Langage C++

- CIR2 -

TP1 : Du C au C++

Nils Beaussé (nils.beausse@yncrea.fr)

Afin d'éviter les mauvaises habitudes, dans les premiers TPs on se passera de CLION ou autre éditeur intégré (Eclipse, Xcode, Visual Studio et autre), vous n'aurez besoin que d'un éditeur de texte (vim ou le wordpad sous windows, ou notepad++ si vous voulez tout de même profiter de la coloration syntaxique) et d'un terminal de commande WSL.

Attention, toute personne ne respectant pas ces consignes pourra obtenir des malus sur ses notes. L'idée étant de jouer le jeu.

On rappelle la commande de base pour compiler :

```
g++ -Wall main.c -o mon_executable
```

Et pour executer :

```
./mon_executable
```

Créez un dossier TP1 sur le bureau de Windows (dans le terminal sous Linux on y accède en allant à **/mnt/c/Users/VOTRE_NOM/Bureau**). Pour chaque exercice vous créerez un fichier .cpp .

Exercice 0 : Se remettre dans le bain

Dans un premier temps, essayons de se réhabituer au C standard :
Créer un programme C dans exercice0.c, il devra afficher « Coucou, quel est ton nom ? » et demander à l'utilisateur de saisir son nom. (Qui ne devra pas dépasser une certaine taille de 10 caractères).

Une fois ceci fait, et si ce n'est pas déjà fait, mettre le cœur du programme (les printf et les scanf) dans une fonction, déclarée SOUS le code, et dont l'en-tête sera déclaré dans un header .h.

Compilez et testez votre programme !

Dans un second temps on fera en sorte que le nom soit de taille variable et que de fait, le tableau de char soit alloué dynamiquement (interdiction d'utiliser des types tout fait comme string).
Pour ce faire, vous pouvez demander à l'utilisateur la taille de son nom dans un premier temps puis utiliser cette taille pour l'allocation mémoire.

Rappel : En C on alloue la mémoire avec « malloc » et on la libère avec « free ».
Voici leur usage :

```
truc = malloc ( taille ) ;
free(truc) ;
```

Notez qu'on peut utiliser sizeof (un_type_quelconque) pour obtenir la taille d'un type quelconque. Ainsi sizeof(int) donne la taille d'un entier en C. Idem : 3*sizeof(int) donne la taille d'un tableau de 3 int, fort logiquement.

Ainsi il est habituel d'écrire :

```
int* truc;
truc = malloc (10*sizeof(int)) ;
```

pour allouer un tableau de 10 entiers.

Comme maloc renvoie toujours un void* (voir le cours : tous les pointeurs sont identiques d'un point de vue « type », donc void* vaut bien n'importe quel autre pointeurs) il est également habituel, pour être propre, de convertir explicitement ce void* dans le pointeur auquel on l'attribue. Ça ne sert à rien en pratique, mais ça permet de faire plus propre.
La formulation la plus courante, et celle, donc, qu'il faut employer, est celle-là :

```
int* truc;
truc = (int*) malloc (10*sizeof(int)) ;
```

Exercice 1 : Premiers pas en C++

Créez et testez un programme C++ (donc avec des .cpp et des .hpp etc.) qui demande à l'utilisateur de saisir deux valeurs (val1 et val2), qui calcule ensuite leur somme et affiche le résultat comme suit :

"val1 + val2 = resultat"

Si le résultat est positif le message suivant est aussi affiché *"Positif ? true" sinon l'affichage sera "Positif ? false ou nul"*.
On essaiera d'utiliser le modificateur « ***boolalpha*** » !

Note pour la suite :

En C on utilise ***malloc*** et ***free*** pour **l'allocation dynamique**. En C++ on peut utiliser **new** et **delete**, qui sont les variantes modernes de ces opérateurs et qui ont quelques différences

qu'on détaillera plus tard dans le cours. **new** s'utilise ainsi pour créer un tableau de (par exemple) 5 entiers :

```
int* foo;
foo = new int[5];
```

delete est appelé différemment qu'il s'agisse d'un tableau alloué ou non (contrairement au **free** de **malloc**). On utilisera **delete[]** pour les tableaux et **delete** pour un seul élément. Pour le tableau précédent cela donne :

```
Delete[] foo;
```

Il y a une raison à cela, qu'on détaillera par la suite dans le cours.

