Feuille de travaux pratiques n° 3
Pointeurs, chaînes de caractères et caractères

1 Structures et Unions

1. Recopiez le fichier `struct.c`, compilez-le et exécutez-le (les adresses sont affichées en base décimale pour vous éviter des calculs en hexa ;-)). Pourquoi la taille de la structure est de 60 octets et non pas de 57 ? Pourquoi la variable v2 est-elle 64 octets "plus loin" que la variable v1 et non pas 60 ?

   Réponse : Cela illustre le fait que les compilateurs appliquent les restrictions d'alignement définies par les microprocesseurs cibles pour lesquels ils sont destinés. C'est ainsi que les compilateurs ajoutent des octets "d'alignement" dans les structures de données, les variables et le code machine du programme compilé, afin de ne pas violer les restrictions d'alignement définies par le microprocesseur cible.

2. Travail sur les heures (utilisation du type `struct`).

   Comme vous le savez tous, dans une minute il y a 60 secondes, dans une heure il y a 60 minutes et il y a 24 heures dans une journée.

   Dans le Monde, il y a deux façons de "dire" l'heure : le système 24 heures et le système 12 heures (avec am pour *ante meridiem* et pm pour *post meridiem* pour savoir dans quelle partie du jour on se trouve).

   Dans le fichier `heures.h`, se trouvent toutes les déclarations utiles pour traiter le type heure. Dans le fichier `heures.c`, se trouvent toutes les définitions des sous-programmes ainsi que la fonction principale qui permet notamment à tester.

   Vous devez écrire le fichier `heures.c` correspondant au fichier `heures.h`, c'est-à-dire vous devez définir toutes les fonctions qui ont été déclarées dans le fichier de déclarations (ou fichier d'en-têtes, *header files*). Définir une fonction revient à écrire son corps : il faut répéter tout son en-tête et écrire les instructions du corps entre des accolades.

   ```
   Heure mettreHeure (short h, short m, short s);
   /* déclaration de la fonction mettreHeure */
   Heure mettreHeure (short h, short m, short s) {
   /* définition de la fonction mettreHeure */
   ...
   }
   ```

   N'oubliez pas que le fichier devra donc contenir la ligne :

   ```
   #include "heures.h"
   ```

   Conseil : écrivez le corps de vos fonctions les unes après les autres, et testez-les au fur et à mesure.

2 Pointeurs et chaînes de caractères

Les pointeurs, ça se promène dans la mémoire...

1. Copiez le programme `pointeurs.c`. Compilez-le et exécutez-le. Comprenez ce qui se passe.
2. Faites la même chose avec `pointeursbis.c` et `pointeurster.c`.

2.1 Passage des paramètres

1. Copiez le programme `caracteres.c`. Compilez grâce à `make caracteres` et exécutez.

   Après l'appel à la procédure `majCar`, la valeur de la variable `car` n'a pas été modifiée : ce qui est tout à fait normal, car en C, le passage des paramètres se fait par valeur.

   Après l'appel à la fonction `majCarF`, la valeur de la variable `car` n'a toujours pas été modifiée : en fait, c'est la valeur renvoyée par la fonction qui est la majuscule du paramètre.

   Si on veut vraiment modifier la valeur d'une variable, il faut utiliser le passage des paramètres par référence : procédure `majCarP`. Le paramètre `c` est en fait un pointeur sur `char`, et dans la procédure on modifie `*c`.

   L'appel à la procédure `majCarPbis` n'a aucun effet sur la valeur de `car`, car la première instruction de cette procédure est de modifier `c` par un malloc : on a donc perdu la référence au paramètre effectif `car`.

2. Écrivez une procédure `ecrireChaine`, qui écrit sur la sortie standard le contenu d'une chaîne passée en paramètre, caractère par caractère. On supposera que cette chaîne suit la convention de C, c'est-à-dire que c'est un tableau de caractères se terminant par `'\0'`. Testez-la.

3. Écrivez une procédure `majChaine`, qui met en majuscules le contenu de la chaîne passée en paramètre. Testez-la.

4. Écrivez maintenant une fonction `majChaineF`, qui renvoie la chaîne en majuscules, sans modifier la chaîne passée en paramètre. Il y aura donc forcément une allocation. Testez-la, puis écrivez la procédure `majChaineP` correspondante : on ne peut donc plus renvoyer la chaîne, il faut la passer par référence... N'ayez pas peur des `**` ! Et si vous voulez, vous pouvez écrire sur la sortie standard des adresses grâce au format %p de la fonction `printf`.

2.2 Chaînes de caractères

1. Voici un fichier de déclarations `chaines.h` qui regroupe différents sous-programmes traitant le type chaîne. Vous devez écrire le fichier `chaines.c` correspondant (n'oubliez pas la fonction qui teste tout ça). Chaque sous-programme se charge de l'allocation de la chaîne résultat s'il y a lieu.

2. Écrivez les fonctions `myStrcat` et `myStrcpy` qui sont les répliques exactes des fonctions `strcat` et `strcpy` fournies par le fichier de déclarations `string.h`. Elles ne se préoccupent pas du tout de savoir si la chaîne destination a suffisamment de place pour faire la manipulation voulue.

```
char *strcat(char *dst, const char *src);
char *strcpy(char *dst, const char *src);
```