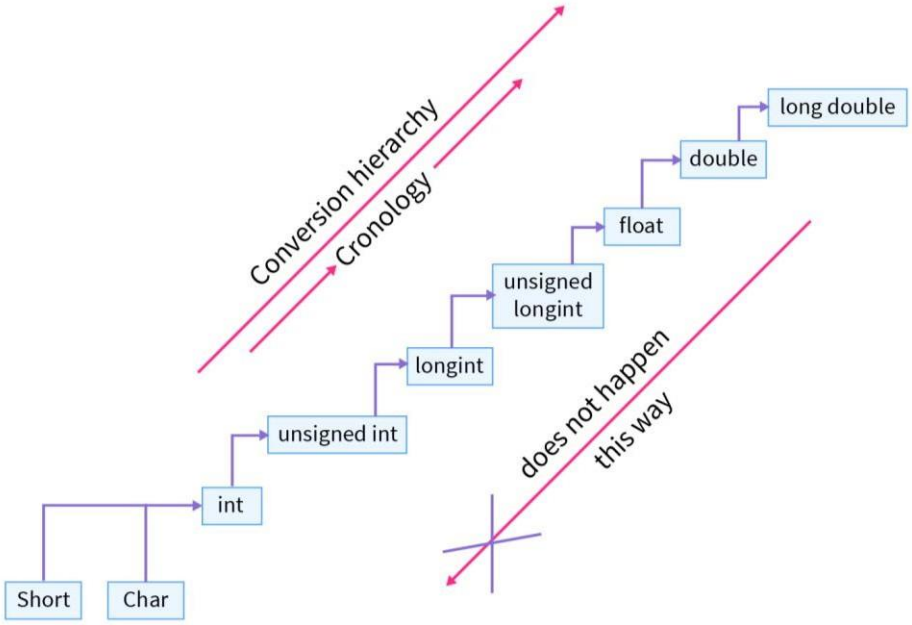
OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Type conversion and type casting

Type conversion or type casting of variables refers to changing a variable of one datatype to another. Type conversion is done implicitly, whereas typecasting has to be done explicitly by the programmer.

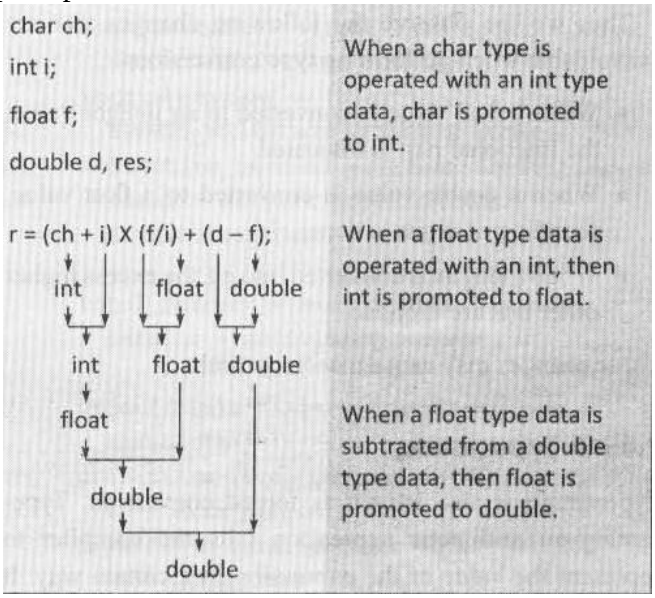
Type conversion

Type conversion is done when the expression has variables of different types. To evaluate the expression , the datatype is promoted from lower to higher level where the hierarchy of data types can be given as : double, float, long, short and char.



Example :

float x;
int y=3
x=y; //the integer data type is promoted to float. This is known as Promotion.



- float operands are converted to double.
- char or short operands whether signed or unsigned are converted to int.

- if any one operand is double, the other is also converted to double. Hence the result is also double.
- If any one operand is long , the other is also converted to double. Hence the result is also long.

Type casting

It is also known as forced conversion. Type casting an arithmetic expression tell the compiler to represent the expression in a certain way.

It is done when the value of higher datatype has to be converted into the value of lower data type. But it has to be done by the programmer and not in the control of compiler.

