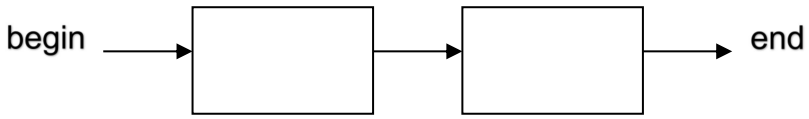
Program Control Structure

A control structure is a block of programming that analyzes variables and chooses a direction in which to go based on given parameters. The term flow control details the direction the program takes (which way program control "flows").

Sequential

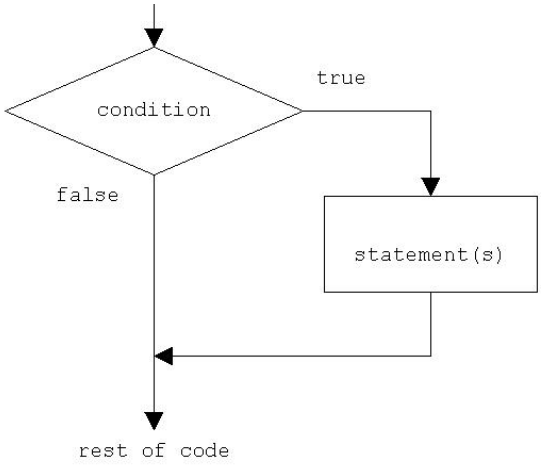
It is the set of program instructions which follow one another and are to be executed unconditionally (not dependent on any program conditions). Instructions are put in a predefined sequence (just like a queue in a cinema hall) and the next instruction is executed by CPU only after the execution of the previous instruction (C never comes before B).



Selection

It is the set of instructions which are to be executed conditionally i.e. they are executed based on a condition that can be either true or false. Commonly used logic for selection are if condition, if else condition, if else if condition, nested if else condition and switch case condition.

- **If condition** - used in case the given problem has only one condition and only one action. Considering either true or false part, if the given condition is true then the statement will be executed. Otherwise, the control exits from the condition.



- **If else condition** - used if the problem has one condition but two alternative actions. Here, if the condition is true, statement 1 will be executed; otherwise, statement 2 will be executed.

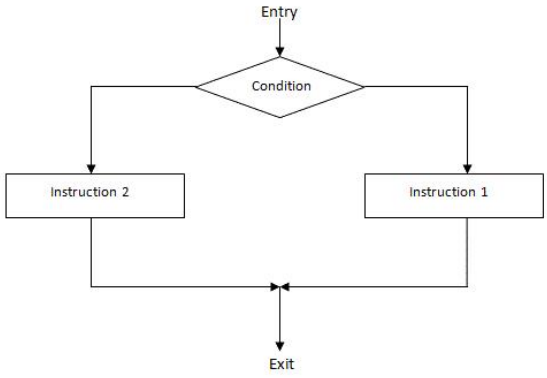


Figure: if else condition

- **If-else if condition** – use this condition if the given problem has more than one interrelated conditions with their respective actions. Here, on a check, if condition 1 is true then, statement 1 is executed. Otherwise, condition 2 is checked and if it is true, statement 2 is executed and so on for next conditions. If all conditions are false, then the last statement will be executed.

