Lab2/Devoir2 – CSI2772A
Mardi/Vendredi 26/29 Septembre 2023
SITE - Université d'Ottawa
Dû EN LIGNE le vendredi 13 Octobre à 22:00
En groupes d'au plus deux étudiant(e)s.
/10

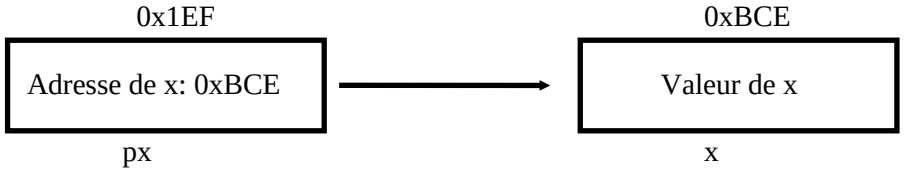
I. Introduction

- Ce lab porte sur :
    i) Rappels sur les pointeurs, Correspondances entre tableau et pointeur, Comparaison pointeur/référence dans un appel de fonction, Passage d'un tableau dans une fonction, Renvoi d'un pointeur par une fonction.
    ii) La bibliothèque C **<string.h>**

II. Rappels

II.1. Rappels sur les pointeurs

Un pointeur est une variable qui représente l'adresse d'une donnée ou d'un ensemble de données. Les pointeurs permettent, par exemple, d'échanger des informations entre une fonction et un programme et de retourner à celui-ci **plusieurs** valeurs (ce que ne permet pas une simple instruction **return**). Il est possible de les utiliser pour traiter des tableaux, des chaînes de caractères et les listes chaînées.
Soit **x** une variable représentant une donnée. On peut accéder à cette donnée si on connaît l'adresse de son emplacement mémoire. Cette adresse s'obtient grâce à l'expression **&x**. Si on affecte cette adresse à une seconde variable **px** (px=&x;), cette dernière "pointe" sur la variable **x**. Autrement dit:



Et on peut écrire:
px=&x
x=*px

On obtient deux manières d'accéder au contenu de **x:**
cout << x; et **cout << *px;**

II.2.a) Déclaration de pointeurs

On utilise le caractère ***** pour déclarer un pointeur. Le type spécifié doit correspondre au type de la donnée pointée. Il est possible d'initialiser un pointeur lors de sa déclaration. Pour cela, il faut que la variable sur laquelle il pointe soit déclarée auparavant. Une autre initialisation possible est la valeur **NULL** (qui correspond à 0). Par exemple:

```
int a;
int* pa = &a;
```

II.2.b. Passage de pointeurs dans une fonction

Pour les fonctions traitant des arguments avec un passage par valeur, toute modification des variables locales à la fonction n'affecte en rien les variables de la fonction appelante. Considérons l'exemple suivant:

```
#include<iostream>

using namespace std;

void echange(int i, int j);

int main(void){
	int i = 1;
	int j = 2;

	cout << i << endl << j << endl;   // affiche 1 puis 2
	echange(i, j);                    // affiche 2 puis 1
	cout << i << endl << j << endl;   // affiche 1 puis 2
}

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

La fonction **main()** affiche alors: **1 puis 2**, **2 puis 1** et ensuite 1 **puis 2**.

Il est possible en **C++** d'effectuer un passage par référence qui permet un travail direct sur les variables de la fonction appelante. On utilise pour cela le caractère **&**. La fonction "**echange**" précédente peut s'écrire:

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

La fonction **main()** affiche alors: **1 puis 2**, **2 puis 1** et ensuite **2 puis 1**.

Ce type de passage des paramètres est plus efficace qu'un passage par valeur.

En utilisant une notation avec des pointeurs, cette même fonction peut s'écrire:

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

L'appel de cette fonction se fera en passant des adresses comme argument:

```
int main(void) {
	int a = 2;
	int b = 3;
	echange(&a, &b);
}
```

II.2.c. Passage d'un tableau dans une fonction

Le passage d'un tableau dans une fonction se fait en **spécifiant le nom seul de celui-ci**. Les crochets ne doivent surtout pas être mis lors de l'appel. La définition de la fonction doit bien spécifier que son argument traite un tableau**. En général**, **ce dernier sera déclaré en tableau dynamique**:

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

int main(void){
	int n;
	float moy;
	float valeurs[10];
	//.....
	float moyenne(int n, const float valeurs[]); /* Déclaration d'une fonction
	traitant un entier et un tableau alloué dynamiquement */
	//.....
	moy = moyenne(n, valeurs); /*Appel à la fonction moyenne en passant le nom du
	                                    tableau  seul*/
	//.....
}

float moyenne(int a, const float x[]);          /*Définition de la fonction moyenne */
{
	//....
}
```

Lorsque l'appel à la fonction est fait, les passage des éléments se fait par référence et non par valeur. Le passage par référence présenté repose sur le fait que le nom d'un tableau est un pointeur sur le premier élément de celui-ci. L'emploi du mot réservé **const** permet d'éviter des modifications non souhaitées des éléments du tableau.

II.2.d. Renvoi d'un pointeur (*) par une fonction

La possibilité de faire retourner un pointeur par une fonction permet de retourner un ensemble de données (un tableau, une structure, une union… par exemple) sachant que l'instruction **return** ne permet de retourner qu'une seule valeur

Considérons l'exemple d'une fonction qui compare 2 chaînes de caractères et qui doit retourner la chaîne classée première alphabétiquement. Dans ce cas, nous l'écrivons avec des pointeurs:

```
#include <iostream>
#include <iomanip>
#include <string.h> //nécessaire pour la fonction strcmp

using namespace std;

const char* comp(const char* ch1, const char* ch2);   //La fonction traite 2 pointeurs
                                                      // et retourne l'adresse de la chaîne la plus grande
int main(void) {
       const char* chaine1 = "Bonjour";
       const char* chaine2 = "Salut";
       const char* premier;

       premier = comp(chaine1, chaine2);
       cout << premier;
}

const  char* comp(const char* ch1, const char* ch2)
{
       if (strcmp(ch1, ch2) < 0) return(ch1);   // voir l'aide en ligne du compilateur
                                                // pour la fonction strcmp
       else return(ch2);
}
```

/*OUTPUT*/
Bonjour

Une fonction peut aussi retourner une référence (**&**). Nous réservons cette possibilité pour la programmation objet. Nous aurons alors les mêmes avantages que précédemment pour la gestion d'objets (instances de classes). Un avantage supplémentaire est de pouvoir écrire la fonction comme une **L-Value** (partie gauche) dans une expression:

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

int val[20];

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

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

II.3. Correspondance entre les tableaux et les pointeurs

II.3.a. Définition d'un tableau à l'aide de pointeurs

Les tableaux et les pointeurs étant très "proches", on peut déclarer un tableau sous la forme d'une expression de pointeurs. L'avantage concerne les opérations que l'on peut effectuer plus simplement avec la seconde forme de déclaration.
Mais dans ce cas, aucune place mémoire n'est réservée pour les éléments et on est obligé de réserver celle-ci avec **new(),** allocation dynamique de mémoire.

```
int* tab;
tab = new int[10];                au lieu de                int tab[10];
```

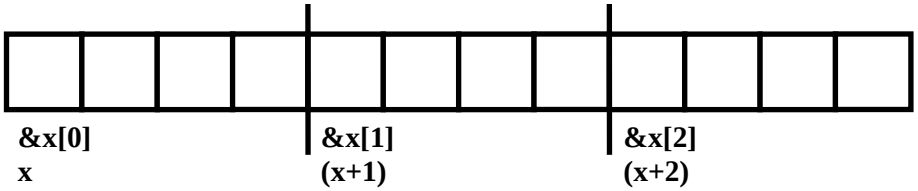
L'instruction **new()** permet de réserver de la place mémoire pour 10 entiers et retourne un pointeur de type **void** qui est converti implicitement en type entier (**(int *)**). Après cette instruction, on peut alors accéder aux éléments grâce au pointeur **tab**.
Si le tableau n'est plus nécessaire dans la suite du programme, il est possible de libérer la mémoire réservée grâce à l'instruction **delete**:

```
delete tab;
```

