OBJECT ORIENTED PROGRAMMING(CO-203)

LAB FILE

SUBMITTED BY:

SANSKAR OJHA

2K21/CO/418

SUBMITTED TO:

Mr. AMAN KUMAR PANDEY SIR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

SHAHBAD DAULATPUT, BAWANA ROAD, DELHI

INDEX :

11	**WAP to implement Generalization as extension. Use class person with name, age and gender as variables. Extend it by using subclass (Student, Teacher and Admin Staff)**
12	**WAP to implement friend function in Java or C++. Use class Sphere with radius as variable and friend function cylinder (with given height) and calculate the Volume of both.**

OBJECTIVE 1: WAP to create a class, object and calculate salary of Employees.

THEORY :

Class:

A class is a template or a blueprint that binds the properties and functions of an entity. You can put all the entities or objects having similar attributes under a single roof, known as a class. Classes further implement the core concepts like encapsulation, data hiding, and abstraction. In C++, a class acts as a data type that can have multiple objects or instances of the class type.

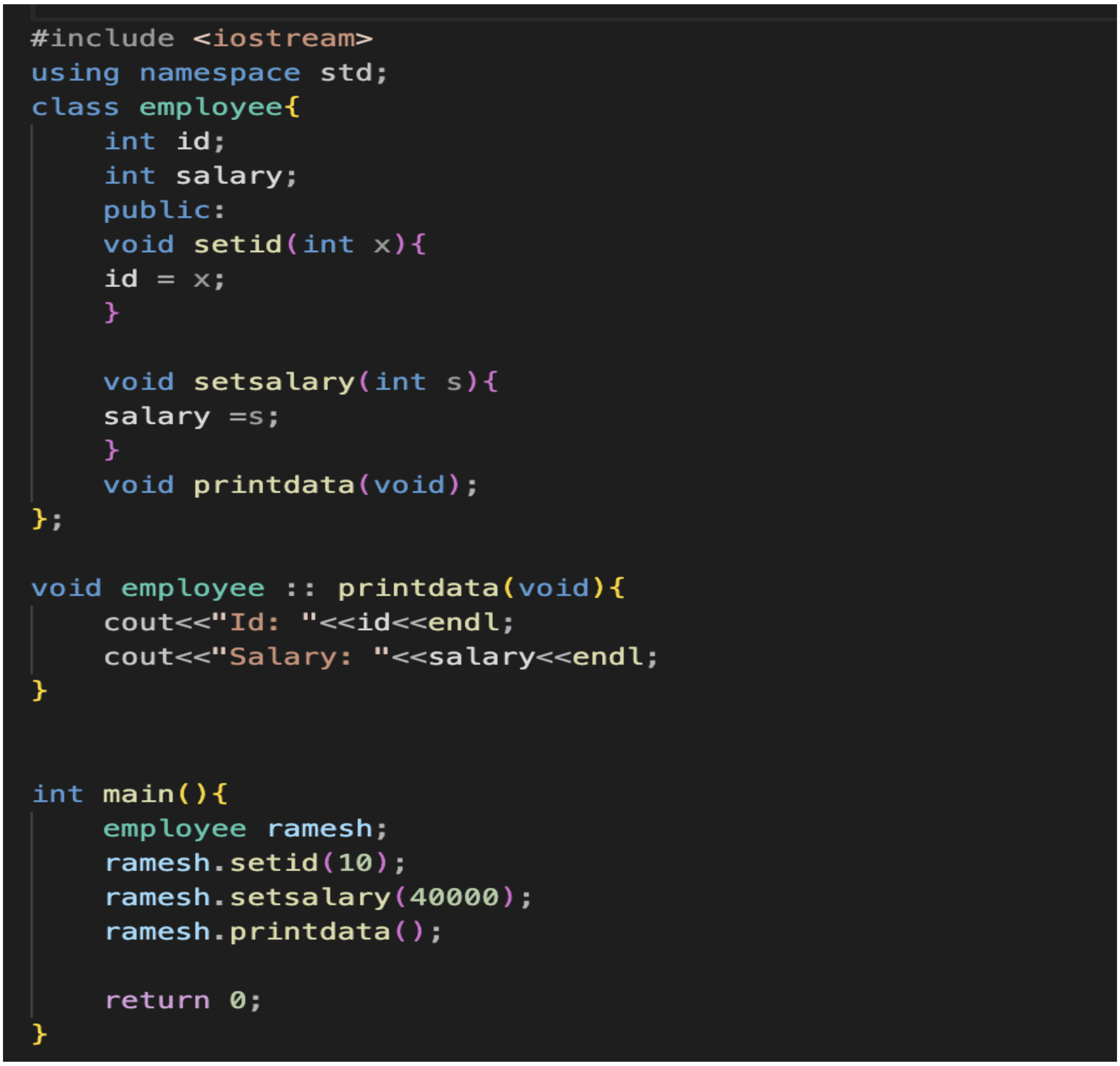
Object:

Objects in C++ are analogous to real-world entities. There are objects everywhere around you, like trees, birds, chairs, tables, dogs, cars, and the list can go on. There are some properties and functions associated with these objects. Similarly, C++ also includes the concept of objects. When you define a class, it contains all the information about the objects of the class type. Once it defines the class, it can create similar objects sharing that information, with the class name being the type specifier.

Significance of class and objects :

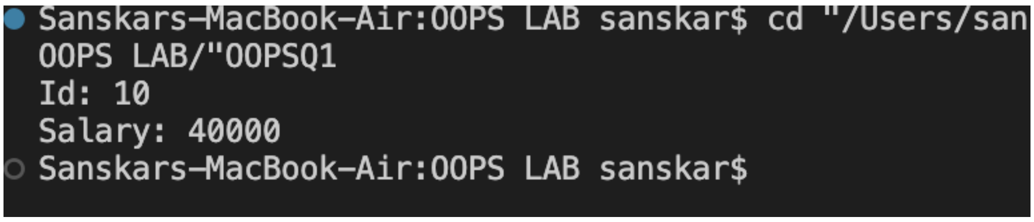
- Data hiding: A class prevents the access of the data from the outside world using access specifiers. It can set permissions to restrict the access of the data.
- Code Reusability: You can reduce code redundancy by using reusable code with the help of inheritance. Other classes can inherit similar functionalities and properties, which makes the code clean.
- Data binding: The data elements and their associated functionalities are bound under one hood, providing more security to the data.
- Flexibility: You can use a class in many forms using the concept of polymorphism. This makes a program flexible and increases its extensibility.

CODE :



```cpp
#include <iostream>
using namespace std;
class employee{
    int id;
    int salary;
    public:
    void setid(int x){
    id = x;
    }

    void setsalary(int s){
    salary =s;
    }
    void printdata(void);
};

void employee :: printdata(void){
    cout<<"Id: "<<id<<endl;
    cout<<"Salary: "<<salary<<endl;
}


int main(){
    employee ramesh;
    ramesh.setid(10);
    ramesh.setsalary(40000);
    ramesh.printdata();

    return 0;
}
```

OUTPUT :



```
Sanskars-MacBook-Air:OOPS LAB sanskar$ cd "/Users/san
OOPS LAB/"OOPSQ1
Id: 10
Salary: 40000
Sanskars-MacBook-Air:OOPS LAB sanskar$
```

LEARNING OUTCOME : The foundation of OOPS i.e. concept of class and object has been learnt like how to declare class and how to create object and use it.

OBJECTIVE 2: WAP to implement Call of Reference and reverse an array using Swap function

THEORY :

Call by reference :

The **call by reference** method of passing arguments to a function copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

Reverse an array using swap function :

To reverse an array, we will use the concept of swapping. Swapping is a term used for interchanging the values at two different locations with each other.

Algorithm :

1.Place the two pointers (let start and end) at the start and end of the array.

2.Swap arr[start] and arr[end]

3.Increment start and decrement end with 1

4.If start reached to the value length/2 or start ≥ end, then terminate otherwise repeat from step 2.

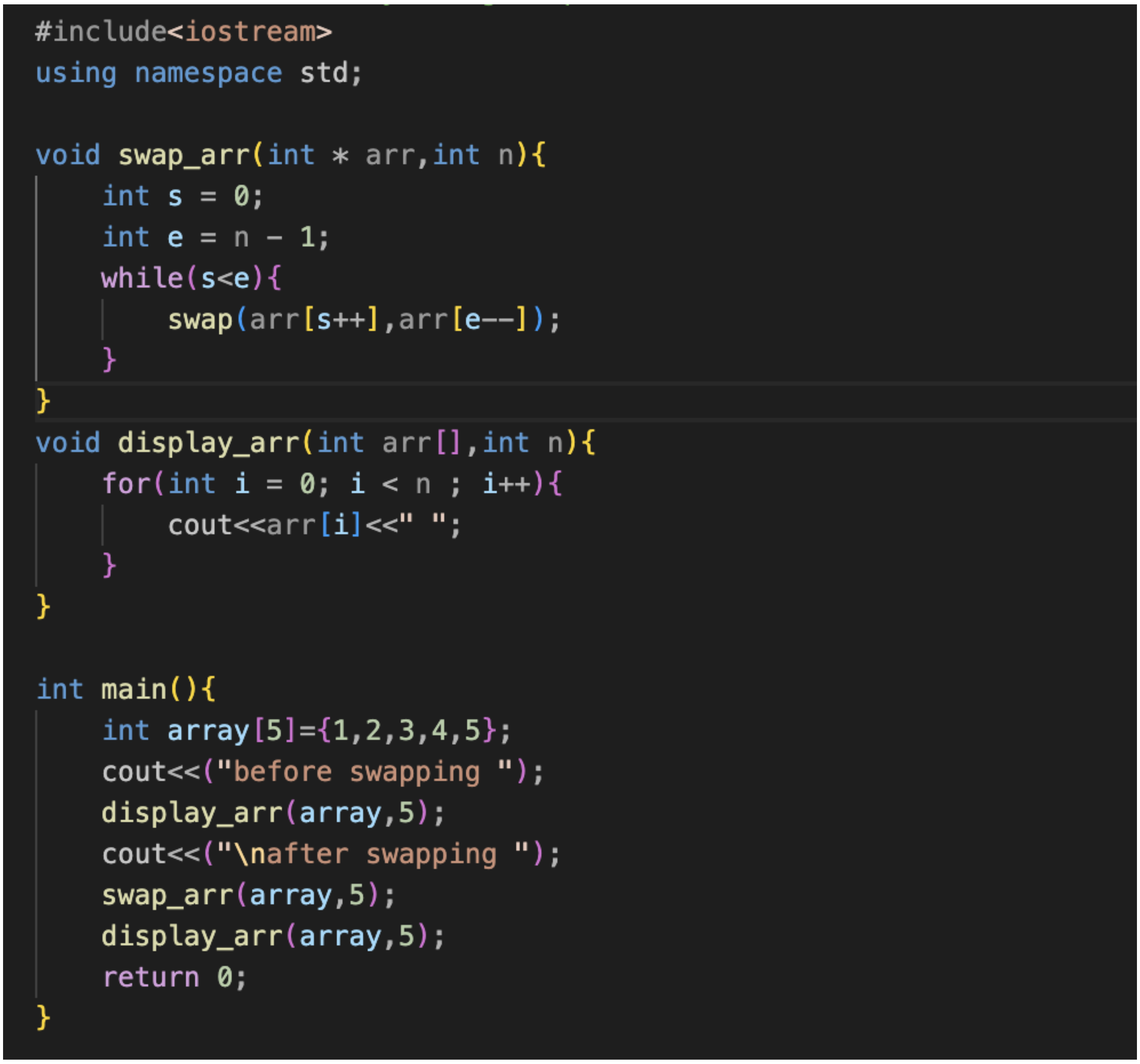
PseudoCode:

```
void revreseArray(int[] arr) {
    start = 0
    end = arr.length - 1
    while (start < end) {
        // swap arr[start] and arr[end]
```

```
            int temp = arr[start]
            arr[start] = arr[end]
            arr[end] = temp
            start = start + 1
            end = end - 1
      }
}
```

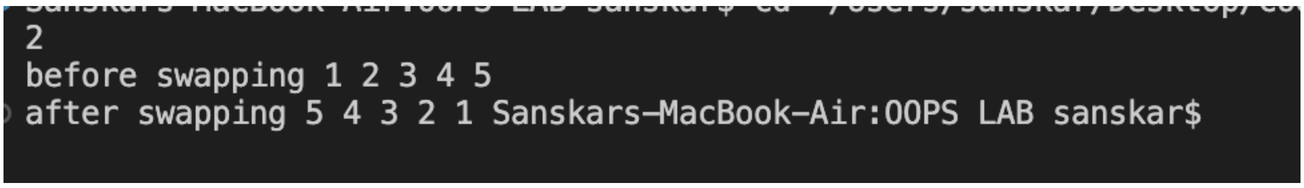
Complexity analysis : Time complexity – O(n) & space complexity – O(1)

CODE :



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

void swap_arr(int * arr,int n){
    int s = 0;
    int e = n - 1;
    while(s<e){
        swap(arr[s++],arr[e--]);
    }
}
void display_arr(int arr[],int n){
    for(int i = 0; i < n ; i++){
        cout<<arr[i]<<" ";
    }
}

int main(){
    int array[5]={1,2,3,4,5};
    cout<<("before swapping ");
    display_arr(array,5);
    cout<<("\nafter swapping ");
    swap_arr(array,5);
    display_arr(array,5);
    return 0;
}
```

OUTPUT :



