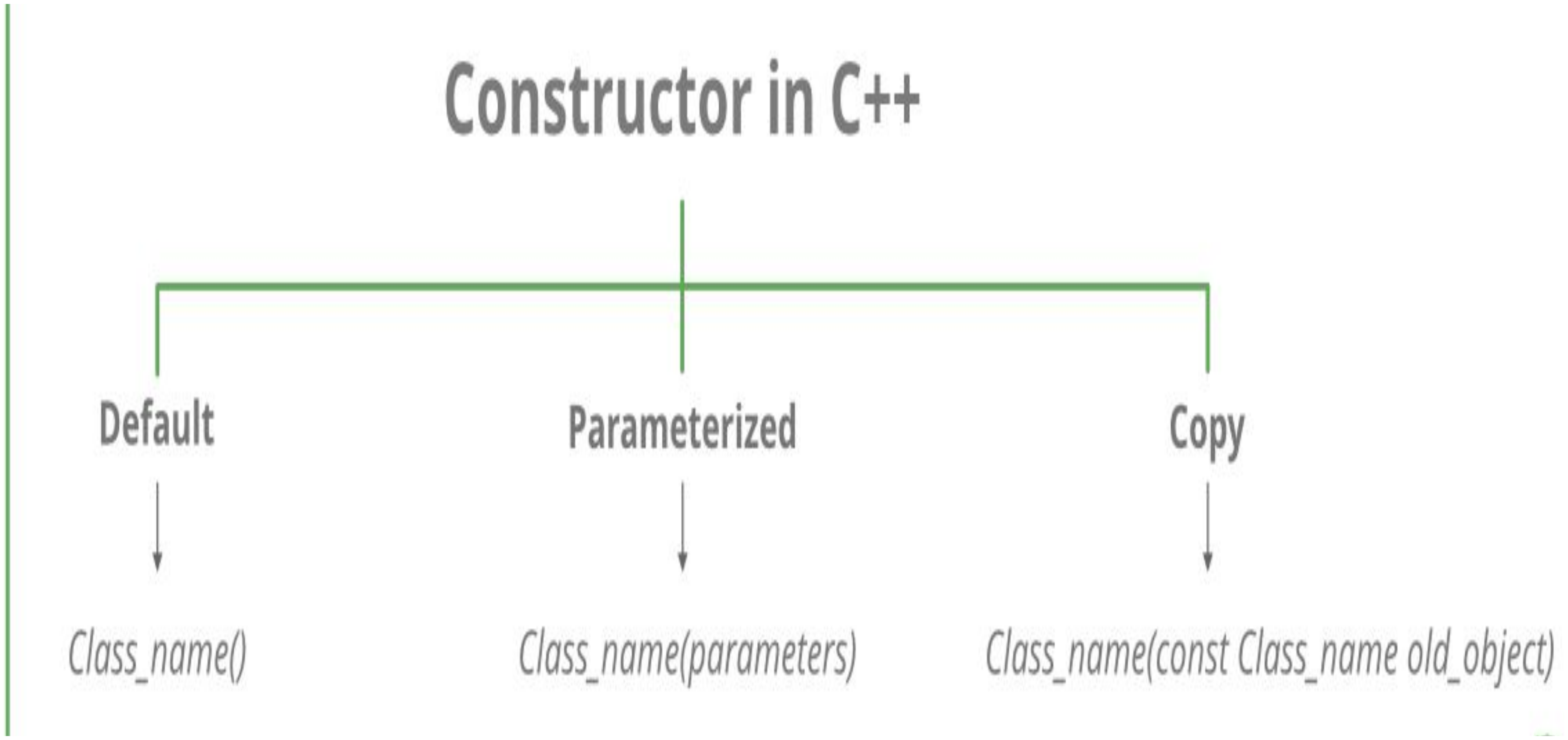
Constructors and Destructors

Constructor Function

- A Constructor is a special member function whose task is to initialize the object of its class.
- Its name is same as class name.
- The Constructor is invoked whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class.


