TP de Système n° 1 : Révisions de C

Page de TD et TP :

`http://didel.script.univ-paris-diderot.fr/claroline/course/index.php?cid=SYSTEME2011`

Exercice 1 : Pointeurs et tableaux

On représente des ensembles d'entiers par la structure suivante, où on suppose le tableau trié :

```
struct ensemble {
  unsigned int taille;
  int *tab;
};
```

1. Écrire les fonctions `allouer_ensemble` et `liberer_ensemble`.
2. Écrire une fonction `membre_ensemble` qui teste si un entier est dans un ensemble donné.
3. Écrire une fonction `saisir_ensemble` qui demande à l'utilisateur d'entrer un ensemble.
4. Ecrire une fonction d'affichage qui prend en argument un ensemble et en affiche les éléments avec des `*` (voir figure 1).

5				
0	2	4	5	8

`*␣*␣**␣␣*`

FIGURE 1 – affichage d'un ensemble

5. Écrire une fonction qui prend en argument deux ensembles et calcule leur union.

Exercice 2 : Listes chaînées circulaires

On considère le type `struct liste_chainee_circulaire` défini par :

```
struct liste_chainee_circulaire {
  int valeur;
  struct liste_chainee_circulaire *suivant;
};
```

Le champ `suivant` du dernier élément d'une liste de type `liste_chainee_circulaire` pointe sur le premier élément de la liste.

1. Ecrire une fonction qui prend en arguments une valeur entière et une liste chaînée circulaire et renvoie le nombre d'occurrences de la valeur dans la liste.

2. Ecrire une fonction qui prend en arguments deux listes chaînées circulaires et les concatène, sans création de nouvelle liste.
3. Ecrire une fonction qui prend en arguments deux listes chaînées circulaires et crée une nouvelle liste qui est une concaténation des deux précédentes.
4. Ecrire une fonction qui prend en arguments une valeur entière et une liste chaînée circulaire et supprime de cette dernière toutes les occurrences de la valeur transmise (on évitera de laisser traîner en mémoire des données auxquelles on n'a plus accès).

Exercice 3 : Un peu de débugage

Téléchager le fichier `demineur.c`. Ce fichier contient le programme d'un jeu de démineur avec des erreurs.

1. Copiez le fichier précédent dans votre répertoire et compilez-le avec l'option `-g`.
2. Dans un terminal, tapez la commande `ulimit -c unlimited` qui vous permet de créer un fichier `core` lorsqu'une erreur de segmentation se produit lors de l'exécution d'un programme.
3. De même qu'en `Java`, on peut obtenir des *backtraces* pour l'exécution d'un programme erroné en lançant `gdb` ou `ddd` avec comme arguments le nom de l'exécutable et le nom du fichier `core`, et en utilisant la commande `bt full` dans l'invite de `gdb`. En faisant des *backtraces* sur l'exécution erronée du programme, corrigez `demineur.c`.

Exercice 4 : Chaînes de caractères

On appelle *tautogramme* un poème dont chaque vers ne mentionne que des mots commençant pas la même lettre. Voici un exemple de tautogramme de deux vers : Les mots du premier vers commencent tous par "t", ceux du second commencent tous par "s".

```
triste, transi, tout terni, tout tremblant,
sombre, songeant, sans sûre soutenance.
```

Écrire une fonction

```
int tautogramme(char *poeme)
```

renvoyant `1` si la chaîne stockée dans `poeme` est un tautogramme et `0` sinon. On fera sur cette chaîne les hypothèses suivantes :

- chaque mot est formé de caractères minuscules non accentués, *i.e.* de valeurs de `char` comprises entre `'a'` et `'z'`.
- les autres caractères, quels qu'ils soient, ne font partie d'aucun mot.
- chaque vers est séparé du suivant par un ou plusieurs retours chariot. La chaîne de devra être lue qu'une fois au plus (aucun retour en arrière). La fonction devra renvoyer sa réponse le plus rapidement possible. Elle ne peut appeler aucune autre fonction (pas même `strlen`).

Attention aux cas limites : vers ne contenant aucun mot ou bien un seul mot, vers ne commençant pas par un mot, vers final non suivi d'un retour chariot, poème vide, etc..