Licence 2 SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 1 d'exercices

Exercice : Somme de polynômes creux

Dans chacune des trois représentations ***statiques*** suivantes, il est demandé d'écrire un programme en langage C (programme principal, plus éventuellement des sous-programmes ou fonctions) réalisant la somme S = P + Q de deux polynômes donnés P,Q.

1- Représentation implicite des degrés

Chaque polynôme P,Q et S de degré ≤ n est représenté dans un tableau unidimensionnel de taille n+1.

1-a Ecrire un sous-programme de saisie des données, où seuls les monômes non nuls seront saisis, et de remplissage automatique des tableaux P et Q.

1-b Ecrire un programme principal calculant la somme S.

2- Représentation non ordonnée et explicite des degrés

Seuls les monômes non nuls sont représentés dans deux tableaux unidimensionnels (coefficient, degré) de taille le nombre de monômes non nuls.

2-a Ecrire un sous-programme de saisie des données, où seuls les monômes non nuls seront saisis, et de remplissage automatique des tableaux P et Q.

2-b Ecrire un programme principal calculant la somme S, qui appelle deux fonctions réalisant la suppression et l'ajout d'un monôme dans la structure de donnée.

3- Représentation ordonnée et explicite des degrés

Seuls les monômes non nuls sont représentés dans l'ordre croissant des degrés dans deux tableaux unidimensionnels (coefficient, degré) de taille le nombre de monômes non nuls.

3-a Ecrire un sous-programme de saisie des données, où seuls les monômes non nuls seront saisis, et de remplissage automatique des tableaux P et Q (vérification de la contrainte sur l'ordre croissant des degrés).

3-b Ecrire un programme principal calculant la somme S, qui appelle deux fonctions réalisant la suppression et l'ajout d'un monôme dans la structure de donnée.

UVHC-ISTV

Licence 2SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 2 d'Informatique

Représentation statique de Liste chaînée dans un tableau

Soit une liste séquentielle chaînée de valeurs ordonnées. Une liste chaînée signifie que l'on représente le lien (ou chaînage) vers le suivant et que la notion de séquentiel correspond à la séquence des liens logiques. Cette liste va être représentée par un tableau d'éléments où chaque élément est composé de la valeur entière et de l'indice de son suivant .

typedef int Indice ; typedef struct { int val ; Indice suiv ; } Element ;
typedef struct { Element t [MAX] ; Indice premier ; } Liste ;

1) Donner deux exemples de représentation de la liste ci-dessous a) tout d'abord en occupant des positions consécutives dans le tableau à partir de la position 0 puis b) en occupant des positions non consécutives commençant à la position 6.

3 9 11 15 18

Ecrire la procédure de saisie de N valeurs ordonnées à ranger dans des positions consécutives de *z*

void saisie (int n, Liste * pz) ; Liste saisie2 (int n)

2) Dans le même tableau, on veut représenter deux listes : la liste précédente de valeurs et la liste des places disponibles dans le tableau : Illustrer ce que l'on pourrait avoir si la liste des valeurs était vide (et donc la liste des disponibles pleine) !
Modifier le type *Liste* pour tenir compte de ces deux listes en ajoutant un indice supplémentaire *dispo* .
Ecrire le sous-programme d'initialisation de la liste vide *init* ou *init2* :

void init (Liste * pz) ; Liste init2 ( ) ;

3) On veut ajouter la valeur *v* dans la liste *z* : pour cela on prend la première case disponible , donnée par *dispo* que l'on modifiera, puis on recherche entre quelles valeurs de la liste *z* caser *v*, et enfin on modifie les liens.
Sur l'exemple initial, illustrer l'ajout de la valeur 10. On écrira *ajout* ou *ajout2* :

void ajout (Liste * pz , int v) ; Liste ajout2 (Liste z , int v) ;

Ré écrire la procédure de saisie de N valeurs ordonnées à ranger dans des positions consécutives de *z : Liste* en utilisant *ajout* ou *ajout2* .

4) Recherche de l'existence d'une valeur

```
int existe (Liste x, int rech, Indice deb ) { if ( deb = = -1 | | rech < x. t [deb ] . val ) return 0 ;
                                              else if ( x. t[ deb] . val = = rech ) return 1 ;
                                                                                  else return existe ( x, rech, x . t [ deb ] . suiv );
                                                                                  }
```

Donner une solution itérative de la fonction *existe*.

FACULTATIF : Suppression d'une valeur (FACULTATIF)

a) Ecrire la fonction de suppression d'une valeur v qui ne se trouve pas dans la première position de la liste chaînée, la fonction renvoyant l'indice de la case libérée dans le tableau (on recherche la valeur puis on modifie les liens : le précédent sera relié au suivant) .

Indice supp1 (Liste z, int v) ;

b) Ecrire le sous-programme général de suppression de la valeur *v* de la liste *z* : on traitera le cas particulier de la valeur en tête de liste , puis les autres cas en utilisant *supp1*, et enfin on modifiera la liste des disponibles en y rajoutant la place libérée.

void suppr (Liste * pz, int v) ; Liste suppr2 (Liste z , int v) ;

Facultatif **Arbre Binaire de Recherche** (cf poly pages 134-135 et 167)

4.1Expliquer et justifier la structure Abr à partir d'un exemple

typedef struct { indice place; Indice fg, fd; } Noeud; typedef struct { Noeud a [MAX]; Indice racine; } Abr ;

4.2 Soit la fonction récursive qui recherche l'existence de la valeur val dans la liste x en utilisant l'ABR y :

int existe (Liste x, Abr y, Indice rac, int val) { if (rac == -1) return 0 ; else if (x.t[rac].num == val) return 1 ;
else if (val < x.t[rac].num) return existe (x, y, y.a [rac].fg, val); else return existe (x, y, y.a [rac].fg, val) ;

Donner une solution itérative

Licence 2 Info & Math semestre 4 && IUP GMI sem 2 : ALGO3

Solutions à la Fiche de TD 1 d'Informatique (semaine du 27/01)

I. Représentation statique de Liste chaînée dans un tableau (15 points)

Soit une liste séquentielle chaînée de valeurs ordonnées. Une liste chaînée signifie que l'on représente le lien (ou chaînage) vers le suivant et que la notion de séquentiel correspond à la séquence des liens logiques. Cette liste va être représentée par un tableau d'éléments où chaque élément est composé de la valeur entière et de l'indice de son suivant .

typedef int Indice ; typedef struct { int val ; Indice suiv ; } Element ; typedef struct { Element t [MAX] ; Indice premier ; } Liste ;

1) Donner deux exemples de représentation de la liste ci-dessous a) tout d'abord en occupant des positions consécutives dans le tableau à partir de la position premier=0 puis b) en occupant des positions non consécutives commençant à la position premier=6.

a) 3 1 9 2 11 3 15 4 18 –1 b) 9 1 11 5 18 -1 15 3 3 0

Ecrire la procédure de saisie de N valeurs ordonnées à ranger dans des positions consécutives de *z : Liste* .

Liste saisie (int n) {int i ; Liste z ; z . premier = 0 ; for (i = 0 ; i < n ; i ++)
{scanf («%d »,&z .t [i].val) ; z .t [i].suiv = i+1 ; } z .t [n-1]. suiv = - 1 ;
return z ; }

2) Dans le même tableau, on veut représenter deux listes : la liste précédente de valeurs et la liste des places disponibles dans le tableau : Illustrer ce que l'on pourrait avoir si la liste des valeurs était vide (et donc la liste des disponibles pleine) !

? 1 ? 2 ? 3 ? 4 ? 5 ? 6 ? 7 … premier = -1

Modifier le type *Liste* pour tenir compte de ces deux listes en ajoutant un indice supplémentaire *dispo* .

typedef struct { Element t [MAX] ; Indice premier , dispo; } Liste ;

Ecrire le sous-programme d'initialisation de la liste vide *init* ou *init2* : ***void init (Liste * pz) ; Liste init2 ( ) ;***

Liste init2 ( ) { Liste z ; int i ; z . premier = -1 ; z . dispo = 0 ; z . t [MAX - 1] .suiv = -1 ;
for (i = 0 ; i < MAX-1 ; i++) z . t [i] .suiv = i + 1 ; return z ; }
void init (Liste * pz) { int i ; pz -> premier = -1; pz -> dispo = 0; pz -> t [MAX - 1] .suiv = -1;
for (i = 0 ; i < MAX-1 ; i ++) pz -> t [i] .suiv = i + 1 ; }

3) On veut ajouter la valeur *v* dans la liste *z* : pour cela on prend la première case disponible , donnée par *dispo* que l'on modifiera, puis on recherche entre quelles valeurs de la liste *z* caser *v*, et enfin on modifie les liens. Sur l'exemple initial, illustrer l'ajout de la valeur 10.

a) 3 1 9 **5** 11 3 15 4 18 –1 **10 3** b) 9 **2** 11 5 **10 1** 18 -1 15 3 3 0

On écrira *ajout* ou *ajout2* : ***void ajout (Liste * pz , int v) ; Liste ajout2 (Liste z , int v) ;***

void ajout (Liste * pz , int * tete, int v) { int k = pz->dispo ; if (*tete ==-1 || v<=pz->t[*tete] . val)
{ pz ->dispo= pz->t[k] .suiv ; pz ->t[k]. val=v ; pz ->t[k]. suiv = tete ; *tete =k; }
else ajout (pz , & pz -> t [*tete] . suiv, v) ; }
Liste ajout2 (Liste z , int * tete, int v) { int k = z . dispo ; if (*tete == -1 || v <= z . t[*tete] . val)
{ z . dispo = z . t[k] .suiv ; z . t[k]. val=v ; z . t[k]. suiv = tete ; *tete =k; return z ; }
else return ajout (z , & z . t [*tete] . suiv, v) ; }

Ré écrire la procédure de saisie de N valeurs ordonnées à ranger dans des positions consécutives de *z : Liste* en utilisant *ajout*

Liste saisie (int n) {int i , v ; Liste z ; z . premier = -1 ; for (i = 0 ; i < n ; i ++)
{scanf («%d »,& v) ; ajout (& z ,& z . premier, v) ; } return z ; }

4) Recherche de l'existence d'une valeur

int existe (Liste x, int rech, Indice deb) { if (deb = = -1 | | rech < x. t [deb] . val) return 0 ;
else if (x. t[deb] . val = = rech) return 1 ; else return existe (x, rech, x . t [deb] . suiv); }

Donner une solution itérative de la fonction *existe*.

int existe (Liste x, int rech) { Indice deb= x . premier; while (deb != -1 && rech>= x .t[deb] .val)
deb= x . t [deb] . suiv; if (deb == -1) return 0 ; else return 1 ; }

Calculer la complexité en nombre de comparaisons. **Au mieux 2+1 comparaisons; au pire 2*N +1 +1=2*N+2 = θ(N)**

FACULTATIF Suppression d'une valeur

a) Ecrire la fonction de suppression d'une valeur v qui ne se trouve pas dans la première position de la liste chaînée, la fonction renvoyant l'indice de la case libérée dans le tableau (on recherche la valeur puis on modifie les liens: le précédent sera relié au suivant). ***Indice supp1 (Liste z, int v)***

Indice supp1 (Liste z, int *deb, int v) { int k=*deb; if (k < 0 || v < z . t[k] . val) return –1 ;

if (z .t[k] .val== v) {*deb= z .t[k].suiv; return k ;} else return supp1 (z, & z . t [*deb] .suiv , v); }

Indice supp2 (Liste z, int v) { int deb, prec = z . premier; while (deb >= 0 && v > z . t[deb] . val)

{ prec = deb ; deb= z . t[deb].suiv; } if (deb < 0 || v < z . t[deb] . val) return –1 ;

else { z . t [prec] . suiv = z . t [deb] .suiv; return deb; } }

a) Quelle est la **complexité** en nombre de comparaisons et en nombre de modifications d'éléments.

