TP 5

Programmation en C (LC4)

Semaine du 25 février 2008

1 Jouons avec les pointeurs

Dans cette section, on convertit des pointeurs en `unsigned int`, pour voir un peu comment cela se passe dans les entrailles de la machine.

Exercice 1 Soit le code suivant :

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

int main(int argc,char** argv) {
  int *t=malloc(4*sizeof(int));
  printf("%u %u %u %u %u\n",sizeof(int),(unsigned int)t,
         (unsigned int)(t+1),(unsigned int)&t[1], (unsigned int) (&t[1]-t));
  exit(0);
}
```

Sachant que les entiers sont codés sur 4 octets, que pensez vous qu'il va afficher ? Faites l'expérience pour vérifier.

► **Correction**

L'affichage est le suivant :
- sizeof(int) : 4
- (unsigned int)t : adresse de la premiere case du tableau t
- (unsigned int)(t+1) : adresse de la premiere case du tableau t (4 octets de plus que celle d'au dessus)
- (unsigned int)&t[1] : adresse de t[1] (c'est la même qu'au dessus)
- (unsigned int) (&t[1]-t) : 1 adresse de t[1] - adresse de t[0]

Exercice 2 De même avec ce code :

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

struct ploum {char x; char y; int z; };
struct plam {int x; char y;};
int main(int argc,char** argv) {
  struct ploum a;
  printf("%u %u %u %u %u\n",sizeof(char),(unsigned int)&a,
         (unsigned int)&(a.x),(unsigned int)&(a.y), (unsigned int) &(a.z));
  printf("%u\n",sizeof(struct plam));
  exit(0);
}
```

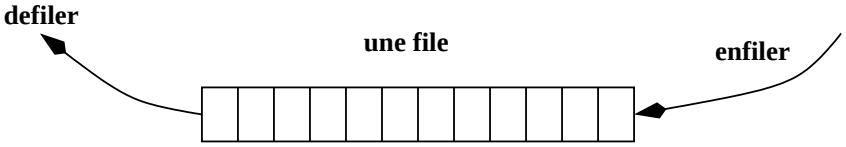
Ici, les résultats sont un peu plus surprenants.

► **Correction**

Cet exercice est surtout pour que les élèves comprennent qu'on ne peut allouer une structure "plam" de la façon suivant "2*sizof(int)+sizeof(char), parce qu'on peut pas être sûr que le compilateur fasse exactement ça derriere donc il faut bien mettre "sizeof(struct plam)". L'affichage est le suivant :
- sizeof(char) : 1
- (unsigned int)&a : adresse de la structure a
- (unsigned int)&(a.x) : adresse du champ x de a (la même que a car c'est son premier élément)
- (unsigned int)&(a.y) : adresse du champ y de a (= adresse du champ x de a plus 1)
- (unsigned int)&(a.z) : adresse du champ z de a (adresse > adresse du champ y de a mais qui est congue à 0 modulo 4). En fait, l'ordi préfère "aligner" les entiers sur des adresses mémoires qui tombent pile parce qu'après c'est plus simple pour lui pour accéder en mémoire.
- sizeof(struct plam) : 8 (à cause de ce qu'il y a au dessus)

2 Files

Une file est une structure de donnée dans laquelle les premiers éléments ajoutés sont les premiers à être récupérés (comme dans une file d'attente) :



Nous allons utiliser la structure `file_t` et le type `file_t` suivants :

```
struct file_s {
    size_t debut, fin, capacite;
    int *elements;
};
typedef struct file_s file_t;
```

Une file d'entiers sera implémentée (cf. figure) en stockant tous les éléments de la file dans le tableau pointé par le champ `elements`. Ce tableau aura pour taille la valeur indiquée par le champ `capacite`.

Exercice 3 Écrire les fonctions

```
file_t *alloue_file(size_t capacite);
void libere_file(file_t *file);
```

qui, respectivement,
- alloue et initialise une file pouvant contenir `capacité - 1` éléments (La fonction renverra NULL si l'allocation mémoire échoue ou si `capacité` vaut 0) : on initialisera les champs `debut` et `fin` à `0` et le tableau d'éléments devra contenir autant de cases que `capacité`
- libère l'espace mémoire occupé par une file et ses éléments.

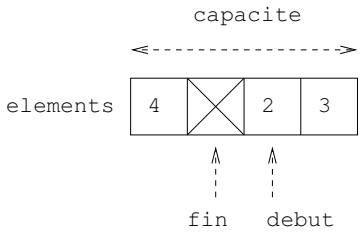
► **Correction**

```
file_t *alloue_file(size_t capacite) {
    file_t *file;
    if (!capacite)
        return NULL;
    if ((file = malloc(sizeof(struct file_s))) == NULL)
```

```
        return NULL;
    if ((file->elements = malloc(capacite * sizeof(int))) == NULL) {
        free(file);
        return NULL;
    }
    file->debut = file->fin = 0;
    file->capacite = capacite;
    return file;
}

void libere_file(file_t *file) {
    free(file->elements);
    free(file);
}
```

L'élément au début de la file se trouve dans la case d'indice `debut` et l'élément à la fin de la file se trouve dans la case précédant la case d'indice `fin`. La file est vide lorsque les indices de tableau `debut` et `fin` sont égaux. Le nombre d'éléments de la file est égal au nombre de cases utilisées du tableau, et on verra plus loin qu'après un certain nombre d'opérations sur la file, on peut avoir `fin < debut` (c'est le cas sur la figure suivante où la taille de la file est 3).



