Functions with default argument values

- You can specify default values for parameters which will be taken if no argument is sent
- Syntax similar to variable initialization
- Must be specified only once – in first declaration
- Must be specified from right most, in order

Default values for arguments

```
int sum(int a,int b,int c=0)
{
    return a+b+c;
}


/* wrong */
/* b has no default value */
float largest(float a=0,float b)
{
   return a>b?a:b;
}
```

```
int  main()
{
   int x,y,z;
   x=13;y=23;

   cout<<sum(x,y,z);

   cout<<sum(x,y);//sum of
  //two numbers. Z is 0

   }
```

Reference parameter

- When a function has a reference parameter, the modification to this is reflected in caller also
- Similar to a pointer parameter
- But has a cleaner syntax
- For large structs and objects, passing a reference parameter causes faster execution

```
int  main()
{
        int x,y,z;
        x=13;y=23;
        swap(x,y);
        cout<<x<<y;
        //prints 2313


}
void swap(int&a,int&b)
{
        int t=a;
        a=b;
        b=t;
}
```

Returning a reference

- Returning a reference lets the returned value to be modified by caller.

```
int &incr(int );
int main()
{
        int n=10;
        incr(n);//a modified

}

int &incr(int b)
{
        static int a=10;
        cout<<a;
        a+=1;
        return a;
}
```

Output :

10

Classes and objects

Class

- Class encapsulates both data and functions together
- Class describes features of an object
- Class members can have access specifiers like
  - **private**
  - **public**
  - **protected**

- Default specifier is private
- Data hiding – make data members private and functions public.

```
#include<iostream>
using namespace std;
class sample1
{
    private:
        float x;
    public:
        float y;
        void setdata(float a,float b)
        {
            x=a;
            y=b;
        }
        float getx()
        {
            return x;
        }
        float gety()
        {
            return y;
        }
void print()
{
cout<<"\n we are in class member function";
cout<<"\n x value is::"<<x;
cout<<"\n y value is::"<<y;
}
};
int main()
{
class sample1 s1;
s1.print();//initially x and y having garbage values
s1.setdata(10.32,45.56);
cout<<"\n we are in main";
cout<<"\n x value is::"<<s1.getx();
cout<<"\n y value is::"<<s1.gety();
s1.print();
//s1.x=13.32        //error because x is private member
of  a  class
s1.y=23.33;  //it works because y is public member of a
class
s1.print();
cout<<"\n";
return 0;
}
```

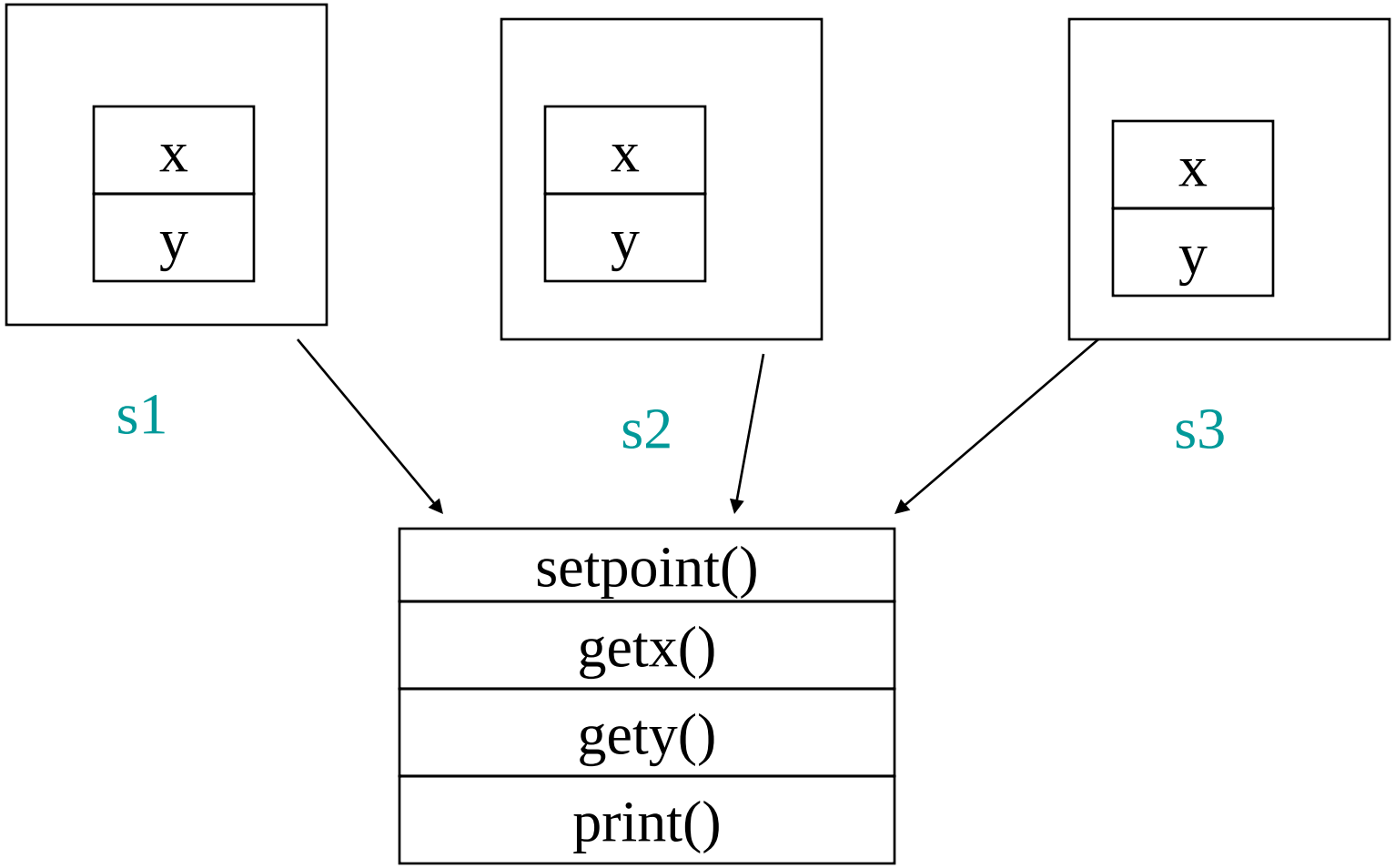
Output:

```
we are in class member function
 x value is::-1.94756
 y value is::3.35772e-39
 we are in main
 x value is::10.32
 y value is::45.56
 we are in class member function
 x value is::10.32
 y value is::45.56
 we are in class member function
 x value is::10.32
 y value is::23.33
```

