Lab2 – CSI2372A
Thursday September 26th /Tuesday October 1st
SITE - University of Ottawa Fall 2024
Due ONLINE Tuesday, October 8th at midnight
In groups of no more than two students
/10

- ## Objectives

i) Pointers, Pointer/reference comparison in a function call, passing an array to a function, return of a pointer by a function.
ii) The C <string.h> library

I. Pointers

II.a. Declaring pointers

Example:
int a;
int *pa=&a;

II.b. Passing pointers to a function

For functions that handle arguments with pass-by-value, any changes to variables local to the function do not affect variables in the calling function. For example:

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

void exchange(int i, int j);

int main() {
	int i = 1;
	int j = 2;
	cout << i << endl << j << endl;   // displays 1 then 2
	exchange(i, j);                  // displays 2 then 1
	cout << i << endl << j << endl; // displays 1 then 2
}

void exchange(int i, int j) {
	int tempo = i;
	i = j;
	j = tempo;
	cout << endl << i << endl << j << endl;
	return;
}
```

It is possible in C ++ to perform a passage by reference which allows direct work on the variables of the calling function. We use the & character for this. The previous "*exchange*" function can be written:

```
void exchange(int& i, int& j){
	int tempo = i;
	i = j;
	j = tempo;
	cout << endl << i << endl << j << endl;
	return;
}
```

The *main ()* function then displays: **1 then 2, 2 then 1 and then 2 then 1**.
This type of parameter passing is more efficient than passing by value. Programmers often use it even though the values of the calling function should not be changed. In this case, in order to avoid an accidental modification of the variable in the called function, the reserved word "***const***" is used:

```
void display(const int& i) {
	i = 12;            // compilation error
	cout << i;
}
```

Using a notation with pointers, this

```
void exchange(int* pi, int* pj) {
	int tempo = *pi;
	*pi = *pj;
	*pj = tempo;
	cout << endl << *pi << endl << *pj << endl;
	return;
}
```

Calling this function will be done by passing addresses as arguments :

```
int main() {
	int a = 2;
	int b = 3;
	cout << endl << a << endl << b << endl;  // displays 2 then 3
	exchange(&a, &b);                       // displays 3 then 2
	cout << endl << a << endl << b << endl; // displays 3 then 2
}
```

II.c. Passing an array to a function
Example:

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

int main() {
	int n;
	float moy;
	float val[10];
	//.....
	float average(int n, const float values[]); /* Declaration of a function
	dealing with an integer and an array dynamically allocated */
	//.....
		moy = average(n, val);    /*average function call*/
	//.....
}

float average(int a, const float x[]) /* average function definition */
{
	//....
}
```

Using the reserved word ***const*** helps prevent unwanted changes to array elements.

II.d. Return of a pointer (*) by a function

Consider the example of a function which compares 2 strings and which must return the string ranked first alphabetically. In this case, we write it with pointers:

```
#include <iostream>
#include <iomanip>
#include <string.h>             //for strcmp

using namespace std;

const char* comp(const char* ch1, const char* ch2);      //the function return
the address of the largest string

int main(){
	const char* chain1 = "Hello";
	const char* chain2 = "Hi";
	const char* premier;

	premier = comp(chain1, chain2);
	cout << premier;
}

const char* comp(const char* ch1, const char* ch2){
	if (strcmp(ch1, ch2) < 0) return(ch1);
	else return(ch2);
}


/*OUTPUT*/
Hello
```

A function can also return a reference (**&**). We reserve this possibility for object programming. We will then have the same advantages as previously for the management of objects (instances of classes). An additional advantage is being able to write the function as an **L-Value** (left part) in an expression:

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

int val[20];

int& tab(int i){
	return val[i];
}

int main(){
	tab(0) = 3;
	tab(1) = 4;
}
```

III. Arrays and pointers

Since arrays and pointers are very "close", we can declare an array in the form of a pointers expression. But, in this case, no memory space is reserved for the elements and we have to reserve it with *new*(), dynamic memory allocation.

```
int* tab;
tab = new int[10];                  instead of        int tab[10];
```

The **new()** instruction allows to reserve memory space for 10 integers and returns a pointer of type ***void*** which is implicitly converted to integer type ((**int ***)). After this instruction, we can then access the elements using the ***tab*** pointer.
If the array is no longer needed in the rest of the program, it is possible to free the reserved memory using the **delete** instruction:

```
delete tab;
```

For a **multi-dimensionnal array** :
An array with several dimensions can be expressed in terms of pointers to a group of arrays. For an array of dimension 2 of 10 by 20, we can declare this array in the form:

```
int* x[10];                instead of            int x[10][20];
```

IV. C library <string.h>

The **C language** provides a library of character string manipulations: <**string.h**>. This library has been kept in the **C ++** language but it is advantageously replaced by other C ++ libraries such as the <string> library or even the **MFC** (Microsoft Foundation Class) "**CString**" library.
We have there the functions of the C library. Some of them are in the table below. For more information on using these functions, see the compiler's online help:

Function	Role
memcpy	Copying characters between two memory areas
memcmp	Compare two memory areas
memset	Initializes a memory area
strcpy	Copying a string
strcmp	Compare 2 strings
strcat	Concatenates 2 strings

V. Lab Assignment 2

/10

Exercise 1 : (2 Marks)

Modify the following program, just by inserting the display operator, in such a way that it displays the following values::