```
Sanskars-MacBook-Air:OOPS LAB sanskar$ cd "/Users/sanskar/Desktop/co
2
before swapping 1 2 3 4 5
after swapping 5 4 3 2 1 Sanskars-MacBook-Air:OOPS LAB sanskar$
```

LEARNING OUTCOME :

Basic concepts of array reversal using swap functions along with passing of array in an function using call by reference.

OBJECTIVE 3: WAP to Calculate the area of Circle and Square.

THEORY :

In this question, parameterized constructor is called and simply area of circle and square has been printed using print function. Both the data i.e. radius and side has been private in class while other functions are in public.

Constructor :

A constructor is a class's member function that is used to initialize objects in a class. In C++, when an object which is the class's instance, is created, the constructor is called automatically. Thus, a constructor is a special member function of the class.

Parameterized constructor :

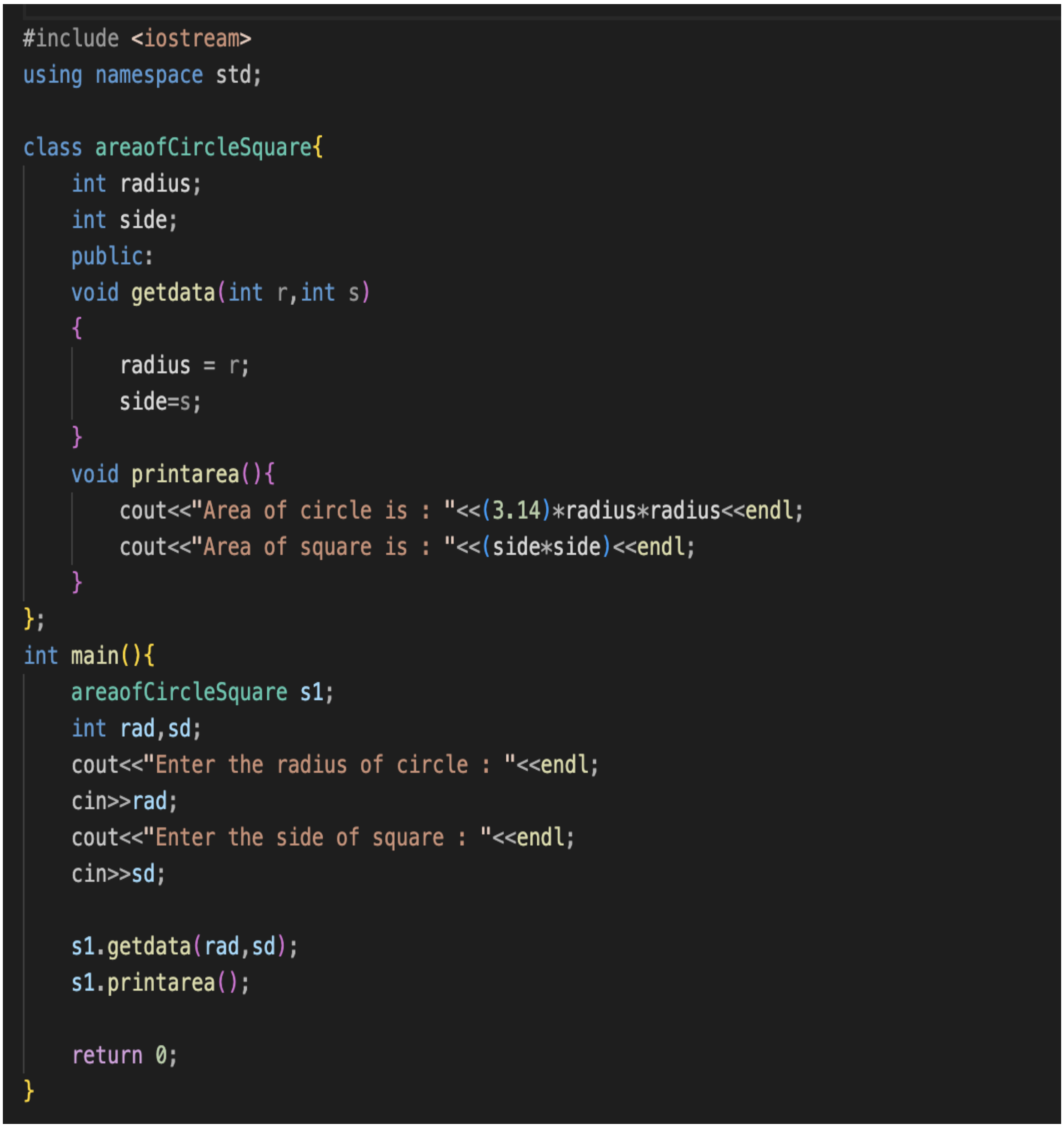
Arguments can be passed to constructors. When an object is created, these arguments help initialize an object. To create a parameterized constructor in C++, we can add parameters to a function like it can be added to any other function. When the body of the constructor is defined, the parameters are used to initialize the object.

Access specifier can be either private or protected or public. In general access specifiers are the access restriction imposed during the derivation of different subclasses from the base class.

- private access specifier

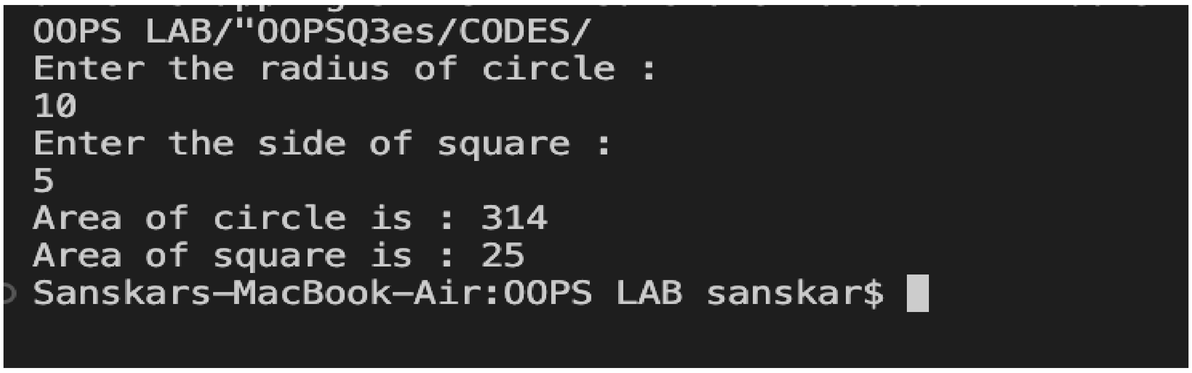
- protected access specifier
- public access specifier

CODE :



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

class areaofCircleSquare{
    int radius;
    int side;
    public:
    void getdata(int r,int s)
    {
        radius = r;
        side=s;
    }
    void printarea(){
        cout<<"Area of circle is : "<<(3.14)*radius*radius<<endl;
        cout<<"Area of square is : "<<(side*side)<<endl;
    }
};
int main(){
    areaofCircleSquare s1;
    int rad,sd;
    cout<<"Enter the radius of circle : "<<endl;
    cin>>rad;
    cout<<"Enter the side of square : "<<endl;
    cin>>sd;

    s1.getdata(rad,sd);
    s1.printarea();

    return 0;
}
```

OUTPUT :



```
OOPS LAB/"OOPSQ3es/CODES/
Enter the radius of circle :
10
Enter the side of square :
5
Area of circle is : 314
Area of square is : 25
Sanskars-MacBook-Air:OOPS LAB sanskar$
```

LEARNING OUTCOME :

Concepts like parameterized constructor creation and calling, using access specifier within a class and printing function inside class.

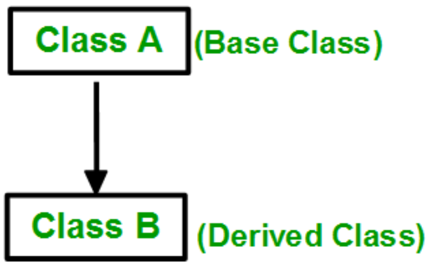
OBJECTIVE 4: WAP to implement Single, Multiple and Multi-level Inheritance

THEORY :

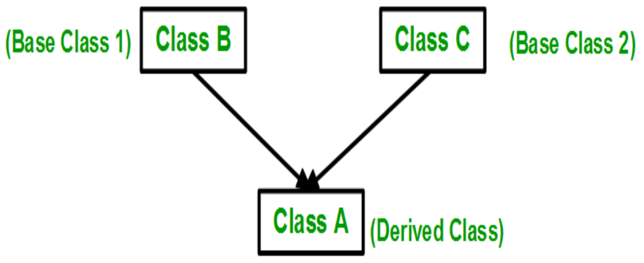
Inheritance is one of the most important aspects of Object Oriented Programming (OOP). The key to understanding Inheritance is that it provides code re-usability. In place of writing the same code, again and again, we can simply inherit the properties of one class into the other. There are 5 types of inheritance, but in this program we are using three types of inheritance. They are:

1. Single inheritance: This is a form of inheritance in which a class inherits only one parent class. This is the simple form of inheritance and hence also referred to

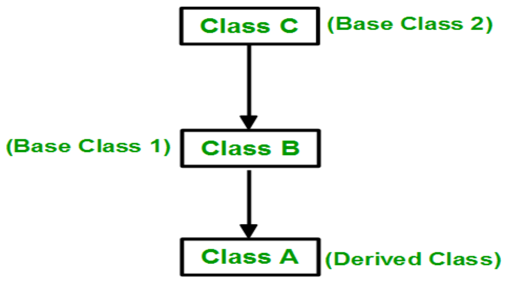
as simple inheritance. Here the class Child is inheriting only one class Parent.



2. Multiple Inheritance: An inheritance becomes multiple inheritances when a class inherits more than one parent class. The child class after inheriting properties from various parent classes has access to all of their objects.

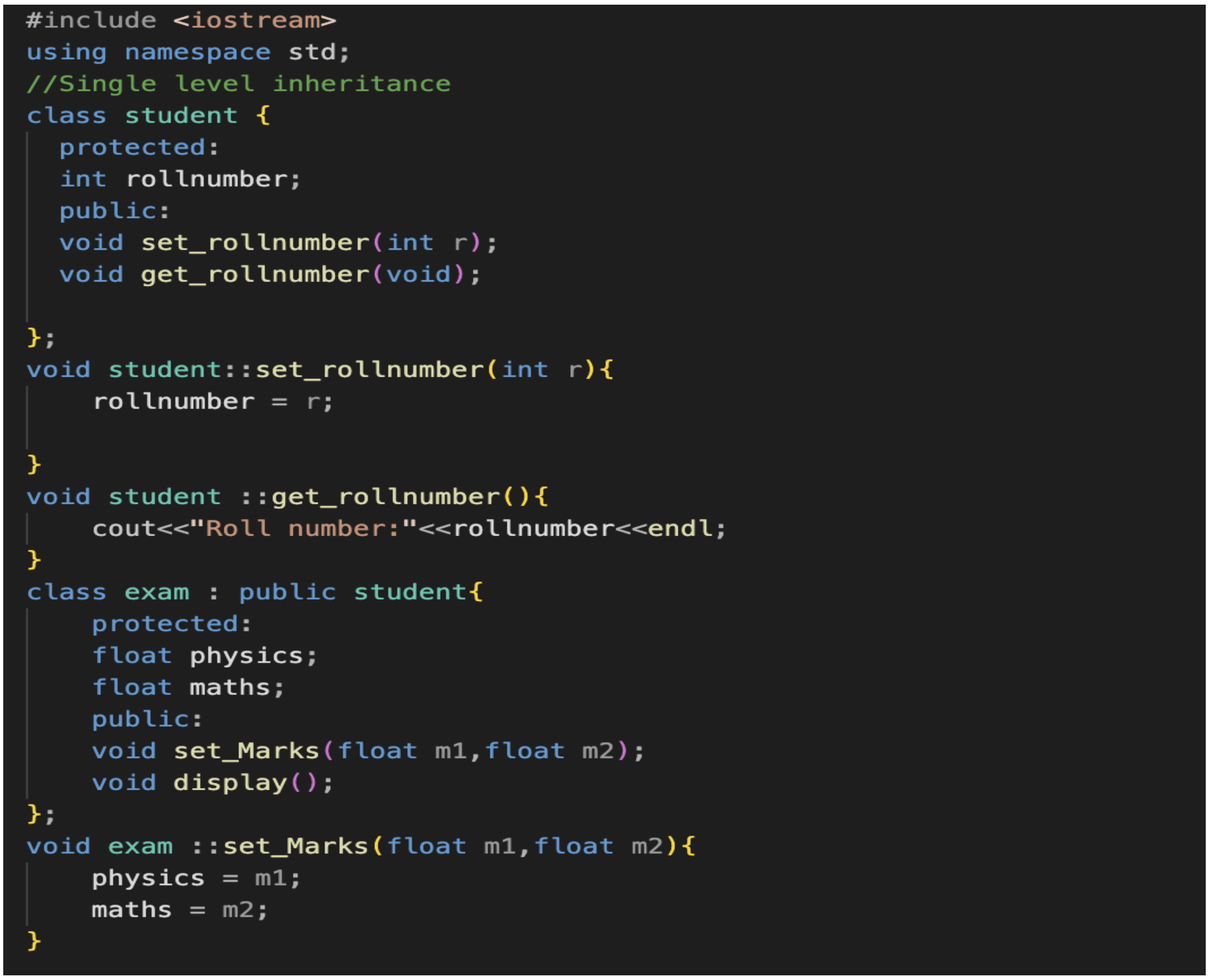


3. Multi-level Inheritance: In this type of inheritance, a derived class is created from another derived class.

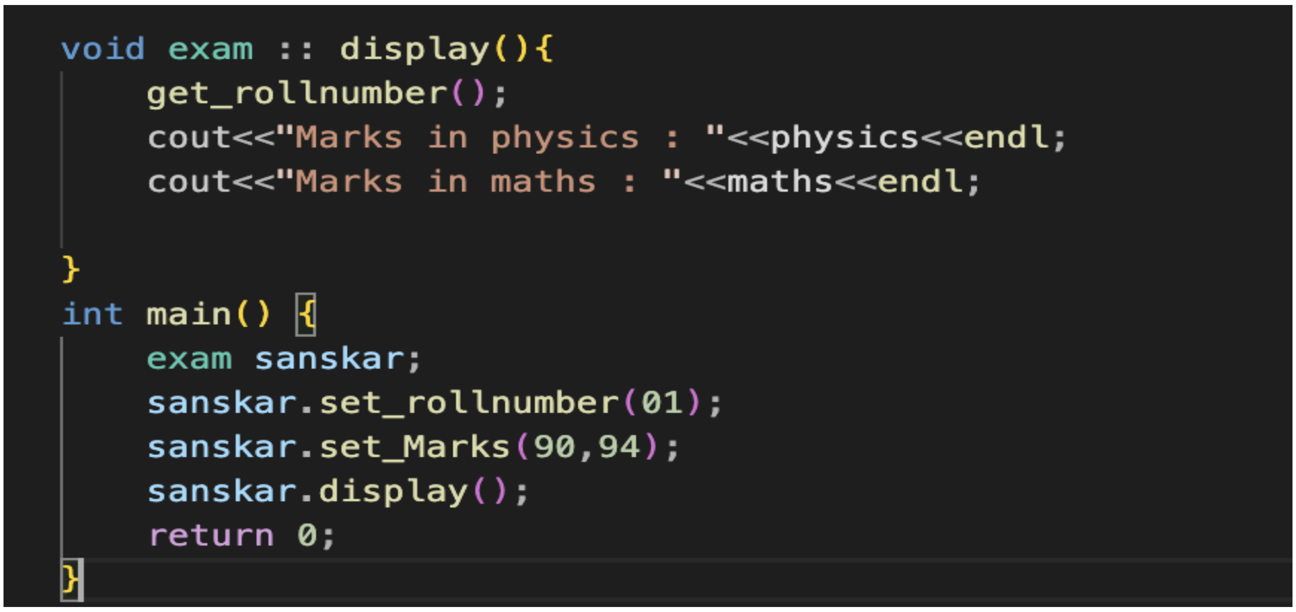


CODE :

FOR SINGLE LEVEL INHERITANCE :