Exercice 4 Écrire les fonctions

```
int est_vide(file_t *file);
size_t taille(file_t *file);
```

qui, respectivement,

- indique si une file donnée en paramètre est vide
- indique le nombre d'éléments d'une la file

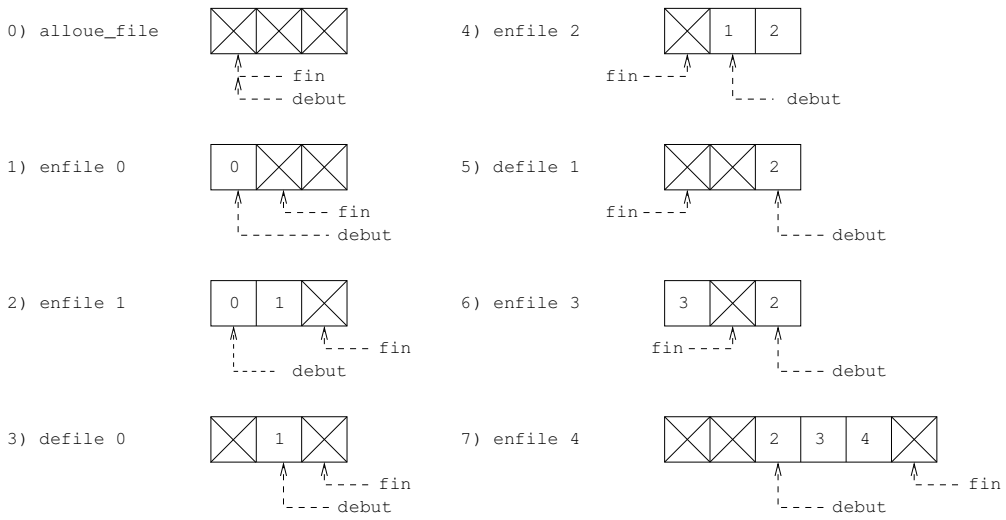
► **Correction**

```
int est_vide(file_t *file) {
    return (file->debut == file->fin);
}

size_t taille(file_t *file) {
    if (file->fin < file->debut)
        return ((file->capacite - file->debut) + file->fin);
    else /* file->fin >= file->debut */
        return (file->fin - file->debut);
}
```

Dans la figure suivante, on montre comment les indices `debut` et `fin` ainsi que le tableau pointé par `elements` sont modifiés par différentes opérations successives sur une file. Les indices `debut` et `fin` sont autorisés à dépasser la fin du tableau pour revenir au début. Lorsqu'on veut ajouter (enfiler) un élément à la fin de la file, on place l'élément dans la case indicée par `fin` et on décale l'indice de fin vers la droite : s"il ne reste qu'une case disponible, on redimensionne le tableau avec `realloc()` à deux fois sa taille initiale (et on déplace éventuellement les éléments de la file).

Lorsqu'on veut enlever (défiler) le début de la file, on décale l'indice de début vers la droite. Lors de ces opérations de décalage, on devra faire attention à toujours rester dans le tableau, quitte à passer de la dernière à la première case.



Exercice 5 Écrire les fonctions

```
int enfile(file_t *file, int n);
int defile(file_t *file);
```

qui, respectivement,
- enfile un entier `n` (la fonction renverra `n` si l'opération se termine avec succès et une valeur différente de `n` si l'éventuelle (ré)allocation mémoire a échouée)
- défile l'entier au début de la file et le place dans l'entier passé en argument par pointeur (la fonction renverra NULL si la file est vide).

► **Correction**

```
int enfile(file_t *file, int n) {
    size_t i;
    int *t;
    if (taille(file) == file->capacite - 1) {
        if ((t = realloc(file->elements,
                         2 * file->capacite * sizeof(int))) == NULL)
            return 0;
        file->elements = t;
        if (file->fin < file->debut) {
            for (i = 0; i < file->fin; i++)
                file->elements[file->capacite + i] = file->elements[i];
            file->fin += file->capacite;
        }
        file->capacite *= 2;
    }
    file->elements[file->fin] = n;
    file->fin++;
    if (file->fin == file->capacite)
        file->fin = 0;
    return 1;
}

int defile(file_t *file, int *n) {
```

```
    if (est_vide(file))
        return 0;
    *n = file->elements[file->debut];
    file->debut++;
    if (file->debut == file->capacite)
        file->debut = 0;
    return 1;
}
```