Algorithmique et programmation

Pointeurs sur fonctions

R.Gosswiller

Sommaire

Principes

Rappels sur les pointeurs

Définition

Un pointeur est une variable typée dans laquelle est rangée une adresse mémoire.

Syntaxe

```
1 int a = 10;
2 (int *) ptr;  //l'adresse d'une variable qui contiens un entier
3 ptr = &a;  //ptr contiens l'adresse de a / pointe vers a
```

Définitions

Dans cet exemple, on dit que ptr référence a.

Pointeurs vers variables

Syntaxe

```
1  *ptr = a;  // *ptr -> 'ce vers quoi pointe ptr'
```

Définitions

On utilise l'opérateur * pour déréférencer ptr, c'est à dire obtenir la valeur référencée par le pointeur.

Attention

Les pointeurs vers des variables de types différents sont eux-même différents entre eux.
(int *)+1 != (double *)+1

Architecture Von Neumann

Découpage d'une machine

1. L'unité arithmétique et logique (UAL ou ALU en anglais) ou unité de traitement : son rôle est d'effectuer les opérations de base ;
2. L'unité de contrôle, chargée du « séquençage » des opérations ;
3. **La mémoire qui contient à la fois les données et le programme qui indiquera à l'unité de contrôle quels sont les calculs à faire sur ces données.**
4. Les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.

Problématique

Que ce passe-t-il si un pointeur vise la partie de la mémoire qui contiens les programmes plutôt que les données ?

Pointeurs vers fonctions

Principe

Définitions

Il est possible de faire un pointeur vers une fonction.

Exemple

```
1                int (*pf)(int);
```

Définitions

Ici, pf est un pointeur vers une fonction qui à comme argument d'entrée un entier et qui renvoie un entier.

Attention

Comme pour des pointeurs vers des types de variables différents, des pointeurs vers des fonctions d'arguments d'entrée ou de retour différents ne se mélangent pas !

Utilisation

Compilation

```
1  #include <stdio.h>
2
3
4  static int triple(int a)
5  {
6      return a * 3;
7  }
8
9
10 int main(void)
11 {
12     int (*pt)(int) = &triple;
13
14     printf("%d.\n", (*pt)(3));    //affiche 9
15     return 0;
16 }
```

Syntaxe avancée

Attention

La syntaxe des pointeurs de fonction n'est pas aussi rigide que celle d'un pointeur de variable !

Syntaxe

```
1 pf = &triple; //correct
2 pf = triple;  //correct aussi
3
4 triple(3); //correct
5 (*pf)(3);  //correct
6 (pf)(3);   //erreur
```

Principe

Faire un appel de fonction, c'est aussi utiliser un pointeur.

Manipulation des pointeurs

Pointeurs comme arguments

Pointeur comme argument

```
1  #include <stdio.h>
2
3  static int triple(int a)
4  {return a * 3;}
5
6  static int quadruple(int a)
7  {return a * 4;}
8
9  void affiche(int a, int (*pf)(int))
10 {printf("%d.\n", (*pf)(a));}
11
12 int main(void)
13 {
14     affiche(3, &triple);
15     affiche(3, &quadruple);
16     return 0;
17 }
```

Intérêt

Renvoi d'adresse (callback)

```
1  #include <stddef.h>
2  #include <stdio.h>
3
4  static void affiche_pair(int a)
5  {printf("%d->pair\n", a);}
6  static void affiche_impair(int a)
7  {printf("%d->impair\n", a);}
8
9  static void (*affiche(int a))(int)
10 {    if (a % 2 == 0)
11         return &affiche_pair;
12     else
13         return &affiche_impair;}
14
15 int main(void){
16     void (*pf)(int);
17     int a = 2;
18     pf = affiche(a);
19     (*pf)(a);
20     return 0;}
```

Exceptions

Restrictions

Certaines opérations de pointeurs de peuvent pas être faites avec des pointeurs de fonctions.

Exemple

```
1 int (*pf)(int) = doubleurEntiers;
2 pf = pf + 1; //erreur
```

Restrictions

Avec des fonctions qui utilisent void ou un nombre d'aguments non-constant, certaines conditions s'appliquent...
...que nous verrons dans le prochain cours.

Conclusion

- typeRetour (*nomPointeur)(arguments) créé un pointeur de fonction
- Ces pointeurs peuvent alors être manipulés dans d'autres fonctions
- Attention aux différentes restrictions !