

Exercices de base de Programmation – Arbre Programmatique

Exercice 1

- Ecrire la fonction puissance qui renvoie la valeur de a^b , a et b entiers positifs.
- Ecrire la fonction factoriel qui renvoie le factoriel de a ($a!$), a entier positif étant passé en paramètre.
- Réécrire la fonction factoriel, qui sera alors de type void et qui utilisera les paramètres adéquats.

Exercice 2

- Ecrire une fonction qui saisit une liste d'entiers strictement positifs, (la saisie étant terminée lorsque la valeur saisie est négative ou nulle) et qui renvoie la valeur maximale saisie.

Exercice 3

Votre programme reproduit le jeu où l'utilisateur doit deviner en un nombre d'essais maximum un nombre déterminé de manière aléatoire.

Le programme se divise donc en deux étapes successives:

- le tirage aléatoire du nombre à deviner, appelé N_secret .
- la recherche par l'utilisateur de N_secret .

Tirage aléatoire:

Le tirage aléatoire de N_secret compris dans $[min1,max1]$ ou $[min2,max2]$:
 $N_secret \in [min1,max1] \cup [min2,max2]$.

Les valeurs $min1, min2, max1$ et $max2$ devront être saisies et le programme réitère le tirage aléatoire tant que N_secret n'appartient pas à l'un des deux intervalles.

Recherche par l'utilisateur de N_secret :

L'utilisateur doit saisir dans un premier temps le nombre maximum d'essais.

Dans un second temps, jusqu'à ce que la solution soit trouvée ou que le nombre maximum d'essais soit atteint :

- l'utilisateur propose un nombre,
- le programme lui indique alors
 - si le nombre proposé est inférieur, supérieur ou égal à N_secret
 - le nombre d'essais restant.

Enfin, le programme informe l'utilisateur s'il a gagné ou perdu.

1 Révision de notions de langage C - Chaînes de caractères

1.1 Initialisation d'une chaîne de caractères

Ecrire une fonction de type void qui initialise une chaîne de caractères passée en paramètre. La longueur de la chaîne est également passée en paramètre. La chaîne doit être initialisée à "NON SAISIE".

1.2 Saisie d'une chaîne de caractères

Ecrire une fonction de type void qui assure la saisie d'une chaîne de caractères passée en paramètre. La longueur de la chaîne est également passée en paramètre.

1.3 Affichage d'une chaîne de caractères

Ecrire une fonction de type void qui assure l'affichage d'une chaîne de caractères passée en paramètre.

1.4 Analyse du contenu d'une chaîne de caractères

Ecrire une fonction de type int qui renvoie 1 si tous les caractères de la chaîne passée en paramètre sont des lettres 'a'...'z' ou 'A'...'Z'. La fonction renvoie 0 sinon. La longueur de la chaîne est également passée en paramètre.

1.5 Tableau de chaînes de caractères

Ecrire une fonction de type void qui contient un tableau de 10 chaînes de 50 caractères en variable locale. Cette fonction implante un menu qui effectue

- l'initialisation des chaînes du tableau,
- la saisie d'une chaîne, dont l'indice dans le tableau est saisi par l'utilisateur,
- l'affichage d'une chaîne, dont l'indice dans le tableau est saisi par l'utilisateur,
- la vérification que la chaîne ne contient que des caractères, dont l'indice dans le tableau est saisi par l'utilisateur,
- la transformation miroir de la chaîne, dont le numéro dans le tableau est saisi par l'utilisateur. Par exemple, la chaîne "FOOT" devient "TOOF".

2 Algorithmes de tri

2.1 Tri élémentaire

Ecrire une fonction qui utilise un tableau à 3 entiers. La fonction saisit les 3 éléments du tableau, les trie par ordre croissant et affiche le tableau ainsi trié.

2.2 Tri par sélection

Ecrire une fonction qui utilise un tableau à N entiers. La fonction saisit N éléments du tableau, appelle une fonction de tri par ordre croissant et affiche le tableau ainsi trié.

La fonction de tri utilisée est le tri par sélection. On procède de la manière suivante. On commence par rechercher l'élément de plus petite valeur du tableau pour l'échanger avec celui en première position. Puis on recherche l'élément ayant la deuxième petite valeur pour l'échanger avec celui en deuxième position et l'on continue jusqu'à ce que le tableau soit entièrement trié. L'implantation est la suivante. Pour tout i compris entre 1 et N, on échange $t[i]$ avec l'élément de valeur minimale parmi $t[i+1] \dots t[N]$.

