LAB 1

REVIEW | FIT@HCMUS | OOP

Objective

Review some programing techniques such as string, array, file handling, etc.

Tutors

- Truong Tan Khoa (truongtankhoa1190@gmail.com)
- Le Ngoc Thanh (lnthanh@fit.hcmus.edu.vn)

Description

EXERCISE 1-1.

Write program to write a string with content: "Hello, World" to the disk file.

EXERCISE 1-2.

Write program to read all strings in input.txt and display on screen. After that, count number of distinct words.

EXERCISE 1-3.

Write a complete program that opens a file for input and a file for output, reads temperature data from the input file and saves some statistics in the output file. The input file will look like this:

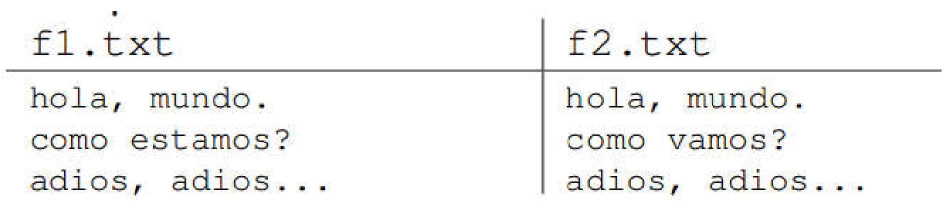
```
7
70.0 69.0 69.0 68.0 67.0 67.5 67.3 68.0 69.2 72.1 74.3 78.3 79.0 80.2 80.5 80.2 79.3 78.3
74.0 73.2 73.0 72.1 71.0 71.0
72.4 69.4 69.4 68.4 67.4 67.2 67.7 68.4 69.2 72.1 74.7 78.7 79.4 82.2 82.2 82.2 79.4 78.7
74.4 77.2 77.4 72.1 73.4 73.4
70.0 69.0 69.0 68.0 67.0 67.2 67.7 68.0 69.2 73.1 74.7 78.7 79.0 80.2 80.2 80.2 79.0 78.7
74.0 76.2 76.0 73.1 71.0 70.0
71.1 69.1 69.1 69.1 67.1 67.2 67.7 69.1 69.2 73.1 74.7 78.7 79.1 83.2 83.2 83.2 79.1 78.7
74.1 77.2 77.1 73.1 72.1 72.1
70.6 69.6 69.6 68.6 65.6 65.2 65.0 68.6 69.2 72.1 74.0 78.0 79.6 79.2 79.2 79.2 79.6 78.0
74.6 78.2 78.6 72.1 71.6 70.6
70.3 68.3 68.3 68.3 67.3 67.5 67.7 68.3 68.5 73.1 74.7 77.7 79.3 85.5 85.5 85.5 79.3 77.7
74.3 77.5 77.3 73.1 72.3 70.3
69.4 70.4 70.4 68.4 67.4 67.2 67.7 68.4 70.2 72.1 74.7 78.7 79.4 80.2 80.2 80.2 79.4 78.7
74.4 77.2 77.4 72.1 71.4 73.4
```

What you see is an integer at the beginning that represents how many rows will follow (7 rows for 7 days of temperature data). Then each of the 7 rows has exactly 24 doubles. The input file represents temperature readings over a 24-hour period for seven days (your program should work for input files with more or fewer days; perhaps the input files has 2 days of data, or 600 days of data; there will always be 24 doubles for each day, however). Your program should read this file and, for each day, find the ***average temperature***, the ***high temperature***, and the ***low temperature***. This information, for each day, should be saved in the output file.

EXERCISE 1-4.

Design a program for reading two text files, "f1.txt" and "f2.txt", writing on the screen the lines that differ in both files, adding "< " if the line corresponds to "f1.txt", and " >" if it corresponds to "f2.txt".

Example:



f1.txt	f2.txt
hola, mundo.	hola, mundo.
como estamos?	como vamos?
adios, adios...	adios, adios...

The output should be:

< como estamos?

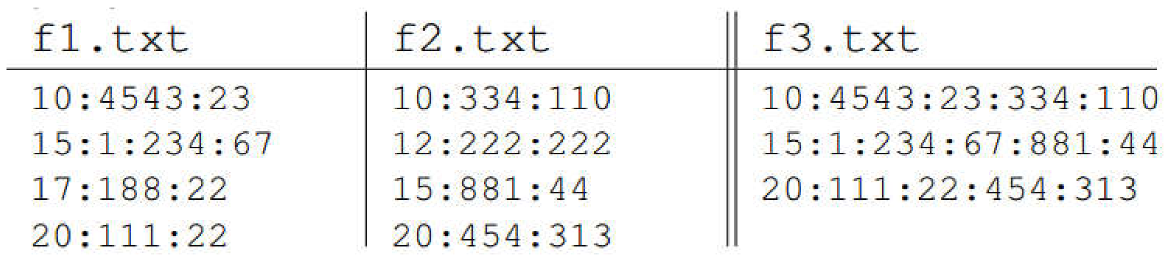
> como vamos?

EXERCISE 1-5.

Develop a function called "finfichero" receiving two arguments: the first one must be a positive integer n, and the second the name of a text file. The function must print on the screen the last n lines of the given file. (Notes that the file maybe has 100000000000000 lines)

EXERCISE 1-6.

