Méthodologie de conception et de programmation

Cours 3

N. de Rugy-Altherre

1. Allocation dynamique de mémoire
2. Structures
3. Tableaux
4. Chaînes de caractères

Allocation dynamique de mémoire

Rappel

Un pointeur p est une variable contenant une adresse. *p est la valeur pointée par p.

malloc/free

```
void * malloc(size_t size);
void free(void * p);
```

- `malloc` : réserve une zone mémoire contigüe de `size` *bytes* et retourne un pointeur vers son premier élément ;
- `free` : libère la mémoire allouée (qui peut alors être réutilisée).

Allocation dynamique de mémoire

Exemple

```
int * p = malloc(sizeof(int));
*p = 18;
free(p);
```

À voir des exemples dans ArtEoz

Attention

Si on ajoute à la suite :

```
printf("%p", p);     // affiche une adresse
printf("%d", *p);    // erreur de segmentation
```

1 Allocation dynamique de mémoire

2 Structures

3 Tableaux

4 Chaînes de caractères

Structures

Les enregistrements vus en AP1 existent aussi en C.

Définition

Un type enregistrement en C se définit par :

- un nom ;
- une liste de champs.

Exemple

```
struct eleve {
    float moyenne;
    int annee;
};
```

Structures

Erreurs fréquentes

- Il ne faut pas donner de valeur aux champs définis dans une structure.
- Il ne faut pas oublier le point virgule ; final.

Structures

Définition

Le mot clé **struct** fait partie du type d'une variable enregistrement. L'accès à un champ se fait comme en AP1.

Exemple

```
struct eleve {
    float moyenne;
    int annee;
};

int main(int argc, char * argv[]) {
    struct eleve eleve1, eleve2;
    eleve1.moyenne = 10.2;
    eleve1.annee = 1;
    eleve2.moyenne = 4.2;
    eleve2.annee = 1;
}
```

Structures

typedef

Pour éviter le mot clef **struct** à chaque définition de variable, on peut créer un alias :

```
typedef struct eleve eleve_t;
```

Par convention le nom du nouveau type sera suffixé par _t.

Exemple

```
struct eleve {
    float moyenne;
    int annee;
};
typedef struct eleve eleve_t;

int main(int argc, char * argv[]) {
    eleve_t eleve1, eleve2;
    ...
}
```

Structures et mémoire

En mémoire, une variable de type **`struct`** occupe un espace contigu, découpé en autant de blocs qu'il y a de champs.

Attention

Contrairement aux tableaux (ci-après), une variable de type **`struct`** n'est pas un pointeur vers une case mémoire, mais désigne bien le bloc mémoire contenant tous les champs.

Les enregistrements ne sont pas encore implémenter sur ArtEoz.

Structures et pointeurs

Pointeur vers structure

Si `var` est une variable de type **`struct`** `mon_type` alors un pointeur vers `var` est de type **`struct`** `mon_type *` :

```
struct mon_type * p = &var;
```

Syntaxe abrégée

Pour accéder à un champ :

```
(*p).champ = val;  et  x = (*p).champ;
```

ou de façon équivalente :

```
p->champ = val;  et  x = p->champ;
```

Structures et pointeurs

Exemple

```
eleve_t eleve1;
eleve_t * p_eleve1, p_eleve2;

p_eleve1 = &eleve1;
p_eleve2 = malloc(sizeof(eleve_t));

p_eleve1->moyenne = 10.2;
p_eleve1->annee = 1;
p_eleve2->moyenne = 4.2;
p_eleve2->annee = 1;
```

Structures et listes chaînées

Exemple

```
struct element {
    int data;
    struct element * next;
};
typedef struct element liste_t;

int main(int argc, char * argv[]){
    liste_t l = NULL; //liste vide

    l = malloc(sizeof(liste_t));
    l->data = 10;
    l->next = NULL;

    l->next = malloc(sizeof(liste_t));
    l->next->data = 20;
    l->next->next = NULL;
}
```

1 Allocation dynamique de mémoire

2 Structures

3 Tableaux

4 Chaînes de caractères

Tableaux

Deux sortes de tableaux en C

- **statiques** : dimension connue avant la compilation ;
- **dynamiques** : dimension connue à l'exécution.

Tableaux

Tableaux statiques

- un tableau statique a une dimension fixée avant la compilation ;
- en pratique on surdimensionne un tableau, et on conserve la taille effective dans une variable.

Exemple

```
int tab[100];
int len = 40;
for (i = 0; i < len; i++) {
    tab[i] = i;
}
```

Tableaux

Deux façons d'initialiser un tableau : lors de la déclaration avec une syntaxe spéciale, ou après en le parcourant via ses indices.

