**TELECOM Lille1** **Mai 2011**

Nom : Prénom : N° : Durée : 2 heures

## QROC Module A31TTN - Structures de données

*Aucun document n'est autorisé, les réponses sont à donner sur la présente feuille.*

***Exercice 1 –*** *Récursivité*

On considère les deux fonctions *f*, *g* suivantes :

```
int f (long x, long y) {
   long a=1;
   if(x==0)    return 1;
   else if(x==1)   return y;
  else{
  for(int i=0;i<x;i++)
       a=a*y;
       return a;
     }
}
```

```
int g(long x,long y)
{
  if(x==0)
      return 1;
  else if(x==1)
     return y;
  else
     return (y*g(x-1,y));
}
```

**1)** Donnez leurs résultats retournés par ces fonctions pour **x=4** et **y=2**. Que pouvez-vous remarquer? Expliquez, en particulier, la trace d'exécution de la fonction *g* ?

**2)** Que font alors les fonctions f, g ci-dessus ? Donnez des noms significatifs à ces fonctions?

***Exercice 2 –*** *Listes chaînées*

*On considère la déclaration de la structure de données ainsi que les fonctions **X**, **Y** et **Z** suivantes :*

```
struct element
{
   int val;
   struct element *nxt;
};
 typedef element* llist;
```

```
llist X(llist liste, int valeur)
{
   /* …
   element* Element = malloc(sizeof(element));
    /* …
   Element->val = valeur;
    /* …
   Element->nxt = NULL;
    if(liste == NULL)
   {
      /* …
      return Element;
   }
   else
   {
      /* …

      element* temp=liste;
      while(temp->nxt != NULL)
      {
         temp = temp->nxt;
      }
      temp->nxt = Element;
      return liste;
   }
}
```

**1)** Illustrer à l'aide d'un exemple (et un schéma) ce que fait cette fonction ***X***. Mettez du commentaire devant (/*…) afin de rendre le code plus clair. Donnez un nom significatif.

```
int Y(llist liste, int valeur)
{
   int i = 0;
    /* …
   if(liste == NULL)
      return 0;
    /* …
   while((liste = rechercherElement(liste, valeur)) != NULL)
   {
      /* …
      liste = liste->nxt;
      i++;
   }
   /* …
   return i;
}
llist rechercherElement(llist liste, int valeur)
{
   element *tmp=liste;
   /* ….
   while(tmp != NULL)
   {
      if(tmp->val == valeur)
      {
         /* …
         return tmp;
      }
      tmp = tmp->nxt;
   }
   return NULL;
}
```

**2)** Commentez le code de la fonction ***Y*** en ajoutant du texte devant (/*…). Donnez un non significatif.

```
int Z(llist liste)
{
   /* …
   if(liste == NULL)
      return 0;

   /* …

   return Z(Z->nxt)+1;
}
```

**3)** Commentez le code de la fonction Z (devant /*…). Donnez alors un non significatif.

***Exercice 3*** *– Les arbres binaires*

On considère la suite de nombres suivante : 96, 99, 46, 64, 58, 20, 10, 63, 91, 8.

**1)** Partant d'un arbre binaire de recherche vide, on ajoute successivement les valeurs de la suite ci-dessus. Donnez en justifiant votre raisonnement l'arbre binaire résultat *(il ne s'agit pas de donner la version équilibrée de l'arbre)*.

**2)** Vérifier si l'arbre obtenu est équilibré, s'il n'est pas équilibré, équilibrez-le. Expliquez les opérations de rotation effectuées.

**3)** On considère la structure de données suivante pour représenter cet arbre :

```
typedef struct Arbre {
        int Noeud;
        struct Arbre * SAG;
        struct Arbre * SAD;
} Arbre;
```

On considère aussi la fonction suivante Mystere:

```
Mystere (Arbre * Racine) {
        if (Racine!=NULL)
        {
                Mystere (Racine->SAD);
                printf("%d ",Racine->Noeud);
                Mystere (Racine->SAG);
        }
}
```

**a)** Expliquez ce que fait cette fonction Mystere? Donnez alors un non significatif à cette fonction. Donnez le résultat de cette fonction sur l'arbre originale (non équilibré) obtenu dans 1).

b) Donnez le résultat de cette fonction sur l'arbre équilibré que vous avez obtenu dans 2). Comparer les deux listes obtenues dans a) et b). Expliquez l'intérêt d'équilibrage des arbres.

4) Proposez une fonction récursive qui permet de calculer le nombre de feuilles d'un arbre.

5) Proposez une fonction récursive qui calcule le produit des nœuds de l'arbre.

**Bonne chance.**