Ecole préparatoire en Sciences et Techniques d'Oran
Module : Informatique 2
$2^{me}$ année, Semestre 4, 2011/2012 — Devoir surveillé
Mai 2012 — Durée : 01h30mn

# Partie cours :6.50 pts

## Exercice 1 : (les arbres) 3.50 pts

1. Soit l'arbre binaire suivant :

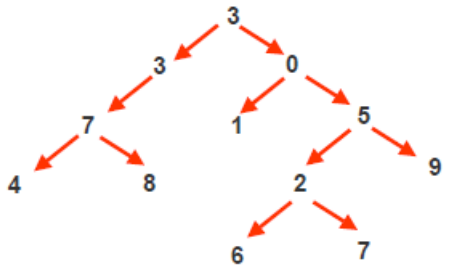


FIGURE 1 – Arbre binaire

   (a) donnez la structure générale en C++ d'un arbre binaire,
   (b) donner l'algorithme de parcours en profondeur préfixe,
   (c) appliquez cet algorithme sur l'arbre de la figure 1, et donnez l'ordre préfixe des noeuds (faire la trace de l'algorithme),

## Exercice 2 : (les graphes) 3.00pts

Soit le graphe suivant :

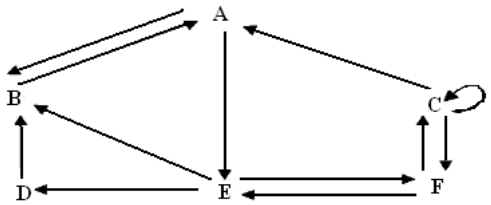


FIGURE 2 – Graphe orienté

1. donnez les arcs incidents à $E$ vers l'extérieur,
2. donnez la matrice d'incidence conrrespondant au graphe de la figure 2.

# Partie TD : 13.50 pts

## Exercice 1 : (la récursivité) 2.00pts

Pour un entier $m > 0$ donné, la suite de *Syracuse* est définie par :

$$\begin{cases} u_0 = m \\ u_{n+1} = \begin{cases} \frac{u_n}{2} & si \quad u_n \quad est \quad pair \\ u_n \times 3 + 1 & si \quad u_n \quad est \quad impair \end{cases} \end{cases}$$

donnez la fonction recursive dont le prototype *int syracuse(int n, int m)*, qui calcule la suite définie ci-dessus.

## Exercice 2 : (la complexité algorithmique) 4.25pts

Soit le programme en C++ suivant :

```
if (a>b)
   for(int i=5; i<=n-5;i++)
     for(int j=i-5; j<=i+5; j++)
        x=x+3;
else
   for(int i=1; i<=n;i++)
      for(int j=1; j<=n; j++)
          for(int k=1; k<=n; k++)
             x=x+b;
```

Etudiez le nombre d'additions réalisées par l'algorithme ci-dessus dans le meilleur cas, le pire cas, puis dans le cas moyen en supposant que les tests ont une probabilité de $\frac{1}{2}$ d'être vrai.

## Exercice 3 : (les listes chaînées) 7.25pts

Un rectangle est défini par sa couleur, sa hauteur et sa largeur :

1. donnez la structure *ListRec* d'une liste chaînée, dans laquelle on pourra ranger les informations de plusieurs rectangles,
2. donnez les fonctions :
   (a) *Ajouter*, pour ajouter un élément au niveau de l'en-tête de la liste,
   (b) *Supprimer*, pour supprimer un élément de la liste au niveau de l'en-tête,
   (c) *Afficher*, pour afficher la couleur et la surface de chaque rectangle rangé dans la liste,
3. en faisant appel, aux fonctions ci-dessus, donnez le programme principal qui permet de :
   – saisir et insérer les informations de 100 rectangles,
   – afficher la couleur et la surface de chaque rectangle de la liste,
   – supprimer les 10 derniers éléments insérés,

Ecole préparatoire en Sciences et Techniques d'Oran
Module : Informatique 2
$2^{me}$ année, Semestre 4, 2011/2012 — Corrigé type du devoir surveillé
Mai 2012 — Durée : 01h30mn

# Partie cours :6.50 pts

## Exercice 1 : (les arbres) 3.50 pts

1. la structure générale en C++ d'un arbre binaire : (0.25pts *5)

```
struct Noeud
         {
           TypeElt elt;
           Noeud * gauche;
           Noeud * droit;
         }
Noeud * T; // la racine de l'arbre
```