```
4
1
1
1
1
1
```

```
/*code*/
#include <iostream>
using namespace std;

int main(void) {
	int tab[10];
	int* p;

	for (int i = 0;i < 10;i++) {
		tab[i] = i * i;
	}
	tab[2] = tab[1];
	tab[2] = *(tab + 1);
	*(tab + 2) = tab[1];
	*(tab + 2) = *(tab + 1);
	p = &tab[0];
	p = tab + 1;
	tab[4] = *p;
}
```

Exercise 2 (3 Marks)

The attached *main* program performs the "insertion" sort of an array of 10 integers. The insertion sort method is the one used by a card player when sorting the cards he has received. He takes the first card and compares it to all the following ones. If he encounters a smaller card, he places it before the reference card (all the cards between the first and the smaller one are then shifted). He repeats this operation until the last card.
The program uses a sort function which performs the sorting by processing 2 parameters: the number of elements and the name of the array.

a) Write the *sort* function using the array formalism. Put the definition of this function in a separate file myFile2.cpp.
b) Complete the attached file myFile2.h including the declaration of the *sort* function.

/*Example of output*/

```
Displaying the unsorted array :
2 4 88 20 3 55 87 134 2 5

Displaying the sorted array :
2 2 3 4 5 20 55 87 88 134
```

Exercise 3 (2 Marks)

Complete the following function,

```
int** triangleInf(int n);
```

in the attached file MyFile3.cpp at the indicated location. MyFile3.cpp contains the main program main (DO NOT MODIFY).

The `triangleInf` function returns a "lower triangular matrix" with n rows (passed as parameter) representing a Pascal triangle. The first row of the matrix will have one element, the second two, the third three …

The element (n, p) of Pascal's triangle can be calculated using the following properties:

Element	Value
(n, 0)	1
(n, p)	0 if p > n
(n, p)	Value of (n-1, p-1) + Value of (n-1, p) if $0 < p \leq n$

Example :

	p = 0	**p = 1**	**p = 2**	**p = 3**
n = 0	1			
n = 1	1	1		
n = 2	1	2	1	
n = 3	1	3	3	1

You should get the following output:

/*OUTPUT*/

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

Exercise 4 (3 Marks)
We want to handle 5 strings declared in the form of an array of 5 pointers to characters. The program will start by entering these 5 strings (maximum string size: 20 characters). The 5 strings must be entered ending with a tab (see ***cin.getline*** in Lab 1). The program then displays the 5 entered strings then the following menu:

1) Display of character strings.
2) Replacement of one string by another
3) Sorting strings
4) Exit the program.

Each action offered by the menu is performed by a separate function. The menu display is also performed by a function which will return the choice entered by the user. If the choice is not known, the menu is displayed again.

Write the following functions:

a) The function *display* which displays the strings stored in an array passed as the first parameter and its size as the 2nd parameter :

```
void display(char* tab[], int const& nbre);
```

b) The function *sort* which sorts the strings using insertion sort. You can use for example *strcpy* and *strcmp* (explained above):

```
void sort(char* tab[], int const& nbre);
```

c) The function *replace* which replaces a string in the table (tab of size nbre) with another string entered on the keyboard. The maximum size of the string is passed in the 3rd parameter (size):

```
void trier(char* tab[], int const& nbre);
```

For this question, 2 files are necessary :
- MyFile4.h which contains the necessary header files.
- MyFile2.cpp which contains the *main* program and the definition of the functions.

/*Example of output*/

```
Enter the 5 character strings ending with a tab and a line return
Hello
my dear
how are you
Fine
Bye

The string 0 is : Hello
The string 1 is : my dear
The string 2 is : how are you
The string 3 is : Fine
The string 4 is : Bye


                Menu

1) Strings display.
2) Replacement of a string by an other one
3) Sorting strings
4) Exit of the program
```

```
Your choice :2

Enter the string number to modify: 1
Enter the new string: Sir


                Menu

1) Strings display.
2) Replacement of a string by an other one
3) Sorting strings
4) Exit of the program

Your choice :1

The string 0 is : Hello
The string 1 is : Sir
The string 2 is : how are you
The string 3 is : Fine
The string 4 is : Bye


                Menu

1) Strings display.
2) Replacement of a string by an other one
3) Sorting strings
4) Exit of the program

Your choice :3


                Menu

1) Strings display.
2) Replacement of a string by an other one
3) Sorting strings
4) Exit of the program

Your choice :1

The string 0 is : Bye
The string 1 is : Fine
The string 2 is : Hello
The string 3 is : Sir
The string 4 is : how are you


                Menu

1) Strings display.
2) Replacement of a string by an other one
3) Sorting strings
4) Exit of the program

Your choice :4
```

Submit your work ONLINE (one zip file only) before Tuesday October 8th at midnight

Instructions

- Create a directory that you will name Assignment2_ID, where you will replace ID with your student number (the one submitting the assignment).
Put all the following files in your compressed directory Assignment1_ID.zip for submission in the Brightspace Virtual Campus.

Files :

- ✓ *README.txt*
- ✓ *myFile1.cpp*
- ✓ *myFile2.cpp*
- ✓ *myFile2.h*
- ✓ *myFile3.cpp*
- ✓ *myfile4.h*
- ✓ *hyfile4.cpp*

- Don't forget to add comments in each program to explain the purpose of the program, the functionality of each method and the type of its parameters as well as the result.
- In the Assignment1_ID directory, create a text file named README.txt, which should contain **the names of the two students**, as well as a brief description of the content:

Student Name:
Student Number:
Course Code: CSI2372A