Développement logiciel (S4)

Initiation au langage C
4- Les pointeurs

Objectifs : Être capable de développer des applications basiques de type console en C. Être capable d'utiliser les pointeurs.

Avertissement

Tout au long de ce TP, vous répondrez au questions posées sur votre compte-rendu. Ce compte-rendu devra inclure les programmes réalisés. Ces programmes devront être commentés !

Qu'est-ce qu'un pointeur ?

Introduction

Les pointeurs sont une fonctionnalité assez complexe pour les débutants en C. Il est possible de développer de nombreux programmes sans utiliser les pointeurs. Toutefois, l'utilisation des pointeurs peut simplifier le codage dans de nombreux cas et c'est un outil qu'un développeur en C doit forcément connaître.

Définition

Un pointeur est une variable qui contient l'adresse d'une autre variable.

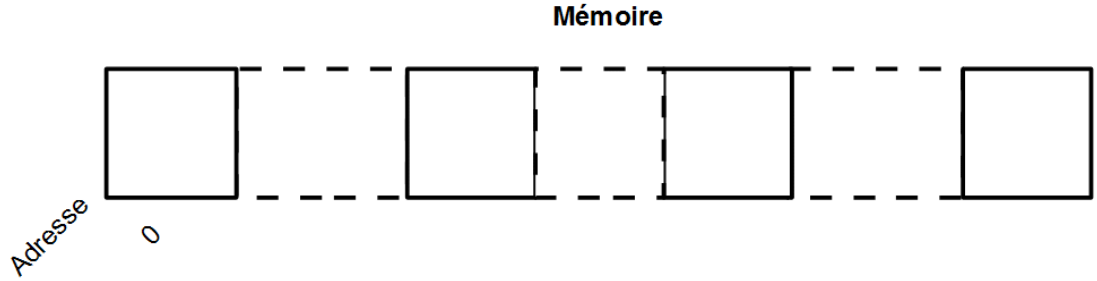
Petites manipulations sur les adresses

On rappelle que le symbole &i représente l'adresse de la variable i. C'est la syntaxe que l'on utilise lorsque on utilise la fonction scanf.

Manipulation n°1

Q.1) Saisir et exécuter le programme donné ci-après.

Q.2) Compléter le schéma suivant en renseignant les adresses et les données.



Programme à tester

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i = 3 ; // déclaration de deux entiers : i =3
    int j = 4 ; // et j= 4

    printf("Valeur de i = %d \n", i); // affiche la valeur de i
    printf("Valeur de l'adresse de i = %d \n", &i); // affiche l'adresse de i

    printf("Valeur de j = %d \n", j); // affiche la valeur de j
    printf("Valeur de l'adresse de j = %d \n", &j); // affiche l'adresse de j

    return 0;
}
```

Première utilisation d'un pointeur (tour de magie?)

Le petit programme suivant est une première application des pointeurs. C'est un petit tour de magie en C. Il permet de modifier une variable (ici la variable i) sans l'appeler !

Manipulation n°2

Q.1) Saisir et exécuter le programme donné ci-après. Conclure.

Q.2) Exécuter le programme pas à pas avec le débugger. Indiquer les différentes valeurs des variables après chaque instruction.

Programme à tester

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i = 0 ; // déclaration de i ;
    int *ptr = NULL; // declaration du pointeur (d'entier)
    ptr = &i ; // initialisation du pointeur à l'adresse de i

    i = 3 ; // initialisation de i à la valeur 5

    // avant
    printf("Valeur de i = %d \n", i);

    // opération sur pointeurs
   *ptr = 4 ; // on affecte à la valeur pointée par ptr la valeur 4

   // après
    printf("Valeur de i = %d \n", i);

    return 0;
}
```

L'opération a permis de modifier la variable i en utilisant le pointeur ptr.

Pour réaliser ce tour de passe-passe, cela c'est fait en plusieurs temps.

La ligne :

```
int *ptr = NULL; // déclaration du pointeur (d'entier)
```

permet de déclarer le pointeur ptr qui est un pointeur d'entier. Remarque : pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (en majuscules).

Ensuite, la ligne :

```
ptr = &i ; // initialisation du pointeur ptr sur l'adresse de i
```

permet d'initialiser le pointeur ptr avec l'adresse de la variable i.

Enfin, la ligne :

```
*ptr = 4 ; // on affecte à la valeur pointée par ptr la valeur 4
```

permet d'affecter 4 à la valeur pointée par ptr, ce qui correspond à la variable i !

