

DECISION MAKING, ITERATIVE AND OTHER CONTROL STATEMENTS

C program is a set of statements which are normally executed sequentially in the order. However, we have number of situations where we may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met.

Control Statements:

Control statements help the computer to execute a certain logical statements and decide whether to enable the control of the flow through a certain set of statements or not.

It is used to direct the execution of statements under certain conditions.

Types of Control Statements:

1. Decision Making or Conditional Statements
 - a) Simple if statements
 - b) If-else statements
 - c) Nested if-else statements
 - d) Else-if ladder
 - e) Switch Statements
2. Goto Statements
3. Loop Control Statements
 - a) While loop
 - b) Do-while loop
 - c) For loop

Decision Making Statements

Decision-Making Statements are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not.

These decision-making statements in programming languages decide the direction of the flow of program execution. These Decision- Making Statements are also called as **Conditional Statements**

Need of Decision Making Statement in programming

We need to make some decisions and based on these decisions we will execute the next block of code. For example,

In C if x occurs then execute y else execute z.

There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r.

This condition of C else-if is one of the many ways of importing multiple conditions.

Types of Decision making statements

1. Simple if statements
2. If-else statements
3. Nested if-else statements
4. Else-if ladder
5. Switch Statements

Simple If Statements

'if' is keyword .It is used to execute a set of statements when the logical condition is true.

Syntax:

```
If (condition)
{
Statements;
}
```

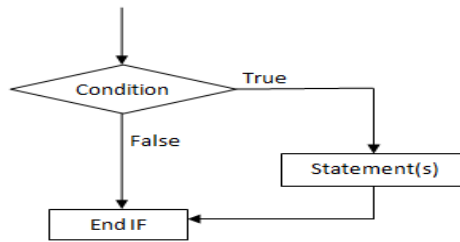


fig: Flowchart for if statement

The statement is executed only when condition is true. In case condition is false then compiler skip the line within the if block.

//program for simple if statement

```

#include<stdio.h>
int main()
{
    int n;
    printf (" enter a number:");
    scanf("%d",&n);
    If (n>10)
    {
        Printf(" number is grater");
    }
    return 0;
}
  
```

Output:

Enter a number:12
Number is greater

If else Statements

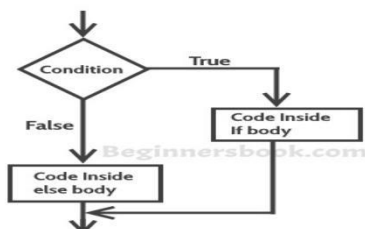
The if else statement is a decision-making statements that is used to decide whether the part of the code will be executed or not based on the specified condition.

If the given condition is true, then the code inside of the if block is executed, otherwise the code inside the else block is executed.

Syntax:

```

if (condition)
{
    Statements;
}
else
{
    Statements;
}
  
```



// program for if else

```

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

```
int a;
printf("enter a value:");
scanf("%d",&a);
if(a%2==0)
{
printf("even");
}
else
{
printf("odd");
}
return 0;
}
```

Output :1

enter a value : 5

odd

Output 2:

enter a value: 88

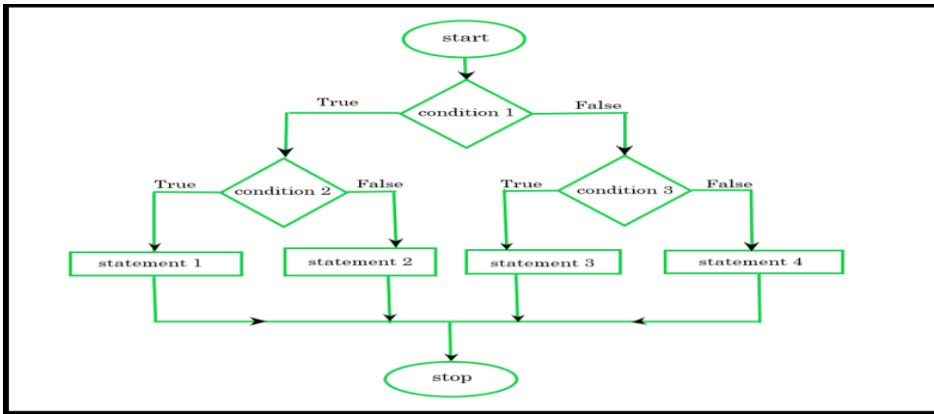
even

Nested if else statement

A nested if else statement is an if statement inside another if statement. **(or)** Either if block or else block or both contains another if statement or if else statements then it is called as nested if else statements.

Syntax:

```
if(condition 1)
{
if(condition 2)
{
Statement 1;
}
else
{
statement 2;
}
}
else
{
if(condition 3)
{
Statements 3;
}
else
{
Statements 4;
}
}
}
```



// C program for largest of 3 numbers

```

#include<stdio.h>
int main ()
{
    int a,b,c;
    printf("enter three values:");
    scanf("%d%d%d",&a,&b,&c);
    {
        if(a>b)
        {
            if(a>c)
            {
                printf("a is greater");
            }
        }
        else
        {
            printf("c is greater");
        }
    }
    else
    {
        if(b>c)
        {
            printf("b is greater");
        }
        else
        {
            printf("c is greater");
        }
    }
    return 0;
}
  
```

Output 1:

enter three values : 5 6 7

c is greater

Output 2:

enter three values: 20 15 10

a is greater

Output 3:

enter three values: 55 78 39

b is greater

Else if statements or else if ladder

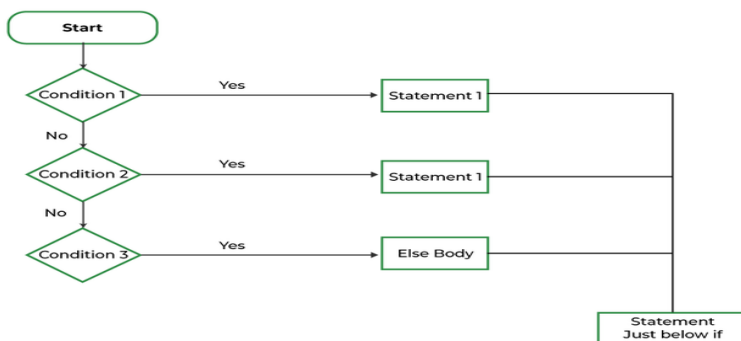
The else if ladder statement is an extension of if-else statements, it is used in the scenario where there are multiple cases to be performed for different conditions.

If a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then else if block will be executed. At last, if none of the statements is true then else block will execute.

Syntax is :-

```
if (condition)
{
    Statement1;
}
else if (condition)
{
    statement2;
}
else if (condition)
{
    statement3;
}
else
{
    statement4;
}
```

This process continues until there is no if statement in the last block. If one of the conditions is satisfied, the condition, other nested "else if" would not be executed.



// C program to find average and to print grade

```
#include<stdio.h>
int main ()
{
    int s1,s2,s3,s4,total;
    float avg;
    char grade;
    printf("Enter 4 subject marks");
    scanf("%d %d %d %d",&s1,&s2,&s3,&s4);
    total = s1+s2+s3+s4;
    avg = total/4;
    printf("Total marks :%d\n",total);
    printf("Average : %.2f\n",avg);
    if(avg>60);
    {
        printf("Grade = A\n",grade);
    }
```

```

}
elseif(avg>=50 && avg<60);
{
printf("Grade = B\n",grade);
}
elseif(avg>=35 && avg>50);
{
printf("Grade = C\n",grade);
}
else
{
printf("Grade = D\n",grade);
}
return 0;
}

```

Output:

Enter 4 subject marks: 10 20 40 50

Total marks: 120

Avg = 30.00

Grade = d

Switch Statement

Switch Statement tests the value of variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed.

Rules for Switch Statement:

1. Switch expression must be an integer or character type.
 2. The case value must be an integer or a character constant.
 3. The case value can be used only inside the switch statements.
 4. The break keyword in each case indicates the end of a particular case.
- If there is no break statements found in case, all the cases will be executed after the matched case.

5. Case labels always ended with (:) :

Syntax:

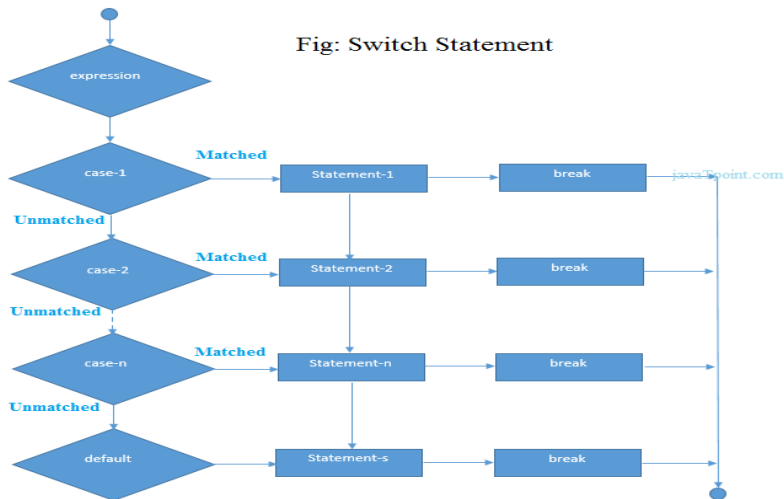
Switch (expression)

