PIC 10A: Homework 5 and 6

Michael Andrews
UCLA Mathematics Department

March 1, 2021

Start early!

These homeworks have tricky parts to them, so you should start them ASAP.

In the first homework, you will write a `class` called `ComplexFrac` which will build on the `Frac` class I made in lecture.

In the second homework, you will write your own `string` class: `MyString`.

Homework 5

A template to answer this homework is provided on CCLE. It contains five files:

- `Fracs.hpp` and `Fracs.cpp` contain the `Frac` class which I defined in lecture. The class declaration is in the `.hpp` file. Any leftover member function definitions appear in the `.cpp` file.

  YOU SHOULD NOT CHANGE THESE TWO FILES.

  You may notice that I have destroyed the default constructor. I have done this on purpose. YOU ARE NOT ALLOWED TO FIX THIS. So do not ever call the default constructor for `Frac` explicitly or implicitly (by not using an initializer list in `ComplexFrac`).

- `main.cpp` contains some demonstration code. You can edit this file as much as you like, and you should edit this file in order to test the functionality of the new class that you create.

- `ComplexFracs.hpp` and `ComplexFracs.cpp` are the files which you will turn in.

  Currently there is a declaration of a class called `ComplexFrac` in the `.hpp` file. It has two `private` member variables: `real` and `imag`. It has one `public` constructor with no parameters which creates an instance of `ComplexFrac` with `real==0` and `imag==0`. It has one `public` member function called `print`. The definition of `print` is given in the `.cpp` file. You should read and understand what it is doing.

The goal of this question is to add more functionality to the class `ComplexFrac` so that it can perform complex number arithmetic (amongst other things).

A *complex number* is a number of the form $a + bi$ where a and b are real numbers. In order to use the `Frac` class from lectures, we will take a and b to be fractions. All you need to know about complex numbers to answer this question is that $a + bi = c + di$ exactly when $a = c$ and $b = d$, and $+, -, \times, \div$ are defined as follows:

$$\begin{aligned}
(a+bi) + (c+di) &= (a+c) + (b+d)i \\
(a+bi) - (c+di) &= (a-c) + (b-d)i \\
(a+bi) \times (c+di) &= (ac-bd) + (ad+bc)i \\
(a+bi) \div (c+di) &= (a+bi) \times \left(\frac{c}{c^2+d^2} + \frac{-d}{c^2+d^2} \cdot i \right).
\end{aligned}$$

Notice that I have defined division in term of multiplication (this is on purpose - see task 3).

Look over the page for your list of tasks!

1. **Constructors.** I have already given `ComplexFrac` a default constructor. Define four more constructors:

   - The first should have 1 parameter of type `Frac`.
   - The second should have 2 parameters of type `Frac`.
   - The third should have 1 parameter of type `int`.
   - The fourth should have 4 parameters of type `int`.

   What should they do? You can hopefully guess, but…

   (a) The first is for constructing a complex number with imaginary part equal to `0`.

   (b) The second does the "obvious" thing.

   (c) The third takes an integer n and constructs the complex number $\frac{n}{1} + \frac{0}{1}i$
   (just like how when I wrote `Frac`, one constructor used n to construct the fraction $\frac{n}{1}$).
   It should help to recall the different constructors that `Frac` has.

   (d) The fourth takes `ints` $\mathrm{re}_{\mathrm{num}}, \mathrm{re}_{\mathrm{den}}, \mathrm{im}_{\mathrm{num}}, \mathrm{im}_{\mathrm{den}}$ to make $\frac{\mathrm{re}_{\mathrm{num}}}{\mathrm{re}_{\mathrm{den}}} + \frac{\mathrm{im}_{\mathrm{num}}}{\mathrm{im}_{\mathrm{den}}}i$. It should help to recall the different constructors that `Frac` has.

   Why do you think I chose not to make a constructor using two `ints`?

2. **Comparison.** Define a member function `isEqual` that performs a similar job to the function of the same name belonging to `Frac`.

3. **Arithmetic.**

   Declare member functions `add`, `subtract`, `multiply`, and `divide` in the `.hpp` file. If the parameters and/or `return` type are unclear, look back to what I did with `Frac`. Define these member functions in the `.cpp` file.

   You should expect defining multiplication and division to be a bit awkward. In some ways, it would be easier if we had overloaded the `+`, `-`, `*`, `/` operators, but we're delaying a detailed discussion of such things until PIC 10B.

   Your division function can call your multiplication function once you create $\frac{c}{c^2+d^2} + \frac{-d}{c^2+d^2} \cdot i$.

4. **Public versus private.** The member functions you defined in 1-3 should all be `public`. If you need to use auxilary functions, they should be `private`.

   THERE SHOULD BE NO PUBLIC MEMBER VARIABLES.

