Lab3/Devoir3 – CSI2772A
Mardi/Vendredi 3/6 Octobre 2023
SITE - Université d'Ottawa
Dû EN LIGNE le vendredi 20 Octobre à 22:00
En groupes d'au plus deux étudiant(e)s.
/10

I. Introduction

Ce lab porte sur les structures de données chaînées.

II. Rappels

II.1. Structures

Les structures permettent d'utiliser un ensemble d'éléments de types différents identifiables par un seul nom.

La forme générale d'une définition de structure est la suivante:

```
struct Type {
	membre1;
	membre2;
	membre n;
};
```

Exemple :

```
struct Compte {
	int numero;
	char etat;
	char nom[80];
	float solde;
};
```

A) Déclaration de variables de type structure

Des variables peuvent être déclarées comme type de la structure ainsi définie. L'instruction suivante déclare 2 variables respectant la structure Compte:

```
Compte clientl, client2;
```

Il est possible de condenser la définition de la structure et la déclaration des variables:

```
struct Compte{
	int numero;
	char etat;
	char nom[80];
	float solde;
} client1, client2;
```

Il est aussi possible de déclarer une structure comme membre d'une autre structure à condition que la structure membre soit déclarée auparavant:

```
struct Date {
              int jour;
              int mois;
              int annee;
};

struct Compte {
              int numero;
              .....
              Date date_solde;
};

Compte client[100]; /*Déclaration d'un tableau client de 100 variables de type Compte*/
```

L'initialisation d'une variable de type structure au moment de sa déclaration se fait sous forme de liste:

```
Date date1 = { 12,9,1986 };
```

B. Accès aux éléments d'une structure

On accède à un membre de la structure en spécifiant le nom de la variable suivi par le nom du membre, le tout séparé par un point: **"variable.membre".**

Exemples:

```
struct Compte {
       int numero;
       char nom[80];
} client1, clients[10];


client1.numero;      //accès à l'élément "numero" de la variable "client1"
client1.nom[2];      //accès au 3ème caractère de l'élément tableau "nom" dans la
                     //variable "client1"
++client1.numero;    //incrémente la valeur de l'élément "numero" dans la variable
                     //"client1"
clients[4].nom[1];   //accès au 2ème élément du tableau "nom" dans le 5ème élément du
                     //tableau  de variable "clients"
```

II.2.b. Les unions

La définition d'une union est semblable à celle d'une structure. La réservation mémoire est faite en fonction du membre le plus grand.
Soit la définition d'une union de nom **"Reference":**

```
union Reference {
       char couleur[8];
       int taille;
};
```

A. Déclaration de variables de type union

Comme pour les structures, déclarer une union revient à déclarer un nouveau type de données. Déclarer des variables **chemise** et **blouse** répondant au type **Reference** revient à écrire:

```
Reference chemise, blouse;
```

Il est aussi possible de condenser la définition de l'union avec la déclaration des variables:

```
union Reference {
	char couleur[8];
	int taille;
} chemise, blouse;
```

B. Accès aux éléments d'une union

L'affectation des membres de l'union se fera sur l'un **OU** l'autre:

```
	chemise.couleur = "bleu";
	blouse.taille = 40;
```

La lecture du contenu de "**chemise.taille**" et de "**blouse.couleur**" ne retournera aucune valeur exploitable.

II.2. Passage d'une structure ou une union à une fonction

Suivant les compilateurs, il existe différentes manières de passer une structure complète en paramètre à une fonction. Nous ne verrons que la méthode par pointeur:

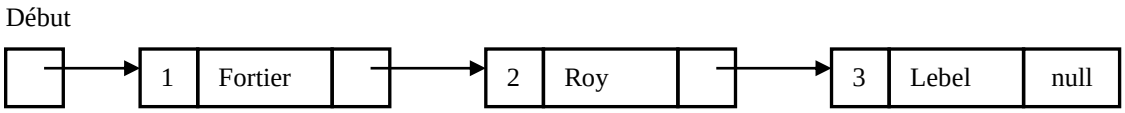
```
#include <iostream>
#include <string.h>
using namespace std;

struct Enregistrement		//definition d'un type Enregistrement
{
	char nom[10];
	int numero;
};

void maj(Enregistrement* pe);		//définition de la fonction maj

int main(void) {
	Enregistrement client = { "Fortier",666 };//déclaration d'une variable client de type Enregistrement

	cout << client.nom << ends << client.numero << endl;

	maj(&client);		//appel de la fonction maj en passant l'adresse de client

	cout << client.nom << ends << client.numero << endl;
}

void maj(Enregistrement* pe) {
	strcpy_s(pe->nom, "Delisle");
	pe->numero = 999;
}

/*OUTPUT*/
Fortier666
Delisle999
```

II.3. Structures auto référentielles

Les structures auto référentielles sont très utiles dans les applications mettant en œuvre des structures de données chaînées telles que les listes. Nous ne traiterons dans ce Lab que les listes chaînées simples. L'idée de base est que chaque élément de la liste contient un pointeur sur l'élément suivant. L'intérêt consiste en une gestion simplifiée de la liste; le retrait, l'ajout d'un élément se fait en jouant sur les pointeurs. La définition d'une structure auto référentielle en liste simple à l'aide d'une structure se fait de la manière suivante:

```
struct liste {
	int numero;
	char nom[80];
	liste* suivant;
};
```

Le membre "**suivant**" est un pointeur sur une structure de même type. En créant plusieurs variables répondant à ce genre de structure, on pourra organiser la liste chaînée suivante:



III. Devoir 3

Exercice 1: (4 POINTS : 2 pts chaque)

a) Complétez le code C++ (*main*()) ci-dessous pour construire un jeu de 32 cartes. Le jeu est représenté par un tableau de 32 éléments distincts, chacun de type `Card`. Vous devez remettre un fichier ***monFichier1a.cpp***.

