DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi- 110042

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OBJECT ORIENTED PROGRAMMING LAB FILE

CO-203

Submitted By:
Sandesh Shrestha
2K21/CO/417
A6 batch

Submitted To:
Mr. Aman Kumar Pandey
Department of COE

INDEX: LIST OF PROGRAMS

S.No.	Program Name	Date	Remarks
1)	WAP to create a class, object and calculate salary of Employees.	07/09/22	
2)	WAP to implement Call of Reference and reverse an array using Swap function.	07/09/22	
3)	WAP to Calculate the area of Circle and Square.	14/09/22	
4)	WAP to implement Single, Multiple and Multi-level Inheritance.	14/09/22	
5)	WAP to implement Polymorphism by using Sum function of 2 and 3 variables.	21/09/22	
6)	WAP to implement Abstraction using Calc Class and perform different operations (add, subtract, multiply and divide).	21/09/22	
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.	28/09/22	
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)	28/09/22	
9)	WAP to implement Encapsulation. Create class Cylinder and function to calculate Volume (be sure to protect variable volume from outside world.)	19/10/22	

10)	WAP to implement constructors and destructors of a class. Create a Animal Class with its attributes. (type= mammal, scavenger, arial, amphibians, aquatic, etc. and eating habit= carnivorous, herbivorous, etc.)	19/10/22	
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.	02/11/22	
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.	02/11/22	

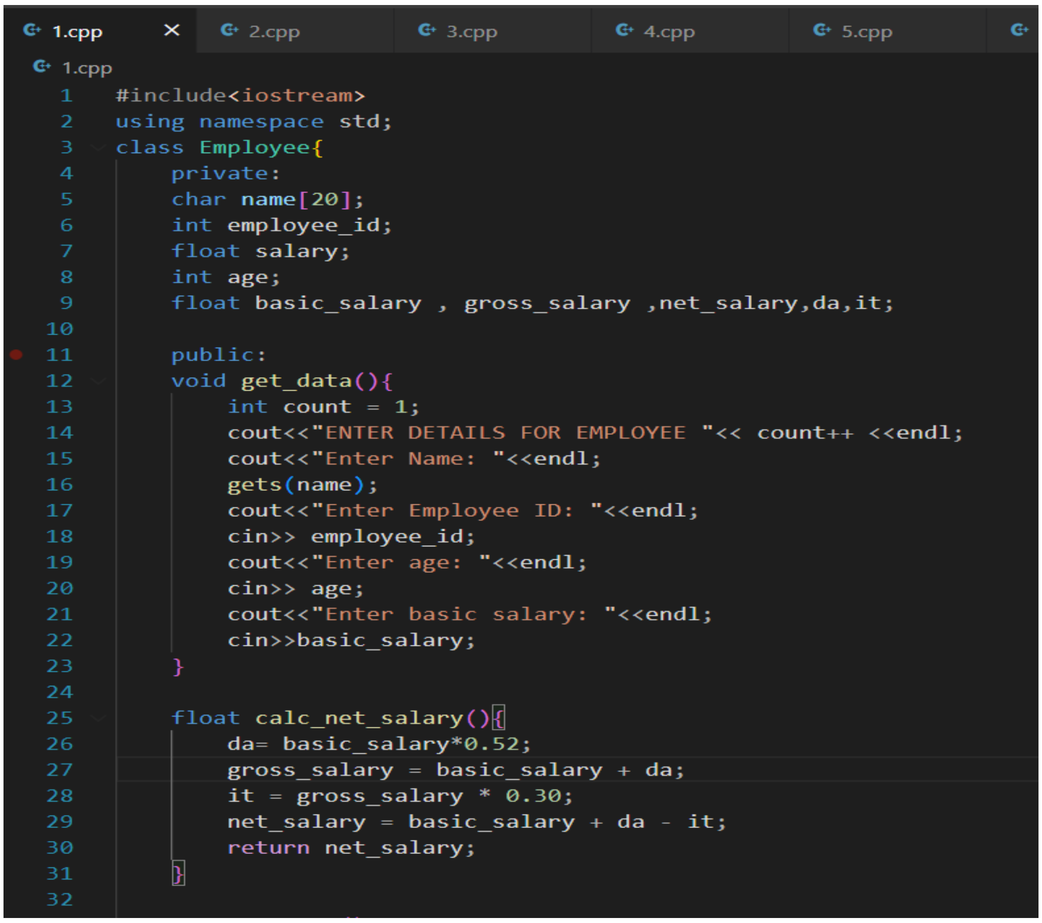
Program 1

Program Objective: WAP to create a class, object and calculate salary of Employees.

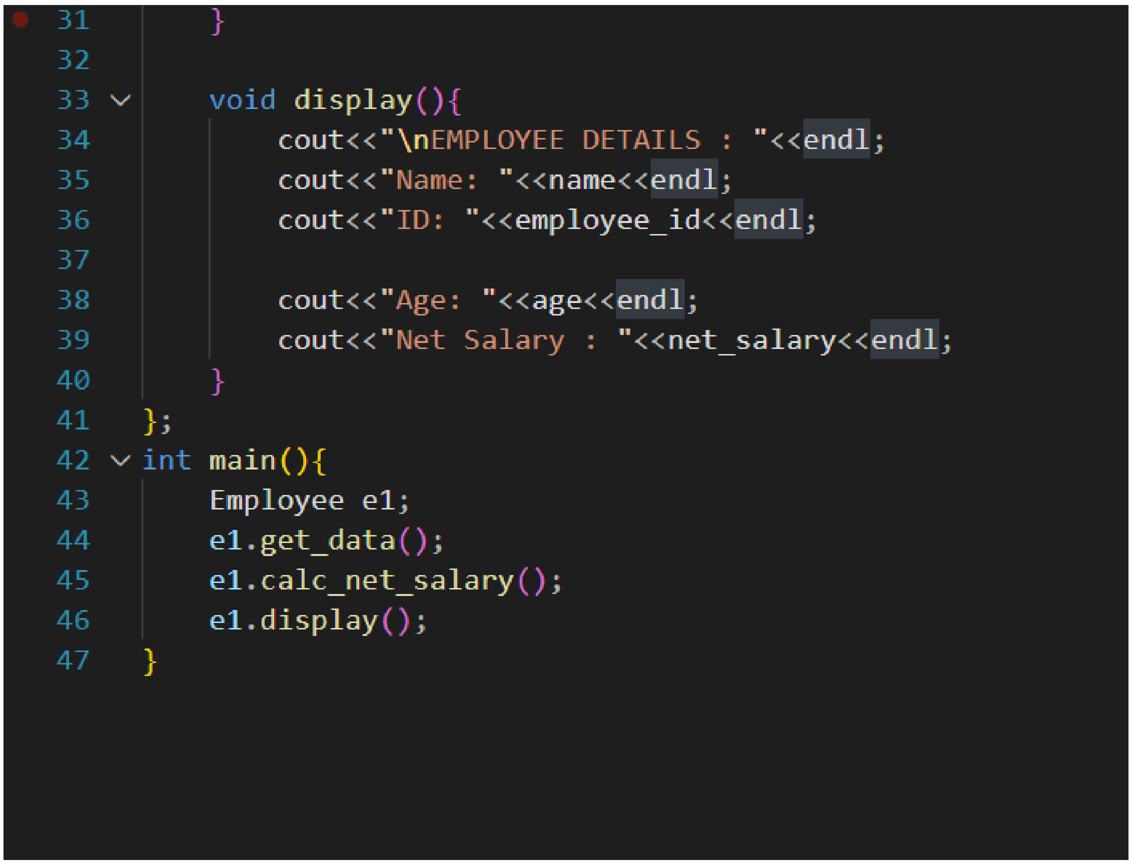
Program theory: Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions as members. A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.An object is an instantiation of a class. An object is an instance of a class. In simple words, we can say that an object is a variable of type class. Classes are defined using either keyword class or keyword struct, with the following **syntax**:

class **class_name** {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
} **object_names** ;

Program Code:

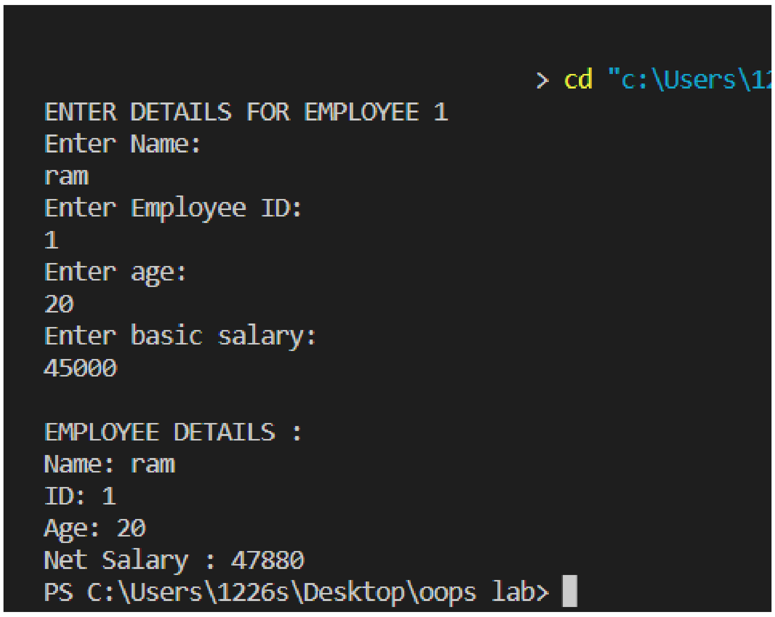


```cpp
#include<iostream>
using namespace std;
class Employee{
    private:
    char name[20];
    int employee_id;
    float salary;
    int age;
    float basic_salary , gross_salary ,net_salary,da,it;

    public:
    void get_data(){
        int count = 1;
        cout<<"ENTER DETAILS FOR EMPLOYEE "<< count++ <<endl;
        cout<<"Enter Name: "<<endl;
        gets(name);
        cout<<"Enter Employee ID: "<<endl;
        cin>> employee_id;
        cout<<"Enter age: "<<endl;
        cin>> age;
        cout<<"Enter basic salary: "<<endl;
        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<<"\nEMPLOYEE DETAILS : "<<endl;
        cout<<"Name: "<<name<<endl;
        cout<<"ID: "<<employee_id<<endl;

        cout<<"Age: "<<age<<endl;
        cout<<"Net Salary : "<<net_salary<<endl;
    }
};
int main(){
    Employee e1;
    e1.get_data();
    e1.calc_net_salary();
    e1.display();
}
```

Program Output:



```
                                          > cd "c:\Users\1
ENTER DETAILS FOR EMPLOYEE 1
Enter Name:
ram
Enter Employee ID:
1
Enter age:
20
Enter basic salary:
45000

EMPLOYEE DETAILS :
Name: ram
ID: 1
Age: 20
Net Salary : 47880
PS C:\Users\1226s\Desktop\oops lab>
```

Learning Outcome: The foundation of OOPS i.e., basic concept of class and object has been learnt for example: how to declare class and how to create objects of that class and perform various operations.

