Examen – Session principale

Matière : Atelier programmation II Enseignants : Majdi JRIBI et Chayma MEHDI Filière : MPI Nombre de pages : 6 pages	Semestre: Second semestre Date: 25 Mai 2016 Durée: 1h30 Documents : ***non autorisés***

Les réponses doivent être rédigées obligatoirement sur les feuilles de réponse (pages 5 et 6)

L'examen contient 6 pages. Seulement les pages 5 et 6 sont à rendre.

Exercice 1 : Liste chainée

Considérons les déclarations suivantes :

```
typedef struct Noeud
{
 int valeur;
struct Noeud * suivant;
}      Noeud;

typedef Noeud * liste;
```

Déterminer le rôle de chacune des fonctions suivantes (Répondre sur les feuilles de réponses)

Question 1

// Ici la liste prem est ordonnée par ordre croissant

```
liste traitement_1 (int x, liste prem)
{
liste pointeur;
if (prem = = NULL | | x <= prem → valeur )
{  pointeur = (liste ) malloc (sizeof(Noeud));
   pointeur → valeur = x;
   pointeur → suivant = prem;
   prem = pointeur;
}
else
   prem → suivant = traitement_1 ( x, prem→suivant);
return prem;
}
```

Question 2

```
liste traitement_2 (int x, liste * pprem)
{ liste x;
if ( *pprem == NULL) return(Null);
else
   if ( (*pprem) → valeur != x )
     return(traitement_2 ( x, &( (*pprem) → suivant) ));
   else
   { x = * pprem;
    * pprem = (*pprem) →suivant;
    return ( x );
   }
}
```

Question 3

```
liste traitement_3(liste la)
{
  liste temp, temp1, temp3;
  int x;
  for(temp=la ; temp!=NULL ; temp=temp→suivant)
  {
   temp3=temp;
   x=temp→valeur;
   for(temp1=temp→suivant ; temp1!=NULL ;
temp1=temp1→suivant)
   {
    if(x > temp1→valeur)
    {
      temp3=temp1;
      x=temp3→valeur;
    }
   }
   temp3→valeur=temp→valeur;
   temp→valeur=x;
  }
  return(la);
}
```

Question 4

```
void traitement_4 ( liste lst , liste * p_lst1 , liste * p_lst2 )
{
        if ( lst == NULL || lst → suivant == NULL )
        {
                * p_lst1 = lst ;
                * p_lst2 = NULL ;

        }
        else
        {
        liste lst1 = NULL , lst2 = NULL ;
        traitement_4 ( lst → suivant → suivant , & lst1 , &
lst2 ) ;
        lst → suivant → suivant = lst2 ;
        * p_lst2 = lst → suivant ;
        lst → suivant = lst1 ;
        * p_lst1 = lst ;
        }
}
```

Question 5

// Ici les listes lst1 et lst2 sont triées par ordre croissant

```
void traitement_5 ( liste lst1 , liste lst2 , liste * p_lst ) {
liste lst ;
if ( lst1 == NULL )
   * p_lst = lst2 ;
else if ( lst2 == NULL )
   * p_lst = lst1 ;
   else
   {
     if ( lst1 → valeur < lst2 → valeur) {
       traitement_5 ( lst1 → suivant , lst2 , & lst ) ;
       lst1 → suivant = lst ;
      * p_lst = lst1 ;
     }
     else {
       traitement_5 ( lst1 , lst2 → suivant , & lst ) ;
       lst2 → suivant = lst ;
      * p_lst = lst2 ;
     }
   }
}
```

Question 6

```
void traitement_6 ( liste * p_lst )
{
        liste lst1 , lst2 ;
        if ((* p_lst) != NULL && (* p_lst ) → suivant !=
NULL )
                {
        traitement_4 (* p_lst , & lst1 , & lst2 ) ;
        traitement_6 (& lst1 ) ;
        traitement_6 (& lst2 ) ;
        traitement_5 ( lst1 , lst2 , p_lst ) ;
                }
}
```

Exercice 2 : Arbre binaire et arbre binaire de recherche

Considérons le code écrit en langage C qui concerne des traitements sur les **arbres binaires et les arbres binaires de recherche.** Répondre aux questions qui figurent sur les feuilles de réponses.

```
typedef struct arb
   { int val;
     struct arb *fg;
     struct arb *fd;
    } arb;

typedef arb* arbre;
```

Question 1

```
arbre fonction_1(int v, arbre a, arbre b)
{
  arbre p= (arbre) malloc (sizeof(arb));
  p→val = v;
  p→fg= a;
  p→fd = b;
  return p;
}
```

Question 2

*// Ici l'arbre (*prac) est binaire de recherche*

```
void fonction_2 (int v, arbre *prac)
{
 if (*prac == NULL)
   *prac = fonction_1 (v, NULL, NULL);
 else if ( (*prac) →val) >= v)
        fonction_2 (v, & ((*prac)→ fg) );
      else
        fonction_2 (v, & ((*prac) → fd) );
}
```

Question 3

// Ici l'arbre ar est un arbre binaire et ar !=NULL

```
int fonction_3 ( arbre ar , int *min , int *max)
{
int i ;
*min = *max = ar→val ;
if ( ar→fg != NULL)
if ( !fonction_3 ( ar→fg , &i , max) || ! ( ar→val >
    *max) )
return 0 ;
if ( ar→fd != NULL)
if ( ! fonction_3 ( ar→fd , min , &i ) || ! ( ar→val
<= *min ) )
return 0 ;
return 1 ;
}
```

Question 4

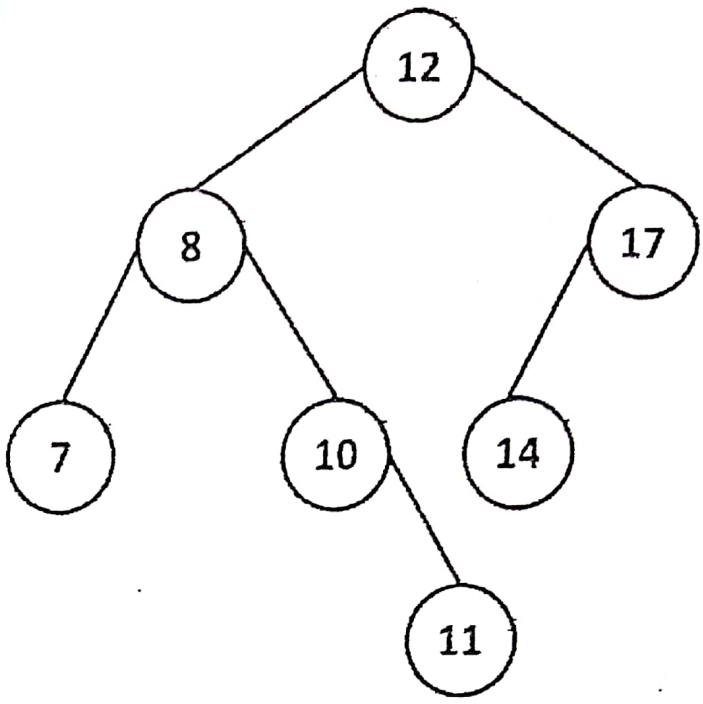
```
arbre fonction_4 (arbre A)
{
        int tmp;
        arbre y;
        if(A!=NULL && (A→fg)!=NULL)
        {
        y = A→fg;

        tmp = A→val;
        A→val = y→val;
        y→val = tmp;

        A→fg = y→fg;
        y→fg = y→fd;
        y→fd = A→fd;
        A→fd = y;

        }
 return(A);
}

/*****************************************/
```



Arbre A