Call by value: In this method a copy of entire object is created to pass into another function. When we make changes in the object inside calling function then only local copy of the object will be affected, there is no change in called function. Both copies of the object will be identical.

```
class Sample
{
      int x;
      public:
      Sample(int i) // constructor
      {
            x = i;
            cout<<"Constructing Object with "<<i<<"\n";
       }
      ~Sample() // destructor
       {
            cout<<"Destroying Object having "<<x<<"\n";
       }
      void put_x(int i)
       {
            x = i;
       }
```

```
int get_x(void)
{
        return x;
}
};

void afunc(Sample S1)
{
    S1.put_x(2) ; // change value of x in the object
     cout<<"This is x local to afunc()\n";
    cout<<"x = "<<S1.get_x()<<"\n";
 }
void main()
{
    Sample S(1); // create object with value 1
    cout<<"This is the x in main()\n";
    cout<<"x = "<<S.get_x()<<"\n";
    afunc(S); // pass object S by value to afunc()
    cout<<"Back in main()\n";
    cout<<"x = "<<S.get_x()<<"\n";
}
```

```
Constructing Object with 1
This is the x in main()
x=1
This is x local to afunc()
x=2
Destroying Object having 2
Back in main()
x=1
```

How to pass an object within the class member function as an argument

```
class Demo
{
private:
        int a;
public:
        void set(int x)
        {
                a = x;
        }
 void sum(Demo ob1, Demo ob2)
 {
        a = ob1.a + ob2.a;
 }
void print()
 {
        cout<<"Value of A : "<<a<<endl;
 }
};
```

```
int main()
 {
//object declarations
 Demo d1; Demo d2; Demo d3;
//assigning values to the data member
d1.set(10);
d2.set(20); //passing object d1 and d2
d3.sum(d1,d2); //printing the values
d1.print();
d2.print();
d3.print();
 return 0;
}
```

```
Value of A : 10
Value of A : 20
 Value of A : 30
```

Passing Objects through References

Call by reference: In this method no separate copy of the object is created, instead we are passing address of object to the function. When an object is modified inside the calling function then the actual object is also affected in the called function

```
class MyClass //simple class
 {
       public:
       int memberFun()
       {
             return 1;
       } // one public function
 };
 int fun(MyClass &object) // passing object by reference
 {
             object.memberFun(); // passed object uses its member function
 }
int main()
{
       MyClass object;
       int i = fun(object); // passing object to the function fun

}
```

Return object from a function

```
class Student
 {
       public: int stuId;
       int stuAge;
      string stuName; /* In this function we are returning the * Student object. */
      Student input(int n, int a, string s)
      {
      Student obj;
      obj.stuId = n;
      obj.stuAge = a;
      obj.stuName = s;
      return obj;
      }

      void disp(Student obj)
      {
            cout<<"Name: "<<obj.stuName<<endl;
            cout<<"Id: "<<obj.stuId<<endl;
             cout<<"Age: "<<obj.stuAge<<endl;
      }
};
```

```
int main()
{
       Student s;
       s = s.input(1001, 29, "Negan");
      s.disp(s);
      return 0;
}
```

```
Name: Negan
Id: 1001
 Age: 29
```

Const member functions

A function becomes const when const keyword is used in function's declaration. The idea of const functions is not allow them to modify the object on which they are called.

```
class Test
{
        int value;
    public:
    Test(int v = 0)
    {
        value = v;
    }
    // We get compiler error if we add a line like "value = 100;“ // in this function.
      int getValue() const
          {return value;}
};
 int main() {
            Test t(20);
            cout<<t.getValue();
            return 0;
}
```