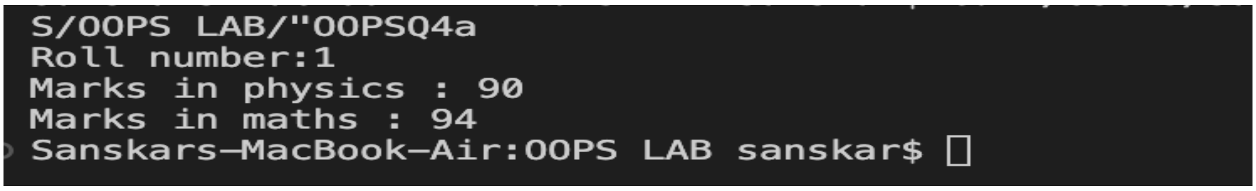


```cpp
#include <iostream>
using namespace std;
//Single level inheritance
class student {
  protected:
  int rollnumber;
  public:
  void set_rollnumber(int r);
  void get_rollnumber(void);

};
void student::set_rollnumber(int r){
    rollnumber = r;

}
void student ::get_rollnumber(){
    cout<<"Roll number:"<<rollnumber<<endl;
}
class exam : public student{
    protected:
    float physics;
    float maths;
    public:
    void set_Marks(float m1,float m2);
    void display();
};
void exam ::set_Marks(float m1,float m2){
    physics = m1;
    maths = m2;
}
```



```cpp
void exam :: display(){
    get_rollnumber();
    cout<<"Marks in physics : "<<physics<<endl;
    cout<<"Marks in maths : "<<maths<<endl;

}
int main() {
    exam sanskar;
    sanskar.set_rollnumber(01);
    sanskar.set_Marks(90,94);
    sanskar.display();
    return 0;
}
```

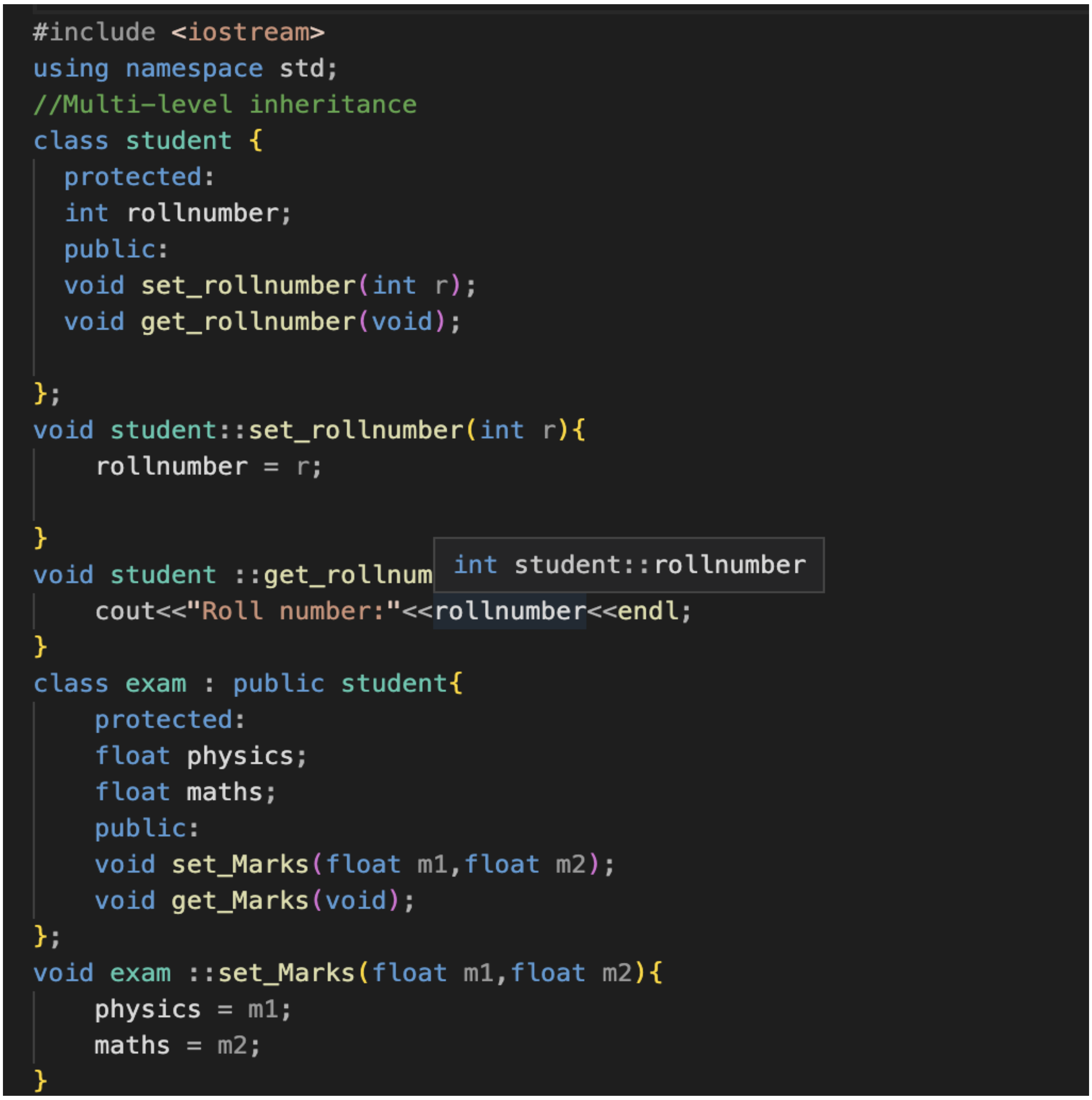
OUTPUT :



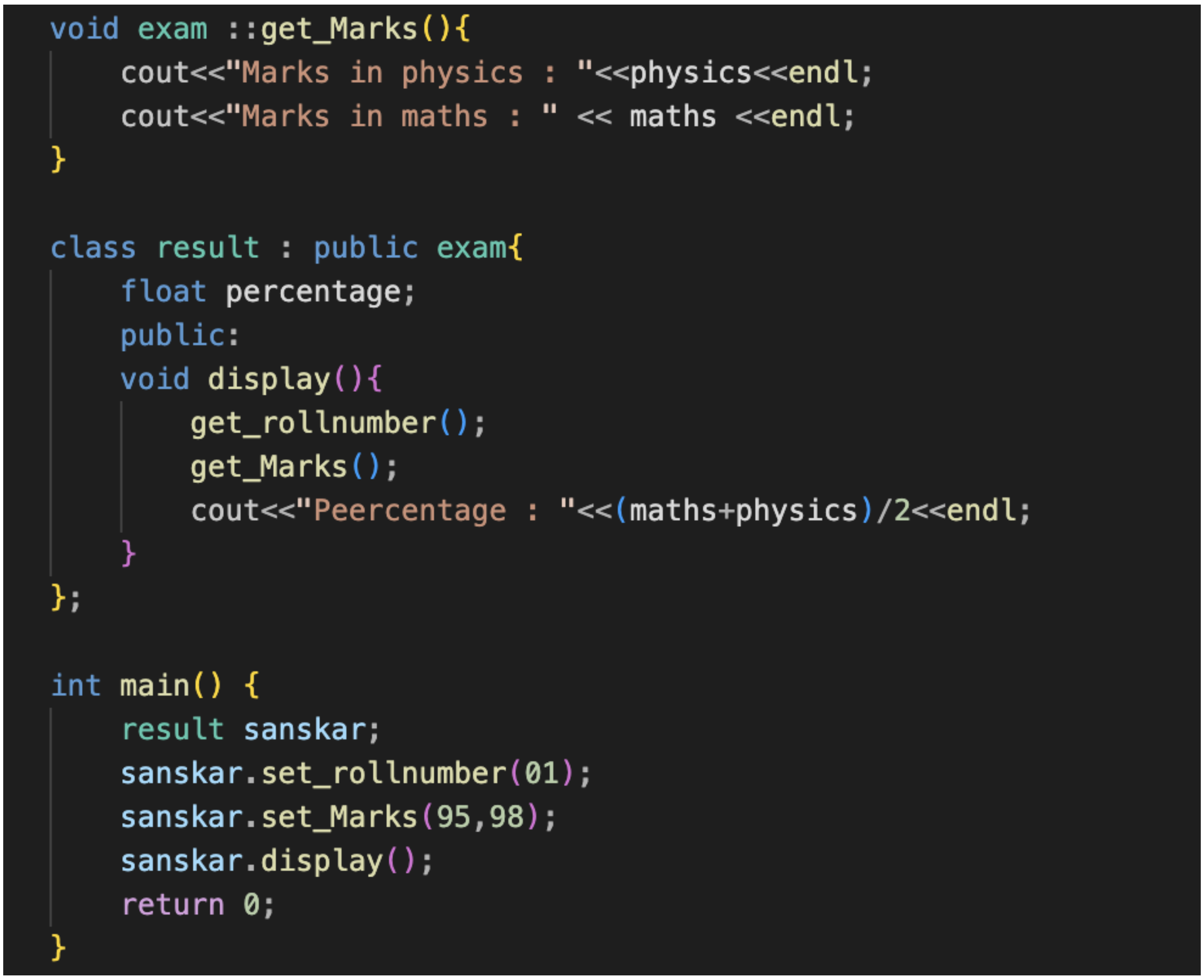
```
S/OOPS LAB/"OOPSQ4a
Roll number:1
Marks in physics : 90
Marks in maths : 94
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

FOR MULTI-LEVEL INHERITANCE:

CODE :

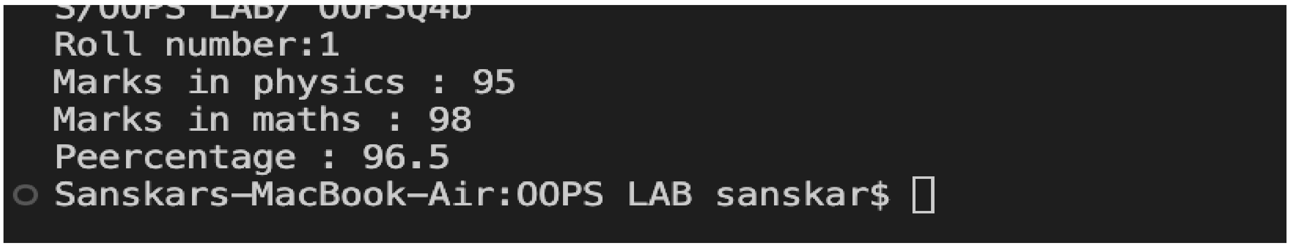


```
#include <iostream>
using namespace std;
//Multi-level inheritance
class student {
  protected:
  int rollnumber;
  public:
  void set_rollnumber(int r);
  void get_rollnumber(void);

};
void student::set_rollnumber(int r){
    rollnumber = r;

}
void student ::get_rollnum int student::rollnumber
    cout<<"Roll number:"<<rollnumber<<endl;
}
class exam : public student{
    protected:
    float physics;
    float maths;
    public:
    void set_Marks(float m1,float m2);
    void get_Marks(void);
};
void exam ::set_Marks(float m1,float m2){
    physics = m1;
    maths = m2;
}
```



```cpp
void exam ::get_Marks(){
    cout<<"Marks in physics : "<<physics<<endl;
    cout<<"Marks in maths : " << maths <<endl;
}

class result : public exam{
    float percentage;
    public:
    void display(){
        get_rollnumber();
        get_Marks();
        cout<<"Peercentage : "<<(maths+physics)/2<<endl;
    }
};

int main() {
    result sanskar;
    sanskar.set_rollnumber(01);
    sanskar.set_Marks(95,98);
    sanskar.display();
    return 0;
}
```

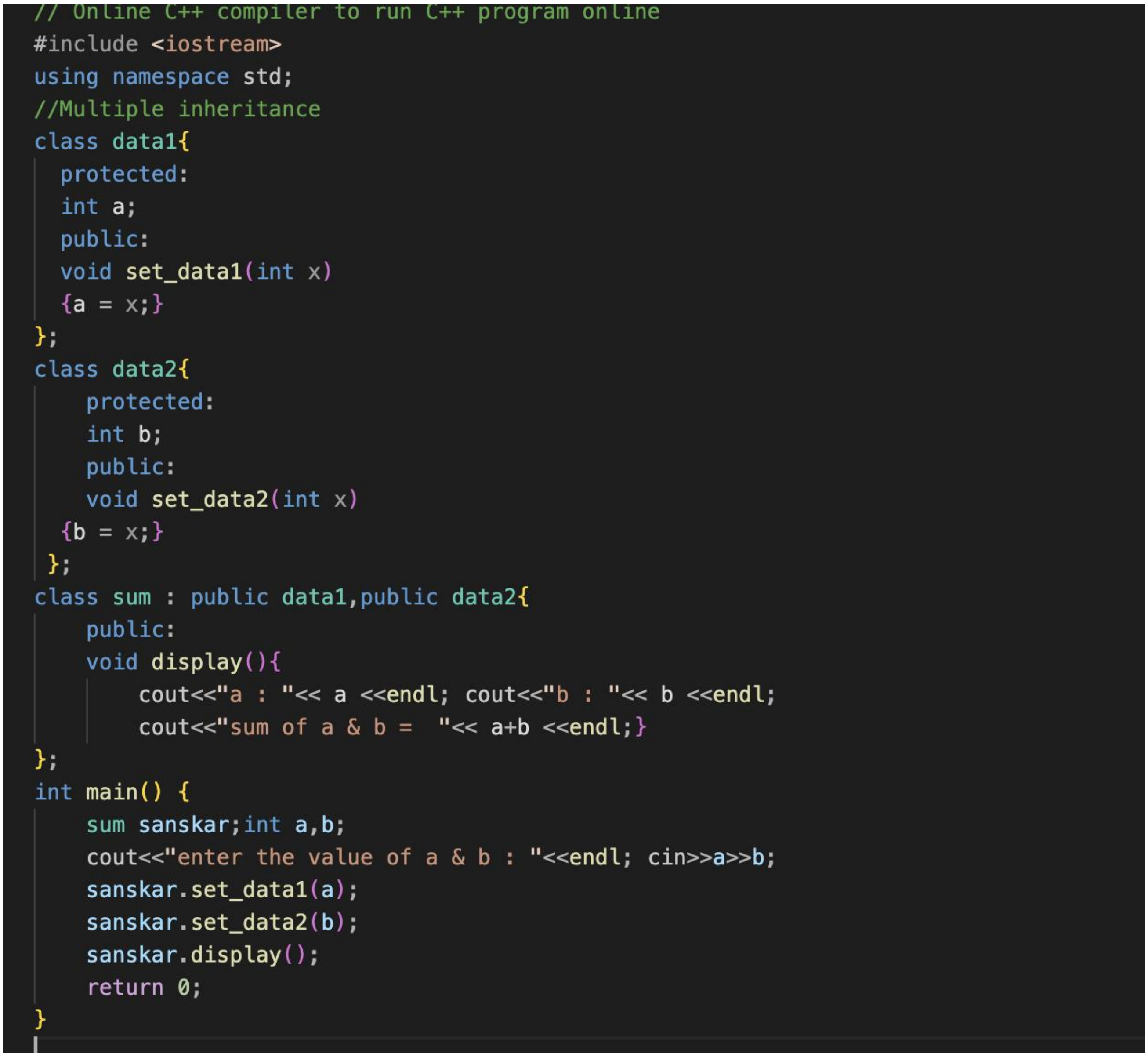
OUTPUT :



```
s/OOPS LAB/ OOPSQ4b
Roll number:1
Marks in physics : 95
Marks in maths : 98
Peercentage : 96.5
Sanskars-MacBook-Air:OOPS LAB sanskar$
```

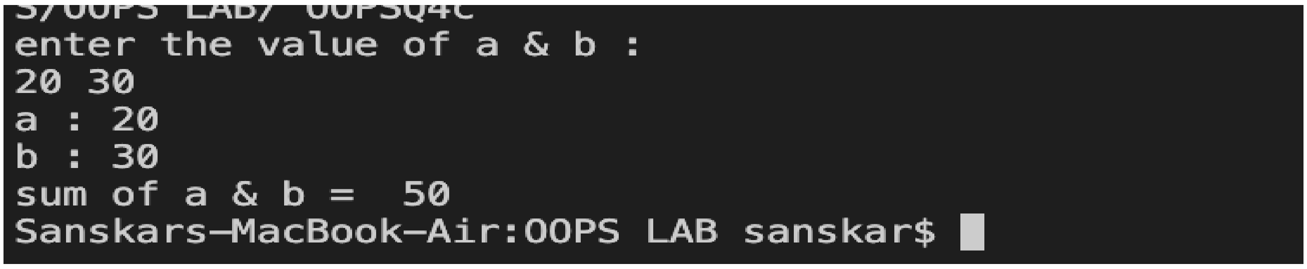
FOR MULTIPLE INHERITANCE :

CODE :



```
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;
//Multiple inheritance
class data1{
  protected:
  int a;
  public:
  void set_data1(int x)
  {a = x;}
};
class data2{
    protected:
    int b;
    public:
    void set_data2(int x)
  {b = x;}
 };
class sum : public data1,public data2{
    public:
    void display(){
        cout<<"a : "<< a <<endl; cout<<"b : "<< b <<endl;
        cout<<"sum of a & b =  "<< a+b <<endl;}
};
int main() {
    sum sanskar;int a,b;
    cout<<"enter the value of a & b : "<<endl; cin>>a>>b;
    sanskar.set_data1(a);
    sanskar.set_data2(b);
    sanskar.display();
    return 0;
}
```

OUTPUT :



```
s/OOPS LAB/ OOPSQ4c
enter the value of a & b :
20 30
a : 20
b : 30
sum of a & b =  50
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Inheritance and its types has been learnt. Single, Multiple and Multilevel inheritance along with their applications in different cases in oops has been learnt.