Exemple

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]
U	N	E	P	H	R	A
A	N	E	P	H	R	U
A	E	N	P	H	R	U
A	E	H	P	N	R	U
A	E	H	N	P	R	U

TAB. 1 – Tri par sélection.

2.3 Tri par insertion

Ecrire une fonction qui utilise un tableau à N entiers. La fonction saisit N éléments du tableau, appelle une fonction de tri par ordre croissant et affiche le tableau ainsi trié.

La fonction de tri utilisée est le tri par insertion. On procède de la manière suivante. On considère les éléments les uns après les autres en insérant chacun à sa place parmi ceux déjà triés (et gardés comme tels). Pour insérer l'élément couramment considéré, on déplace simplement les éléments qui lui sont supérieurs un cran vers la droite et on l'insère dans la place laissée vacante. Pour chaque i compris entre 1 et N, les éléments $t[0], \dots, t[i]$ sont mis en ordre par insertion de $t[i]$ à sa place dans la liste déjà triée des éléments de $t[0], \dots, t[i-1]$.

Exemple

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]
U	N	E	P	H	R	A
N	U	E	P	H	R	A
E	N	U	P	H	R	A
E	N	P	U	H	R	A
E	H	N	P	U	R	A
E	H	N	P	R	U	A
A	E	H	N	P	R	U

TAB. 2 – Tri par insertion.

2.4 Tri par bulles

L'algorithme de tri est le suivant : on effectue autant de passes que nécessaires en échangeant les éléments adjacents s'ils sont dans un mauvais ordre relatif.

Exemple

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]
U	N	E	P	H	R	A
N	U	E	P	H	R	A
E	U	N	P	H	R	A
A	U	N	P	H	R	E
A	U	N	P	H	R	E
A	N	U	P	H	R	E
A	H	U	P	N	R	E
A	E	U	P	N	R	H
A	E	P	U	N	R	H
A	E	N	U	P	R	H
A	E	H	U	P	R	N
A	E	H	P	U	R	N
A	E	H	N	U	R	P
A	E	H	N	P	R	U

TAB. 3 – Tri par bulles.

2.5 Comparaison des algorithmes de tri

Mesurer et comparer la complexité algorithmique des trois algorithmes de tri.

3 Gestion d'une pile

La structure de données à accès restreinte la plus importante est la pile. Seules deux opérations sont permises, d'une part empiler (insérer au sommet) et dépiler (supprimer au sommet) un élément. Exemple d'utilisation de la pile pour évaluer une expression arithmétique :

$$5 * (((9 + 8) * (4 * 6)) + 7). \quad (1)$$

Une pile constitue un mécanisme idéal pour mémoriser les résultats intermédiaires d'un calcul. L'exemple peut être calculé par les appels :

```
empiler (5);  
empiler (9);  
empiler (8);  
empiler (dépiler () + dépiler ());  
empiler (4);  
empiler (6);  
empiler (dépiler () * dépiler ());  
empiler (dépiler () * dépiler ());  
empiler (7);  
empiler (dépiler () + dépiler ());  
empiler (dépiler () * dépiler ());
```

```
printf ("%d", dépiler ( ) )
```

L'expression précédente peut s'exprimer alors en écriture post-fixée comme

$$598 + 46 * *7 + * \quad (2)$$

3.1 Saisie d'une expression

Ecrire une fonction qui saisit dans une chaîne de caractères une expression arithmétique classique (voir équation (1)). Ce tableau ne peut contenir que les chiffres ainsi que les caractères '(', ')', ',' et '*'. Ce test doit être effectué lors de la saisie de chaque caractère. Cette fonction a deux paramètres la chaîne et le nombre de caractères saisis.

3.2 Conversion en écriture post-fixée

Ecrire une fonction de type void qui traduit la chaîne de caractères contenant une expression valide écrite en pré-fixé (voir equation(1)) vers une expression valide en post-fixé (voir equation(2)). Cette fonction a 4 paramètres, à savoir la chaîne contenant l'expression en écriture pré-fixée, son nombre de caractères, la chaîne contenant l'expression en écriture post-fixée ainsi que son nombre de caractères.

3.3 Evaluation d'une expression en écriture post-fixée

Ecrire une fonction de type int qui évalue l'expression écrite en post-fixé contenue dans une chaîne de caractères. Cette fonction a deux paramètres la chaîne et le nombre de caractères de l'expression.