Program 2

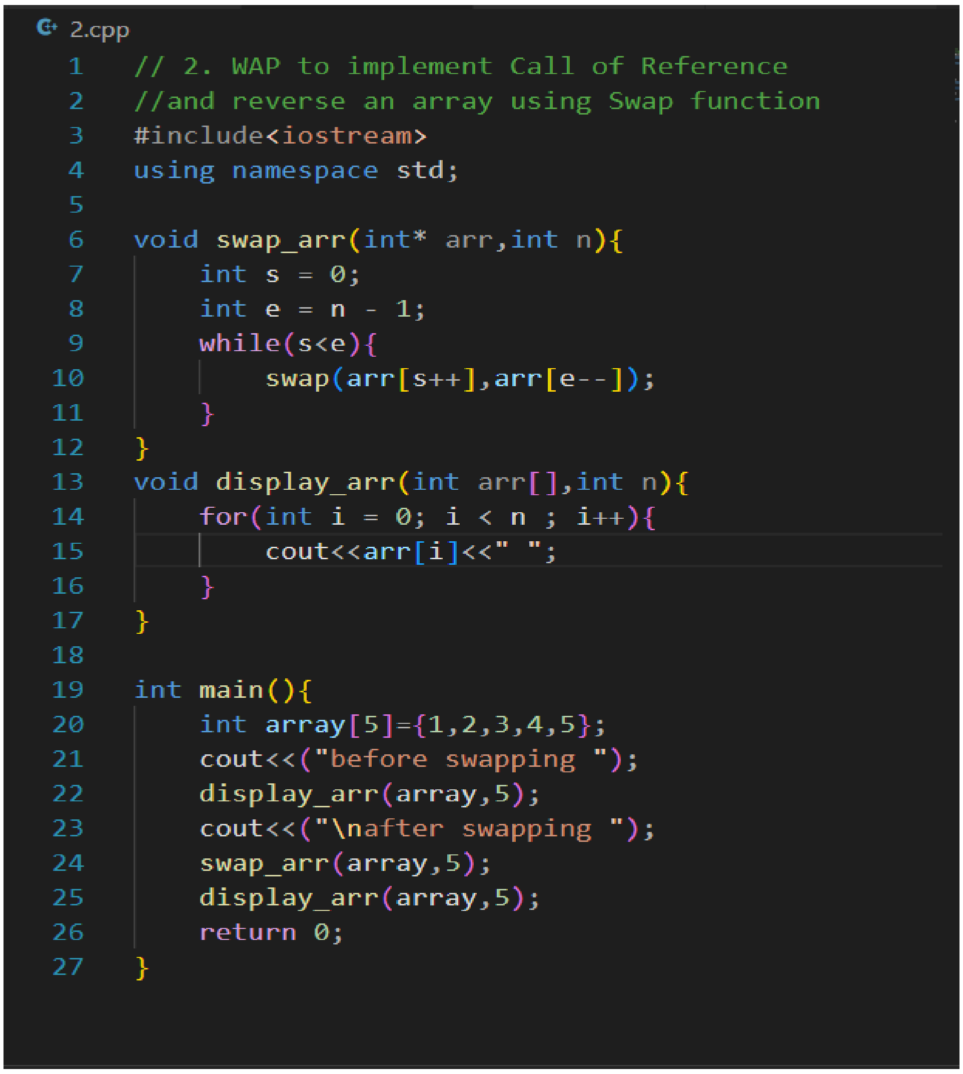
Program Objective: WAP to implement Call of Reference and reverse an array using Swap function.

Program theory: An array is a collection of items of same data type stored at contiguous memory locations. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). The base value is index 0 and the difference between the two indexes is the offset.

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.

To pass the value by reference, argument reference is passed to the functions just like any other value. So accordingly, you need to declare the function parameters as reference types.

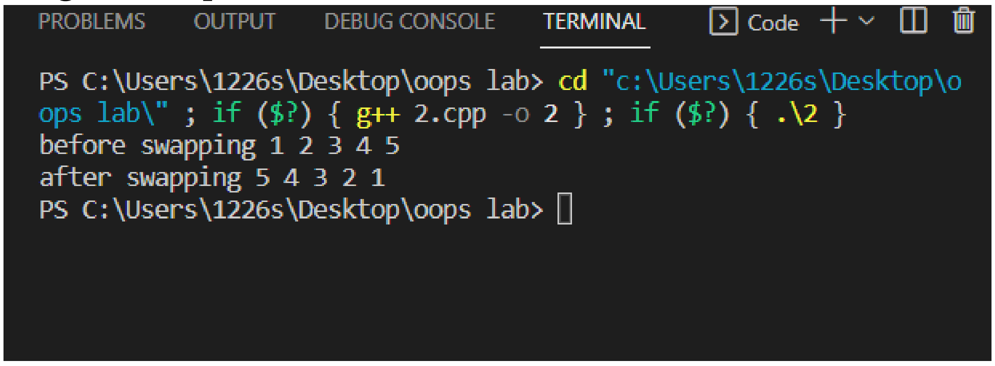
Program Code:



2.cpp

```cpp
// 2. WAP to implement Call of Reference
//and reverse an array using Swap function
#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;
}
```

Program Output:



```
PS C:\Users\1226s\Desktop\oops lab> cd "c:\Users\1226s\Desktop\o
ops lab\" ; if ($?) { g++ 2.cpp -o 2 } ; if ($?) { .\2 }
before swapping 1 2 3 4 5
after swapping 5 4 3 2 1
PS C:\Users\1226s\Desktop\oops lab> 
```

Learning Outcome:

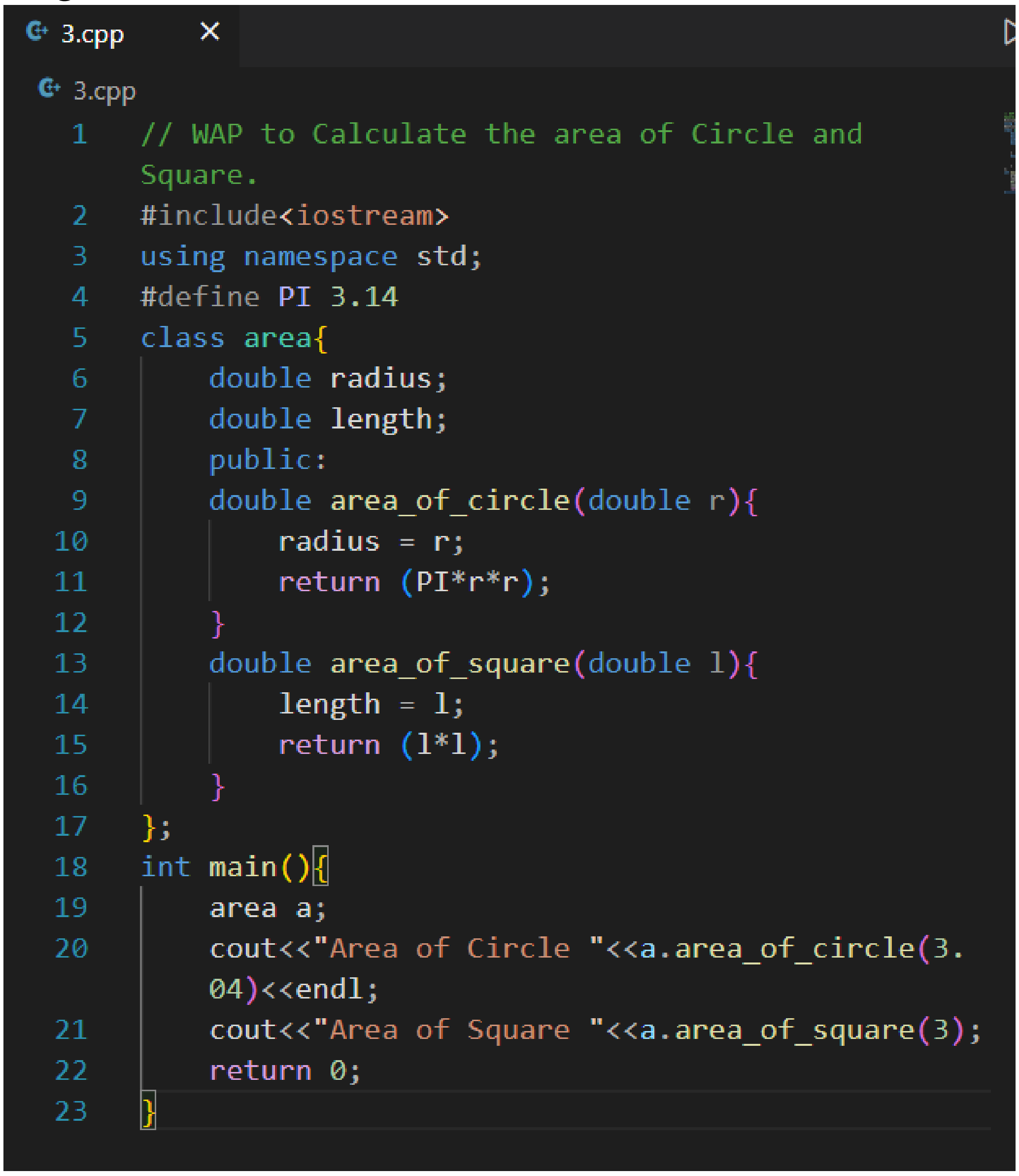
Basic concepts of array reversal using swap functions along with passing of array to a function using call by reference.

Program 3

Program Objective: WAP to Calculate the area of Circle and Square.

Program theory: Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions as members. A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects. 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.
Area of circle: PI * radius ^ 2
Area of square: length ^ 2

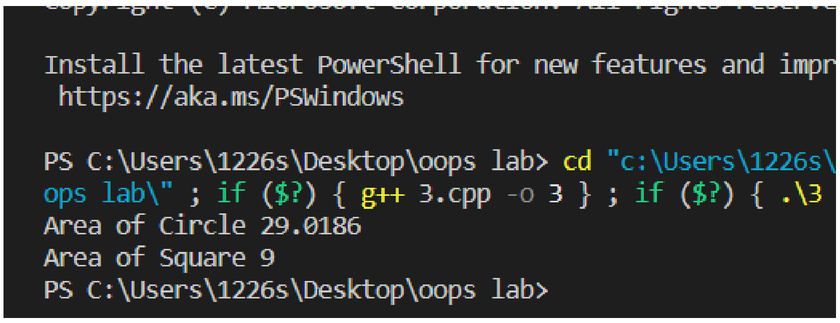
Program Code:



3.cpp

```cpp
// WAP to Calculate the area of Circle and Square.
#include<iostream>
using namespace std;
#define PI 3.14
class area{
    double radius;
    double length;
    public:
    double area_of_circle(double r){
        radius = r;
        return (PI*r*r);
    }
    double area_of_square(double l){
        length = l;
        return (l*l);
    }
};
int main(){
    area a;
    cout<<"Area of Circle "<<a.area_of_circle(3.04)<<endl;
    cout<<"Area of Square "<<a.area_of_square(3);
    return 0;
}
```

Program Output:



```
Install the latest PowerShell for new features and impr
 https://aka.ms/PSWindows

PS C:\Users\1226s\Desktop\oops lab> cd "c:\Users\1226s\
ops lab\" ; if ($?) { g++ 3.cpp -o 3 } ; if ($?) { .\3
Area of Circle 29.0186
Area of Square 9
PS C:\Users\1226s\Desktop\oops lab>
```

Learning Outcome:

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

Program 4

Program Objective: WAP to implement Single, Multiple and Multi-level Inheritance.