```

{
case value 1:
block 1;
break;
case value 2:
block 2;
break;
case value 3:
block 3;
break;
:
:
case value n:
block n;
break;
default:
default block;
break;
}

```

Fig: Switch Statement



// Program for switch statement

```
#include<stdio.h>
int main ()
{
    int num;
    printf("Enter a number");
    scanf("%d",&num);
    switch(num)
    {
        case 7:
            printf("value is 7);
            break;
        case8:
            printf("value is 8);
            break;
        case 9:
            printf("value is 9);
            break;
        default:
            printf("out of range");
            break;
    }
    return 0;
}
```

Output :

Enter a number : 8
value is 8

// Program without break statement

```
#include<stdio.h>
int main ()
{
    int num = 0;
    printf("Enter a number :");
    scanf("%d",&num);
    switch (num)
    {
        case 10:
```

```

printf("number is equal to 10\n");
case 20:
printf("number is equal to 20\n");
case 30:
printf("number is equal to 30\n");
default:
printf(" number is out of range");
}
return 0;
}

```

Output:

```

Enter a number: 10
number is equal to 10
number is equal to 20
number is equal to 30

```

// C program to perform arithmetic operations using Switch statements

```

#include<stdio.h>
int main ()
{
int a,b,c;
char oper;
printf("Enter operator:");
scanf("%c",oper);
printf("enter a,b values :");
scanf("%d%d",&a,&b);
switch oper;
{
case '+':
c = a+b;
printf("sum = %d",c);
break;
case '-':
c = a-b;
printf("diff = %d",c);
break;
case '*':
c = a*b;
printf("multi = %d",c);
break;
case '/':
c= a/b;
printf("div = %d",c);
break;
case '%':
c = a%b;
printf("mod =%d",c);
break;
default :
printf("invalid operator. Try again");
}
return 0;
}

```


State the importance of break statement with switch and illustrate

The **break** statement in a **switch** statement is crucial for controlling the flow of execution. When a **break** statement is encountered inside a **case**, it terminates the **switch** statement and transfers the control outside the **switch** block. Without **break** statements, the control would “fall through” to subsequent cases, leading to unintended behavior.

Here’s an illustration to demonstrate the importance of the **break** statement in a **switch** statement :

```
#include<stdio.h>
int main ()
{
    int choice;
    printf("Enter a number between 1 and 3:");
    scanf("%d",&choice);
    switch (choice)
    {
        case 1:
            printf("You choose option 1\n");
            break;
        case 2:
            printf("You choose option 2\n");
            break;
        case 3:
            printf("You choose option 3\n");
            break;
        default:
            printf("invalid choice\n");
    }
    return 0;
}
```

In this example, the user is prompted to enter a number between 1 and 3. The switch statement then evaluates the entered value. Without the **break** statements, if there the user enters 1, all the subsequent cases would be executed, leading to unexpected behavior.

For instance, if the user enters 1 without **break** statements:

Enter a number between 1 and 3 : 1

You chose option 1.

You chose option 2.

You chose option 3.

By using **break** statements after each **case**, the program correctly prints the corresponding message and exits the **switch** statements:

Enter a number between 1 and 3 :

You chose option 1.

The break statement ensures that only the code associated with the matched case is executed, preventing fall – through to subsequent cases.

Compare Conditional operator with if-else statement

Conditional Operator (? :) :

Syntax : Condition ? expression _if_true : expression _if_false;

Usage :

- The conditional operator is a ternary operator that evaluates a condition and returns one of two expressions based on whether the condition is true or false.
- It is concise and is often used for simple conditional assignments in a single line.

Int x = 5;

int y = (x > 0) ? 10 : -10; // If x > 0, y = 10; otherwise, y = -10.

If-else Statement:

Syntax: if (condition)

```
{  
    // code to be executed if the condition is true  
}  
else  
{  
    // code to be executed if the condition is false  
}
```

Usage :

- The if else statement is a control flow statement that allows you to execute different block of code based on the evaluation of a condition.
- It is more flexible than the conditional operator and can handle more complex conditions and multiple statements.

Example:

```
int x = 5;  
int y;  
if (x > 0) {  
    y = 10;  
} else {  
    y = -10;  
}
```

Comparison :

Comparison	Conditional Operator	If-else statement
Readability	The conditional operator is concise and might be more suitable for simple assignments.	The if else statement is generally more readable and can handle more complex scenarios with multiple statements.
Flexibility	The conditional operator is more suitable for simple conditions and assignments.	The if else statement provides more flexibility for handling complex conditions and executing multiple statements in each block.
Code Structure	The conditional operator can make the code more compact but might sacrifice readability for complex conditions.	The if else statements allows for a more natural code structure when dealing with more extensive conditional logic.

Compare if-else with switch statements

If-else Statement:

Syntax:

```
if (condition) {  
    // code to be executed if the condition is true  
} else {  
    // code to be executed if the condition is false  
}  
  
// code to be executed if the condition is false  
}
```

Usage:

- The if else statement is a control flow statement that allows you to execute different block of code based on the evaluation of a condition.
- It is flexible and can handle a wide range of conditions and expressions.

Example:

```

int x = 5;
if (x > 0) {
    // code when x is positive
} else {
    // code when x is non-positive
}

```

Switch Statement:

Syntax:

```

switch (expression) {
    case constant1:
        // code to be executed if expression equals constant1
        break;
    case constant2:
        // code to be executed if expression equals constant2
        break;
    // ...
    default:
        // code to be executed if expression doesn't match any case
}

```

Usage :

- The switch statement is used to perform multiway branching based on the value of a expression.
- It is often used when there are multiple possible values for the expression, and different actions need to be taken for each value.

Example:

```

int day = 3;
switch (day) {
    case 1:
        // code for Monday
        break;
    case 2:
        // code for Tuesday
        break;
    // ...
    default:
        // code for other days
}

```

Comparison:

Comparison	If-else Statements	Switch Statements
Condition	The if –else statement is more general-purpose and can handle complex conditions involving relational operators, logical operators, and expressions.	The switch statement is specifically designed for multiway branching based on the equality of a value with different constants.
Expression	The if-else statement can evaluate any Boolean expression.	The switch statement evaluates an integral expression or character constant.
Readability	The if-else statements is generally more suitable for handling a variety of conditions and scenarios.	The switch statement can be more readable and concise.
Fall -through	The if-else statement doesn't have fall-through behavior, making it easier to understand the flow of control.	In a switch statement, if a break statement is omitted, control can “fall through” to subsequent cases.

Use Cases	Use the if –else statement when dealing with conditions involving expressions, logical operators, and when there are relatively few cases.	Use Switch statements when the control flow depends on the equality of an expression with different constant values.
-----------	--	--

Defining looping or Iteration

Sequence of statements is executed until some conditions for termination of the loop are satisfied.

A program loop consists of two segments:

- a) Body of the loop
- b) Control statements

Control statements test certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

1. The while statement
2. The do-while statement
3. The for statement

A looping process,in general include four steps:

1. Setting and initialization of a condition variable.
2. Execution of statements in the loop.
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

Initialization : it is assignment statement that is used to set the loop control variable.

Condition :It is relational expression that determining when the loop will exit.

Increment/ Decrement:defines how the loop control variable will change each time loop is repeated.

While Statement

The while is an entry- controlled loop statement.

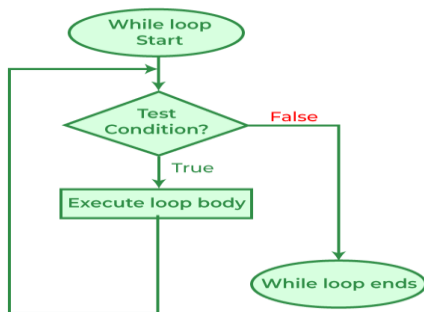
The test condition is evaluated and if the condition is true,then the body of the loop is executed after execution of the body,the test condition is once again evaluated if it is true.

This process of repeated execution of the body continues until the test condition becomes ‘false’ and control is transferred out of the loop.

Syntax:

```
While(test condition)
{
    Body of loop;
}
```

Flow chart:



Example :

```
i=0;
While(i<=5)
{
    printf("%d\n",i);
```

```
i++;
```

```
}
```

Output:

1

2

3

4

5

#C program to check sum of digits of a given number

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