Constructor in C++
Default
Parameterized
Copy
Class_name()
Class_name(parameters)
Class_name(const Class_name old_object)

Simple Program

```
#include<iostream.h>
class integer
{
int m,n;
public:
Integer(); // constructor declared
};
integer :: integer( ) // constructor defined
{
m= 0;
n=0;
}
```

Default Constructors

- A constructor that accepts no parameter is called default constructor.
- The default constructor for class A is A::A().
- If no such constructor is defined, then compiler supplies a default constructor.

Example:

```
class integer
{ int m,n;
public:
integer(  ); // default constructor declared
};
```

Characteristics:

- They should be declared in public section.
- They are invoked automatically when objects are created.
- They do not have return types, not even void and therefore cannot return values.
- Like other C++ functions, they can have default arguments.
- Constructors cannot be virtual
- We cannot refer to their addresses
- They cannot be inherited, though a derived class can call the base class constructor

Parameterized Constructors

- The constructor that can take arguments is called parameterized constructor.

```
Class integer
{
int m, n;
Public :
integer(int x, int y); // parameterized constructor
};
integer::integer(int x, int y)
{m=x;
n=y;
}
```

- When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly.
- integer e = integer (0, 50); // Explicit call

- integer e(0, 50);   // Implicit call

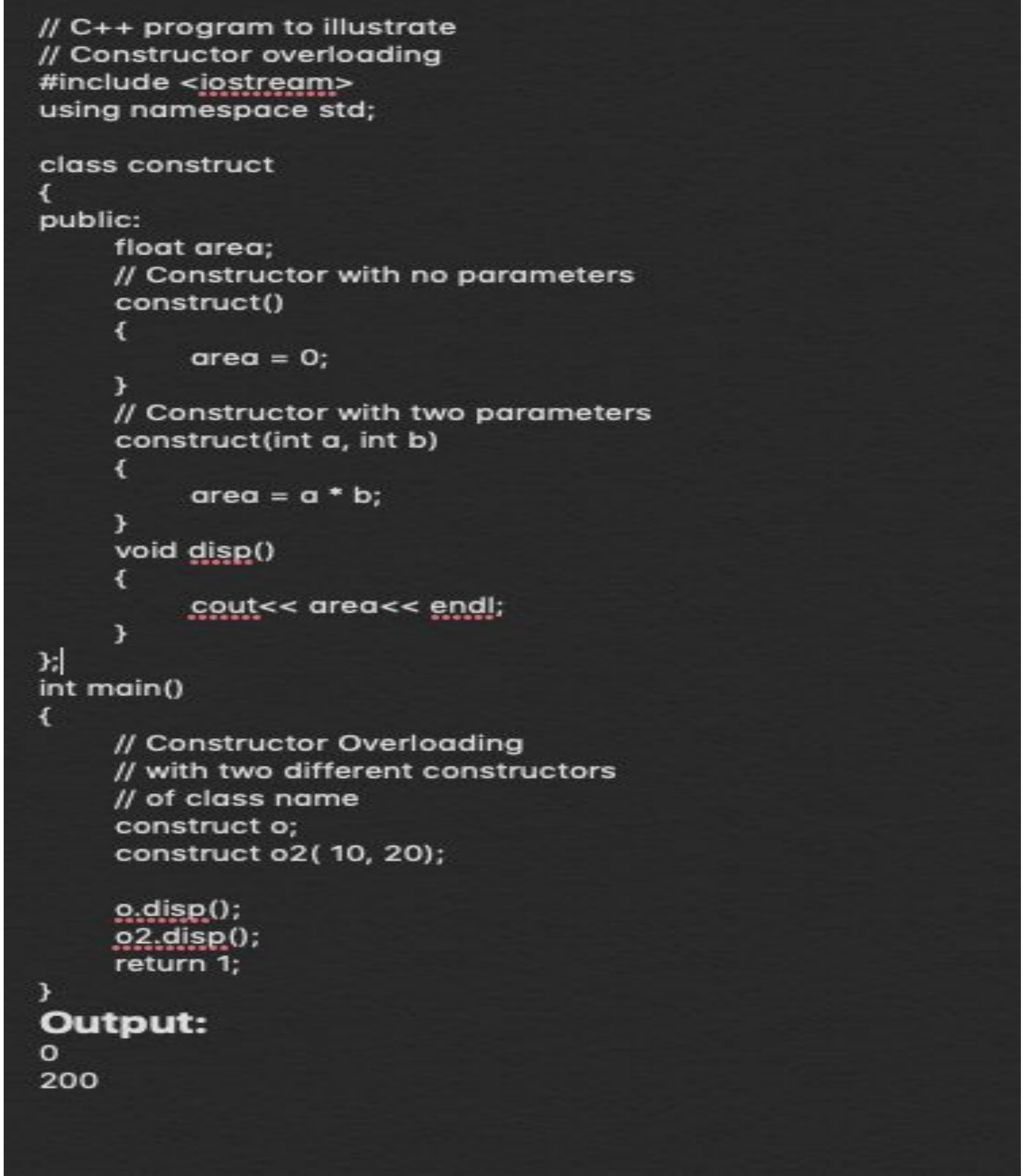
Uses of Parameterized constructor:

