ENSEIRB 2006-2007 - Première année
PG101 - Programmation en C

TD 5 : Pointeurs et structures

Cette feuille de TD traite des pointeurs, de la structuration de la mémoire, des adresses en mémoire, et du fonctionnement des appels de fonctions. Elle introduit ensuite la définition des nouveaux types, particulièrement les structures et les énumérations.

Dans votre répertoire nommé `TDc/`, créez un sous-répertoire `TDc/td5/` (avec la commande `mkdir` dans un terminal). Positionnez-vous dans ce sous-répertoire (avec la commande `cd`), vous y resterez toute cette séance.

1 Pointeurs

Exercice 1: Échange

1. Écrivez une fonction `echange` qui a 2 paramètres de type `int` et qui en échange les valeurs. Vous écrirez une fonction principale qui définit et initialise deux variables entières et qui en affiche les valeurs avant et après avoir appelé `echange`
2. Compilez puis exécutez votre programme. Que constatez-vous ?
3. Instrumentez les fonctions `main` et `echange` afin de mettre en évidence le problème
4. Proposez une solution et implantez-là. Compilez le nouveau programme et exécutez-le
5. Instrumentez à nouveau les fonctions `main` et `echange` pour mettre en évidence les modifications qui expliquent le changement de comportement de votre programme

Exercice 2: Un retour sur `scanf`

Dans les feuilles de TD précédentes, la lecture d'un caractère au clavier stocké dans une variable `c` de type `char`, s'écrivait : `scanf("%c", &c)`. Expliquez pourquoi ne n'écrivions pas plutôt : `scanf("%c", c)`

Exercice 3: Pointeurs et espace mémoire

Copiez-collez le programme suivant dans un buffer `emacs`, compilez-le et exécutez-le.

```
#include <stdio.h>

int main() {
  char *c;
  short *s;
  int *n;
```

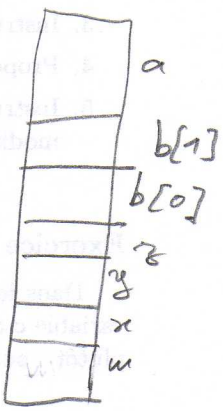
```
  long *l;
  float *f;
  void *v;

  printf("Taille de c=%u, *c=%u\n", sizeof(c), sizeof(*c));
  printf("Taille de s=%u, *s=%u\n", sizeof(s), sizeof(*s));
  printf("Taille de n=%u, *n=%u\n", sizeof(n), sizeof(*n));
  printf("Taille de l=%u, *l=%u\n", sizeof(l), sizeof(*l));
  printf("Taille de f=%u, *f=%u\n", sizeof(f), sizeof(*f));
  printf("Taille de v=%u, *v=%u\n", sizeof(v), sizeof(*v));

  return 0;
}
```

Que constatez-vous ?

Exercice 4: Pile et adresses

Copiez-collez le programme suivant dans un buffer **emacs**, compilez-le et exécutez-le.

```
#include <stdio.h>

int g;

int f(int x, int y[], int *z) {
int w;

  printf("*** f\n");
  printf("Adresse de x : %p\n", &x);
  printf("Adresse de y : %p\n", &y);
  printf("Adresse de z : %p\n", &z);

  printf("Adresse de w : %p\n", &w);

  printf("Adresse de g : %p\n", &g);

  return 0;
}

int main() {
  int a, b[2];

  printf("*** main\n");
  printf("Adresse de a : %p\n", &a);
  printf("Adresse de b : %p\n", &b);

  printf("Adresse de g : %p\n", &g);
```



```
  f(a, b, &a);

  return 0;
}
```

Que constatez-vous ? Expliquez les relations ou l'absence de relation entre les différentes adresses affichées.

Exercice 5: Les appels récursifs vus depuis la pile

Reprenez le programme de calcul de la factorielle par une fonction récursive que vous avez écrit lors du TD précédent. Modifiez la fonction `fact` pour qu'elle affiche l'adresse et la valeur de son paramètre `n` avant et après l'appel récursif. Que constatez-vous ?

Exercice 6: Tableaux et pointeurs

Copiez-collez le programme suivant dans un buffer `emacs`, compilez-le et exécutez-le.

