ENSF 619
Lab 4 – Friday Oct 9, 2020

Department of Electrical & Computer Engineering
University of Calgary

M. Moussavi, PhD, PEng

Important Note: This is another individual lab assignment

This is another important lab assignment and it is aimed to help you understanding very essential topics such as dynamic allocation of memory in C++, and concepts such as copying objects in C++.

Due Date: Friday Oct 16, before 5:00 PM

Marking scheme

- Exercise A 16 marks
- Exercise B 4 marks
- Exercise C 4 marks
- Exercise D Not marked

Exercise A: Design A Dynamic Array Class

Read This First:

This exercise is designed to give you some insight about a resizable array class. It also shows other feature of C++ including proper way of copying objects of this class.

What to do:

Download files in the files `MyArray.h` and `lab4ExA.cpp`. Then read the file `MyArray.h`, which tells you what the `MyArray` class interface is and what the member variables are. Pay close attention to the comment that describes the memory management strategy for the `MyArray` class.
Read the file `lab4ExA.cpp`, which demonstrates how a `MyArray` object could be used.
The file `MyArray.cpp` is missing. It's your job to write this file.

What to Submit:

Submit your file `MyArray.cpp` and your program output as part of your lab report

Exercise B: Using C++ library classes, `vector` and `string`

The objective of this exercise is to gain some experience in understanding the C++ library classes, `vector`, and `string`.

What to Do:

Download the files `lab4ExB.cpp` from D2L. In this file there is a declaration of `vector <string>`. If you compile and run this program it creates the following output:

```
ABCD
EFGH
IJKL
MNOP
QRST
```

Let's visualize this output as a matrix of letters (5 rows and 4 columns):

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P
Q	R	S	T

Your job is to complete the definition of the function called `transpose` that creates a new object of `vector<string>` where its strings are the transpose of the original vector:

A	E	I	M	Q
B	F	J	N	R
C	G	K	O	S
D	H	L	P	T

To test your program, you can change the values of the constants `ROWS` and `COLS`, in the `main` function to make sure your function works with other sizes of the `String_Vector`.

What to Submit:

Submit the definition of your function transpose and the program's output.

Exercise C: C++ File I/O

The objective of this exercise is to help you understanding the basics of file I/O in C++.

What to Do:

Download the files `lab4ExC.cpp` from D2L. If you read this file carefully you will realize that this simple C++ program creates a binary file (for example: `cities.bin`), that contain several records of type `struct City`. Each record has two double types represent x and y coordinates of a city on Cartesian Plan, followed by the name of the city, which is stored as a C-string in an array of characters with 30 elements. The program has several functions, the implementation of one of them, called `print_from_binary`, is missing. Your task will be to write the definition of the missing function. This function is supposed to read the content of the binary file created by the program, and display the its content (records both on the screen and into a text file, using in the following format:

```
Name: Calgary, x coordinate: 100, y coordinate: 50
```

Here is the functions prototype and interface comment:

```
void print_from_binary(char* filename);
/* PROMISES: uses ifstream library object to open the binary file named
 * "filename", reads the content of the file which are objects of struct City
 * (one record at a time), and displays them on the screen. It also saves the records
 * into a text-file that its name must be filename argument, but with the extension of .txt
 */
```

What to Submit:

Submit the definition of your function `print_from_binary` and the content or the generated text file, as part of your lab report in PDF format.

Exercise D: Array of Pointers and Command Line Arguments

Read This First

Up to this point in our C and C++ programs, we have always used `main` functions without any arguments. In fact, both languages, allow us to write programs that their `main` functions receives arguments. Here is the heading of such a function:

```
int main(int argc, char **argv){ ... }
```

Of course, we never call a `main` function from anywhere in our program, and we never pass arguments to it. A `main` function can only receive its arguments from command line. In other words, when a user runs the program from command-line, he/she can pass one or more arguments to the `main`. Good examples of this type of programs are many of the Linux and Unix commands such as: `cp`, `mv`, `gcc`, `g++`. For example, the following `cp` command receives the name of two files, to make file `f.dat` a copy of file `f.txt`:

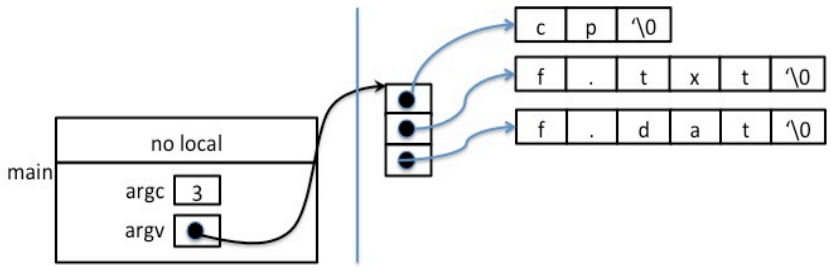
```
cp f.txt f.dat
```

The first token in the above `cp` command is the program's executable file name, followed by two file names. This information will be accessible to the `main`. How does it work? Here is the answer:

To access the command-line arguments, a C/C++ `main` function can have arguments as follows:

```
#include <iostream>
using std::cerr;
int main(int argc, char **argv) {
    if(argc != 3) {
        cerr << "Usage: incorrect number of argument on the command line...\n"";
        exit(1);
    }
    // MORE CODE AS NEEDED.
    return 0;
}
```

Where, `argc` is the number of tokens on the command-line. In our `cp` command example, the value of `argc` should be always `3` (one for program name and two for the file names). If the value of `argc` is not `3`, the program terminates after giving an appropriate message. In other words, this argument is mainly used for error-checking to make sure user has entered the right number of tokens on the command line. The delimiter to count for the number of tokens on the command- line is one or more spaces. The second argument is a pointer-to-pointer, which points to an array of pointers. Each pointer in this array points to one of the string tokens on the command line. The following figure shows how `argv[0]`, `argv[1]`, and `argv[2]` point to the tokens on the command line:



The following code shows how you can simply access and display the command line strings in a C++ program:

```
cout << "The program name is: " << argv[0] << endl;
```

```
cout << "The first string on the command line is: " << argv[1] << endl;
cout << "The second string on the command line is: " << argv[2] << endl;
```

And, here is the output of this code segment for our `cp` command example:

```
The program name is: cp
The first string on the command line is: f.txt
The second string on the command line is: f.dat
```

The exact location of the memory allocated for command-line arguments depends on the underlying OS and the compiler, but for most of the C/C++ systems it is a special area on the stack that is not used for the activation records.

Read This Second

Since this part of exercise B deals with the command line entries by the user, you are **strongly** recommended to implement, compile and run the program on a Cygwin terminal or Mac terminal.

You may be interested in knowing whether it is possible to pass command-line arguments to your program from inside an IDE environment such as Visual C++ or XCode. The answer is yes! It is possible.

Most of the commonly used IDEs provide some way of entering the command line arguments into C/C++ project. For example, in the newer versions of the XCode for the Mac computers, you can click on the 'Edit Scheme', under the 'Product' menu option, and enter the command line argument strings on the 'arguments' tab. A similar way is also available in Visual Studio: Right-click on the project, then select 'Properties' -> Debugging, and enter your arguments into the arguments box. If you choose to use this option, you should seek for more details by consulting the help options available on your target IDE, or search on the Internet.

Our experience shows, using the command line argument from inside some of the IDEs is not always straightforward and figuring out how exactly it works, can be sometimes time-consuming.

What to Do:

Download the file `lab4ExD.cpp` from D2L. This is a simple program that uses the insertion-sort algorithm to sort an array of integer numbers. Now, you should take the following steps:

1. Read the given program carefully to understand what it does.
2. If you are working from ICT lab (which is recommended), compile the program from Cygwin terminal using the following command to create an executable file called `sort`:
   `g++ -Wall lab4ExD.cpp -o sort`
3. Run the program using the following command:
   `./sort`
   It should print the list of several integer numbers, followed by the same list, sorted in ascending order.
4. Now change the value of the local variable `sort_order` from 1 to 2, recompile the program, and run it again. Now it should sort the array in descending order.
5. Your task from this point is to modify the `main` function to have access to the command line arguments. We want to be able to run the program with the options of sorting the numbers either in ascending or descending order, based on the user's input on the command-line.
   Considering that the program's executable name is `sort`, users must be able to run the program from the command-line in one of the following formats:
   `./sort -a`
   Or:
   `./sort -d`

   In the first command the option `-a` (no spaces between dash and a) stands for the ascending, and the second command with the option `-d` stands for the descending order.

Depending on the user's selection of one of these options, the program must sort an array accordingly. To be more precise, the `main` function in this program must check the following conditions:

- If `argc > 2` the program should give the following message and terminate:
  `Usage: Too many arguments on the command line`

- If `argc == 1`, (meaning that the user didn't enter any of the two options), the program should use ascending sort as its default order.

- If `argv[1]`, is NOT one of the following: `-a, -A, -d`, or `-D`, the program should give the following message and terminate:
  `Usage: Invalid entry for the command line option.`

Then, the program should:

- Set the value of `sort_order` to 1, if `argv[1]` points to the C-string: "`-a`" or "`-A`".
- Set the value of `sort_order` to 2, if `argv[1]` points to the C-string: "`-d`" or "`-D`".

Note: as you should recall from material discussed earlier in the course, you cannot compare C-strings by simply using relational operators. Therefore, you need to call C library function `strcmp`. You may refresh your memory regarding this function by reading your course notes, any of your textbooks, or an online sources such as cplusplus.com at: http://www.cplusplus.com/reference/cstring/strcmp.

What to Submit:

As part of your lab report, submit your `lab4ExE.cpp` file and the output of the program that shows sorting the array in ascending and descending orders.