CSC 211 Assignment 2

A DNA Class

Due Wednesday, March 22nd by 11PM

Lateness

For this assignment, submissions after the deadline will be penalized by 20% per day, with no submissions accepted after 11PM on Friday, March 24th. Exceptions to this policy will not be granted except under extraordinary circumstances.

Background

You've worked with DNA sequences several times this semester.

Now you're going to implement a **class** to represent DNA sequences, including reading from and writing to the FASTA format.

Getting Started

- Get into your Docker development environment (and into the `data` subdirectory)
- Download the assignment framework with `git clone https://github.com/csc211/a2`
- You will see this `Readme.md` along with a C++ **header** file (`dna.h`) and a compile script `compile`
- You **cannot** yet compile, because you need to create at least one other file first.

Requirements

- You **must implement** the DNA class.
  - This means that you create a file `dna.cpp` and implement all the methods defined in the header file
- All you will submit to gradescope is this *implementation file*.
- However, in order to *test* your code, you'll need to write a `main.cpp` that tests out your class implementation.

- You are responsible for making sure that every method of the `DNA` class is implemented according to the specifications given in the header file.
- **You must not change the header file**
  - This is because when the autograder compiles your `dna.cpp`, it will use **my** `dna.h` and `main.cpp`.
  - My header file is the same as yours; my main.cpp is a secret.

Helpful hints

READ THE HEADER FILE THOROUGHLY

- I cannot stress this enough. It is the canonical specification for this assignment.

Revision control

Use git to keep track of changes. Since you have used git to clone the assignment repository, it's already a git repository, so you don't need to initialize it.

- `git add *` followed by `git commit -m "some helpful notes"`
- you can use `git log` to remind yourself what you've done
- you can use `git revert` or `git reset` to *undo* changes if you manage to break something.
- Here's a helpful tutorial for `git revert` and reset.

Helper functions

You should not need any additional helper functions. Adding them to the class without changing the header file is a more advanced topic, and you should not change the header file!

Comments

Comments are a little different here. Since I've provided the interface (the header file), you don't need to explain what your methods do. I've already taken care of that.

What you should do is explain your *representation.* For example, how do you choose to represent the sequence? Anything clever or subtle (such as iterating through a string backwards, or using a non-obvious data structure like a map) should be explained.

Compiling

The `compile` script for this assignment is a little more complicated than you've seen. It takes an optional argument, `-nolink`.

- If you call `./compile` as you have in the past, it will build an executable, `dnatest`, as long as you have a `main.cpp` with a `main()` function. You will be able to run that `dnatest` executable however you have designed it to work.
- If you call `./compile -nolink` it will *compile* the dnatest.cpp into an *object file* (you will see `dna.o` if you do `ls` on the command line).
- What exactly this object file contains is the subject of CSC411, but a high-level description is that it is the class, compiled, with no main function, and thus not a complete program. With a few more commands, (primarly the `ar` command, if you're curious) it could be turned into a *library*.
- For your purposes, `./compile -nolink` is a way to make sure your class implementation *compiles* without worrying about having a working `main.cpp` yet. To test your class implementation, you'll need a `main.cpp` of your design.

Exceptions

You'll note that the header file requires two methods to `throw` exceptions in the case of invalid input.

Remember how the official solution for revcomp.cpp had the somewhat awkward error-handling method of returning an empty string if it encountered bad input? You're going to get a little exposure to the "right" way to inform client code of errors in C++.

For now, you don't have to *handle* exceptions. You just have to say something like:

```
std::runtime_error("informative error message here");
```

when you encounter a bad input (hint: you'd better use a more informative error message!)

We will talk more about exceptions in class.

Use Valgrind

Just like with previous labs and assignments, I encourage you to use valgrind to ensure you have no memory errors.

Grading Rubric

For this assignment, correctly passing all tests on Gradescope is worth 80% of your grade. The remaining 20% is based on reasonable commenting habits, and the structure and organization of your code.

Note: in future assignments, we may incorporate your `git log` as part of the grading rubric. So, it is wise to get into the habit of checking in your changes to git with this first assignment.

Submitting

You will submit `dna.cpp` via Gradescope, where its functional correctness will be graded automatically. You can submit as many times as you like; only the last submission will count towards your grade.

Contents of `dna.h`

```
#ifndef _dna_h
#define _dna_h

#include <iostream>
#include <string>
#include <vector>

/*
A class to represent DNA sequences, related to the FASTA format.
A DNA object has two member fields: a header, which is a sequence of characters beginning wi
and a sequence, which is a sequence of characters containing only A, C, T, G, and N.
*/

class DNA {

public:

    /*
    Default constructor
    */
    DNA();

    /*
    Constructor with separate arguments for header and sequence
    If `sequence` contains invalid characters, throws a std::runtime_error
    */
    DNA(std::string header, std::string sequence);

    /*
    Constructor that expects an input file stream to a FASTA-formatted file
    If the first line is not a valid header (does not begin with a `>`)
    or if the sequence part of the file contains invalid characters
    (other than newlines, which are removed),
    throws a std::runtime_error
    */
    DNA(std::ifstream infile);

    /*
    Getter methods
    */
    std::string getSequence();
    std::string getHeader();

    /*
    Produces FASTA-formatted output as a string,
    with the sequence wrapped to `columns` number of characters (default 80)
    */
    std::string toFasta(int columns=80);

    /*
    Returns a new DNA object whose `header` is the same as this object's,
    and whose underlying `sequence` is the reverse complement as this object's.
    */
    DNA revcomp();

    /*
    Searches the sequence for `query`, a string, AND its reverse complement.
    Returns the index of the first instance of `query` or its reverse complement,
    or `string::npos` if not found.
    */
    size_t find(std::string query, size_t start);

private:

    /*
    the equality operator does not test full equality.
    It returns true if and only if d1 and d2 have the same sequence
    (even if their headers differ), and false otherwise.
    */
    friend bool operator==(DNA d1, DNA d2);

    /* instance variables */
    std::string seq;
    std::string hdr;

};


#endif
```