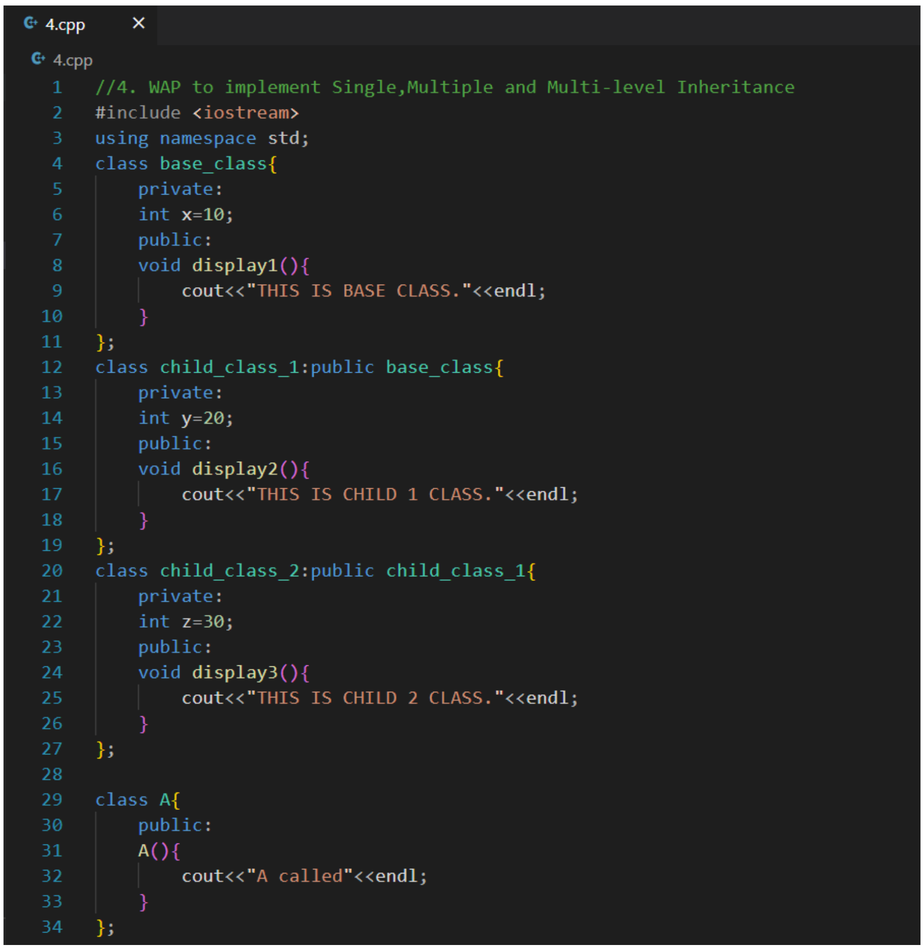
Program theory: The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class. **Types Of Inheritance: -**

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

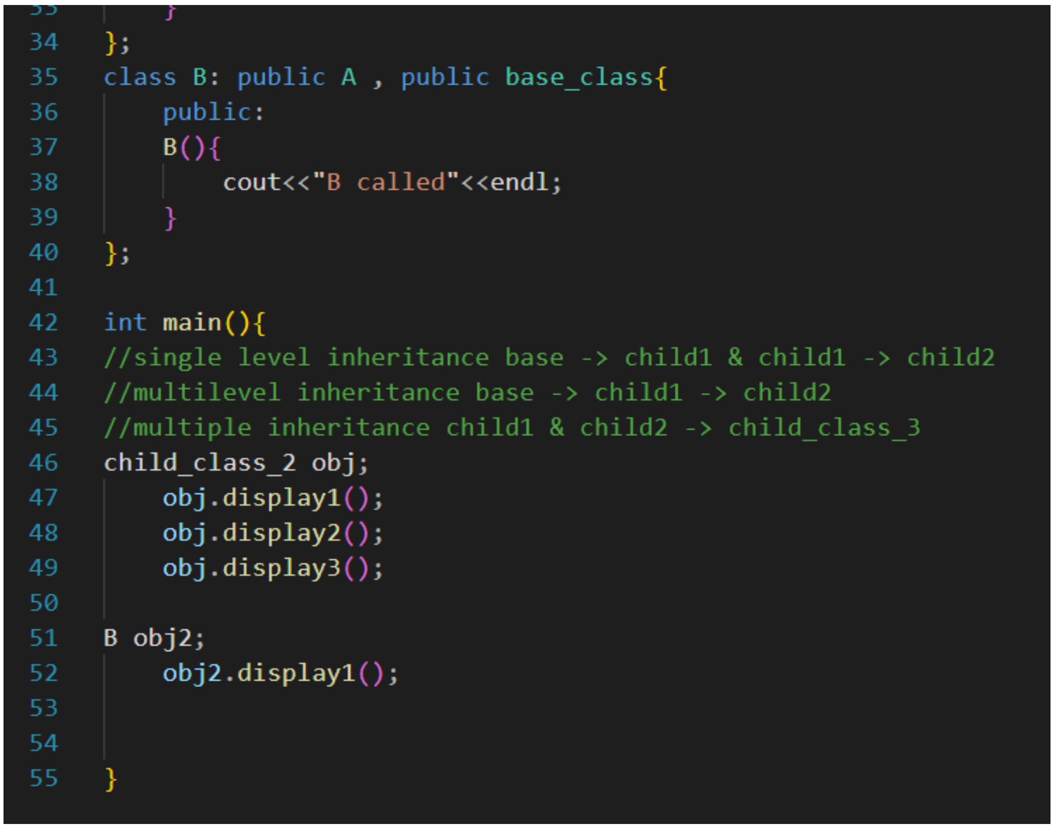
Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e., one subclass is inherited by one base class only. When one class inherits another class, it is known as single level inheritance.

Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e., one subclass is inherited from more than one base class. Multilevel Inheritance: A derived class is created from another derived class

Program Code:

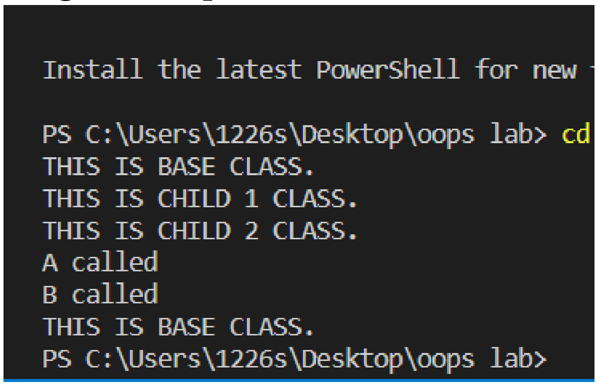


```cpp
//4. WAP to implement Single,Multiple and Multi-level Inheritance
#include <iostream>
using namespace std;
class base_class{
    private:
    int x=10;
    public:
    void display1(){
        cout<<"THIS IS BASE CLASS."<<endl;
    }
};
class child_class_1:public base_class{
    private:
    int y=20;
    public:
    void display2(){
        cout<<"THIS IS CHILD 1 CLASS."<<endl;
    }
};
class child_class_2:public child_class_1{
    private:
    int z=30;
    public:
    void display3(){
        cout<<"THIS IS CHILD 2 CLASS."<<endl;
    }
};

class A{
    public:
    A(){
        cout<<"A called"<<endl;
    }
};
```



```
    }
};
class B: public A , public base_class{
    public:
    B(){
        cout<<"B called"<<endl;
    }
};

int main(){
//single level inheritance base -> child1 & child1 -> child2
//multilevel inheritance base -> child1 -> child2
//multiple inheritance child1 & child2 -> child_class_3
child_class_2 obj;
    obj.display1();
    obj.display2();
    obj.display3();

B obj2;
    obj2.display1();


}
```

Program Output:



```
Install the latest PowerShell for new

PS C:\Users\1226s\Desktop\oops lab> cd
THIS IS BASE CLASS.
THIS IS CHILD 1 CLASS.
THIS IS CHILD 2 CLASS.
A called
B called
THIS IS BASE CLASS.
PS C:\Users\1226s\Desktop\oops lab>
```

Learning Outcome:

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

Program 5

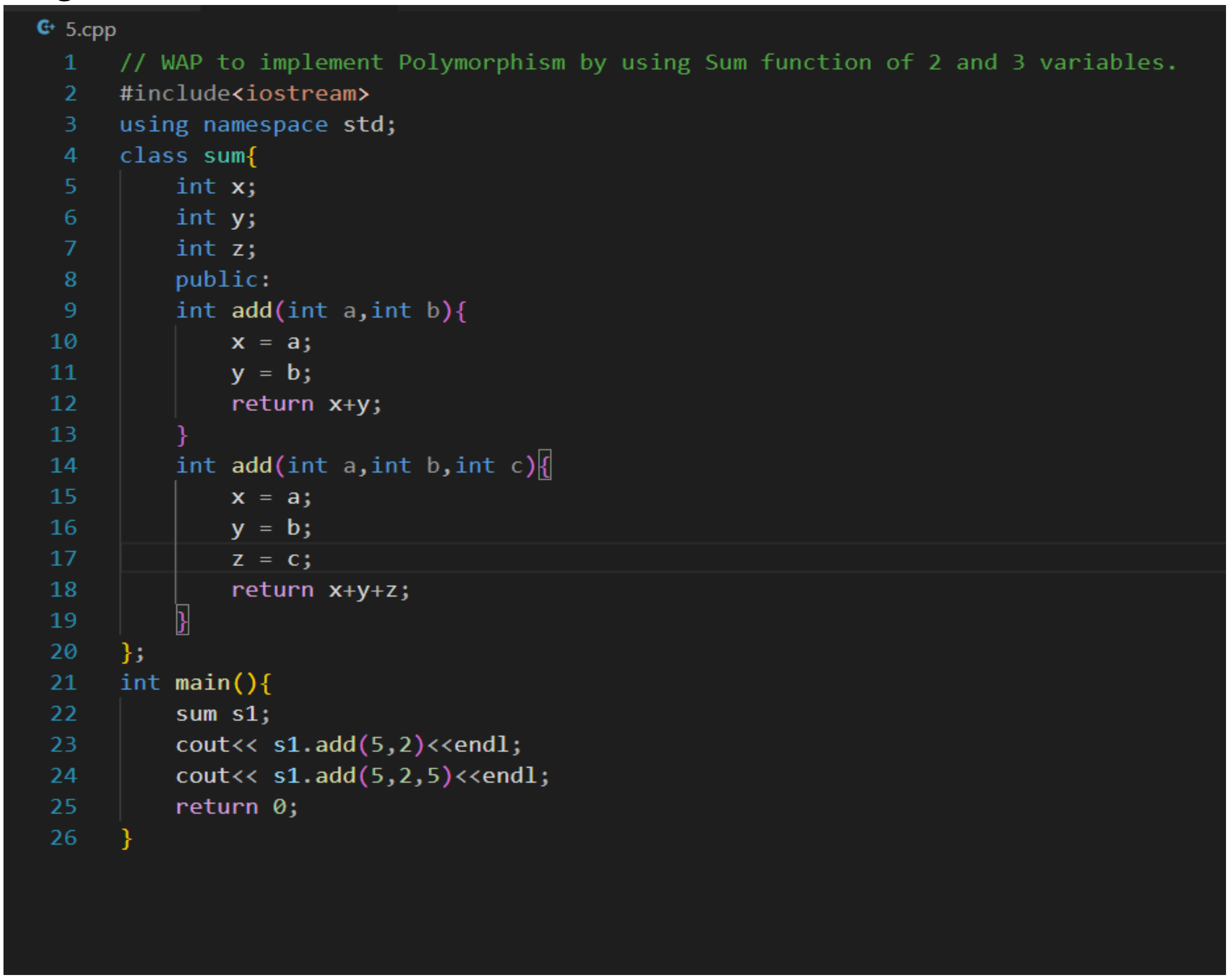
Program Objective: WAP to implement Polymorphism by using Sum function of 2 and 3 variables.