II.3.b. Tableaux à 1 dimension

Le nom d'un tableau est un pointeur sur son premier élément. L'adresse de ce premier élément peut donc s'écrire:
x ou **&x[0]**

De la même façon, l'adresse de l'élément suivant peut s'obtenir par les expressions: **&x[1]** ou **(x+l)**. Cette dernière expression n'est pas une opération arithmétique mais un calcul d'adresse. Ainsi avec un tableau **x** de 3 entiers et sachant qu'un entier représente 4 octets, l'image mémoire sera:



On accédera au contenu de chaque élément à travers des pointeurs de la façon suivante:

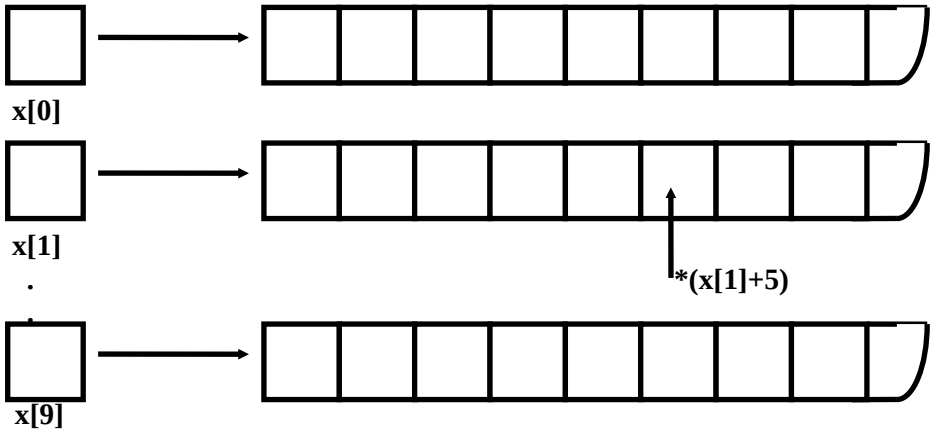
***(x)**	au lieu de	**x[0]**
***(x+l)**	au lieu de	**x[1]**

II.3.c. Tableaux à plusieurs dimensions

Un tableaux à plusieurs dimensions peut s'exprimer en termes de pointeurs vers un groupe de tableaux. Pour un tableau de dimension 2 de 10 par 20, on peut déclarer ce tableau sous la forme:

```
int* x[10];                au lieu de          int x[10][20];
```

On obtient alors un tableau de 10 pointeurs qui pointeront chacun sur un tableau à une dimension comme le montre l'image mémoire suivante (1 carré équivaut à 4 octets):



On obtient alors 10 pointeurs sur 10 tableaux de **n** éléments. On remarque que le nombre d'éléments de chacun des tableaux à une dimension n'est pas précisé. Là encore, il faudra prévoir une allocation dynamique de mémoire pour chacun des tableaux à une dimension. L'accès à l'élément x[1][5] s'écrira sous forme de pointeur de la manière suivante:

***(x[1]+5)** au lieu de **x[1][5]**

III. La bibliothèque C <string.h>

Le **langage C** fournit une bibliothèque de manipulations de chaînes de caractères: **<string.h>**. Cette librairie a été conservée dans le langage C++ mais elle est avantageusement remplacée par d'autres bibliothèques **C++** comme la bibliothèque **<string>** ou encore la bibliothèque **MFC** (**M**icrosoft **F**oundation **C**lass) "**CString**".