Nb de comparaison min = 1 et max = n ; Nb de modifications = 1

b) Ecrire le sous-programme général de suppression de la valeur *v* de la liste *z* : on traitera le cas particulier de la valeur en tête de liste , puis les autres cas en utilisant *supp1*, et enfin on modifiera la liste des disponibles en y rajoutant la place libérée. ***Liste suppr2 (Liste z , int v) ;***

void suppr (Liste * pz, int v) { int k ; if (pz -> premier != -1)
{ if (pz -> t[pz -> premier] . val != v) k =supp2 (*pz , v) ;
else { k = pz -> premier ; pz -> premier = pz -> t[k] . suiv ; }
pz -> t [k] .suiv = pz -> dispo ; pz -> dispo = k ; } }

UVHC-ISTV

Licence 2SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 2 d'Informatique

Exercice 1 : **Pointeurs**

Soit la séquence de code suivante:

```
typedef char* ptch ;   ptch x , y ;
/*1*/    x=(char *) malloc (sizeof (char) );
/*2*/    *x= 'a';
/*3*/    y= *x;
/*4*/    *x= 'b';
printf ("  caractere 1 = %c, caractere 2 =%c" , *x, *y);
```

1. Cette séquence comporte une erreur à la compilation: laquelle ? Corrigez la. Quelle est le message affiché ?

2. On remplace l'instruction 3 par l'instruction: `*y = *x ;`

Cette séquence entraîne une erreur à l'exécution: laquelle? Corrigez la. Quelle est le message affiché ?

3. Réécrire la même séquence avec des valeurs de type réel.

Exercice 2

Expliquer le déroulement de ce programme instruction par instruction

```
void main ( void)
{ int n1 = 1, *p = &n1, *q, ** pp3 ,  n2 = 0;
  char  s [8],  *ch = «licence», *c;
  printf («%d  %d  %d  %d  %d \n», n1, &n1, p,*p, &p);
  q= p;
  printf (« %d  %d %d \n», q,*q, &q);
  q= &n2;   n1=*q - 3;
  printf ( « %d  %d  %d \n», n1, n2, *q);
  p = q;
  printf («%d %d %d  %d \n», n1, n2, *p, *q);
  pp3 = &q;
  printf («%d  %d %d %d \n , &pp3, pp3, *pp3, ** pp3);
  strcpy ( s, ch );    *s= 'm';
  printf (« %d  %s  %s \n», s, s, ch);
  printf («%c %c \n», s[3], ch[6]);
  for ( c = s ; *c != '\0' ; c ++ )  printf  (« %c   », *c) ;
  while ( c != s) printf  (« %c \n  », *c - - ) ;
}
```

Exercice 2 Expliquer le déroulement de ce programme :

```
#define  MAX  10
void  main ( void )
{       int a [ MAX ], *b, *c , i = 0;
        char  d [ MAX ], *e , *f, g [ MAX ] ;
        while ( i < MAX )       scanf ( « %d », &a [ i  ++ ] ) ;
        while ( i > 0 )   printf ( « %d », a [  - - i ] ) ;
        printf ( « \n ») ;          c = a ;
        while ( ++ i  < MAX ) printf ( « %d     », a [ i ]) ;
        printf ( « \n ») ;          *b = a [ 0] ;
        while ( i - -   > 0 )       printf ( « %d     », * c ++ ) ;
        while (  i  < MAX )     *b ++ = a [ i ++] ;
        scanf (« %s », d) ;  scanf (« %s », e) ;
        strcpy ( g, d) ;   strcpy (f, d) ;
}
```

Exercice 3 : commenter

```
typedef struct e { int val ; struct e * suiv ; } Elem ;
Elem x , *p, * var;
x . val = 2 ;
p = ( Elem *) malloc (sizeof (Elem) ) ;
(*p).val = 5 ;    (*p).suiv = ( Elem ) malloc (sizeof (Elem) ) ;
var  = p ;
(*p).(*suiv).val = 8 ;   p = p -> suiv ; p -> suiv = NULL ;
```

UVHC-ISTV

Licence 2 Info semestre 4 : ALGO et SD de base
Fiche de TD 1 d'Informatique

Exercice 1 : Saisie d'une liste chaînée

typedef struct a { float val ; struct a * suiv ; } Element ;
typedef Element * Lc ;
Lc saisie1 ( ) { Lc p = (Lc) malloc (sizeof (Element)) ; printf (« entrer un réel : ») ;
scanf (« %f », & p->val) ; p- >suiv = NULL ; return p ; }

On peut créer une liste chaînée à l'endroit (ajout en fin de liste) ou à l'envers (ajout en tête). La solution proposée peut être itérative ou récursive. La solution peut être une fonction renvoyant un pointeur ou une procédure avec un paramètre résultat.

1. Ecrire le programme principal qui va tester la fonction saisie1.
2. Ecrire un sous-programme ***saisie2*** qui saisit 2 éléments en utilisant ***saisie1***. Réécrire ***saisie2bis*** sans utiliser ***saisie1.***
3. La saisie à l'envers correspond au fait que l'on ajoute un élément en tête de la liste et donc le fait de répéter ceci amène à saisir les valeurs dans l'ordre inverse de leur future position dans la liste (d'où le nom à l'envers).
   3.1 Ecrire une fonction permettant d'ajouter la valeur x en tête de la liste *prem* .
   Lc ajout_tete (float x, Lc prem)
   3.2 Transformer la fonction précédente en une procédure permettant d'ajouter la valeur x en tête de la liste *prem* .
   void ajout_tete2 (float x, Lc * pprem)
   3.3 Ecrire une fonction itérative *saisie_envers* qui répète *nb* fois la lecture d'un réel et son ajout en tête en utilisant la fonction ou la procédure précédente et tester la dans le programme principal.
   Lc saisie_envers1 (int nb)
   3.4 Ecrire une procédure itérative qui fait les mêmes opérations qu'en 3.3 et tester la dans le programme princpal :
   void saisie_envers2 (int nb, Lc * p)
   3.5 Ecrire maintenant un sous-programme récursif et tester le dans le programme principal.
   void saisie_envers3 (int nb, Lc * p*)
4. La saisie à l'endroit correspond au fait que l'on ajoute un élément en queue de la liste et donc le fait de répéter ceci amène à saisir les valeurs dans l'ordre exact de leur future position dans la liste (d'où le nom à l'endroit).
   4.1 Ecrire une fonction récursive *saisie_endroit* qui saisit *nb* valeurs :
   Lc saisie_endroit (int nb)
   4.2 Transformer la fonction récursive précédente en une procédure *saisie_endroit2* qui fait les mêmes opérations :
   void saisie_endroit2 (int nb, Lc * p)
   4.3 Ecrire une fonction itérative *saisie_endroit3* qui fait les mêmes opérations en créant et lisant la tête, enfin répétant *nb-1* fois la lecture et l'ajout en queue de liste. Pour cela on utilisera 2 variables locales de type Lc : une qui gardera la tête et l'autre qui se déplacera dans la liste :
   Lc saisie_endroit3 (int nb) { Lc tete, crt ; /* . . . */ }
   4.4 Ecrire une procédure itérative *saisie_endroit4* qui fait les mêmes opérations.
   void saisie_endroit4 (int nb, Lc * p)

Exercice 2 facultatif: Liste chaînée ordonnée de noms (ordre croissant)

#define MAX 25
typedef char Nom [MAX] ;
typedef struct name { Nom nom ; struct name * suiv ; } Node ;
typedef Node * Lcn ;
int strcmp (char * c1, char * c2) ; void strcpy (char * c1, char * c2) ; /*cf poly initiation au C p. 72*/

1. Ecrire une fonction *ajout_tete* permettant d'ajouter en tête de la liste chaînée un nom.
   Lcn ajout_tete (Nom nm, Lcn p)
2. Ecrire une procédure d'affichage de la liste chaînée.
3. Ecrire le programme principal qui teste les 2 sous-programmes en ajoutant 2 à 3 noms à une liste vide.
4. Ecrire une fonction récursive d'ajout d'un nom *nm* à sa place dans la liste chaînée ordonnée p.
   Lcn ajout (Nom nm, Lcn p)
5. Transformer le programme principal pour tester ce sous-programme en ajoutant 2 à 3 noms à une liste vide.
6. Réécrire une fonction itérative *ajout2* qui fait la même opération. Pour cela on utilisera 2 variables locales *courant* et *precedent* de type *Lcn.* On vérifiera si l'ajout doit se faire en tête. Si ce n'est pas le cas, on déplacera les deux pointeurs jusqu'à ce que l'on encadre la future position de l'ajout.
7. Transformer le programme principal pour tester ce sous-programme.
8. Ecrire une procédure récursive *saisie* qui crée une liste chaînée ordonnée de *nb noms* lus.
   void saisie (int nb, Lcn *p)
5. Donner une version itérative de *saisie.*
6. Modifier le programme principal.

UVHC-ISTV

Licence 2 SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 4 d'Informatique

Exercice 1 : **Liste chaînée mono-directionnelle LCM**

Dans cet exercice, nous allons représenter les vecteurs de réels par des LCM.

typedef struct el { float val ; struct el *suiv ; } Element ;
typedef Element * Vecteur ;

1. Ecrire une fonction *Affichage* d'un vecteur représenté par une LCM . Ecrire une autre fonction d'affichage à l'envers. Ecrire la programme principal pour tester ces deux fonctions.

2. Ecrire une fonction récursive *float prodscalaire (Vecteur x, Vecteur y)* qui calcule le produit scalaire des deux vecteurs x et y. Le produit scalaire de deux vecteurs est défini par ProdScalaire(x, y) = $\sum$ x(i)*y(i) pour tous les i entre 1 et n, n est le nombre d'éléments des deux vecteurs. Lorsque les deux vecteurs ont des longueurs différentes la fonction affiche un message d'erreur.

4. Donner la version itérative de *prodscalaire*.

5. Ecrire une fonction récursive *vecteur MaxVect(Vecteur x, Vecteur y)* qui crée un vecteur contenant le maximum des deux vecteurs x et y élément par élément.

