ENIG	Les ABRs	
	Matière : Programmation C_partie 2 Classe(s) : GCR1	Année universitaire : 2023-2024

Exercice 1

Ecrire une fonction qui prend en paramètre un nombre entier ainsi que deux arbres binaires et renvoie un arbre dont la racine contient le nombre passé en paramètre et les deux sous-arbres sont ceux donnés en paramètre.

Exercice 2

Ecrire une fonction qui affiche les valeurs des nœuds d'un ABR par ordre croissant.

Exercice 3

Ecrire une fonction qui calcule la somme des valeurs rangées dans un arbre binaire.

Exercice 4

Ecrire une fonction qui indique si un entier donné est dans un arbre binaire de recherche. La fonction doit retourner 1 si le nombre existe, 0 si non.

Exercice 5

Ecrire une fonction permettant de comparer deux arbres binaires de recherche. La fonction doit retourner 1 si les deux arbres sont différents, 0 si les deux arbres sont identiques.

CamScanner

Partie I:

-1-
```
typedef struct cellule
{int v;
 struct cellule * suiv, prec;
}cellule;
cellule * liste;
```

-2-
```
int debut(int a, liste l)   // adresse de la premiere cellule
{ if(l!=NULL)
  { if (l->v == a)
      return 1;
    return 0;
  }
}
int fin(int a, liste l)
{ liste aux = l;
  if(l!=NULL)
  { while(aux->suiv != NULL)
      aux = aux->suiv;
    if(aux->v == a)
      return 1;
    return 0;
  }
}
```

-3-
```
liste ajout_tete(int a, liste l)
{ liste new;
  new = (cellule *)malloc(sizeof(cellule));
  new->v = a;
  new->prec = NULL;
  new->suiv = l;
  l->Prec = new;
  return l;
}
```

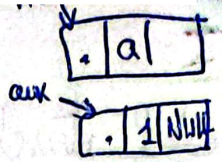
```
liste ajout_fin(int a, liste l)
{ liste new; liste aux = l;
  new = (cellule*) malloc(sizeof(cellule));
  new->v = a;
  while(aux->suiv != NULL)
    aux = aux->suiv;
  new->prec = aux;
  new->suiv = NULL;
  aux->suiv = new;
  return l;
}
```

-4-
```
void afficher(liste l)
{ liste aux = l;
  while(aux->suiv != NULL)
    aux = aux->suiv;
  // aux pointe sur la derniere cellule
  while(aux->prec != NULL)
  { printf("%d\n", aux->v);
    aux = aux->prec;
  }
}
```

Partie II:

```
liste ajout(int a; liste l)
{ liste aux = l, new;
  while(aux->v < a && aux->suiv != NULL)
    aux = aux->suiv;
  if(aux->suiv == NULL)
    return(ajout_fin(a, l));
```

```
else if (aux -> prec == NULL)
        return (ajout_debut(a, l));
    else {
    new = (cellule *) malloc (sizeof(cellule));
    new -> v = a;
    new -> suiv = aux;
    new -> prec = aux -> prec;
    (aux -> prec) -> suiv = new;
    aux -> prec = new;
    }
    return l;
}
```



-6-

```
liste supprimer(int a, liste l)
{ liste aux = l;
  while (aux -> v != a && aux != NULL)
      aux = aux -> suiv;
  if (aux -> v == a)
      if (aux -> suiv == NULL) // dernier elt
          (aux -> prec) -> suiv = NULL;
          free(aux);
      return l;
  if (aux -> prec == NULL)
          (aux -> suiv) -> prec = NULL;
          l = aux -> suiv;
          free(aux);
      } return l;   // un milieu
  (aux -> prec) -> suiv = aux -> suiv
  (aux -> suiv) -> prec = aux -> prec;
  free(aux);
```

* Construire les arbres binaires de recherche suivants (le 1er elt est la racine)

