**Object Oriented Programming**

**Lab 10**

**Submitted To:**

Ma’am Amber Madeeha Zeb

**Submitted By:**

Manaal Waseem

FA18-BCE-074

**In Lab:**

**Task 1:**

**Consider the class**

class base

{

public:

virtual void iam()

{

cout << “base\n”;

}

};

**a. Derive two classes from class base, and for each define iam() to write out the name of the class.**

**b. Declare objects of each class, and call iam() from them.**

**c. Assign the address of objects of the derived classes to base pointers and call iam() through the pointers.**

**d. Remove the virtual keyword from the base class member function, run your code again, and compare the results.**

**Code:**

1 #include <iostream>

2 #include <string>

3

4 **using namespace std**;

5

6 **class** employee //employee class

7 {

8 **public**:

9 **virtual void** Salary()= 0;

10 **virtual void** display()= 0;

11 };

12

13 **class** Salaried\_employee : **public** employee

14 {

15 **private**:

16 **string** name; //employee name

17 **long** employee\_number; //employee number

18 **double** emp\_salary;

19 **public**:

20 **void** Salary()

21 {

22 **cout** << "Enter Name: " << **endl**; **cin** >> name;

23 **cout** << "Enter Employee Number: " << **endl**; **cin** >> employee\_number;

24 **cout** << "Enter Salary: " << **endl**; **cin** >> emp\_salary;

25 }

26 **void** display()

27 {

28 **cout** << "Name: " << name << **endl**;

29 **cout** << "Employee Number: " << employee\_number << **endl**;

30 **cout** << "Salary: " << emp\_salary << **endl**;

31 }

32 };

33

34 **class** Hourly\_employee : **public** employee

35 {

36 **private**:

37 **string** name; //employee name

38 **long** employee\_number; //employee number

39 **double** emp\_salary;

40 **public**:

41 **void** Salary()

42 {

43 **cout** << "Enter Name: " << **endl**; **cin** >> name;

44 **cout** << "Enter Employee Number: " << **endl**; **cin** >> employee\_number;

45 **cout** << "Enter Salary: " << **endl**; **cin** >> emp\_salary;

46 }

47 **void** display()

48 {

49 **cout** << "Name: " << name << **endl**;

50 **cout** << "Employee Number: " << employee\_number << **endl**;

51 **cout** << "Salary: " << emp\_salary << **endl**;

52 }

53 };

54

55 **class** Commissioned\_employee : **public** employee

56 {

57 **private**:

58 **string** name; //employee name

59 **long** employee\_number; //employee number

60 **double** emp\_salary;

61 **public**:

62 **void** Salary()

63 {

64 **cout** << "Enter Name: " << **endl**; **cin** >> name;

65 **cout** << "Enter Employee Number: " << **endl**; **cin** >> employee\_number;

66 **cout** << "Enter Salary: " << **endl**; **cin** >> emp\_salary;

67 }

68 **void** display()

69 {

70 **cout** << "Name: " << name << **endl**;

71 **cout** << "Employee Number: " << employee\_number << **endl**;

72 **cout** << "Salary: " << emp\_salary << **endl**;

73 }

74 };

75

76 **int** main()

77 {

78 employee \*E[3];

79 **char** choice;

80

81 **for**(**int** i=0; i<3; i++)

82 {

83 **cout**<< "Enter the type of Employee(m, h, c): " <<**endl**;

84 **cin**>> choice;

85 **if**(choice=='m')

86 E[i]= **new** Salaried\_employee;

87 **else**

88 **if**(choice=='h')

89 E[i]= **new** Hourly\_employee;

90 **else**

91 **if**(choice=='c')

