Développement C **L2 CUPGE 2018–2019**

Travaux pratiques Types

I Exercice : *Déclarations de variables*

Pré-requis : Thème Entrées/Sorties, exercice 1.

Rappel : Elle est déclarée par son type (qui est le type des valeurs qu'elle peut contenir) et son adresse dans la mémoire. Quelques règles pour l'utilisation de variables :

1. toute variable doit être déclarée dans une fonction ;
2. Le compilateur assure que toute variable déclarée est définie (a une case correspondante dans la mémoire) de manière unique pour chaque utilisation de la fonction ;

   En pratique, on ne fait pas de différence entre une variable et la case dans la mémoire qui lui est associée.

3. une variable est déclarée par un nom et un type :

   Exemple 1 :
   - `int x` déclare une variable de nom `x` et de type `int`
   - `char c` déclare une variable de nom `c` et de type `char`

4. Pour les déclarations complexes, on écrit les opérations qu'il faut faire sur la variable pour obtenir le type au début de la déclaration :

   Exemple 2 : `char * argv[]` signifie que `* argv[i]` (pour un entier `i`) est de type `char` ;
   Donc `argv[i]` est l'adresse d'une case qui contient un `char`
   Donc `argv` est un tableau dont les cases contiennent des adresses de cases de type `char`.

(a) Dans le répertoire ~/workspace/src/Types, ouvrir un nouveau fichier `exercice1.c`, et écrire un programme minimal.

(b) Modifiez le programme pour afficher la valeur de `argc`.

(c) Compilez et exécutez le programme.

(d) Exécutez le programme en mettant des mots et des nombres après le nom du programme (sur la même ligne). Que vaut `argc` ?

II Exercice : *Adresses*

Les adresses ne sont pas des entiers, car les opérations possibles sont différentes (on ne peut pas additionner, diviser ou multiplier des adresses), mais elles sont stockées dans la mémoire dans le même format que les entiers. L'adresse d'une variable `x` est obtenue avec `& x`. Une adresse peut être affichée avec la directive `%p`.

(a) Créez un nouveau programme déclarant deux variables `x0` et `x1` de type `int`, et affichez les adresses de ces 2 variables.

(b) On peut soustraire une adresse à une autre. Le résultat est un entier long (de type `long int`, avec la directive d'affichage `%ld`). Modifiez le programme précédent pour afficher la différence entre les adresses de `x0` et `x1`.

(c) Ajoutez une variable `x2` et calculez les différences.

(d) Pour la suite, on suppose que les variables sont déclarées dans l'ordre `x0`, `x1`, `x2`. Suit on suit la logique de la question précédente, l'adresse de `x2` est l'adresse de `x0` plus 2, et celle de `x1` est celle de `x0` plus 1. Donc en utilisant `*`, essayez de changer la valeur des variables `x1` et `x2` juste en utilisant l'adresse de `x0` (Il faut initialiser toutes les variables à zéro pour que le compilateur soit satisfait).

(e) Changez la valeur à l'adresse de `x0` moins 2 ou `x0` moins 4, et affichez `argc`.

(f) Changez les types de `x0`, `x1`, et `x2` en `char` au lieu d'`int`. Que remarque-t'on ?

(g) Pour conclure, `sizeof` calcule le nombre de cases qu'il faut pour stocker une variable. Affichez ce nombre pour les types suivants :

`char`, `short int` , `int`, `long int`, `float`, `double`, `int *`, `char *`

III Exercice : *Tableaux*

Ce qu'on a vu lors de l'exercice précédent n'est pas "normal'' : on a utilisé notre connaissance du fonctionnement de gcc pour savoir quelle était l'adresse d'autres variables à partir d'une adresse connue. La méthode *standard* pour déclarer plusieurs variables du même type en C est de déclarer un tableau.

(a) Dans un nouveau fichier `exercice.c` et :

1. remplacer les 3 déclarations de variables par :

```
int x[3] ;
```

2. remplacer partout `x0` par `x[0]`, `x1` par `x[1]`, et `x2` par `x[2]`.

Compilez et exécuter le programme.

La déclaration `int x[3]` déclare une suite (un tableau) de 3 variables `x[0]`, `x[1]`, et `x[2]`.
`x` n'est pas une variable !

(b) Affichez la valeur de `x` en utilisant la directive des adresses. Que remarque-t'on ?

(c) Affichez l'adresse de `x`. Affichez les tailles de `x` et de `& x`. Que remarque-t'on ?

Les tableaux ont les propriétés suivantes :
- `x[i] = * ( x + i )`
- `& x[i] = & * ( x + i ) = x + i`

(d) Vérifiez ces égalités en simplifiant le programme, et en vérifiant que les résultats sont les mêmes.