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

int main() {
  int t[] = {1,2,3};
  int *p = NULL;

  printf("Valeur de t=%p\n", t);
  printf("Valeur de p=%p\n", p);

  return 0;
}
```

1. Que constatez-vous ? Que signifie `NULL` ?
2. Ajoutez les lignes suivantes juste avant l'instruction `return 0` :

   ```
   printf("*** Modification de p\n");
   p = t;
   printf("Valeur de p=%p\n", p);
   printf("t[1]=%d\n", t[1]);
   printf("Modification de p[1]\n");
   p[1] = 77;
   printf("t[1]=%d\n", t[1]);
   ```

   Compilez puis exécutez à nouveau le programme. Que constatez-vous ?
3. Ajoutez les lignes suivantes immédiatement après celles que vous venez d'ajouter :

   ```
   printf("*** Modification de t\n");
   t = p;
   ```

   Compilez votre programme. Expliquez ce que vous observez

Que concluez-vous sur les pointeurs et les tableaux ?

Exercice 7: Pointeurs et chaînes de caractères

Copiez-collez le programme suivant dans un buffer `emacs`, compilez-le et exécutez-le.

```
#include <stdio.h>

int main() {
  char s1[] = "Bonjour!";
  char *s2 = "Bonjour!";

  printf("Valeur de s1=%p\n", s1);
  printf("Affichage de la chaine s1 : %s\n", s1);
  printf("Valeur de s2=%p\n", s2);
  printf("Affichage de la chaine s2 : %s\n", s2);

  return 0;
}
```

1. Que constatez-vous ? Quelle différence y-a-t'il dans les initialisations de `s1` et `s2` ?
2. Ajoutez les lignes suivantes immédiatement avant l'instruction `return 0`

   ```
   printf("*** Modification de s1[0]\n");
   s1[0] = '$';
   printf("Affichage de la chaine s1 : %s\n", s1);
   printf("*** Modification de s2[0]\n");
   s2[0] = '$';
   printf("Affichage de la chaine s2 : %s\n", s2);
   ```

   Compilez puis exécutez le programme. Expliquez ce que vous constatez
3. Remplacez les lignes que vous venez d'ajouter par les lignes suivantes :

   ```
   printf("*** Modification de s2\n");
   s2 = "Bonsoir!";
   printf("Valeur de s2=%p\n", s2);
   printf("Affichage de la chaine s2 : %s\n", s2);
   ```

   Compilez puis exécutez ce programme. Que constatez-vous ?
4. Ajoutez les lignes suivantes à la suite de celles que vous venez d'ajouter :

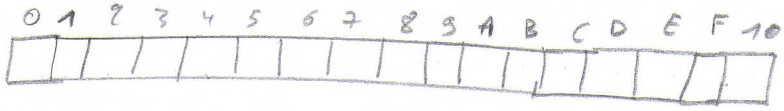
   ```
   printf("*** Modification de s1\n");
   s1 = "Bonsoir!";
   printf("Valeur de s1=%p\n", s1);
   printf("Affichage de la chaine s1 : %s\n", s1);
   ```

   Compilez ce programme. Expliquez ce que vous observez.

Que concluez-vous sur les pointeurs et les chaînes de caractères ?

Exercice 8: Arithmétique de pointeurs

Copiez-collez le programme suivant dans un buffer `emacs`, compilez-le et exécutez-le.




```
#include <stdio.h>

int main() {
  int *p = 0, i;

  for (i=0; i<5; i++)
    printf("%p\n", p+i);

  return 0;
}
```

1. Expliquez les valeurs affichées. Que pouvez-vous conclure de l'utilité de déclarer le type référencé par un pointeur en langage C ?
2. Comment faudrait-il modifier le programme pour se déplacer d'octet en octet dans la mémoire ?

Exercice 9: Références

Modifiez le programme suivant pour qu'il utilise l'arithmétique de pointeur plutôt que la référence via l'opérateur `[.]`.

```
#include <stdio.h>

int main() {
  char s[] = "Bonjour!";
  int i;

  i=0;
  while (s[i] != '\0') {
    printf("%c", s[i]);
    i++;
  }
  printf("\n");

  return 0;
}
```

Déduisez-en une expression équivalente à `t[i]` en arithmétique de pointeur

Exercice 10: Structure des tableaux

Écrivez un programme qui définit un tableau `t` bidimensionnel et qui met en évidence le fait que `t` est un tableau de tableau

Exercice 11: Pointeurs et variable locales

Copiez-collez le programme suivant dans un buffer `emacs`. Compilez-le et exécutez-le.

