Boussaid Ismail

Janvier 2012

Université de Mostaganem 2009-2010
Département d'Informatique

Examen du Module ASD1
S03 Licence Informatique

Date : 18-02-2010
Durée : 1H40

Exercice 1 : (4 points) **Ecrire la solution de cet exercice sur la feuille N°3 fournie**

Soit le programme suivant :

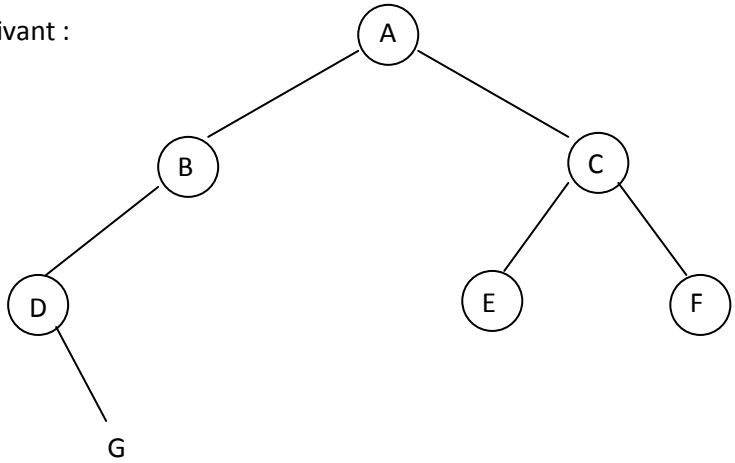
```
#include <stdio.h>

void  modifier  (int x, int *y)
{
        x = x + (*y);
        *y = (*y)*2 + x;
}

int  main ( )
{
        int  a=2, b=4;

        modifier (b, &a);
        printf ("\n %d\t%d", a, b);

        modifier (a, &b);
        printf ("\n %d\t%d", a, b);

        modifier (a, &a);
        printf ("\n %d\t%d", a, b);

        modifier (b, &b);
        printf ("\n %d\t%d", a, b);

        return 0;
}
```

Donner dans l'ordre (dans le tableau de la feuille N°3) les résultats affichés par ce programme :

Exercice 2 : (6 points) Les questions (1, 2, 3) sont indépendantes.

Ecrire la solution de cet exercice sur la feuille N°3 fournie

1) Soit l'arbre binaire suivant :



1) En se basant sur la représentation séquentielle d'un arbre binaire, dessiner le contenu du tableau utilisé pour cette représentation.

2) Donner le résultat du parcours infixé (LVR) et postfixé (LRV) de l'arbre binaire plus haut.

3) Proposer une fonction de parcours par niveau d'un arbre binaire de telle manière que les nœuds appartenant au même niveau soient visités de droite à gauche.

Note : pour la question N°3 : on suppose que le type « TreePointer » est déclaré (pointeur sur un nœud de l'arbre). De plus il est possible d'utiliser une pile ou une file. Il n'est pas nécessaire d'implémenter les opérations sur ces structures.

Exercice 3 : (10 points)

Dans cet exercice, les listes chaînées (simples ou doubles) manipulées sont des listes **d'entiers triées par ordre croissant**. Les listes manipulées sont de deux types :

- Liste simplement chaînée sans nœud de tête
- Liste à chaînage double circulaire avec nœud de tête

1) Ecrire les déclarations nécessaires (pour les deux types de liste)

Les questions suivantes sont indépendantes.

2) Ecrire une fonction « **Transformer (X)** » qui reçoit en paramètre une liste simplement chaînée X et construit une liste identique à chainage double, circulaire et avec nœud de tête. La liste résultat est évidemment triée par ordre croissant. La fonction retourne la liste construite.

3) Ecrire une fonction « **EliminerDoublons (Y)** » qui reçoit une liste à chaînage double circulaire avec nœud de tête Y et élimine les répétitions dans Y (si une valeur entière est répétée plusieurs fois dans la liste, la fonction garde une seule occurrence de cette valeur).

4) Ecrire une fonction « **Fusionner (X, Y)** » qui reçoit 2 listes à chainage double, circulaires et avec nœud de tête et les fusionne en une seule liste à double chainage, circulaire et avec nœud de tête. La fonction ne crée pas de nouveaux nœuds mais utilise les nœuds de X et Y pour former la liste résultat. De plus, si une valeur entière apparaît dans X et dans Y, la fonction garde une seule copie dans la liste résultat. La fonction retourne la liste résultat.

