Le langage C - Les Pointeurs

3 février 2010

Introduction

Sans pointeur : Accès par nom de variable

- Chaque nom de variable correspond à une zone mémoire
- Accès à la valeur de cette zone par son nom de variable :

```
int a = 12 ;
printf("%d",a) ;
```

- `int b = a ;`
  - Création d'une nouvelle zone mémoire pour b
  - Duplication dans b de la valeur contenue dans a

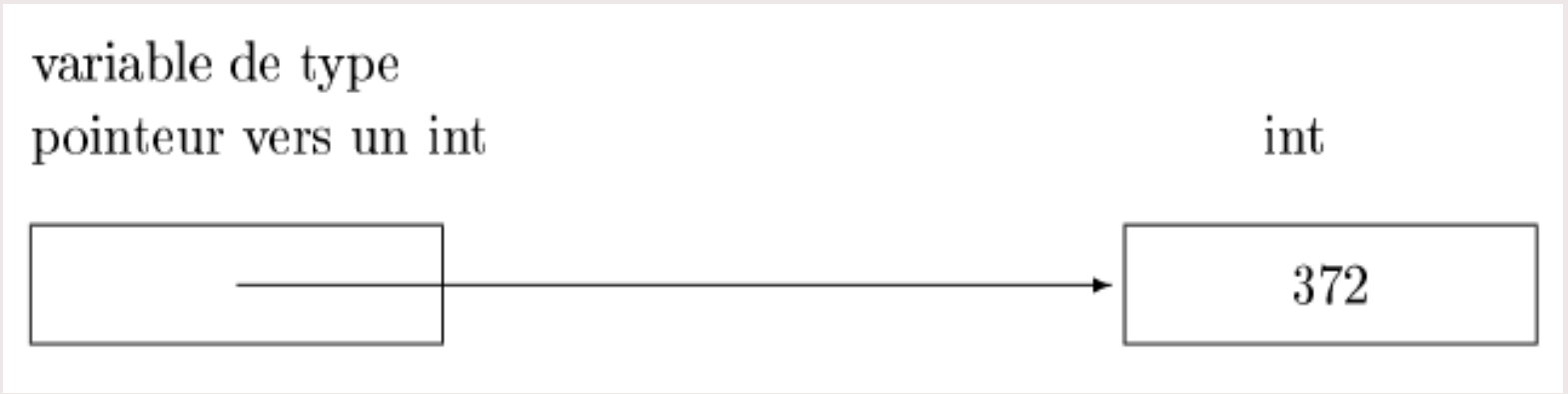
Introduction

Avec pointeur : Accès par adresse mémoire

- Différencier l'adresse mémoire et son contenu
- Pointeur : Contient une adresse mémoire et "pointe" vers elle
- Permet de connaître où est stocké la valeur
- Permet de créer des **paramètres de sortie**

Introduction

Représentation



Techniques de base

Définition

- Indiquer le type de la variable pointée
- Écriture :
  - `type_donnee *nom_pointeur ;`
  - `type_donnee* nom_pointeur ;`

Nouvel opérateur

* signifie "est un pointeur vers"

Techniques de base

Exemples

```
int* p ; // un pointeur vers un entier
float* q ; // un pointeur vers un float
char* r ; // un pointeur vers un caractère
int** s ; // un pointeur vers un pointeur vers un
entier
```

Attention !

- **Les valeurs sont indéfinies à la création !**
- **Comme toute variable, initialiser les pointeurs avant utilisation !**

Techniques de base

Affectation

- Initialiser le pointeur par l'opérateur **&** qui renvoie l'adresse mémoire d'une variable
- Vous le connaissez déjà !

```
int a ;
scanf("%d",&a) ;
```

Exemple

```
int toto = 5 ;
int *tata, *titi ;
tata = &toto ;
titi = tata ;
```

Techniques de base

Indirection

Accéder au contenu de l'adresse mémoire par l'opérateur *

Exemple

```
int a ; // variable
int* p ; // pointeur vers un entier
p = &a ; // affectation du pointeur
*p = 7 ; // initialisation de la valeur
```

a vaut alors 7.
L'appel *p s'appelle **indirection** ou **déréférencement**.

Techniques de base

A retenir !

* et & sont des opérateurs inverses :

```
*(&a) == a
&(*p) == p
```

Techniques de base

Affichage

- Il est possible d'afficher l'adresse mémoire avec le format `%x`, `%X` ou `%p` (valeur hexadécimale) :

```
int a=12;
printf("Valeur de a :%d",a);
printf("Adresse de a :%p",&a);
```

Passage par référence

Exercice

Écrire en pseudo-code une procédure "tripler (ES n : Entier)" qui triple la valeur d'une variable. Écrire un programme principal qui utilise cette procédure.

Problème

Comment implémenter ça en C ?

- Sans pointeur impossible : passage **par valeur.**
- Avec pointeur : passage **par référence** !

