CSC3002 23Fall Assignment 1

Due 23:59, Oct. 8, 2023

About the First Assignment

- Submission to OJ platform is required, while submission to Blackboard again is not necessary. You only need to upload all the content in the corresponding ***.cpp*** (the *.cpp* file that has a main() function in it) file to OJ.
- You may need to construct a new **Qt console application** for **each problem**[1], and add the corresponding .h (if needed) and .cpp files to the project. **DO NOT add the given file into the sample-project** since StanfordCPPLib is not needed in this assignment.
- If you failed to input in Qt default run window, go to project -> Build&Run -> Run -> check the box "Run in terminal". This operation can enable an extra terminal for normal input and output.
- If you wish to directly input the sample file instead of inputting manually, go to project -> Build&Run -> Run -> view the "Working directory" and copy the file you want to input to this directory -> add the line "< [filename]" into the bar "Command line arguments". Remember to replace [filename] with the filename that you want to input.
- If you need to use the header file in your OJ submission version, the file path is different from that on your PC. You should use `#include "CSC3002OJActive/assignment1/[filename]"` on the OJ version to include the header file.
- Feel free to use the functions in the *lib.cpp* in any problem of this assignment. Do not forget to include the header file, whose path on OJ is `#include "CSC3002OJActive/assignment1/lib.h"` .

Problem 1 (Exercise 2.9 & 2.10, Points: 25)

Problem Description The combinations function $C(n, k)$described in this chapter determines the number of ways you can choose k values from a set of n elements, ignoring the order of the elements. If the order of the value matters - so that, in the case of the coin example, choosing a quarter

first and then a dime is seen as distinct from choosing a dime and then a quarter - you need to use a different function, which computes the number of permutations. This function is denoted as $P(n, k)$, and has the following mathematical formulation:

$$P(n,k) = \frac{n!}{(n-k)!}$$

Although this definition is mathematically correct, it is not well suited to implementation in practice because the factorials involved can get much too large to store in an integer variable, even when the answer is small. For example, if you tried to use this formula to calculate the number of ways to select two cards from a standard 52 -card deck, you would end up trying to evaluate the following fraction:

$$\frac{80,658,175,170,943,878,571,660,636,856,403,766,975,289,505,440,883,277,824,000,000,000,000}{30,414,093,201,713,378,043,612,608,166,064,768,844,377,641,568,960,512,000,000,000,000}$$

even though the answer is the much more manageable $2652 (52 \times 51)$.

The $C(n, k)$ function from the text and the $P(n, k)$ function come up often in computational mathematics, particularly in an area called combinatorics, which is concerned with counting the ways objects can be combined.

[1]You can check the last section in the Tut 1 for detailed procedure.

Requirements Please finish the **TODO** part in the file *combinatorics.cpp*. Write a function `permutations(n, k)` that computes the $P(n, k)$ function without calling the fact function in the file combinatorics.cpp. When you write the implementation, make sure to rewrite the code for the combinations function so that it uses the efficiency enhancements suggested for permutations. Also `combinations(n, k)` that computes the $C(n, k)$ function need to be completed.

DO NOT modify the main() part, which is for the test unit.

In & Out Your program receives two integers in one line, split by a space. Then the program outputs the permutation and the combination of these two integers. You may see the file *in/p1_ in.txt* and *out/p1_ out.txt* for standard output.

Problem 2 (Exercise 3.20, Points: 25)

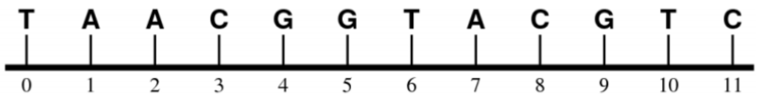
Problem Description *There is no gene for the human spirit. - Tagline for the 1997 film GATTACA*

The genetic code for all living organisms is carried in its DNA-a molecule with the remarkable capacity to replicate its own structure. The DNA molecule itself consists of a long strand of chemical bases wound together with a similar strand in a double helix. DNA's ability to replicate comes from the fact that its four constituent bases-adenosine, cytosine, guanine, and thymine-combine with each other only in the following ways:

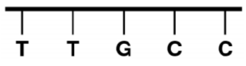
- Cytosine on one strand links only with guanine on the other, and vice versa
- Adenosine links only with thymine, and vice versa.

Biologists abbreviate the names of the bases by writing only the initial letter: **A**, **C**, **G**, or **T**.

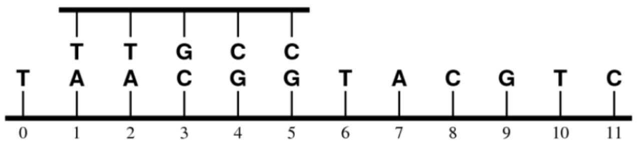
Inside the cell, a DNA strand acts as a template to which other DNA strands can attach themselves. As an example, suppose that you have the following DNA strand, in which the position of each base has been numbered as it would be in a C + + string:



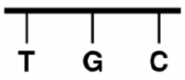
Your mission in this exercise is to determine where a shorter DNA strand can attach itself to the longer one. If, for example, you were trying to find a match for the strand



the rules for DNA dictate that this strand can bind to the longer one only at position 1:



By contrast, the strand



matches at either position 2 or position 7.