Si z = MaxVect (x, y) alors z(i) = max (x(i), y(i)) si x(i) et y(i) existent

Si x(i) (resp. y(i)) n'existe pas alors z(i) = y(i) (resp. x(i)).

X =	4	51	20	3	10	2	4	55	22	30	10	1
Y=	1	51	1	12	2	0	22					
Z = MaxVect =	4	51	20	12	10	2	22	55	22	30	10	1

6. On dispose de deux listes chaînées mono-directionnellles L1 et L2 de réels triés par ordre croissant. Ces deux listes sont connues par les deux pointeurs *p1* et *p2* de type ***Pnœud.***

6.1. Déclarer le type ***Pnoeud***.

6.2. Ecrire la fonction récursive fusion qui permet de fusionner les deux listes l1 et l2 afin d'obtenir une liste contenant l'ensemble des éléments dans l'entête est :

Pnoeud fusion (Pnœud p1, Pnœud p2)

N.B. : après fusion, les 2 listes L1 et L2 ne sont pas modifiées

Exercice 2 facultatif:

Soit la liste chaînée de noms définie par les types suivants :

typedef struct name {char nom [20] ; struct name *suiv ; } Node;
typedef Node * Lcn ;

1. Ecrire une fonction qui saisit une liste chaînée de noms à l'endroit ou à l'envers(cf TD 3).
Lc saisie (int nb)

2. Ecrire une fonction récursive ***max*** qui calcule la plus grande valeur de la liste chaînée.

3. Ecrire le programme principal qui teste ces deux fonctions : saisir une liste puis rechercher et afficher le max de cette liste.

4. Ecrire une fonction récursive ***supprime*** qui permet de supprimer de la liste chaînée toutes les valeurs supérieures à une valeur maximale donnée comme paramètre.

5. Compléter le programme principal qui teste ces deux fonctions .

6. Ecrire une fonction ***inverse*** qui inverse la liste chaînée (le premier élément devient le dernier, le deuxième élément devient l'avant dernier, ... etc.).

UVHC-ISTV

Licence 2 SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 5 d'Informatique
Sujet : arbre binaire

Ex 1 : Arbre binaire de Personnes (arbre généalogique ascendant) : partie pratique

Soit le type Personne:

```
#define MAX  20
typedef  char  Chaine [ MAX] ;
typedef  struct  {      Chaine nom , pren  ; } Personne ;
```

1. Définir les types ***Noeud*** et ***Arbre*** correspondant à une représentation dynamique (avec pointeurs).
2. Ecrire un sous-programme de saisie d'un arbre avec 4 noeuds : vous, père, mère et un grand-parent
   Arbre saisie ( ) ; ***void saisie2 (Arbre * prac) ;***
3. Ecrire le programme principal pour tester ce sous-programme.
4. Ecrire les 3 procédures de parcours d'un arbre pour afficher les valeurs. ***void ordre (Arbre x) ;***
5. Ecrire le programme principal qui saisit l'arbre avec 4 noeuds et l'affiche suivant les 3 parcours.
6. Ecrire une fonction qui compte le nombre de noeuds. ***int nbnoeud (Arbre x) ;***
7. Compléter le programme principal …
8. Ecrire une fonction qui compte le nombre de feuilles. ***int nbfeuilles (Arbre x) ;***
9. Compléter le programme principal …
10. Ecrire une fonction qui calcule la hauteur de l'arbre. ***int hauteur (Arbre x) ;***
11. Compléter le programme principal …
12. Ecrire une fonction qui dit si tel nom existe dans l'arbre. ***int existe (Arbre x, Chaine nm) ;***
13. Facultatif : Ecrire une fonction booléenne qui ajoute une Personne comme étant le fils droit de telle autre Personne. Le résultat est vrai si correct et faux si la place est prise.
    int ajout (Arbre *px, Chaine p, Chaine nm) ;
14. Facultatif : Comment transformer la fonction précédente pour ajouter 1 ou 2 ascendants à une personne ?

Ex 2 facultatif : Expression arithmétique et arbre

1 Définition

- Rappeler les différentes notations d'une expression arithmétique: algébrique, post-fixée, pré-fixée
- Représenter sous la forme d'un arbre binaire: traiter l'exemple suivant (x-(2+y))+((x+(y+2))*3) (ex)

2 Représentation d'un arbre binaire

Sachant que chaque valeur est de type *Element*, décrire le type *Arbre*

- Solution 1 avec pointeurs (cf cours)
- Solution 2 simulée avec un tableau de 3 champs: valeur de type *Element* et *fg, fd* de type *Indice*

Donner la représentation, sous forme de tableau, de l'arbre de (ex)

3 Parcours

- Rappeler les 3 parcours
- Dérouler les 3 parcours sur l'exemple de l'arbre relatif à (ex)
- En utilisant la solution 2 précédente, écrire la procédure récursive PREORDRE appliquée à l'*arbre* X en utilisant une procédure TRAITER (*Element z*)

solution avec un second paramètre de type *Indice* ***void preordre (Arbre x, Indice i) ;***

solution bis avec un seul paramètre x ***void preordre (Arbre x) ;***

UVHC-ISTV

Licence 2 SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 6 d'Informatique
Sujet : arbre binaire ordonné

Ex 1 : Les arbres binaires de recherche

On veut utiliser un arbre binaire de recherche pour représenter les mots d'un dictionnaire. Chaque nœud contient un mot et deux pointeurs : fg et fd pointant sur les sous arbres gauche et droit.

Voici les types de données à utiliser :

```
typedef  char  Tabcar [20] ;
typedef struct e { Tabcar mot ; struct e *fd, *fg ;} Nœud ;
typedef Nœud * Pointeur ;
```

1) Ecrire un sous-programme récursif qui permet de rajouter le mot m à l'arbre de mots pointé par racine.
   Pointeur ajout (Pointeur racine, Tabcar m) ;
   void ajout2 (Pointeur *prac, Tabcar m) ;
2) Ecrire le programme principal qui teste ce sous-programme en ajoutant 2 ou 3 mots.
3) Ecrire une procédure récursive ***int nbr_mot (Pointeur racine)*** calculant le nombre de mots contenus dans la structure.
4) Ecrire une fonction ***Affichage*** qui permet d'afficher l'ensemble ordonné des mots contenus dans l'arbre.
5) Ecrire le programme principal qui ajoute 2 ou 3 mots à un arbre vide puis l'affiche.

Facultatif :

6) En utilisant la fonction ***ajout***, écrire une fonction ***recopie*** qui permet de lire, à partir d'un fichier, une suite de mots (chaîne de 20 caractères maximum) et la stocke dans un arbre binaire de recherche de mots.
   Pointeur recopie(char nomfichier[15]) ;
   void recopie2 (char nomfichier [15], Pointeur *prac) ;
7) Ecrire une fonction ***int recherche (Pointeur racine, Tabcar m)*** permettant de rechercher le mot m dans l'arbre pointé par racine. Si le mot m appartient à l'arbre le résultat sera 1 sinon le résultat sera 0.

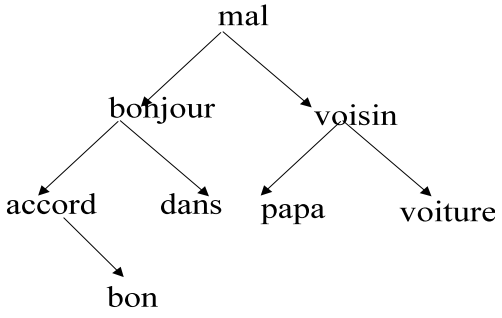
Ecrire une fonction **Pointeur supprime (Pointeur racine, Tabcar m)** qui permet de supprimer de l'arbre le mot m. (N.B : Il y a trois cas à traiter)

Ex 2 facultatif : Tri par arbre binaire

On veut écrire une procédure de tri d'un tableau T de N valeurs *réelles* en utilisant un *arbre* binaire ordonné (ordre horizontal) tel qu'à chaque noeud on ait Valeur de FG <= Valeur du noeud < Valeur de FD

On crée l'arbre à partir de chaque valeur du tableau T puis on parcourt l'arbre en symétrique.

1. Simuler sur un exemple de 8 valeurs extraites d'un tableau
2. Ecrire une procédure AJOUT d'une valeur v dans l'*arbre* X : ***Arbre ajout (Arbre x, float v) ;***
   - Solution récursive, ***void ajout2 (Arbre * px, float v) ;***
   - Solution itérative.
3. Ecrire la procédure de création de l'arbre. ***Arbre creation (Tab t) ;***
4. Ecrire la procédure d'affichage des valeurs dans l'ordre croissant. ***void affich (Arbre x) ;***
5. Ecrire une procédure de recopie de l'*arbre* dans le *tableau* T. ***void recopie (Arbre x, Tab t, int * pi) ;***
6. Ecrire la procédure de tri en utilisant les procédures précédentes. ***void tri (Tab t, int n) ;***
7. Evaluer la complexité du tri en nombre d'itérations: création puis recopie.



UVHC-ISTV

Licence 2 Info semestre 4 : ALGO et SD de base
Fiche I d'exercices complémentaires

Exercice 1 Evaluation d'une expression arithmétique ou logique

On s'intéresse au traitement des expressions arithmétiques ou logiques utilisées dans les programmes écrits dans un langage de programmation classique : les algorithmes d'évaluation et de transformation d'écriture d'expressions nécessitent l'utilisation d'une pile. L'expression peut être écrite :

- Sous forme complètement parenthésée (ECP) ((A+B) * C)
- Sous forme préfixée (EPR) * + A B C
- Sous forme postfixée (EPO) A B + C *
- Sous forme infixée (EIN) (A + B) * C

Evaluer une expression consiste à calculer la valeur de l'expression pour les valeurs associées à chaque variable.
On supposera que l'expression est représentée par une chaîne de caractères terminée par # et de taille <MAX. On se limitera aux variables réelles identifiées par un seul caractère et aux 4 opérateurs '+' , '-', '*' et '/'.