Conseil : il est préférable d'appeler les pointeurs par un nom explicite pour faciliter la lecture des programmes (exemple : ptr, ptr_a, etc...)

Applications des pointeurs : fonction à deux sorties

Nous allons voir ici un exemple d'utilisation des pointeurs. Une fonction à deux sorties.

Une fonction à 2 sorties

On étudie ici une fonction toute simple qui possède deux entrées, les entiers a et b et deux sorties, les entiers S et P. S représente la somme de a et b (S = a+b) et P représente le produit de a et b (P = a.b).

En C, la fonction ne retourne qu'une grandeur de sortie (avec l'instruction RETURN). Il n'est pas prévue de retourner plusieurs valeurs !

Tentative sans pointeurs

On pourrait envisager de récupérer les résultats S et P par les arguments de la fonction comme le fait le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>

void SommeetProduit(int x, int y , int S, int P );

int main()
{
    int a = 0;
    int b = 0;
    int Somme = 0;
    int Produit = 0 ;

    // Affichage et saisie données
    printf("Calcul de somme et produit \n" );
    printf("a = ");
    scanf("%d", &a);
    printf("b = ");
    scanf("%d", &b);

    // Calcul
    SommeetProduit(a,b, Somme, Produit);
    // affichage
    printf("Somme = (a+b) = %d \t  Produit = (a.b)= %d \n", Somme, Produit );

    return 0;
}
    void SommeetProduit(int x, int y, int S, int P )
    {
        S = x + y ;
        P = x*y ;
        printf("S= %d et P=%d \n",S,P) ;
        return ;
    }
```

Remarque :

```
void SommeetProduit(int x, int y , int S, int P );
```

Le mot clé void (vide en anglais) représente une fonction qui ne retourne aucune valeur !

Manipulation n°3

Q.1) Tester le programme précédent (fichier donné) et analyser à l'aide du debugger pourquoi il ne marche pas ! Conclure.

Utilisation des pointeurs

Nous allons maintenant voir qu'en utilisant des pointeurs, il est possible de modifier avec succès le programme précédent.

Programme modifié

```
#include <stdio.h>
#include <stdlib.h>

void SommeetProduit(int x, int y , int *ptr_S, int *ptr_P );

int main()
{
    int a = 0;
    int b = 0;
    int Somme = 0;
    int Produit = 0 ;

    // Affichage et saisie données
    printf("Calcul de somme et produit \n" );
    printf("a = ");
    scanf("%d", &a);
    printf("b = ");
    scanf("%d", &b);

    // Calcul
    SommeetProduit(a,b, &Somme, &Produit);
    // affichage
    printf("Somme = (a+b) = %d \t  Produit = (a.b)= %d \n", Somme, Produit );

    return 0;
}
    void SommeetProduit(int x, int y, int *ptr_S, int *ptr_P )
    {
        *ptr_S = x + y ;
        *ptr_P = x*y ;
        return ;
    }
```

Manipulation n°4

Q.1) Modifier le fichier de la manipulation précédente pour obtenir le fichier donné ci-dessus.

Q.2) Exécuter le programme et vérifier qu'il fonctionne correctement.

Q.3) Observer les variables avec le debugger.

Q.4) Conclure en expliquant ce qui a permis de faire fonctionner le programme.

Mini-projet : Calculs sur les nombres complexes

On se propose d'écrire un programme qui calcule la partie réelle et la partie imaginaire d'un nombre complexe en fonction du module et de l'argument de son nombre complexe.
Les données d'entrées sont deux réels module et argument, les grandeurs de sortie sont deux réels partie réelle et partie imaginaire.

Manipulation n°5

Q.1) Rappeler les formules permettant de calculer partie réelle et partie imaginaire en fonction de module et argument. Si vous êtes nuls en maths, vous pourrez vous faire aider par l'ami Google !

Q.2) Écrire un programme permettant de résoudre le problème donné. Attention, l'argument doit être demandé en degrés. Vous développerez une fonction ayant 2 entrées (module et argument) et 2 sorties (partie réelle et imaginaire).

Q.3) Bonus : les étudiants ayant terminé en avance, développerons la fonction réciproque qui permet de calculer module et argument en fonction de partie réelle et imaginaire. Dans un premier temps on se limitera au cas où a est ≥ 0 (ce qui est généralement le cas dans les problèmes d'électricité/électronique), puis on tiendra compte de tous les cas ($a=0$, $a<0$ et $a>0$ conduisent à des arguments différents).

Conseil : penser à utiliser la bibliothèque math.h pour les fonctions trigonométriques.

Retrouvez d'autres cours et documents sur :
http://www.louisreynier.com