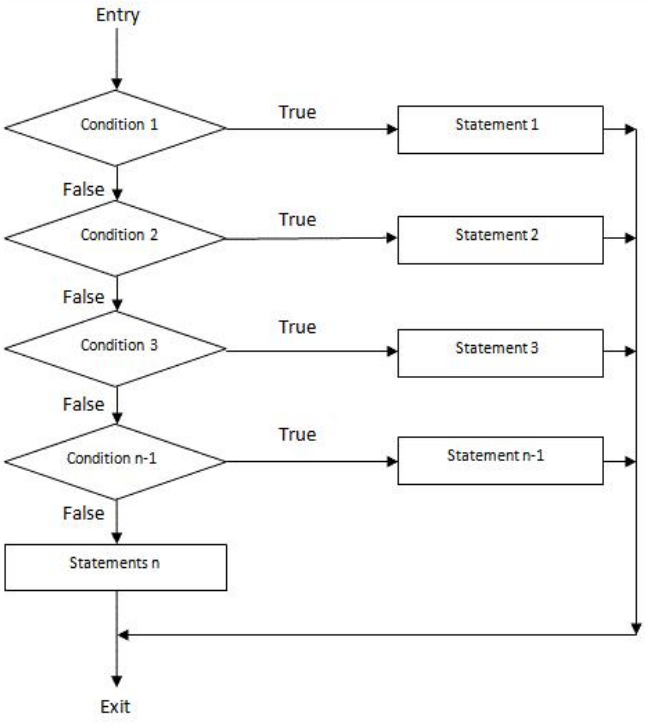


Figure: if else if condition

- **Nested if else condition** - an entire if-else statement which is written within the body of if part or else part of another if else statement. This condition is used when a condition is to be checked that is inside another condition at a time in the same program, to make a decision.

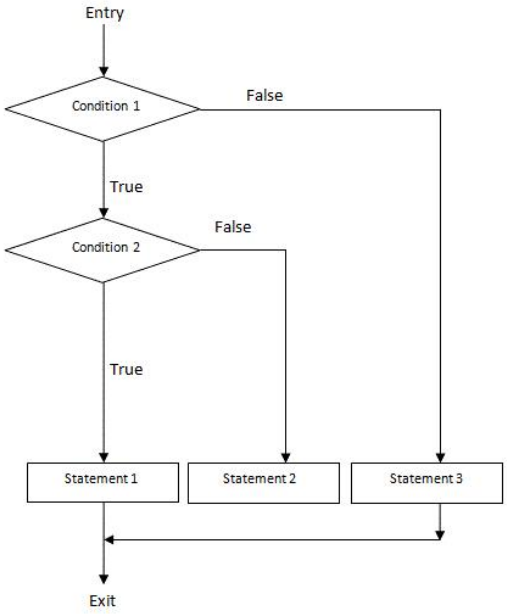


Figure: nested if else condition

- **Switch case condition** - If the given problem has one condition and respective more than two actions, then in this type of case scenario, you can use Switch case condition. It is the multiple branching statements which checks the value of the variable to the case value and then, the statements that are associated with it will be executed. If any expression does not match any of the case value, then the default statement will be executed.

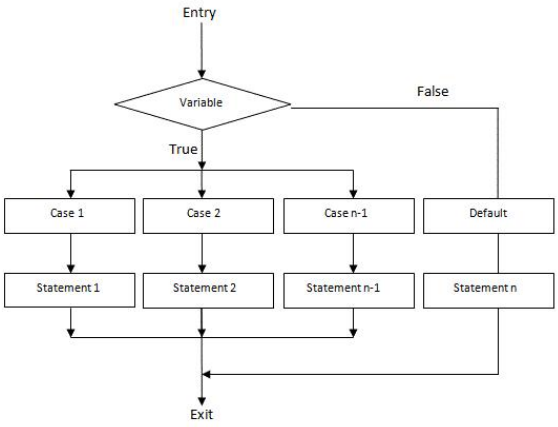


Figure: switch case condition

Repetition

These are the computer instructions which are to be performed repeatedly and conditionally i.e. loop statements are driven by the loop condition. Commonly used logic for iteration are while loop, do while loop and for a loop.

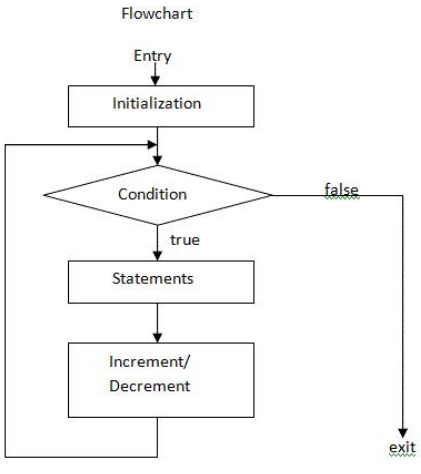


figure: while loop Iteration

- **While loop** - In this loop, first, the condition is checked by the

computer and if the condition turns out to be true, then the statement inside the loop is executed. This process is repeated and the value of increment and decrement operator is always changing. When the condition is false, the loop stops.

- **Do while loop** - In this loop, first, the computer checks the initial value; second executes the statements inside the loop and finally, checks the condition. The process is repeated for next pass, if the condition is true. Otherwise, the loop stops. If the condition is initially false, it will execute for at least one time.

