LABORATOIRE 0 – Initiation aux laboratoires
Mardi/Vendredi 12/15 Septembre 2023
CSI2772A – Automne 2023
SITE - Université d'Ottawa
Aucune soumission n'est exigée

- **Objectifs**

- Environnement de développement : Microsoft Visual Studio 2022
- DevC (frontend à gcc) est également installé
- Un premier programme en C++
- Mots réservés, Opérateurs, Types de donnée de base, Les spécificateurs de type, Les constantes, Conversion explicite, Entrées/sorties (cin, cout , cerr et clog), Structures de contrôle ;

I. Introduction

- Veuillez démarrer Microsoft Visual Studio.
- Créez un nouveau projet se nommant **Lab0**. Pour cela, sélectionnez le choix "**new**" dans le menu "**file**". Puis cliquez sur l'onglet "**Projects**".
- Choisissez de développer un programme de type "**Win32 Console Application**" et saisissez le nom de votre projet avant de valider l'écran.
- L'environnement de développement vous crée alors les fichiers nécessaires au projet **Lab0**.
- Les fichiers C++ sont en *.cpp (on peut utiliser ausi *.cxx, *.cc et .C); *.hh sont les fichiers en tête (*.hpp, *.hxx, et *.h sont utilisés).

I.1. Notion de projet

L'écriture de programme peut se faire dans un seul fichier source. Cependant, il est fort conseillé de développer en utilisant plusieurs fichiers afin d'améliorer la lisibilité et la maintenance du programme.

I.2. Création des fichiers sources

Créez un premier fichier nommé "**monfichier1.h**". Pour cela, sélectionnez encore le choix "**new**" dans le menu "**file**". Puis cliquez sur l'onglet "**Files**". Choisissez un type de fichier "**C/C++ Header File**" et saisissez le nom de votre fichier. Veillez à ce que ce fichier soit ajouté au projet Lab0 avant de valider l'écran.

L'environnement de développement lance l'éditeur sur le fichier "**monfichier1.h**". Vous pouvez alors saisir le contenu de ce fichier:

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

Sauvegardez et fermez ce fichier.

Créez maintenant de la même manière un fichier de type "**C++ Source File**" nommé "**monfichier1.cpp**" et qui s'ajoute, bien entendu, au projet Lab0. L'édition de ce fichier est alors lancée. Placez le contenu suivant dans ce fichier:

```
#include "monfichier1.h"
int main(void) {
	cout << "Bonjour" << endl;
	cout << "Bienvenue en CSI2772A !" << endl;
}
```

Sauvegardez le fichier. Vous venez de créer votre premier projet sous **Microsoft Visual C++**.

I.3. Compilation et exécution du programme :

Une fois le projet créé, vous pouvez visualiser la manière dont vos fichiers ont été organisés par l'environnement. Pour cela, sélectionnez l'onglet "**File**" dans la fenêtre "**Workspace**". Notez au passage qu'un autre onglet permet d'accéder à l'aide.

Compilez votre projet en sélectionnant l'option "**Compile …**" dans le menu "**Build**". Exécutez votre programme en choisissant l'option "**Execute …**" dans le menu "**Build**".

II. Rappels

II.1. Caractères, mots réservés et opérateurs

II.1.a. Caractères

Pour écrire en langage C++, on peut utiliser les 26 lettres de l'alphabet en majuscule et minuscule (C++ fait la différence). A ces caractères, il faut ajouter les caractères spéciaux suivants:

! * + \ " < # (= | { > %) ~ ;] / ^ - [: / ? & _] ' . <espace>

Certaines combinaisons de caractères ont aussi une signification particulière. Entres autres, on peut trouver **/*** et // pour les commentaires, ou encore **\n** pour un retour à la ligne.

II.1.b. Les mots réservés

On entend par mots réservés les mots clefs qui possèdent une signification particulière en C++. Ils ne pourront pas être utilisés pour nommer une variable. En voici la liste:

asm, auto, bad_cast, bad_typeid, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, except, extern, finally, float, for, friend, goto, if, inline, int, long, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, try, type_info, typedef, typeid, union, unsigned, using, virtual, void, volatile, while, xalloc

II.1.c. Les opérateurs dans le langage C++

Opérateurs	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	reste de la division entière (modulo)

Tableau 1: Les opérateurs arithmétiques de base.

Opérateurs	Signification
-	signe négatif
--	Décrémentation
++	Incrémentation
!	Négation
sizeof	Retourne la taille en octets

Tableau 2: Les opérateurs unaires.