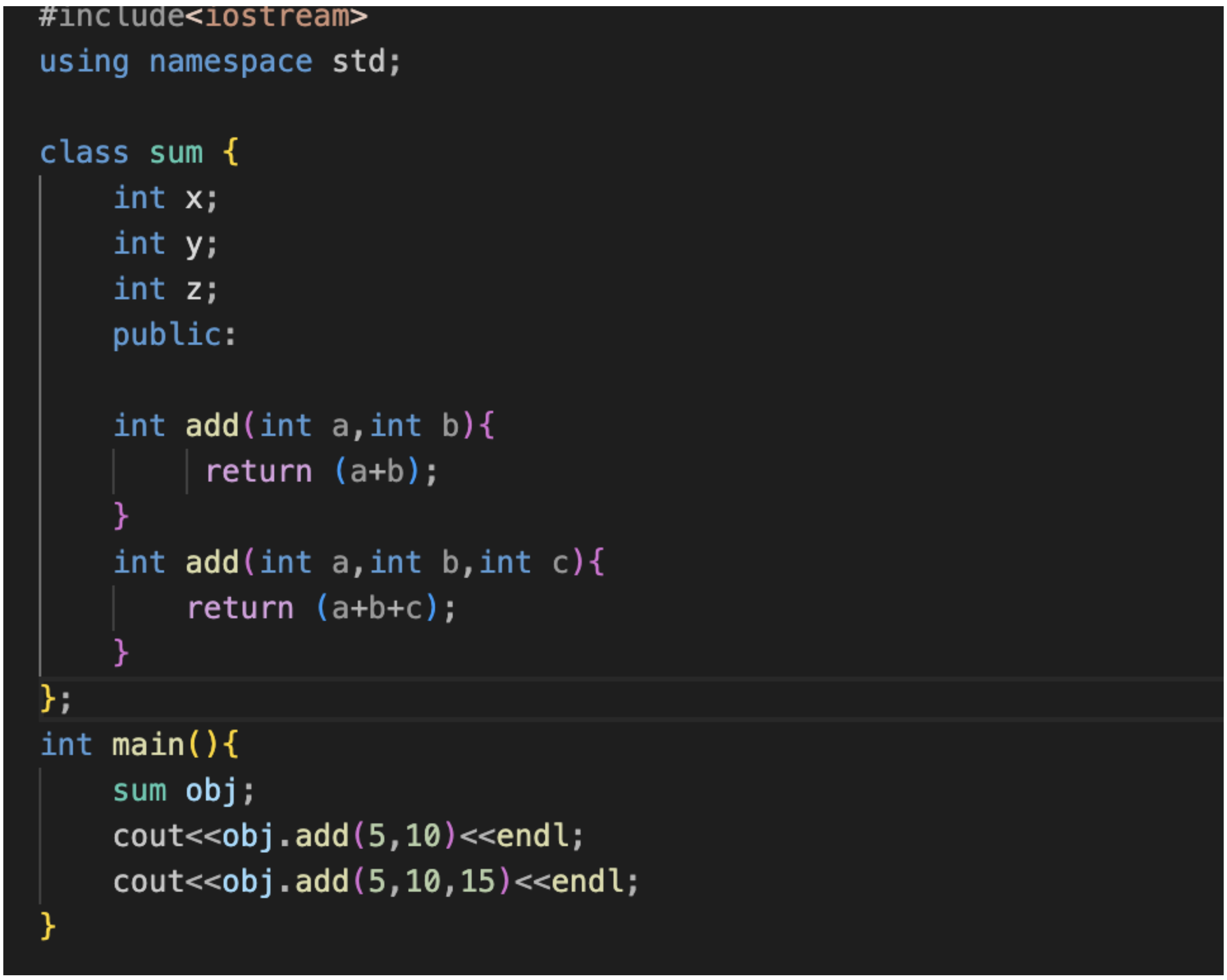
OBJECTIVE 5: WAP to implement Polymorphism by using Sum function of 2 and 3 variables.

THEORY :

When the same entity (function or object) behaves differently in different scenarios, it is known as Polymorphism in C++. The concept of polymorphism offers great scalability and boosts the code's readability. You can define polymorphism to have your unique implementation for the same method within the Parent class.

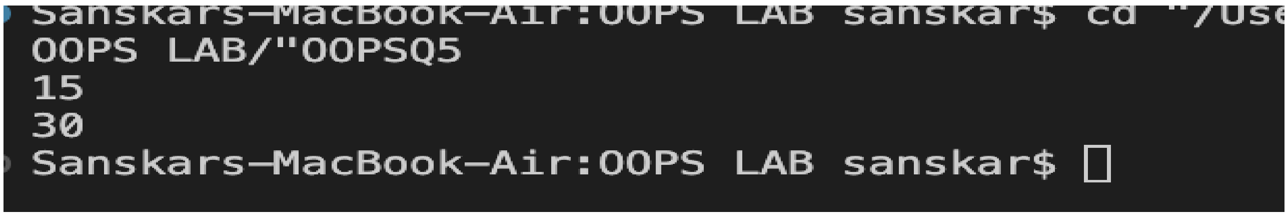
Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments. In the function overloading function will call at the time of program compilation. It is an example of compile-time polymorphism. Readability of the program increases by function overloading. It is achieved by using the same name for the same action.

CODE :



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

class sum {
    int x;
    int y;
    int z;
    public:

    int add(int a,int b){
        return (a+b);
    }
    int add(int a,int b,int c){
        return (a+b+c);
    }
};
int main(){
    sum obj;
    cout<<obj.add(5,10)<<endl;
    cout<<obj.add(5,10,15)<<endl;
}
```

OUTPUT :



```
Sanskars-MacBook-Air:OOPS LAB sanskar$ cd "/Use
OOPS LAB/"OOPSQ5
15
30
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Introduction to polymorphism and its use. Implementation of function overloading with different attributes and its application.

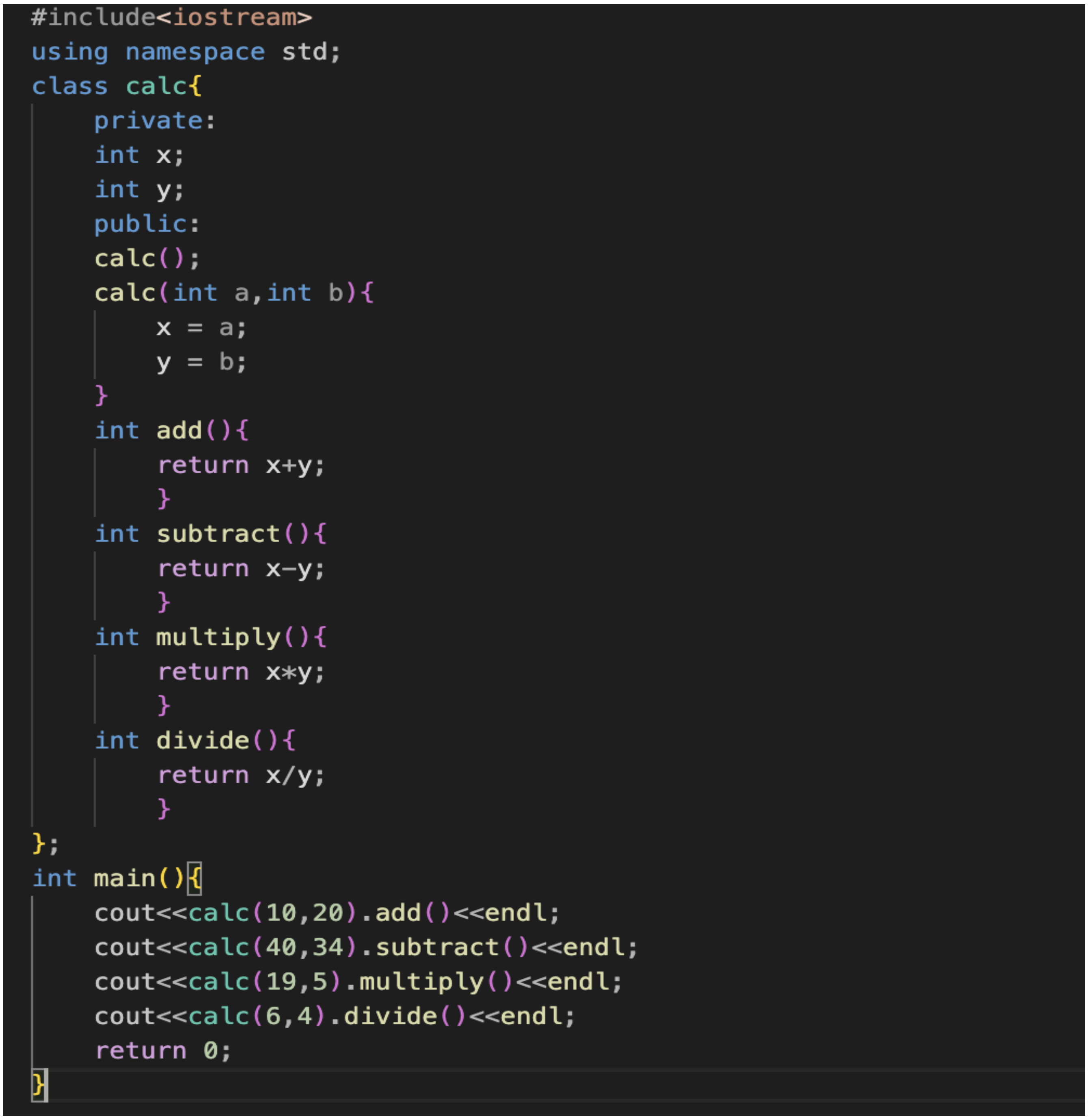
OBJECTIVE 6: WAP to implement Abstraction using Calc Class and perform different operations (add, subtract, multiply and divide).

THEORY :

Abstraction is one of the key concepts of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

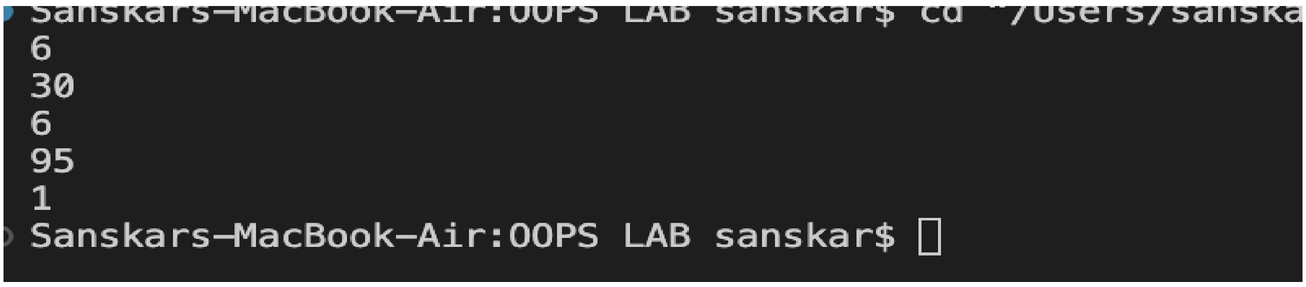
Objects in an OOP language provide an abstraction that hides the internal implementation details. Similar to the coffee machine in your kitchen, you just need to know which methods of the object are available to call and which input parameters are needed to trigger a specific operation. But you don’t need to understand how this method is implemented and which kinds of actions it has to perform to create the expected result.