Example

float salary =10000.000
int sal
sal =(int) salary;
Type casting is also done in arithmetic operations to get correct results. For example when dividing two integers , the result can be of float type. Also when multiplying two integers the result can be of long int . so to get correct precision type casting can be done.
int a=500, b=70;
float res;
res = (float)a/b;

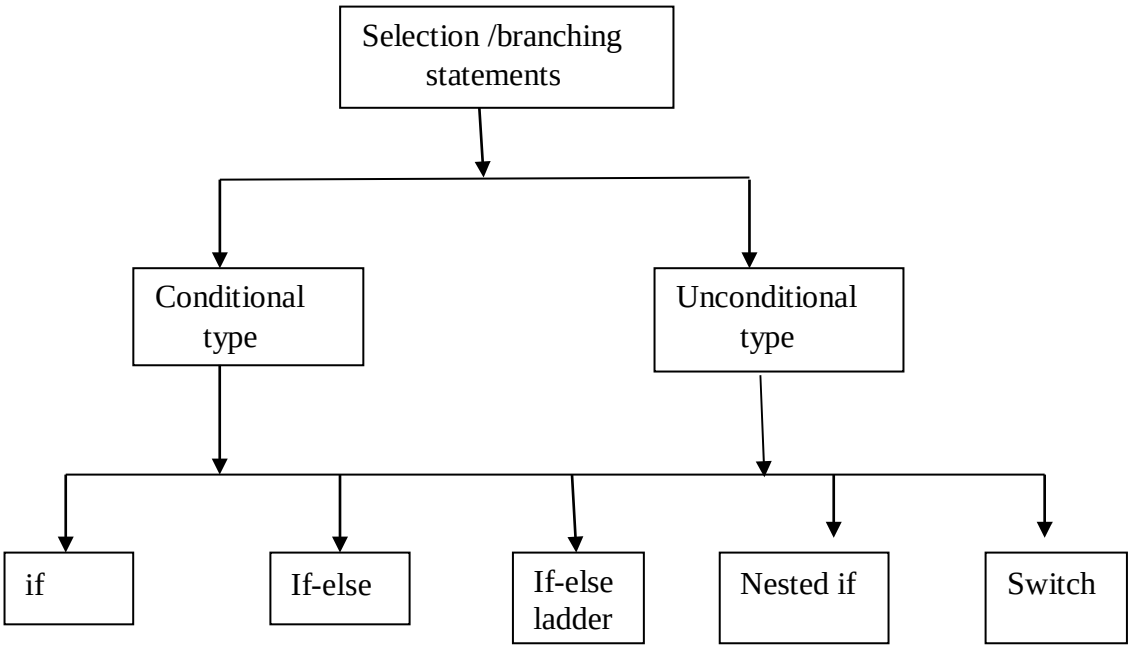
Let us look at some more examples of typecasting.

- `res = (int)9.5;`
  9.5 is converted to 9 by truncation and then assigned to `res`.
- `res = (int)12.3 / (int)4.2;`
  It is evaluated as 12/4 and the value 3 is assigned to `res`.
- `res = (double)total/n;`
  `total` is converted to double and then division is done in floating point mode.
- `res = (int)(a+b);`
  The value of `a+b` is converted to integer and then assigned to `res`.
- `res = (int)a + b;`
  `a` is converted to int and then added with `b`.
- `res = cos((double)x);`
  It converts `x` to `double` before finding its cosine value.

Decision control and looping statements

Introduction to decision control:

- The code in c program is executed sequentially from the first line of the program to its last line, i.e., the second statement is executed after the first, the third statement is executed after the second and so on.
- C support two types of decision control statements that can alter the flow of sequence of instructions. These include conditional and unconditional branching.



Conditional statements:.

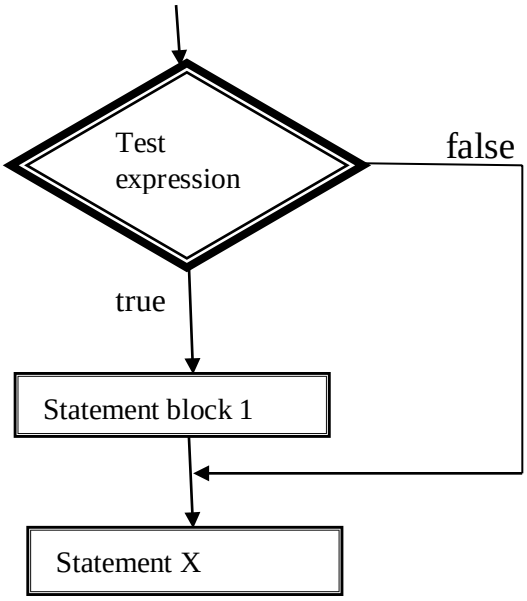
- The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not.
- The decision control statement includes:
    1. if statement
    2. if- else statement
    3. if-else ladder
    4. Nested if
    5. Switch

if statement:

The if statement is the simplest form of decision control statements that is frequently used indecision making.