1) Rappeler les opérations sur une pile représentée par une liste chaînée de réels :

typedef Elem * Pile ;
Pile initpile ( ) ; int pilevide (Pile) ; Pile empiler (Pile , float) ; Pile depiler (Pile, float *) ;
void initpile2 (Pile *) ; int pilevide (Pile) ; void empiler2 (Pile * , float) ; void depiler2 (Pile *, float *) ;

2) Evaluation d'une expression postfixée à l'aide d'une pile.
   2.1 Soit l'expression A B * C D + / avec A=20, B=4, C=9, D=7 : évaluer la en montrant les états successifs de la pile.
   2.2 De l'exemple précédent, en déduire les règles pour cette évaluation.
   2.3 Ecrire la fonction itérative ***float evalpost (chaine epo)*** qui évalue la chaine EPO en utilisant la pile.
3) Transformation de la représentation complètement parenthésée en une représentation postfixée.
   void cppost (chaine ecp, chaine epo)
   3.1 Simuler le déroulement de la transformation en utilisant une pile : ((A * B) + C)#
   3.2 De l'exemple précédent, en déduire les règles de transformation.
   iquement correcte.
   3.4 Améliorer cet algorithme afin qu'il détecte les erreurs de syntaxe dans l'expression complètement parenthésée.
4) Ecrire la fonction d'évaluation d'une expression préfixée supposée syntaxiquement correcte.
   float evalpr (chaine epr)
5) Passage de la représentation infixée à la représentation postfixée.
   5.1 Simuler le déroulement de la transformation en montrant les états successifs de la pile et de l'expression avec l'expression A * B + C / (D + E)
   5.2 En déduire les règles de la transformation
   5.3 Ecrire la procédure de transformation en utilisant la pile représentée par une liste chainée
   void inpost (chaine ein, chaine epo)

Exercice 2 facultatif : Evaluation de sous-programmes récursifs à l'aide d'une pile

Soit les 3 procédures suivantes

```
void f1 ( int k ) { if ( k>0) { printf («%3d », k) ; f1(k-2) ; } }
void f2 ( int k ) { if ( k>0) { f2(k-1) ; printf («%3d », k) ; } }
void f3 ( int k ) { if ( k>0) { f3(k-1) ; printf («%3d », k) ; f3(k-2) ; } }
```

1) Dérouler les 3 procédures f1, f2, f3 avec k=4.
2) On définit une pile d'entiers représentée par un tableau : définir le type *Pile* , ainsi que les opérations *init*, *empiler, depiler, pilevide.*
3) Remplacer la procédure récursive f2 par une procédure itérative qui affiche la même suite de valeurs (on utilisera une pile d'entiers représentée par un tableau).
4) Proposer une solution sans pile sur ce cas particulier de procédure.

Remplacer la procédure récursive f3 par une procédure itérative qui affiche la même suite de valeurs (on utilisera une pile d'entiers représentée par un tableau).

UVHC-ISTV

Licence 2 Info semestre 4 : ALGO et SD de base

Fiche II d'exercices complémentaires

```
#define MAXNOM  20
#define  MAXNUM  10
typedef   char  type_nom  [  MAXNOM ] ;
typedef  char   type_num  [  MAXNUM ] ;
typedef  struct {      type_nom  nom ;   type_num  num  ;        } Abonne ;
```

Exercice 1 : File d'attente représentée par une Liste avec 2 pointeurs de tête et queue

Une liste avec 2 pointeurs de tête et queue, TQlist est une liste chaînée simple. Cependant on y ajoute un pointeur de queue qui repère le dernier élément de la liste. Le suivant du dernier élément prend toujours une valeur indéfinie (ni NIL, ni autre).

1: Donner les types permettant de construire des listes *TQList* de *Abonne* et une structure *FA* regroupant *tete* et *queue*.

2: Ecrire la fonction (*fa_init*) initialisant une *FA*.

3: Ecrire la fonction (*fa_vide*) testant si une *FA* est vide.

4: Ecrire le sous-programme (*enfiler*) d'insertion en queue d'un *Abonne* dans une *FA*.

```
FA  enfiler  ( FA  f,  Abonne a )
void enfiler2 ( FA  * pf,  Abonne a )
```

5: Ecrire le sous-programme (*defiler*) de retrait en tête d'un *Abonne* dans une *FA*.

6: Ecrire la procédure (*afficher*) d'affichage des *Abonne* d'une *FA*.

7: Ecrire la fonction (*longueur*) donnant le nombre d'*Abonne* d'une *FA*.

Exercice 2 : File d'attente représentée par une Liste chaînée circulaire avec 1 pointeur queue

```
typedef  Element  * Lca ;
typedef   struct  {  Abonne  ab ;  Lca  suiv ; } Element ;
typedef  Lca  FA ;
```

Dans une liste circulaire appelée ici *FA*, le suivant du dernier élément est le premier élément. On prévoit un pointeur de queue qui repère le dernier élément de la liste. Le suivant du dernier élément correspond à la tête de la liste et donc de la File d'attente.

1. Ecrire une procédure itérative d'affichage des *Abonne* de la *FA*.
   ```
   void  afficher ( FA  queue )
   ```
2. Proposer une solution récursive.
3. Ecrire la fonction longueur donnant le nombre d'*Abonne* d'une *FA*.
4. Ecrire le sous-programme (*enfiler*) d'insertion en queue d'un *Abonne* dans cette *FA*.
   ```
   FA  enfiler ( FA queue ,  Abonne a )
   void  enfiler2 ( FA * pqueue, Abonne a)
   ```
5. Ecrire le sous-programme (*defiler*) de retrait en tête d'un *Abonne* dans cette *FA*.
6. Quel est l'intérêt de cette représentation. ?

UVHC-ISTV

Licence 2 SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 6 d'Informatique
Sujet : arbres n-aires et Arbre généalogique descendant

Un arbre généalogique descendant est souvent un arbre n-aire : Par exemple Julie a 2 enfants Jonathan et Gontran. Jonathan a 3 enfants : Pauline, Sonia et Paul. Gontran a une fille Antonine. Le degré de l'arbre est de 3 ; l'arbre est dit 3-aire ou n-aire avec n=3. Le degré de chaque nœud est variable puisqu'il dépend du nombre d'enfants de ce nœud. Julie est la racine de l'arbre.

I Représentation de l'arbre (à faire en cours)

Il s'agit de mémoriser les arbres et leurs relations de dépendance père fils.

I-1 Représenter graphiquement l'arbre généalogique précédent.

I-2 Mémorisation par listes de fils:On a un tableau de nœuds et pour chacun une liste de fils (fils du nœud)

Donner les types correspondants. Avec cette structure de données, les insertions et les suppressions de nœuds ne sont pas faciles à gérer. De plus il est difficile de donner un maximum pour la déclaration si on peut ajouter des éléments.

I-3 Allocation contigüe (tableau ou fichier) : L'arbre peut être mémorisé dans un espace contigü équivalent à un tableau à 2 dimensions avec 4 colonnes. L'espace mémoire est réservé à la déclaration en précisant un maximum (nombre max de nœuds et nombre max de fils). Cet espace peut être réservé en mémoire centrale ou sur mémoire secondaire. Les pointeurs sont alors des indices de tableau ou des numéros d'enregistrement, s'il s'agit d'un fichier. On peut choisir –1 pour indiquer l'absence de successeurs pour un nœud. Donner les types correspondants.

I-4 Allocation dynamique : Les éléments sont alloués au cours de la construction de l'arbre et reliés entre eux. Le nombre de pointeurs présents dans chaque noeud dépend du degré de l'arbre. Si le degré de chaque nœud est constant, cette mémorisation est parfaite. Donner les types correspondants pour un arbre 3-aire.

Dans le cas général d'un arbre n-aire, on remplace les 3 pointeurs par une liste chaînée : donner les types.

I-5 Transformation d'un arbre n-aire en un arbre binaire .

*typedef struct n { char nom [20] ; struct n * fils, * frere ; } Nœud ; typedef Nœud * Pnoeud ;*

On mémorise pour chaque nœud un pointeur vers son premier fils (le plus âgé) -lien vertical- et un vers son frère immédiat plus jeune –lien horizontal- . Ces deux liens (fils et frère) ou (lien vertical et lien horizontal) suffisent pour représenter tout arbre n-aire. Transformer l'arbre 3-aire de I-1 en binaire.

II Pratique : Quelques algorithmes sur des arbres n-aires transformés en arbres binaires (Arbre Généalogique Descendant)

*typedef struct n { char nom [20] ; struct n * fils, * frere ; } Nœud ; typedef Nœud * Pnoeud ;*

II-0 Créer un arbre n-aire: On reprendra un algorithme vu précédemment.

II-1 Afficher les feuilles n-aires (c-a-d les personnes n'ayant pas d'enfant) : Une feuille n-aire a un sous-arbre gauche vide, le pointeur sur le premier fils est à NULL. Ecrire l'algorithme de parcours d'un arbre binaire avec écriture des noeuds ayant un sous-arbre gauche vide.

void listerfeuillenaire (Pnoeud rac)

II-2 Afficher les descendants n-aires d'un nœud.

void descendantnaire (Pnoeud rac)

II-3 Ecrire une fonction qui recherche la place ou l'adresse (pointeur) d'un nom dans l'arbre généalogique

Pnoeud cherche (char nm [ ], Pnoeud rac)

II-4 Ecrire une fonction qui calcule le nombre de personnes dans l'arbre.

int nbpers (Pnoeud p)

Il faut penser à p lui-même, aux frères cadets de p et à leurs descendants, aussi bien qu'aux descendants directs de p.

II-5 *Facultatif :* Ecrire une fonction qui calcule le nombre de descendants directs de nm dans l'arbre.

*int nbdesc (char * nm, Pnoeud rac)*

II-*6 Facultatif* : Ecrire une fonction qui dit si nom appartient à la liste d'enfants dont p est le pointeur sur le père

*int existefils (Pnoeud p , char * nom)*

II-7 *Facultatif :* Ecrire une fonction qui retourne le pointeur sur le père de l'enfant nom dans l'arbre p .

*Pnoeud pere (Pnoeud p, char * nom)*

II-8 *Facultatif* **:** Ecrire une procédure qui affiche la *liste des ascendants de nom*, c-a-d père, grand-père,…

Ecrire une procédure de création d'une nouvelle personne d'adresse p ayant 1 nom, sans descendant ni frère

Ecrire une procédure *d'insertion du nouveau né nom* à la fin de la liste des enfants dont p est le pointeur sur aîné.

Une naissance vient de se produire et on désire inscrire ce nouvel enfant dans l'arbre : écrire la procédure.

UVHC-ISTV

Licence 2 Info semestre 4 : ALGO et SD de base
Fiche TD 5
Sujet : arbres de Recherche équilibrés

Rappels du cours

arbre binaire équilibré = arbre 'bien équilibré', lorsque chacun de ses noeuds possède un sous-arbre à gauche et un sous-arbre à droite ayant le même nombre de noeuds à une unité près.
arbre équilibré en profondeur ou H-équilibré= arbre dont la profondeur (la hauteur) des sous-arbres ne varie que d'une unité au maximum.

Exercice 1. Complexité de la Recherche

1. Représenter graphiquement la structure des 4 ABR construits par les insertions suivantes:

arbre A :	tataratata,	tata,	tete,	titi,	toto,	turlututu,	tutu
arbre B :	tutu,	turlututu,	toto,	titi,	tete,	tata,	tataratata
arbre C :	titi,	tataratata,	tete,	turlututu,	toto,	tutu,	tata
arbre D :	tata,	tataratata,	titi,	tete,	turlututu,	toto,	tutu

2. Comparer la recherche d'une valeur dans ces arbres binaires en calculant le nombre moyen de comparaisons.

Exercice 2 : Construction d'un arbre binaire

```
Pnœud   arbre_be (  int n )
        {       int   nb_g,  nb_d,  nb_GplusD = n-1; Pnœud p; if (n == 0)   return  NULL;
                nb_g = nb_GplusD / 2;   nb_d = nb_GplusD - nb_g;
                p = ( Pnœud ) malloc (sizeof (Nœud) );
                p->fg = arbre_be ( nb_g );
                scanf ( «%d »,  &p->num);
                p->fd  =  arbre_be ( nb_d );
                return p;
        }
```

1. Définir les types ***Noeud*** et ***Pnoeud***
2. Dérouler cet algorithme sur les valeurs 5 8 13 15 20 avec n =5
3. Dérouler cet algorithme sur les valeurs 20 15 13 8 5 avec n =5
4. Dérouler cet algorithme sur les valeurs 8 13 5 20 15 avec n =5
5. Qu'en conclure ?