CODE :



```cpp
#include<iostream>
using namespace std;
class calc{
    private:
    int x;
    int y;
    public:
    calc();
    calc(int a,int b){
        x = a;
        y = b;
    }
    int add(){
        return x+y;
        }
    int subtract(){
        return x-y;
        }
    int multiply(){
        return x*y;
        }
    int divide(){
        return x/y;
        }
};
int main(){
    cout<<calc(10,20).add()<<endl;
    cout<<calc(40,34).subtract()<<endl;
    cout<<calc(19,5).multiply()<<endl;
    cout<<calc(6,4).divide()<<endl;
    return 0;
}
```

OUTPUT :



```
Sanskars-MacBook-Air:OOPS LAB sanskar$ cd "/Users/sanska
6
30
6
95
1
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Things like data hiding and its importance along with concept of Abstraction has been learnt.

OBJECTIVE 7: WAP to implement Overloading using Motor as a class and it should calculate its monthly EMI if principle is passed as a parameter (take one P(int) and another P(float). **Rate =15 % and time=2 years.**

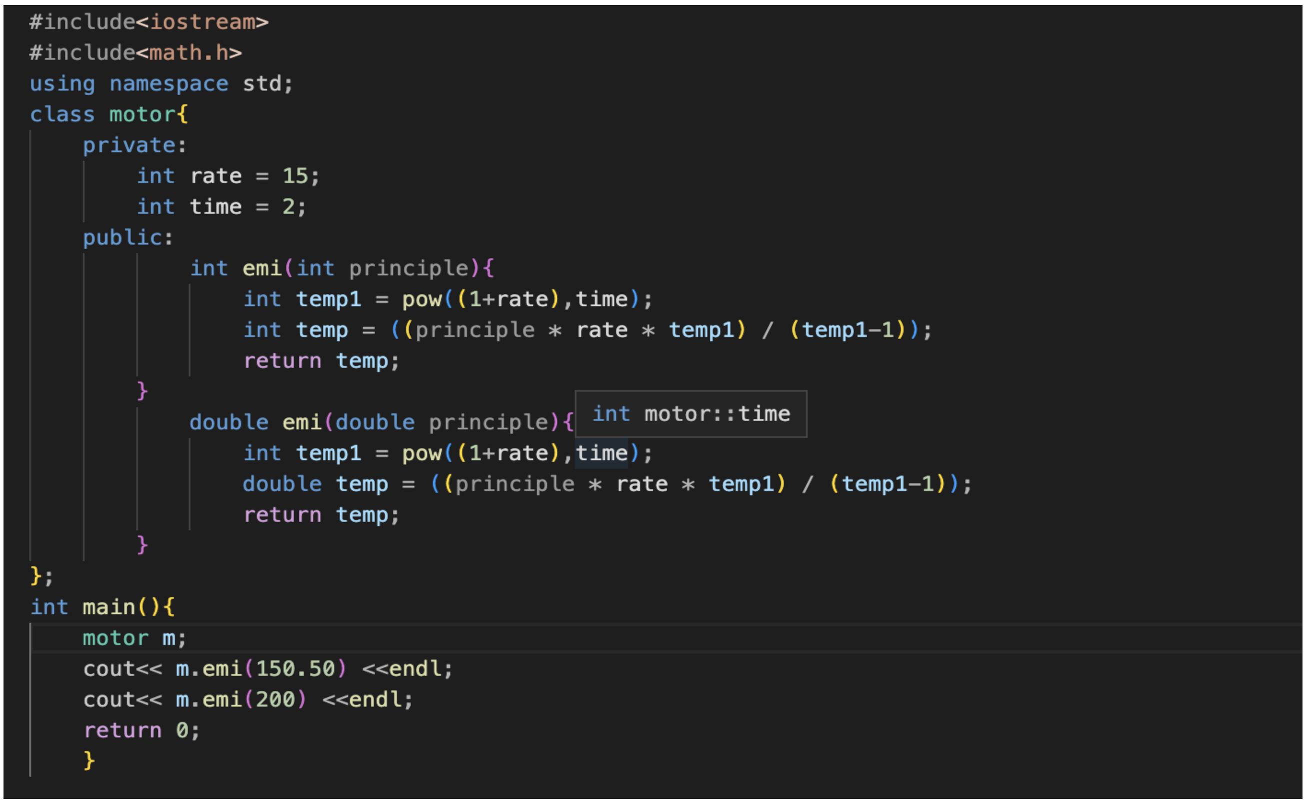
THEORY :

Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments. In the function overloading function will call at the time of program compilation. It is an example of compile-time polymorphism. Readability of the program increases by function overloading. It is achieved by using the same name for the same action.

The easiest way to remember this rule is that the parameters should qualify any one or more than one of the following conditions:

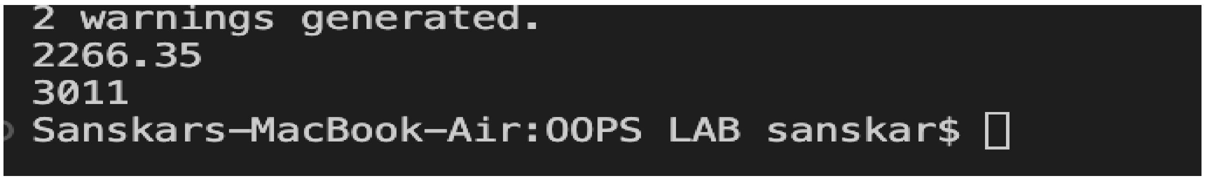
- They should have a different type
- They should have a different number
- They should have a different sequence of parameters.

CODE :



```cpp
#include<iostream>
#include<math.h>
using namespace std;
class motor{
    private:
        int rate = 15;
        int time = 2;
    public:
            int emi(int principle){
                int temp1 = pow((1+rate),time);
                int temp = ((principle * rate * temp1) / (temp1-1));
                return temp;
        }
            double emi(double principle){ int motor::time
                int temp1 = pow((1+rate),time);
                double temp = ((principle * rate * temp1) / (temp1-1));
                return temp;
        }
};
int main(){
    motor m;
    cout<< m.emi(150.50) <<endl;
    cout<< m.emi(200) <<endl;
    return 0;
    }
```

OUTPUT :



```
2 warnings generated.
2266.35
3011
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

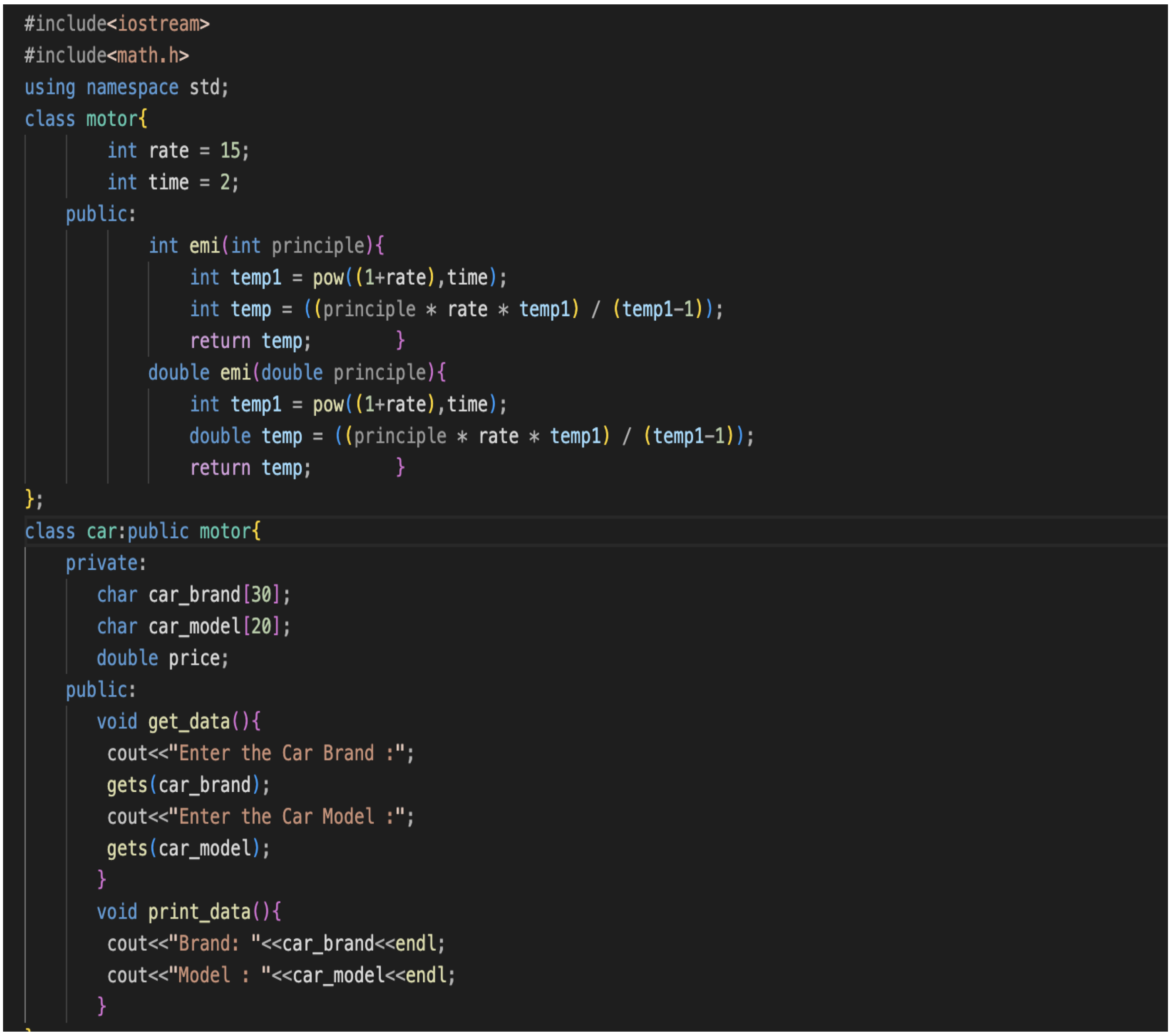
Function overloading with different parameters(different datatypes) and basic emi calculation has been done.

OBJECTIVE 8: WAP to implement Overriding using Motor (base class) and Car (derived class) and it should show Car details and calculate EMI (P=3Lacs , r=15% and t =2years)

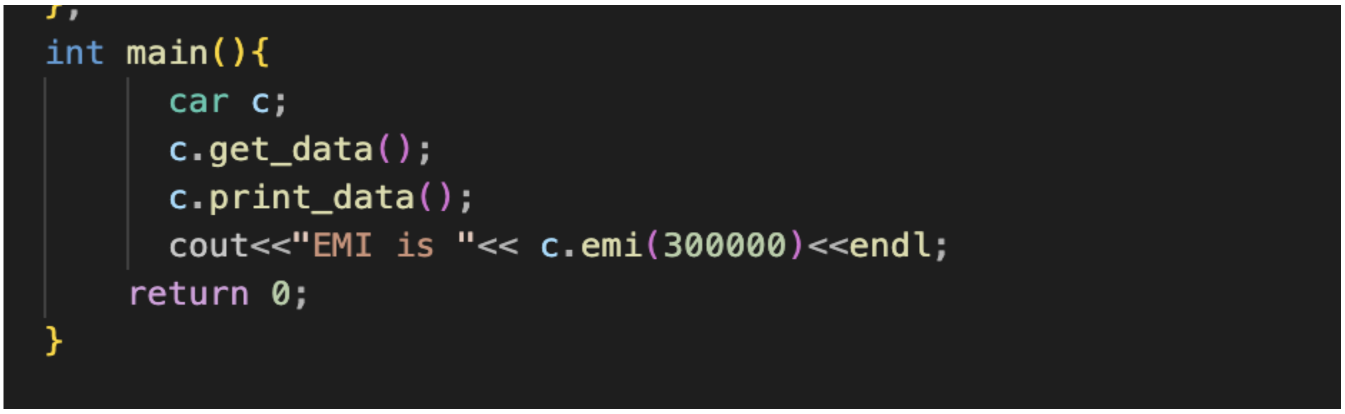
THEORY :

Function overriding in C++ is a feature that allows us to use a function in the child class that is already present in its parent class It is used to achieve runtime polymorphism.
The child class inherits all the data members, and the member functions present in the parent class. If you wish to override any functionality in the child class, then you can implement function overriding. Function overriding means creating a newer version of the parent class function in the child class.

CODE :

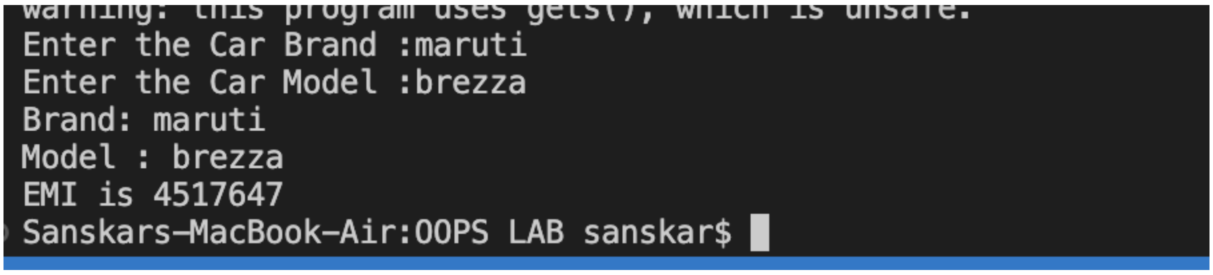


```
#include<iostream>
#include<math.h>
using namespace std;
class motor{
        int rate = 15;
        int time = 2;
    public:
            int emi(int principle){
                int temp1 = pow((1+rate),time);
                int temp = ((principle * rate * temp1) / (temp1-1));
                return temp;        }
            double emi(double principle){
                int temp1 = pow((1+rate),time);
                double temp = ((principle * rate * temp1) / (temp1-1));
                return temp;        }
};
class car:public motor{
    private:
       char car_brand[30];
       char car_model[20];
       double price;
    public:
       void get_data(){
        cout<<"Enter the Car Brand :";
        gets(car_brand);
        cout<<"Enter the Car Model :";
        gets(car_model);
       }
       void print_data(){
        cout<<"Brand: "<<car_brand<<endl;
        cout<<"Model : "<<car_model<<endl;
       }
```



```
int main(){
        car c;
        c.get_data();
        c.print_data();
        cout<<"EMI is "<< c.emi(300000)<<endl;
    return 0;
}
```

