

Examen – Session principale

Matière : Atelier programmation II
 Enseignant : M. Majdi JRIBI
 Filière : MPI
 Nombre de pages : 6 pages

Semestre: Second semestre
 Date: 28 Mai 2015
 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 : Arbre binaire et arbre binaire de recherche

Considérons les déclarations suivantes :

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

typedef arb* arbre;
```

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

Question 1

// Ici l'arbre ar est binaire de recherche

```
void fonction_1( arbre ar )
{
if( ar != NULL)
{
fonction_1( ar->fg );
if( ar->fg != NULL)
printf( " , " );
printf( "%d" , ar->val );
if( ar->fd != NULL)
printf( " , " );
fonction_1( ar->fd );
}
```

Question 2

// Ici l'arbre ar est un arbre binaire

```
void fonction_2( arbre ar )
{
if( ar == NULL)
printf( "_" );
else {
printf( "(" );
fonction_2( ar->fg );
printf( ",%d , " , ar->val );
fonction_2( ar->fd );
printf( ")" );
}
}
```

Question 3

// Ici les arbres ar1 et ar2 sont des arbres binaires

```
int fonction_3 ( arbre ar1 , arbre ar2 )
{
if( ar1 == NULL)
return ( ar2 != NULL );
else
{
if( ar2 == NULL)
return 1 ;
else
return ( ( ar1→val != ar2→val )
|| fonction_3 ( ar1→fg , ar2→fg )
|| fonction_3 ( ar1→fd , ar2→fd ) );
}
}
```

Question 4

// Ici l'arbre ar est binaire de recherche

```
arbre fonction_4 ( arbre ar , int va )
{
if( ar == NULL)
return NULL;
if( va == ar→val )
return ar ;
if( va < ar→val )
return fonction_4 ( ar→fg , va );
else
return fonction_4 ( ar→fd , va );
}
```

Question 5

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

```
int fonction_5 ( arbre ar , int *min , int *max)
{
int i ;
*min = *max = ar→val ;
if( ar→fg != NULL)
if( !fonction_5 ( ar→fg , &i , max ) || !( ar→val > *max ) )
return 0 ;
if( ar→fd != NULL)
if( !fonction_5 ( ar→fd , min , &i ) || !( ar→val <= *min ) )
return 0 ;
return 1 ;
}
```

Question 6

// Ici l'arbre ar est binaire de recherche

```
arbre fonction_6 ( arbre ar , int va )
arbre noeud = ar , * pere = &ar ;
arbre nouveau_noeud , *nouveau_pere ;
while ( noeud != NULL ) {
if( va == noeud→val )
break ;
if( va < noeud→val ) {
pere = &noeud→fg ;
noeud = noeud→fg ;
} else {
pere = &noeud→fd ;
noeud = noeud→fd ;
}
}
if( noeud != NULL ) {
if( noeud→fg == NULL ) {
if( noeud→fd == NULL ) {
*pere = NULL ;
free ( noeud ) ;
} else {
*pere = noeud→fd ;
free ( noeud ) ;
}
} else {
if( noeud→fd == NULL ) {
*pere = noeud→fg ;
free ( noeud ) ;
} else {
nouveau_noeud = noeud→fd ;
nouveau_pere = &noeud→fd ;
while ( nouveau_noeud != NULL )
if( nouveau_noeud→fg != NULL ) {
nouveau_pere = &nouveau_noeud→fg ;
nouveau_noeud = nouveau_noeud→fg ;
}
noeud→val = nouveau_noeud→val ;
*nouveau_pere = nouveau_noeud→fd ;
free ( nouveau_noeud ) ;
}
}
}
return ar ;
}
```

Exercice 2 : Liste chainée

Considérons le code écrit en langage C qui concerne des traitements sur les listes chainées.
Répondre aux questions qui figurent sur les feuilles de réponses

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

typedef struct element * Pelement;
typedef struct liste * FListe;

typedef struct element
{
    int x;
    Pelement suivant;
}Element;

typedef struct liste
{
    Pelement premier;
    Pelement courant;
    Pelement dernier;
}Liste;

/// Fonction Traitement_1
void Traitement_1(FListe L)
{
    L = (FListe) malloc ( sizeof(Liste));
    L->premier = (Pelement) malloc ( sizeof(Element));
    L->courant = (Pelement) malloc ( sizeof(Element));
    L->dernier = (Pelement) malloc ( sizeof(Element));
    L->premier = NULL;
    L->courant = NULL;
    L->dernier = NULL;
}

/// Fonction Traitement_2
void Traitement_2(FListe L, Pelement nouveau)
{
    Nouveau->suivant = L->premier;
    L->premier = nouveau;
    if( L->dernier ==NULL )
        L->dernier = nouveau;
}
```

```
// Fonction Traitement_3
```

```
void Traitement_3(FListe L, int n)
{
    printf("***** Traitement_3 *****\n");
    int i;
    Pelement Pel;
    for(i=1; i<=n; i++){
        Pel = (Pelement)malloc( sizeof (Element));
        Pel->x = i;
        Traitement_2(L, Pel);
    }
}
```

```
// Fonction Traitement_4
```

```
void Traitement_4(FListe L){
    L->courant = L->premier;
    printf("Liste = [ ");
    while (L->courant != NULL){
        printf("%d, ", L->courant->x);
        L->courant = L->courant->suivant;
    }
    printf(" ]\n");
}
```

```
// Fonction Traitement_5
```

```
void Traitement_5(FListe L){
    Pelement el = L->dernier;
    Pelement avDernier;
    L->courant = L->premier;
    while(L->courant->suivant->suivant !=NULL){
        L->courant = L->courant->suivant;
    }
    avDernier = L->courant;
    free(avDernier->suivant);
    avDernier->suivant = NULL;
    L->dernier = avDernier;
}
Liste l;
FListe maListe = &l;
```

```
// main()
```

```
main(){
    Traitement_1(maListe);
    Traitement_3(maListe, 5);
    Traitement_4(maListe);
    Traitement_5(maListe);
    Traitement_4(maListe);
}
```