Correction Devoir semestriel **(S3)**
Module : **Informatique**

Exercice 1

1. La fonction **dupliquer :**

```
element * dupliquer(element * debut){
 element * tmp, * nouv;

 tmp = debut;
 while(tmp!=NULL){
  if(tmp->donnee % 2 == 0){
   nouv = (element *) malloc(sizeof(element));
   nouv->donnee = tmp-> donnee;
   nouv->suivant = tmp->suivant;
   tmp->suivant = nouv;
   tmp = nouv->suivant;
  }
  else
   tmp = tmp->suivant;
 }
 return debut;
}
```

2. La fonction **incluse :**

```
int incluse(element * A, element * B){
 element * tmp, * ptr;

 tmp = B;
 while(tmp!=NULL){
  ptr=A;
  while(ptr != NULL){
   if(ptr->donnee == tmp->donnee){
    ptr = ptr->suivant;
    tmp = tmp->suivant;
   }
   else
    break;
  }
  if(ptr == NULL)
   return 1;
  else
   if(tmp->donnee != A->donnee)
    tmp = tmp->suivant;
 }
 return 0;
}
```

3. La fonction **transferer :**

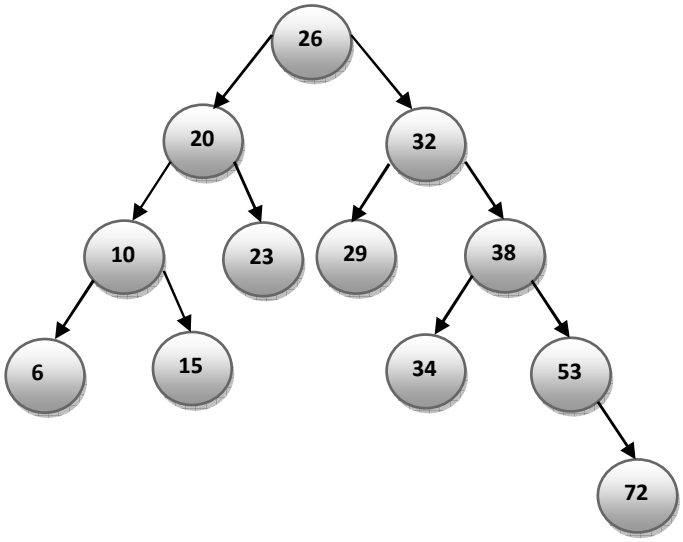
```
file transferer2(file * F){
 file Q;
 pile * P;
 int x;

 Q.tete = NULL;
 P = NULL;
 while(est_vide(*F)==0){
  x = defiler(F);
  if(x%2 == 0)
   P = empiler(P, x);
  else
   Q = enfiler(Q, x);
 }
 while(est_vide(P) == 0){
  x = depiler(&P);
  *F = enfiler(*F, x);
 }
 while(est_vide(Q) == 0){
  x = defiler(&Q);
  P = empiler(P, x);
 }
 while(est_vide(P) == 0){
  x = depiler(&P);
  Q = enfiler(Q, x);
 }
 return Q;
}
```

Exercice 2

Soit la liste des valeurs suivantes :

26 20 32 38 53 10 29 34 23 6 15 72

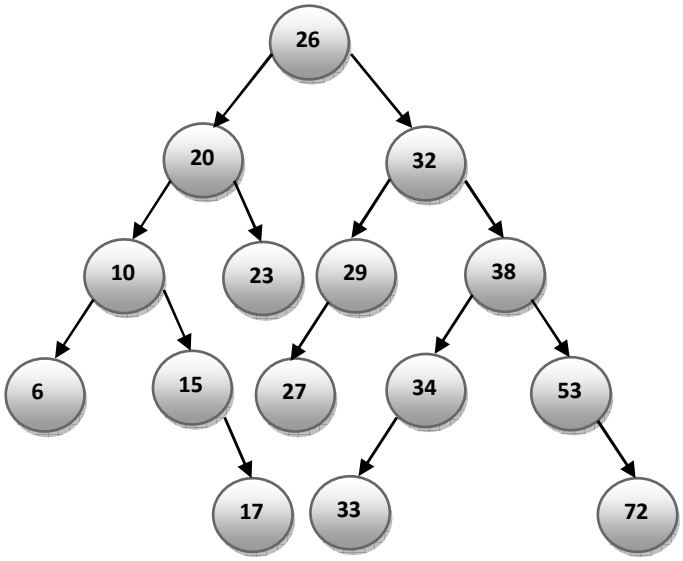
1. L'arbre binaire de recherche (ABR) correspondant à cette liste:



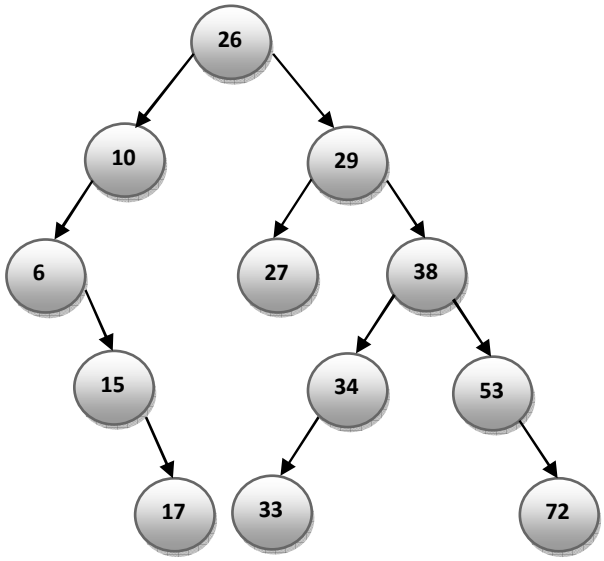
2. L'ordre des nœuds visités selon le parcours préfixé de l'arbre :

26 20 10 6 15 23 32 29 38 34 53 72

3. L'ABR obtenu après ajout successive des valeurs **17**, **27** et **33**:



4. L'ABR obtenu après suppression successive des valeurs **32, 23** et **20**:



5. Une fonction nommée **afficher_feuilles** qui permet d'afficher les points doubles d'un arbre binaire :

```
void afficher_feuilles(noeud * racine){
 if(racine != NULL){
  if(racine->gauche == NULL && racine->droit == NULL)
   printf("%d\t", racine->donnee);
  else{
   afficher_feuilles(racine->gauche);
   afficher_feuilles(racine->droit);
  }
 }
```

```
}
```

6. Une fonction **hauteur** qui permet de calculer le nombre de nœuds d'un arbre binaire :

```
int max(int a, int b){

 if (a > b)
  return a;
 else
  return b;
}

int hauteur(noeud * racine){

 if (racine != NULL){
  return 1 + max(hauteur(racine->gauche), hauteur(racine->droit));
 }
 else
  return -1;
}
```

7. Ecrire une fonction récursive **P_equilibre** qui permet de vérifier si un arbre binaire est parfaitement équilibré :

```
int AVL(noeud * racine){
 if(racine != NULL)
  if(abs(hauteur(racine->gauche)-hauteur(racine->droit))<=1)
   if(AVL(racine->gauche) == 1 && AVL(racine->droit) == 1)
    return 1;
   else
    return 0;
  else
   return 0;
 else
  return 1;
}
```

Exercice 3

1. La liste obtenue après exécution de la fonction **operation** contient les éléments suivants :

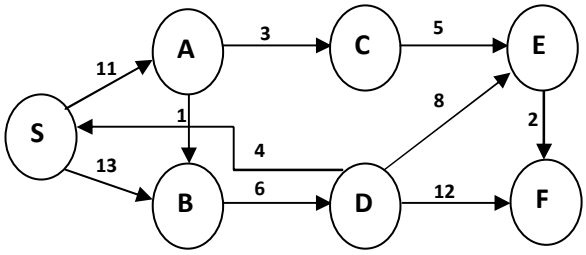
<12, 8, 11, 6, 5>

2. La pile obtenue après exécution de la fonction **mystere :**

P = [8, 6, 10, 11, 15], 15 est le sommet de la pile.

Exercice 4

1. Le graphe correspondant à la représentation :



2. L'ordre des sommets visités selon le parcours en profondeur du graphe :
   S A B D E F C

3. Le plus court chemin depuis S jusqu'à F :

Ensemble des sommets	Choix du sommet	S	A	B	C	D	E	F
		0	∞	∞	∞	∞	∞	∞
S A B C D E F	**S**	-	**11(S)**	13(S)	-	-	-	-
A B C D E F	**A**	-	-	12(A)	**14(A)**	-	-	-
B C D E F	**B**	-	-	-	-	18(B)	-	-
C D E F	**C**	-	-	-	-	-	**19(C)**	-
D E F	**D**	-	-	-	-	-	-	30(D)
E F	**E**	-	-	-	-	-	-	**21(E)**
F	**F**	-	-	-	-	-	-	-

Nous avons utilisé l'algorithme de Dijkstra puisque le graphe contient un circuit et des poids positifs sur les arcs, le plus court chemin du sommet **S** au sommet **F** est : **S A C E F**, sa longueur est de **21**.

4. Si la valeur de l'arc (B, D) devient **-6**, **nous utilisons l'algorithme de Ford BELLMAN pour l'obtention du plus court chemin.**