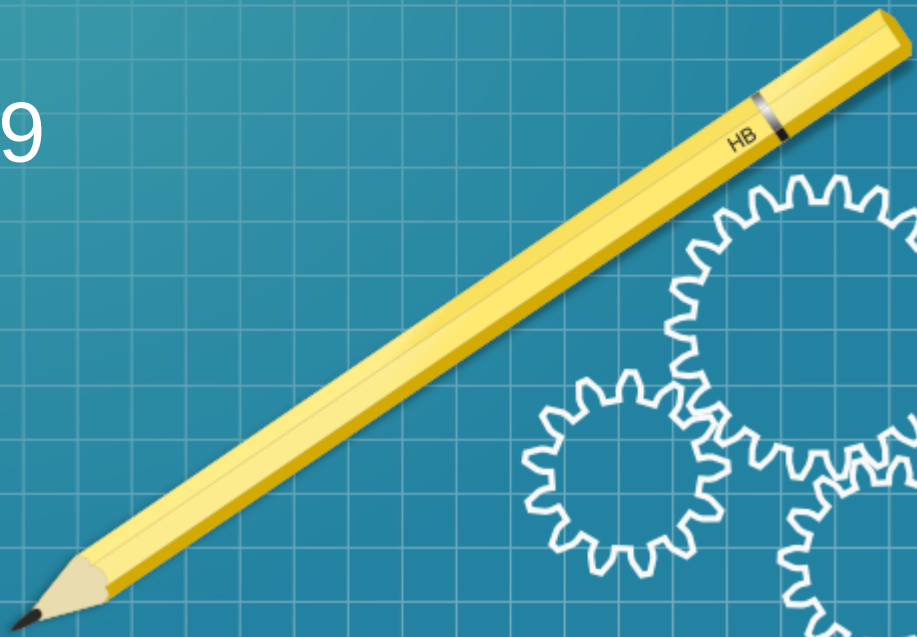
Classes: A Deeper Look

Chapter 9



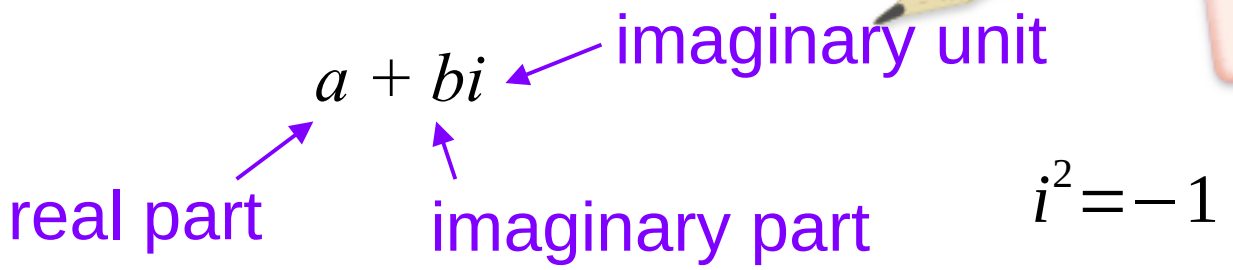
Today we will

Define and use `class Complex`, representing complex numbers in rectangular form: $a + bi$.

While working on it, we will discuss the following terminology:
- interface and implementation
- include guards in header files
- throwing exceptions

After we finished working with it, we will see how we can access public class methods through
- *objects*,
- *references* and
- *pointers*.

Complex numbers (rectangular form)



To-do list:

- use `double` variables to represent the `private` data of the class
- provide constructor that enables an object of this class to be *initialized when it's declared*
- the constructor should contain *default values* in case no initializers are provided
- provide `public` member functions that perform the following tasks:
  - `add`: adds two complex numbers
  - `subtract`: subtracts the two complex numbers
  - `toString`: returns a string representing the complex number

Complex numbers (rectangular form)

How about multiplication and division of complex numbers?

$$(a+bi)(c+di)=\ldots=(ac-bd)+(ad+bc)i$$

$$\frac{a+bi}{c+di}=\frac{(a+bi)(c-di)}{(c+di)(c-di)}=\ldots=\frac{(ac+bd)+(bc-ad)i}{c^2+d^2}=$$

$$=\frac{ac+bd}{c^2+d^2}+\frac{bc-ad}{c^2+d^2}i$$

Also, don't forget that division by 0 is undefined!

See complexNumber.h, complexNumber.cpp, and testingComplexNumber.cpp

Complex numbers (rectangular form)

Next, let's see how we can access public class methods through *objects*, *references* and *pointers*.

See complexNumber.h, complexNumber.cpp, and testingComplexNumber2.cpp

Constructors and Destructors

We already know that *constructor* is called when an object is created.

Similarly, the *destructor* is called when an object's lifetime ends:

- program is terminated, or
- end-of-scope is reached, or
- explicit `delete` statement is called, …

Destructor does not release the object's memory (it is done by other entity), it preforms *termination housekeeping*: closes opened files, releases dynamically allocated memory, etc.

Constructors and Destructors for Objects in Global Scope

Constructors are called for objects defined in *global scope* (*global namespace scope*) before any other function (including main) begins execution.

The corresponding *destructors* are called when `main` terminates.

Function `exit()` forces the program to terminate immediately and does not execute the *destructors* of local objects. `exit()` is often used when a fatal unrecoverable error occurs.

Function `abort()` performs similarly to `exit()` but forces the program to terminate immediately, without allowing programmer-defined clean up code of any kind to be called. `abort()` is usually used to indicate an abnormal termination.

Appendix F has more information on `exit()` and `abort()`

Constructors and Destructors for Non-static Local Objects

The *constructor* for a non-static local object is called when execution reaches the point where that object is defined.

The corresponding *destructor* is called when the execution leaves the object's scope.

Constructors and destructors of non-static local objects are called each time execution enters and leaves the scope of the object

Exception: when `exit()` or `abort()` functions are called, the destructors are not called.

Constructors and Destructors for static Local Objects

The *constructor* for a *static local object* is called only once, when execution reaches the point where that object is defined first time.

The corresponding *destructor* is called when `main` terminates or the program calls function `exit()`.

Global and *static objects* are destroyed in the reverse order of their creation.

Destructors are not called for static objects if the program terminates with a call to function `abort()`.

Constructors and Destructors

Let's take a look at an example that demonstrates the order in which constructors and destructors are called for *global*, *local* and *local static* objects. We will use the following class:

```
class CreateAndDestroy {

public:
   CreateAndDestroy(int id, std::string msg) :
    ObjectID(id),message(msg) {
      cout << "Object " << ObjectID << "
         constructor runs \t" << message << endl;
   }

   ~CreateAndDestroy() {
      cout << "Object " << ObjectID << " destructor
         runs \t" << message << endl; }

private:
   int ObjectID; // id of the object
   std::string message; // describes the object };
```

HW assignment

1) Exercise 9.23

Self Study

Read *Section 9.6.2* and then do exercise 9.17

Suggested exercises

(*not for grade, but the questions related to these will appear on a quiz or a test*):

1) Chapter 9, Summary and all Self-Review Exercises

2) Chapter 9, Exercise: 9.3, 9.14