   THERE SHOULD BE NO PUBLIC GETTERS OR SETTERS.

5. **Const. correctness.** Make sure that you are `const` correct.

6. **Testing.** Make sure to use your `main.cpp` to test your functions.

   Wolfram Alpha has a good complex number calculator.

Homework 6

A template to answer this homework is provided on CCLE. It contains six files:

- main.cpp. It's very short and shows you the capability of MyString that you start with.

  By allowing you to print and assign normally, you can easily test the member functions that you define.

- MyString.hpp and MyString.cpp are the files which you will turn in.

- MyString.hpp. This contains the definition of the class MyString.

  DO NOT DELETE #include "students-ignore1.h" or #include "students-ignore2.h".

  Always have the end of this file read:

  ```
  #include "students-ignore1.h"
  };

  #include "students-ignore2.h"
  #endif /* MyString_hpp */
  ```

  The beginning says

  ```
  public:
      MyString()       : s(0)   {}
      MyString(char c) : s(1,c) {}

      MyString substr(size_t pos, size_t len = -1) const;

      size_t  find(char c, size_t pos =  0) const;
      size_t rfind(char c, size_t pos = -1) const;

      size_t  find(const MyString& str, size_t pos =  0) const;
      size_t rfind(const MyString& str, size_t pos = -1) const;

  private:
      std::vector<char> s;
  ```

  We'll store the string as a vector of chars which we keep private.

  You can see a constructor with no parameters which creates the empty MyString.

  You can see a constructor with one parameter which creates a one character MyString.

  I have provided declarations of substr, and overloaded find and rfind.

- MyString.cpp. This contains incomplete definitions of substr, find, and rfind.

- `students-ignore1.h`, `students-ignore2.h`, `students-ignore3.h`.

  Include these files, but you can safely ignore what they say. Why have I put these in?

  - They allow you to `cout` and `cin` as usual.
  - They allow you to concatenate as usual.
  - They allow you to initialize using a `string literal`.
  - They also allow you to use `find` and `rfind` with a `string literal`, provided you have defined these functions for a `MyString`.

Here are your tasks...

1. This part requires you to make lots of very short definitions. Make these definitions in the class declaration. Each of mine is one line long. **Remember that your member variable `s` has all of the member functions of `vector<char>` available to it.**

   (a) Define a new constructor that has two parameters: `size_t n, char c`. It should create a new `MyString` with `n` characters all equal to `c`.

   (b) Define member functions `size`, `length`, `empty`, `push_back`, `pop_back` just like for `string`. **See here for the correct function signatures.** (`size` and `length` are synonyms.)

   (c) Define an overloaded member function `resize` (each definition is still just one line).

   ```
   void resize (size_t n);
   void resize (size_t n, char c);
   ```

   This should resize the `MyString` to a length of `n` characters.

   If `n` is smaller than the current length, the `MyString` should be shortened to its first `n` characters, removing the characters beyond the `n`th.

   If `n` is greater than the current length, the `MyString` should be extended by inserting at the end as many characters as needed to reach a size of `n`.

   If `c` is specified, the new elements should be initialized as copies of `c`, otherwise, they take on the value `'\0'`.

2. In `MyString.cpp`, give appropriate definitions for `substr`, and overloaded `find` and `rfind`.

   DO NOT DELETE `#include "students-ignore3.h"`.

   Some important comments:

   - In `MyString.cpp` I have put the "empty" function definitions in order from easiest to most difficult.

     You can find good descriptions of what these functions are supposed to do here, although you should be familiar with what they are supposed to do already!

   - The first declaration of `find` reads

     ```
     size_t find(char c, size_t pos = 0) const;
     ```

     This means that if `str` is an instance of `MyString`, calling `str.find('!')` is the same as calling `str.find('!',0)`. When `pos` is not specified it takes on the *default value* of `0`.

     You should write your definition to work for any value of `pos`.

   - Notice that a `size_t` can never be negative. When I `return -1` in the "empty" definitions, that is cast to a `size_t` and `static_cast<size_t>(-1)` is the biggest `size_t` there is: $2^{64}-1$ on most machines. Don't mess up by writing a `while` loop for `rfind` that says something like `while(pos >= 0)`. Such a `while` loop would go on forever. Also, don't hack your way around this by converting to `int`s: that loses information and could create another bug. Instead use a `while` loop like `while(pos != -1)`. In order to make this comparison, `-1` is correctly cast as a `size_t`.

   - Because `size_t` cannot be negative you should be careful about subtracting numbers. In `substr` you may want to check whether `pos < s.size()` before performing the subtraction `s.size()-pos`.

   - Finally, here is a new `main.cpp` to test `substr`, `find`, and `rfind`.