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

enum Color { club, diamond, spades, heart };
enum Face { seven, eight, nine, ten, jack, queen, king, ace };

struct Card{
	Color color;
	Face face;
};

int main(){
	Card game[32];
	//VOTRE CODE VIENT ICI
}
```

b) Écrivez une fonction appelée *testPair*() pour déterminer si un joueur de poker a une paire :

```
bool testPair(const Hand& h);
```

La fonction reçoit une main (un tableau de cinq cartes différentes) en entrée et affiche vrai si et seulement si les cinq cartes ont au moins une paire (une paire est formée de deux cartes de même face (ex : deux valets)). Vous devez remettre le fichier ***monFichier1b.cpp*** ci-joint complété à la place indiquée (seulement).

/*SORTIE*/

I have at least: 1pair

Exercice 2: (6 POINTS)

Cet exercice consiste à réaliser un programme qui permet d'initialiser une liste simplement chaînée (structures de données chaînées) avec les données suivantes:

student
grade;

Après la saisie du premier élément de la liste, le programme propose le menu suivant (programme ***myLinkedList.cpp*** ci-joint qui fournit le programme principal `main` (À NE PAS MODIFIER) et le fichier en tête correspondant ***myLinkedList.h***):

1) Affichage de la liste chainée complète.
2) Ajout d'un nouvel élément dans la liste.
3) Suppression d'un élément de la liste.
4) Calcul de la moyenne de l'ensemble des élèves.
5) Sortie du programme.

Questions :

Écrire les fonctions suivantes dans le fichier ci-joint ***myLinkedList.cpp***:

1) Fonction *add*: (1.5 pts)
Cette fonction ajoute un nouvel élément à la liste. Le nouvel élément est inséré dans la liste dans une position définie par l'utilisateur, en saisissant au clavier le numéro de l'élément qui le précède. La validité de ce numéro doit être vérifié par le programme.
On créé l'élément inséré dans la liste avec réservation de mémoire pour cet élément. Ses données membres (*student* et *grade*) sont saisis au clavier.
La fonction *add* prend comme arguments, un pointeur sur une structure de type *Evaluation*, et le nombre d'éléments existants dans la chaine.
Elle retourne un pointeur sur une structure de type *Evaluation* qui correspond au premier élément de la liste chainée.

2) Fonction *remove*: (1.5 pts)
Cette fonction supprime un élément de la liste. Le numéro de l'élément à supprimer est saisi au clavier et doit être valide.
La fonction *remove* prend comme arguments, un pointeur sur une structure de type *Evaluation* et le nombre d'éléments existants.
Elle retourne un pointeur sur une structure de type *Evaluation* qui correspond au premier élément de la liste chainée.

3) Fonction *display*: (1.5 pts)
Cette fonction affiche la liste chainée complète, *student* et *grade* de chaque élément (Un élément par ligne).
Elle prend comme arguments, un pointeur sur une structure de type *Evaluation.*

4) Fonction *average* : (1.5 pts)
Elle calcule la moyenne des notes de l'ensemble des élèves et l'affiche.
Elle prend pour arguments, un pointeur sur une structure de type *Evaluation* et le nombre d'éléments existants.
Elle retourne 1 si tout se passe bien, 0 sinon.

```
/*Exemple de sortie*/
```

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:2
After which element you want to insert ? (0 for start): 0

Entering the item from the chained list.

Enter the student: Jane Benoit
Enter the grade: 80

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:2
After which element you want to insert ? (0 for start): 1

Entering the item from the chained list.

Enter the student: Jack Fortier
Enter the grade: 75

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:1

Student :Jane Benoit
The grade is :80

Student :Jack Fortier
The grade is :70

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:4

The average of the class is: 75

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:3
what is the number of the element to delete ?: 2

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:1

Student :Jane Benoit
The grade is :80

1) Display of the complete linked list.
2) Insert an element
3) Remove an element.
4) Calculation of the class average.
5) Exit the program.

Your choice ?:5

IV. Créer et soumettre un seul fichier zip

Directives

- Créez un répertoire que vous nommerez *Devoir3_ID*, où vous remplacerez ID par votre numéro d'étudiant (celui qui soumet le devoir).
  Mettez tous les fichiers suivants dans votre répertoire compressé *Devoir3_ID.zip* pour soumission dans le campus virtuel Brightspace.

  Fichiers :
  - ✓ *README.txt*
  - ✓ ***monFichier1a**.cpp*
  - ✓ ***monFichier1b**.cpp*
  - ✓ ***myLinkedList**.h*
  - ✓ ***myLinkedList**.cpp*

- N'oubliez pas d'ajouter des commentaires dans chaque programme pour expliquer le but du programme, la fonctionnalité de chaque méthode et le type de ses paramètres ainsi que le résultat.
- Dans le répertoire *Devoir3_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.