CS 31 Worksheet Week 8 Solutions

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

Solutions are written in red. The solutions for **programming** problems are not absolute, it is okay if your code looks different; this is just one way to solve the specific problem.

If you have any questions or concerns please contact your LA or go to any of the LA office hours.

Concepts: Classes and Structs

1. Conceptual Questions (Vishal)
    - What's the main difference between declaring a type with the keyword **`struct`** and declaring it with the keyword **`class`**?

      When using `struct`, until you specify otherwise, the compiler treats members as if you said `public:`, whereas for `class`, the assumption is `private:`.

    - Why should you not allow data members to be public?

      So users cannot manipulate data they are not supposed to have access to (they might set it to an unexpected value and cause the class/struct to behave unexpectedly)

    - What is the purpose of having private member functions in a class? Can you give some examples of when they would be used?

      Private member functions are useful as helper functions for public methods. We don't want these functions to be called outside the function we defined them for, so we make them private to prevent the user from using those functions in isolation. In general, if we don't want a user to use the functions, we make them private.

    - What happens if you forget to deallocate memory once you're done with the object?

      There will be a **memory leak**. This happens when a program finishes but memory stored on the heap is not deallocated. Normal variables are stored on the stack and are deallocated automatically when a

variable goes out of scope (the brackets) while dynamically allocated variables are created on the heap and must be explicitly deleted to prevent a memory leak. This is one of the reasons why we test our programs on g31, as opposed to relying solely on the Xcode/Visual Studio compilers; the added compiler flags enabled in g31 will help catch memory leaks.

- (True/False) A class may have more than one constructor.

  True. You can have constructors that take different arguments to initialize different member variables. This is called overloading.

```
class MyClass {
      public:
            MyClass() {
                  a = 0;
                  b = 0;
            }
            MyClass(int a, int b) {
                  // we need to use 'this' because our
                  private member variables have the same name
                  as our arguments (the 'this' pointer refers
                  to our member variables)
                  // often, member variables are prefixed
                  with m_ (ex: m_a) to avoid this issue

                  this->a = a;
                  this->b = b;
            }
      private:
            int a;
            int b;
};
```

- (True/False) A class may have more than one destructor.

  False. Since a destructor cannot have parameters or a return type, it is impossible to create more than one destructor for a class.

- If you have an object pointed to by a pointer, which operator is used with the pointer to access the object's members?

  You can either dereference the object and access the members normally or use the **->** (left arrow) operator.

```
(*object).member means the same thing as object->member
// this includes functions! ex: object->member_function()
```

2. Write a class Person that has two private data members: (Michelle Bai)
    - `m_age` (an int)
    - `m_catchphrase` (a string).

   The Person class should have a default constructor that initializes its data members to reasonable values and a second constructor that initializes the data members to the values of its parameters. In addition, Person should have three public member functions:
    - `getAge()`, which returns the Person's age
    - `haveBirthday()`, which increments the Person's age by 1
    - `speak()`, which prints the Person's catchphrase.

```
class Person
{
       public:
            Person()
            {
                  m_age = 0;
                  m_catchphrase = "";
            }

            Person(int age, string catchphrase)
            {
                  m_age = age;
                  m_catchphrase = catchphrase;
            }
            int getAge() const
            {
                  return m_age;
            }

            void haveBirthday()
            {
                  m_age++;
            }

            void speak() const
            {
                  cout << m_catchphrase << endl;
            }

      private:
            int m_age;
            string m_catchphrase;
};
```

3. A line in Euclidean space can be represented by two parameters, **m** and **b** from its slope-intercept equation **y = mx + b.** Here **m** represents the slope of the line and **b** represents the line's y-intercept.
Write a class that represents a line. Your class must have a simple constructor that initializes the line's **m** and **b**. Next, define a member function with the following prototype:

```
double intersection(Line line2);
```

This function must compute the x-coordinate where this line and another line (line2) intersect.

```
double m1 = 2;
double b1 = 3;
double m2 = -2;
double b2 = 7;
Line line1(m1, b1);
Line line2(m2, b2);
cout << line1.intersection(line2) << endl;    // prints 1.0
```

This function must compute the x-coordinate where this line and another line (line2) intersect.

```
class Line
{
      public:
            Line(double m, double b)
            {
                  m_m = m;
                  m_b = b;
            }

            double m() const
            {
                  return m_m;
            }

            double b() const
            {
                  return m_b;
            }

            double intersection(Line line2)
            {
                  if (m_m == line2.m())
                  {
                        // same slope! SO the lines either are
                                    coincident or parallel
                        // spec doesn't specify what we should do here,
                                    so return
```

```
                        // whatever; in the real world we may want to
                                    throw an exception
                        // (which aren't discussed in CS 31)
                        return 0;
                  }
                  return (line2.b() - m_b)/(m_m - line2.m());
            }

            private:
                  double m_m;
                  double m_b;
      };
```

Bonus: There are a few ways in which this problem specification is incomplete; they are not related to C++, but to the problem domain. What are they?

ANS: As mentioned in the comments above, the spec does not tell us what we should return if the two lines are coincident or parallel.

4. Write a program that **repeatedly** reads an age and a catchphrase from the user and uses them to **dynamically** allocate a Person object, before calling the Person's speak() function and then **deallocating** the Person object. (Aki)

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

int main()
{
      int age;
      string catchphrase;
      while(true) // repeats indefinitely
      {
            cout << "Please enter an age: ";
            cin >> age;
            cin.ignore(10000, '\n');
            cout << "Please enter a catchphrase: " << endl;
            getline(cin, catchphrase);
            Person* p = new Person(age, catchphrase);
            p->speak();
            delete p;
      }
}
```

5. Write a class called Complex, which represents a complex number. Complex should have a default constructor and the following constructor: (Michelle Lee)

```
Complex(int real, int imaginary);
// -3 + 8i would be represented as Complex(-3, 8)
```

Additionally, the class should contain two functions: sum and print. Calling sum should set the calling object to the sum of the 2 complex numbers passed as arguments. Print should print which complex number the object represents. You may declare any private or public member variables or getters/setters you deem necessary. Your code should work with the example below.

```
int main() {
(1)   Complex c1(5, 6);
(2)   Complex c2(-2, 4);
(3)   Complex* c3 = new Complex();

(4)   c1.print();
(5)   c2.print();
(6)   cout << "The sum of the two complex numbers is:" << endl;
(7)   c3->sum(c1, c2);
(8)   c3->print();
(9)   delete c3;
}

// The output of the main program:
5+6i
-2+4i
The sum of the two complex numbers is:
3+10i
```

Bonus: What would happen if we swapped line (8) and (9)?

```
class Complex {
    int m_real;
    int m_imaginary;
      public:
    Complex() {}
    Complex(int real, int imaginary) {
        m_real = real;
        m_imaginary = imaginary;
    }
    void print() {
        cout << m_real << "+" << m_imaginary << "i" << endl;
    }
    void sum(Complex c1, Complex c2) {
        m_real = c1.m_real + c2.m_real;
        m_imaginary = c1.m_imaginary + c2.m_imaginary;
    }
};
```

What would happen if swapped the order of (8) and (9)? How would it change the output?

After deleting the object pointed to by c3, an attempt to follow the pointer c3 is undefined behavior. The program might crash, print weird values (perhaps because the memory used by the deleted object was overwritten with some bookkeeping information the storage manager uses), print 3+10i (if the memory used was not overwritten), or do something else.