92 E[i]= **new** Commissioned\_employee;

93

94 E[i]->Salary();

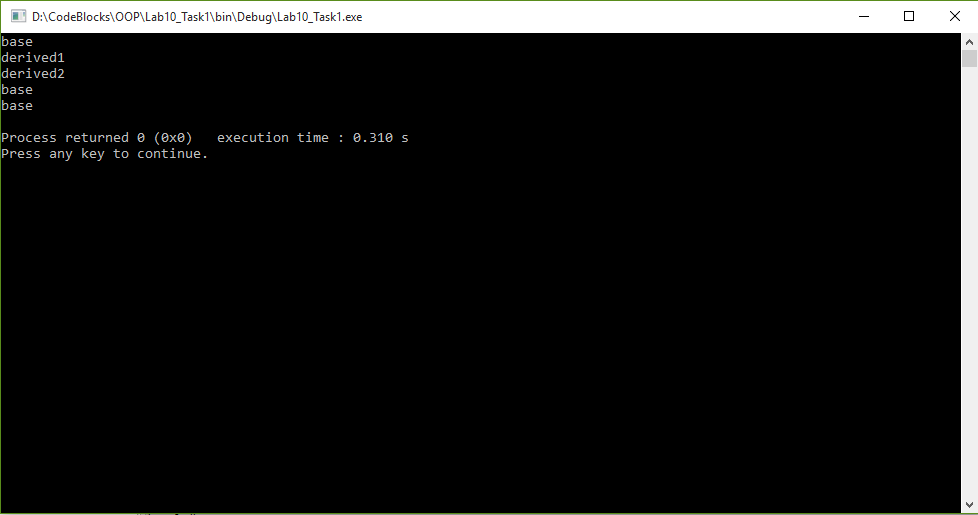
95 E[i]->display();

96 }

97 **return** 0;

98 }

**Output:**



**Task 2:**

**Develop a simple payroll application. There are three kinds of employees in the system: salaried employee, hourly employee, and commissioned employee. The system takes as input an array containing employee objects, calculates salary polymorphically, and generates report.**

**Make Employee an abstract class. Declare salary() and display() as pure virtual functions in it. Derive salaried employee (monthly), hourly employee (per hour basis), and commissioned employee (bonus on completing each target) from base class Employee. The display() function should show employee no, employee name, and salary of all employees.**

**Code:**

1 #include <iostream>

2

3 **using namespace std**;

4

5 **class** Shape

6 {

7 **protected**:

8 **double** d1;

9 **double** d2;

10 **public**:

11 **virtual void** get\_data()

12 {

13 **cout** << "Enter first value: " << **endl**; **cin** >> d1;

14 **cout** << "Enter second value: " << **endl**; **cin** >> d2;

15 }

16 **virtual void** display\_area()= 0;

17 };

18

19 **class** Triangle : **public** Shape

20 {

21 **private**:

22 **double** tri\_area;

23 **public**:

24 **void** display\_area()

25 {

26 tri\_area=(0.5)\*d1\*d2;

27 **cout**<< "Triangle Area: " << tri\_area <<**endl**;

28 }

29 };

30

31 **class** Rectangle : **public** Shape

32 {

33 **private**:

34 **double** rec\_area;

35 **public**:

36 **void** display\_area()

37 {

38 rec\_area=d1\*d2;

39 **cout**<< "Rectangle Area: " << rec\_area <<**endl**;

40 }

41 };

42

43 **int** main()

44 {

45 Shape \*s1, \*s2;

46

47 s1= **new** Rectangle;

48 s1->get\_data();

49 s1->display\_area();

50

51 s2= **new** Triangle;

52 s2->get\_data();

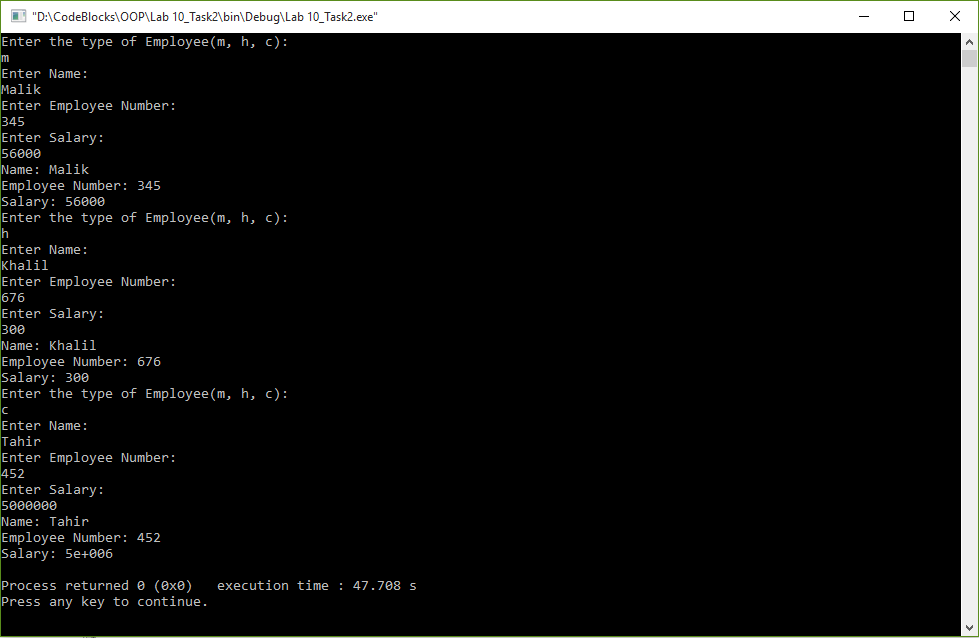
53 s2->display\_area();