Solution

Exercice 1 :

	a	b
1er affichage	10	4
2nd affichage	10	22
3ème affichage	40	22
4ème affichage	40	88

Exercice 2 :

1) Représentation séquentielle de l'arbre binaire

A	B	C	D	--	E	F	--	G	--	--	--	--	--	--
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

2) Parcours LVR (infixé) : DGBAECHFI
Parcours LRV (postfixé) : GDBEHIFCA

3) Fonction de parcours par niveau (de droite à gauche) :

```
void  LevelOrder2 (TreePointer  p)
{
        TreePointer  r ;

        r = p ;

        while (r !=NULL)
        {
                printf (« %s », r->data) ;
                if (r->RightChild!=NULL)        AddQ (r->RightChild);
                if (r->LeftChild !=NULL)        AddQ (r->LeftChild);
                r = DeleteQ ();
        }
}
```

Exercice 3 :

a) Déclarations :

```
typedef  struct  Node ;                          /* type de chaque nœud de la liste */
typedef  struct  Node  *NodePointer ;            /* type d'un pointeur vers un nœud de la liste */
struct  Node  {
     int  value ;
     NodePointer  next ;
};
```

```
typedef  struct  Node2 ;                         /* type de chaque nœud de la liste */
typedef  struct  Node2  *Node2Pointer ;          /* type d'un pointeur vers un nœud de la liste */
struct  Node2  {
     int  value ;
     Node2Pointer  prev, next ;
};
```

b) Transformer une liste simplement chaînée en une liste à chaînage double, circulaire avec nœud de tête

```
Node2Pointer  Transformer (NodePointer X)
{
   NodePointer  p;          /* pour parcourir X */
   Node2Pointer Y, Z, W;

   /* Création du noeud de tête de la liste résultat */
   Y  =  (Node2Pointer)malloc (sizeof(struct Node2)) ;
   Z = Y;
   W = Y;

   p = X;

   while (p!=NULL)
   {
      W = (Node2Pointer)malloc (sizeof(struct Node2)) ;
      W->value = p->value;

      W->prev = Z;
      Z->next = W;
      Z = W;

      p = p->next;
   }

   Y->prev = W;
   W->next = Y;

   return Y;
}
```

c) Elimination des doublons

```
void EliminerDoublons (Node2Pointer X)
{
   Node2Pointer  p, r;

   p = X->next;

   while (p!=X)
   {
      r=p->next;
      while ((r!=X) && (r->value==p->value))
      {
         p->next = r->next;
         r->next->prev = p;

         free (r);

         r = p->next;
      }
      p = p->next;
   }
}
```

d) Fusionner de deux listes à double chaînage

```
Node2Pointer Fusionner (Node2Pointer X, Node2Pointer Y)
{
   Node2Pointer Z, p, q, r;

   /* creer le noeud de tête de la liste résultat */
   Z = (Node2Pointer)malloc (sizeof(struct Node2));
   Z->next = Z;
   Z->prev = Z;
   r = Z;

   /* Parcours de X et Y pour la fusion */

   p = X->next;
   q = Y->next;

   while ((p!=X)&&(q!=Y))
   {
      if (p->value < q->value)          {   r->next = p;
                                            p->prev = r;
                                            p = p->next;
                                        }
      else    if (p->value > q->value)    {   r->next = q;
                                            q->prev = r;
                                            q = q->next;
                                        }
            else                        {   r->next = p;
                                            p->prev = r;
                                            p = p->next;
                                            q = q->next;
```

```
                        }
        r = r->next;
    }

    if (p!=X)   {   r->next = p;
                    p->prev = r;
                    Z->prev = X->prev;
                    X->prev->next = Z;
                }
    else        {   r->next = q;
                    q->prev = r;
                    Z->prev = Y->prev;
                    Y->prev->next = Z;
                }
    return Z;
}
```

Toute suggestion est la bienvenue, n'hésitez pas à me contacter !

Facebook : https://www.facebook.com/mezyana.dz

Mail: Boussaid.ismail@gmail.com

Ainsi vous trouverez d'autre PDF sur : http://mezyana.wordpress.com/