2. l'algorithme de parcours en profondeur préfixe : (0.25pts *5)

```
void parcours_profondeur_prefixe(Noeud*T)
{
  if (! est_vide(T))
  {
    traiter(T); // Traiter le noeud T
    parcours_profondeur_prefixe(gauche(T));
    parcours_profondeur_prefixe(droit(T));
  }
}
```

3. application de l'algorithme sur l'arbre : (1.00 pts)
   la fonction *parcours_profondeur_prefixe* traite les noeuds de l'arbre dans l'ordre : racine, gauche, droite, ce qui donne :

   3-3-7-4-8-0-1-5-2-6-7-9

## Exercice 2 : (les graphes) 3.00pts

1. les arcs incidents à $E$ vers l'extérieur : (0.5pts*3)

   (E,B),(E,D) et (E,F)

2. la matrice d'incidence conrrespondant au graphe : (0.25pts*6)

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 1 | 0 |
| B | 1 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | 0 | 1 | 0 | 0 | 1 |
| D | 0 | 1 | 0 | 0 | 0 | 0 |
| E | 0 | 1 | 0 | 1 | 0 | 1 |
| F | 0 | 0 | 1 | 0 | 1 | 0 |

# Partie TD : 13.50 pts

## Exercice 1 : (la récursivité) 2.00pts

```
int syracuse(int n, int m)
{
    if(n==0) return m;        (0.5 pt)
    else
```

```
    if (syracuse(n-1,m)\%2) return (syracuse(n-1,m)*3+1);      (0.5+0.5 pt)
    else return (syracuse(n-1,m)/2);       (0.5pt)
}
```

## Exercice 2 : (la complexité algorithmique) 4.25pts

si ($a\,b$) le nobre d'additions est : (1.00 pt)

$$C1(n) = \sum_{i=5}^{n-5} \sum_{j=i-5}^{i+5} 1 = \sum_{i=5}^{n-5} 11 = 11 * \sum_{i=5}^{n-5} = 11 * (\sum_{i=1}^{n-5} - \sum_{i=1}^{4}) = 11 * (\sum_{i=1}^{n-5} - 4) = 11 * (n - 5 - 4) = 11(n - 9).$$

sinon le nombre d'additions est : (1.00 pt)

$$C2(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} 1 = n.n.n = n^3$$

Ce qui donne :
– le meilleur cas : $Min_A(n) = 11(n - 9)$ (0.75 pt)
– le pire cas : $Max_A(n) = n^3$ (0.75 pt)
– le cas moyen : $Moy_A(n) = \frac{1}{2}.11.(n - 9) + \frac{1}{2}.n^3 = \frac{1}{2}.n^3 + \frac{11}{2}.n - \frac{9}{2}$ (0.75 pt)

## Exercice 3 : (les listes chaînées) 7.25pts

```
#include <iostream>
#include <string.h>
using namespace std;

/***La structure***/      (0.25pt *6)
struct  ListRec{
 string couleur;
 float hauteur;
 float largeur;
 ListRec *next;
};
struct ListRec* tete;

/*** La fonction Ajouter ***/       (0.25pt * 7)
ListRec * Ajouter(ListRec *& tete,string c,float h, float l)
{
    ListRec *lr = new ListRec;
    if (!lr) cout << "Plus de mémoire" << endl;
    else{
        lr->couleur = c;
        lr->hauteur = h;
        lr->largeur = l;
        lr->next = tete;
        tete = lr;
    }
}

/*** La fonction Suprimer ***/        (0.25pt * 4)
ListRec * Supprimer(ListRec *&tete)
{
 ListRec *lr = tete;
 tete = lr->next;
 delete lr;
}

/*** La fonction Afficher ***/      (0.25pt * 4)
void Afficher(ListRec * tete)
```

```
{
 ListRec *lr = tete;
 while(lr!=NULL)
 {
  cout << "Couleur = " << lr->couleur;
  cout << "Surface = " << lr->hauteur * lr->largeur;
  }
}

/*** Le programme principal ***/        (0.25pt * 8)
main( )
{
 tete = NULL;
 string c;
 float h,l;
 for(int i=0; i<100; i++)
 {
  cout << "Couleur = " ; cin >> c;
  cout << "Hauteur = "; cin >> h;
  cout << "Largeur = "; cin >> l;
  Ajouter(tete,c,h,l);
 }
 Afficher(tete);
 for(int i=0; i<10; i++)
 {
   Supprimer(tete);
 }
}
```