TD 04 Programmer en C (4)

A faire TD 4.

Table des matières

1 Allocation dynamique **2**

1.1 Objectif . . . 2

1.2 Allocation . . . 2

1.3 Libération . . . 3

1.4 Reallocation . . . 3

1.5 Exercices . . . 3

2 Les fichiers **3**

2.1 Ouverture . . . 3

2.2 Fermeture . . . 4

2.3 Entrées/Sorties formatées . . . 4

2.4 Exercices . . . 4

1 Allocation dynamique

Les variables que nous avons créées jusqu'ici étaient construites automatiquement lors de la phase de compilation. Elles ont une durée de vie limitée à la fonction ou méthode où elles sont déclarées. On parle ***d'allocation statique***.

Mais il est souvent nécessaire de créer des variables dont la durée de vie dépasse la portée locale et dont la taille n'est connue qu'à l'exécution. On parle ***d'allocation dynamique***.

1.1 Objectif

Disposant de la structure personne suivante

```
struct Personne {
    char *nom;
    char *prenom;
    struct Date naissance;
};
```

avec

```
struct Date {
    int jour;
    int mois;
    int annee;
};
```

on souhaite constuire un annuaire dont le nombre d'entrées est variable et non borné

```
struct Annuaire {
    int taille;
    struct Personne *tableau;
};
```

1.2 Allocation

On va utiliser la fonction `malloc` pour **allouer dynamiquement** (réserver) une zone mémoire.

La séquence suivante initialise un annuaire avec 20 personnes :

```
struct Annuaire annuaire;
annuaire.taille = 20;
annuaire.tableau = malloc( annuaire.taille *
                           sizeof(struct Personne) );
```

Le paramètre est le nombre d'octets à réserver : ici c'est le nombre de personnes multiplié par la taille (en nombre d'octets) d'une `struct Personne`.

Le résultat de `malloc` est un **pointeur non typé** (de type `void*`), qui est converti implicitement en `struct Personne *` lors de l'affectation. Il indique où se trouve le premier octet de la zone, qui correspond à l'adresse du premier élément du tableau.

Attention, l'allocation peut retourner `NULL` en cas d'échec (allocation impossible parce que toute la mémoire est occupée). En principe, il faut le vérifier.

1.3 Libération

Une zone qui a été allouée dynamiquement doit être libérée explicitement par un appel à `free` quand on n'en a plus besoin (avant de perdre la dernière référence à son adresse mémoire), sinon on aura une **fuite mémoire**.

```
free( annuaire.tableau );
annuaire.tableau = NULL;  // précaution
```

Un petite précaution a été prise pour ne pas risquer de libérer plusieurs fois la même zone mémoire. Les appels `free(NULL)` sont sans effet.

1.4 Reallocation

La fonction `realloc` libère un bloc de mémoire précédemment alloué, en réserve un nouveau de la taille demandée et copie le contenu de l'ancien bloc dans le nouveau.

- Dans le cas où la taille demandée est inférieure à celle du bloc d'origine, le contenu de celui-ci sera copié à hauteur de la nouvelle taille.
- À l'inverse, si la nouvelle taille est supérieure à l'ancienne, l'excédant n'est pas initialisé.

La fonction attends deux arguments :

- l'adresse d'un bloc précédemment alloué à l'aide de la fonction d'allocation,
- la taille du nouveau bloc à allouer.

Elle retourne l'adresse du nouveau bloc ou un pointeur NULL en cas d'erreur.

```
annuaire.tableau = realloc( annuaire.tableau,
                            30 * sizeof(struct Personne) );
```

1.5 Exercices

Ecrire les fonctions suivantes :

```
void afficher_annuaire( const struct Annuaire *ptr_annuaire );
void detruire_annuaire( struct Annuaire *ptr_annuaire );
int  ajouter_personne(  struct Annuaire *ptr_annuaire,
                        const struct Personne *ptr_nouveau);
```

2 Les fichiers

Un fichier du disque dur est associé, une fois ouvert, à une variable de type `FILE*`, pointeur sur une structure définie dans `<stdio.h>`. On ne veut pas connaitre le contenu de cette structure, il suffira de fournir ce pointeur aux fonctions de manipulation de fichier.

2.1 Ouverture

Pour ouvrir un fichier on utilise la fonction

```
FILE* fopen( const char* filename, const char* mode) ;
```

- `filename` est le nom du fichier, chemin éventuellement inclus
- `mode` est une chaîne de 2 ou 3 caractères qui indique le mode d'accès

— r (read) : lecture seulement. Le curseur est positionné au début du fichier. Le fichier doit déjà exister.
— w (write) : écriture seulement. Le curseur est positionné au début du fichier. Celui-ci est créé s'il n'existe pas, son ancien contenu est écrasé s'il existe.
— a (append) : écriture seulement. Le curseur est positionné à la fin du fichier. Celui-ci est créé s'il n'existe pas. C'est le mode à utiliser pour compléter un fichier existant.

Exemple

```
f = fopen( "donnees.txt", "rw" ) ;
```

2.2 Fermeture

Il est indispensable de fermer un fichier avant la fin du programme pour éviter la perte de données (en cas d'écriture).

```
fclose( f ) ;
```

2.3 Entrées/Sorties formatées

Les principales fonctions disponibles sont les suivantes :

```
int fprintf( FILE *f, const char* format, ... valeurs ... );
int fscanf ( FILE *f, const char* format, ... adresses ... );

char* fgets( char* s, int max, FILE *f ); // lecture d'une chaîne
char* fputs( char* s, FILE *f );          // écriture d'une chaîne

int fgetc( FILE *f );           // lecture d'un caractère
int fputc( FILE *f, char c );   // écriture d'un caractère

ssize_t getline ( char** ptr_line, size_t *n ,
                  FILE *f ); // lecture d'une ligne
```

— Pour la fonction `fgets`, max est la taille de la chaine `s` pré-allouée (en intégrant le caractère '0' qui sera ajouté à la fin). Le retour à la ligne 'n' est inclu dans la chaine.
— Pour la fonction `getline`, on peut fournir un pointeur sur une chaine `ptr_line` pré-allouée de taille `*n`. Cette fonction réalloue cette zone si nécessaire (avec la fonction `realloc`). Si `ptr_line` est un pointeur `NULL`, une nouvelle chaine sera allouée. Dans tous les cas, `ptr_line`et `n` seront mis à jour. Le retour à la ligne 'n' est inclu dans la chaine.

2.4 Exercices

Ecrire les fonctions suivantes pour charger et sauvegarder un annuaire sur disque :

```
int lire_fichier( struct Annuaire *ptr_annuaire, const char *nomfic );
int ecrire_fichier( const struct Annuaire *ptr_annuaire, const char *nomfic );
```