Information, Calcul et Communication

Composante Pratique: Programmation C++

Semaine 14: exemples

Analyse des consequences d'un débordement de tableau (array)

Complément sur la manipulation d'une chaîne de caractères (string)

Où sont nos données ? Pile ? Tas ?

Pour comprendre certains bugs il est nécessaire de savoir où se trouve l'espace mémoire dans lequel existent nos données:

- **vector** (cours semaine 12) : une petite structure de taille fixe sur la **pile** mémorise des pointeurs vers l'espace de mémorisation des données dans le **tas**.

- **string** : des tests avec sizeof montrent aussi qu'une petite structure de taille fixe est mémorisée sur la **pile** et que la chaîne de caractère elle-même est mémorisée dans le **tas**.

- **array** : les données existent sur la pile

sizeof_string.cc

```cpp
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
    string vide;
    cout << sizeof(vide)   << endl;

    string courte("EPFL");
    cout << sizeof(courte)             << endl;

    string longue("12345678901234567890123456789012345678901234567890");
    cout << sizeof(longue)             << endl;

    return 0;
}
```

sizeof_array.cc

```cpp
#include <array>

using namespace std;

int main()
{
    array<int,1> seul;
    cout << sizeof(seul)   << endl;

    array<int,10> dix;
    cout << sizeof(dix)   << endl;

    array<array<int,10>,3> grille3x10;
    cout << sizeof(grille3x10)   << endl;


    return 0;
}
```

Effet d'un débordement de tableau lors d'un appel de fonction

Un débordement de tableau sur la pile peut modifier l'adresse de retour d'une fonction, ici il s'agit de l'adresse de retour quand l'appel de f() est terminé.

adresse_retour.cc

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

void f();

int main()
{
    cout << "main() appelle la fonction f() après ce message" << endl;
    f();
    cout << endl <<"L'appel de f() est terminé / nous revoilà dans main()..." << endl ;

    return 0;
}
```

```cpp
void f()
{
    array<int,2> tab_a={1,2};
    array<int,2> tab_b={3,4};

    while(true)
    {
        size_t i(0);
        int  val(0);
        cout << endl << "lecture d'entiers positifs dans tab_a" << endl;
        do
        {
            cin  >> val;
            if(cin.eof()) return; // retour dans main() si on frappe Ctrl-D
            if(val > 0)   tab_a[i]=val;
            ++i;
        }while(val > 0); // fin de lecture si la valeur lue est négative

        cout << "Affichage des 2 tableaux" << endl;
        cout << "tab_a[0] : " << tab_a[0] << "   tab_a[1] : " << tab_a[1] <<endl;
        cout << "tab_b[0] : " << tab_b[0] << "   tab_b[0] : " << tab_b[1] << endl;
    }
}
```

Analyse d'une chaîne de caractère: conversion romain-> entier

On utilise une variante de la méthode **find** de string à la ligne 44: rom2dec.cc

romain.find(rom[i], start) recherche la string **rom[i]** à partir de l'indice **start** dans **romain**

```
31  unsigned rom2dec(string romain)
32  {
33      if(romain.size() == 0) return 0;
34
35      // ensemble des 13 string de motifs corrects dans un nombre romain
36      array<string,13>  rom({"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"});
37      array<unsigned,13> val({1000,900,500,400,100, 90,  50, 40,  10, 9,   5,   4,  1 });
38
39      unsigned taille_motif(1);
40      unsigned nombre(0);
41      size_t   start(0);
42      for(size_t i(0); i < 13 ; ++i)
43      {
44          while(romain.find(rom[i],start) == start)
45          {
46              nombre += val[i];
47              start  += taille_motif ;
48          }
49          taille_motif = taille_motif%2 + 1;
50      }
51      return nombre;
52  }
```