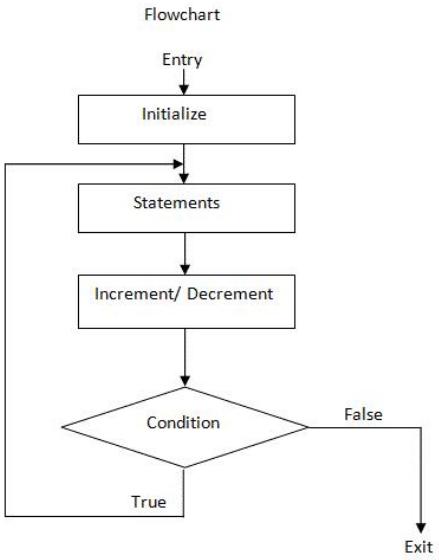


figure: do while loop

- **For loop** - It is the most commonly used loop. It consists of 3 expressions; initialization, condition and counter, which are defined within a statement.

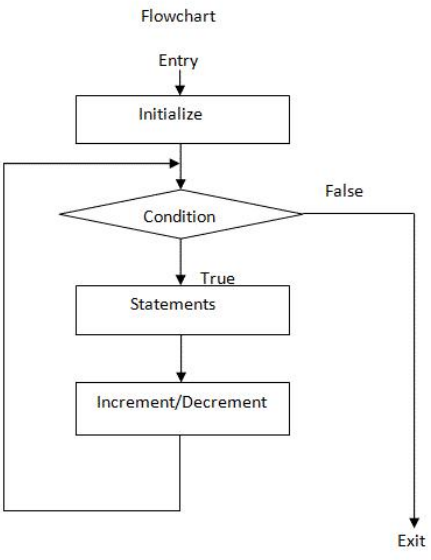


figure: for loop

C# Method Declaration

Method is the building block of object-oriented programming. It combines related code together and makes program easier.

In C# method declaration, you can declare method by following way:

```
<Access Specifier> <Return Type> <Method Name>(Parameter list)
{
   Body
}
```

Example

```
public void add()
{
    Body
}
```

In the example:

- the **public** is an **access specifier**
- **void** is a **return** data type that **return nothing**
- **add( )** is a method name
- There is no **parameter** define in **add( )** method.

If the function **returns integer** value, then you can define function as follow:

```
public int add()
{
    Body
    return integer_variable/value;
}
```

If the function is **returning string** value, then you can define function as follow:

```
public string printname()
{
```

```
.        Body
.        return string_variable/value;
.    }
```

You must remember:

- Whenever use return data type with method, must return value using **return keyword** from body
- If you don't want to return any value, then you can use **void** data type.

Sample Program

```
.    namespace Declaring_Method
.    {
.      class Program
.       {
.         string name, city;
.         int age;
.
.         // Creating method for accepting details
.         public void acceptdetails()
.          {
.            Console.Write("\nEnter your name:\t");
.            name = Console.ReadLine();
.
.            Console.Write("\nEnter Your City:\t");
.            city = Console.ReadLine();
.
.            Console.Write("\nEnter your age:\t\t");
.            age = Convert.ToInt32(Console.ReadLine());
.          }
.
.         // Creating method for printing details
.         public void printdetails()
.          {
.            Console.Write("\n\n==================");
```

```
.           Console.Write("\nName:\t" + name);
.           Console.Write("\nCity:\t" + city);
.           Console.Write("\nAge:\t" + age);
.           Console.Write("\n====================\n");
.         }
.
.       static void Main(string[] args)
.         {
.           Program p = new Program();
.           p.acceptdetails();
.           p.printdetails();
.           Console.ReadLine();
.         }
.     }
. }
```

Output

```
Enter your name:     Steven Clark

Enter Your City:     California

Enter your age:      47



==================================
Name:   Steven Clark
City:    California
Age:    47
==================================
```

GUIDELINE WHILE CREATING METHOD

- You can define multiple functions within a class.
- If you are using return data type instead of void, then must return appropriate value with return keyword.

1. Calling Method Or Function (C#)

After creating function, you need to call it in Main() method to execute.

In order to call method, you need to **create object** of containing class, then followed by **dot(.)** operator you can call the method.