Syntax:

```
if ( test Expression)
{
   Statement1;
 }
Statement2;
```



- .The if block may include one statement or n statements enclosed within curly brackets. First the test expression is evaluated.
- If the test expression is true, the statement of *if block* (statement 1 to n) are executed, otherwise these statements will be skipped and the execution will jump to statement.
- The statement in an if construct is any valid c language statement and the test expression is any valid C language expression that may include logical operators.
- Note that there is no semicolon after the test expression. This is because the condition and statement shouldbe placed together as a single statement.

Example:

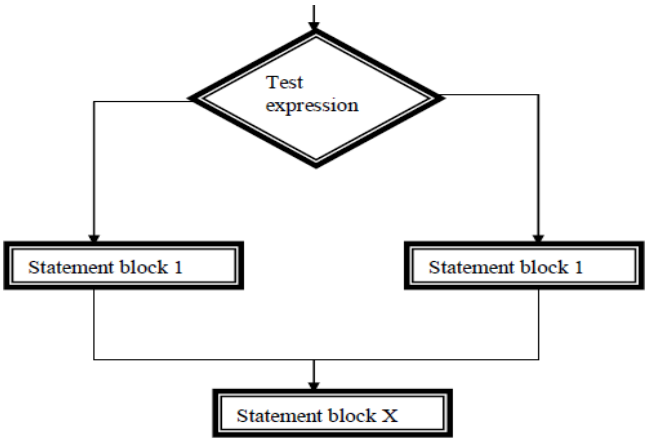
```
#include
<stdio.h>
int main ()
{
  /* local variable definition */
 int a = 10;
  /* check the boolean condition using if statement
  */
   if( a < 20 )
  {
    /* if condition is true then print the following */
  printf("a is less than 20\n" );
  }
  return 0;
}
```

if-else statement:

- The if-else statement is an extension of simple if statement.
- The test expression is evaluated, if the result is true, the statement followed by the expression is executed else if the expression is false, the statement is skipped by the compiler.

Syntax:

```
if (test Expression)
{
    Statement1; → true-block
}
else
{
   Statement2; →true-block
}
 Statement X;
```



- If the Expression is true (or non-zero) then **Statement1** will be executed; otherwise ifit is false (or zero),then **Statement2** will be executed.
- In this case either true block or false block will be executed, but not both.
- In both the cases, the control is transferred subsequently to the **Statement X.**

Example:

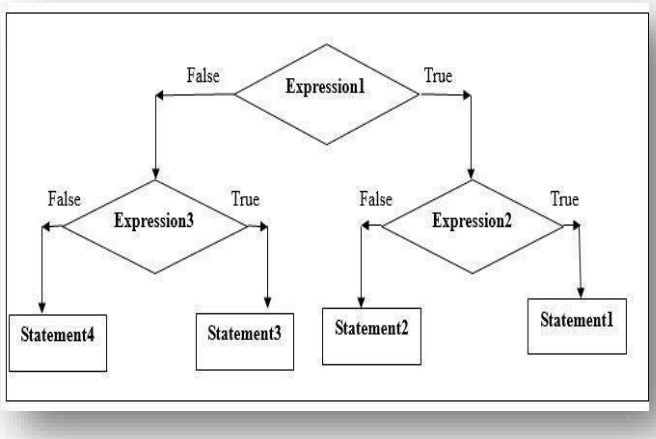
```
#include <stdio.h>int

main () {

  /* local variable definition */int a
  = 100;

  /* check the boolean condition */if( a <
  20 ) {
    /* if condition is true then print the following */
    printf("a is less than 20\n" );
  } else {
    /* if condition is false then print the following */
    printf("a is not less than 20\n" );
  }
  printf("value of a is : %d\n", a);return 0;
}
```

Nested if .. else statement:

When a series of decisions are involved, we have to use more than one if..else statement in nested form as shown below in the general syntax.