On y rencontre les fonctions de la librairie C. Certaines d'entre elles sont dans le tableau ci-dessous. Pour plus d'information sur l'utilisation de ces fonctions, reportez vous à l'aide en ligne du compilateur:

Fonction	Rôle
memcpy	Copie de caractères entre deux zones mémoires
memcmp	Compare deux zones mémoires
memset	Initialise une zone mémoire
strcpy	Copie une chaîne de caractères
strcmp	Compare 2 chaîne de caractères
strcat	Concatène 2 chaînes de caractères

IV. Devoir 2

Exercice 1 : (2 POINTS)

Modifier le programme ci-contre, en y introduisant juste l'opérateur d'affichage. de telle manière qu'il affiche les valeurs suivantes :

4
1
1
1
1
1

```
/*Programme*/

#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;
}
```

Soumettre un fichier séparée *MonFichier1.cpp* qui contient votre programme.

Exercice 2 (3 POINTS)

Le programme principal ci-joint effectue le tri "par insertion" d'un tableau de 10 entiers. La méthode du tri par insertion est celle qu'utilise un joueur de carte lorsqu'il trie les cartes qu'il a reçu. Il prend la première carte et la compare à toutes les suivantes. S'il rencontre une carte plus petite, il la place avant la carte de référence (toutes les cartes entre la première et celle plus petite sont alors décalées). Il répète cette opération jusqu'à la dernière carte.
Le programme utilise une fonction *trier* qui réalise le tri en traitant 2 paramètres: le nom du tableau et le nombre de ses éléments.

a) Écrire la fonction *trier* en utilisant le formalisme tableau.

```
void trier(int tab[], int size);
```

Compléter la définition de cette fonction dans le fichier ci-joint *MonFichier2.cpp* à la place indiquée. *MonFichier2.cpp* contient le programme principal `main` (à NE PAS MODIFIER).

b) Compléter le fichier ci-joint *MonFichier2.h* en incluant la déclaration de la fonction *trier*.

/*Exemple de Sortie*/

Affichage du tableau non trie :
2
4
8
20
3
55
87
13
2
5

Affichage du tableau trie :
2
2
3
4
5
8
13
20
55
87

Exercice 3 : (2 POINTS)

Compléter la fonction suivante,

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

dans le fichier *MonFichier3.cpp* ci-joint à la place indiquée. *MonFichier3.cpp* contient le programme principal `main` (à NE PAS MODIFIER).

La fonction `triangleInf` retourne une "matrice triangulaire inférieure" à *n* lignes (passé en paramètre) représentant un triangle de Pascal. La première ligne de la matrice aura un seul élément, la deuxième deux, la troisième trois …
L'élément (*n*, *p*) du triangle de Pascal peut être calculée en utilisant les propriétés suivantes:

Élément	Valeur
(n, 0)	1
(n, p)	0 si p > n
(n, p)	Valeur de (n-1, p-1) + valeur de (n-1, p) si 0 < p ≤ n

Exemple :

	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

Vous devez obtenir la sortie suivante :

/*Sortie*/
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

Exercice 4 : (3 POINTS)

On veut manipuler 5 chaînes de caractères déclarées sous la forme d'un tableau de 5 pointeurs vers caractères.
Le programme commencera par la saisie de ces 5 chaînes (taille maximale des chaînes : 20 caractères). Les 5 chaînes doivent être saisies en se terminant par une tabulation (voir **cin.getline** dans le Lab 1).
Le programme affiche ensuite les 5 chaînes saisies puis le menu suivant:

1) Affichage des chaînes de caractères.
2) Remplacement d'une chaine par une autre
3) Tri des chaines
4) Sortie du programme.

Chaque action proposée par le menu est réalisée par une fonction distincte *menu* fournie ci-joint. L'affichage du menu est aussi réalisé par une fonction qui retournera le choix saisi par l'utilisateur. Si le choix n'est pas connu, le menu est affiché à nouveau.

Écrire les trois fonctions suivantes dans le fichier `MonFichier4.cpp` ci-joint :

a) La fonction *display* qui affiche les chaines de caractères stockées dans un tableau passé en premier paramètre et sa taille en 2ème paramètre :

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

b) La fonction *trier* qui trie les chaines passées en paramètres (tab) en utilisant un tri par insertion. Vous pouvez utiliser par exemple *strcpy* et *strcmp (expliqu*ées ci haut dans la partie Rappels).