private v/s public

- Public member of a class can be accessed by all
- Private member can be accessed only by the member functions of the same class
- Generally, the data members are made private and member functions public (data hiding – helps in validation)

Objects



- Each object will have its own copy of data members
- But not member functions.

Constructors

- Functions which initialize the object at the point of creation
- Automatically called at object creation
- Name is class name , no return type(not void also)
- Can be overloaded
- If no user defined constructors, compiler provides default constructor which does nothing

Default constructor

- Constructor which can be called with 0 parameters
- Is necessary when creating an array of objects
- E.g.

```
class sample1
{
    public:
        sample1()
        {
            x = y=0;
        }
};
```

- If a point class has default constructor, then its gets called when we say

```
sample1 s1;//s1.x = s1.y=0
```

```
#include<iostream>
using namespace std;
class sample1
{
    private:
        float x;
    public:
        float y;
        sample1()
        {
            x=y=0;
        }
        float getx()
        {
            return x;
        }
        float gety()
        {
            return y;
        }
void print()
{
cout<<"\n we are in class member function";
cout<<"\n x value is::"<<x;
cout<<"\n y value is::"<<y;
}
};

int main()
{
class sample1 s1;
s1.print();//now x and y are intialized with 0 while c
retion of object s1
cout<<"\n we are in main";
cout<<"\n x value is::"<<s1.getx();
cout<<"\n y value is::"<<s1.gety();
s1.print();
s1.y=23.33;
s1.print();
cout<<"\n";
return 0;
}
```

Output:

```
we are in class member function
x value is::0
y value is::0
we are in main
x value is::0
y value is::0
we are in class member function
x value is::0
y value is::0
we are in class member function
x value is::0
y value is::23.33
```

Parameterized constructors

```
Class sample1
{-------
    public:
    sample1(float a, float b)
    {
        x = a;
        y=b;
    }
};
```

For these, at object creation must supply arguments as follows

```
sample s1(10,20);
```

- s
- 10 and 20 are sent as arguments to constructor

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

int main()
{
class sample1 s1(10.23,45.54);
s1.print();
//now x and y are initialized with some values while c
reation of object s1
cout<<"\n we are in main";
cout<<"\n x value is::"<<s1.getx();
cout<<"\n y value is::"<<s1.gety();
s1.print();
s1.y=23.33;
s1.print();
cout<<"\n";
return 0;
}
```

Output:

```
we are in class member function
 x value is::10.23
 y value is::45.54
 we are in main
 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::23.33
```

Destructor

- Functions Called when an object is destroyed
- Must do the cleaning up
- If not written, compiler provides one
- Same name as class with ~ prefix, no par.and no return type
- Can not be overloaded
- E.g.

```
sample1::~sample1()
{
}
```

Member functions definition outside the class

we can provide body to the member functions, constructors, destructors out side the class also.

Syntax to member function:

```
Return_Type Class_Name  :: Function_name (arguments)
{
        --------------
}
```

Syntax to constructors:

```
Class_Name  :: Constructor_name (arguments)
{
        --------------
}
```

Syntax to destructor:

```
Class_Name  :: ~destructor_name ()
{
        --------------
}
```

```
#include<iostream>
using namespace std;
class sample1
{
	private:
		float x;
	public:
		float y;
		sample1(float a,float b);
		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);
		cout<<"\n we are in main";
		s1.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 destructor
```

```
#include<iostream>
using namespace std;
class sample1
{
	private:
		float x;
	public:
		float y;
		sample1(float a,float b);
		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;  //error
cout<<"\n we are in main";
s1.print();
cout<<"\n";
return 0;
}
```