Exercice 2

Créer et tester une fonction C++ qui :

a. demande à l'utilisateur d'entrer une taille (n) d'un tableau (utiliser le mot-clé auto pour déclarer "n")
b. alloue un espace mémoire pour ce tableau si la taille est positive
c. demande à l'utilisateur d'entrer "n" valeurs et les affiche
d. calcule la moyenne des "n" valeurs et l'affiche
e. libère la mémoire allouée par le tableau en fin de programme

Exercice 3

Ecrire et tester 3 fonctions qui ont le même nom à savoir `max` et qui calculent le maximum des variables données comme arguments :

1. Prototype de la première fonction `max(int n)` dans ce cas c'est la valeur de "n" qui va être retournée
2. Prototype de la deuxième fonction `max(int m, int n)`

3. Prototype de la troisième fonction `max(int m, int n, int r)`
4. Tester les 3 fonctions ensembles.
5. Proposer une variante du programme sans utiliser trois fonctions séparées mais en utilisant les arguments optionnels.

Exercice 4

Programmer et tester en C++ deux fonctions permettant d'échanger les contenus de 2 variables de type int fournies en argument :

- **1ère version** : la transmission des variables concernées est faite par adresse (valable en C et C++)
- **2ème version** : la transmission des variables concernées est faite par référence (valable en C++)

Exercice 5

1. Écrire une fonction `sum(n)` qui renvoie la somme des entiers de `1 à n`.

   Exemple d'utilisation :

```
int i = 6;
int s = sum(i); // s contient 21
```

2. Modifier le prototype de la fonction pour qu'elle effectue la somme de `k à n`. Si `k` n'est pas précisé, la valeur `k = 1` sera utilisée par défaut.

   Exemple d'utilisation :

```
int i = 6;
int s1 = sum2(i); // s1 contient 21
```

```
int s2 = sum2(i, 4); // s2 contient 15
```

3. Modifier la fonction pour qu'elle ne renvoie plus rien et stocke le résultat dans un entier donné en argument, passé par référence.

   Exemple d'utilisation :

```
int i = 6;
int s = 0;
sum3(i, s); // s contient 21
sum3(i, s, 4); // s contient 15
```

4. Modifier la fonction pour qu'elle ne modifie pas `s` si `k` a une valeur incorrecte.

   Exemple d'utilisation :

```
int i = 6;
int s = 0;
sum4(i, s, 42); // s contient 0
sum4(i, s, 4); // s contient 15
```

5. Dans certains cas, on ne peut pas savoir si une modification a bien eu lieu ou non. Par exemple :

```
int i = 6;
int s = 15;
sum5(i, s, 42); // s contient 15 (non modifié)
sum5(i, s, 4); // s contient 15 (modifié)
```

Modifier l'implémentation de sum() pour que l'on puisse distinguer ces deux cas.

Note pour la suite :

En C on stocke des chaines de caractères dans un tableau de char.
En C++ on peut utiliser un type intelligent nommé « string ». Pour l'utiliser on doit inclure string en début de programme (#include <string>). String est un type intelligent très pratique car il alloue automatiquement et dynamiquement ce qui lui est passé en argument.

```
std::string message = "Hello";          // création de la chaîne "Hello"
message += " World !";                  // concaténation de " Word !"
std::cout << message.size << std::endl; // affiche la taille du message.
```

Exercice 6

1) Créer une structure nommée « eleves » qui contiendra 4 données : son âge (**un entier**), son poids (**un flottant**), son nom (**une chaine de caractère** char[] de taille 20) et si il est sympa avec le prof ou non (**un booléen** nommé « sympa »).

Penser à utiliser « typedef » !

2) On va appeler une fonction nommée « remplir » qui sera appelé ainsi :

```
int i = 6;
```

remplir(pierre, 32) pour l'âge.

remplir(pierre, 70.5) pour poids.

remplir(pierre, « pierre ») pour son nom.

remplir(pierre, sympaOuNon) pour sa sympatitude.

L’idée étant d’utiliser la surcharge de fonction et les références pour y arriver.

1. Etablir des vérifications pour que l’âge et le poids ne puissent pas dépasser des valeurs logiques et que le nombre de caractère du nom ne dépasse pas la taille du tableau.
2. Construire une fonction affichage qui prends en paramètre la structure et affiche tous les paramètres.
3. Modifier votre algorithme pour utiliser string à la place de char[]. Créer un tableau de 20 noms d’élèves (tableau de string) et créer une fonction capable de piocher aléatoirement parmi ces noms sans remise (une fois un nom pioché il ne faut plus le re-piocher).
4. Initialiser un tableau de 20 élèves aléatoirement grâce à votre fonction, les autres paramètres doivent aussi être aléatoire, et les trier grâce à une fonction capable d’effectuer un tri radix en fonction de l’âge.