If method is **static**, then there is **no need** to create object and you can directly call it followed by class name.

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Declaring_Method
{
    class Program
    {
        string name, city;
        int age;

        public void acceptdetails()
        {
            Console.Write("\nEnter your name:\t");
            name = Console.ReadLine();

            Console.Write("\nEnter Your City:\t");
            city = Console.ReadLine();

            Console.Write("\nEnter your age:\t\t");
            age = Convert.ToInt32(Console.ReadLine());
        }

        public void printdetails()
        {
            Console.Write("\n\n====================");
            Console.Write("\nName:\t" + name);
            Console.Write("\nCity:\t" + city);
            Console.Write("\nAge:\t" + age);
            Console.Write("\n====================\n");
        }

        static void Main(string[] args)
        {
            //creating object of class Program
            Program p = new Program();
            p.acceptdetails(); // Calling method
            p.printdetails(); // Calling method
            Console.ReadLine();
        }
    }
}
```

Output

```
Enter your name:     Steven Clark


Enter Your City:      California


Enter your age:       47


==================================

Name:   Steven Clark

City:     California

Age:     47

==================================__
```

If method is declared **static,** then you can directly call the method without creating object of containing class.

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Calling_method
{
    class print
    {
        public static void printname()
        {
            Console.WriteLine("Steven Clark");
            Console.ReadLine();
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // call directly static method with class name
            print.printname();
        }
    }
}
```

Output

```
Steven Clark__
```

2. C# Static Method and Variables

Whenever you write a function or declare a variable, it **doesn't** create an instance in a memory until you create an object of the class.

But if you declare any function or variable with a **static** modifier, it directly creates an instance in a memory and acts globally. The static modifier doesn't reference any object.

How to: It is very easy to create static modifier with variables, functions and classes. Just put **static** keyword before the return data type of method.

```
namespace Static_var_and_fun
{
    class number
    {
        // Create static variable
        public static int num;
        //Create static method
        public static void power()
        {
            Console.WriteLine("Power of {0} = {1}", num, num * num);
            Console.ReadLine();
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter a number\t");
            number.num = Convert.ToInt32(Console.ReadLine());
            number.power();
        }
    }
}
```

Output

```
Enter a number  7

Power of 7 = 49 __
```

3. C# Main Method

In C# programming the **Main( )** method is where program starts execution.

It is the main entry point of program that executes all the objects and invokes method to execute. It is the main place where a program starts the execution and end.

There can be only one Main ( ) method in C#.

It must be inside a **class**, and must be declared with **static** modifier.

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Main_Method
{
    class check
    {
        float num, percent;
        public void accept()
        {
            Console.Write("\nEnter your marks. (Total Mark = 850):\t");
            num = float.Parse(Console.ReadLine());
        }
        public void print()
        {
            percent = (float)num / 850 * 100;
            if (percent < 35)
            {
                Console.WriteLine("Sorry!!! You are fail. your marks is " + percent);
            }
            else if (percent > 35 && percent < 50)
            {
                Console.WriteLine("You got grade D and your percentage marks is " + perce
nt);
            }
            else if (percent > 50 && percent < 60)
            {
                Console.WriteLine("You got grade C and you percentage marks is " + percen
t);
```

```
            }
            else if (percent > 60 && percent < 75)
            {
                Console.WriteLine("You got grade B and your percentage marks is " + perce
nt);
            }
            else
            {
                Console.WriteLine("You got grade A and your percentage marks is " + perce
nt);
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Starting execution
            // Creating object of class check
            check chk = new check();
            chk.accept(); //Invoking accept method
            chk.print(); //Invoking print method
            Console.ReadLine();
        }
    }
}
```

Output

