Série 11: pointeur

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

Exercices semaine7 du MOOC : pointeur et allocation dynamique

Exercice complémentaire (ExC)

ExC 4: évaluation de code sans le compiler (niveau 1)

p1 pointe sur **a**, **p2** et **p3** pointent sur **b**. Les affectations avec **indirection** sur **p2** ou sur **p3 à gauche de l'opérateur '='** modifient **b**.

De même l'affectation avec une **indirection** sur **p1 à gauche de l'opérateur '='** modifie a. Les valeurs finales sont:

a: 27

b: 20

ExC 5 : Type et valeur d'expressions avec des pointeurs et un tableau à-la-C (niveau 1)

a) La valeur de l'expression *ptr_1 + *ptr_2 est 12.
La valeur de l'expression c [1] == *ptr_2 est 1.
La valeur de l'expression *c + b est 3.
La valeur de l'expression *(c+b) est 7.
Celle de l'expression ptr_1 == ptr_2 ? 2 * *ptr_1 : 3 * *ptr_2 ; est 6.

b) La valeur des variables a et b après exécution de l'instruction *ptr_1 = *ptr_2 + a
est 12 pour a et b reste constant, de valeur 2.

c) Le type de l'expression &ptr_1 est int **.
Le type de l'expression *ptr_1 est int.

d) L'instruction c++; n'est pas valide, car c est un pointeur constant ; il ne peut pas être à gauche d'un symbole d'affectation. On dit encore que c n'est pas une lvalue (le compilateur produira un message d'erreur).

Par contre ptr_3++; est valide ; sa nouvelle valeur d'adresse est l'adresse de l'élément suivant du tableau c.

ExC 6: passage d'arguments à la fonction main() et manipulations de chaînes à-la-C (niveau 1)

a) afficher toutes les chaînes à la suite sur une ligne :

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
     for(int i(0); i<argc ; ++i)
     {
          cout << argv[i] << "  ";
     }
     cout << endl;

    return 0;
}
```

b) afficher les chaînes selon des options :

```
#include <iostream>

using namespace std;


bool valid_option(char c, bool& reverse, bool& single)
{
     if( c == 'r')
     {
          reverse = true;
          return true;
     }
     else if( c == 's')
     {
          single = true;
          return true;
     }

     cout << "Incorrect option : accepted options are "
               "r for reverse and s for single"  << endl;

     return false;
}
```

```
int main(int argc, char* argv[])
{
     if(argc > 1)
     {
          bool reverse(false), single(false), option(false);

          if(argv[1][0] == '-') // une option devrait être présente
          {
               if(valid_option(argv[1][1], reverse, single))
               {
                    if(argv[1][2] != '\0') // éventuelle seconde option
                         if(!valid_option(argv[1][2], reverse, single))
                              return 0;
               }
               else return 0;
               option = true;
          }

          if(reverse)
          {
               for(int i(argc-1); i > 1 ; --i)
               {
                    cout << argv[i] << "  ";
                    if(single)
                         cout << endl;
               }
          }
          else
          {
               for(int i(option?2:1); i<argc ; ++i)
               {
                    cout << argv[i] << "  ";
                    if(single)
                         cout << endl;
               }
          }
          cout << endl;
     }
    return 0;
}
```

ExC 7 : Réseau d'amis (niveau 2)

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

struct Personne; // forward declaration = predeclaration

typedef vector<const Personne*> Liste_Personnes;

struct Personne
{
  string nom;
  Liste_Personnes amis; // pointeurs vers les amis
};


bool est_ami_avec(const Personne& pers, const string& nom);
void afficher_ami(const Personne& pers);
void ajouter_ami(Personne& pers, const Personne& new_friend);
void enlever_ami(Personne& pers, const Personne& old_friend);


bool est_ami_avec(const Personne& pers, const string& nom)
{
      for (auto element : pers.amis) {
            if (element->nom == nom) return true;
    }
    return false;
}

void ajouter_ami(Personne& pers, const Personne& new_friend)
{
    if (not est_ami_avec(pers, new_friend.nom))
        pers.amis.push_back(&new_friend);
}

void enlever_ami(Personne& pers, const Personne& old_friend)
{
    if (pers.amis.size() >0){
          for(size_t i(0) ; i < pers.amis.size() ; ++i){
                if(pers.amis[i]->nom == old_friend.nom){
                      if(i != pers.amis.size()-1)
                            pers.amis[i] = pers.amis.back();
                      pers.amis.pop_back();
                      return;
                }
           }
    }
}

void afficher_ami(const Personne& pers)
{
      cout << endl << "Les amis de " << pers.nom << " sont :"  << endl;
      for (auto element : pers.amis) {
            cout << element->nom << endl;
      }
      cout << endl;
}
```

```
// ensemble de tests
int main()
{
      Personne alice={ "Alice" , {} };
      Personne bob  ={ "Bob"   , {} };
      Personne asma ={ "Asma"  , {} };
      Personne bart ={ "Bart"  , {} };
      Personne zorro={ "Zorro" , {} };
      Personne vide ={ ""      , {} };

      alice.amis.push_back(&bob);
      asma.amis.push_back(&bob);
      bart.amis.push_back(&bob);

      bob.amis.push_back(&asma);
      bob.amis.push_back(&bart);
      bart.amis.push_back(&alice);

      afficher_ami(alice);
      afficher_ami(bob);
      afficher_ami(asma);
      afficher_ami(bart);
      afficher_ami(zorro);
      afficher_ami(vide);

      if(est_ami_avec(bob,"Asma"))
            cout << "Bob est ami de Asma" << endl;

      if(!est_ami_avec(bob,"Asma"))
            cout << "Bob n'est pas ami de Zorro" << endl;

      ajouter_ami(alice, bob);
      ajouter_ami(alice, zorro);
      ajouter_ami(zorro, bob);

      afficher_ami(alice);
      afficher_ami(bob);
      afficher_ami(zorro);

      enlever_ami(alice,bob);
      enlever_ami(bob,asma);
      enlever_ami(bob,bart);
      enlever_ami(bob,zorro);

      afficher_ami(alice);
      afficher_ami(bob);
      afficher_ami(zorro);
  return 0;
}
```