- It is used to initialize the various data elements of different objects with different values when they are created.
- It is used to overload constructors.

Constructor Overloading

- In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments.
- Overloaded constructors essentially have the same name (exact name of the class) and differ by number and type of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

- Example:



```
// C++ program to illustrate
// Constructor overloading
#include <iostream>
using namespace std;

class construct
{
public:
        float area;
        // Constructor with no parameters
        construct()
        {
                area = 0;
        }
        // Constructor with two parameters
        construct(int a, int b)
        {
                area = a * b;
        }
        void disp()
        {
                cout<< area<< endl;
        }
};
int main()
{
        // Constructor Overloading
        // with two different constructors
        // of class name
        construct o;
        construct o2( 10, 20);

        o.disp();
        o2.disp();
        return 1;
}
```

Output:

```
0
200
```

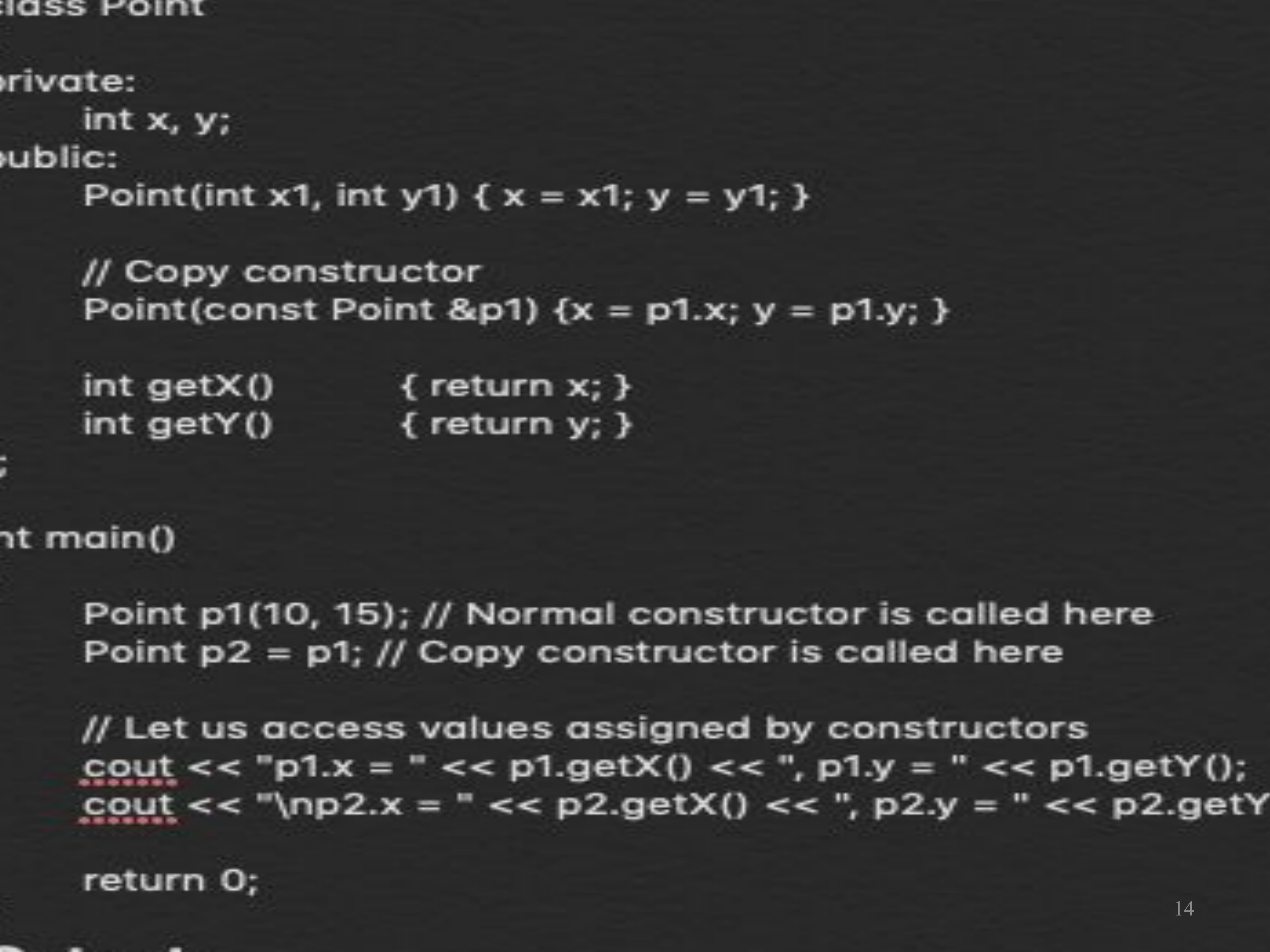
Copy Constructor

- A copy constructor is a member function that initializes an object using another object of the same class.
- A copy constructor has the following general function prototype:

```
ClassName (const ClassName &old_obj);
```

In C++, a Copy Constructor may be called in the following cases:

- 1. When an object of the class is returned by value.
- 2. When an object of the class is passed (to a function) by value as an argument.
- 3. When an object is constructed based on another object of the same class.
- 4. When the compiler generates a temporary object.



```
lass Point

rivate:
      int x, y;
ublic:
      Point(int x1, int y1) { x = x1; y = y1; }

      // Copy constructor
      Point(const Point &p1) {x = p1.x; y = p1.y; }