1/ 13 - 11 - 10 - 2 - 0 - 20 - 18 - 16 - 30 - 3 - 25

2/ sachant que chaque caractère est représenté par son code ASCII :

d - g - l - y - a - B - x - 3 - m - o - p - e

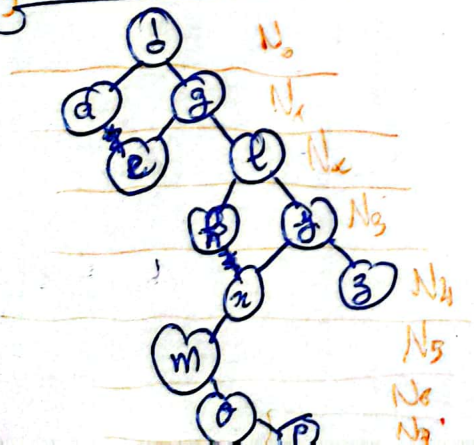
3/ 50 - 20 - 10 - 60 - 100 - 70 - 80 - 90 - 30 - 40

* Compléter le tableau

ABR	Parcours en préfixé (G racine D)	Parcours en infixé (G racine D)	Parcours en postfixé (G D racine)	Parcours en largeur
1/	13-11-10-2-0-3-5-20-18-16-30	0-2-3-5-10-11-13-16-18-20-30	0-5-3-2-10-11-16-18-20-13	13-11-20-10-18-30-2-16-0-3-5
3/	50-20-10-30-40-60-70-100-80-90	10-20-30-40-50-60-70-80-90-100	10-40-30-20-80-90-70-100-60-50	50-20-60-10-30-100-40-70-80-90
2/	d-a-g-e-l-B-y-x-3-m-o-p	a-g-e-d-l-B-m-o-3-x-y-p	a-e-g-B-p-l-o-m-3-y-x-d	d-a-g-e-l-B-y-x-3-m-o-p

y affiche rien sur le screen





Exo2 :

1) * Comment peut valider les expressions arithmétique binaire.

Sachant chaque expression est bien parenthésée

exp ((2+3)-2) → valide

exp (2+7)-(5+2)) → non valide

2) * soit le parcours infixé : 2-5-7-11-15-17-22-25 et 11 est la racine donner l'ABR

3) * soit le parcours postfixé : 0-6-5-11-17-20-21-18-15-10 et 10 est la racine, donner l'ABR

Rep :

1) liste chainé des parenthèses ouvrantes

principe de validation

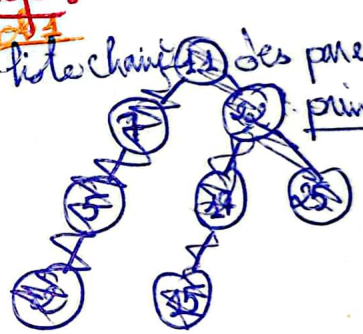
(→ ajouter un nœud)

) → supprimer un nœud → si liste déjà vide ⇒ non valide.

à la fin.

liste vide ⇒ valide

liste non vide ⇒ non valide



Arbre binaire des opérations

opérandes

racine : opération

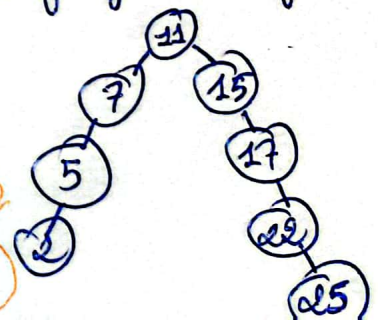
(: créer fils gauche / arcs

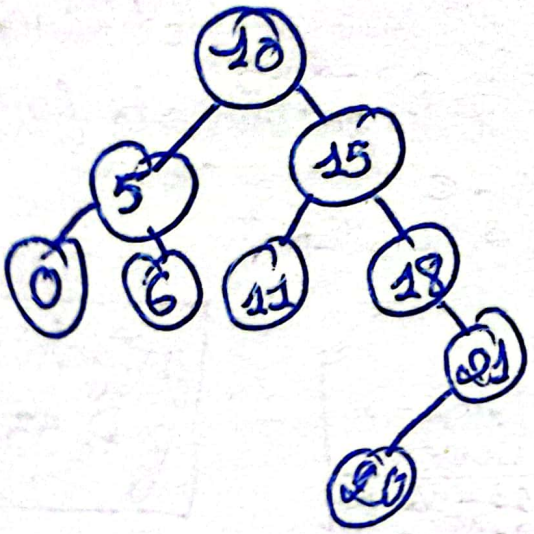
) : créer fils droit

les feuilles : les nombres



exp valide ssi pour chaque nœud ∃ fils gauche + fils droite





G
D
racine
postfixé

pour tous les noeuds
les fils et trouvé
de droit à gauche
par exp : 0. 6-5
0 et 6 sont les fils de