Exercice 3 pratique

A partir d'un arbre saisi à partir du clavier, par exemple en utilisant la fonction arbre_be, on sera amené à écrire quelques fonctions que l'on testera.

1) Ecrire une fonction qui permet de savoir si un arbre est équilibré en profondeur (hauteur).
2) Ecrire une fonction qui permet de savoir si un arbre est bien équilibré.
3) Ecrire une fonction qui permet de savoir si un arbre est ordonné de manière croissante (ABR).

Exercice 4 : *Facultatif* Ajout et Construction d'un arbre binaire équilibré

Introduits dans les années 60 par Adelson-Velski et Landis, ces ABR sont H-équilibrés (ou en profondeur). On mémorise un facteur d'équilibrage **bal** (profondeur du sous-arbre droit diminuée de celle du sous-arbre gauche) égal à 0,1,-1.

```
typedef struct elem {int val, bal;  struct elem *fg, *fd;}Noeud;
typedef Noeud * Pnoeud;
```

Construire à la main un arbre binaire équilibré en indiquant la valeur de BAL à chaque fois :
10 17 8 20 25
puis un autre avec 10 17 8 20 15 25

UVHC-ISTV

Licence 2 SPI parcours Info semestre 4 : ALGO et SD de base
Fiche de TD 5bis d'Informatique

Exercice 1: Liste chaînée à plusieurs niveaux

On souhaite gérer une population composée de plusieurs familles limitées à deux générations.
Au premier niveau la liste est ordonnée par ordre alphabétique sur les noms (chaîne de 20 caractères ***Ch20***) de famille tous différents (liste chaînée de familles)
Une famille est une enregistrement avec un nom, un pointeur sur une liste de parents (maximum deux), un pointeur sur une liste d'enfants (nombre illimité) et un pointeur sur une liste de voitures (marque, numéro: ***Ch20***). Chaque personne (parent ou enfant) est définie par un prénom(de type ***Ch20***) et un sexe et appartient à une seule famille.

a. Représenter par une figure un exemple d'une population de trois familles.
b. Définir les types ***Ch20, Famille, Personne, Voiture*** , ainsi que ***Pfamille, Ppersonne, Pvoiture*** .
c. Ecrire une fonction itérative entière ***int nbfamillesansparent (Pfamille pop)*** /* parents décédés */
d. Ecrire une fonction récursive entière ***int nbcelibat (Pf amille pop)*** /* 1 seul parent */
e. Ecrire une fonction récursive entière ***int nbenfant (Pfamille pop)*** /* nb total d'enfants */
f. Un vol de voiture s'est produit. On désire connaître le propriétaire de la voiture. Ecrire sous forme récursive une fonction qui recherche l'adresse du propriétaire dans la liste chaînée de familles:
Pfamille appartvoit (Pfamille pop, Ch20 numero)

Exercice 2: Liste chaînée avec sentinelle :

On a vu dans la solution itérative de la saisie à l'endroit que l'on devait traiter le cas particulier du premier avant de traiter la saisie des autres. Une méthode pour éviter ceci est de prévoir un premier élément fictif qui joue le rôle de sentinelle de telle sorte que le premier vrai élément inséré en tête se range après la sentinelle.

a. Initialisation : La fonction suivante est erronée, corriger la :
Lc init ( ) { Lc sentinelle = (Lc) malloc (sizeof (Element)) ; return sentinelle ; }
b. Saisie à l'endroit : Ecrire la version itérative de la saisie à l'endroit d'une liste chaînée avec sentinelle.
c. Si on suppose que la liste est ordonnée selon les degrés, décrire par un dessin comment insérer une nouvelle valeur dans la liste chaînée puis écrire le sous-programme.

Exercice 3 pratique: Liste Circulaire

Dans une liste circulaire *CList*, le suivant du dernier élément est le premier élément.

a. Donner les types permettant de construire des listes circulaires de réels.
b. Ecrire la fonction (*clist_init*) retournant un pointeur et initialisant une *CList*.
Ecrire une procédure *init2* permettant d'initialiser un paramètre x de type CList.
c. Ecrire la fonction booléenne (*clist_vide*) testant si une *CList* est vide.
d. Ecrire la fonction (*inserer*), qui renvoie un pointeur, d'insertion en tête d'un réel dans une *CList*.
Réécrire une procédure *inserer2* où le paramètre *tete* est en entrée-sortie.
e. Ecrire la fonction (*retirer*) de retrait en queue d'un dans une *CList*.

Facultatif :
Réécrire *retirer* sous la forme d'une procédure avec un paramètre d'entrée sortie.
Ecrire la procédure (*afficher*) d'affichage des valeurs dans une *CList*.
Ecrire la fonction (*longueur*) donnant le nombre de valeurs d'une *CList*

Licence 2 Info & Math semestre 4 && IUP GMI sem 2 : ALGO4

Fiche I I I d'exercices complémentaires

DS d'ALGO4 (mercredi 24 mars 2004) Durée 1h30 et sans document

typedef struct { char nom [20], pren [20] ; } Personne ;

Exercice 1

Soit une liste chaînée non ordonnée de personnes repérées par un nom et un prénom.
typedef struct s { Personne personne ; struct s * suiv ; } Elem ;
typedef Elem * LP ;

1) Ecrire le sous-programme de création d'une liste chaînée de n personnes.
2) Ecrire le sous-programme d'affichage de toutes les personnes dont le nom commence par un caractère (paramètre).
3) Ecrire le sous-programme de suppression d'une personne repérée par son nom.
4) Ecrire le programme principal créant la liste avec n lu, affichant les noms commençant avec 'c', supprimant un nom lu, et enfin réaffichant la liste .

Exercice 2

Soit la même liste de personnes mais supposée ordonnée.

1) Ecrire le sous-programme d'ajout d'une personne dans la liste à sa place.
2) Ecrire le sous-programme de fusion de 2 listes ordonnées dans une troisième sans changer les deux premières.
3) Ecrire le sous-programme de suppression des doublons (personnes ayant les mêmes nom et prénom). Quelle est la complexité de cet algorithme en termes de comparaisons ?

Exercice 3

(a - (b + (c * d)))

Pour l'expression arithmétique complètement parenthésée, décrire et expliquer la suite des opérations permettant

1) de la transformer en notation inversée post-fixée en utilisant une pile (ne pas oublier de dessiner l'évolution de la pile)
2) d'évaluer une telle expression complètement parenthésée en utilisant deux piles (ne pas oublier de dessiner l'évolution des piles)
3) d'évaluer cette expression si elle n'est plus complètement parenthésée:

a–(b+c*d)

Barème indicatif: ex1: 8 (2+2+3+1) ex2: 7 (2+3+3) ex3: 5 (1.5+2.5+1)

UVHC-ISTV

Licence 2 Info & Math semestre 4 && IUP GMI sem 2 : ALGO4

Fiche III bis d'exercices complémentaires

DS d'Informatique (31/03/2001 - durée 2h sans document)

Exercice 1: Polynômes creux représentés par une liste chaînée de monômes

Un polynôme peut être vu comme une somme de monômes . Chaque monôme est caractérisé par son degré et son coefficient. Un polynôme creux est de degré élevé en ayant une majorité de monômes nuls.

Dans cette représentation par une liste chaînée, on ne retient que les monômes de coefficient non-nul.

On définit le type ***Monome*** sous la forme d'un enregistrement. Puis avec ce type **Monome**, on définit le type ***Nome*** sous la forme d'un enregistrement à deux champs et le type ***Poly*** sous la forme d'un pointeur sur ***Nome.***

a. Définir les types ***Monome***, ***Nome***, ***Poly***.

b. Comment sera représenté le polynôme constant nul (P(X) = 0) ?

c. On rappelle que la saisie à l'endroit correspond au fait que l'on ajoute un élément en queue de la liste et donc le fait de répéter ceci amène à saisir les valeurs dans l'ordre exact de leur future position dans la liste (d'où le nom à l'endroit). Donner la fonction récursive de saisie à l'endroit d'une variable de type ***Poly*** ***Poly saisieP (Poly p)*** en utilisant le sous-programme ***Monome saisieM ( );***

d. Donner une solution itérative de saisie à l'endroit ***saisieP*** .

e. Donner un sous-programme de calcul de la dérivée d'un polynôme ***p*** (***p*** est transformé). Attention : on ne représente pas le monôme à coefficient nul ***Poly deriver (Poly p)***

f. Donner le sous-programme de calcul de la somme de deux polynômes (***p*** et ***q*** restent inchangés).
Poly Somme (Poly p , Poly q)

Exercice 2: Liste chaînée à plusieurs niveaux

On souhaite gérer une population composée de plusieurs familles limitées à deux générations.

Au premier niveau la liste est ordonnée par ordre alphabétique sur les noms (chaîne de 20 caractères ***Ch20***) de famille tous différents (liste chaînée de familles)

Une famille est une enregistrement avec un nom, un pointeur sur une liste de parents (maximum deux), un pointeur sur une liste d'enfants (nombre illimité) et un pointeur sur une liste de voitures (marque, numéro: ***Ch20***). Chaque personne (parent ou enfant) est définie par un prénom(de type ***Ch20***) et un sexe et appartient à une seule famille.

g. Représenter par une figure un exemple d'une population de trois familles.

h. Définir les types ***Ch20, Famille, Personne, Voiture*** , ainsi que ***Pfamille, Ppersonne, Pvoiture*** .

i. Ecrire une fonction itérative entière ***int nbfamillesansparent (Pfamille pop)*** /* parents décédés */

j. Ecrire une fonction récursive entière ***int nbcelibat (Pf amille pop)*** /* 1 seul parent */

k. Ecrire une fonction récursive entière ***int nbenfant (Pfamille pop)*** /* nb total d'enfants */

l. Un vol de voiture s'est produit. On désire connaître le propriétaire de la voiture. Ecrire sous forme récursive une fonction qui recherche l'adresse du propriétaire dans la liste chaînée de familles:
Pfamille appartvoit (Pfamille pop, Ch20 numero)

Exercice 3: traiter au choix soit la partie A soit la partie B

Partie A Liste chaînée avec sentinelle : On a vu dans la solution itérative de la saisie à l'endroit que l'on devait traiter le cas particulier du premier monôme avant de traiter la saisie des autres. Une méthode pour éviter ceci est de prévoir un premier élément fictif qui joue le rôle de sentinelle de telle sorte que le premier vrai élément inséré en tête se range après la sentinelle.

b. Initialisation : La fonction suivante est erronée, corriger la :
Poly init () {Poly sentinelle= (Poly) malloc (sizeof (Nome)); return sentinelle ; }

e. Saisie à l'endroit : Ecrire la version itérative de la saisie à l'endroit d'une liste chaînée avec sentinelle.

f. Si on suppose que la liste est ordonnée selon les degrés, décrire par un dessin comment insérer un nouveau Monôme dans la liste chaînée puis écrire le sous-programme.