Application

Solution

```
void tripler (int* n) {
  *n = *n * 3;
}

int main() {
  int i = 2;
  printf("i = %d\n",i);
  tripler(&i);
  printf("i = %d\n",i);
  return 0;
}
```

Exercices

- Expliquez la présence du & dans la commande `scanf("%d",&n) ;`
- Écrire en C une procédure avec paramètre de sortie qui réalise la somme de 2 entiers dans un 3^e. Écrire un programme principal qui utilise cette procédure.
- Écrire en C une procédure qui permute deux valeurs. Écrire un programme principal qui utilise cette procédure.

Application

Solution

```
void somme (int x, int y, int* s) {
  *s = x + y;
}
void permute (int* a, int* b) {
  int c;
  c = *a;
  *a = *b;
  *b = c;
}
int main() {
  int i = 2;
  int j = 5;
  int k;
  somme(i,j,&k);
  printf("%d + %d = %d\n",i,j,k);
  permute(&i,&j);
  printf("i,j = %d,%d\n",i,j);
  return 0;
}
```

Concept

Pointeurs en tant que début de tableau

Un pointeur peut aussi représenter l'adresse de la première case d'un tableau. Pour parcourir un tableau, on peut ajouter ou soustraire un entier à ce pointeur, ce qui augmente ou diminue l'adresse de p de n*taille(type de *p). Attention aux débordements !

Exemple

```
int tab[5];
int* p;

p = &tab[0]; // pointeur sur tab[0]
p += 4; // pointeur sur tab[4]
p -= 3; // pointeur sur tab[1]
p -= 2; // oh oh...
```

Initialisation

Exemple

```
int tab[100];
int* p = &tab[0];
for (i=0; i<100; i++)
  *(p+i) = 0;
```

Notez

`tab == &tab[0]`

Exemple amélioré

```
int tab[100];
for (i=0; i<100; i++)
  *(tab+i) = 0;
```

Opérations

Opérations

- Autorisées :
  addition, soustraction, comparaison
- Interdites :
  multiplication, division, opérateurs logiques

Exercice

- Écrire en C une procédure qui reçoit un tableau d'entiers et qui renvoit par paramètre de sortie sa plus petite et sa plus grande valeur.

Tableaux dynamiques

Problématiques

- Inconvénients du tableau statique :
  - surconsommation de l'espace mémoire
  - dépassement de capacité
- Tableau dynamique : taille déterminée durant l'exécution du programme
- Comment allouer un tableau sans connaître sa taille par avance ("CréerTab") ?

Tableaux dynamiques

Principe

- Réservation d'une partie de la mémoire
- Utilisation de cette partie
- Libération de la zone

Allocation dynamique

Méthodes impliquées

- `sizeof(element)` opérateur qui retourne la taille en octets d'un élément (type, variable ou pointeur)
- `malloc(t)` qui réserve un espace mémoire de t octets et retourne un pointeur d'adresse du début de la zone. Si `pointeur == NULL`, alors problème d'allocation
- la conversion explicite (appelée aussi "opération de cast") qui consiste en une modification du type de donnée forcée.
- `free(p)` qui libère l'espace mémoire du pointeur p. Pour une gestion correcte de la mémoire, il est indispensable de libérer la mémoire allouée quand on n'en a plus besoin, sinon, votre RAM risque de se remplir inutilement ("fuite de mémoire") !

Tableaux dynamiques

Exemple d'allocation

```
#include <stdio.h>
#include <stlib.h>
int main() {
   int* tab; //tableau dynamique d'entiers
   printf("Combien de cases?");
   int n;
   scanf("%d",&n);

   tab = (int*)malloc(n*sizeof(int)); //réservation
   if(tab == NULL){ // l'allocation a planté
       printf("Erreur!");
       return 1;
   }
   ....
   free(tab); //libération
   return 0;
   }
```

Renvoyer un pointeur par une fonction

Attention

Il est parfaitement possible de renvoyer un pointeur (par exemple, un tableau) par une fonction. Mais attention : vous ne pouvez renvoyer que les pointeurs alloués dynamiquement ! Toute mémoiré allouée statiquement est détruite à la sortie de la fonction.

A ne pas faire

```
int* mauvaisExemple() {
  int tab[100] ;
  int i ;
  for (i=0 ;i<100 ;i++)
    tab[i] = 0 ;
  return tab ;
}
```

A faire

```
int* bonExemple() {
  int* tab = (int*)malloc(100*sizeof(int)) ;
  int i ;
  for (i=0 ;i<100 ;i++)
    tab[i] = 0 ;
  return tab ;
}
```

Notez

Astuce

On peut utiliser `sizeof` pour détecter le nombre de cases d'un tableau en C (statique ou dynamique) :

```
int* tab = bonExemple();
printf("octets utilisés par le tableau : %d\n", sizeof(tab));
printf("octets utilisés pour une case : %d\n", sizeof(tab[0]));
printf("donc nb de cases : %d\n", sizeof(tab)/sizeof(tab[0]));
...
free(tab); // ne pas oublier !
```