Algorithmique et programmation

Pointeurs

R.Gosswiller

Tableaux à deux dimensions

Tableaux à deux dimensions

Principe

Un tableau à deux dimentions est un tableau défini par deux entiers statiques et un type de contenu
Tableau de tableaux
Cases numérotées en deux dimensions

Rôle

Composition d'informations
Gestion multiple de données
Tableaux de chaines (ex : argv)

Tableaux à deux dimensions

Utilisation double de l'opérateur [ ]

Syntaxe

```
1  #define LIG 10
2  #define COL 20
3  int tab[LIG][COL];
4  tab[3][4] = value;
5
6  char * tab[20];
7  tab[0] = "hello";
8  tab[1] = "world";
9
10 printf("%c", tab[1][1]); // Affiche o
11 printf("%s", tab[1]); // Affiche world
```

L'adressage mémoire

Rappel sur l'hexadecimal

- Représentation en base 16(Décimal : base 10, Binaire : base 2)
- Regrouper les bits 4 par 4
- Synthétiser la représentation binaire
- 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- $27_d = (00011011)_b = 0x1B$
- Utilisé pour représenter les adresses mémoire

Comprendre l'adressage mémoire

Définition

La mémoire, en informatique, est l'espace de stockage de la machine. Il s'agit de la zone où sont lues et écrites toutes les informations liées à un programme ou un fichier.

La mémoire

Assemblage de bits

- 8bits = 1 octet(byte)
- Chaque octet a son adresse propre

Comprendre l'adressage mémoire

Définition

En informatique, créer une variable correspond à allouer en mémoire une suite d'octets(continue) dont la taille dépend du type de la variable : 4 octets pour un int, 2 pour un short int, ... On appelle ce procédé **l'allocation** de mémoire.

Principe

- Type statique = taille statique
- Tableau = taille statique
- $Taille = tailledutype * nombredecases$

La fonction sizeof

Principe

La fonction sizeof renvoie la taille en octets d'un élément stocké en mémoire ou d'un emplacement typé.

Syntaxe

```
1 sizeof(int);    //renvoie 4
2
3 sizeof(element);
```

Les pointeurs

Définir les pointeurs

Problème

Comment créer une référence vers des données de taille dynamique ?

Solution

Définir des variables directement liées aux adresses

Définir les pointeurs

Définition

Un pointeur est une variable contenant une adresse mémoire.
Cette adresse pointe vers une autre case mémoire, servant à stocker une donnée.

Principe

- Donnée de type adresse
- Lié à une données typée
- Taille fixe (4 octets pour 32bits, ou 8 octets pour 64 bits)

Syntaxe

```
1 int * iValue;
```

A quoi sert un pointeur ?

Rôle

- Stocker une adresse
- Travailler sur les adresses (permanentes) plutôt que sur les valeurs (limitées par leur portée et leur taille)
- Tableaux et structures dynamiques
- Fonctions qui ont besoin de communiquer plus qu'une valeur au reste du programme

Opérations sur les pointeurs

Opérateur *

Principe

Créer un poiteur vers un type de variable
Pointer sur la donnée stockée dans la case pointée
Modifier la valeur poitée à une adresse

Syntaxe

```
1  int a = 5;
2  int * ptr = &a;
3  printf("%p", ptr);    //0x1526847..
4  printf("%d",*ptr);    //5
5
6  *ptr = 144;
7  printf("%d",*ptr);    //144
8  printf("%d", a);      //144
```

Opérateur &

Principe

Récupération d'une adresse de variable
L'opérateur & appliqué à un pointeur donne l'adresse d'un pointeur (et non l'adresse pointée)

Syntaxe

```
1 int a = 5;
2 int * ptr = &a;
3 printf("%p",&a);      //0x1526847..
4 printf("%p", ptr);    //0x1526847..
5 printf("%p",&ptr);    //0x4D256F4..
```

Opérations sur cases pointées

Principe

Un pointeur permet d'opérer sur l'adresse d'une variable
Une variable peut être modifiée sans être renvoyée par une fonction

Syntaxe

```
1  void add(int * a,int * b) {
2          *a = *a + *b;
3  }
4  int main(int argc, char * argv[]){
5          int a = 5;
6          int b = 12;
7
8          add(&a,&b);
9          printf("%d", a); // Affiche 17
10
11         return 0;
12 }
```

Récupération de variables multiples

Principe

Le contenu d'une case mémoire peut être modifié et conservé hors fonction
Modification de plusieurs valeurs dans une même fonction

Syntaxe

```
1  int square(int * a,int * b) {
2          *a = *a * *a;
3          *b = *b * *b;
4          return 0;
5  }
6  int main(int argc, char * argv[]){
7          int a = 5;
8          int b = 12;
9
10         square(&a,&b);
11         printf("%d,%d", a, b); // Affiche 25,144
12         return 0;
13 }
```

Pointeurs et tableaux

Tableaux d'adresses

Principe

t[i] est équivalent à *(t + i)

Le compilateur comprends ce i comme i*sizeof(type de variable du tableau)
Tout pointeur est assimilable à l'accès à la mémoire en tant que tableau !

- Ce tableau parcours l'ensemble de la mémoire
- Limitations autour de ce qui à été attribué au programme

Tableaux d'adresses

Décalage d'adresses puis pointage vers la case demandée

Syntaxe

```
1                       char * ch = "couscous";
2                       int lim = strlen(ch);
3                       for(int i=0;i<lim;i++){
4                               printf("%c",ch[i]);
5                       }
6
7                       for(int i=0;i<lim;i++){
8                               printf("%c",*(ch+i));
9                       }
```

Pointeurs de pointeurs

Principe

Récupéré pour les adresses de variables
Utilisé pour les pointeurs composés (ou pointeurs de pointeurs)
Fonctionnement basé sur des tableaux et tableaux à 2 dimensions

Syntaxe

```
1  int main(int argc, char** argv)
2  {
3          int i = 0;
4          while (i < argc)
5          {
6                  printf("%s\n", *(argv+i));
7                  i++;
8          }
9  }
```

En résumé sur les pointeurs

- Pointeur de taille fixe pour des variables de taille dyanimique
- Manipulation d'adresse et parcours de tableau
- Attention aux pièges entre variables et pointeurs !
- Renvoi d'informations multiples quand c'est nécessaire