Opérateurs	Signification
<	inférieur à
<=	inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	égal à
!=	différent de
&&	ET logique
||	OU logique
?	Si
:	Sinon

Tableau 3: Les opérateurs relationnels, logiques et conditionnels.

Opérateurs	Signification
=	Affectation
+=	Affectation avec addition
-=	Affectation avec soustraction
*=	Affectation avec multiplication
/=	Affectation avec division
%=	Affectation avec modulo

Tableau 4: Les opérateurs d'affectation.

Note : les opérateurs permettant d'effectuer des opérations sur les objets, ou sur les pointeurs ne sont pas présents. Ces derniers seront vus au cours des prochains laboratoires.

II.2. Les Types de donnée de base, Les spécificateurs de type, Les constantes

II.2.a. Les types de données de bases

Le langage C++ possède 4 types de données simples: les entiers (type **int**), les caractères (type **char**), les réels (type **float**) et les doubles réels (type **double**). A chaque type correspond un domaine de validité. Celui-ci dépend du compilateur et de l'architecture de la machine utilisé.

Avec le compilateur **Visual C++** et sur une architecture à base de **Pentium**, les résultats obtenus sont:

1 octets pour les caractères
4 octets pour les entiers
4 octets pour les réels
8 octets pour les doubles

Les caractères prennent des valeurs allant de **0** à **255** pour représenter les valeurs d'un caractère affichable ou non. Pour connaître la correspondance, on se référera à la table des caractères **ASCII**. Les entiers (signés) peuvent varier de **-2.147.483.647** à **+2.147.483.648.** Les types **float** et **double** représentent des nombres réels en **base 10**, le second possédant plus de chiffres significatifs que le premier.

II.2.b. Les spécificateurs de type

A partir de ces types de bases on peut définir des types étendus grâce aux spécificateurs de type suivants: **long**, **short**, **signed** et **unsigned**.

Pour les entiers, on trouvera:

long int	équivalent à **int**
short int	(2 octets) qui permet des valeurs entières de **-32.768** à **32.767**
signed int	équivalent à **int**
unsigned int	qui permet des valeurs entières de **0** à **4.294.967.295**
short signed int	équivalent à **short int**
short unsigned int	(2 octets) qui permet des valeurs entières de **0** à **65535**.

Pour les caractères, on trouvera:

signed char	équivalent à **char**
unsigned char	qui permet des valeurs de **-128** à **+127**

Pour les réels, on trouvera:

float	(4 octets)
double	(8 octets)
long double	(12 octets)

A) La déclaration des variables

A partir de tous les types basiques et étendus, on peut maintenant déclarer des variables. Pour cela, on donne le type souhaité suivi du ou des noms de variables. On trouvera par exemple:

int a;	Déclaration d'une variable **a** entière
unsigned char adc;	Déclaration d'une variable **adc** de type caractère non signé
char tab[30], b;	Déclaration d'un tableau **tab** de 30 variables caractères et d'une variable **b** du même type
float adf;	Déclaration d'une variable ad**c** de type réel

Une déclaration de variable peut se faire à n'importe quel endroit du code.

B) L'affectation d'une variable.

L'affectation d'une variable se fait avec le symbole "=" suivi de la valeur ou de l'expression puis se termine par un point virgule. Une valeur peut être formatée différemment suivant qu'elle soit entière, octal, caractère, chaîne ou hexadécimale. Quelques exemples:

a = 255;	affecte la valeur entière **255** à **a**
a = 0x000E;	affecte une valeur hexadécimal (**14** en base 10) à **a**.
a = 010;	affecte une valeur octale (**8** en base 10) à **a**.
a = 'a';	affecte le code ASCII du caractère **a** (**97** en base 10) à **a**
tab[2] = 3;	affecte la valeur entière **3** au **troisième** élément du tableau **tab**
adf = 1.123E+3;	affecte la valeur **1123** à **adf**
adf = 0.968E-12;	affecte la valeur **0.000000000000968** à **adf**

II.2.c. Les constantes

Une constante est une variable qui ne peut être modifiée au cours d'un. La création de cette constante se fait à travers le mot réservé **const**. Son initialisation est obligatoire lors de sa déclaration. Si vous souhaitez, par exemple, déclarer et définir une constante réelle **Pi** contenant la valeur **3.14159**, vous placerez dans votre programme :

```
const  float Pi = 3.14159;
```

