CSC 211 Assignment 1

DNA Reverse Complement

Due Friday, February 17th by 11PM

Background

DNA (deoxyribonucleic acid) is the molecule that carries the genetic information of all life on earth. DNA is made up of four distinct *bases*: adenosine, cytosine, guanine, and thymine, and the sequence of these bases determines the genetic basis of an organism. While in computer programs we use *bits* of information, which have two possible values, each base of a DNA molecule has four possible values; we use the letters A, C, G, and T to represent the four bases.

You may have heard that DNA forms a *double helix*. It does this because two strands of DNA bind together in a peculiar manner: each A binds to a T in the other molecule, each T with an A, each C with a G, and each G with a C. This particular pairing is called **Watson-Crick pairing**.

Here is an example of one strand with the sequence `GATTACA` paired with its reverse complement, `TGTAATC`. Note that the reverse complement is read *in reverse* because the molecules themselves have a direction (the biochemistry of this is beyond the scope of this class).

```
GATTACA
|||||||
CTAATGT
```

Since some genes are on what we'll call the "forward" strand, and others are on the "backward" strand, one basic task in molecular biology is to identify the **reverse complement** of a DNA sequence. The reverse complement is the complementary bases, read backwards.

Your job for this assignment is to write a C++ program that computes the reverse completement of a DNA sequence provided as a command-line argument.

Reverse Complement

Given a DNA sequence like `GATTACA`, your program will return the reverse complement, which in this instance would be `TGTAATC`. Each base is replaced

with its **complementary** base; essentially, each `G` is replaced with a `C`, each `C` with a `G`, each `A` with a `T`, and each `T` with an `A`, and the result is reversed, hence *reverse complement*.

Getting Started

- Get into your Docker development environment
- Download the assignment framework with `git clone https://github.com/csc211/a1`
- You will see this `Readme.md` along with a C++ source file (`revcomp.cpp`) and a compile script `compile`
- You can now compile the assignment by typing `./compile` and it will produce an executable program called `revcomp`. However, it won't do anything yet!

Requirements

Your program must take one command-line argument, which should be the DNA sequence (specified as all capital letters) for which you are to compute the reverse complement.

If given a valid all-caps DNA sequence, your program must print its reverse complement to *standard output* (this is just printing using `printf()` or `cout <<`) the reverse complement followed by a newline (`"\n"` or `std::endl`) and nothing else, and it should exit with a return code of 0.

If your program is given an *invalid* DNA sequence (e.g. containing anything other than a sequence of capital A,C,T, and G) then it must print nothing, and (importantly) exit with a return code of 1 (or `EXIT_FAILURE`). This can be accomplished with `exit(EXIT_FAILURE)` which are defined in `<cstdlib>`.

If your program is given fewer than or more than 1 command-line argument, it must also exit with a return code of 1.

Helpful hints

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 revcomp.cpp` 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.

Write helper functions where it makes sense. You should *not* have all of your code in `main()`. In particular, the code to compute the reverse-complement of a DNA sequence should not 'know' about the code to parse `argv`. It is fine to mix the validity-checking of the DNA sequence with the reverse-complement code.

Comments should explain what your functions do. They should not narrate the code as if the reader does not know C++.

Compile early and often!

The compiler is your friend, not your enemy. Compile after you write a little bit of code; it's far easier to find a problem if you know it's in the last few lines of code you wrote.

Use Valgrind

Just like with Lab 1, I encourage you to use valgrind to ensure you have no memory errors.

Something like `valgrind ./revcomp ACTGGATA` will report any memory errors in your program.

Don't be concerned by "still reachable" memory in the Leak Summary. Just pay attention to the Error Summary.

```
student@cc635484f6a3:~/data/assignments/a1-soln$ valgrind ./revcomp "GATTACx"
==58== Memcheck, a memory error detector
==58== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==58== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==58== Command: ./revcomp GATTACx
==58==
==58==
==58== HEAP SUMMARY:
==58==     in use at exit: 72,704 bytes in 1 blocks
==58==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==58==
==58== LEAK SUMMARY:
==58==    definitely lost: 0 bytes in 0 blocks
==58==    indirectly lost: 0 bytes in 0 blocks
==58==      possibly lost: 0 bytes in 0 blocks
==58==    still reachable: 72,704 bytes in 1 blocks
==58==         suppressed: 0 bytes in 0 blocks
```

```
==58== Rerun with --leak-check=full to see details of leaked memory
==58==
==58== For counts of detected and suppressed errors, rerun with: -v
==58== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result codes

I've told you to return 0 or 1 to indicate successful or unsuccessful result codes. But how can you check that your program is doing the right thing?

You can check the result code immediately after running your program by telling the shell to display (`echo`) the special variable `$?`.

Here are two examples of me running my solution and checking the error code:

```
student@cc635484f6a3:~/data/assignments/a1-soln$ ./revcomp "GATTACA"
TGTAATC
student@cc635484f6a3:~/data/assignments/a1-soln$ echo $?
0
student@cc635484f6a3:~/data/assignments/a1-soln$ ./revcomp "Gattaca"
student@cc635484f6a3:~/data/assignments/a1-soln$ echo $?
1
```

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 revcomp.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.