C++ Problem Set

I/0 Stream

-Use the C++ I/O streams to write a program which prompts the user to enter two integer values and output a message stating the value of each integer entered followed by their average. Calculate the average in a separate function called from main().

-Write a program that will print out the characters 'a'...'z' and their integer values in hexidecimal format.

Dynamic-memory allocation

-Given the following Student structure:

```
struct Student {
    char* name;
    int id;
    float gpa;
}
```

write a program that will dynamically create an array of Students structures. Your code should prompt the user the number of students in the class, create the appropriately sized array, initialize the id members, and then free up the allocated memory.

Function Overloading

-In the program you wrote for the first problem, overload your averaging function to take in two floats and return a float. In addition, modify the code to prompt the user for two additional floating-point values and output their average.

-Please write two functions with the same overloaded function name length() which takes an integer or a character string as argument. Given an integer argument, function length should return the number of digits of the integer. Given a string parameter, the function should return the number of non-null characters in the string. Your main program should test your overloaded function with the integers 123 and -11 (which length() should return the value 3) and the string "Spring is close".

Inlined Functions

-Find a situation where the following macro for squaring a number won't work properly

```
#define SQ(x)  x*x
```

Rewrite the macro as an inline function and show that new function produces the correct result.

Call-by-Reference

-Write a function sort which will have six arguments. The arguments will be of type reference-to-int. The task of the sort function is to sort the arguments in ascending order. Test out your function in the main program.

Operator Overloading

-Overload the division operator / to work with complex numbers. Use the structure called complex defined in the course handout. Test out your code by dividing 4 +7i by 5 +9i.

-Implement an association that maps character strings to numbers (in any way you want). To do this overload the subscript operator []. Statements such as the following should work in your main code:

```
t["harry"]=12;
```

Class Basics

-Write a program that declares a class Evaluator which can be used to evaluate expressions. The class should have two data members to store the operands. It should have member functions which can give values to the operands. It should also have four member functions to perform addition, multiplication, subtraction, and division of the operands. For example, here is a way this class could be used

```
Evaluator e;
```

```
e.setright(17.0);
e.setleft(25.3);
cout << e.add() << endl;
```

Constructors and Destructors

-Rewrite the Evaluator class used in the last problem so that is has a constructor which can be used to initialize the operands.

-Write a constructor for the following class which will initialize the data members. Your constructor should also have default values for the arguments. Test out the constructor in a main program.

```
class Employee {
        int age;
        int jobcode;
  public:
        char* name;
};
```

-Write a constructor for the following class which will dynamically create the data array to have a size wd*ht. Also write a destructor for this which frees the memory. Test your work in a main program.

```
class Image {
    int wd,ht;
    char *data;
};
```

-Design and use a stack of integers class. Your class should include a constructor and destructor which dynamically create and destroy the memory for the stack. It should implement basic stack capabilities: push a value to the top of the stack, pop a value from the top of the stack; calculate the number of items currently in the stack, print out all the stack contents, and inquiry functions to see if the stack is full or empty.

Operators with Objects

-Create and use a class called Name_and_Address. Define **<<** and **>>** for objects of this class.

-Create and use a histogram class by overloading the [] operator to index with floating arguments.

-Write and use a Cycle class which implements a "cyclic" variable that can be incremented by one, and cycles through the values 0, 1, 2,..., maxCount-1, 0, 1, 2, etc. This class should overload both the preincrement and postincrement version of the operator ++. Also design a user-defined conversion to convert integers to a Cycle class object, and from a Cycle class object to an integer.

Friend Functions

-Declare and use a class called Date which has three integer data members which hold the month, day, and year of a date. The class should have constructors and a display member function to print out the date in the "slash" format: 11/23/1998. The Date class should overload the ++ operator to increment the day by one. In addition, the Date class should have a friend function called equal which will determine if two Date objects are identical.

Static Data Members

-Modify the class Point defined in the class handouts so that it includes a static data member number which keeps track of the total number of Point objects in existence during the course of the program. Test your new Point class in a main program that both creates and destroys point objects in a number of ways.

Basic Inheritance

-First create a class called Circle with a data member for the radius, a constructor function, and a function called area which will return the area of the circle. Then use the Circle class to make a derived class called Cylinder. At the least, Cylinder should have a constructor function and a function called volume which returns the volume of the cylinder. Test out your class hierarchy in a main program.

-Given the following simple Calculator class:

```
class Calculator {
   public:
      int add(int a, int b) { return(a+b); }
      int sub(int a, int b) { return(a-b); }
};
```

derive a new class that adds memory to the class so that the result of a calculation can be stored. Your new class should have member functions which can clear the memory, and store and recall values from the memory. In addition there should be functions to add to the memory some value and subtract from the memory some value. In your main program test your class hierarchy.

-Write a constructor and a copy constructor for the class Derived that properly initializes its variables and its base class.Test out your work in a main program.

```
class Base {
   int w;
  public:
   Base(int wv) {w=wv;}
};
class Derived : public Base {
   int &x;
   int y;
  public:
      // constructor
      // copy constructor
};
```

-Consider the code segment given below, write a code definition for the function foo external to the class body of Y so that all three variables name c will be assigned the value of e. Test out your foo function in a main program.

```
char c;

class X {
    public:    char c;
 };
 class Y : public X  {
    public:    char c;
    void foo(char e) ;
};
```

-Write and use a class called Time which inherits the Cycle class implemented in a previous problem. This class will keep track of seconds, minutes, and hours (24 hours period). Overload the output operator << to display the time in the colon format, i.e., 12:30:00. Overload the increment operators so that they work correctly in the derived class Time.

Virtual Functions

-Given the following class

```
class Base {
   public:
       virtual void iam() { cout << "Base\n"; }
};
```

derive two classes from Base and for each define iam() to write out the name of the class. Create objects of theses classes and call iam() for them. Assign addresses of objects of the derived classes to Base* pointers and call iam() using the pointers.

-What does the following program output? How would you fix it so that the Label's display function ends up being executed?

```
#include <iostream.h>
class Form {
  public:
    virtual void display(void) { cout << "First Name: \n"; }
};
class Label : public Form {
  public:
    void display(void) { cout << "Well, what is your name?\n"; }
};
void showit(Form f) { f.display(); }
main() {
  Label l;
  showit(l);
}
```