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

c) La fonction *replace* qui remplace une chaine du tableau (*tab* de taille *nbre*) par une autre chaine saisie au clavier. La taille maximale de la chaine est passée en 3ème paramètre (size) :

```
void replace(char* tab[], int const& nbre, int const& size)
```

Pour cette question, 2 fichiers sont nécessaires:
- *MonFichier4.h* fourni ci-joint et qui contient les fichiers d'entête nécessaires et les déclarations des fonctions.
- *MonFichier4.cpp* à compléter aux places indiquées et qui contient le programme principal `main` (à NE PAS MODIFIER) et la définition des fonctions.

/*Exemple de Sortie*/

Saisissez les 5 chaines de caracteres en les terminant par une tabulation et un retour chariot :
Hello
my dear
how are you
Fine
Bye

La chaine 0 est : Hello
La chaine 1 est : my dear
La chaine 2 est : how are you
La chaine 3 est : Fine
La chaine 4 est : Bye

Menu
1) Affichage des chaines de caracteres.
2) Remplacement d'une chaine par une autre
3) Tri des chaines
4) Sortie du programme.

Votre choix :3
Menu

1) Affichage des chaines de caracteres.
2) Remplacement d'une chaine par une autre
3) Tri des chaines
4) Sortie du programme.

Votre choix :1
La chaine 0 est : Bye
La chaine 1 est : Fine
La chaine 2 est : Hello
La chaine 3 est : how are you
La chaine 4 est : my dear

Menu
1) Affichage des chaines de caracteres.
2) Remplacement d'une chaine par une autre
3) Tri des chaines
4) Sortie du programme.

Votre choix :2
Donnez le numero de la chaine a modifier: 2
Saisissez la nouvelle chaine: Bonjour

Menu
1) Affichage des chaines de caracteres.
2) Remplacement d'une chaine par une autre
3) Tri des chaines
4) Sortie du programme.

Votre choix :1
La chaine 0 est : Bye
La chaine 1 est : Fine
La chaine 2 est : Bonjour
La chaine 3 est : how are you
La chaine 4 est : my dear

Menu
1) Affichage des chaines de caracteres.
2) Remplacement d'une chaine par une autre
3) Tri des chaines
4) Sortie du programme.

Votre choix :4

V. Créer et soumettre un seul fichier zip

Directives

- Créez un répertoire que vous nommerez *Devoir2_ID*, où vous remplacerez ID par votre numéro d'étudiant (celui qui soumet le devoir).
  Mettez tous les fichiers suivants dans votre répertoire compressé *Devoir2_ID.zip* pour soumission dans le campus virtuel Brightspace.

  Fichiers :
  - ✓ *README.txt*
  - ✓ *monfichier1.cpp*
  - ✓ *monfichier2.cpp*
  - ✓ *monfichier2.h*
  - ✓ *monfichier3.cpp*
  - ✓ *monfichier4.cpp*
  - ✓ *monfichier4.h*

- N'oubliez pas d'ajouter des commentaires dans chaque programme pour expliquer le but du programme, la fonctionnalité de chaque fonction et le type de ses paramètres ainsi que le résultat.
- Dans le répertoire *Devoir2_ID*, créez un fichier texte nommé *README.txt*, qui devra contenir **les noms des deux étudiant(e)s**, ainsi qu'une brève description du contenu :

Nom étudiant :
Numéro d'étudiant :
Code du cours : CSI2772A

Fraude scolaire :

Cette partie du devoir a pour but de sensibiliser les étudiants face au problème de fraude scolaire (plagiat). Consulter les liens suivants et bien lire les deux documents:
https://www.uottawa.ca/administration-et-gouvernance/reglement-scolaire-14-autres-informations-importantes

https://www.uottawa.ca/administration-et-gouvernance/sites/www.uottawa.ca.administration-et-gouvernance/files/processus_de_traitement_des_cas_de_fraude_academique_-_nov_2019.pdf

Les règlements de l'université seront appliqués pour tout cas de plagiat.
En soumettant ce devoir :
1. vous témoignez avoir lu les documents ci-haut ;
2. vous comprenez les conséquences de la fraude scolaire.