Série 9: `struct`

Lien avec le MOOC Initiation à la Programmation (en C++)

Exercices semaine6 du MOOC : 2nde partie `struct`

- **Document Tutoriel « définition et utilisation d'une structure Personne »**

- **Document Exercices semaine 6 du MOOC (partie struct)**

  - **Exercice 19 : nombres complexes (niveau 1)**
  - passage de structure en paramètre à une fonction

  - **Exercice 20 : QCM (niveau 2)**
  - structure et vector

- **Document Exercices additionnels semaine 6 du MOOC (partie struct)**

  - **Exercice 17 : fractions (niveau 2)** ▪ Illustration du principe de séparation des fonctionnalités : une seule fonction est responsable de la simplification d'une fraction ; toutes les autres fonctions l'utilisent pour simplifier le résultat de l'opération dont elles sont responsables.

Exercice Complémentaire (ExC)

ExC 4 : Using C++ min() and max() functions and sort() algorithm (in English)

1) C++ offers two standard functions that achieve the comparison of two values of any type:

• The function **min**(a,b) returns the minimum of the two parameters passed by const reference

• The function **max**(a,b) returns the maximum of the two parameters passed by const reference

For example, the following code :

```
#include <iostream>
using namespace std; int
main ()
{
    cout << "min(1,2)==" << min(1,2) << '\n';
    cout << "min('z','a')==" << min('z','a') << '\n';
    cout << "max(3.14,2.72)==" << max(3.14,2.72) << '\n';
    return 0;
}
```

Print this output:

```
min(1,2)==1 min('z','a')==a
max(3.14,2.72)==3.14
```

ExC4.1 Exercise : read 4 values (either integer, floating point or character) and use the functions to get the min and the max of those 4 values.

2) C++ also ease the task of sorting a **vector** of any type ! You can use this feature for the project.

Here is how to proceed:

- **#include <algorithm>** // add at the beginning of your code
- Use the **begin()** and **end()** methods provided for a **vector** object. These methods return an *iterator* used by the **sort**() function to know the boundaries of the sorting task in a standardized way.

Here is a first example that modify the vector v containing integer values :

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

int main ()
{
   vector<int> v({9, 2, -1, 0, 3, 7, -5, 11});

   cout << "Avant le tri " << endl;
   for( auto elem : v)
       cout << elem << ' ' ;

   //par défaut, le tri est effectué dans l'ordre croissant
   sort(v.begin(), v.end());

   cout << endl << endl << "Après le tri " << endl;
   for( auto elem : v)
       cout << elem << ' ' ;

   return 0;
}
```

In case you need to manipulate a more complex data structure or if you want to specify a specific sorting algorithm, you can write the instructions in a function that you provide as a parameter to the **sort** function. The provided function must have two parameters (the value of 2 data structures you want to compare) and it must return a boolean value. For example, let's image you want to sort the vector **v** from the previous example in the decreasing order instead of the increasing order. We are free to name the function as we want ; let's call it **my_sorting_order**. Here is its definition:

```
bool my_sorting_order(int i, int j)
{
 return i > j; }
```

The call to the sort function should now be the following to obtain a sorted vector in decreasing order:

```
sort(v.begin(), v.end(), my_sorting_order);
```

exC4.2 exercise: sorting a **vector** of **struct** according to different fields of the **struct** Define a structure **Person** with 2 fields: a **name** and an **age**.
Create and initialize a **vector** of **Person** with a loop ; quit the loop when a negative value is provide for the age. Create 2 functions that allow you to sort the vector according to the name or according to the age of the Person. Print the two sorted vectors in the terminal.

ExC4.3 exercise: sorting a **vector** of **vector**
Declare an empty **vector** of **vector** of **double**: vector<vector<double>> **tab**
Declare a **vector** of 2 **double** values : vector<double> **element**(2);

Write a loop that fills 2 values in **element** and add **element** to **tab** with **push_back()** (until the value pair is a specific value pair e.g. (-1,-1)

Sort the tab according to the first or second elements. For this create 2 functions that allow you to sort the vector tab according to the first or the second value of each of its elements.

Print the two sorted vectors in the terminal.

Example:
Input {{1.2, 3.4}, {0.8, 5.6}, {7.9, 2.8}}

//sort by the first number in an increasing order.
Output1: {{0.8, 5.6}, {1.2, 3.4}, {7.9, 2.8}}

```
//sort by the second number in an increasing order Output2:
{{7.9, 2.8},{1.2, 3.4},{0.8, 5.6}}
```

ExC5: Opérateurs bit à bit (niveau 1)

L'intérêt des opérateurs bit à bit et un exemple d'extraction d'un groupe de 5 bits sont détaillés séparément.

a) Ecrire une fonction **f1** recevant en paramètre un entier non-signé **n** et un second entier non-signé **index** compris entre 0 et 31 et qui renvoie la valeur du bit de rang index dans n.

b) Ecrire une fonction **f2** recevant en paramètre :
- un entier non-signé **n**
- un entier non-signé **index** compris entre 0 et 31
- une **val** entière 0 ou 1

et qui modifie **n** en affectant le bit de rang **index** à la valeur **val**.

c) Ecrire une fonction **f3** recevant en paramètre :
- un entier non-signé **n**
- un entier non-signé **witdh** compris entre 1 et 32
- un entier non-signé **shift** compris entre 0 et 32-width
- une **val** entière non-signée comprise entre **0** et $\mathbf{2^{width} -1}$

et qui modifie **n** en insérant **val** sur les bits situés entre les rangs **shift** et **shift+width-1**

Complément Projet : outil de recherche de bug

L'écriture et les test des premières fonctions peut commencer en testant d'abord une seule fonction, puis une fois validée, on peut passer au test de la fonction suivante etc… Dans ce contexte, le fait de travailler avec des vector peut conduire à un arrêt brutal du programme avec « segmentation fault » ou « bus error » « core dump »… Il faut alors identifier la ligne de code fautive. La localisation en faisant afficher des messages avec cout est populaire mais pas toujours efficace.

Le débugger ddd peut nous aider pour trouver les bugs pendant l'exécution:

- Pour utiliser un programme de déverminage, il faut ajouter l'option **`-g`**

  `g++ -Wall -std=c++11 -g -o monprogramme monprogramme.cc`

- Lancer le dévermineur dans un terminal : **`ddd monprogramme`**
- Démarrer **`monprogramme`** dans **`ddd`** avec run ou run *arguments* en cas de redirection
- Après un crash, choisir Backtrace dans le menu **status** pour savoir l'état de la Pile On y voit la succession d'appels de fonctions à partir de main()

- Pour suspendre l'exécution du programme à des endroits précis utiliser le bouton « **Break** »
- Pour exécuter pas à pas : **`next`** ou **`step`**
- Pour regarder le contenu d'une variable :
  - mettre la souris dessus
  - écrire dans la fenêtre ddd la commande **`print`** ***`nom_variable`***
  - écrire dans la fenêtre ddd la commande **`display`** ***`nom_variable`***
    - La valeur de la variable est alors affichée à chaque pas de programme.