Program theory: The word "polymorphism" means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. 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.

<u>Types of Polymorphism</u>

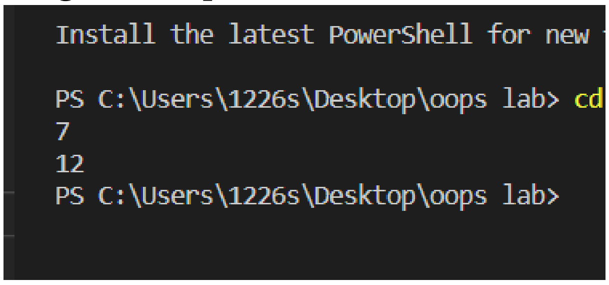
- Compile-time Polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.
- Runtime Polymorphism: This type of polymorphism is achieved by Function Overriding. Late binding and dynamic polymorphism are other names for runtime polymorphism. The function call is resolved at runtime in runtime polymorphism. In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime.

Program Code:



```
5.cpp
// WAP to implement Polymorphism by using Sum function of 2 and 3 variables.
#include<iostream>
using namespace std;
class sum{
    int x;
    int y;
    int z;
    public:
    int add(int a,int b){
        x = a;
        y = b;
        return x+y;
    }
    int add(int a,int b,int c){
        x = a;
        y = b;
        z = c;
        return x+y+z;
    }
};
int main(){
    sum s1;
    cout<< s1.add(5,2)<<endl;
    cout<< s1.add(5,2,5)<<endl;
    return 0;
}
```

Program Output:



```
Install the latest PowerShell for new

PS C:\Users\1226s\Desktop\oops lab> cd
7
12
PS C:\Users\1226s\Desktop\oops lab>
```

Learning Outcome:

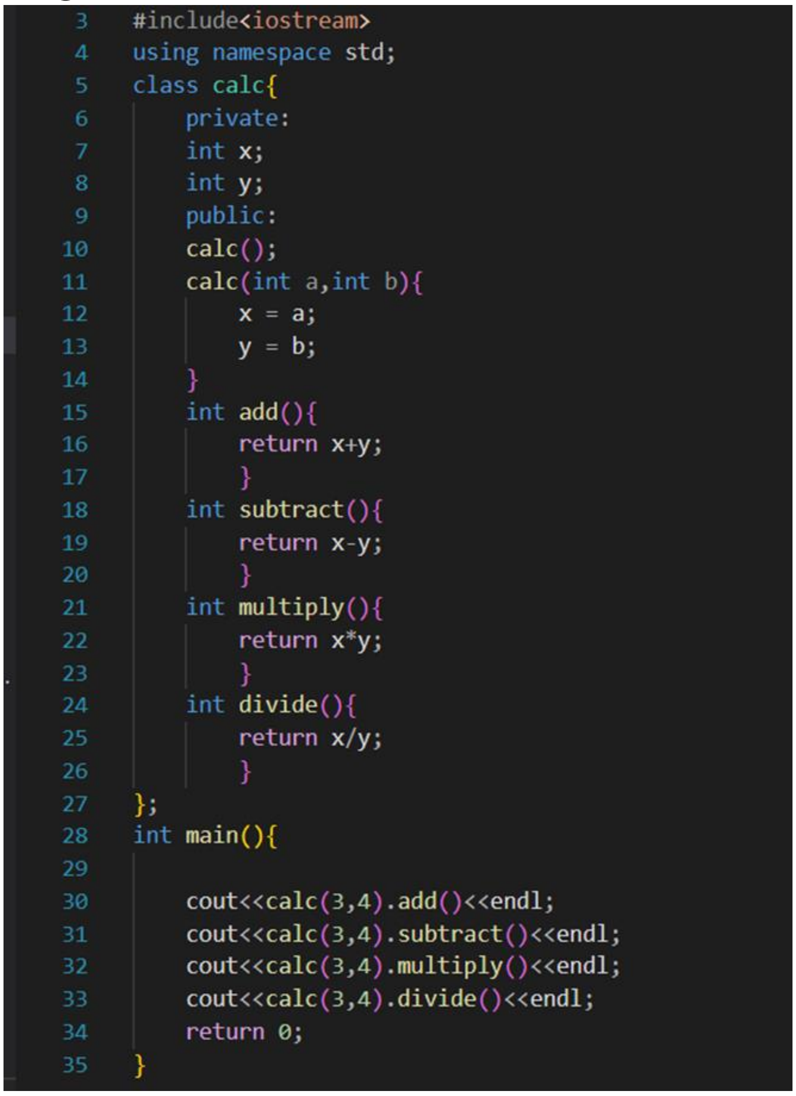
Introduction to polymorphism and its use. Implementation of function overloading with different attributes and its application.

Program 6

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

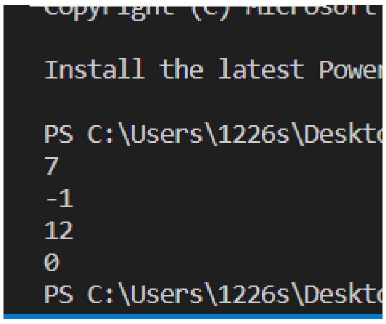
Program theory: Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. It is a process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program. It is a programming technique that depends on the separation of the interface and implementation details of the program.

Program 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(3,4).add()<<endl;
    cout<<calc(3,4).subtract()<<endl;
    cout<<calc(3,4).multiply()<<endl;
    cout<<calc(3,4).divide()<<endl;
    return 0;
}
```

Program Output:



```
Copyright (C) Microsoft

Install the latest Powe

PS C:\Users\1226s\Deskt
7
-1
12
0
PS C:\Users\1226s\Deskt
```

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

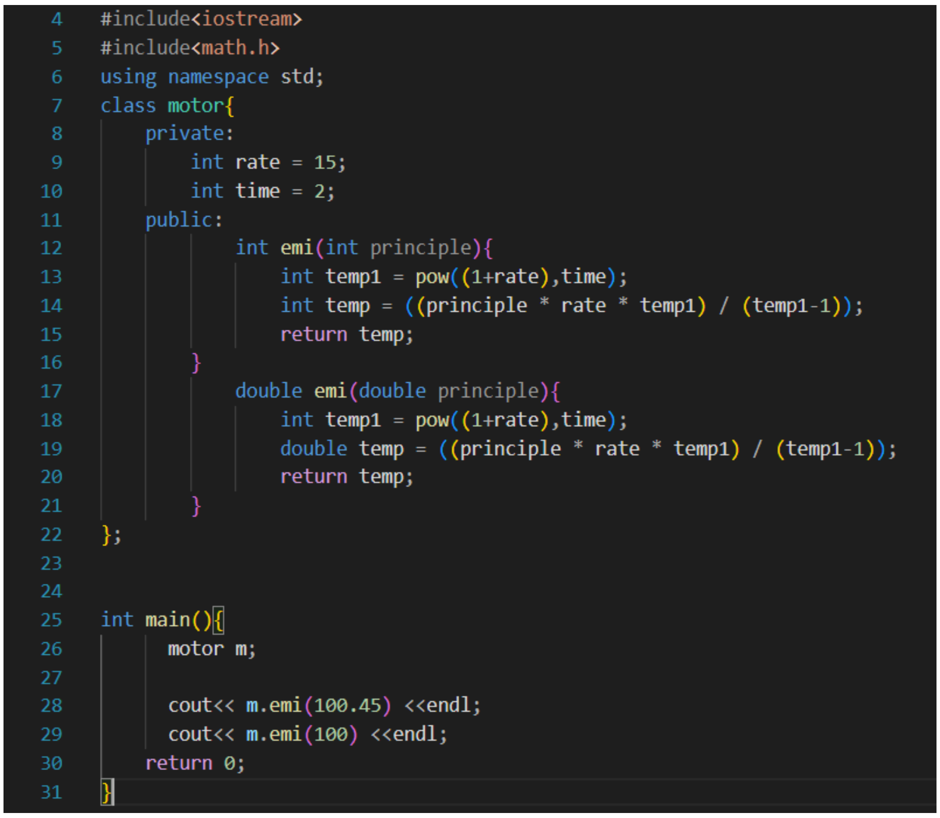
Program 7

Program Objective: 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.

Program theory: C++ lets you specify more than one function of the same name in the same scope. These functions are called overloaded functions, or overloads. Overloaded functions enable you to supply different semantics for a function, depending on the types and number of its arguments. Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++. The parameters should follow any one or more than one of the following conditions for Function overloading:

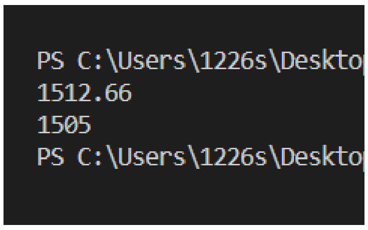
- Parameters should have a different type
- Parameters should have a different number
- Parameters should have a different sequence of parameters.

Program 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 temp1 = pow((1+rate),time);
                double temp = ((principle * rate * temp1) / (temp1-1));
                return temp;
        }
};


int main(){
      motor m;

      cout<< m.emi(100.45) <<endl;
      cout<< m.emi(100) <<endl;
    return 0;
}
```

Program Output:



```
PS C:\Users\1226s\Deskto
1512.66
1505
PS C:\Users\1226s\Deskto
```