Given two text files "f1.txt" and "f2.txt", in which each line is a series of numbers separated by ":", and assuming that the lines are in ascending order by the first number, make a function to read both files line by line writing in the file "f3.txt" the common lines, like in the following example:



f1.txt	f2.txt	f3.txt
10:4543:23	10:334:110	10:4543:23:334:110
15:1:234:67	12:222:222	15:1:234:67:881:44
17:188:22	15:881:44	20:111:22:454:313
20:111:22	20:454:313	

EXERCISE 1-7.

Write a program that stores the personal information of a student (such as full name, age, address) in a structure that you will define and then write the record to *binary file*.

EXERCISE 1-8.

Write a program that extracts student data include full name, age, address from the *binary file* which is saved in Exercise 1-4. And print to console window.

EXERCISE 1-9 (SUMMARIZE).

Do these:

1) declare ex1 as pointer to char
2) declare ex2 as pointer to pointer to char
3) declare ex3 as array 10 of pointer to char
4) declare ex4 as pointer to array 30 of char
5) declare ex5 as array 10 of pointer to array 500 of char
6) declare ex6 as const pointer to int
7) declare ex7 as pointer to const int.
8) write a function that takes a pointer to char named pc, and returns it
9) write a function that takes a pointer to int named pi, and returns it as a pointer to float
10) write a function that takes a pointer to int named pi, and verifies if the value pointed to by pi is odd or even, and displays a message saying which one it is
11) implement a dispatch table of three functions which print "Catfish", "chrisname" and "devonrevenge" and let the user choose which one to run by entering 1, 2 or 3

EXERCISE 1-10 (CHALLENGE - NOT REQUIRED).
Retrieve a long text pointed to by a pointer and count how many times each character occurs in the file. Create a visual representation of the histogram on the command line. The output should look similar to:

```
 : [#################################...................................]
;: [#####################...............................................]
a: [#############################################################.......]
c: [###########.........................................................]
d: [#################################...................................]
e: [###########################################################.........]
f: [###############################################.....................]
g: [#######.............................................................]
h: [############################################........................]
i: [########################################################............]
j: [#################################################...................]
```

Your function should take a single string argument and return a dynamically allocated array of 26 integers representing the count of each of the letters a . . z respectively. Your function should be case insensitive, i.e., count 'A' and 'a' as the occurrence of the letter a.

EXERCISE 1-11 (CHALLENGE - NOT REQUIRED).
Real-world data from measuring devices or sensors is generally produced asynchronously relative to the processing routine(s). The data may appear in bursts that occur faster than can be individually processed. An effective way to handle this situation is to employ a *buffer*. A *buffer* is simply an array into which a data-*generating* process writes data, and from which a separate data-*processing* routine subsequently retrieves it. In the short-term, if the data is being generated at a faster rate than can be processed, the buffer fills while the processing routine catches up. In the long-term (assuming the average processing rate exceeds the average generation rate), the data processing will be able to remain ahead of or keep pace with the generating process.
There are a number of ways to implement buffering. One simple way is to employ a double-buffering technique in which a generating process fills one buffer, then provides the address of the filled buffer to the processing routine and allocates a new buffer to begin filling. The processing routine extracts the data from the filled buffer and disposes the buffer when finished, then awaits the next "filled" buffer.

In this problem, we'll simulate a basic double-buffering scheme using dynamically allocated arrays of integers. You will need to construct a data generation function and a data processing function and then randomly call them from a simulation loop in the main program.

1) Construct a function: double getProb() that will return a pseudo-random value between 0.0 and 1.0.
2) Construct a function: int* generateData(int* &inbuf, int &count) that will simulate asynchronous data generation by obtaining a random number between 0 and 9 (you do not need to use getProb() for this) and saving it in an input buffer. Your function should do the following:
   - Generate a random integer between 0 and 9 and add it to the buffer at the array location specified by the count argument. (Note the buffer is passed as a pointer variable reference.)
   - Increase the count.
   - If the buffer is full: a) return the address of the full buffer, b) reset the count to zero, c) allocate a new buffer and d) save its address in the pointer variable passed as the first argument.
   - If the buffer is not full, then return NULL.
3) Construct a function: void processData(int* &outbuf, int &count, int &total) to simulate "processing" of the asynchronous data. Do the following:
   - If the output buffer pointer is NULL, do nothing, i.e., just return.
   - Otherwise: obtain the buffer value at [count] and add it to total, then increase the count.
   - If all elements in the buffer have been exhausted, then reset the count to zero, delete the (dynamically allocated) buffer and set the buffer pointer argument to NULL.
4) Finally, include the following code in a main simulation program that will call the generateData and ProcessData functions in a loop.

```
const int BUFSIZE=10;
const int ITERATIONS=50;
int main()
{
        int *fillbuffer = new int[BUFSIZE];
        int fillcnt=0;
        int *processbuffer = NULL;
        int processcnt=0;
        int tcount = 0;
        for(int i=0; i<ITERATIONS; i++)
        {
                int *temp;
                if (getProb() <= 0.40 ) {
                        temp = generateData(fillbuffer,fillcnt);
                        if ( temp != NULL )
                                processbuffer = temp;
                }
                if(getProb() <= 0.60)
                        processData(processbuffer,processcnt,tcount);
                cout << fillcnt << '\t' << processcnt << endl;
        }
        cout << "Total value: " << tcount << endl;
        return 0;
}
```