```
int main()
```

```
{
```

```
int n,sum=0,r;
```

```
printf("Enter n value);
```

```
scanf("%d",&n);
```

```
while(n!=0)
```

```
{
```

```
r=n%10;
```

```
sum=sum+r;
```

```
n=n/10;
```

```
}
```

```
printf("sum of digits:%d",sum);
```

```
return 0;
```

```
}
```

Output:

Enter n value: 456

sum of digits :15

Do-while Statement

The do-while constructor provides an exit-control loop and therefore the body of the loop is always executed at least once.

The program proceeds to evaluate the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated.

Syntax:

```
do
```

```
{
```

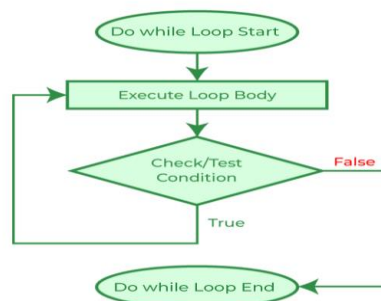
```
body of loop;
```

```
}
```

```
while(test-condition)
```

Flow chart:

Example:



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

```
int main ()
```

```
{
```

```

int k;
k=1;
do
{
printf("%d",k);
k++;
}
while(i<=5);
return 0;
}

```

Output:

1 2 3 4 5

#C program to check the given number is Armstrong or not

```

#include<stdio.h>
int main()
{
int n,r,sum=0,temp;
printf("Enter n value");
scanf("%d",&n);
n=temp;
do
{
r=r%10;
sum=sum+(r*r*r);
n=n%10;
}
while(n!=0);
if(sum==temp)
{
printf("Armstrong");
}
else
{
printf("Not Armstrong");
}
return 0;
}

```

Output:1

Enter n value 153

Armstrong

Output :2

Enter n value 123

Not Armstrong

For loop :

The for loop is another entry-controlled loop that provides a more concise loop control structure.

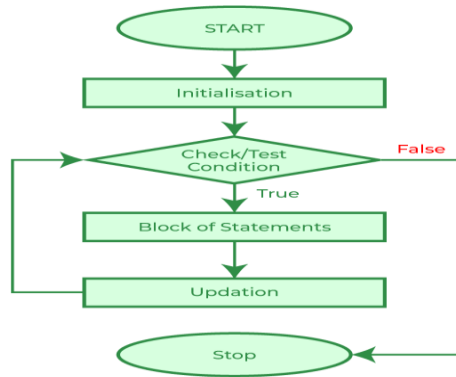
Syntax:

```

for(initialization;test condition;increment/decrement);
{
body of loop;
}

```

Flow chart:



Example:

```

#include<stdio.h>
int main()
{
  int k=1;
  for(k=1;k<=5;k++)
  {
    printf("%d",k);
  }
}
  
```

Output:

1 2 3 4 5

C program to print fibonacci numbers

Fabonacci series is a sequence of numbers where each number is the sum of two preceding ones.

0 ,1,1,2,3,5,8,13,21,34,.....

```

#include<stdio.h>
int main()
{
  int l,n,first=0,second=1,next;
  printf("Enter the number of terms:");
  scanf("%d",&n)
  for(l=0;l<n;l++)
  {
    printf("%d , ",first);
    first=second;
    next=first+second;
    second=next;
  }
  return 0;
}
  
```

Output:

Enter the number of terms: 9

0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 ,

Compare Different loops:

While loop	For loop	Do-while loop
Condition will be checked before entering into the body of the	Condition will be checked before entering into the body of the	Condition will be checked at the end of the loop

loop	loop	
Loop variable is incremented in the body of the loop	Increment is in for command	Loop variable is incremented in the body of the loop
No semicolon after the condition in the syntax	No semicolon after the condition in the syntax	There is a semicolon after the condition in the syntax
The control will never enter into the loop if the condition is not true for the first time	The control will never enter into the loop if the condition is not true for the first time	The control will enter a loop even if the condition is not true for the first time
Syntax: while(condition) { Statements; }	Syntax: for(initialization, condition, updating) { Statements; }	Syntax: do { Statements; } While(condition);

Nesting loops

Nested loop is a loop inside another loop. This means that there is an inner and outer loop.

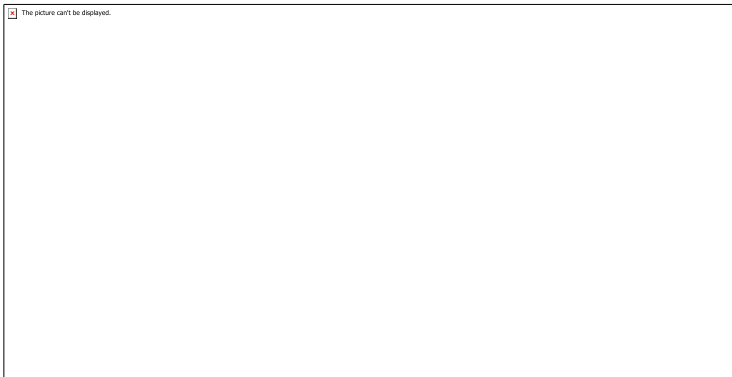
The inner loop is completely contained within the body of the outer loop.

Each time outer loop runs, the inner loop runs several times.

Syntax:

```
for(initialization;condition;updatation)
{
for(initialization;condition;updatiion)
{
Statements of inner loop;
}
Statements of outer loop;
}
```

Flow chart:



C program to print the given pattern

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



```

int main ()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        for (j=0;j<i;j++)
        {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}

```

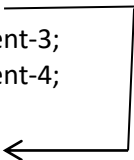
Break and Continue Statements

Break: it is a keyword ,used to terminate the loop or exit from the block.It is used with for ,while, do-while
Syntax:

```

{
    Statement-1;
    Statement-2;
    break;
    Statement-3;
    Statement-4;
}
exit

```



When break is occurred it exit the loop

Example:

```

#include<stdio.h>
int main()
{
    int i;
    for(i=1;i<=5;i++)
    {
        printf("%d",i);
        if(i==3)
        break;
    }
    return 0;
}

```

Output:

12345

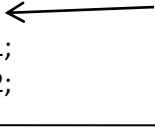
Continue: It is a keyword,used for continue the next statement or iteration of the loop.Used with for,while and do-while.

Syntax:

```

{
    Statement-1;
    Statement-2;
    continue;
    Statement-3;
    Statement-4;
}

```



When continue occurs ,it skips the current iteration and again goes to starting of loop.

Example:

```
#include<stdio.h>
int main ()
{
int l;
for(1=1;l<=5;l++)
{
if(l==2)
continue;
printf("%d",l);
}
}
```

Output:
1345

Goto statements

It is used to alter the normal-sequence of program execution by transferring the control to some other part of the program.

Syntax:

1.forward jump

```
goto lable;
statements;

lable:
statements; ←
```

2.Backword jump

```
lable:
Statements; ←
goto lable;
```

Example:

```
#include<stdio.h>
int main ()
{
printf("hello");
goto l1;
printf("how are you");
l1:
printf("hi");
}
```

Output:
hello
Hi

Differentiate Break and Continue

Break	continue
The break statement is used to exit from the loop constructs	The continue statements is not used to exit from the loop constructs
The break statement is usually used with the switch statement,and it can also use it within the while loop,do-while,for	The continue statement is not used with the switch statement but it can be used within the while loop,do-while,for
When a break statement is encountered then the control is exited from the loop construct immediately	Continue statement is encountered then the control automatically passed from the beginning of the loop statements

Syntax : break;	Syntax: continue;
Break statements uses switch and label statements	It doesn't use switch and label statements
Leftover iterations are not executed after the break statements	Leftover iterations can be executed even if continue keyword appeared in loop

Structured Programming

Structured programming is a method to design and develop programs. C is called a structured programming language because C language divides the whole program into smaller modules or procedures or functions. Each function handles a specific task.

Advantages:

1. The program is divided into smaller modules, which each module handles the individual tasks.
2. A structured program is simple and easy to understand its logic.
3. It takes less time to develop the modules.
4. All modules can be combined to form a complete program.
5. A structured programming helps to ensure well designed programs that are easy to write, read, debug and maintain compared to those that are unstructured.

In C language there are three basic control structures:

1. Sequence (straight line) structure
2. Selection (branching) structure
3. Repetition (looping) structure

Sequence.

Lines or blocks of code are written and executed in sequential order.

Example:

```
x = 5
y = 11
z = x + y
WriteLine(z)
```

Repetition.

Repeat a block of code (Action) while a condition is true. There is no limit to the number of times that the block can be executed.

While condition

 action

End While

Example:

```
x = 2
While x < 100
    WriteLine(x)
    x = x * x
```

End

Selection.

Execute a block of code (Action) if a condition is true. The block of code is executed at most once.

If condition Then

 action

End If

Example:

```
x = ReadLine()
If x Mod 2 = 0
    WriteLine("The number is even.")
End If
```