II.3. Conversion explicite (casting)

Il est possible d'affecter à une variable le résultat de la conversion d'une autre variable (pour des types simples et dans la mesure du possible). Pour cela, le type de la variable affecté sera suivi de la variable à convertir encadrée par des parenthèses:

```
char c = 'x';
int i;
i = int(c);
```

ou encore

```
double d = 1.99;
int e;
d = double(e);
```

II.4. Entrées/sorties

II.4.a. Les opérateurs d'entrée/sorties

Pour permettre le dialogue avec l'utilisateur dans un terminal, le langage **C++** fournit en standard les flots:

- **cin** : qui représente l'entrée standard (par défaut, le clavier)
- **cout** : qui représente la sortie standard (par défaut, le terminal)
- **cerr** : qui représente la sortie d'erreur standard avec mémoire tampon (par défaut, le terminal)
- **clog** : qui représente la sortie d'erreur standard sans mémoire tampon (par défaut, le terminal)

L'emploi de ces flots dans un programme nécessite le chargement du fichier d'entête "**iostream.h**".

Notez que ces flots d'entrée/sorties sont définis pour des types simples de données comme des entiers, des caractères, des réels ou encore des chaînes de caractères.

les entrée/sorties sont présentés avec les flots en utilisant les notations propres à la programmation objet (opérateur de portée **::**, méthodes…).

L'envoi de données vers l'écran et la saisie de données depuis le clavier se fait en utilisant respectivement les opérateurs << et >>. L'opérateur << est associé avec un des flots de sortie **cout**, **cerr** ou **clog** pour permettre l'envoi de données vers le terminal. L'opérateur >> est associé avec le flot d'entrée **cin** pour permettre la saisie de donnée. L'exemple suivant montre un emploi basique de ces règles d'écriture:

Exemple 1:

```
#include <iostream>

using namespace std;

void main(void) {
	int a;                              //Déclaration d'un entier
	char c;                             //Déclaration d'un caractère
	cout << "Entrez un entier: ";       //Affiche la chaîne de caractères
	cin >> a;                           //Permet la saisie d'un entier stocké dans a
	cout << "Entrez un caractere: ";
	cin >> c;                           //Permet la saisie d'un caractère stocké dans c
}
```

/*Exemple de Sortie*/
Entrez un entier: 10
Entrez un caractere: Hello

II.4.b. Manipulateurs d'entrée/sorties

Un certain nombre de manipulateurs sont fournis par le langage (tableau 5) afin d'améliorer les opérations sur les entrée/sorties. On peut retrouver ceux-ci en éditant le fichier "**ostream.h**". D'autres manipulateurs qui traitent un argument sont disponibles en chargeant le fichier d'entête "**iomanip.h**".

Manipulateurs	Fonction
flush	Vide et efface la mémoire tampon du flot concerné. Force ainsi la lecture ou l'écriture des données contenues dans le flot.
endl	Insère dans le flot un caractère de retour à la ligne et vide et efface la mémoire tampon du flot concerné.
ends	Insère dans le flot un caractère de fin de chaîne de caractères ("\0" ou NULL) et vide et efface la mémoire tampon.
dec	Toutes les lectures et écritures suivantes sont faites en effectuant une conversion décimale des données.
hex	Toutes les lectures et écritures suivantes sont faites en effectuant une conversion hexadécimale des données.
oct	Toutes les lectures et écritures suivantes sont faites en effectuant une conversion octale des données.
ws	Permet de filtrer les caractères "blancs" (espace, tabulation, etc…) sur un flot en entrée uniquement.
setw(int i)	Permet de définir la taille du champs dans lequel la valeur sera insérée pour la prochaine entrée/sortie. La taille est donnée par l'entier **i**.
setbase(int i)	Définit la base qui sera utilisée pour l'affichage des prochaines données. La variable **i** vaut **10** (décimal), **8** (octal) ou **16** (hexadécimal).
setfill(char c)	Donne grâce à la variable **c** le caractère de remplissage qui sera utilisé pour la prochaine entrée/sortie.
setprecision(int i)	Définit la précision sur **i** chiffres pour la prochaine entrée/sortie.
setiosflags(flag)	Permet de modifier les caractéristiques du flot (présentation, conversion…)
resetiosflags(flag)	Permet de réinitialiser les caractéristiques du flot.