Partie B Piles et Files d'attente : Ces structures sont des listes particulières. Contrairement aux fichiers et aux vecteurs , elles ne servent généralement pas à garder de façon plus ou moins définitive des informations.

a. Quelles sont les différences entre pile et file d'attente ? Comment sont-elles gérées ?

b. Dérouler la procédure suivante avec n =5
void proc (int n) { Pile p= pile_vide ( ) ; p = empiler (p, n--) ; while (n>0 || ! pile_vide (p))
{ while (n>0) p = empiler (p, n --) ; n = depiler (&p) ; printf (« %2d », n) ; n -= 2; } }

c. Définir le type ***Pile*** et écrire les sous-programmes ***empiler*** et ***depiler***

Barême indicatif: Ex 1: 9pts (1.5+0.5+1.5+1.5+1.5+2.5) ; Ex2: 7 pts (1.5+1.5+4*1); Ex3: 4 pts (0.5+1.5+2)

UVHC-ISTV

Licence 2 Info & Math semestre 4 && IUP GMI sem 2 : ALGO4

Fiche III ter d'exercices complémentaires

DS d'ALGO4 (lundi 24 mars 2003)

Durée 1h30 et sans document

Exercice 1 : Liste chainée

```
typedef struct  a   {      int val ;
                           struct a * suiv;
                    } Elem;
typedef Elem * Lc;
```

1) Ecrire la fonction qui permet d'initialiser une liste chaînée.
2) Ecrire la fonction qui permet d'ajouter l'entier x en tête de la liste chaînée p :Lc.
3) Ecrire la fonction qui permet d'ajouter l'entier x en queue de la liste chaînée p :Lc.
4) Ecrire la fonction récursive qui permet de créer une liste chaînée de nb valeurs. Expliquer bien les choix que vous faites !
5) Ecrire la fonction récursive permettant de supprimer la valeur x de la liste chaînée. Proposer une solution qui retourne aussi un booléen indiquant si la suppression a pu se faire. S'il y a plusieurs occurrences de cette valeur, on supprimera la première rencontrée.
6) Donner une version itérative de la fonction de suppression.

Exercice 2 : Expression arithmétique et Pile

Pour les deux expressions arithmétiques en notation inversée (post-fixée ou pré-fixée), décrire et expliquer la suite des opérations permettant d'évaluer une telle expression en utilisant une pile (ne pas oublier de dessiner l'évolution de la pile) :

2 3 - 4 5 6 - - *

- + - 3 4 5 * 2 3

Barème indicatif : Ex 1 : 16 points (1+2+3+4+4+2) Ex 2 : 4 points (2+2)

UVHC-ISTV

Licence 2 Info & Math semestre 3 ALGO3
Fiche de TD 9 d'Informatique
Sujet : arbres de Recherche équilibrés

Rappels du cours

arbre binaire équilibré = arbre 'bien équilibré', lorsque chacun de ses noeuds possède un sous-arbre à gauche et un sous-arbre à droite ayant le même nombre de noeuds à une unité près.
arbre équilibré en profondeur ou H-équilibré= arbre dont la profondeur (la hauteur) des sous-arbres ne varie que d'une unité au maximum.

Exercice 1. Complexité de la Recherche

1. Représenter graphiquement la structure des 4 ABR construits par les insertions suivantes:

arbre A :	tataratata,	tata,	tete,	titi,	toto,	turlututu,	tutu
arbre B :	tutu,	turlututu,	toto,	titi,	tete,	tata,	tataratata
arbre C :	titi,	tataratata,	tete,	turlututu,	toto,	tutu,	tata
arbre D :	tata,	tataratata,	titi,	tete,	turlututu,	toto,	tutu

2. Comparer la recherche d'une valeur dans ces arbres binaires en calculant le nombre moyen de comparaisons.

Exercice 2 : Construction d'un arbre binaire

```
Pnœud   arbre_be (  int n )
        {       int   nb_g,  nb_d,  nb_GplusD = n-1; Pnœud p; if (n == 0)   return  NULL;
                nb_g = nb_GplusD / 2;   nb_d = nb_GplusD - nb_g;
                p = ( Pnœud ) malloc (sizeof (Nœud) );
                p->fg = arbre_be ( nb_g );
                scanf ( «%d »,  &p->num);
                p->fd  =  arbre_be ( nb_d );
                return p;
        }
```

6. Définir les types ***Noeud*** et ***Pnoeud***
7. Dérouler cet algorithme sur les valeurs 5 8 13 15 20 avec n =5
8. Dérouler cet algorithme sur les valeurs 20 15 13 8 5 avec n =5
9. Dérouler cet algorithme sur les valeurs 8 13 5 20 15 avec n =5
10. Qu'en conclure ?

Exercice 3 pratique

A partir d'un arbre saisi à partir du clavier, par exemple en utilisant la fonction arbre_be, on sera amené à écrire quelques fonctions que l'on testera.

4) Ecrire une fonction qui permet de savoir si un arbre est équilibré en profondeur (hauteur).
5) Ecrire une fonction qui permet de savoir si un arbre est bien équilibré.
6) Ecrire une fonction qui permet de savoir si un arbre est ordonné de manière croissante (ABR).

Exercice 4 : *Facultatif* Ajout et Construction d'un arbre binaire équilibré

Introduits dans les années 60 par Adelson-Velski et Landis, ces ABR sont H-équilibrés (ou en profondeur). On mémorise un facteur d'équilibrage **bal** (profondeur du sous-arbre droit diminuée de celle du sous-arbre gauche) égal à 0,1,-1.

```
typedef struct elem {int val, bal;  struct elem *fg, *fd;}Noeud;
typedef Noeud * Pnoeud;
```

Construire à la main un arbre binaire équilibré en indiquant la valeur de BAL à chaque fois :
10 17 8 20 25
puis un autre avec 10 17 8 20 15 25

UVHC-ISTV

Licence 2 Info & Math semestre 3 ALGO3
Fiche de TD 10 d'Informatique
Sujet : arbres n-aires et Arbre généalogique descendant

Un arbre généalogique descendant est souvent un arbre n-aire : Par exemple Julie a 2 enfants Jonathan et Gontran. Jonathan a 3 enfants : Pauline, Sonia et Paul. Gontran a une fille Antonine. Le degré de l'arbre est de 3 ; l'arbre est dit 3-aire ou n-aire avec n=3. Le degré de chaque nœud est variable puisqu'il dépend du nombre d'enfants de ce nœud. Julie est la racine de l'arbre.

I Représentation de l'arbre

Il s'agit de mémoriser les arbres et leurs relations de dépendance père fils.

I-1 Représenter graphiquement l'arbre généalogique précédent.

I-2 Mémorisation par listes de fils:On a un tableau de nœuds et pour chacun une liste de fils (fils du nœud)

Donner les types correspondants. Avec cette structure de données, les insertions et les suppressions de nœuds ne sont pas faciles à gérer. De plus il est difficile de donner un maximum pour la déclaration si on peut ajouter des éléments.

I-3 Allocation contigüe (tableau ou fichier) : L'arbre peut être mémorisé dans un espace contigü équivalent à un tableau à 2 dimensions avec 4 colonnes. L'espace mémoire est réservé à la déclaration en précisant un maximum (nombre max de nœuds et nombre max de fils). Cet espace peut être réservé en mémoire centrale ou sur mémoire secondaire. Les pointeurs sont alors des indices de tableau ou des numéros d'enregistrement, s'il s'agit d'un fichier. On peut choisir –1 pour indiquer l'absence de successeurs pour un nœud. Donner les types correspondants.

I-4 Allocation dynamique : Les éléments sont alloués au cours de la construction de l'arbre et reliés entre eux. Le nombre de pointeurs présents dans chaque noeud dépend du degré de l'arbre. Si le degré de chaque nœud est constant, cette mémorisation est parfaite. Donner les types correspondants pour un arbre 3-aire.

Dans le cas général d'un arbre n-aire, on remplace les 3 pointeurs par une liste chaînée : donner les types.

I-5 Transformation d'un arbre n-aire en un arbre binaire .

*typedef struct n { char nom [20] ; struct n * fils, * frere ; } Nœud ; typedef Nœud * Pnoeud ;*

On mémorise pour chaque nœud un pointeur vers son premier fils (le plus âgé) -lien vertical- et un vers son frère immédiat plus jeune –lien horizontal- . Ces deux liens (fils et frère) ou (lien vertical et lien horizontal) suffisent pour représenter tout arbre n-aire. Transformer l'arbre 3-aire de I-1 en binaire.

II Quelques algorithmes sur des arbres n-aires transformés en arbres binaires (Arbre Généalogique Descendant)

*typedef struct n { char nom [20] ; struct n * fils, * frere ; } Nœud ; typedef Nœud * Pnoeud ;*

II-1 Afficher les feuilles n-aires (c-a-d les personnes n'ayant pas d'enfant) : Une feuille n-aire a un sous-arbre gauche vide, le pointeur sur le premier fils est à NULL. Donner l'algorithme de parcours d'un arbre binaire avec écriture des noeuds ayant un sous-arbre gauche vide.

void listerfeuillenaire (Pnoeud rac)

II-2 Afficher les descendants n-aires d'un nœud.

void descendantnaire (Pnoeud rac)

II-3 Ecrire une fonction qui recherche la place ou l'adresse (pointeur) d'un nom dans l'arbre généalogique

Pnoeud cherche (char nm [ ], Pnoeud rac)

II-4 Ecrire une fonction qui calcule le nombre de personnes dans l'arbre.

int nbpers (Pnoeud p)

Il faut penser à p lui-même, aux frères cadets de p et à leurs descendants, aussi bien qu'aux descendants directs de p.

II-5 *Facultatif :* Ecrire une fonction qui calcule le nombre de descendants directs de nm dans l'arbre.

*int nbdesc (char * nm, Pnoeud rac)*

II-*6 Facultatif* : Ecrire une fonction qui dit si nom appartient à la liste d'enfants dont p est le pointeur sur le père

*int existefils (Pnoeud p , char * nom)*

II-7 *Facultatif :* Ecrire une fonction qui retourne le pointeur sur le père de l'enfant nom dans l'arbre p .

*Pnoeud pere (Pnoeud p, char * nom)*

II-8 *Facultatif* **:** Ecrire une procédure qui affiche la *liste des ascendants de nom*, c-a-d père, grand-père,…

Ecrire une procédure de création d'une nouvelle personne d'adresse p ayant 1 nom, sans descendant ni frère

Ecrire une procédure *d'insertion du nouveau né nom* à la fin de la liste des enfants dont p est le pointeur sur aîné.

