ISIMA 1ère Année : Langage C – 28 janvier 2013

Les barèmes sont donnés à titre indicatif.

EXERCICE 2 (2 pts) à répondre sur une copie simple

Ecrire une fonction qui transforme une chaîne de caractères en un réel double précision.

Exemples :

```
"-32.475" → -32.475
"  32.475" → 32.475
"+32.475" → 32.475
```

N.B. Les fonctions suivantes peuvent être utiles :

`int isdigit(int c);` teste si le caractère `c` est un chiffre.
`int isspace(int c);` teste si le caractère `c` est un espace.

EXERCICE 3 (RAMASSE-MIETTES) (12 pts) à répondre sur une feuille double

Le but de ce problème est de créer un mécanisme que l'on appelle un ramasse-miettes (ou *garbage collector*) pour empêcher les fuites de mémoire d'un programme. Evidemment, nous n'allons nous intéresser qu'à une version extrêmement simple. L'idée est de garder en mémoire (dans une liste chaînée spécifique) toutes les allocations dynamiques qui ont été faites et à la demande de l'utilisateur (ou bien en fin de programme) de libérer toutes les ressources en mémoire.

a) Avec quel programme a-t-on généré cette sortie écran ? Qu'est ce que cela veut dire ? (1 pt)

```
==16156== HEAP SUMMARY:
==16156==     in use at exit: 6,159 bytes in 33 blocks
==16156==   total heap usage: 37 allocs, 4 frees, 6,287 bytes allocated
==16156==
==16156== LEAK SUMMARY:
==16156==    definitely lost: 0 bytes in 0 blocks
==16156==    indirectly lost: 0 bytes in 0 blocks
==16156==      possibly lost: 0 bytes in 0 blocks
==16156==    still reachable: 6,159 bytes in 33 blocks
==16156==         suppressed: 0 bytes in 0 blocks
==16156== Rerun with --leak-check=full to see details of leaked memory
==16156==
==16156== For counts of detected and suppressed errors, rerun with: -v
==16156== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

b) Toutes les allocations mémoire sont gardées dans une liste chaînée triée sur l'adresse mémoire de la zone allouée. Définir une cellule de liste qui contient l'adresse mémoire d'une zone allouée, sa taille (en octets) et un pointeur sur la cellule suivante. (0,5 pt)

c) Définir une variable globale `__gPoolBegin` qui est un pointeur sur la tête de la liste. Le pointeur est initialisé à `NULL` pour plus de commodité (cela signifie que la liste est vide). (0,5 pt)

d) Ecrire la fonction `mallocGC()` qui prend exactement les mêmes paramètres que la fonction standard `malloc()` et qui se comporte exactement comme elle et qui, en cas de succès pour une taille non nulle, ajoute un enregistrement dans la liste chaînée **à la bonne place**. Si la taille demandée est nulle ou l'allocation est impossible, la fonction renvoie `NULL`. (1,5 pts)

e) Ecrire la fonction `displayGC()` qui affiche sur la sortie standard le contenu du ramasse-miettes : ordre de l'élément, pointeur et taille. (1 pt)

f) Proposer une implémentation de la fonction `void GC()` qui vide complètement la liste chaînée des allocations et les allocations qui y sont mémorisées. (1 pt)

g) Ecrire une fonction `reallocGC()` qui se comporte presque comme la fonction `realloc()` d'origine : on ne demande pas de traiter le cas où la taille est nulle. Au cas où vous auriez oublié le fonctionnement de `realloc()`, voici un extrait de la page `man` : (2 pts)

```
void * realloc(void *ptr, size_t size);
```

The `realloc()` function tries to change the size of the allocation pointed to by `ptr` to `size`, and returns `ptr`. If there is not enough room to enlarge the memory allocation pointed to by `ptr`, `realloc()` creates a new allocation, copies as much of the old data pointed to by `ptr` as will fit to the new allocation, frees the old allocation, and returns a pointer to the allocated memory. If `ptr` is `NULL`, `realloc()` is identical to a call to `malloc()` for `size` bytes. If `size` is zero and `ptr` is not `NULL`, a new, minimum sized object is allocated and the original object is freed.

h) Ecrire une fonction `main()` qui crée deux tableaux dynamiques de réels de même taille donnée en ligne de commande, initialise les valeurs à $1/i^2$ pour l'un et à $1/i^3$ pour l'autre et affiche les sommes de ces vecteurs. Vous devrez utiliser les fonctions écrites précédemment. (1 pt)

i) Quelle notion permet de faire de la compilation "intelligente" (par exemple si un fichier est modifié ?). Note : `makefile` n'est pas la réponse. (0,5 pt)

j) Supposons que toutes les fonctions relatives au ramasse-miettes soient écrites dans les fichiers `GC.h` et `GC.c` et votre programme dans `prog.c`. Ecrire le fichier `makefile` correspondant même si c'est trivial. Donner un exemple d'utilisation. (1,5 pts)

k) Ajouter une règle pour le nettoyage des fichiers objets. Donner un exemple d'utilisation (1 pt)

l) Supposons que vous sachiez (vous devriez avec de la documentation) transformer votre travail du jour (le ramasse-miettes) en bibliothèque, comment écrire la ligne de compilation du programme à exécuter si la bibliothèque se trouve dans le répertoire courant et s'appelle `libGC.a` ? (0,5 pt)