QROC A31 Structure de données et graphes

Juin 1996

Documents distribués et notes personnelles autorisés.

Il faut impérativement répondre sur la copie-sujet.

Faire d'abord au brouillon, aucun double de sujet ne pourra être distribué.

Récursivité (5 points)

Exercice 1

Soit la relation de récurrence de la fonction C(n, k) définie par :

$$\begin{cases} 1 \quad si \quad k = 0 \\ 1 \quad si \quad k = n \\ 0 \quad si \quad k > n \\ C(n-1,k-1) + C(n-1,k) \quad si\ 0 < k < n \end{cases}$$

a) Déduire la fonction récursive en C correspondante.

b) Donner une trace du déroulement de l'appel C(4,2), sous la forme d'un arbre.

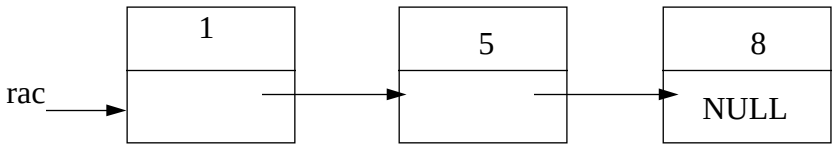
c) **Sans faire de calcul de complexité**, pouvez vous intuitivement en se basant sur la question précédente donner une idée sur le temps de calcul que peut demander cette fonction.

Liste chaînée

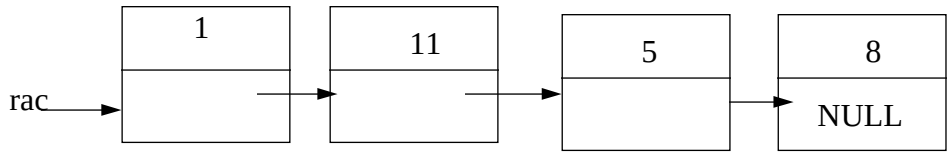
Liste chaînée (5 points)

Exercice 1

On considère la structure chaînée initiale suivante :



Où rac désigne le pointeur de début de cette liste. On veut créer la liste suivante :



Parmi les cinq portions de codes proposées une seule réalise cette transformation (répondre dans le tableau réponse). Toutes ont en commun les déclarations suivantes :

```
typedef struct list_noeud *LISTE;
typedef struct list_noeud {
        int valeur;
        struct list_noeud *suiv;
        };
LISTE rac;

/*********************Proposition  1*********************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
rac->suiv=t;
t->suiv=rac->suiv->suiv->suiv;
t->valeur=11;
/*********************Proposition  2 *************************/
LISTE t;
rac=rac->suiv->suiv;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac;
t->valeur=11;
rac=t;
/*********************proposition  3 *********************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac->suiv;
t->valeur=11;
rac->suiv=t;
/*****************proposition 4*************************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac->suiv;
t->valeur=11;
rac=t;
/************************proposition 5************************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac;
t->valeur=11;
rac=t;
```

Exercice 2

Dans cet exercice nous souhaitons être capable de représenter un nombre quelconque de polynômes différents tant qu'il y' a de la mémoire disponible. En général, nous voulons représenter le polynôme :

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + ... + a_0 x^{e_0}$$

où les a_i sont des coefficients entiers non nuls et les e_i sont des exposants entiers positifs tels que $e_{m-1} > e_{m-2} > .... > e_1 > e_0 >= 0$.

Nous souhaitons représenter les polynômes sous forme de listes simplement chaînées. Une structure adéquate pour représenter un polynôme peut être définie de la façon suivante :

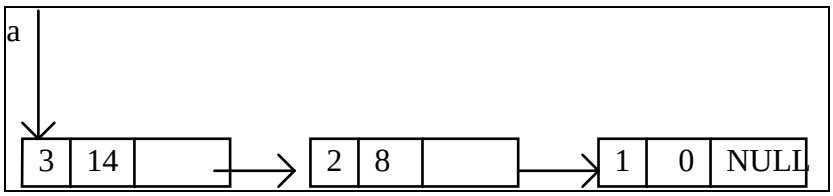
```
typedef struct noeud *poly;

    typedef struct noeud {
            int coefficient;
            int exposant;
            poly lien;
    };
```

où les champs coefficient et exposant désignent respectivement les coefficients et les exposants du polynôme.