Syntax

```
if (Expression1)
{
  if(Expression2)
  {
        Statement1;
  }
  else
 {
        Statement2;
  }
}
else if(Expression3)
{
        Statement3;
}
else
{
        Statement4;
}
```



- If Expression1 is true, check for Expression2, if it is also true then Statement1 isexecuted.
- If Expression1 is true, check for Expression2, if it is false then Statement2 is executed.
- If Expression1 is false, then Statement3 is executed.
- Once we start nesting if ..else statements, we may encounter a classic problem known as dangling else.
- This problem is created when no matching else for every if.
- C solution to this problem is a simple rule “always pair an else to most recentunpaired if in the currentblock”.
- Solution to the dangling else problem, a compound statement.
- In compound statement, we simply enclose true actions in braces to make thesecond if a compoundstatement.

Example:

```
include <stdio.h>
int main()
{
int age;
```

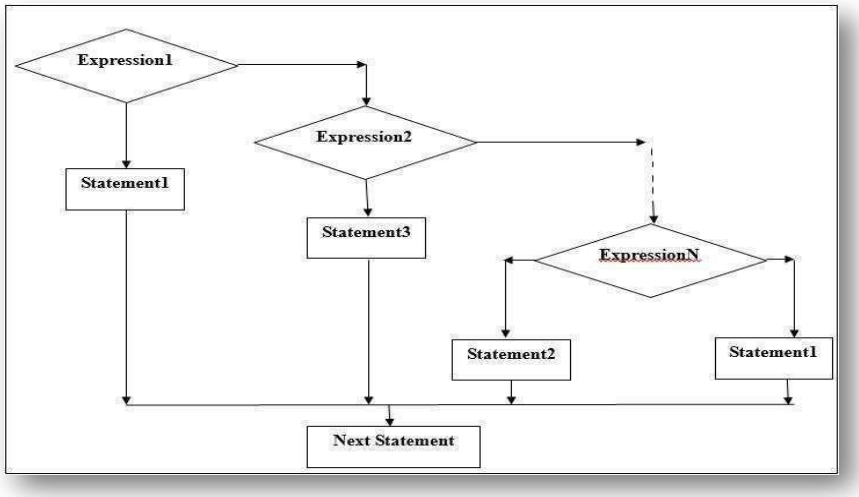
```
printf("Please Enter Your Age Here:\n");
scanf("%d",&age);
 if ( age < 18 )
 {
 printf("You are Minor.\n"); printf("Not Eligible to Work");
 }
 else
 {
 if (age >= 18 && age <= 60 )
 {

 printf("You are Eligible to Work \n");
 printf("Please fill in your details and apply\n");
 }
 else
 {
 printf("You are too old to work as per the Government rules\n");
 printf("Please Collect your pension! \n");
 }
 }
 return 0;
 }
```

if –else ladder:

There is another way of putting ifs together when multipath decisions are involved. A multi path decision is a chain of ifs in which the statement associated with eachelse is an if.

syntax:

```
if (Expression1)
{
        Statement1;
}
        else if(Expression2)
        {
                Statement2;
        }
                else if(Expression3)
                {
                        Statement3;
                }
                        else
                        {
                                Statement 4;
                        }
Next Statement ;
```



- This construct is known as the else if ladder.
- The conditions are evaluated from the top (of the ladder), downwards. As soon as true condition is found, the statement associated with it is executed and control transferred to the Next statement skipping the rest of the ladder.
- When all conditions are false then the final else containing the default Statement4 will be executed.
- It is not necessary that every if statement should have an else block as C supports simple if statements. After the first test expression or the first if branch, the programmer can have as many else-if branches as he wants depending on the expressions that have to be tested.

Example:

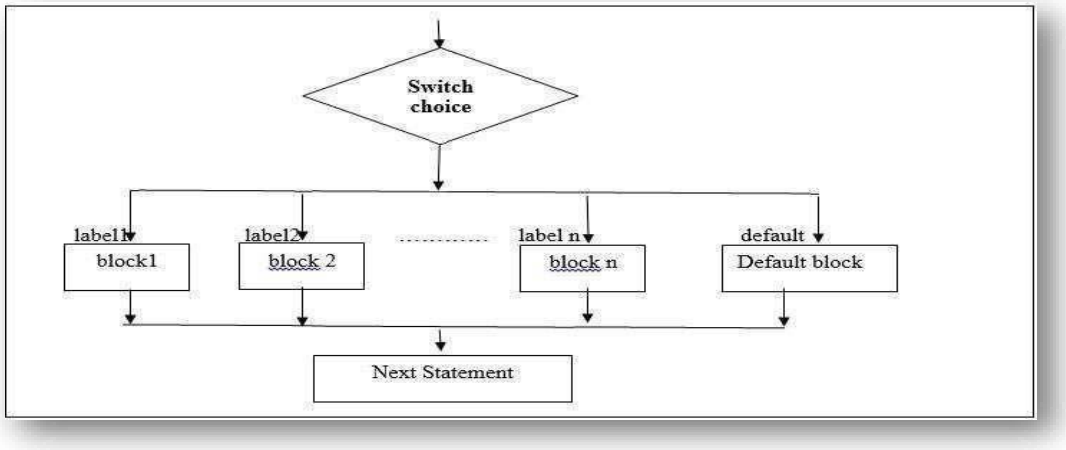
```
#include<stdio.h>
      void main( )
        {
        int a=20, b=5,c=3;
        if((a>b) && (a>c))
        {
        printf("A is greater\n");
        }
       else if((b>a) && (b>c))
        {
       printf("B is greater\n");
        }
        else if((c>a) && (c>b))
        {
        printf("C is greater\n");
        }
        else
        {
        printf("all are equal");
        }
      }
```

Switch case statement:

- C language provides a multi-way decision statement so that complex else-if statements can be easily replaced by it. C language's multi-way decision statement is called **switch**.
- General **syntax** of switch statement is as follows:

```
switch(choice)
{
                case label1:    block1;
                                break; case
                 label2: block2;
                                break;
                case label3:    block-3;
                                break;
                default:        default-block;
                                break;
}
```

- Here *switch, case, break and default* are built-in C language words.
- If the choice matches to label1 then block1 will be executed else if it evaluates to label 2 then block2will be executed and so on.
- If choice does not matches with any case labels, then default block will be executed.



The choice is an integer expression or characters.

- The label1, label2, label3,….are constants or constant expression evaluate to integerconstants.
- Each of these case labels should be unique within the switch statement. block1, block2, block3,are statement lists and may contain zero or more statements.
- There is no need to put braces around these blocks. Note that case labels end with colon(:).
- Break statement at the end of each block signals end of a particular case andcauses an exit from switch statement.
- The default is an optional case when present, it will execute if the value of the choicedoes not match with any of the case labels

Example:

Label → Number	Label → Character
#include<stdio.> #include<stdlib.h >void main( ) { int ch,a,b,res; float div; printf("Enter two numbers:\n"); scanf("%d%d",&a,&b); printf("1.Addition\n 2.Subtraction\n 3.Multiplication\n 4.Division\n 5.Remainder\n"); printf("Enter your choice:\n"); scanf("%d",&ch); switch(ch) { case 1: res=a+b; break; case 2: res=a-b; break; case 3: res=a*b; break; case 4:div=(float)a/b; break; case 5: res=a%b; break; default: printf("Wrong choice!!\n"); } printf("Result=%d\n",res); } }	#include<stdio.> #include<stdlib.h >void main( ) { int a,b,res; char ch; float div; printf("Enter two numbers:\n"); scanf("%d%d",&a,&b); printf("a.Addition\n b.Subtraction\n c.Multiplication\n d.Division\n e.Remainder\n"); printf("Enter your choice:\n"); scanf("%c",&ch); switch(ch) { case 'a': res=a+b; break; case 'b': res=a-b; break; case 'c': res=a*b; break; case 'd': div=(float)a/ b;break; case 'e' : res=a%b; break; default: printf("Wrong choice!!\n"); } printf("Result=%d\n",res); }

In this program if ch=1 case '1' gets executed and if ch=2, case '2' gets executed and so on.