54

55 return 0;

56 }

**Output:**



**Task 3:**

**Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive two specific classes called triangle and rectangle from the base shape. Add to base class, a member function get\_data() to initialize base class data members and another member functions display\_area() to compute and display the area of figures. Mark the display\_area() as a virtual function and redefine this function in the derived class to suit their requirements.(Use pure virtual function)**

**Code:**

1 #include <iostream>

2

3 **using namespace std**;

4

5 **class** Shape

6 {

7 **protected**:

8 **double** d1;

9 **double** d2;

10 **public**:

11 **virtual void** get\_data()

12 {

13 **cout** << "Enter first value: " << **endl**; **cin** >> d1;

14 **cout** << "Enter second value: " << **endl**; **cin** >> d2;

15 }

16 **virtual void** display\_area()= 0;

17 };

18

19 **class** Triangle : **public** Shape

20 {

21 **private**:

22 **double** tri\_area;

23 **public**:

24 **void** display\_area()

25 {

26 tri\_area=(0.5)\*d1\*d2;

27 **cout**<< "Triangle Area: " << tri\_area <<**endl**;

28 }

29 };

30

31 **class** Rectangle : **public** Shape

32 {

33 **private**:

34 **double** rec\_area;

35 **public**:

36 **void** display\_area()

37 {

38 rec\_area=d1\*d2;

39 **cout**<< "Rectangle Area: " << rec\_area <<**endl**;

40 }

41 };

43 **int** main()

44 {

45 Shape \*s1, \*s2;

46

47 s1= **new** Rectangle;

48 s1->get\_data();

49 s1->display\_area();

50

51 s2= **new** Triangle;

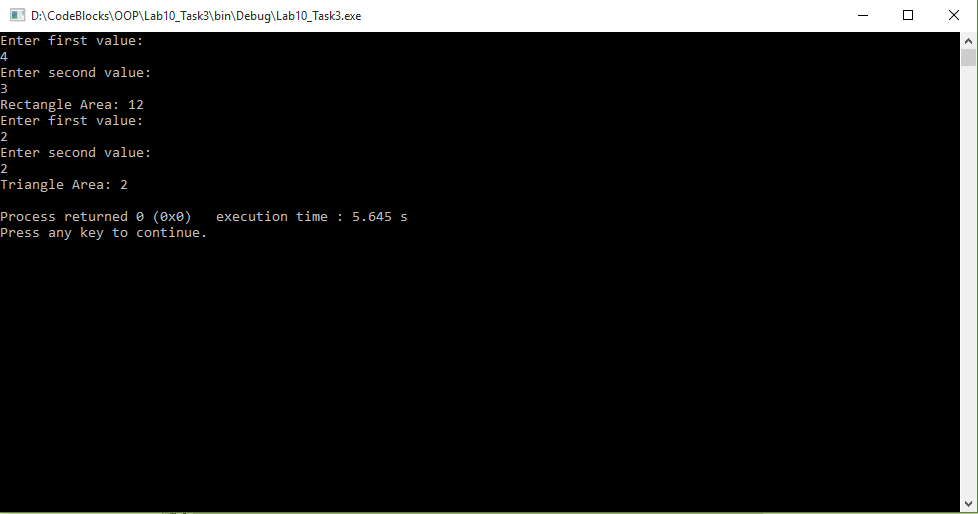
52 s2->get\_data();

53 s2->display\_area();

54 return 0;

55 }

**Output:**



**Post Lab:**

**Task 1:**

**Create a class hierarchy that performs conversions from one system of units to another. Your program should perform the following conversions,**