Exemple :

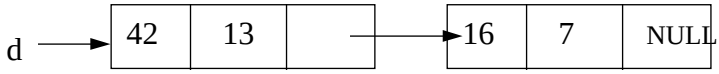
Soit le polynôme $a(x)=3x^{14}+2x^8+1$

Ce polynôme peut être représenté sous la forme suivante :



1) Ecrire une fonction **poly Derive(p poly)** qui permet de calculer la dérivée d'un polynôme.

Exemple : la dérivée de a(x) est $a'(x)=14*3*x^{13}+2*8*x^7$, d'où la liste chaînée suivante :

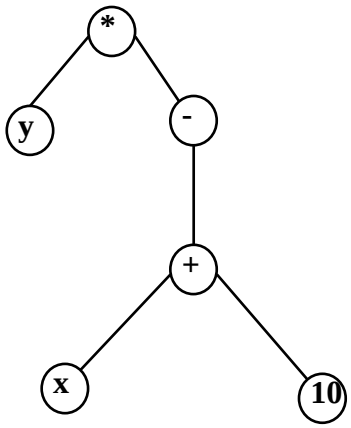


Arbres (5 points)

Exercice 1

Une expression arithmétique peut être facilement représentée par un arbre. En fait, les arbres d'expression spécifient l'association des opérandes d'une expression et de ses opérateurs d'une manière uniforme, sans qu'il soit nécessaire de se préoccuper du placement des parenthèses.

Exemple : l'expression suivante (y*(-(x+10))) peut être représentée par l'arbre



1) Pour chacune des expressions :

i) ((x+y)*(x+z)+5)

ii) $\Big(\big(x-y\big)*z+\big(\big(y-w\big)+2\big)\Big)*x$

Construisez l'arbre d'expression.

2) Une structure de données possible pour représenter les arbres définis précédemment peut être définie de la façon suivante (vue dans le cours):

```
typedef struct node *tree_pointer;
typedef struct node {
        char data;
        tree_pointer fils_gauche, fils_droit;
        };
```

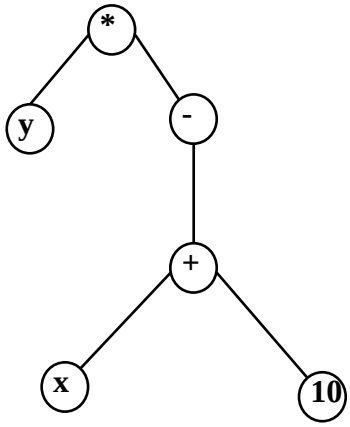
2.1) Donner la représentation chainée de l'expression i)

2.2) Dans cette partie nous ne considérons pas comment l'arbre a été créee mais nous suppons qu'il existe.

Soit la fonction mystere définie par :

```
void mystere(tree_pointer ptr){
if (ptr) {
        mystere(ptr->fils_gauche);
        mystere(ptr->fils_droit);
        printf("%c",ptr->data);
        }
}
```

En appliquant la fonction mystere sur l'arbre suivant :



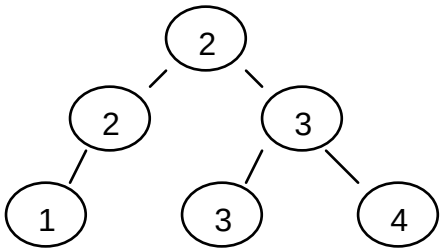
a) Donner le résultat de l'affichage de la fonction mystere(racine), où racine est la racine de l'arbre?

b) Donner la trace de l'exécution de la fonction mystere.

c) Que fait la fonction mystere ?

Exercice 2 :

Soit l'arbre suivante :



1) Proposez une structure de données qui permet de représenter cet arbre.

2) Ecrire la fonction qui retourne la somme des noeuds d'un arbre.

Exemple : pour l'arbre précédent la valeur retournée serait 15.

Conseil : La somme de noeuds d'un arbre vide est zéro; La somme de noeuds d'un arbre non vide est égale à la valeur du noeud plus la somme des neuds du sous arbe gauche et des noeuds du sous arbre droit.