Iterative statements:
Iterative statements are used to repeate the execution of a list of statements,depending on the value of an integer expression. C language supports 3 types of iterative statements also know as"looping statements".They are

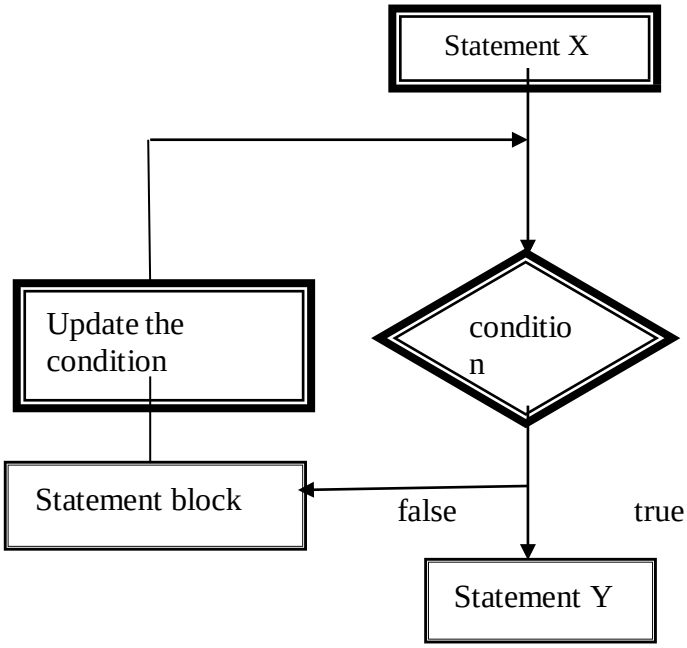
1. While
2. Do-while loop
3. For loop

While loop (pre-tested/entry controlled):

- The while loop provides a mechanism to repeate one or more statements while a particular condition is true.
- In while loop the condition is tested before any of the statements in the statement block is executed.If the condition is true , only the the statements will be executed otherwise if the condition is false,the control will jump to statement Y ,which is the immediate statement outside the while loop block.
- From the flow chart diagram ,it is clear that we nee dto constantly update the condition of the while loop.'
- The while loop will execute as long as the condition is true
- Note that if the condition is never update and the condition never becomes false then the computer will run into an infinite loop which is never desirable.
- A while loop is also referred to as a top-checking loop since the contol condition is placed as the first line of the code.If the control condition evaluates to false ,then the statements enclosed in the loop are never executed.

Syntax:

```
Statementx;
 while (condition)
 {
     statement-block;
 }

Statement Y;
```

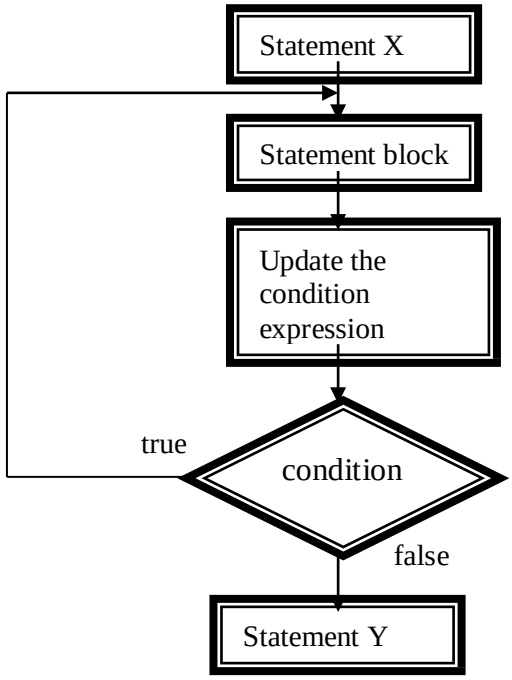


Example:

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

do-while: It is a post-test loop (also called exit controlled loop) it has two keywords *do* and *while*. The General syntax:

```
Statement X
do
  {
    statement-block;

   }while(condition);

 Statement Y
```



- The do-while loop is similar to the while loop. The only difference is that in do-while loop, the test condition is tested at the end of the loop.

- Now that the test condition is tested at the end, this clearly means that the body of the loop gets executed at least once.
- Note that the test condition is enclosed in parentheses and followed by a semicolon. The statement blocks are enclosed within curly brackets. The curly bracket is optional if there is only one statement in the body of the do-while loop.
- The major disadvantage of using a do-while loop is that it always executes at least once, even if the user enters some invalid data, the loop will execute. One complete execution of the loop takes place before the first comparison is actually done.

