ISIMA 1ère Année : Langage C – 7 janvier 2010

Durée : 2h. Documents autorisés : 1 feuille A4 rect-verso manuscrit.
Soigner la présentation et la lisibilité de la copie. Le crayon et la gomme sont acceptés, pas les ratures.
Les barèmes sont donnés à titre indicatif.

EXERCICE 1 (2,5 pts) :

- Réécrire le code suivant avec la structure de contrôle `switch` :

```
if ( a==1 || a==2 || a==3) b=10 ;
else {
      if (a==4) b=20 ;
      else b=0 ;
}
```

- Expliquer la signification de l'instruction

```
a += (a%2) ? 0 : 1 ;
```

- Qu'est-ce qu'un pointeur en langage C ? A quoi servent les pointeurs ? Peut-on se passer de pointeurs en C, pourquoi .

EXERCICE 2 (3 pts) :

Considérer le programme suivant :

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

#define N 10

void action(double * d1, double * d2)
{
      double * t = d2;
      d2         = d1;
      d1         = t;
}

int main(int argc, char** argv)
{
      double * a = (double *) malloc(N*sizeof(double));
      double * b = (double *) malloc(N*sizeof(double));
      int i;

      for(i=0;i< N; ++i) {
            a[i] = i;
            b[i] = 100 + i;
      }

      action(a, b);

      for (i=0; i<N; ++i)
            printf("%f %f\n", a[i], b[i]);

      return 0;
}
```

a) Que fait ce programme ? Donner la trace d'une exécution de ce programme.
b) Si on exécute la commande `valgrind` qui permet entre autres la vérification de la fuite de mémoire dessus, quel genre de message devrait-on obtenir ?

EXERCICE 3 (2 pts) :

Ecrire une fonction qui retourne la valeur entière décimale d'une chaîne de caractères représentant un nombre binaire. La taille du nombre binaire sera passé en paramètre.
(Exemple : 1010 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 10)

EXERCICE 4 (3 pts) :

Ecrire un programme qui duplique un fichier texte. Le nom du fichier à dupliquer sera passé en argument de ligne de commande. Le nom de la copie sera formé du nom de fichier auquel on ajoute `_bis` avant l'extension.

Exemple : `duplique exo1.c` génère le fichier `exo1_bis.c`

EXERCICES 5 (RAMASSE-MIETTES) (9,5 pts) :

Le but de ce problème est de créer un mécanisme pour empêcher les fuites de mémoire d'un programme que l'on appelle un ramasse-miettes (un *garbage collector*). Evidemment, nous n'allons nous intéresser qu'à une version extrêmement simple. L'idée est de garder en mémoire 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.

Nous allons stocker toutes les demandes de mémoire dans un tableau statique de taille `MAX_P` pouvant contenir n'importe quel type de pointeur.

a) Définir une constante symbolique `MAX_P` de valeur `50`. (0,5 pt)

b) Définir un tableau `gPool` de pointeurs génériques et de taille `MAX_P`. (0,5 pt)

c) Définir une variable entière `gPoolIndex` initialisée à `-1`. Cette variable contiendra l'indice du prochain élément à insérer dans le tableau `gPool`. (0,5 pt)
`gPool` et `gPoolIndex` sont deux variables globales comme leur nom l'indique.

d) Proposer une implémentation de la fonction `void creerGC()` qui initialise tous les éléments de `gPool` au pointeur nul et qui donne une bonne valeur à `gPoolIndex`. Attention, le traitement de `creerGC()` ne doit pas pouvoir être effectué plusieurs fois dans un programme. (1 pt)

e) Proposer l'implémentation de la fonction `void GC()` qui vide correctement toute la mémoire allouée dans le tableau `gPool`. Chaque élément devra être remis à nul au cas où cette fonction serait appelée plusieurs fois dans le programme. Cette opération n'est possible que si le ramasse-miettes a été correctement défini. (1 pt)

f) Proposer l'implémentation de la fonction `detruireGC()` qui détruit toutes les ressources utilisées par le ramasse-miettes. (1 pt)

g) Proposer un prototype et une implémentation de la fonction `allouer()` qui retourne un pointeur générique correspondant à la zone mémoire allouée dont la taille (entier non signé) est donnée en paramètre. Cette fonction renvoie le pointeur nul si le ramasse-miettes n'a pas été correctement défini, si la taille est égale à `0`, si le tableau `gPool` est plein et enfin si l'allocation est impossible. Il faudra afficher un message sur la sortie d'erreur standard si le pool de pointeurs est plein. (1,5 pt)

h) Proposer le prototype et une implémentation de la fonction `reallouer()` capable de modifier un pointeur passé en paramètre et d'allouer une nouvelle zone mémoire dont la taille est spécifiée en paramètre. Les conditions de fonctionnement sont similaires à la fonction `allouer()`. Attention, le pointeur ne doit pas être changé si l'allocation est impossible. Le pointeur passé en paramètre doit appartenir à `gPool`. (1,5 pt)

i) Créer une structure autoréférentielle `agenda` qui contient les champs `nom` (un tableau de 80 caractères) et `num` (un tableau de 12 caractères) et un champ `suivant` pour modéliser un carnet d'adresse par liste chaînée. (0,5 pt)

j) Ecrire une fonction `main()` pour créer une liste chaînée avec une tête réelle (un pointeur) et deux ou trois éléments. Cette fonction teste les fonctions du ramasse-miettes : la création, la destruction, l'allocation et la réallocation. On affichera la liste chaînée. (1,5 pt)