Copy constructor

- A *copy constructor* is used to initialize its object with another object.
- When an object is created from other object or when an object is sent as a parameter to a function and when an object is returned from a function
- A copy constructor of a class C has a single parameter of type C&(reference

```
class sample1
{
      public:
                  sample1(sample1 &s)//copy constructor
                  {----
                  }

};
int main()
{
  sample1 s1,
  sample1 s2=s1;
   ----
   ---
```

```
#include<iostream>
using namespace std;
class sample1
{
      private:
            float x;
      public:
            float y;
            sample1(float a,float b);
            sample1(sample1 &s)
            {
                  x=s.x;
                  y=s.y;
            }
            void print();
            ~sample1();
};
sample1::sample1(float a,float b)
{
      cout<<"\n we are in constructor";
      x=a;
      y=b;
}
void sample1::print()
{
            cout<<"\n we are in class member function";
            cout<<"\n x value is::"<<x;
            cout<<"\n y value is::"<<y;
}
sample1::~sample1()
{
            cout<<"\n we are in destructor";
}

int main()
{
class sample1 s1(10.23,45.54);
class sample1 s2(s1); //calls copy constructor
class sample1 s3=s1;// calls assignment operator
cout<<"\n we are in main";
s1.print();
s2.print();
s3.print();
cout<<"\n";
return 0;
}
```

Output:

```
we are in constructor
we are in main
we are in class member function
x value is::10.23
y value is::45.54
we are in class member function
x value is::10.23
y value is::45.54
we are in class member function
x value is::10.23
y value is::45.54

we are in destructor
we are in destructor
we are in destructor
```

Shallow copy

- Compiler provided copy constructor performs a shallow copy
- If there is any pointer member, this pointer is copied to new object, instead of allocating separate memory for the object
- Both objects point at same memory. Causes memory error, when one of them is destroyed
- Deep copy, should allocate memory and then copy the content

Dynamic memory allocation

- In c++, free store (dynamic memory) can be allocated using new or malloc

```
int *p = new int;
int *p2 = new int[10];//allocate memory for 10 ints
---
--
delete p;
delete []p2;
```

Dynamic memory allocation

```
#include<iostream>
using namespace std;


int main()
{
        sample1 *p1 = new sample1(3.43,57.98);
                ---
                ---
                ----
                delete p1;
}
```

A new operator also calls constructor and delete operator also calls the destructor

Friend function

- Friend functions are non-member functions which can access all members of the class
- Keyword “friend” must be mentioned with function declaration inside class body only.
- An entire class can be made “friend” of another class
- Two classes can be mutual friends – requires forward declaration

```
#include<iostream>
using namespace std;
class two;
class one
{
      int i;
      public:
      one(int m):i (m){}
      void display()
      {
            cout<<"\n one class member value is::"<<i;
      }
      friend void add(one,two);
};
class two
{
int j;
public:
two(int m):j (m){}
void display()
{
            cout<<"\n two class member value is::"<<j;
}
friend void add(one,two);
};
void add(one x,two y)
{
cout<<"\n addition of one and two classes members is::"<<
x.i+y.j;
}
int main()
{
class one a(10);
a.display();
class two b(20);
b.display();
add(a,b);
cout<<"\n";
return 0;
}
```

const and static data members

- If a class has a **static data** member, this member is common for all objects of the class
- It should be **defined** outside the class.

```
#include<iostream>
using namespace std;
class sample1
{
int i;
static int f;
public:
sample1()
{
      cout<<"\n\n we are in no argumented constructor..";
      i=0;
      f++;
      print();
}
sample1(int a)
{
      cout<<"\n\n we are in one argumented constructor..";
      i=a;
      f++;
      print();
}
void print()
{
cout<<"\n"<<f<<" number of object(s) are created..";
cout<<"\n normal class member value is::"<<i;
}
~sample1()
{
            cout<<"\n we are in destructor..\n";
}
};
int sample1::f=0;

int main()
{
class sample1 c1,c2(10),c3,c4(20);
return 0;
}
```

- If a class has const data member, then it should be initialized in the constructor's initialiser list

```
class sample1
{
private:
        int i;
         const int m;
public:
        sample1(int, int);
        void print();
        ~sample1();
};
int sample1::f=0;//definition of static member
sample1::sample1(int a ,int m1):m(m1)       //should use initialiser for constants
{
      i= a;
}
```

const function

- If an object is const, then we can not call ordinary function with it but only const member functions.
- A const object can call only const function.
- A const function can not modify the state of the object

```
class sample1
{
private:
        int i;
        const int m;
public:
        sample1(int);
        void print();
        ~sample1();
};
 int main()
{
     const sample1 s1(2,3);
      s1.print(); //error because const p1 can call only const functions
---
--
}
```

```
class sample1
{
private:
        int i;
public:
        sample1(int);
        void print();
        ~sample1();
        void print() const;//this function can not change any data member
};
void sample1::print() const
{
        cout<<i;
    i++;// error because const function can't change the state of the Object
}
```

Static function

- A static member function can be called directly using class name
- Static function does not get implicit **this** pointer
- Static function can not access non-static data members
- Static function can not be virtual

```
class sample1
{
private:
        int i;
         static int f;
public:
        sample1(int);
        void print();
        ~sample1();
        static void printNP();
};
int sample1 :: f=0;          //definition of static member
void sample1 :: printNP()
{
cout<<f;             //ok
cout<<i;             //syntax error  because I is not a static member
cout<<this;          //syntax error because does not get implicit this pointer
}
```

Inline function

- It is a function which is not called but its code is expanded at each point of invocation.

- A member function written inside class body becomes inline by default.
- To make any function inline, keyword inline can be used.