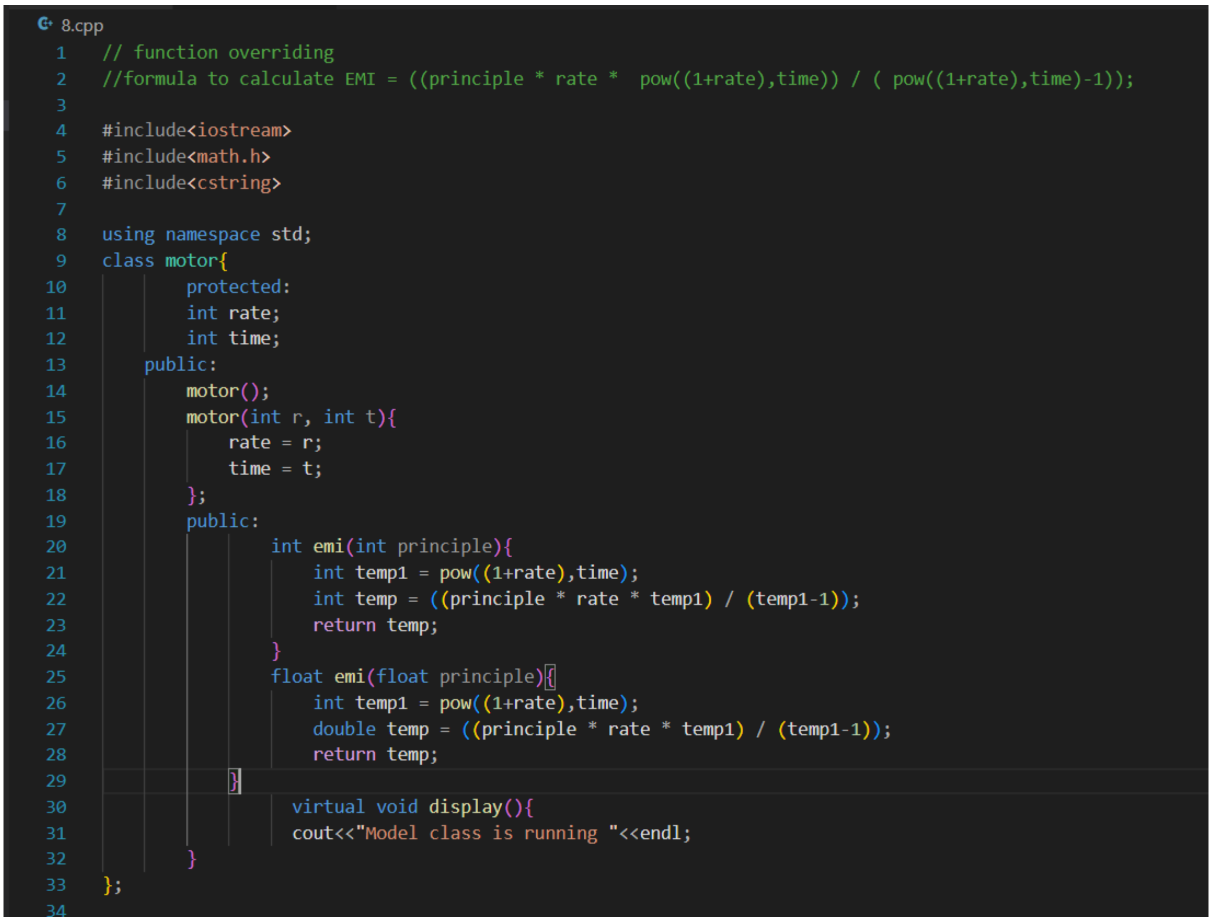
Learning Outcome: Things like data hiding and its importance along with concept of Abstraction has been learnt.

Program 8

Program Objective: 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)

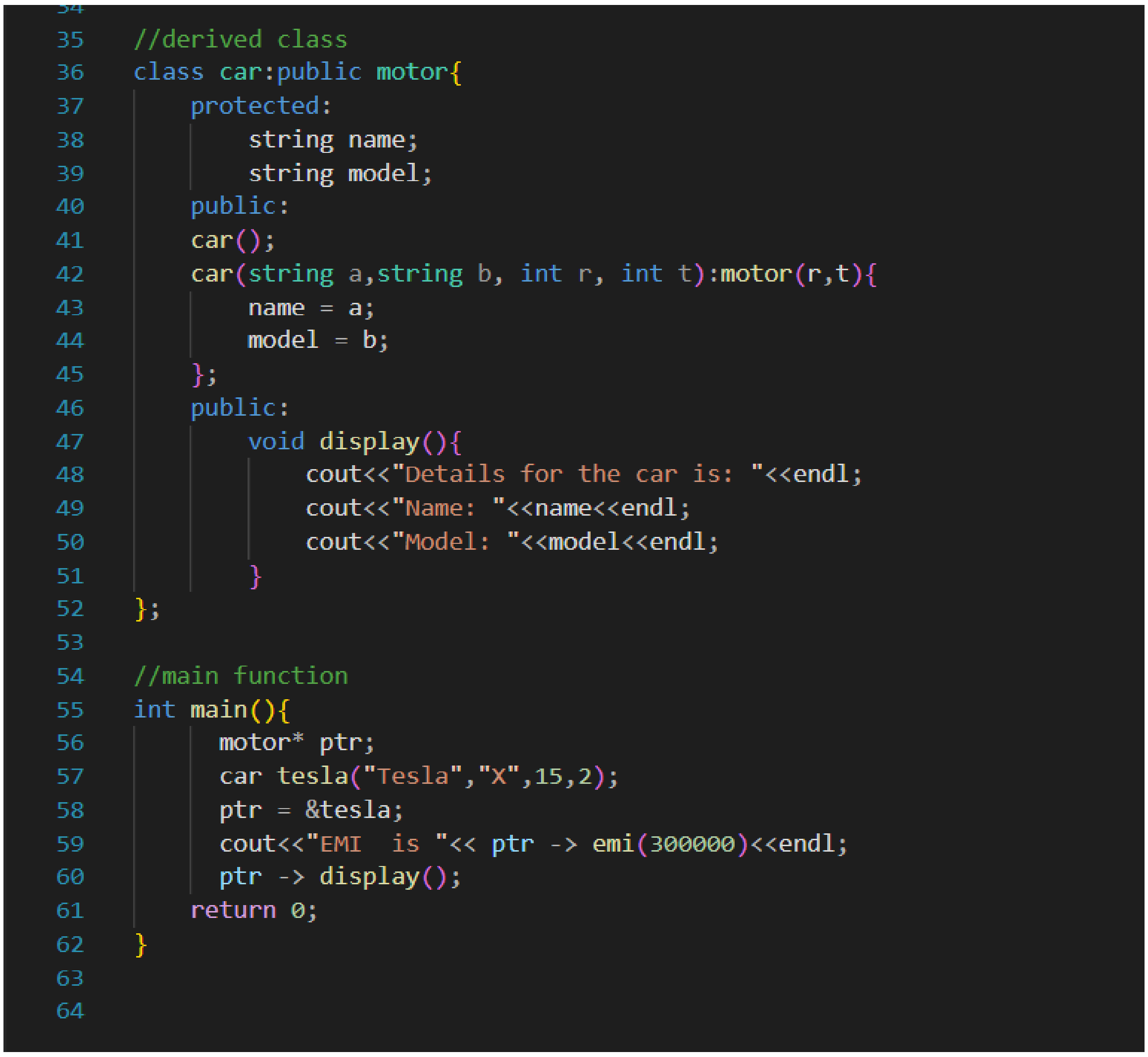
Program theory: If derived class defines same function as defined in its base class, it is known as function overriding in C++. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the function which is already provided by its base class.
Suppose, the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed.
This is known as function overriding in C++. The function in derived class overrides the function in base class.

Program Code:



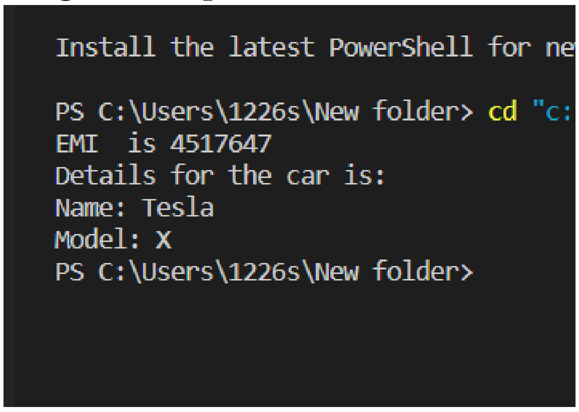
8.cpp

```cpp
// function overriding
//formula to calculate EMI = ((principle * rate *  pow((1+rate),time)) / ( pow((1+rate),time)-1));

#include<iostream>
#include<math.h>
#include<cstring>

using namespace std;
class motor{
        protected:
        int rate;
        int time;
    public:
        motor();
        motor(int r, int t){
            rate = r;
            time = t;
        };
        public:
                int emi(int principle){
                    int temp1 = pow((1+rate),time);
                    int temp = ((principle * rate * temp1) / (temp1-1));
                    return temp;
                }
                float emi(float principle){
                    int temp1 = pow((1+rate),time);
                    double temp = ((principle * rate * temp1) / (temp1-1));
                    return temp;
            }
                  virtual void display(){
                  cout<<"Model class is running "<<endl;
        }
};
```



```cpp
//derived class
class car:public motor{
    protected:
        string name;
        string model;
    public:
    car();
    car(string a,string b, int r, int t):motor(r,t){
        name = a;
        model = b;
    };
    public:
        void display(){
            cout<<"Details for the car is: "<<endl;
            cout<<"Name: "<<name<<endl;
            cout<<"Model: "<<model<<endl;
        }
};

//main function
int main(){
      motor* ptr;
      car tesla("Tesla","X",15,2);
      ptr = &tesla;
      cout<<"EMI  is "<< ptr -> emi(300000)<<endl;
      ptr -> display();
    return 0;
}
```

Program Output:



```
Install the latest PowerShell for ne

PS C:\Users\1226s\New folder> cd "c:
EMI  is 4517647
Details for the car is:
Name: Tesla
Model: X
PS C:\Users\1226s\New folder>
```

Learning Outcome:

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

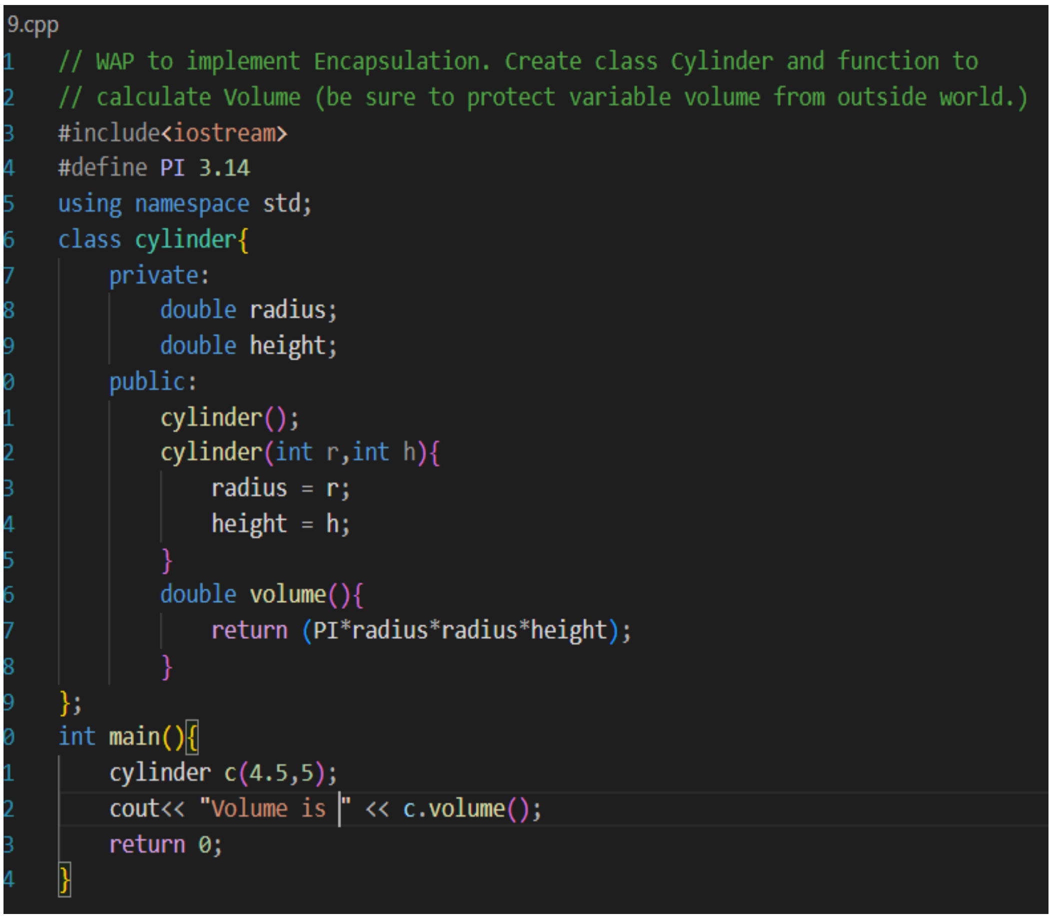
Program 9

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

Program 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.

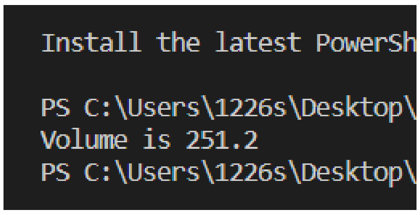
Program Code:



9.cpp

```cpp
// WAP to implement Encapsulation. Create class Cylinder and function to
// calculate Volume (be sure to protect variable volume from outside world.)
#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(4.5,5);
    cout<< "Volume is " << c.volume();
    return 0;
}
```

Program Output:



```
Install the latest PowerSh

PS C:\Users\1226s\Desktop\
Volume is 251.2
PS C:\Users\1226s\Desktop\
```

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.

Program 10

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

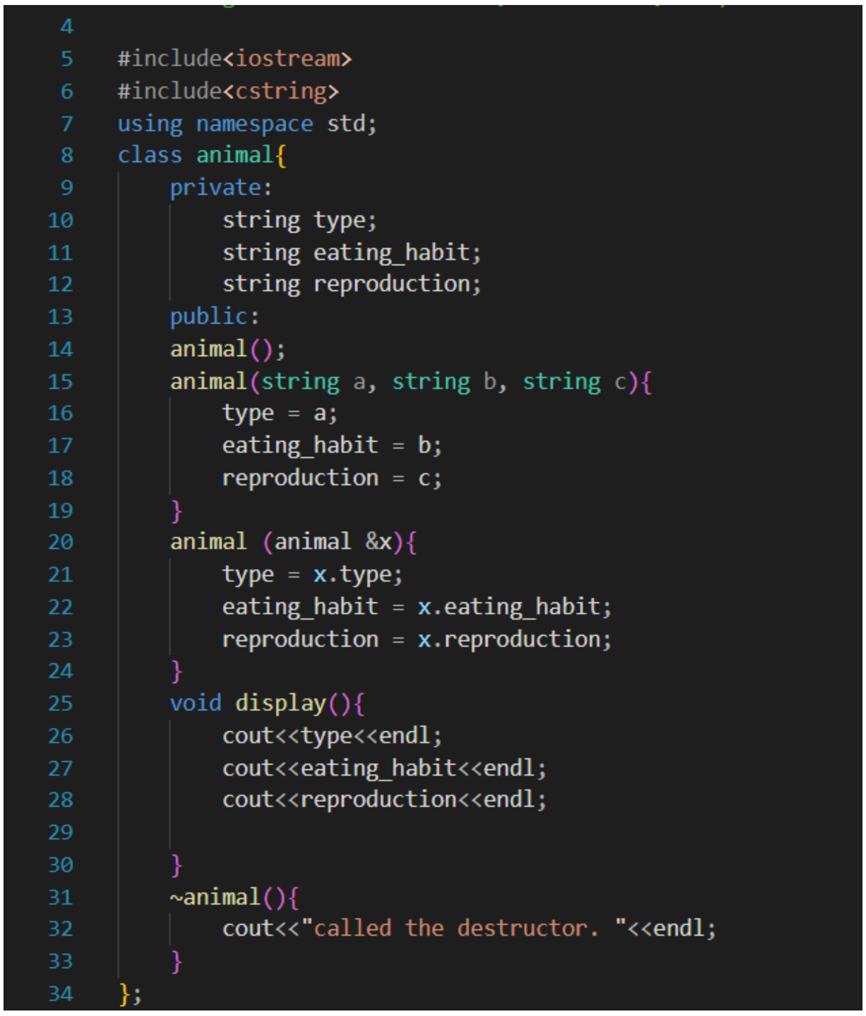
Program theory: Constructor and Destructor are the special member functions of the class which are created by the C++ compiler or can be defined by the user. Constructor is used to initialize the object of the class while destructor is called by the compiler when the object is destroyed.

Types of Constructors in C++

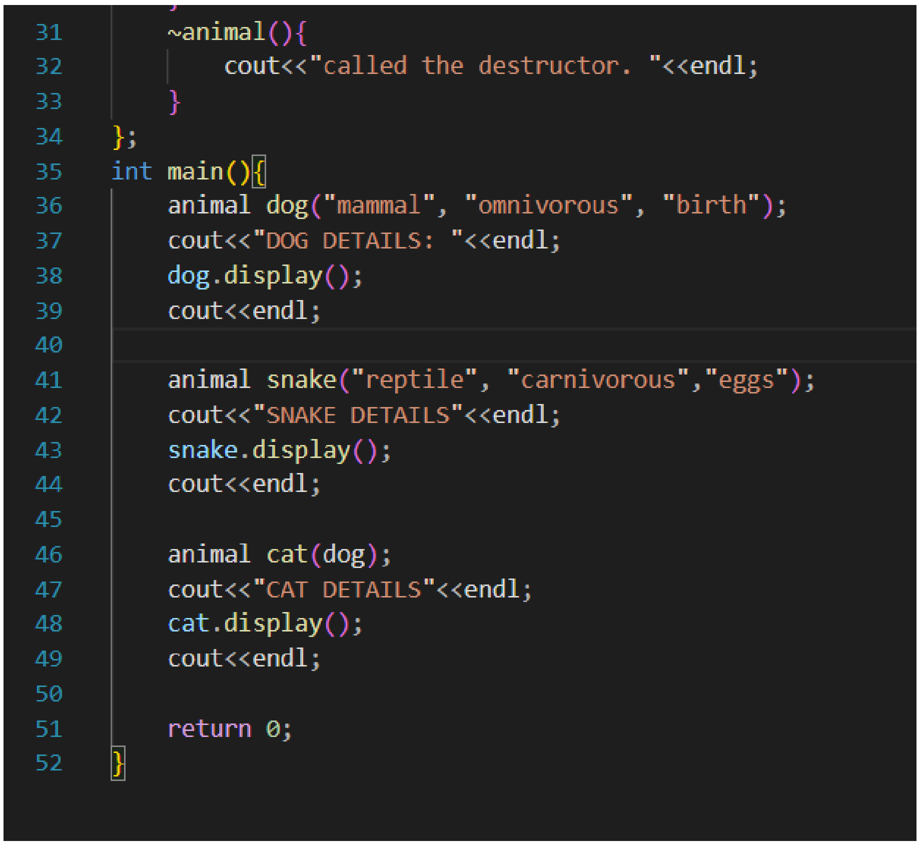
Constructors are of three types:

- Default Constructor: Default constructor is the constructor which doesn't take any argument. It has no parameter.
- Parametrized Constructor: These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.
- Copy Constructor: These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object. We will study copy constructors in detail later.

Program Code:

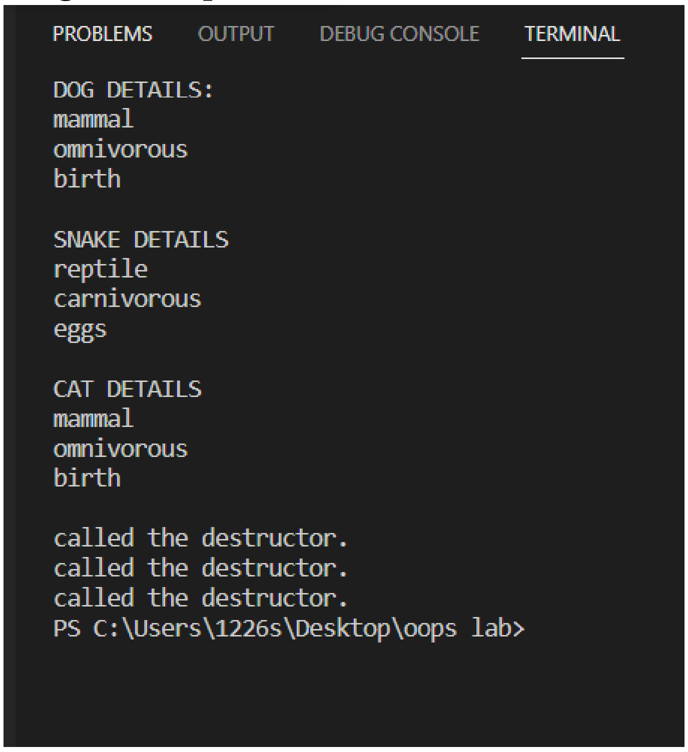


```cpp

#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
    ~animal(){
        cout<<"called the destructor. "<<endl;
    }
};
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 cat(dog);
    cout<<"CAT DETAILS"<<endl;
    cat.display();
    cout<<endl;

    return 0;
}
```

Program Output:



```
DOG DETAILS:
mammal
omnivorous
birth

SNAKE DETAILS
reptile
carnivorous
eggs

CAT DETAILS
mammal
omnivorous
birth

called the destructor.
called the destructor.
called the destructor.
PS C:\Users\1226s\Desktop\oops lab>
```