3.4 Implantation de la pile avec des tableaux

Il est indispensable de garder deux indices en variable globale, pour le début de la pile (début) et pour la queue de la pile (fin). Si début = fin, alors la pile est vide. Ecrire les fonctions InitialiserPile (char * pile), Dépiler (char * pile), Empiler (char valeur, char *pile).

4 Implantation de la pile avec une liste chaînée

Cette implantation est plus souple que l'implantation par tableaux, car elle ne nécessite pas de connaître a priori le nombre d'éléments présents dans la pile.

Une liste chaînée est une liste de noeuds liés entre eux (voir TP2).

4.1 Structure pour recevoir une liste chaînée

Créer la structure de données ayant deux champs : la valeur de l'élément dans la pile et le pointeur pointant sur l'élément suivant.

4.2 Initialisation de la pile

Ecrire la fonction qui initialise la pile.

4.3 Empiler

Ecrire la fonction qui empile un élément dans la pile, à savoir l'insertion d'un noeud dans la liste chaînée après la tête de la liste.

4.4 Dépiler

Ecrire la fonction qui dépile un élément de la pile, à savoir l'extraction du noeud dans la liste chaînée après la tête de la liste.

4.5 Lister

Ecrire la fonction qui liste et affiche les éléments de la pile.

5 Arbre binaire de recherche (optionnel)

On appelle arbre binaire de recherche (ABR) une structure de données utile pour ranger des valeurs ordonnées. Chaque nœud de l'arbre porte une valeur (ici nous utiliserons des nombres entiers), et possède deux nœuds fils, à droite et à gauche. Le fils de gauche, s'il existe, porte toujours une valeur inférieure à celle de son père. Celui de droite une valeur supérieure.

Tous les nœuds à gauche d'un nœud valant x portent des valeurs inférieures à x . Tous ceux à droite des valeurs supérieures à x .

Les opérations de base sur un ABR sont les suivantes :

- `insérer(nœud *n, int x)` : range la valeur x dans l'arbre. Pour cela, un nouveau nœud est créé, placé à droite ou gauche d'un nœud de l'arbre existant.
- `existe(nœud *n, int x)` : vérifie si la valeur x se trouve dans l'arbre. Retourne 1 ou 0 selon le cas.

5.1 Représentation

Soit T un arbre vide. Faire un schéma représentant T après l'insertion des nombres suivants : 10, 5, 12, 4, 11 (dans cet ordre).

Faire un autre schéma représentant T après l'insertion des nombres suivants : 8,15,2,10, 5, 7,12, 12,44, 1,3 (dans cet ordre).

5.2 Définition de la structure nœud

Définir en C la structure nœud.

5.3 Insérer

Écrire la fonction `Insérer` qui insère dans l'arbre binaire passé en paramètre l'entier passé en paramètre.

5.4 Existe

Écrire la fonction `Existe` qui renvoie 1 (0) si l'entier passé en paramètre est contenu (sinon) dans l'arbre binaire passé en paramètre.

5.5 Affichage du contenu de l'arbre

Écrire la fonction `afficher(nœud *n)` qui affiche tout le contenu de l'arbre, suivant l'ordre croissant à partir du nœud passé en paramètre.

Écrire la même fonction affiche tout le contenu de l'arbre, suivant l'ordre décroissant.

1 Instructions pour les Travaux Pratiques en Langage C

- LES ENONCES DE TP SONT DISTRIBUES EN TD.
- L'ETUDIANT DOIT ARRIVER EN SEANCES DE TP AVEC L'ANALYSE DU PROBLEME SOUS LA FORME D'ARBRES PROGRAMMATIQUES . CES ARBRES SERONT CORRIGES PAR L'ENSEIGNANT EN DEBUT DE SEANCES DE TP. L'ETUDIANT QUI NE RESPECTE PAS CETTE REGLE NE POURRA PAS TRAVAILLER SUR LA MACHINE TANT QUE LES ARBRES NE SONT PAS CORRIGES.
- L'ETUDIANT PEUT VENIR EN TP AVEC UNE EBAUCHE MANUSCRITE DE PROGRAMME C. LES LISTINGS IMPRIMES OU SOUS FORME DE FICHIERS SONT INTERDITS.
- L'OUTIL DE DEVELOPPEMENT GRATUIT DEV C++ EST DISPONIBLE A L'ADRESSE [http ://telecharger.01net.com/windows/Programmation/langage/](http://telecharger.01net.com/windows/Programmation/langage/). VOUS POUVEZ L'UTILISER A DOMICILE POUR VOUS ENTRAINER.
- L'ENSEIGNANT SE RESERVE LE DROIT DE MODIFIER UNE PARTIE DE L'ENONCE EN DEBUT DE SEANCE DE TP.
- A LA FIN DE LA SEANCE, L'ETUDIANT DOIT RENDRE LE PROGRAMME C ET LE COMPTE-RENDU DU TP AVEC LES ARBRES PROGRAMMATIQUES.
- LES TPs SONT CORRIGES MAIS NON NOTES. LA NOTE DE TP SERA UNIQUEMENT LA NOTE DE L'EXAMEN DE TP.

