PAiC++ March 6, 2023

**Assignment 1: Welcome Back to C++**

***Isn't it a happy thing if you review what you have learned on time?***

* **The Analects of Confucius, *Confucius*, 500 BC**

Most of the assignments in this course are single programs of a substantial size. To get you started, however, this assignment is a series of four short problems that are designed to get you used to file I/O operation in C++ and to introduce the idea of functional recursion. None of these problems require more than a page of code to complete; most can be solved in just a few lines.

# Problem 1 Adding Commas (Chapter 3, exercise 14, page 153)

When large numbers are written out on paper, it is traditional—at least in the United States—to use commas to separate the digits into groups of three. For example, the number one million is usually written in the following form:

1,000,000

To make it easier for programmers to display numbers in this fashion, implement a function

**string addCommas(string digits);**

that takes a string of decimal digits representing a number and returns the string formed by inserting commas at every third position, starting on the right. For example, if you were to execute the main program

int main() {

while (true) {

string digits = getLine("Enter a number: ");

if (digits == "") break;

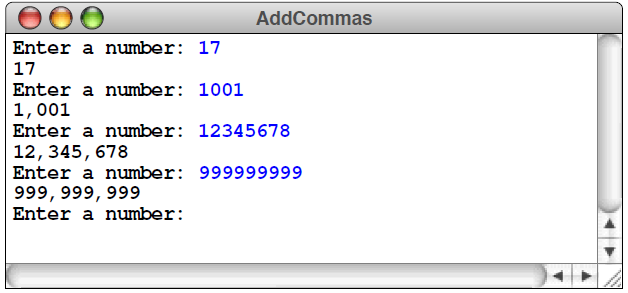
cout << addCommas(digits) << endl;

}

return 0;

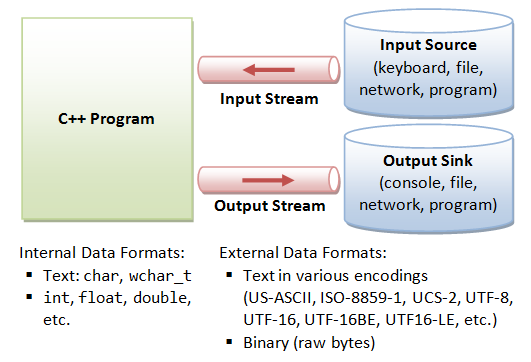
}

your implementation of the **addCommas** function should be able to produce the following sample run:



# Problem 2 Sorting File Data

C/C++ IO are based on streams, which are sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe). In input operations, data bytes flow from an input source (such as keyboard, file, network or another program) into the program. In output operations, data bytes flow from the program to an output sink (such as console, file, network or another program). Streams acts as an intermediaries between the programs and the actual IO devices, in such the way that frees the programmers from handling the actual devices, so as to archive device independent IO operations.



C++ provides both the formatted and unformatted IO functions. In formatted or high-level IO, bytes are grouped and converted to types such as int, double, string or user-defined types. In unformatted or low-level IO, bytes are treated as raw bytes and unconverted. Formatted IO operations are supported via overloading the stream insertion (<<) and stream extraction (>>) operators, which presents a consistent public IO interface.

To perform input and output, a C++ program:

1. Construct a stream object.
2. Connect (Associate) the stream object to an actual IO device (e.g., keyboard, console, file, network, another program).
3. Perform input/output operations on the stream, via the functions defined in the stream's pubic interface in a device independent manner. Some functions convert the data between the external format and internal format (formatted IO); while other does not (unformatted or binary IO).
4. Disconnect (Dissociate) the stream to the actual IO device (e.g., close the file).
5. Free the stream object.

Now there is a **text format file** (named as “***raw-data.txt***”) containing several lines of integers. Your program can solove:

|  |  |
| --- | --- |
| **Task 1**: | Read out the data from raw-data.txt for sorting, and output the result to a **binary format file** named as ***sorted-data.bin***. |
| **Task 2**: | Read out the data from ***sorted-data.bin***, find out the median and output to the screen. |

Note: Starter files contain raw-data.txt, and your program needs to create sorted-data.bin.

# Problem 3 Finding DNA Match (Chapter 3, exercise 20, page 157)

***There is no gene for the human spirit.***

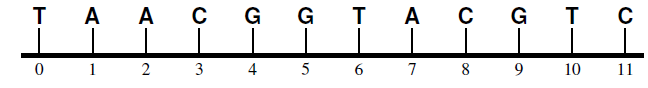
* **Tagline for the 1997 film *GATTACA***

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**.

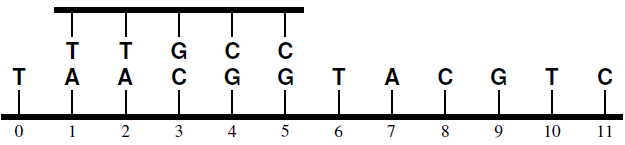
Inside the cell, a DNA strand acts as a template to which other DNA strands can attach themselves. As an example, suppose that you have the following DNA strand, in which the position of each base has been numbered as it would be in a C++ string:



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.

Write a function

**int findDNAMatch(string s1, string s2, int start = 0);**

that returns the first position at which the DNA strand **s1** can attach to the strand **s2**. 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.