Tableau 5: Les manipulateurs d'entrée/sorties

Exemple 2:

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

int main(void) {
	double racine = 123466788.34255;
	int i = 1234889;

	cout << setw(20) << setprecision(3) << setfill('*') << setiosflags(ios::left |
	ios::fixed) << racine << endl;
	cout << hex << i << " * " << dec << i << " * " << oct << i << endl;
}
```

/*SORTIE*/

*123466788.343*******

*12d7c9 * 1234889 * 4553711*

II.4.c. Quelques méthodes (fonctions) associées aux entrée/sorties

Les modifications du comportement d'un flot d'entrée/sortie (en général, on joue essentiellement sur les flots de sortie) sont possible en utilisant des méthodes propres aux flots. Pour accéder à une fonction disponible dans le flot, il faut utiliser le nom du flot puis placer le nom de cette fonction le tout séparé par un point.

Par **exemple**:

```
clog.width(2);                  //fixe la taille du champ pour l'affichage
cout << clog.width();           //affiche la taille du champ qui est fixé
clog.fill('e');                 //définit le caractère de remplissage
cout << clog.fill();            //affiche le caractère de remplissage
clog.precision(4);              // fixe la précision
cout << clog.precision();       //Affiche la précision
```

Les flots d'entrée possèdent d'autres méthodes. Une d'entres elles est particulièrement intéressante pour la saisie d'un chaîne de caractère: la fonction **getline()**. Rappelons qu'une chaîne de caractères peut être déclarées comme un tableau de caractères (vision du Langage C…).

La fonction **getline()** comprend 3 arguments. Le premier fixe la variable chaîne de caractères à initialiser avec la saisie. Le second donne la taille maximale de cette chaîne de caractères. Le troisième argument fixe le caractère qui provoquera la fin de la saisie. Vous pourrez écrire, par exemple:

Exemple 3:

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

int main() {
	char line[100];
	cout << " Donnez une ligne terminee par 't'" << endl;
	cin.getline(line, 100, 't');
	cout << line;
}
```

/*SORTIE*/
Donnez une ligne terminee par 't'
Bonjour tout le monde !
Bonjour

II.5. Structure de contrôle

II.5.a. *if*

Syntaxe générale: if (expression) { instructions; } else { instructions; }
Exemple:

```
if (i < 0) {
		cout << "i est negatif\n" << flush;
	} else {
		cout << "i est positif\n" << flush;
	}
```

Autre manière d'écrire ces instructions:

```
(i < 0) ? cout << "i est negatif\n" : cout << "i est positif\n";
	cout << flush;
```

II.5.b. *while*

Syntaxe générale: while (expression) { instructions; }

Exemple:

```
int i = 100;
	while (i > 0) {
		cout << i << endl;
		i--;
	}
```

III.5.c. *do while*

Syntaxe générale: *do { instructions; } while (expression);*
Exemple:

```
int i = 100;
	do {
		--i;
		cout << i << endl;
	} while (i > 0)
```

III.5.c. *for*

Syntaxe générale: *for (expression1;expression2;expression3) { instructions; }*
Exemple:

```
for (int i = 0;i < 100;i++) {
			cout << i << endl;
		}
```

III.5.d. switch

Syntaxe générale:

```
switch (expression) {
case expression1 : instructions; break;  //On peut se passer de "break"
case expression2 : instructions; break;
.
.
default : instructions;
}
```

Exemple:

```
switch (int i) {
	case '1':
		cout << "i est 1";
		break;  //facultatif
	case '2':
		cout << "i est 2";
		break; //facultatif
	default:
		cout << "Il y a un probleme";
	}
```

IV. Fonction *main*()

La fonction **main()** a la possibilité de traiter des arguments et de retourner une valeur. Cette dernière sera en général générée par l'instruction **return()** à la fin de la fonction **main()** ou par **exit()** à tout autre endroit. L'instruction **exit()** requiert le fichier d'entête "**process.h**". Elle retourne une valeur entière et interrompt le programme. Par défaut, en **C++**, la valeur est **0**.

Exemple 5:

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

int main(int nbre, char* arg[]) {
	if (nbre < 2) {
		cout << endl << "Vous devez donner au moins un argument !!!!" << endl;
		exit(2);
	}
	for (int i = 0;i < nbre;i++) {
		cout << endl << "L'argument  numero " << i << " est " << arg[i];
	}
	return(0);
}
```

Pour exécuter ce programme, vous devez vous placer dans un fenêtre **DOS** et exécutez le programme dans une ligne de commande.