OUTPUT :



```
warning: this program uses gets(), which is unsafe.
Enter the Car Brand :maruti
Enter the Car Model :brezza
Brand: maruti
Model : brezza
EMI is 4517647
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Function overriding concept is applied and emi of a motor has been calculated easily.

OBJECTIVE 9: WAP to implement Encapsulation. Create class Cylinder and function to calculate Volume (be sure to protect variable volume from outside world.

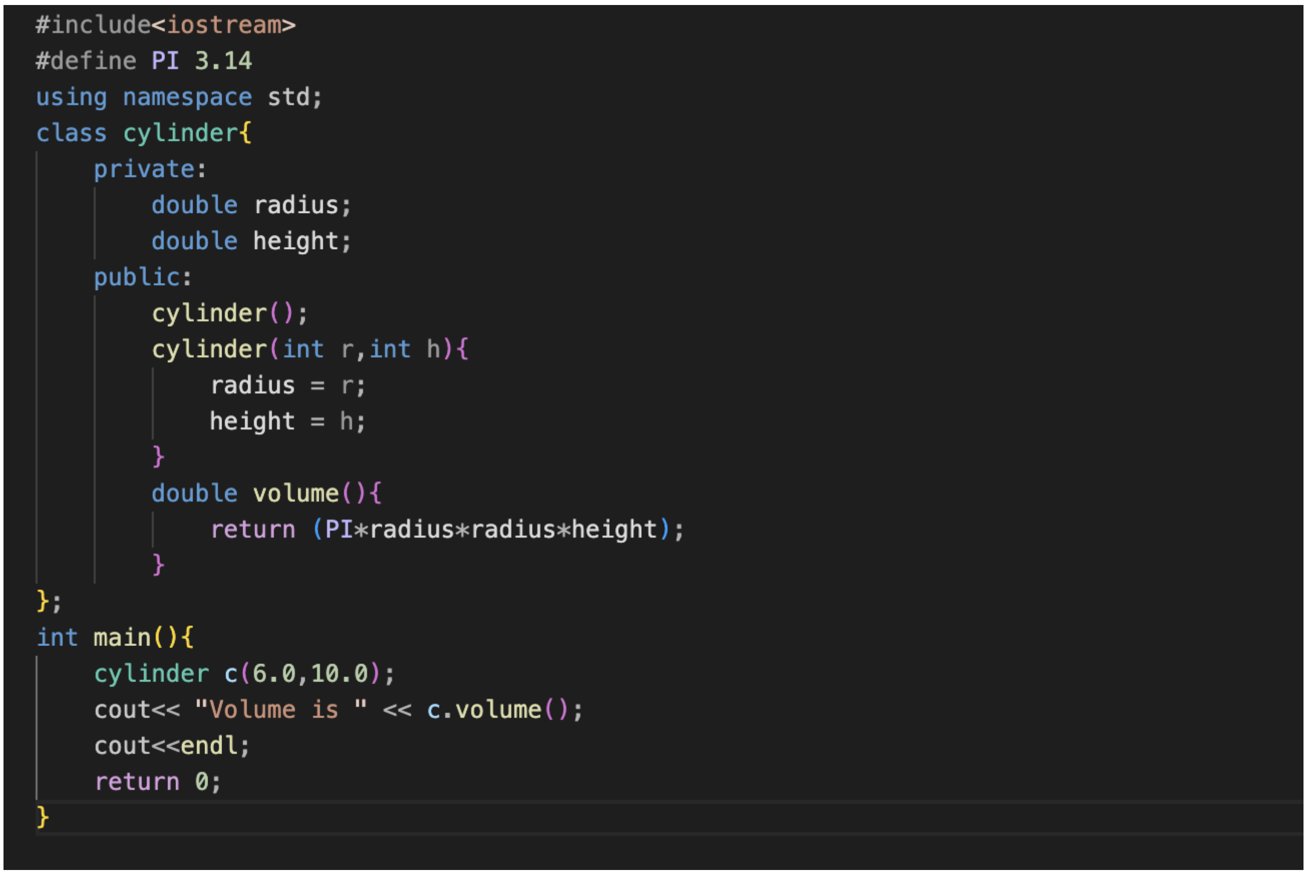
THEORY :

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user. C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called classes. We already have studied that a class can contain private, protected and public members. By default, all items defined in a class are private.

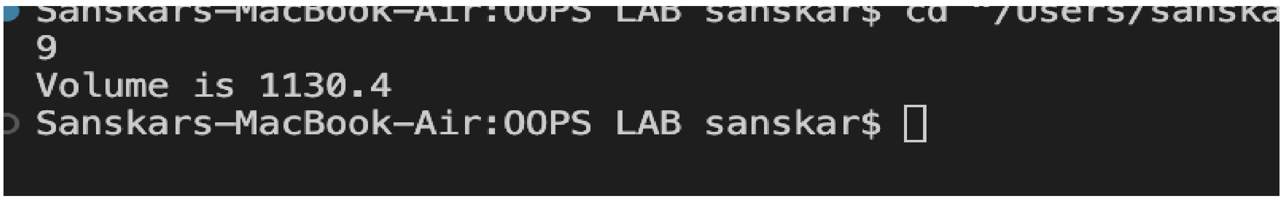
For example: Suppose you go to an automatic teller machine(ATM) and request money. The machine processes your request and gives you money. Here ATM is a class. It takes data from the user(money amount and PIN) and displays data in the form of icons and options. It processes the request(functions). So it contains both data and functions wrapped/integrated under a single ATM. This is called Encapsulation.

CODE :



```cpp
#include<iostream>
#define PI 3.14
using namespace std;
class cylinder{
    private:
        double radius;
        double height;
    public:
        cylinder();
        cylinder(int r,int h){
            radius = r;
            height = h;
        }
        double volume(){
            return (PI*radius*radius*height);
        }
};
int main(){
    cylinder c(6.0,10.0);
    cout<< "Volume is " << c.volume();
    cout<<endl;
    return 0;
}
```

OUTPUT :



```
Sanskars-MacBook-Air:OOPS LAB sanskar$ cd ~/Users/sanska
9
Volume is 1130.4
Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Encapsulation is achieved and learnt in this problem as the main data are private member of the class which can't be accessed by other functions outside class.

OBJECTIVE 10: WAP to implement constructors and destructors of a class. Create a Animal Class with attributes.(type=mammal, scavenger, arial, amphibians, aquatic etc and eating habit = carnivorous, herbivorous, etc)

THEORY :

Constructor and Destructor are the special member functions of the class which are created by the C++ compiler or can be defined by the user.

CONSTRUCTOR:

A constructor is a special member function of a class and shares the same name as of class, which means the constructor and class have the same name. Constructor is called by the compiler whenever the object of the class is created, it allocates the memory to the object and initializes class data members by default values or values passed by the user while creating an object. Constructors don't have any return type because their work is to just create and initialize an object.

There are four types of constructors in C++ :

Default constructor : Default constructor is also known as a zero-argument constructor, as it doesn't take any parameter.

Parameterized constructor : Parameterized constructor is used to initialize data members with the values provided by the user. This constructor is basically the upgraded version of the default constructor
Copy Constructor : If we have an object of a class and we want to create its copy in a new declared object of the same class, then a copy constructor is used.

Dynamic Constructor : When memory is allocated dynamically to the data members at the runtime using a new operator, the constructor is known as the dynamic constructor.

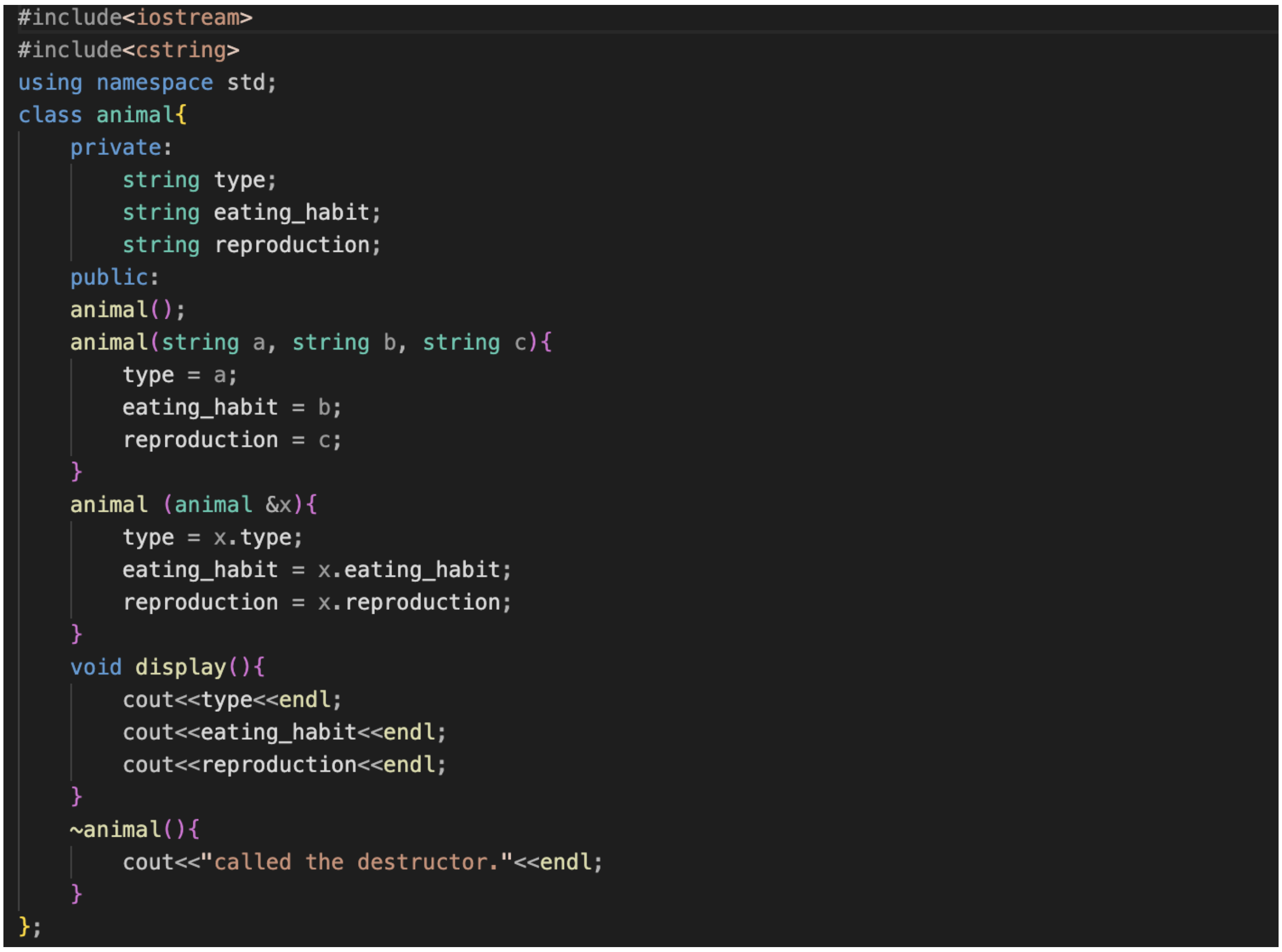
DESTRUCTOR :

Destructor is just the opposite function of the constructor. A destructor is called by the compiler when the object is destroyed and its main function is to deallocate the memory of the object. The object may be destroyed when the program ends, or local objects of the function get out of scope when the function ends or in any other case. Destructor has the same as of the class with prefix tilde(~) operator and it cannot be overloaded as the constructor. Destructors take no argument and have no return type and return value.

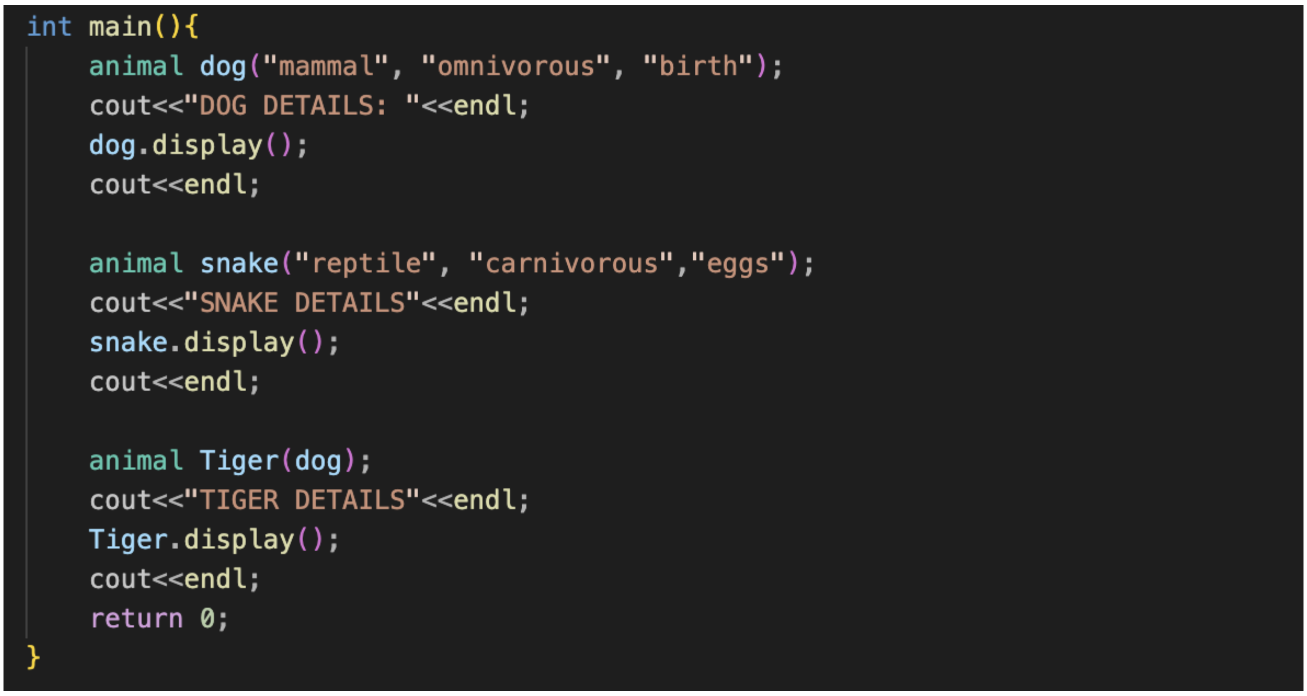
Important Points about the Destructor

- Destructor are the last member function called for an object and they are called by the compiler itself.
- If the destructor is not created by the user then compile creates or declares it by itself.
- A Destructor can be declared in any section of the class, as it is called by the compiler so nothing to worry about.
- As Destructor is the last function to be called, it should be better to declare it at the end of the class to increase the readability of the code.
- Destructor is just the opposite of the constructor as the constructor is called at the time of the creation of the object and allocates the memory to the object, on the other side the destructor is called at the time of the destruction of the object and deallocates the memory.

CODE :

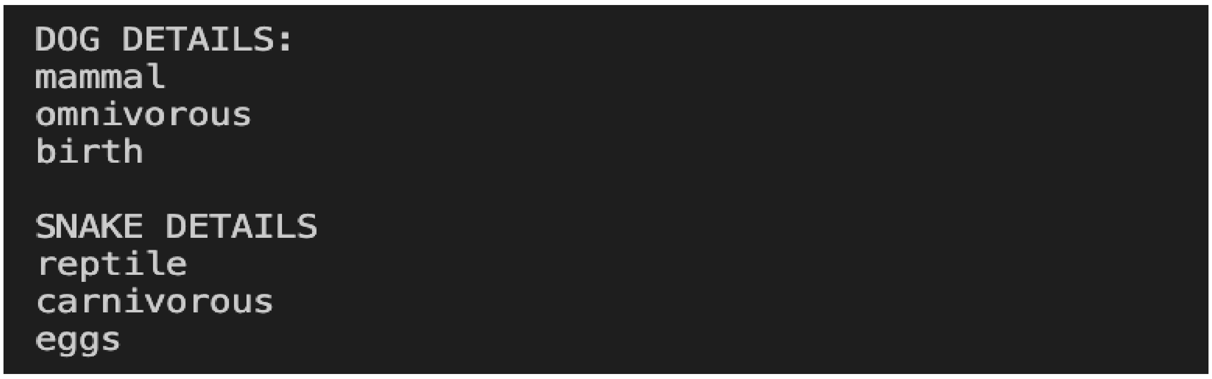


```cpp
#include<iostream>
#include<cstring>
using namespace std;
class animal{
    private:
        string type;
        string eating_habit;
        string reproduction;
    public:
    animal();
    animal(string a, string b, string c){
        type = a;
        eating_habit = b;
        reproduction = c;
    }
    animal (animal &x){
        type = x.type;
        eating_habit = x.eating_habit;
        reproduction = x.reproduction;
    }
    void display(){
        cout<<type<<endl;
        cout<<eating_habit<<endl;
        cout<<reproduction<<endl;
    }
    ~animal(){
        cout<<"called the destructor."<<endl;
    }
};
```