**i. Liters to Gallons,**

**ii. Fahrenheit to Celsius and**

**iii. Feet to Meters**

**The base class convert declares two variables, val1 and val2, which hold the initial and converted values, respectively. It also defines the functions getinit() and getconv(), which return the initial value and the converted value. These elements of convert are fixed and applicable to all derived classes that will inherit convert. However, the function that will actually perform the conversion, compute(), is a pure virtual function that must be defined by the classes derived from convert. The specific nature of compute() will be determined by what type of conversion is taking place.**

**Three classes will be derived from convert to perform conversions of Liters to Gallons (l\_to\_g), Fahrenheit to Celsius (f\_to\_c) and Feet to Meters (f\_to\_m), respectively. Each derived class overrides compute() in its own way to perform the desired conversion.**

**Test these classes from main() to demonstrate that even though the actual conversion differs between l\_to\_g, f\_to\_c, and f\_to\_m, the interface remains constant.**

**Code:**

1 #include <iostream>

2

3 **using namespace std**;

4

5 **class** convert

6 {

7 **protected**:

8 **double** val1;

9 **double** val2;

10 **public**:

11 **void** get\_val()

12 {

13 **cout** << "Enter Initial Value: " << **endl**; **cin** >> val1;

14 }

15 **double** getinit()

16 {

17 **return** val1;

18 }

19 **double** getconv()

20 {

21 **return** val2;

22 }

23 **virtual void** compute()= 0;

24 };

25

26 **class** l\_to\_g : **public** convert

27 {

28 **public**:

29 **void** compute()

30 {

31 val2=val1/3.875;

32 }

33 };

34

35 **class** f\_to\_c : **public** convert

36 {

37 **public**:

38 **void** compute()

39 {

40 val2=((val1-32)\*5)/9;

41 }

42 };

43

44 **class** f\_to\_m : **public** convert

45 {

46 **public**:

47 **void** compute()

48 {

49 val2=val1/3.281;

50 }

51 };

52

53 **int** main()

54 {

55 convert \*c1, \*c2, \*c3;

56

57 c1= **new** l\_to\_g;

58 c1->get\_val();

59 c1->compute();

60 **cout**<< "Convertion: " << c1->getconv() <<**endl**;

61

62 c2= **new** f\_to\_c;

63 c2->get\_val();

64 c2->compute();

65 **cout**<< "Convertion: " << c2->getconv() <<**endl**;

66

67 c3= **new** f\_to\_m;

68 c3->get\_val();

69 c3->compute();

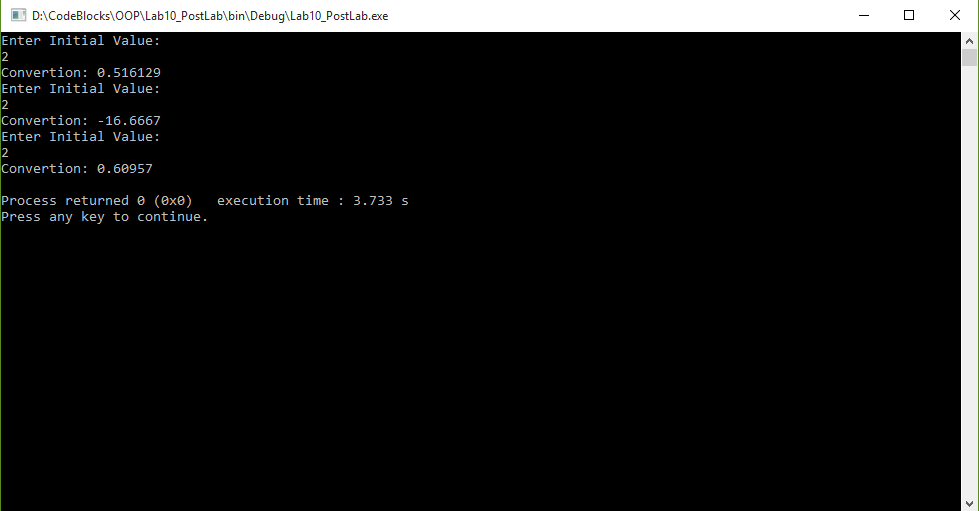
70 **cout**<< "Convertion: " << c3->getconv() <<**endl**;

71

72 **return** 0;

73 }

**Output:**



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**THE END**