Example:

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

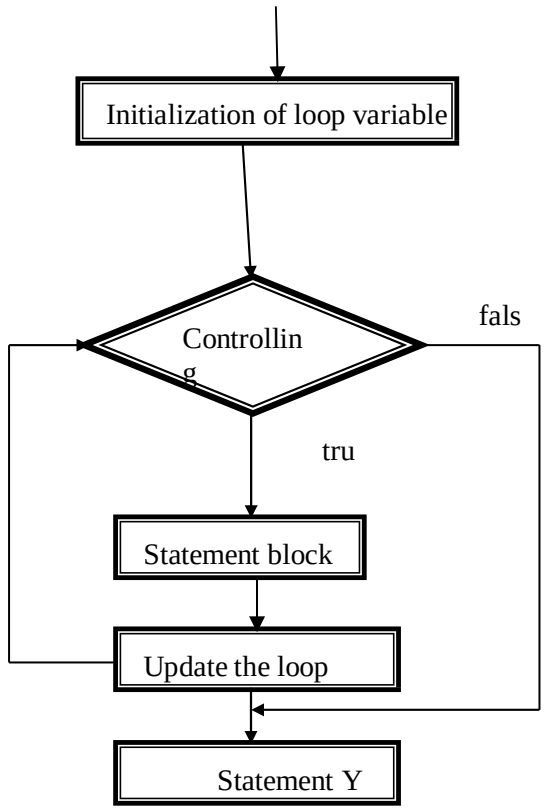
for loop:

- Similar to the while and do-while loops, the for loop provides a mechanism to repeat atask until a particular condition is true.

- For loop is usually known as determinate or definite loop because the programmer knowsexactly how many times the loop will repeat.
- The number of times the loop has to be executed can be determined mathematically by checking the logic of the loop.
- When a for loop is used, the loop variable is initialized only once. With every iteration of the loop, the value of the loop variable is updated and the condition is checked.
- If the condition is true, the statement block of the loop is executed, else the statements comprising the statement block of the for loop are skipped and the control jumps to the immediate statement following the for loop body.
- In the syntax of for loop, initialization of the loop variable allows the programmer to giveit a value.
- Second, the condition specifies that while the condition expression is TRUE the loop should continue to repeat itself.
- Every iteration of the loop must make the condition near to approachable. So, with every iteration the loop variable must be updated.

Syntax:

```
for (initialization; condition;
increment/decrement/update);
{
Statement block;
}
Statement Y;
```

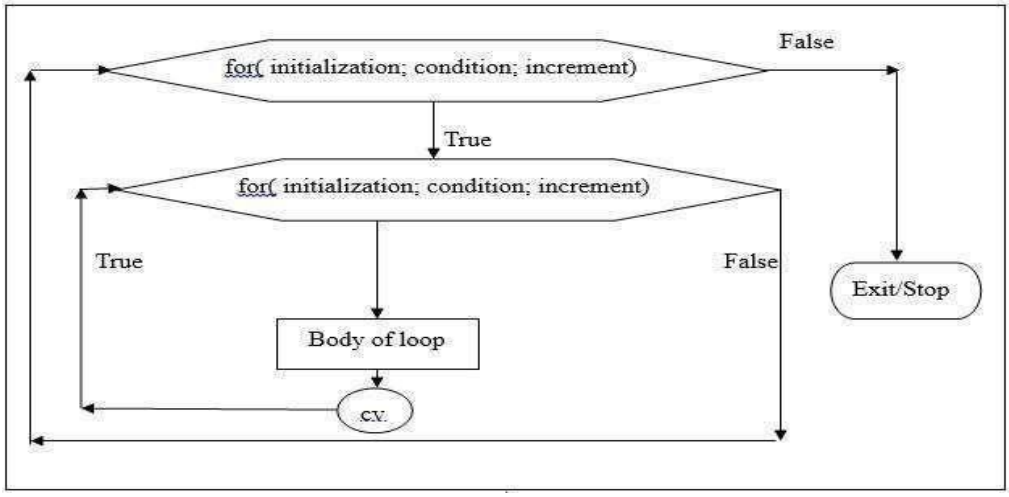


Example:

```
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```



Difference between while and do-while loop:

While	do… while
It is a pre test loop.	It is a post test loop.
It is an entry controlled loop.	It is an exit controlled loop.
The condition is at top.	The condition is at bottom.
There is no semi colon at the end of while.	The semi colon is compulsory at the end of while.
It is used when condition is important.	It is used when process is important.
Here, the body of loop gets executed if and only if condition is true.	Here, the body of loop gets executed atleast once even if condition is false.
SYNTAX, FLOWCHART, EXAMPLE (Same as in explanation)	SYNTAX, FLOWCHART, EXAMPLE (Same as in explanation)

Nested loop:

- Loop that can be placed inside other loops. Although this feature will work with any loop such as while, do-while and for but it is most commonly used with the for loop, becauseit this is easiest to control.
- A for loop can be used to control the number of times that a particular set of statements will be executed. Another outer loop could be used to control the number of times that a whole loop is repeated**.**

Syntax:

The syntax for a **nested for loop** statement in C is as follows −

