CSC230 Lab 2

Goal: This lab includes function and arguments passing. You will use **cin**, **cout**, **two-dimensional array**, **vector**, and **nested for loops**.

Please try your best to finish the lab in class, and submit it to CANVAS.

PART I: Getting started

In Lab1, we wrote a program that reads in two integers, uses these two integers to creates a two dimensional array. In this lab, you will extend the program written in Lab1. The new program should read in the matrix, and check whether there is word match at a given position.

First, create a lab2 directory. Open a terminal, change directory to be lab2.

Second, copy Lab1.cpp to lab2 directory.

Rename the file inside lab2 directory to be Lab2.cpp.

In the Lab1 program, we created the following statement to create the two-dimensional array

```
char  arr[x][y];
```

It is convenient to create a multiple-dimensional array like the above. However, this type of multi-dimensional array has limitations. One problem is that it is a bit difficult to pass a multi-dimensional array as function parameter. In C++, a two-dimensional (also other multi-dimensional) array is stored in row-major order, which means the data are stored row-by-row. Given the array name and the coordinate values, C++ compiler uses the following equation to calculate the address of element arr[i][j]

address of element (i, j) = base address of array
+ i*(number of elements in a row)*(size of an element)
+ j*(size of an element)

To calculate the address of arr[i][j], the compiler must know the width of the two dimensional array. If we have a following code:

```
char arr[x][y];
bool search(char a[][]){
    //...
}
// ...
```

```
search(arr);
```

C++ compiler will report error because the code of function search() does not have the width value. If we modify the above the above code to be:

```
char arr[x][y];
bool search(char a[][y]){
    //...
}
// ...
search(arr);
```

This code works under the situation where y value is **known** before the search() function definition. If y is a variable and its value is unknown, this code does not work because C++ compiler still does not know the **real width** of the array. There are different ways to solve this problem, for example:

- Pass the width value as a parameter to the function. The code inside the function does not use compiler to implicitly calculate the element address, instead the function itself uses the address equation explicitly calculates the address of the element. The performance of this method can be good, but it is a burden for the programmer to write the code.
- Uses vector defined in C++. Vector internally maintains the size information. User does not need to explicitly pass the size information to the function. In today's lab, we will practice vector.

Mathematically, vector is one-dimensional array. In C++, you can imagine that a vector with size m is an **m x 1** array. If each element of this vector is a vector with size n, then the vector is an **m x n** array. Once the vector is defined, you can access the elements just like the way you access array elements. The following sample code shows how to define and access vector.

```
// ...
#include <vector>
// ...
vector< char > test;
test.resize(10);
for(int i = 0; i < test.size(); i++)
  test[i] = 'a' ;
// ...
```

If you need vector to implement two-dimensional array, you can do it like the following code

```
// ...
#include <vector>
// ...

vector< vector<char> > test;
test.resize(10);
for(int i = 0; i < test.size(); i++)
  test[i].resize(y);


test[3][5] = 'c';
// ...
```

In case, you need to know the size of height, you can use test.size() to find out. Similarly, test[0].size() returns the width of the matrix.

In this part, please use vector to replace the two-dimensional array you defined in Lab1 program.

PART II: Read values from command line

Just like JAVA, C++ has internal variables to record the values typed at command line. C++ uses two parameters of main() function to record the values. Operating system passes these values to the main function when the program is launched. The first parameter of main() records the numbers of arguments of the command line. This value is at least 1 because program name is counted into as well. The second parameter is a string array, which contains all the arguments at the command line. The following code shows how to print out each argument of the command line.

```
#include<iostream>
using namespace std;

int main(int argc, char *argv[]){

  for(int i = 0; i < argc; i++)
    cout << argv[i] << endl;
}
```

If we type the command line

./a.out a b c

The result will be

./a.out
a
b
c

In this part, please add argc and argv to the main() function of your lab2 program.

PART III: Read values from command line, check it at the given position of the matrix

In this part, please define a function inside your Lab2 program, the function header is:

```
bool check(vector< vector<char> > &matrix, char *word, int x, int y)
```

where the return type is boolean (true or false), *matrix* is the vector passed from caller, *word* is a string, *x* and *y* are index values.

As you can see *matrix* is a two-dimensional vector. This function checks whether there is a string matches *word* at position (*x, y*) of *matrix*. If there is a match, return true; otherwise, return false.

For example, if you use 0505matrix in the lab, we have the following contents in 0505matrix.
5 5
u r a q o
f t c n j
k r h p r
e a v o t
z h g a h

If we type command
./a.out tcnj 1 1 < 0505matrix

It should print out true.

If the command is
./a.out tcnj 2 1 < 0505matrix

The output is false.

If the command is
./a.out cse 1 1 < 0505matrix

The output is false.

Hint:

1. You can use strlen(word) to get the length of the string.

2. When you compare word with the letters of the matrix, make sure you will not read out of the bound of the matrix.

Wrap up

When you're done, zip three files to lab2.zip

zip lab2.zip Lab2.cpp a.out 0505matrix

Submit lab2.zip to Canvas.

Make sure you logout before you leave!

If you cannot finish lab in class, please save all your files. Next time you login the computer in the lab, you can continue work on your files. Please save them before you logout. If you work in a Linux lab, please save the file to your machine. However, if you are working in the Mac lab, please save the file to a CLOUD. The Mac machine will erase everything you saved once you logout.