Homework 3

EE 262 (Fall 2021)

Department of Electrical and Computer Engineering
Clarkson University

Instructions

Note 1: Please read all instructions carefully before submitting your work. There are **2 questions** in this homework for a total of **70** points.

Note 2: Solve all problems and upload your answers to Moodle using the instructions below. Make sure any code you write works on Polaris before uploading your files. You will likely lose many points if your code doesn't compile. Whenever you write solutions on paper, you need to scan all documents before submitting.

Note 3: For this homework, you can work individually or in groups of up to 2. There should be only one submission per group.

Note 4: `USR` stands for your login ID on Polaris (`polaris.clarkson.edu`). `MJR` represents your major, i.e. CE, SE, EE, CS, etc.

Note 5: In program listings and shell interactions, any **`bold, underlined, constant-width`** text indicates that it is user input.

Note 6: In program listings and shell interactions, any ***`italicized, underlined, constant-width`*** text has to be explicitly replaced by an appropriate user-supplied value.

Note 7: Do not upload any executable or intermediate files as answers to problems, unless specifically asked to do so.

Note 8: You should upload a single file named `USR_MJR_HW3.zip` (or named using the pattern `USR1_MJR_USR2_MJR_HW3.zip` if submitting in a group or two, provided group submissions are allowed) to Moodle. If you have multiple majors, please mention all of them in the file name, e.g. `USR_MJR1_MJR2_HW3.zip`; if in a group of two, the file name will have the pattern `USR1_MJR1_MJR2_USR2_MJR_HW3.zip`, if `USR1` has two majors and `USR2` has one major, provided group submissions are allowed.

Note 9: In the ZIP file, you should have a directory named `hw3`, inside which you should have directories for each problem, named to clearly indicate the problem number, i.e. `p1`, `p2`, etc, each of which should contain the appropriate deliverable. To verify the directory structure before creating the zip file, the `tree` command on Polaris can be used to display the directory contents in a hierarchical fashion, e.g. `tree dirName`.

Note 10: Please read the "Note on academic honesty" in the syllabus. In the directory for *each* problem, please include a `readme.txt` file containing the following:

- It should state whether you worked alone on that probem or list the full names of all the people you discussed it with. This includes the names of the course staff, if any of them helped you with the problem.

- If your answer includes or is inpired by a non-trivial amount of content (including code) from specific sources, please cite all of them prominently.

Note 11: On Polaris, the following command can be used to zip the contents of directory `hw3` into a file: `dest.zip`: `zip -9 -yrq dest.zip hw3`

Note that you will need to rename the ZIP file as per the instructions described earlier. Please test the generated zip file by listing its contents (`unzip -l dest.zip`) and also by extracting and viewing its contents in a separate (temporary) directory (`unzip dest.zip`).

1. [20 points] In `palindrome.cpp`, write a complete C++ program containing a function with the following signature:

   ```
   bool isPalindrome(const char*)
   ```

   This function determines if its argument is a palindrome or not. A sample interaction is shown in Listing 1.

   The following are palindromes:

   - malayalam
   - racecar
   - rac e car
   - able was I ere I saw elba

   Note: For the purposes of this problem, "race car" is not a palindrome, i.e. the white spaces are significant and have to match too.

   To determine the length of a C-style string, you can use the strlen function (http://www.cplusplus.com/reference/cstring/strlen/) for which you will need to include `<cstring>`.

   Your program should accept a single command line argument and print "yes" if it is a palindrome, otherwise it should print "no".

   Note: If the number of command line arguments is incorrect, print a helpful error message describing the correct usage of your program.

   Listing 1: Sample interaction with an executable generated from `palindrome.cpp`.

```
$ g++ palindrome.cpp  -o palindrome
$ ./palindrome ann
no
$ ./palindrome ab ba
usage: palindrome string_to_check
$ ./palindrome
usage: palindrome string_to_check
$ ./palindrome abba
yes
$ ./palindrome test
no
$ ./palindrome "racecar"
yes
$ ./palindrome "rac␣e␣car"
yes
$ ./palindrome "race␣car"
no
$ ./palindrome "able␣was␣i␣ere␣i␣saw␣elba"
yes
$ ./palindrome jaja
no
```

Deliverable: `palindrome.cpp`, `readme.txt`.

2. [50 points] Study Chapter 10 of the textbook. Solve the problem described in Programming Project 7 in Chapter 10 (class for rational numbers).

   The class should be named `rational` and should be in `rational.cpp`. As you begin to design the class, think about the class data members, i.e. how rational numbers will be represented, and then about the operations (i.e. functions) that you need to provide.

   If an attempt is made to create a rational number with denominator 0, print an error message on the standard error (`std::cerr`), and immediately exit with status -1.

   Make sure to always check for a denominator being 0 whenever you are writing functions for this class.

   Copy the contents of the `main` in `driver1.cpp` to your file to test your class; the output produced should be the same as shown in Listing 2. Also, the input function must expect two integers, the first of which is to be interpreted as the numerator. You need to provide all necessary constructors needed for running the client code in `driver1.cpp`.

   Listing 2: Sample interaction with an executable generated from `rational.cpp`.

```
$ g++ -o rational rational.cpp
$ ./rational
r1: 10/3
r2: -4/1
r3: 4/1
r4: -10/3
r6: 22/3

r5 will get input from ifstream (connected to inp1.txt):
enter numerator: enter denominator: r5: -19/3

r7: 2/3
r8: 44/9
r9: -44/57
r9 < r1: 1
r4 < r5: 0
r6 < r8: 0
r1 < r3: 1

y1: 70/1

y2: 0/1
y2 will now get input from cin:
enter numerator: 34
enter denominator: -6
y2: -17/3

Denominator can't␣be␣0.
```

   Note: Whenever you output a rational number, it should be in its reduced form, i.e. you must output 4/3 instead of 8/6. [Hint: just divide both the numerator and denominator by their greatest common divisor (gcd).]

   Note: `rational` objects should be immutable, i.e. once created, their value cannot be modified. Member functions `add`, `sub`, `mul`, `neg`, and `div` should not modify any existing rationals.

Note: Hint: to simplify rational numbers, you can use function `gcd` (see Listing 3) that computes the greatest common divisor of two positive numbers.

Note: As specified in the problem statement, the sign should be part of the numerator, i.e. if the input number is 4/-9, when the output function is called, the number should be shown as -4/9.

Note: Input validation for `ifstream` is not required, i.e. you can assume that the stream or file will have two valid integers that can form a rational number. (You still have to check for the denominator being 0.)

Note: Observe how `driver1.cpp` reads input from both `std::cin` as well as from an `ifstream` connected with the file `inp1.txt`. Also, note how `driver1.cpp` uses the `output` function to write to `std::cout` as well as to the `ofstream` connected with `log1.txt`.

The `input` function should have this signature: `void input(istream& x);`.

The `output` function should have this signature: `void output(ostream& y, string label);`.

Listing 3: A function to compute the greatest common divisor of two positive integers.

```
static unsigned int gcd(unsigned int n1, unsigned int n2) {
    unsigned int tmp;
    while (n2 != 0) {
      tmp = n1;
      n1 = n2;
      n2 = tmp % n2;
    }
    return n1;
  }
```

Deliverable: `rational.cpp`, `readme.txt`.

End