      int getX()          { return x; }
      int getY()          { return y; }
;

nt main()

      Point p1(10, 15); // Normal constructor is called here
      Point p2 = p1; // Copy constructor is called here

      // Let us access values assigned by constructors
      cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
      cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY

      return 0;
```

Multiple constructors in a class

```
Class integer
{int m,n;
Public:
integer() { m=0; n=0;}   //first
integer(int a,int b) {m=a;n=b;}  //second
integer(integer &i) { m=i.m; n=i.n; }  //third
};
```

Statement

```
integer I1;
integer I2(10,20);
integer I3(I2);
```

Destructors

- Just as a default constructor is called when a class object is first instantiated, a default destructor is called when the object is destroyed
- A default destructor cleans up any resources allocated to an object once the object is destroyed
- The default destructor is sufficient for most classes, except when you have allocated memory on the heap

Destructors

- You create a destructor function using the name of the class, the same as a constructor
- A destructor is commonly called in two ways:
  - When a stack object loses scope because the function in which it is declared ends
  - When a heap object is destroyed with the delete operator

Syntax rules for writing a destructor function :

- A destructor function name is the same as that of the class it belongs except that the first character of the name must be a tilde (~).
- It is declared with no return types (not even void) since it cannot even return a value.
- It cannot be declared static ,const or volatile.
- It takes no arguments and therefore cannot be overloaded.
- It should have public access in the class declaration.
-

```
Class  employee
{
Private :
Char name[20];
Int ecode ;
Char  address[30];
Public :
employee ( ) // constructor
~ employee ( ) // destructor
Void getdata( );
Void display( );
};
```