20

```
class StarWars
{
        public: int i;
       StarWars(int x) // constructor
        { i = x; }
        int falcon() const // constant function
        {
              /* can do anything but will not modify any data members */
              cout << "Falcon has left the Base";
        }
        int gamma()
       { i++; }
};
int main()
 {
       StarWars objOne(10); // non const object
        const StarWars objTwo(20); // const object
       objOne.falcon(); // No error
       objTwo.falcon(); // No error
       cout << objOne.i << objTwo.i;
       objOne.gamma(); // No error
       objTwo.gamma(); // Compile time error
 }
```

Falcon has left the Base
Falcon has left the Base
10 20

When a function is declared as const, it can be called on any type of object. Non-const functions can only be called by non-const objects.

mutable Keyword

```
class Zee
 {
        int i;
        mutable int j;
        public:
                Zee()
                {
                        i = 0;
                        j = 0;
                }
                void fool() const
                {
                        i++; // will give error
                        j++; // works, because j is mutable
                }
};
 int main()
{
        const Zee obj;
        obj.fool();
 }
```

Local Classes

A class declared inside a function becomes local to that function and is called Local Class

```
void fun()
{
     class Test  // local to fun
     {
       /* members of Test class */
     };
}

int main()
{
    return 0;
}
```

A local class type name can only be used in the enclosing function.

```
void fun()
{
    // Local class
    class Test
    {
      /* ... */
    };

    Test t;  // Fine
    Test *tp;  // Fine
}
 int main()
{
   Test t;  // Error
   Test *tp;  // Error
   return 0;
}
```

All the methods of Local classes must be defined inside the class only.

```
void fun()
{
    class Test  // local to fun
    {
    public:
         void method() {
         cout << "Local Class method() called";
       }
    };

    Test t;
    t.method();
}
 int main()
{
    fun();
    return 0;
}
```

Method is defined outside the local class

```
void fun()
{
    class Test  // local to fun
    {
    public:
        void method();
    };

    // Error as the method is defined outside the local class
    void Test::method()
    {
        cout << "Local Class method()";
    }
}

int main()
{
    return 0;
}
```

Compiler Error: In function 'void fun()':
error: a function-definition is not allowed here before '{' token

A Local class cannot contain static data members. It may contain static functions though.

```
void fun()
{
      class Test  // local to fun
      {
         static int i;
      };
}

int main()
{
    return 0;
}
```

Compiler Error: In function 'void fun()':
error: local class 'class fun()::Test' shall not have static data member 'int fun()::Test::i'

```
void fun()
{
    class Test  // local to fun
    {
    public:
        static void method()
        {
            cout << "Local Class method() called";
        }
    };

    Test::method();
}

int main()
{
    fun();
    return 0;
}
```

Local Class method() called

Member methods of local class can only access static and enum variables of the enclosing function.

Non-static variables of the enclosing function are not accessible inside local classes.

```
void fun()
{
      static int x;
      enum {i = 1, j = 2};
       // Local class
      class Test
      {
        public:
          void method() {
              cout << "x = " << x << endl;   // fine as x is static
              cout << "i = " << i << endl;   // fine as i is enum
           }
      };
       Test t;
      t.method();                                  x = 0 i = 1
}
 int main()
{
    fun();
    return 0; }
```

```
void fun()
{
    int x;

    // Local class
    class Test
    {
      public:
        void method() {
            cout << "x = " << x << endl;
        }
    };

    Test t;
    t.method();
}

int main()
{
   fun();
   return 0;
}
```

In member function 'void fun()::Test::method()': error: use of 'auto' variable from containing function

Local classes can access global types, variables and functions.

Also, local classes can access other local classes of same function

```
int x;
 void fun()
{
  class Test1 {
      public:
        Test1() { cout << "Test1::Test1()" << endl; }
      };
           class Test2     // Second Local class
      {
          // Fine: A local class can use other local classes of same function
          Test1 t1;
      public:
         void method() {
            // Fine: Local class member methods can access global variables.
            cout << "x = " << x << endl;
         }
      };
       Test2 t;
      t.method(); }
```

```
int main()
{
    fun();
    return 0;
}
```

Test1::Test1() x = 0