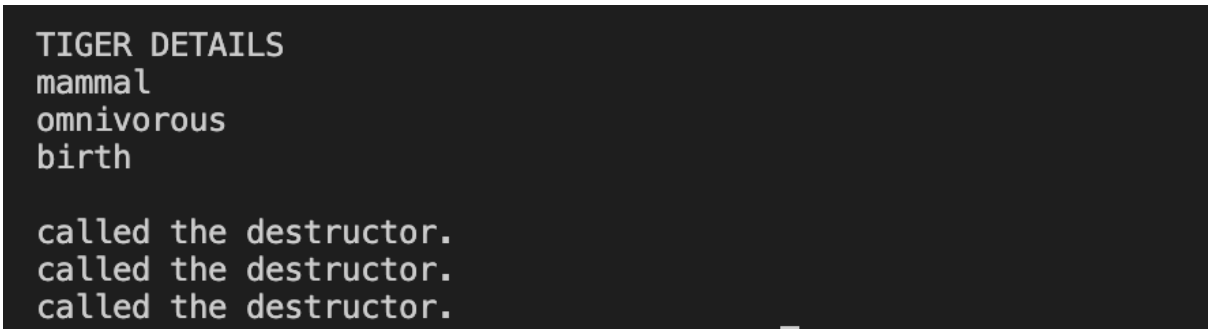


```cpp
int main(){
    animal dog("mammal", "omnivorous", "birth");
    cout<<"DOG DETAILS: "<<endl;
    dog.display();
    cout<<endl;

    animal snake("reptile", "carnivorous","eggs");
    cout<<"SNAKE DETAILS"<<endl;
    snake.display();
    cout<<endl;

    animal Tiger(dog);
    cout<<"TIGER DETAILS"<<endl;
    Tiger.display();
    cout<<endl;
    return 0;
}
```

OUTPUT :



```
DOG DETAILS:
mammal
omnivorous
birth

SNAKE DETAILS
reptile
carnivorous
eggs
```



```
TIGER DETAILS
mammal
omnivorous
birth

called the destructor.
called the destructor.
called the destructor.
```

LEARNING OUTCOME :

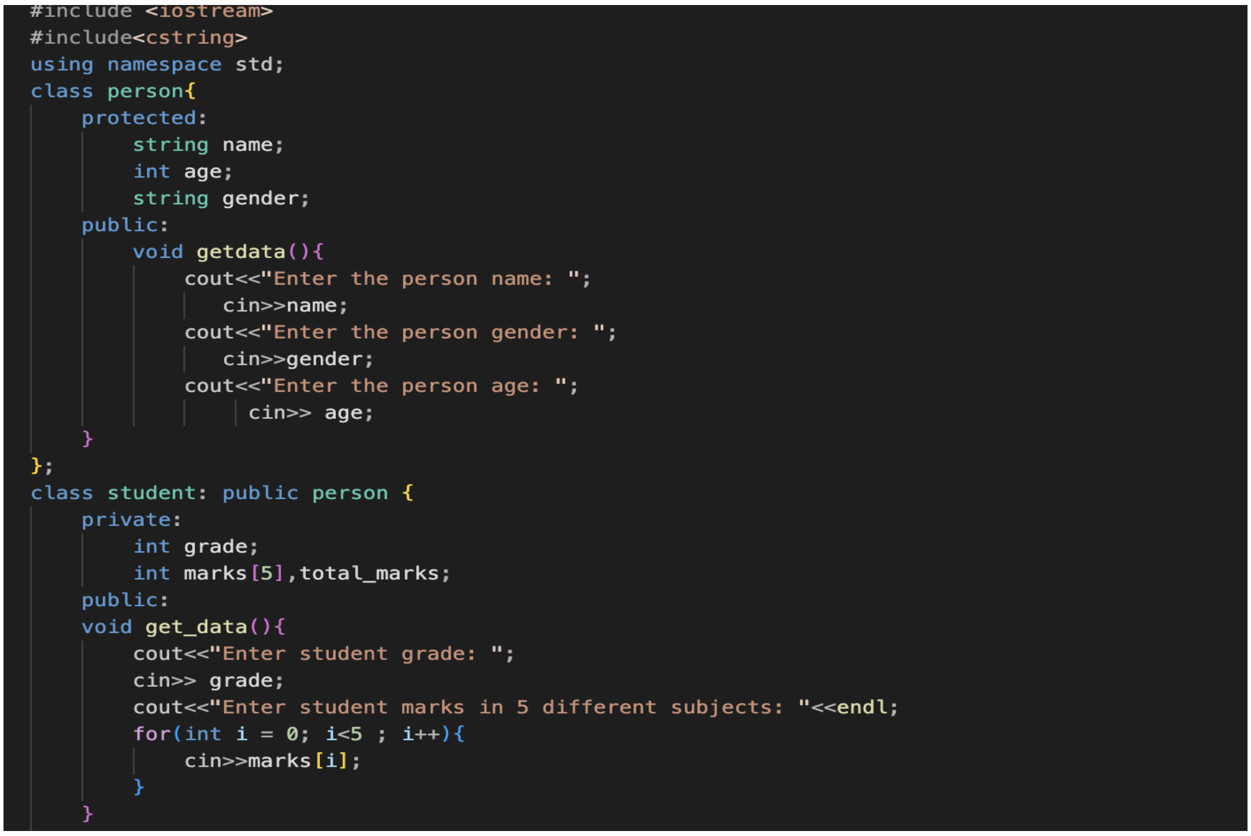
Basic constructors and destructors has been learnt in this problem. Default constructor, parameterized constructor and copy constructor has been used along with destructor.

OBJECTIVE 11: WAP to implement Generalization as extension. Use class person with name, age and gender as variables. Extend it by using subclass (Student, Teacher and Admin Staff)

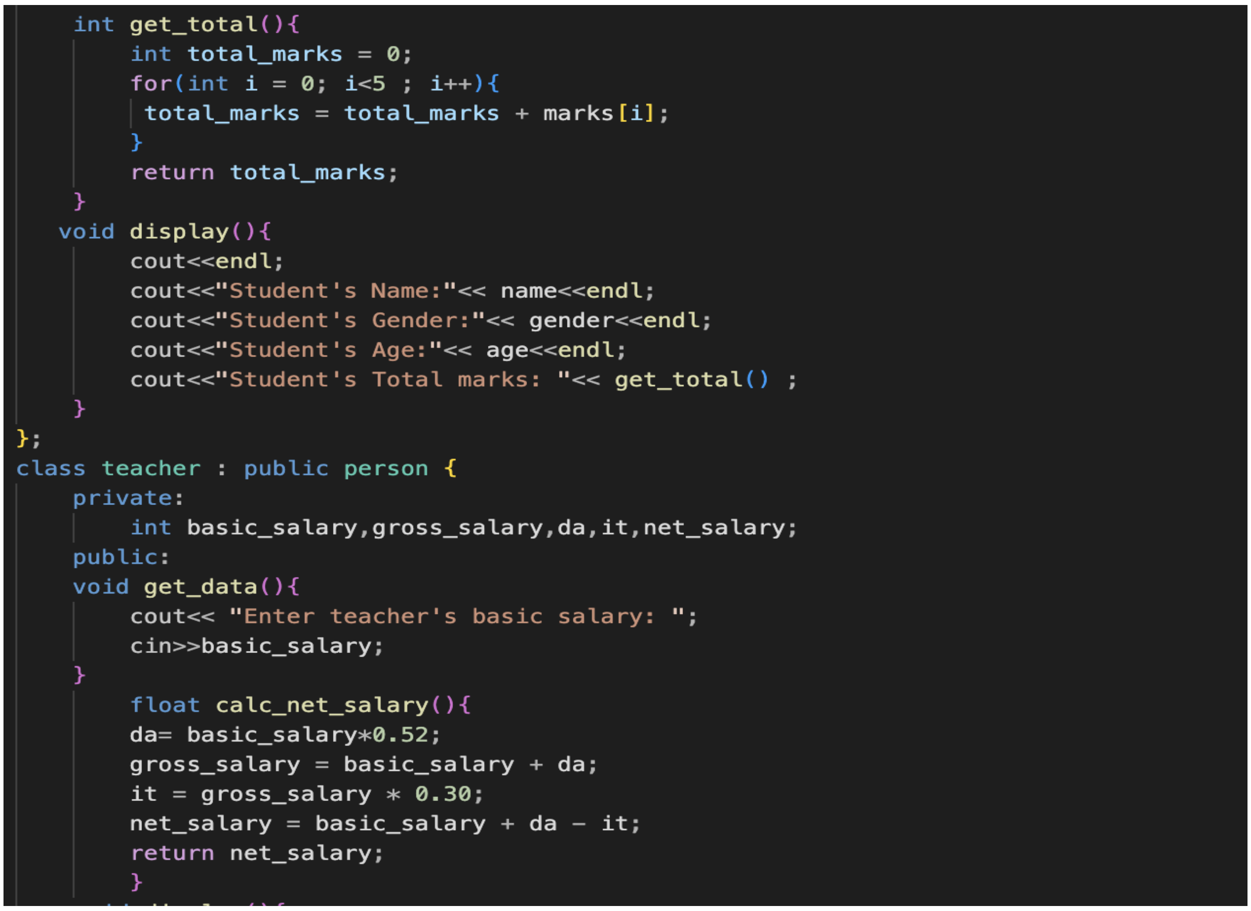
THEORY :

Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass. Shared characteristics can be attributes, associations, or methods. a generalization relationship is a relationship that implements the concept of object orientation called inheritance. The generalization relationship occurs between two entities or objects, such that one entity is the parent, and the other one is the child. The child inherits the functionality of its parent and can access as well as update it.

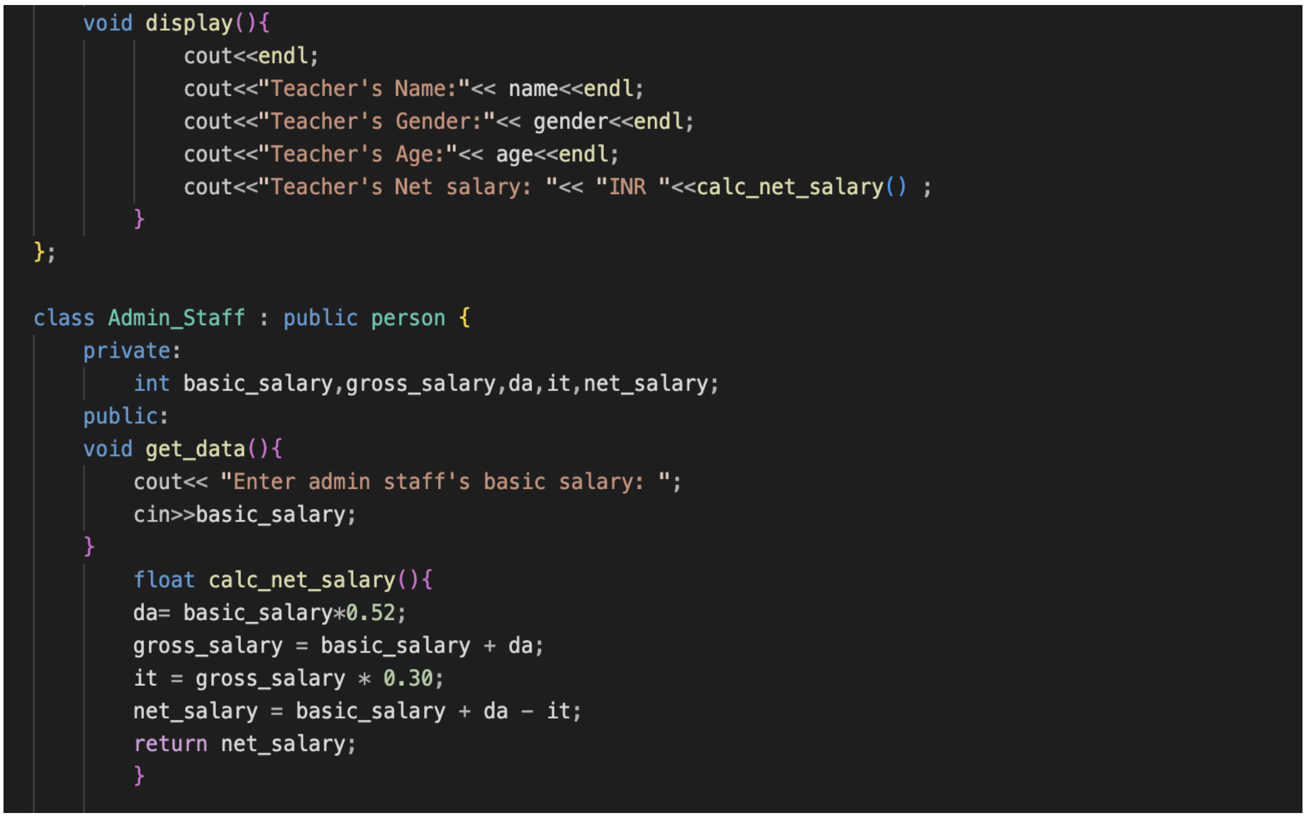
CODE :



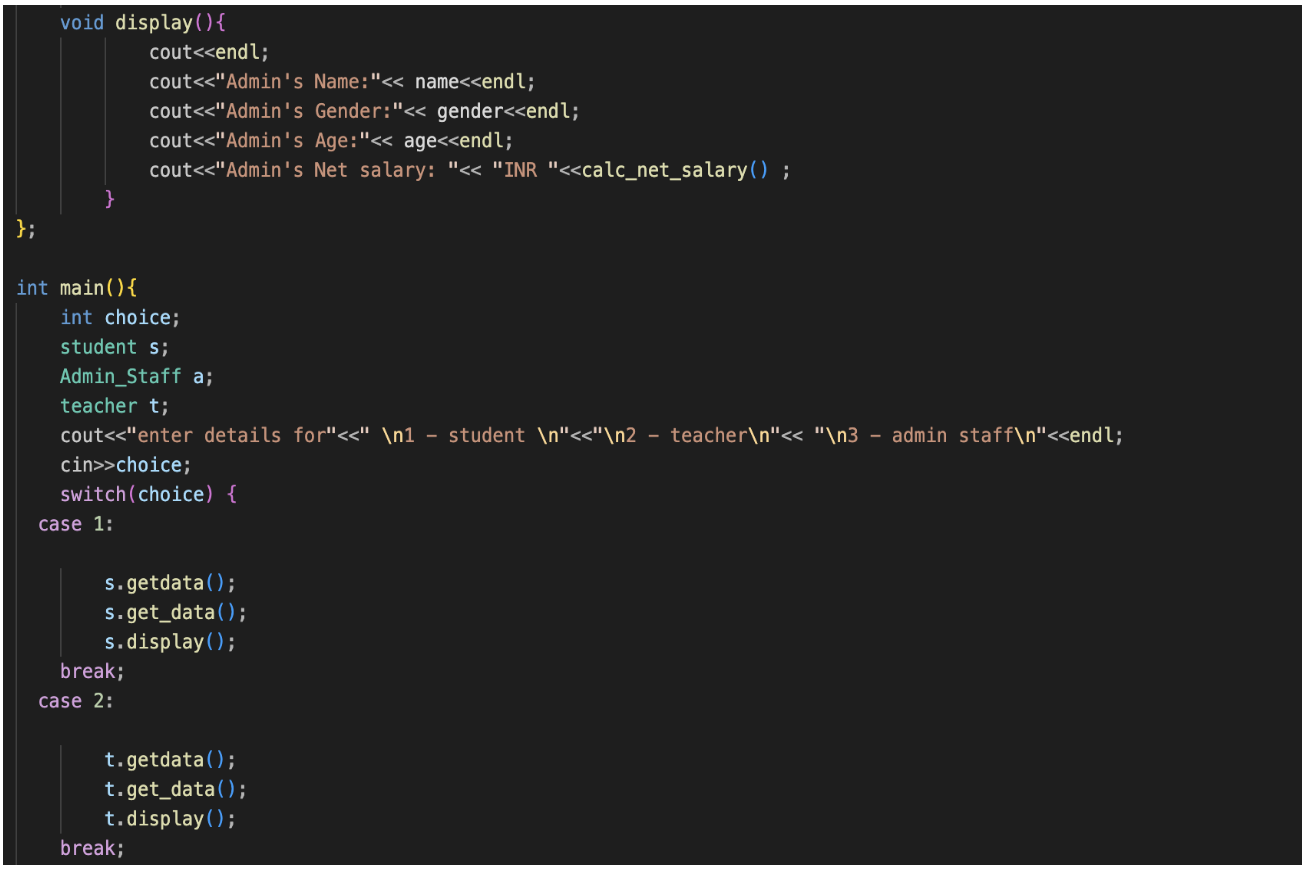
```
#include <iostream>
#include<cstring>
using namespace std;
class person{
    protected:
        string name;
        int age;
        string gender;
    public:
        void getdata(){
            cout<<"Enter the person name: ";
               cin>>name;
            cout<<"Enter the person gender: ";
               cin>>gender;
            cout<<"Enter the person age: ";
                 cin>> age;
    }
};
class student: public person {
    private:
        int grade;
        int marks[5],total_marks;
    public:
    void get_data(){
        cout<<"Enter student grade: ";
        cin>> grade;
        cout<<"Enter student marks in 5 different subjects: "<<endl;
        for(int i = 0; i<5 ; i++){
            cin>>marks[i];
        }
    }
```