Exemple

```
// lors de la declaration
int tab[5] = {1,2,3,4,5};

// apres la declaration
int tab[5];
for (i=0; i<5; ++i) {
    tab[i] = i+1;
}
```

Exemple sur Arteoz

Tableaux

Attention

- Un tableau déclaré mais non initialisé a pour valeurs ce que contenait la mémoire.
- Les indices d'un tableau de taille 100 vont de 0 à 99.
- La syntaxe d'initialisation d'un tableau avec accolades n'est possible que lors de la déclaration.
- En C le dépassement d'indice d'un tableau ne génère pas nécessairement d'erreur bloquante à l'exécution !

Tableaux et pointeurs

Tableau $\simeq$ pointeur

Une déclaration `int tab[100];` :

- réserve une zone mémoire contiguë pour 100 entiers `int` ;
- définie `tab` comme un pointeur vers la première case du tableau.

Exemple

```
int tab[5] = {1,2,3,4,5};
int * p = tab;
printf("tab[0]: %d, *p: %d\n",tab[0], *p);
```

produit l'affichage : `tab[0]: 1, *p: 1`

Exemple

Si `tab[0]` a pour adresse `0x01004`, alors `tab[1]` a pour adresse `0x01008` (car 1 `int` nécessite 4 octets).

Tableaux et pointeurs

Attention

Une variable de type tableau n'est pas modifiable comme un pointeur. On ne peut pas faire :

```
int tab[5] = {1,2,3,4,5};
int val = 0;
tab = &val;
```

ni

```
int tab1[5] = {1,2,3,4,5};
int tab2[5];
tab2 = tab1;
```

Tableaux et pointeurs

Indices et tableaux

Si `tab` est un tableau d'entiers **`int`** alors `tab[i]` revient à :

- récupérer l'adresse de la première case du tableau : `p = &tab[0]` (ou simplement `p = tab`) ;
- calculer l'adresse du i^e élément : `p = p + sizeof(int)*i` ;
- déréférencer ce i^e élément : `*p`.

Cette syntaxe fonctionne avec un pointeur quelconque.

Exemple

```
int tab[5] = {1,2,3,4,5};
int * p = &tab[0]; // ou int * p = tab;
printf("tab[2]: %d, p[2]: %d \n", tab[2], p[2]);
```

produit l'affichage : `tab[2]: 3, p[2]: 3`

Allocation dynamique de tableau

Tableau et malloc

On peut allouer dynamiquement un tableau :

```
int * tab = malloc(sizeof(int)*100);
tab[0] = 10;
tab[1] = 20;
```

puis libérer la mémoire :

```
free(tab);
```

Tableaux à plusieurs dimensions

Définition

En C un tableau `tab` à deux dimensions est en fait un tableau de pointeurs : chaque case `tab[i]` est un tableau (donc un pointeur). Déclaration, initialisation et utilisation sont similaires.

Exemple

```
int tab[5][2] = {{2,1},{4,2},{1,1},{5,6},{2,1}};
printf("tab[0]: %p, tab[0][0]: %d, *tab[0]: %d\n",
        tab[0], tab[0][0], *tab[0]);
```

produit l'affichage :

```
  tab[0]: 0x7ffe, tab[0][0]: 2, *(tab[0]): 2
```

1 Allocation dynamique de mémoire

2 Structures

3 Tableaux

4 Chaînes de caractères

Chaines de caractères

Définition

Une chaîne de caractère en C est un tableau de **char** terminant par le caractère de fin de chaîne : `\0`. L'utilisation de guillemets doubles facilite l'initialisation d'un tel tableau.

Exemple

```
char tab[8] = "exemple";
tab[0] = 'E';
printf("%s\n", tab);
```

produit l'affichage : `Exemple`

Attention

Une chaîne de n caractères est en fait un tableau de $n + 1$ caractères !

Chaînes de caractères

Déclaration et initialisation

- `char s[8] = "bonjour";`
- `char s[] = "bonjour";`
  laisse l'ordinateur calculer la zone mémoire nécessaire ;
- `char * s = "bonjour";`
  laisse l'ordinateur calculer la zone mémoire nécessaire.

  Attention : ici s est un pointeur donc il est modifiable, et sans précaution on peut perdre l'accès à la zone mémoire initiale.

Chaînes de caractères

Bibliothèque

`#include` `<string.h>` pour :

- obtenir la longueur d'une chaîne : `strlen(s)`
- copier une chaîne dans une autre : `strcpy(dest, src)`
- concaténer deux chaînes : `strcat(dest, src)`
- comparer deux chaînes : `strcmp(s1,s2)`
- ...(cf. `man string.h`)