Requirements Write a function `int findDNAMatch (string s1, string s2, int start = 0);` that returns the position at which the DNA strand **s1** can attach to the strand **s2**. If there are multiple matches, all possible matching positions should be returned. As in the `find` method for the string class, the optional start parameter indicates the index position at which the search should start. If there is no match, `findDNAMatch` should return -1.

DO NOT modify the main() part, which is for the test unit.

In & Out Your program receives two lines of strings, **s1** and **s2**. split by a space. If there is any character other than ACGT input, it is supposed to be considered as X and X cannot match anyone, even if X itself.

You can also check the file *in/p2_ in*.txt* and *out/p2_ out*.txt* for standard outputs.

Problem 3 (Exercise 4.8, Points: 25)

Problem Description Even though comments are essential for human readers, the compiler simply ignores them. If you are writing a compiler, you therefore need to be able to recognize and eliminate comments that occur in a source file.

Write a function `void removeComments (istream & is, ostream & os)` that copies characters from the input stream is to the output stream os, except for characters that appear inside C + + comments. Your implementation should recognize both comment conventions:

- Any text beginning with /* and ending with */, possibly many lines later.
- Any text beginning with // and extending through the end of the line.

The real C + + compiler needs to check to make sure that these characters are not contained inside quoted strings, but you should feel free to ignore that detail. The problem is tricky enough as it stands.

Requirements Please finish the file *RemoveComments.cpp*. DO NOT modify the main() part, which is for the test unit.

DO NOT modify the main() part, which is for the test unit.

In & Out Your program receives multiple lines of codes. Then the program outputs the codes after removing all the comments. You may see the file *int/p3_ in.txt* and *out/p3_ out.txt* for standard output.

Hints The "/**/" comments requires removing any characters in between, however, keep all the characters outside, even if these is only a line break outside. The "//" comments required removing all the characters till the end of the line, however, keep the line break to avoid affecting the next line. Meanwhile, you may need to test some comment rule own your IDE to learn to handle the mixed/embedded comment symbols.

Problem 4 (Exercise 4.9, Points: 25)

Problem Description *Books were bks and Robin Hood was Rbinhd. Little Goody Two Shoes lost her Os and so did Goldilocks, and the former became a whisper, and the latter sounded like a key jiggled in a lck. It* *was impossible to read "cockadoodledoo" aloud, and parents gave up reading to their children, and ome gave up reading altogether.... - James Thurber, The Wonderful O, 1957*

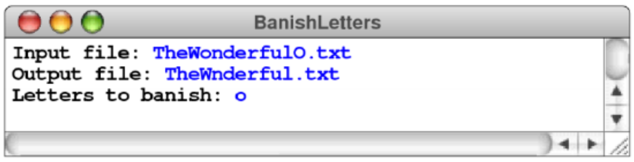
In James Thurber's children's story The Wonderful O, the island of Ooroo is invaded by pirates who set out to banish the letter O from the alphabet. Such censorship would be much easier with modern technology. Write a program that asks the user for an input file, an output file, and a string of letters to be eliminated. The program should then copy the input file to the output file, deleting any of the letters that appear in the string of censored letters, no matter whether they appear in uppercase or lowercase form.

As an example, suppose that you have a file containing the first few lines of Thurber's novel, as follows:

TheWonderfulO.txt

```
Somewhere a ponderous tower clock slowly
dropped a dozen strokes into the gloom.
Storm clouds rode low along the horizon,
and no moon shown.  Only a melancholy
chorus of frogs broke the soundlessness.
```

If you run your program with the input



```
Input file: TheWonderfulO.txt
Output file: TheWnderful.txt
Letters to banish: o
```

it should write the following file:

TheWnderful.txt

```
Smewhere a pnderus twer clck slwly
drpped a dzen strkes int the glm.
Strm cluds rde lw alng the hrizn,
and n mn shwn.  nly a melanchly
chrus f frgs brke the sundlessness.
```

If you try to get greedy and banish all the vowels by entering aeiou in response to the prompt, the contents of the output file would be

```
Smwhr  pndrs twr clck slwly
drppd  dzn strks nt th glm.
Strm clds rd lw lng th hrzn,
nd n mn shwn.  nly  mlnchly
chrs f frgs brk th sndlssnss.
```

Requirements Please finish the file *BanishLetters.cpp*. DO NOT modify the main() part, which is for the test unit

In & Out Statement

Your program receives multiple lines of text input, in which the first line are the characters to be banished. Then the program outputs the banished text. You may see the file *int/p4_in.txt* and *out/p4_out.txt* for standard output.