Une naissance vient de se produire et on désire inscrire ce nouvel enfant dans l'arbre : écrire la procédure.

UVHC-ISTV

Licence 2 Info & Math semestre 3 ALGO3
Fiche de TD 10b d'Informatique

Exercice 0 Rappels sur les arbres

Un arbre binaire peut se représenter sous forme graphique ou sous forme parenthésée.
Par exemple A(B(C,D),E(,F)) signifie un arbre avec 5 nœuds où B et E sont les 2 fils de A,

1) Représenter graphiquement 10(12(5,13(,11)),17(16,19(18,21(20))))
2) Ecrire la procédure d'affichage sous forme parenthésée d'un arbre binaire d'entiers.
3) Transformer la procédure précédente en supprimant un appel récursif sans pile.
4) Ecrire une fonction qui vérifie qu'un arbre binaire n'est pas dégénéré (il l'est si aucun nœud ne possède plus d'un fils).
5) Proposer une solution itérative à la solution précédente.

Sujet arbres de Recherche équilibrés

Rappels du cours

arbre binaire équilibré = arbre 'bien équilibré', lorsque chacun de ses noeuds possède un sous-arbre à gauche et un sous-arbre à droite ayant le même nombre de noeuds à une unité près.
arbre équilibré en profondeur ou H-équilibré= arbre dont la profondeur (la hauteur) des sous-arbres ne varie que d'une unité au maximum.

Exercice 1

7) Ecrire une fonction qui permet de savoir si un arbre est équilibré en profondeur (hauteur).
8) Ecrire une fonction qui permet de savoir si un arbre est bien équilibré.
9) Ecrire une fonction qui permet de savoir si un arbre est ordonné de manière croissante (ABR).

Exercice 2. Complexité de la Recherche

2.1 : Représenter graphiquement la structure des 4 ABR construits par les insertions suivantes:

arbre A : tataratata, tata, tete, titi, toto, turlututu, tutu

arbre B :tutu, turlututu, toto, titi, tete, tata, tataratata

arbre C :titi, tataratata, tete, turlututu, toto, tutu, tata

arbre D : tata, tataratata, titi, tete, turlututu, toto, tutu

2.2: Comparer la recherche d'une valeur dans ces arbres binaires en calculant le nombre moyen de comparaisons.

Exercice 3 : *Facultatif* Ajout et Construction d'un arbre binaire équilibré

Introduits dans les années 60 par Adelson-Velski et Landis, ces ABR sont H-équilibrés (ou en profondeur). On mémorise un facteur d'équilibrage **bal** (profondeur du sous-arbre droit diminuée de celle du sous-arbre gauche) égal à 0,1,-1.

```
typedef struct elem {int val, bal;  struct elem *fg, *fd;}Noeud;
typedef Noeud * Pnoeud;
```

Construire à la main un arbre binaire équilibré en indiquant la valeur de BAL à chaque fois :
10 17 8 20 25
puis un autre avec 10 17 8 20 15 25

UVHC-ISTV

Licence 2 Info & Math semestre 3 ALGO3
Fiche IV d'exercices supplémentaires d'Informatique

Exercice I : Construction d'un arbre binaire

```
Pnœud   arbre_be (  int n )
        {       int   nb_g,  nb_d,  nb_GplusD = n-1; Pnœud p; if (n == 0)   return  NULL;
                nb_g = nb_GplusD / 2;   nb_d = nb_GplusD - nb_g;
                p = ( Pnœud ) malloc (sizeof (Nœud) );
                p->fg = arbre_be ( nb_g );
                scanf ( «%d »,  &p->num);
                p->fd  =  arbre_be ( nb_d );
                return p;
        }
```

11. Définir les types ***Noeud*** et ***Pnoeud***
12. Dérouler cet algorithme sur les valeurs 5 8 13 15 20 avec n =5
13. Dérouler cet algorithme sur les valeurs 20 15 13 8 5 avec n =5
14. Dérouler cet algorithme sur les valeurs 8 13 5 20 15 avec n =5
15. Qu'en conclure ?

Exercice II : Dérouler les 2 algorithmes suivants sur un exemple de 5 noeuds

```
void parcours1 (Pnoeud racine)
{  Pnœud t;
   Pile p = empiler (racine, p);
   while ( !vide(p) )      {       p = depiler (& t, p);   printf (« %d »,  t-> num );
                                    if ( t->fd != NULL) p = empiler ( t->fd, p );
                                    if ( t->fg != NULL) p = empiler ( t->fg, p );
                           }
}
void  parcours2 (Pnoeud racine, int dec )
        {   if ( racine != NULL )
                           {       printf   (« %* d »,  dec, racine->val );      /*  cf poly de C page 22 ex 4  */
                                   parcours2 ( racine->fg , dec + 6);
                                   parcours2 ( racine->fd , dec + 6);
                           }
        }
```

Exercice III Arbres Généalogiques Ascendants (AGA)

Soit l'arbre binaire x de type ***Aga*** défini de la façon suivante :

```
typedef struct  {   char nom[10], prenom[10] ;  int age ; } Individu ;
typedef struct nœud { Individu info ;   struct nœud *pere , *mere ;  }   Element ;
typedef Element *Aga ;
```

(a) Ecrire une fonction récursive qui retourne le nombre d'individus dont le père et la mère sont tous les deux connus et représentés dans l'arbre x : ***int deux_parents_connus (Aga x)***

(b) Ecrire une fonction récursive qui renvoie le nom et le prénom de l'*individu* le plus âgé appartenant à l'arbre x supposé non vide : ***Individu Le_plus_age (Aga x)***

(c) On veut écrire une fonction qui renvoie le nombre de nœuds contenus dans le sous arbre de x dont la racine est l'individu z. Lorsque ce dernier n'existe pas dans x, le résultat renvoyé est 0. Pour cela, on doit écrire deux fonctions : la première doit rechercher l'individu z dans x, alors que la deuxième calcule le nombre de nœuds dans le sous arbre xz : **int rech_nb_nœud (Aga x, Individu z)**, et ***int nb_Nœud (Aga xz)***

(d) On désire afficher la plus longue séquence de parents de l'*Aga x*.

Ceci revient tout simplement à calculer pour chaque noeud la hauteur des deux sous-arbres et à stocker l'information «quel est le chemin le plus long père ou mère ? » dans un champ supplémentaire de la structure ***Individu*** appelé «***plus_long*** ». Ce champ vaut 1 si le nœud «*pere*» a une hauteur supérieure ou égale au nœud «*mere* » et 0 si non. Une fois que les champs «***plus_long*** » ont été mis à jour par une fonction ***hauteur_modif*** , il suffit de descendre à partir de la racine de x à travers le chemin le «*plus long* ».

Après avoir donné la nouvelle structure ***Individu*** modifiée, puis la fonction récursive ***hauteur_modif ,*** donner la fonction qui affiche la plus longue séquence d'individus dans l'arbre x, à partir de la racine de x :

int hauteur_modif (Aga x) et ***void affiche_plus_long (Aga x***)

Exercice IV Liste chaînée Monodirectionnelle

Dans cet exercice on utilisera le type Liste défini de la façon suivante :

typedef struct cellule {int val ; struct cellule *suiv ; } Cellule ;

typedef Cellule *Liste ;

(a) Ecrire une fonction qui renvoie 1 si la *liste p* est triée dans l'ordre croissant et 0 dans le cas contraire :

int test_croissant (Liste p)

(b) Ecrire une fonction qui supprime d'une liste p triée dans l'ordre croissant les éléments appartenant à l'intervalle [deb, fin]:

Liste supprime_intervalle(int deb, int fin, Liste p)

Quelle est la complexité de la fonction en fonction du nombre n d'éléments de p ?

(c) Ecrire une fonction qui crée une nouvelle liste ne contenant que les éléments positifs de la liste p :

Liste positifs (Liste p)

(d) Ecrire une procédure qui éclate la liste p en deux sous listes ***les_pos*** et ***les_neg*** qui correspondent respectivement

- à la liste contenant tous les éléments positifs ou nuls de la liste p
- et celle contenant tous les éléments négatifs de p :

void eclate (Liste p, Liste * les_pos , Liste * les_neg)

UVHC-ISTV

Licence 2 Info & math semestre 4 && IUP GMI sem 2 : ALGO4
Dernière Fiche de TD d'Algorithmique (sur les graphes)

```
#define MAXS 50
typedef   int  Matadj [MAXS] [MAXS] ; /*Nom sommet = nombre entier entre 0 et nbs-1 */
typedef  struct  { int nbs ;  Matadj  M ; } Mgraphe ;
typedef  struct nd { int s; struct nd *suiv ; }  Sommet ;
typedef  Sommet  * Psommet ;
typedef   Psommet  Tabadj [MAXS] ;
typedef  struct  { int nbs ;  Tabadj  T ; }  Lgraphe ;
```

Exercice 1

1) Prendre un exemple de graphe orienté et le représenter sous la forme matricielle (type ***Mgraphe***) : Ecrire le sous-programme de création de la matrice d'adjacence. Que se passe-t-il si le graphe n'est pas orienté ?

2) Ecrire les algorithmes d'insertion et suppression d'un arc (x, y) dans un graphe orienté représenté par

- Matrice d'adjacence ***Mgraphe inser (Mgraphe G , int x, int y)***
- Liste d'adjacence ***Lgraphe inser (Lgraphe G , int x, int y)***

3) Ecrire les algorithmes de transformation de la représentation d'un graphe orienté ;

- Matrice d'adjacence en liste d'adjacence ***Lgraphe transfo (Mgraphe G)***
- Liste d'adjacence en Matrice d'adjacence ***Mgraphe transfo (Lgraphe G)***

4) Ecrire le programme principal qui va lire un graphe, insérer un arc, le transformer, etc…

5) Ecrire le sous-programme de création de la matrice d'incidence à partir de la matrice d'adjacence (cf les définitions du cours).

Exercice 2

Prendre un exemple de graphe orienté et le représenter sous la forme matricielle (type ***Mgraphe***) puis sous forme de la liste des successeurs (type ***Lgraphe***) et enfin avec la liste des prédécesseurs.
Que se passe-t-il si le graphe n'est pas orienté ?

Exercice 3

Reprendre les 2 algorithmes de parcours d'un graphe en profondeur et en largeur (cf cours diapo 178 et 187) et les dérouler sur un exemple.

Exercice 4

Un chemin est une suite d'arcs (a_1, a_2, ..., a_p) telle que pour chaque arc $a_i = (s_{i-1}, s_i)$, l'extrémité s_i de a_i est égale à l'origine de a_{i+1}. La longueur d'un chemin est égale au nombre d'arcs. Pour une matrice d'adjacence M, les éléments de M^k sont égaux au nombre de chemins distincts de longueur k entre tout couple de sommets du graphe (cf poly page 32).