```
for ( init; condition; increment ) {

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

The syntax for a **nested while loop** statement in C programming language is as follows

```
while(condition) {

   while(condition) {
      statement(s);
   }
   statement(s);
}
```

The syntax for a **nested do...while loop** statement in C programming language is as follows −

```
do {
   statement(s);

   do {
      statement(s);
   }while( condition );

}while( condition );
```

NOTE: A final note on loop nesting is that you can put any type of loop inside any other type of loop

Example:

```
for(i=0; i<2; i++)
{
        for(j=0; j<2; j++)
        {
                scanf("%d", &a[i][j])
        }
}
```

Jumping statements:

Break and continue: break and continue are unconditional control construct.

1. **Break**

- It terminates the execution of remaining iteration of loop.
- A break can appear in both switch and looping statements.
- In switch statement if the break statement is missing then every case from the matched case label till the end of the switch, including the default is executed.

Syntax	Example
while(condition) { Statements; if(condition) break; Statements; } **OR** for() { statements;	#include<stdio.h> void main( ) { int i; for(i=1; i<=5; i++) { if(i==3) break; printf("%d", i) } }

if(condition) break; statements; } **OR** do{ if (condition) break; }while(condition);	**OUTPUT** 1 2

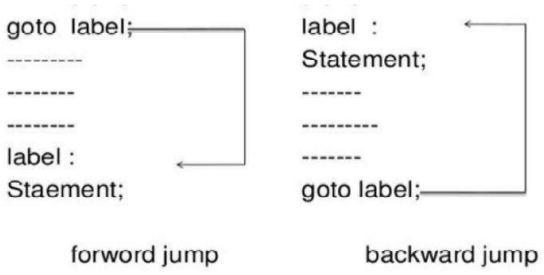
2. Continue

- It terminates only the current iteration of the loop.
- Similar to the break statement the continue statement can only appear in the body of a loop.
- When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

Syntax	Example
while(condition) { Statements; if(condition) continue; Statements; } **OR** for() { statements; if (condition) continue; statements; } **OR** do{ if (condition) continue; }while(condition);	#include<stdio.h> void main( ) { int i; for(i=1; i<=5; i++) { if(i==3) continue; printf("%d", i) } } **OUTPUT** 1 2 4 5

3. GOTO statement:

- The goto statement is used to transfer control to a specified label. However the label must reside in the same function and can appear only before one statement in the same function.
- Here, label is an identifier that specifies the place where the branch is to be made. Label can be any valid variable name that is followed by a colon (:).
- The label is placed immediately before the statement where the control has to be transferred.
- The label can be placed anywhere in the program either before or after the goto statement.
- Whenever the goto statement is encountered the control is immediately transferred to the statements following the label. Therefore goto statement breaks the normal sequential execution of the program.
- If the label is placed after the goto statement , then it is called a ***forward jump*** and in case. it is located before the goto statement, it is said to be a ***backward jump.***



Syntax	Example
goto *label;* statement; statement; *label:*	void main( ) { int a=5, b=7; goto end; a=a+1; b=b+1; *end:* printf("a=%d b=%d", a,b); } **OUTPUT:** 5,7

1. Develop a C program that takes three coefficients (a, b, and c) of a quadratic equation ; (ax2 + bx + c) as input and compute all possible roots and print them with appropriate messages.
2. Explain the working of goto statement in C with example.
3. Explain switch statement with syntax and example
4. Develop a simple calculator program in C language to do simple operations like addition, subtraction, multiplication and division. Use switch statement in your program
5. Explain with syntax, if and if-else statements in C program.
6. Explain with examples formatted input output statements in C
7. Demonstrate the functioning of Bitwise operator in C
8. Distinguish between the break and continue statement
9. Describe any 4 types of operators in C with example
10. Differentiate between type conversion and type casting in C
11. Define looping. Explain different types of looping with suitable example
12. Explain unconditional statements with example
13. Write a c program to find greatest of three numbers using ternary operator
14. List and explain unconditional branching statements with example
15. List and explain conditional branching statements with example