In-Class Assignment-Week 9

1.Write a program to demonstrate function overloading with detailed explanation on every logic you use in the program.

#include <iostream>  
#include <string>  
using namespace std;  
//C++ calls the function with a matching data type.  
//original "add" function('first function').  
void add(float x, float y)  
{  
 cout << "numerical sum = " << (x + y) << endl;  
}  
  
void add(string x, string y)  
{  
 cout << "string sum = " << x << y << endl;  
}  
  
// Driver code  
int main(){  
 //when a float value is passed, the first function is called.  
 add(10, 101);  
 //when a string value is passed, the second function is called.  
 add("10", "101");  
 return 0;  
}

sum = 111

string sum = 10101

Process finished with exit code 0

2.Write a program to demonstrate operator overloading with detailed explanation on every logic you use in the program.

//Operator overloading is a feature of C++ allows us to use the same operator for different purposes.  
#include <iostream>  
#include <string>  
#include <vector>  
using namespace std;  
  
class Gradebook {  
private:  
 vector<string> student\_grades;  
 string grade, student\_id;  
public:  
 Gradebook() : grade(){};//Default constructor to initialize grade(set to 0).  
 //Overload the '-' operator.  
 //When prefix '+' is used, this function is called(first function).  
 //This function essential redifines the '+' operator.  
 void operator + () {  
 grade = "Pass";  
 }  
  
 //Overload the '-' operator.  
 //When prefix '-' is used, this function is called(second function).  
 void operator - () {  
 grade = "Fail";  
 }  
  
 //When suffix '\*' is used, this function is called.  
 void operator \* (string student\_id) {  
 //cin >> student\_id;  
 string student = student\_id + " -> " + grade;  
 student\_grades.push\_back(student);  
 }  
  
 //For outputting specifically assigned grades.  
 //Outputs the specific grade of the given suffix when called.  
 void display() {  
 cout << "Grade: " << grade << endl;  
 }  
  
 //for outputting vector.  
 void display\_student\_grades() {  
 std::cout << "\t\t[GradeBook]" << endl;  
 for(int i=0; i < student\_grades.size(); i++)  
 //easy way to assign an 'ID' to each element in the vector.  
 std::cout << "ID: " << i+10001 << " Name: " << student\_grades.at(i) << '\n';  
 }  
};  
  
int main() {  
 Gradebook set\_stud\_grade; //Create an object of class 'Gradebook'.  
 +set\_stud\_grade;//Calls the first function to set 'grade' to "Pass".  
 set\_stud\_grade.display();//Outputs the grade.  
 -set\_stud\_grade;//Calls the second function to set 'grade' to "Fail".  
 set\_stud\_grade.display();  
   
 //Calls the third function to set pushback the student\_id and grade to the vector.  
 //This stores the student\_id and grade in the vector, allowing for additional unique entries to vector.  
 set\_stud\_grade \* "Alex";  
   
 //Same as above.  
 +set\_stud\_grade;  
 set\_stud\_grade \* "student 2";  
 +set\_stud\_grade;  
 set\_stud\_grade\* "student 3";  
   
 set\_stud\_grade.display\_student\_grades();//calls to display the vector.  
 return 0;  
}

Grade: Pass

Grade: Fail

[GradeBook]

ID: 10001 Name: Alex -> Fail

ID: 10002 Name: student 2 -> Pass

ID: 10003 Name: student 3 -> Pass

Process finished with exit code 0

3. Write a program to demonstrate function overriding with detailed explanation on every logic you use in the program.

#include <iostream>  
using namespace std;  
  
//Base class.  
class Shape{  
public:  
 //data members.  
 double length{3}, width{5}, area{0};  
};  
Shape s{};  
//Derived class from base class 'Shape'.  
class Square : public Shape{  
public:  
 double area(){  
 s.area = (length \* width);  
 return s.area;  
 }  
};  
//By using overriding, the implementation of the function in the derived class can be altered while retaining  
// the same name and signature as the function in the base class.  
// This makes the function polymorphic(modular) and saves time and space.  
//Overriding the 'area' function.  
class Triangle : public Shape{  
public:  
 area(){  
 s.area = (length \* width) / 2;  
 return s.area;  
 }  
};  
  
int main(){  
 Square square{};//To access the 'Square' class.  
 Triangle triangle{};//To access the 'Triangle' class.  
 square.area();//Calls the 'area' function in the derived class 'Square'.  
 cout << "Square area: " << s.area << endl;  
 triangle.area();//Calls the 'area' function in the derived class 'Triangle'.  
 cout << "Triangle area: " << s.area << endl;  
 return 0;  
}

Square area: 15

Triangle area: 7.5

Process finished with exit code 0

4. Write a program to demonstrate abstract class and pure virtual function with detailed explanation on every logic you use in the program.

‘

#include <iostream>  
using namespace std;  
#include <string>  
//Abstract class(base/parent class)  
//A class can be declared abstract as long as it has at least one pure virtual function.  
class Shape {  
protected:  
 // data members.  
 double area;  
 string shape\_name;  
  
public:  
 /\*A pure virtual function is implemented by classes which are derived from  
 \* abstract class. Following is a simple example to demonstrate the same.\*/  
 virtual void square\_area() = 0; // pure virtual function, the '= 0' is what ultimately makes it a pure virtual function.  
 // Constructor for base class 'Shape'. It is called when an object of the base  
 // class is created.  
 Shape(string name) { shape\_name = name; } // Class contructor.  
};  
//This class inherits from Shape and implements square\_area()  
class Calculations : public Shape {  
public:  
 /\*Constructor for derived class 'Calculations'.  
 \* It is called when an object of the derived class is created.  
 \* When an object of the derived class is created,  
 \* the constructor of the base class is also called.\*/  
 Calculations(string name, double l, double w) : Shape(name) { area = l \* w; }  
 void square\_area() {  
 cout << "Area of " << shape\_name << " is " << area << endl;  
 }  
};  
int main() {  
 //pointer of base class, 'Shape', creating a new object, 'Square' of the derived class,  
 // 'Calculations'.  
 Shape \*square = new Calculations("Square", 10.65, 10.34);  
 square->square\_area();  
 return 0;  
}

Area of Square is 110.121

Process finished with exit code 0