```
#include <stdio.h>

int *copie_int(int x) {
  int y;

  y=x;

  return (&y);
}

int main() {
  int a, *b;

  a = 3;
  b = copie_int(a);
  printf("a=%d, *b=%d\n", a, *b);
  printf("a=%d, *b=%d\n", a, *b);

  return 0;
}
```

Expliquez ce que vous constatez à l'écran. Que pouvez-vous dire de la modification qui consiste à remplacer `return (&y)` par `return (&x)` dans la fonction `copie_int` ? Quel principe en déduisez-vous ?

2 Définitions de nouveaux types

2.1 Structures

Les structures permettent de regrouper au sein d'un même type plusieurs valeurs dont les types peuvent être différents, au contraire des tableaux dont tous les éléments on le même type. Une structure se déclare à l'aide du mot-clé `struct` suivi du nom de la structure, et entre accolades, de la liste des *champs* qui la composent

Exercice 12:

Définissez une structure `point` qui permet de représenter des coordonnées dans le plan Euclidien. Vous écrirez un programme qui définit une variable du type `struct point`, qui l'initialise et qui en affiche la valeur

Exercice 13:

Copiez-collez le programme suivant dans un buffer `emacs`, compilez-le et exécutez-le.

```
#include <stdio.h>

#define MAX 10
```

```
struct carte_etd {
  char nom[MAX];
  char prenom[MAX];
  unsigned int num_etd;
  char etablissement[MAX];
};

void carte_etd__affiche(struct carte_etd c) {
  printf("Nom: %s\n", c.nom);
  printf("Prenom: %s\n", c.prenom);
  printf("Numero d'etudiant: %u\n", c.num_etd);
  printf("Etablissement: %s\n", c.etablissement);
}

int main() {
  struct carte_etd c1 = {"Dupont", "Philippe", 430568, "ENSEIRB"}, c2;

  carte_etd__affiche(c1);
  c2 = c1;
  carte_etd__affiche(c2);

  return 0;
}
```

Que déduisez-vous du comportement de l'opérateur d'affectation = lorsqu'il agit sur des structures ?

Exercice 14: Taille d'une structure

Écrire un programme qui définit la même structure `struct carte_etd` que l'exercice précédent, qui définit une variable `c` de ce type, qui affiche la place mémoire occupée par `c`, et celle occupée par chacun de ces champs, ainsi que l'adresse en mémoire de `c`, celle de chacun de ces champs, et la première adresse située après `c`.

1. Que constatez-vous ? Pourquoi manipule-t-on systématique les structures champs à champs ?
2. Ajoutez un champs `annee` de type `short` en deuxième position dans la structure et modifiez votre programme pour qu'il affiche les données relatives à ce champs. Que constatez-vous ?
3. Déplacez le champs `annee` en dernière position de la structure. Que constatez-vous ?

Exercice 15: Structures et pile

Estimez le nombre d'octets qui sont empilés à chaque appel à la fonction `carte_etd__affiche`, puis écrivez un programme qui met ce résultat en évidence. Comment proposez-vous de modifier la définition de cette fonction pour améliorer cet aspect ?

2.2 Types énumérés

Les types énumérés permettent de définir de nouveaux ensembles de valeurs, représentés par des noms symboliques. Par exemple, nous pouvons définir un type `bool` :

```
enum bool {
  VRAI,
  FAUX
};
```

Une valeur de type `bool` aura soit la valeur `VRAI`, soit la valeur `FAUX` et aucune autre valeur possible.

Exercice 16: Énumérations

Nous souhaitons enrichir la structure `carte_etd` d'un champ qui précise la filière et l'année dans lequel l'étudiant est inscrit (informatique 3 ou telecom 1, par exemple).

1. Nous proposons d'introduire un champ `inscription` de type `int` :

   ```
   struct carte_etd {
     char nom[MAX];
     char prenom[MAX];
     unsigned int num_etd;
     char etablissement[MAX];
     int inscription;
   };
   ```

   Que pensez-vous de cette solution ?
2. Proposez une autre solution basée sur la définition d'un type énuméré.
3. Écrivez une fonction `carte_etd__affiche` qui affiche une valeur de type `struct carte_etd` et un programme principal qui définit une variable de ce type et appelle votre fonction

2.3 Définitions de types

Le mot clef `typedef` permet de définir de nouveaux noms de types. La définition d'un type s'écrit comme une définition de variable, en préfixant la définition du mot clef `typedef`. Par exemple :

```
typedef int tab5[5];
```

définit le type `tab5` des tableaux de 5 valeurs de type `int`. Il peut alors être utilisé pour définir des variables :

```
tab5 t;
```

`t` est alors un tableau de 5 valeurs de type `int`, qui a le type `tab5`

Exercice 17: Définitions de types

1. Définissez le type `mat10R` des matrices 10×10 de nombres flottants, puis définissez une variable de type `mat10R`

2. Définissez le type `carte_etd_t` qui correspond au type `struct carte_etd`, puis une variable de type `carte_etd_t`. Pourquoi déconseille-t-on une telle définition ?
3. Pourquoi déconseille-t-on de dissimuler un pointeur derrière une définition de type, comme suit :

```
typedef struct carte_etd *carte_etd_t;
```