1) Ecrire l'algorithme permettant de déterminer le nombre de chemins distincts de longueur k entre tout couple de sommets du graphe.
2) M étant une matrice booléenne, transformer l'algorithme précédent pour trouver l'existence d'un chemin entre tout couple de sommets (algorithme de Warshal).
3) Evaluer la complexité de cet algorithme.
4) Soit un graphe orienté pondéré par les distances entre chaque couple de sommets. Ecrire l'algorithme permettant de déterminer la distance minimale du chemin de longueur k entre tout couple de sommets du graphe. Evaluer la complexité de cet algorithme.
5) Un graphe orienté est dit fortement connexe si tout couple de sommets est relié par un chemin. En utilisant l'algorithme de Warshal (question 2), proposer un algorithme permettant de savoir si un graphe est fortement connexe.

UVHC-ISTV

Licence 2 Info & math semestre 4 && IUP GMI sem 2 : ALGO4
Dernière Fiche de TD d'Algorithmique (sur les graphes)

```
#define MAXS 50
typedef   int  Matadj [MAXS] [MAXS] ; /*Nom sommet = nombre entier entre 0 et nbs-1 */
typedef  struct  { int nbs ;  Matadj  M ; } Mgraphe ;
typedef  struct nd { int s; struct nd *suiv ; }  Sommet ;
typedef  Sommet  * Psommet ;
typedef   Psommet  Tabadj [MAXS] ;
typedef  struct  { int nbs ;  Tabadj  T ; }  Lgraphe ;
```

Exercice 1

1) Prendre un exemple de graphe orienté et le représenter sous la forme matricielle (type ***Mgraphe***) : Ecrire le sous-programme de création de la matrice d'adjacence. Que se passe-t-il si le graphe n'est pas orienté ?

2) Ecrire les algorithmes d'insertion et suppression d'un arc (x, y) dans un graphe orienté représenté par

- Matrice d'adjacence ***Mgraphe inser (Mgraphe G , int x, int y)***
- Liste d'adjacence ***Lgraphe inser (Lgraphe G , int x, int y)***

3) Ecrire les algorithmes de transformation de la représentation d'un graphe orienté ;

- Matrice d'adjacence en liste d'adjacence ***Lgraphe transfo (Mgraphe G)***
- Liste d'adjacence en Matrice d'adjacence ***Mgraphe transfo (Lgraphe G)***

4) Ecrire le programme principal qui va lire un graphe, insérer un arc, le transformer, etc…

5) Ecrire le sous-programme de création de la matrice d'incidence à partir de la matrice d'adjacence (cf les définitions du cours).

Exercice 2

Prendre un exemple de graphe orienté et le représenter sous la forme matricielle (type ***Mgraphe***) puis sous forme de la liste des successeurs (type ***Lgraphe***) et enfin avec la liste des prédécesseurs.
Que se passe-t-il si le graphe n'est pas orienté ?

Exercice 3

Reprendre les 2 algorithmes de parcours d'un graphe en profondeur et en largeur (cf cours diapo 178 et 187) et les dérouler sur un exemple.

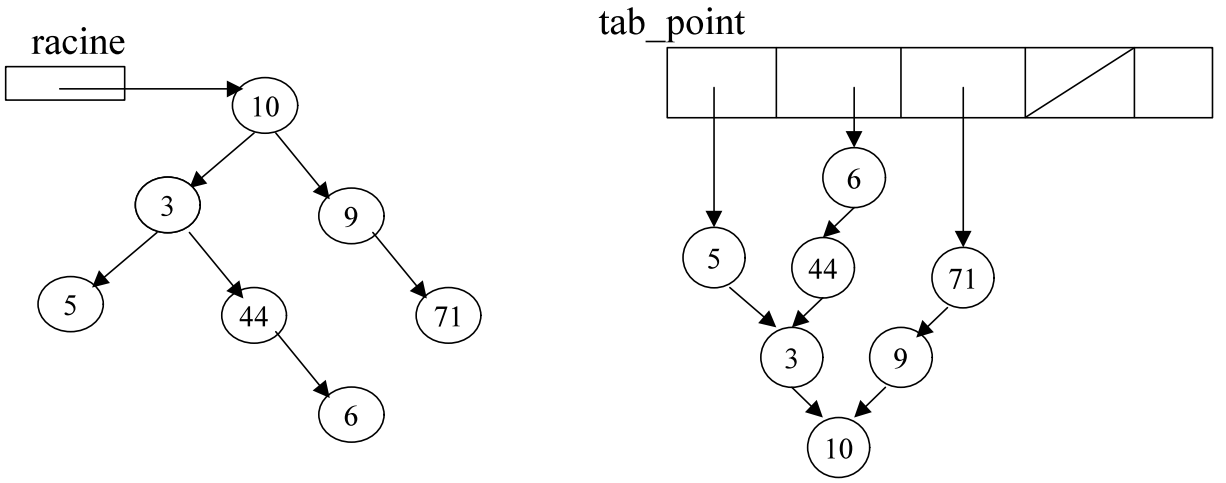
Exercice 4

Un chemin est une suite d'arcs (a_1, a_2, ..., a_p) telle que pour chaque arc $a_i = (s_{i-1}, s_i)$, l'extrémité s_i de a_i est égale à l'origine de a_{i+1}. La longueur d'un chemin est égale au nombre d'arcs. Pour une matrice d'adjacence M, les éléments de M^k sont égaux au nombre de chemins distincts de longueur k entre tout couple de sommets du graphe (cf poly page 32).

1. Ecrire l'algorithme permettant de déterminer le nombre de chemins distincts de longueur k entre tout couple de sommets du graphe.
2. M étant une matrice booléenne, transformer l'algorithme précédent pour trouver l'existence d'un chemin entre tout couple de sommets (algorithme de Warshal).
3. Evaluer la complexité de cet algorithme.
4. Soit un graphe orienté pondéré par les distances entre chaque couple de sommets. Ecrire l'algorithme permettant de déterminer la distance minimale du chemin de longueur k entre tout couple de sommets du graphe. Evaluer la complexité de cet algorithme.
5. Un graphe orienté est dit fortement connexe si tout couple de sommets est relié par un chemin. En utilisant l'algorithme de Warshal (question 2), proposer un algorithme permettant de savoir si un graphe est fortement connexe.

Exercice 2 : Arbre binaire inversé

Dans certaines applications il est intéressant d'accéder directement aux feuilles de l'arbre binaire. Une façon intéressante de représenter l'arbre dans ce cas est d'utiliser un tableau de pointeurs pour accéder directement aux feuilles de l'arbre. La figure suivante illustre cette solution :



1)Arbre binaire 2)Arbre binaire inversé

Figure 2

Le type de chaque nœud est :
typedef struct elem2 { int val ;int nbr_fils, struct elem2 * pnt_père } Noeud2 ;
Le champ **val** contient la valeur du nœud, le champ **nbr_fils** contient le nombre de fils du nœud (entre 0 et 2 pour les arbres binaires), et le champ **pnt_père** est un pointeur sur le père du nœud.
Questions :

a) Donner le type **tabP** de la variable **tab_point** de la représentation 2).
b) Ecrire une fonction somme_feuilles qui calcule la somme des feuilles. Comparer la complexité de cette fonction par rapport à la complexité de la fonction somme_feuilles qui aurait utilisé la représentation 1) de la figure précédente.
c) Ecrire une fonction hauteur qui calcule la hauteur de l'arbre contenu dans le tableau **tab_pointeur.**
d) Ecrire une fonction **ajout** qui permet de rajouter une valeur entière comme fils à un nœud de l'arbre.
L'entête de la fonction est **int ajout (tabP tab_pointeur, int val, int val_pere)**
Val est la valeur à ajouter et val_pere est la valeur du nœud père qui doit exister dans l'arbre.

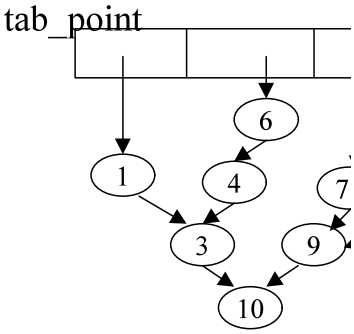
Exemple : ajout (tab_pnt, 8, 9) rajoute une feuille avec la valeur 8 au nœud contenant la valeur 9 qui devient son père.
Dans cette question on s'autorise de passer plusieurs fois par le même nœud. Si le nœud père n'existe pas, ou si ce dernier existe mais a déjà deux fils le résultat de la fonction sera 0, si non le résultat sera 1.

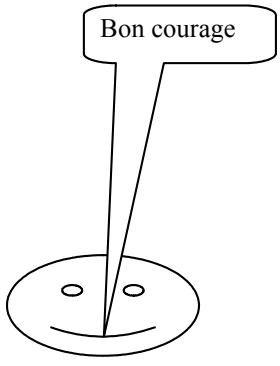
e) Trouver une solution pour éviter de passer plusieurs fois par le même nœud (vous pouvez redéfinir le type noeud2).

Ecrire une fonction **ajout_optimisé** qui réalise la même fonction et qui a les mêmes paramètres que la fonction « ajout » et qui évite de passer (comparer) plusieurs fois la même valeur.

f) Comment peut on adapter la structure d'arbre binaire inversé si on désire utiliser cette représentation pour les arbres n-aires ?

Figure 3 : Arbre de la figure 2 après ajout de la feuille 8 au nœud 9





DEUG 2 MASS && MIAS && STPI
DS d'Informatique (29/03/2002 - durée 2h sans document)

Exercice 1

Soit une liste chaînée non ordonnée de personnes repérées par un nom et un prénom.
typedef struct { char nom [20], pren {20} Personne ;
*typedef struct s { Personne personne ; struct s * suiv ; } Elem ;*
*typedef Elem * LP ;*

5) Ecrire le sous-programme de création d'une liste chaînée de n personnes.
6) Ecrire le sous-programme d'affichage de toutes les personnes dont le nom commence par un caractère (paramètre).
7) Ecrire le sous-programme de suppression d'une personne repérée par son nom.
8) Ecrire le programme principal créant la liste avec n lu, l'affichant, supprimant un nom lu, et enfin la réaffichant .

Exercice 2

Exercice 3

Dérouler les 2 algorithmes suivants sur un exemple de 5 noeuds

```
void inconnu1 ( Arbre1  racine)
{  Arbre1  t;
   Pile p = empiler (racine, p);
   while ( !vide(p) )      {      p = depiler (& t, p);   printf (« %d »,  t-> val1 );
                                   if ( t->fd != NULL) p = empiler ( t->fd, p );
                                   if ( t->fg != NULL) p = empiler ( t->fg, p );
                           }
}
void  inconnu2 ( Arbre2  racine, int dec )
        {   if ( racine != NULL )
                           {      printf   (« %* d »,  dec, racine->val2 );
                                  inconnu2 ( racine->fg , dec + 6);
                                  inconnu2 ( racine->fd , dec + 6);
                           }
        }
```