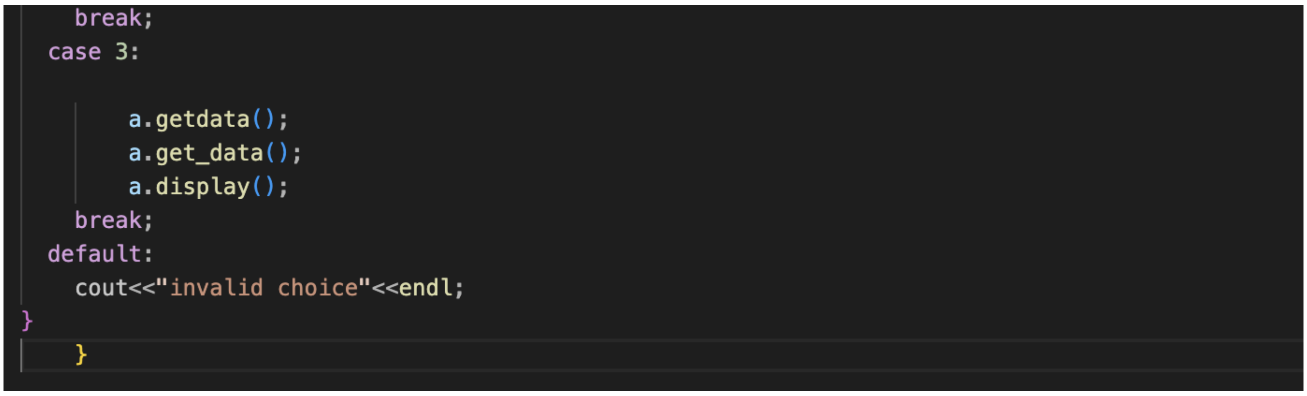
```
    int get_total(){
        int total_marks = 0;
        for(int i = 0; i<5 ; i++){
         total_marks = total_marks + marks[i];
        }
        return total_marks;
    }
   void display(){
        cout<<endl;
        cout<<"Student's Name:"<< name<<endl;
        cout<<"Student's Gender:"<< gender<<endl;
        cout<<"Student's Age:"<< age<<endl;
        cout<<"Student's Total marks: "<< get_total() ;
    }
};
class teacher : public person {
    private:
        int basic_salary,gross_salary,da,it,net_salary;
    public:
    void get_data(){
        cout<< "Enter teacher's basic salary: ";
        cin>>basic_salary;
    }
        float calc_net_salary(){
        da= basic_salary*0.52;
        gross_salary = basic_salary + da;
        it = gross_salary * 0.30;
        net_salary = basic_salary + da - it;
        return net_salary;
        }
```



```cpp
        void display(){
                cout<<endl;
                cout<<"Teacher's Name:"<< name<<endl;
                cout<<"Teacher's Gender:"<< gender<<endl;
                cout<<"Teacher's Age:"<< age<<endl;
                cout<<"Teacher's Net salary: "<< "INR "<<calc_net_salary() ;
            }
    };

    class Admin_Staff : public person {
        private:
            int basic_salary,gross_salary,da,it,net_salary;
        public:
        void get_data(){
            cout<< "Enter admin staff's basic salary: ";
            cin>>basic_salary;
        }
            float calc_net_salary(){
            da= basic_salary*0.52;
            gross_salary = basic_salary + da;
            it = gross_salary * 0.30;
            net_salary = basic_salary + da - it;
            return net_salary;
            }
```

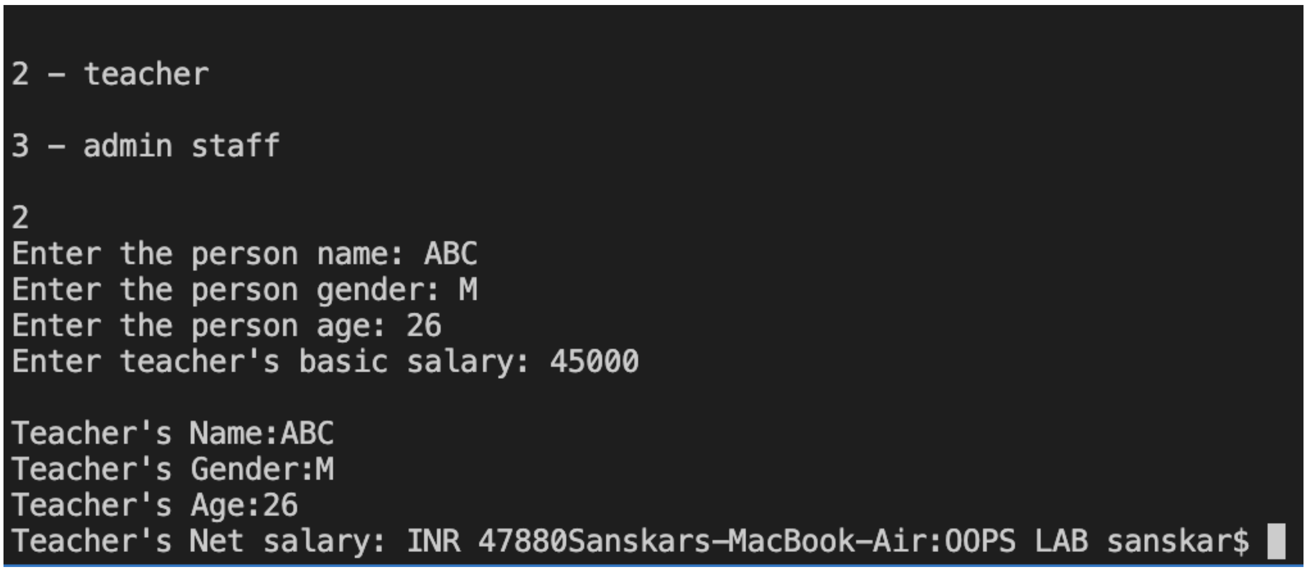


```cpp
    void display(){
            cout<<endl;
            cout<<"Admin's Name:"<< name<<endl;
            cout<<"Admin's Gender:"<< gender<<endl;
            cout<<"Admin's Age:"<< age<<endl;
            cout<<"Admin's Net salary: "<< "INR "<<calc_net_salary() ;
        }
};

int main(){
    int choice;
    student s;
    Admin_Staff a;
    teacher t;
    cout<<"enter details for"<<" \n1 - student \n"<<"\n2 - teacher\n"<< "\n3 - admin staff\n"<<endl;
    cin>>choice;
    switch(choice) {
  case 1:

        s.getdata();
        s.get_data();
        s.display();
    break;
  case 2:

        t.getdata();
        t.get_data();
        t.display();
    break;
```



```
        break;
    case 3:

            a.getdata();
            a.get_data();
            a.display();
        break;
    default:
        cout<<"invalid choice"<<endl;
}
        }
```

OUTPUT :



```
2 - teacher

3 - admin staff

2
Enter the person name: ABC
Enter the person gender: M
Enter the person age: 26
Enter teacher's basic salary: 45000

Teacher's Name:ABC
Teacher's Gender:M
Teacher's Age:26
Teacher's Net salary: INR 47880Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Generalization which is an extension to inheritance is learnt along with codes. The basic of generalization along with extension is achieved.

OBJECTIVE 12: WAP to implement friend function in Java or C++. Use class Sphere with radius as variable and friend function cylinder (with given height) and calculate the Volume of both.

THEORY :

Friend functions of the class are granted permission to access private and protected members of the class of C++. They are defined globally outside the class scope. Friend functions are not member functions of the class. So, what exactly is the friend function. A friend function in C++ is a function that is declared outside a class but is capable of accessing the private and protected members of the class. There could be situations in programming wherein we want two classes to share their members. These members may be data members, class functions or function templates. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class.

Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

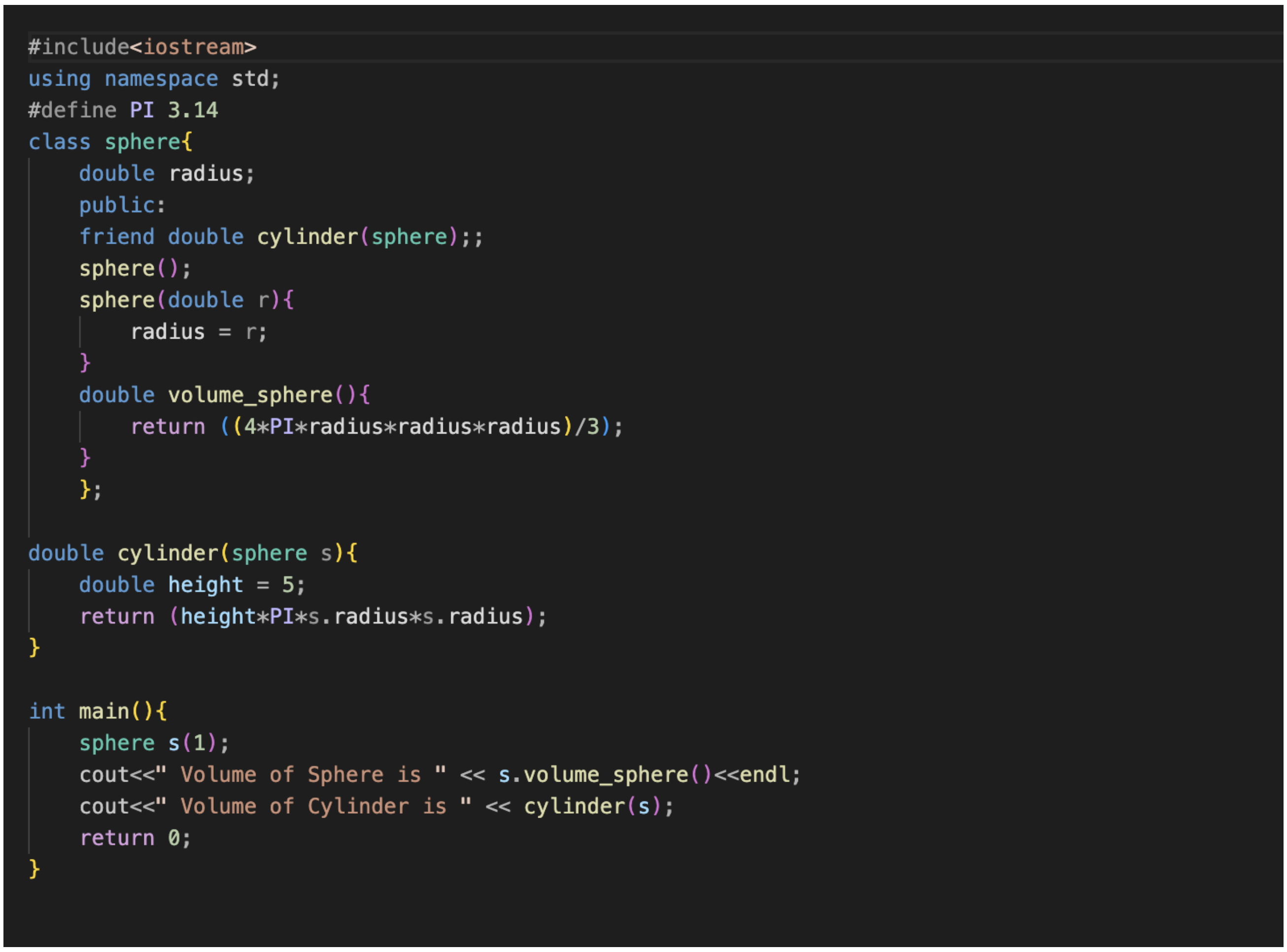
Friend functions in C++ have the following types

- Function with no argument and no return value
- Function with no argument but with return value
- Function with argument but no return value
- Function with argument and return value

Use of Friend function in C++ :

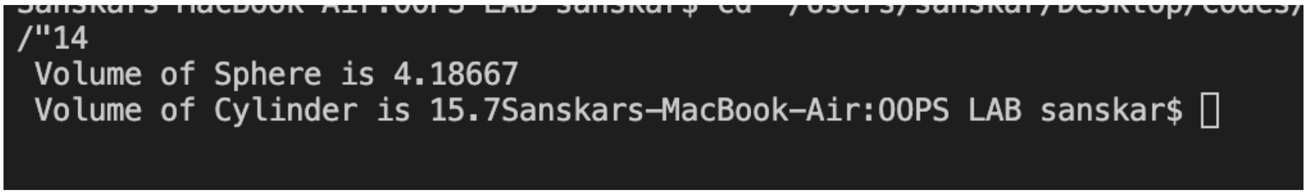
we require friend functions whenever we have to access the private or protected members of a class. This is only the case when we do not want to use the objects of that class to access these private or protected members.

CODE :



```cpp
#include<iostream>
using namespace std;
#define PI 3.14
class sphere{
    double radius;
    public:
    friend double cylinder(sphere);;
    sphere();
    sphere(double r){
        radius = r;
    }
    double volume_sphere(){
        return ((4*PI*radius*radius*radius)/3);
    }
    };

double cylinder(sphere s){
    double height = 5;
    return (height*PI*s.radius*s.radius);
}

int main(){
    sphere s(1);
    cout<<" Volume of Sphere is " << s.volume_sphere()<<endl;
    cout<<" Volume of Cylinder is " << cylinder(s);
    return 0;
}
```

OUTPUT :



```
Sanskars-MacBook-Air:OOPS LAB sanskar$ cd "/Users/sanskar/Desktop/codes/
/"14
 Volume of Sphere is 4.18667
 Volume of Cylinder is 15.7Sanskars-MacBook-Air:OOPS LAB sanskar$ 
```

LEARNING OUTCOME :

Basic of friend function and implementation of friend function is learnt along with its necessity to create friend function which solves different problems in code.