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.

Concepts: Review

1. Conceptual Questions
   ○ What’s the main difference between declaring a type with the keyword struct and declaring it with the keyword class?
     
     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
   ○ What is the purpose of having private member functions in a class? Can you give some examples of when they would be used?
     
     when we don’t need the user to know them
   ○ What happens if you forget to deallocate memory once you’re done with the object?
     
     there will be a memory leak
   ○ (True/False) A class may have more than one constructor.
     
     True.
(True/False) A class may have more than one destructor.

False.

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

use ->

2. Write a class Person that has two private data members:
   - 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.

```cpp
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++;
        }

        string speak()
        {
            return 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:

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

This function must compute the x-coordinate where this line and another line (line2) intersect. You may assume that line2 is guaranteed to intersect this line at a single point.

```cpp
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
```

```cpp
class Line
{
    public:
        Line(double m, double b)
        {
            _m = m;
            _b = b;
        }

        double m() const
        {
            return _m;
        }

        double b() const
        {
            return _b;
        }

        double intersection(Line line2)
        {
            double x = (b2 - _b) / (m1 - line2.m);
            return x;
        }

    private:
        double _m;
        double _b;
};
```
As one might be able to tell from the specs, there is the possibility of the two lines being coincident or parallel. What are some reasonable ways we can handle this case if it was allowed to happen?

Also, notice vertical lines cannot be exactly defined using the framework we have (e.g. x=3), although they can be approximated using a line with a large m.

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.

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

int main()
{
    int age;
    string catchphrase;
    while(true)
    {
        cout << "Please enter an age: ";
        cin >> age;
```
cin.ignore(10000, 'n');
cout << "Please enter a catchphrase: " << endl;
generate(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:

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

Additionally, the class should contain two functions: setToSum and print. The function setToSum should set this object to the sum of the two input Complex objects. The function 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->setToSum(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 swapped the order of (8) and (9)?

```c++
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 setToSum(Complex c1, Complex c2) {
    m_real = c1.m_real + c2.m_real;
    m_imaginary = c1.m_imaginary + c2.m_imaginary;
}
};

If we swap (8) and (9), 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.