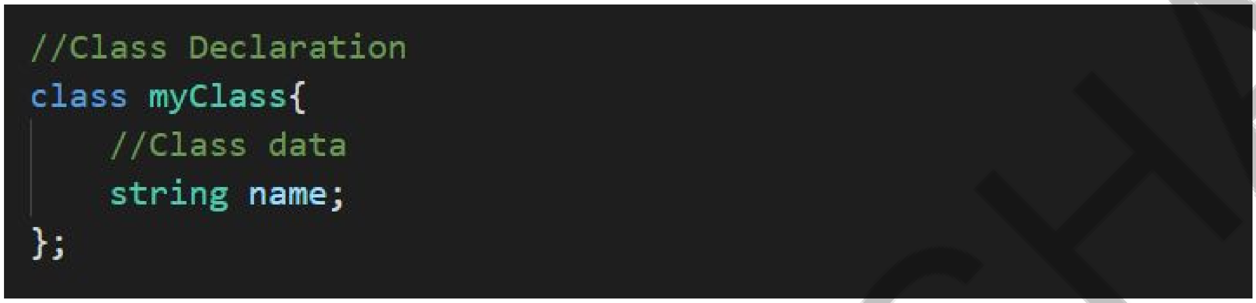
OOPS in C++

Classes in C++

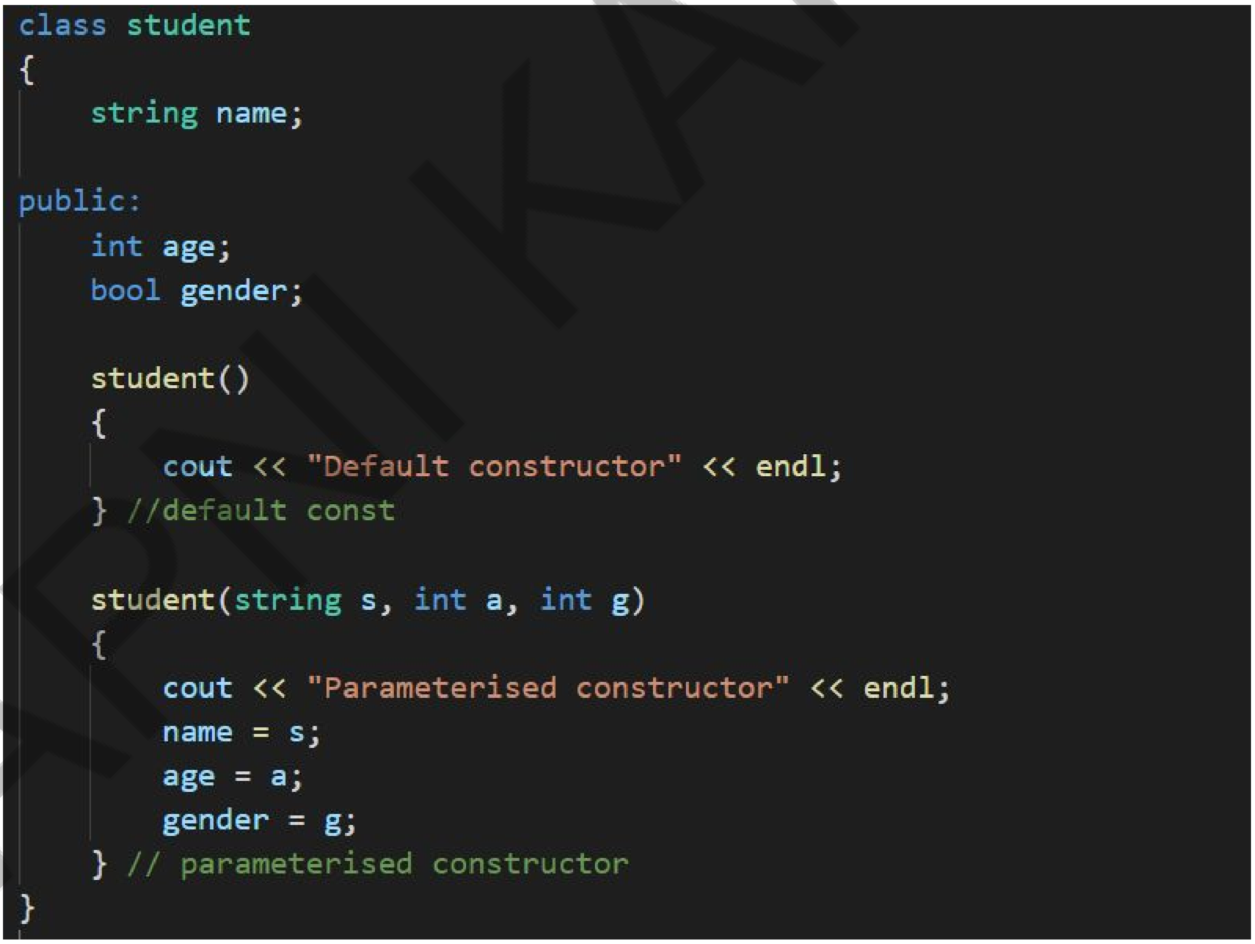


```
//Class Declaration
class myClass{
    //Class data
    string name;
};
```

Classes have objects which have the same data type as defined in the class

Constructors in C++

Invoked as soon as the class objects are created.
It can be parameterized as well.



```
class student
{
    string name;

public:
    int age;
    bool gender;

    student()
    {
        cout << "Default constructor" << endl;
    } //default const

    student(string s, int a, int g)
    {
        cout << "Parameterised constructor" << endl;
        name = s;
        age = a;
        gender = g;
    } // parameterised constructor
}
```