```
Enter your marks. < Total Mark = 850 >:   435


You got grade C and you percentage marks is 51.17647
```

WHY THE MAIN( ) METHOD IS ALWAYS DECLARED WITH STATIC?

- The Main( ) method in C# is always declared with static because it can't be called in another method of function.
- The Main( ) method instantiates other objects and variables but there is no any method there that can instantiate the main method in C#.

4. C# Access Specifiers

What is Access Specifiers in C#?

Access Specifiers defines the scope of a class member. A **class member** can be **variable** or **function**.

List of Access Specifiers

- **Public Access Specifiers**
- **Private Access Specifiers**
- **Protected Access Specifiers**

4.1 Public Access Specifiers (C#)

The class member, that is defined as a **public** can be **accessed** by other class members that are initialized **outside** the **class**.

A **public** member can be **accessed** from anywhere even **outside** the **namespace**.

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Public_Access_Specifiers
{
    class access
```

```
    {
        // String Variable declared as public
        public string name;
        // Public method
        public void print()
        {
            Console.WriteLine("\nMy name is " + name);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            access ac = new access();
            Console.Write("Enter your name:\t");
            // Accepting value in public variable that is outside the class
            ac.name = Console.ReadLine();
            ac.print();

            Console.ReadLine();
        }
    }
}
```

Output

```
Enter your name:    Steven Clark

My name is Steven Clark
```

4.2 Private Access Specifiers (C#)

The **private** access specifiers **restrict** the member variable or function to be called **outside** of the parent class.

A **private** function or variable **cannot be called outside** of the same **class**.

It **hides** its member variable and method from other class and methods. However, you can store or retrieve the value from private access modifiers using **get/set property**.

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Private_Access_Specifiers
{
    class access
    {
        // String Variable declared as private
        private string name;
        public void print() // public method
        {
            Console.WriteLine("\nMy name is " + name);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            access ac = new access();
            Console.Write("Enter your name:\t");
            // raise error because of its protection level
            ac.name = Console.ReadLine();
```

```
            ac.print();
            Console.ReadLine();
        }
    }
}
```

Output

```
Error 1: Private_Access_Specifiers.access.name' is inaccessible due to its protection leve
l __
```

In the above example, you cannot call name variable outside the class because it is declared as private.

4.3 Protected Access Specifiers C#

The **protected** access specifier hides its member variables and functions from other classes and objects.

This type of variable or function can only be **accessed** in **child** class. It becomes very important while implementing **inheritance**.

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Protected_Specifier
{
    class access
    {
        // String Variable declared as protected
        protected string name;
        public void print()
        {
```

```
            Console.WriteLine("\nMy name is " + name);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            access ac = new access();
            Console.Write("Enter your name:\t");
            // raise error because of its protection level
            ac.name = Console.ReadLine();
            ac.print();
            Console.ReadLine();
        }
    }
}
```

Output

```
'Protected_Specifier.access.name' is inaccessible due to its protection level.
```

This is because; the protected member can only be accessed within its child class.

You can use protected access specifiers as follow:

Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Protected_Specifier
{
    class access
```

```
.     {
.         // String Variable declared as protected
.         protected string name;
.         public void print()
.         {
.             Console.WriteLine("\nMy name is " + name);
.         }
.     }
.
.     class Program : access // Inherit access class
.     {
.         static void Main(string[] args)
.         {
.             Program p = new Program();
.             Console.Write("Enter your name:\t");
.             p.name = Console.ReadLine(); // No Error!!
.             p.print();
.             Console.ReadLine();
.         }
.     }
. }
```

Output

```
Enter your name:    Steven Clark

My name is Steven Clark
```

5. Get/Set Modifier (C#)

The **get/set accessor** or **modifier** mostly used **for storing and retrieving** the **value** from the **private field**.

The **get accessor** must **return** a **value** of property type where **set accessor returns void**.

The **set accessor** uses an implicit parameter called **value**.

In simple word, the **get method** used for **retrieving the value** from private field whereas **set method** used for **storing the value** in private variables.

Sample Program

```
. using System;
. using System.Collections.Generic;
. using System.Linq;
. using System.Text;
.
. namespace Get_Set
. {
.     class access
.     {
.         // String Variable declared as private
.         private static string name;
.         public void print()
.         {
.             Console.WriteLine("\nMy name is " + name);
.         }
.
.         public string Name //Creating Name property
.         {
.             get //get method for returning value
.             {
.                 return name;
.             }
.             set // set method for storing value in name field.
.             {
.                 name = value;
.             }
.         }
```

```
    }

    class Program
    {
        static void Main(string[] args)
        {
            access ac = new access();
            Console.Write("Enter your name:\t");
            // Accepting value via Name property
            ac.Name = Console.ReadLine();
            ac.print();
            Console.ReadLine();
        }
    }
}
```

Output

```
Enter your name:    Steven Clark

My name is Steven Clark_
```