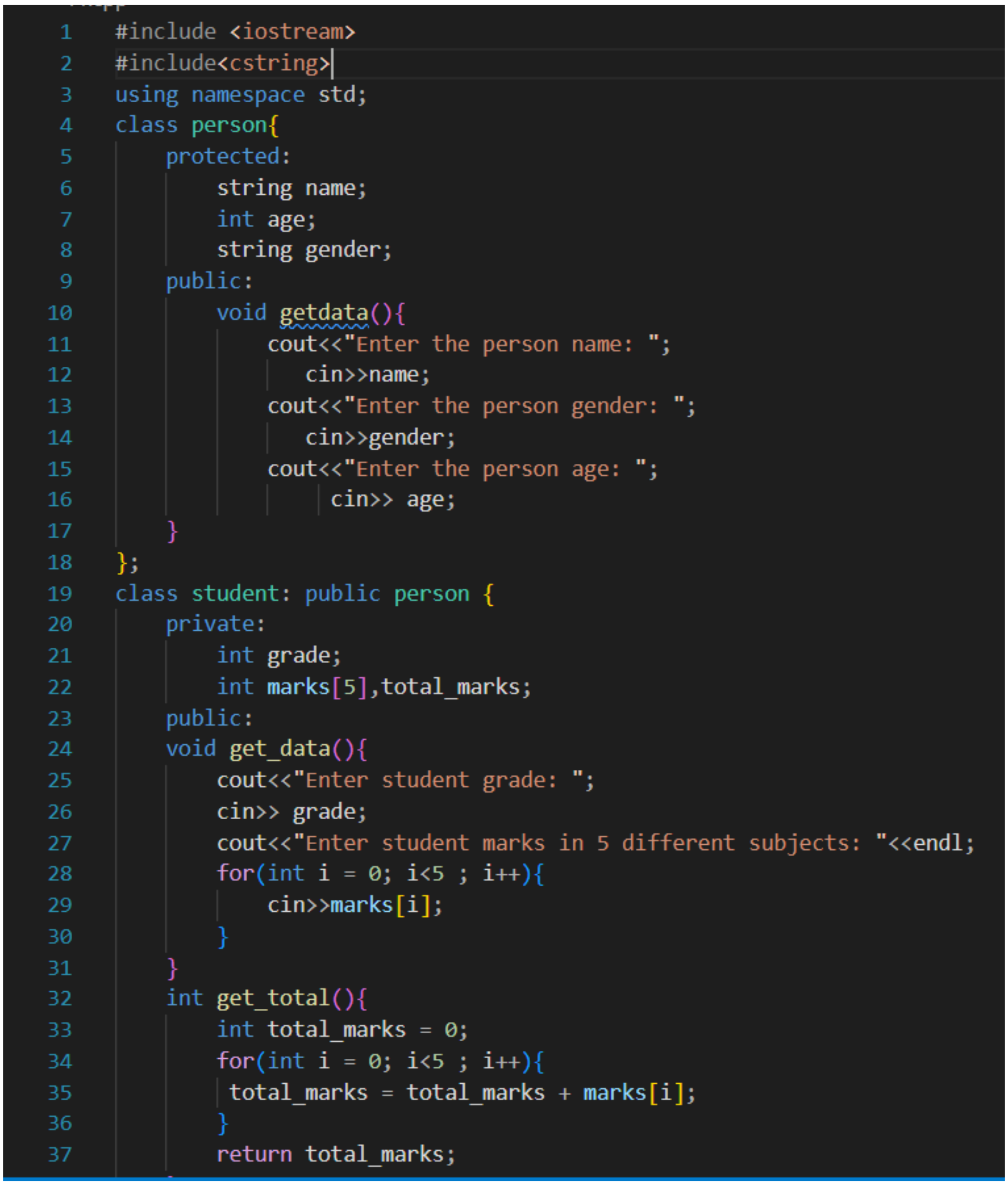
Learning Outcome: Basic constructors and destructors has been learnt in this problem. Default constructor, parameterized constructor and copy constructor has been used along with destructor.

Program 11

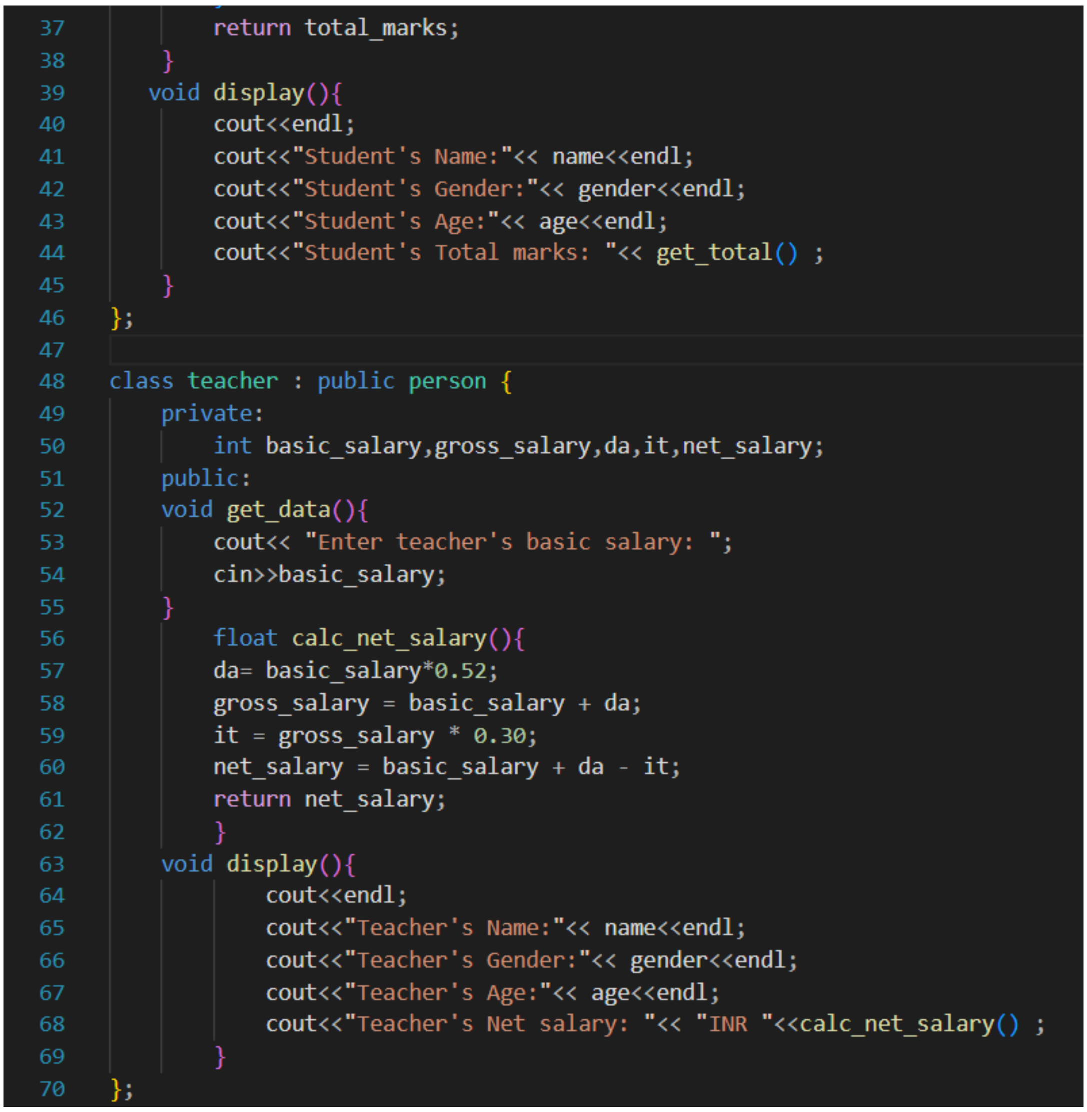
Program Objective: WAP to implement Generalization as extension. Use class person with name, age and gender as variables. Extend it by using subclass Student, Teacher and Admin Staff.

Program theory: In C/C++ domain modeling class diagrams, a generalization relationship, which is also called an inheritance or "an A is a B" (a human is a mammal, a mammal is an animal) relationship, implies that a specialized, child class is based on a general, parent class. Generalization is the process of taking out common properties and functionalities from two or more classes and combining them together into another class which acts as the parent class of those classes or what we may say the generalized class of those specialized classes. All the subclasses are a type of superclass.

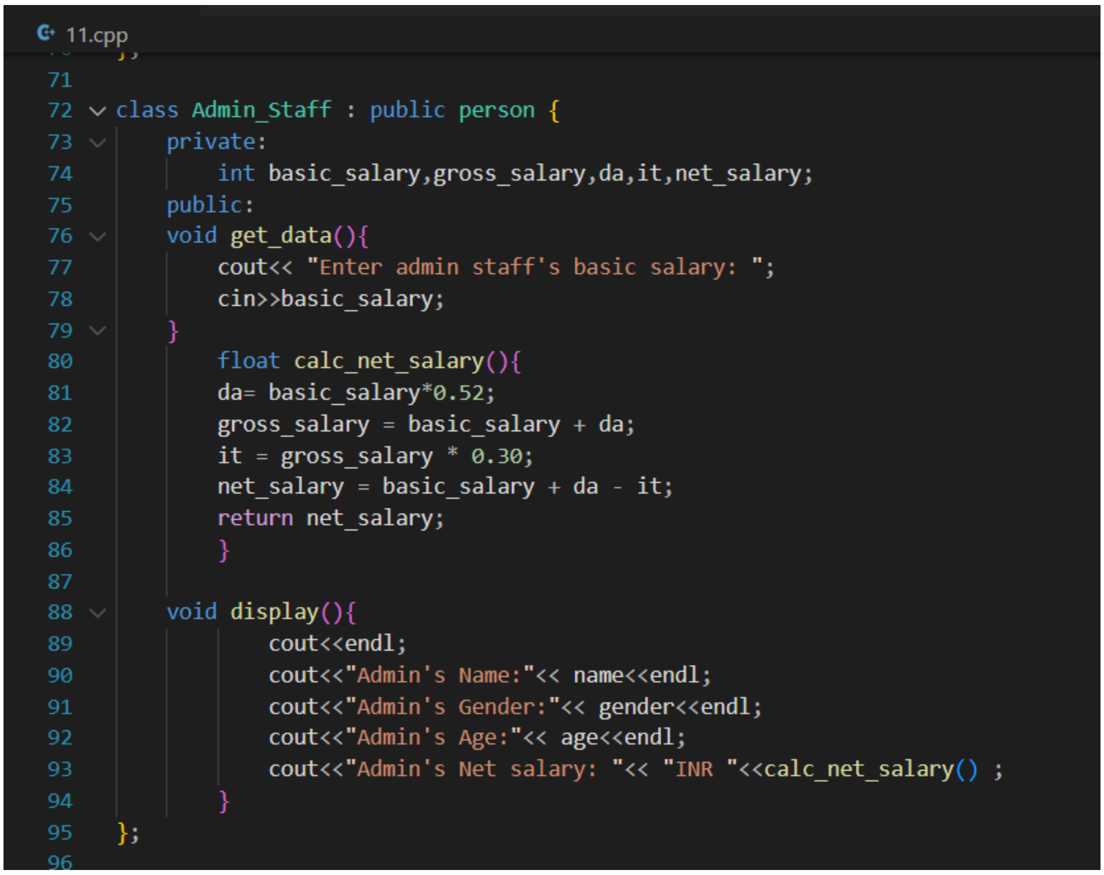
Program Code:



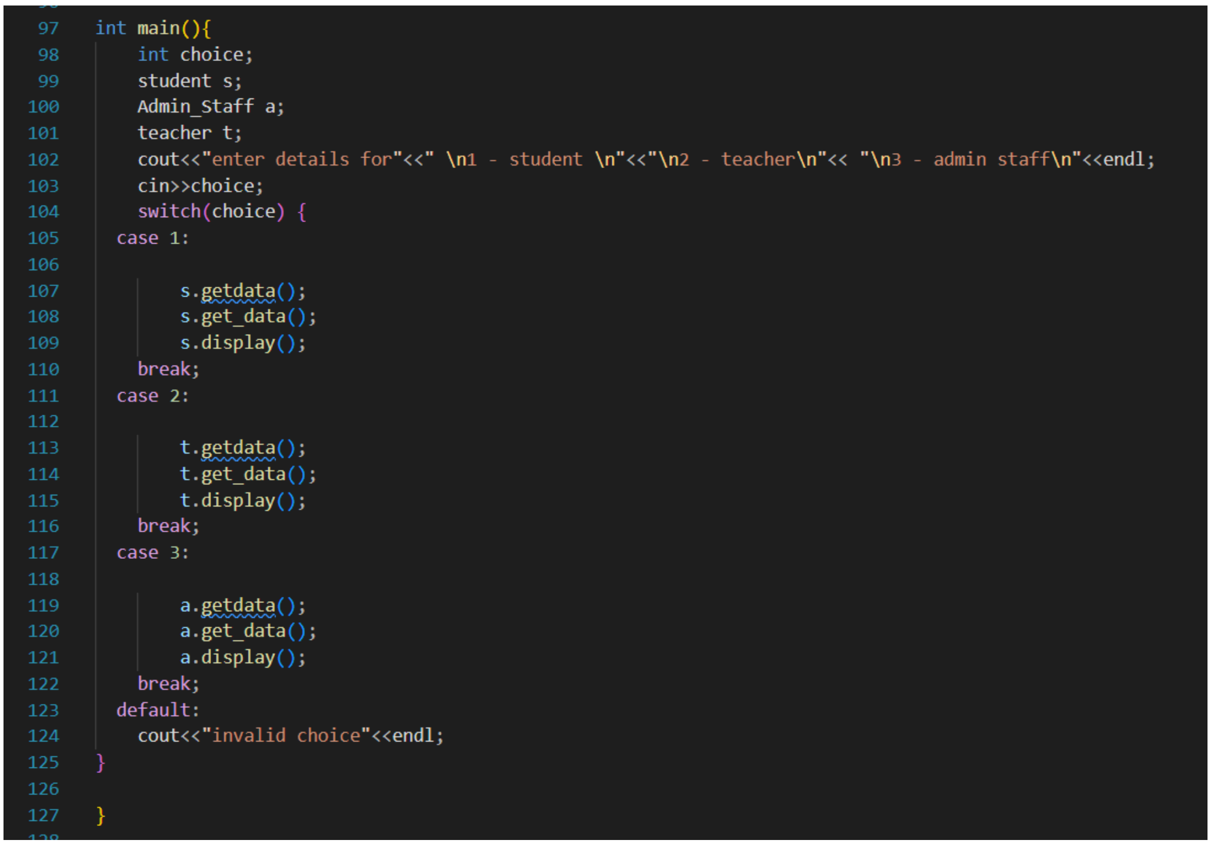
```
#include <iostream>
#include<cstring>
using namespace std;
class person{
    protected:
        string name;
        int age;
        string gender;
    public:
        void getdata(){
            cout<<"Enter the person name: ";
               cin>>name;
            cout<<"Enter the person gender: ";
               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;
```



```cpp
        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;
        }
    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() ;
        }
};
```

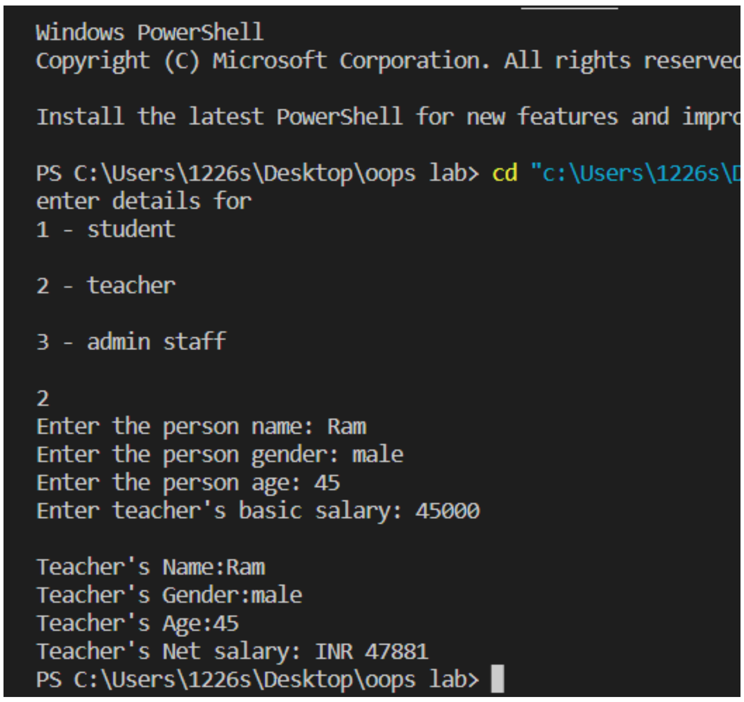


```cpp

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;
        }

    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() ;
        }
};
```



```cpp
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;
  case 3:

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

}
```

Program Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved

Install the latest PowerShell for new features and impro

PS C:\Users\1226s\Desktop\oops lab> cd "c:\Users\1226s\D
enter details for
1 - student

2 - teacher

3 - admin staff

2
Enter the person name: Ram
Enter the person gender: male
Enter the person age: 45
Enter teacher's basic salary: 45000

Teacher's Name:Ram
Teacher's Gender:male
Teacher's Age:45
Teacher's Net salary: INR 47881
PS C:\Users\1226s\Desktop\oops lab> █
```

Learning Outcome:

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

Program 12

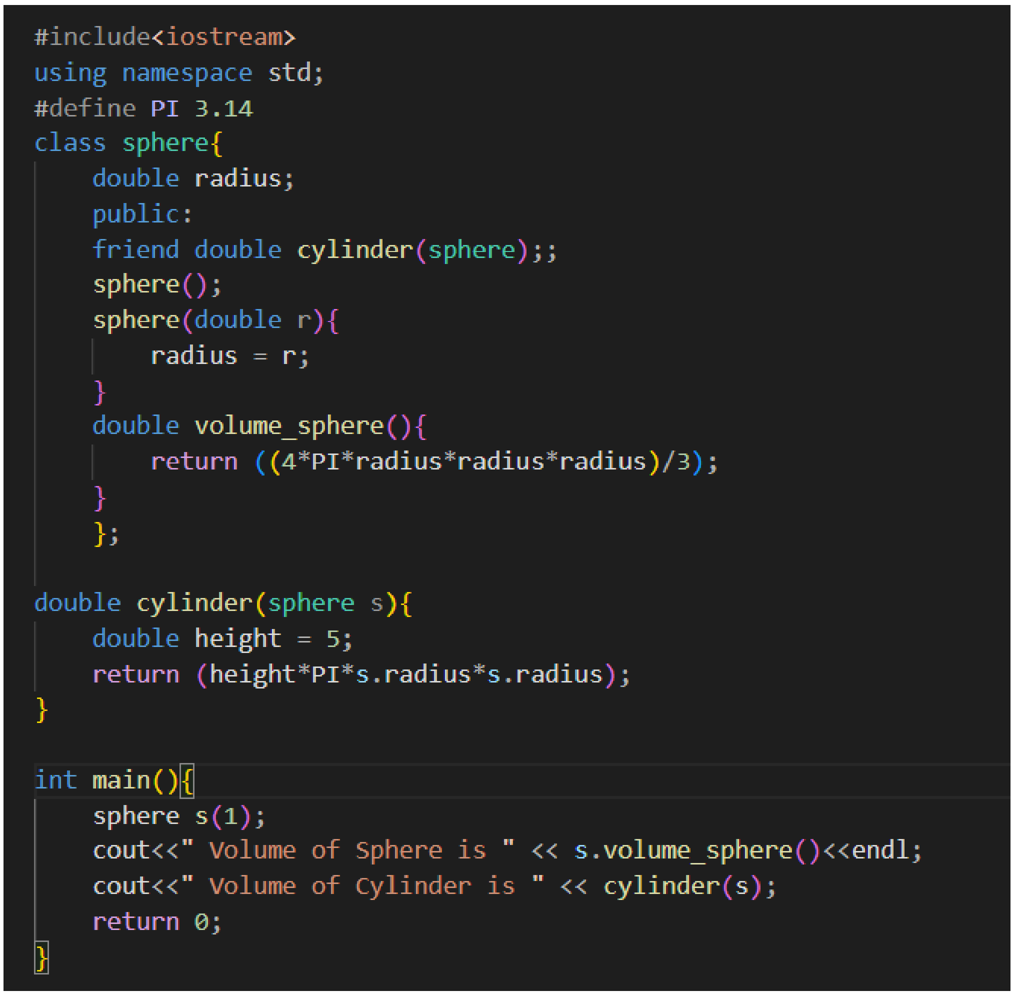
Program Objective: WAP to implement friend function in Java or C++ . Use class Sphere with radius as variable and friend function cylinder (with given height) and calculate the Volume of both.

Program theory: If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.
By using the keyword friend compiler knows the given function is a friend function.
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.
For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.
Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

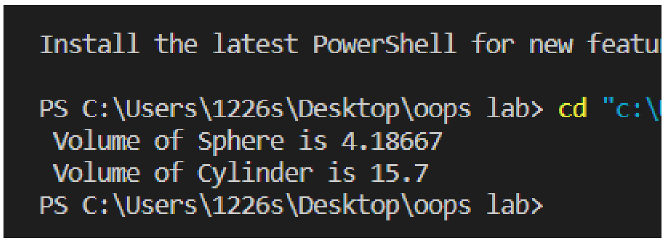
Use of Friend function in C++ :
we require friend functions whenever we have to access the private or protected members of a class. This is only the case when we do not want to use the objects of that class to access these private or protected members.

Program 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;
}
```

Program Output:



```
Install the latest PowerShell for new featu

PS C:\Users\1226s\Desktop\oops lab> cd "c:\
 Volume of Sphere is 4.18667
 Volume of Cylinder is 15.7
PS C:\Users\1226s\Desktop\oops lab>
```

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.