Copy Constructors

To copy the class objects we use the copy constructor. Here a student object is passed, it initializes the values to the new object.

```cpp
student(student &a)
{
    cout << "Copy constructor" << endl;
    name = a.name;
    age = a.age;
    gender = a.gender;
}
```

Destructors in C++

Invoked when the object goes out of scope or is explicitly destroyed by a call to delete.

```cpp
~student() {
    cout << "Destructor called" << endl;
}
```

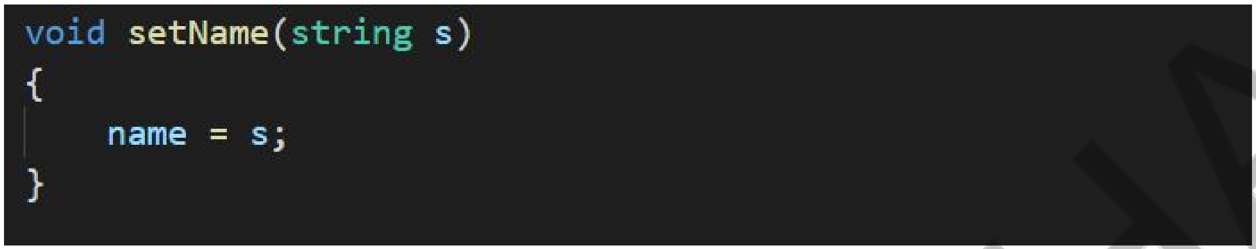
Getters

Getters are public functions used to get private data members in the class.

```cpp
string getName()
{
    return name;
}
```

Setters

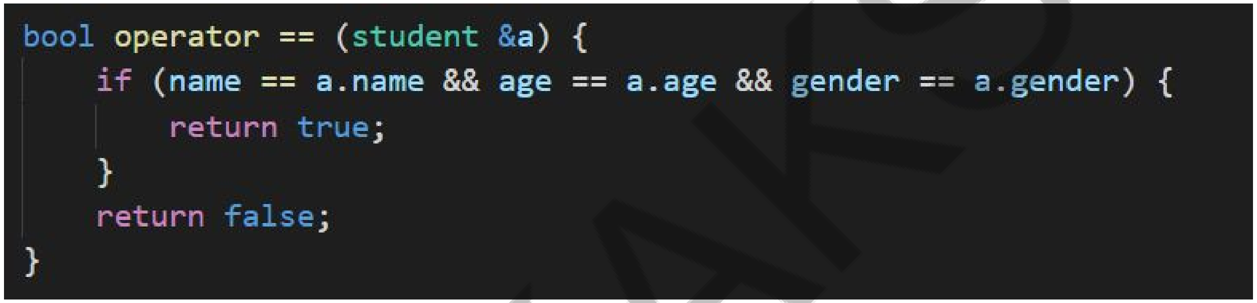
Setters are public functions used to set private data members in the class.



```
void setName(string s)
{
    name = s;
}
```

Operator overloading

We can overload operators(== , + , - ,etc), to work on class objects.



```
bool operator == (student &a) {
    if (name == a.name && age == a.age && gender == a.gender) {
        return true;
    }
    return false;
}
```

In the above code snippet, we can compare two class objects using ==.

Example: Student object a,b, we can compare from a == b.

Shallow Copy

The object and its copy, point to the same memory address. If you make a change in its copy it gets changed in the main copy as well and vice versa.

Deep Copy

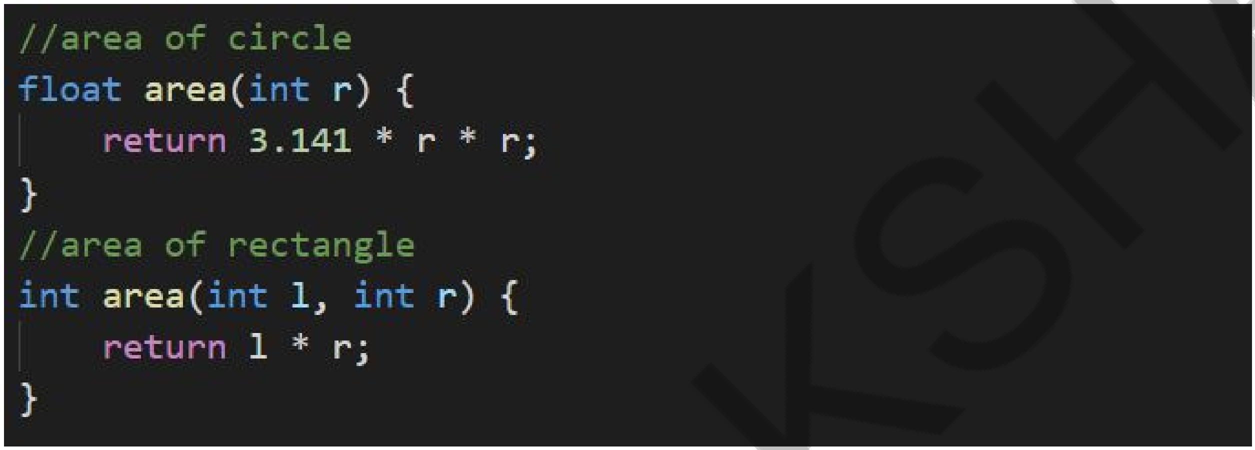
The object and its copy, point to different addresses in the memory. If you make a change in its copy it will not get changed in the main copy and vice versa.

Function Overloading

Functions having the same name but different definitions. The invoked function would depend on the arguments you pass to the functions.

area(5) //for circle

area(5, 10) //for rectangle



```
//area of circle
float area(int r) {
    return 3.141 * r * r;
}
//area of rectangle
int area(int l, int r) {
    return l * r;
}
```