Les enseignants de TP.

2 TP1 : Problèmes de saisie

2.1 Introduction

Le problème à traiter concerne l'acquisition de données. Il s'agit d'associer à une question un domaine de réponses. Plusieurs aspects sont à considérer : comment effectuer une demande, comment obtenir une réponse de l'utilisateur à cette demande et comment valider la réponse de l'utilisateur.

2.2 Notre Problème

2.2.1 Le menu

Il s'agit de réaliser un questionnaire qui se décomposera des rubriques accessibles selon le menu suivant,

- 1 - SAISIE DU NOM
- 2 - SAISIE DU CODE POSTAL
- 3 - AFFICHAGE DES CARACTERISTIQUES DE LA PERSONNE
- 8 - FIN DU PROGRAMME

Dès le lancement du programme, on accède au menu. Pour accéder à une des rubriques possibles, l'utilisateur frappe le numéro correspondant.

2.2.2 Réponses au questionnaire

– Réponse du type alphabétique

Une rubrique est concernée : le nom. Une fois dans cette rubrique, l'utilisateur répond à la question, peut utiliser les flèches gauche et droite pour se déplacer, et valide la saisie par la touche "return".

Le programme doit alors tester si chaque caractère est bien une lettre. Si ce n'est pas le cas, le programme doit alors émettre un message d'erreur et redemander une saisie.

Le nom contiendra au maximum 40 caractères.

– Réponse du type numérique

Une rubrique est concernée : le code postal. Une fois dans cette rubrique, l'utilisateur répond à la question, peut utiliser les flèches gauche et droite pour se déplacer, et valide la saisie par la touche "return". Le programme doit alors tester si chaque caractère est bien un chiffre. Si ce n'est pas le cas, le programme doit alors émettre un message d'erreur et redemander une saisie. Un test de cohérence sera également effectué après la validation. Le code postal doit être compris entre 01000 et 97999. Si la valeur saisie n'est pas cohérente, le programme doit émettre un message d'erreur et demander à nouveau une saisie.

– Affichage des caractéristiques de la personne

Quand l'utilisateur demande cette rubrique, le programme doit afficher toutes les caractéristiques saisies. Si une caractéristique n'a pas été saisie, le programme doit afficher le message "NON SAISI".

2.3 Elaboration du programme

Ecrire un programme en langage C, qui gère les fonctions suivantes :

- Menu : affichage du menu, saisie du choix de la rubrique et appel de la fonction inhérente à chaque rubrique
- saisie_nom : saisie et vérifie le format du nom par l'appel de la fonction format_alphabetique
- format_alphabetique : fonction booléenne qui teste si la chaîne de caractères passée en paramètre est de taille demandée et si elle ne contient que des lettres.
- saisie_codepostal : saisie le code postal, vérifie son format par l'appel de la fonction format_numerique, et vérifie la cohérence par l'appel de la fonction coherence_codepostal

- `format_numerique` : fonction booléenne qui teste si la chaîne de caractères passée en paramètre ne contient que des chiffres.
- `coherence_codepostal` : fonction qui teste si la chaînes de caractères passée en paramètres est cohérente .
Renvoie un code d'erreur.
- `affiche_caracteristique` : fonction qui affiche à l'écran les caractéristiques saisies.

2.4 Contraintes de programmation

- Pour la saisie, utiliser la fonction `scanf ()` .
- Attention à la déclaration des variables qui représentent les caractéristiques de la personne, afin qu'elles puissent être accessibles par les fonctions précitées.
- Ce Tp est long à réaliser. Faites ce TP étape par étape. Ecrivez et validez une rubrique à fois.
- Commencer par l'initialisation des chaînes de caractères ainsi que par l'affichage des chaînes.
- Toutes les variables doivent être locales à des procédures.

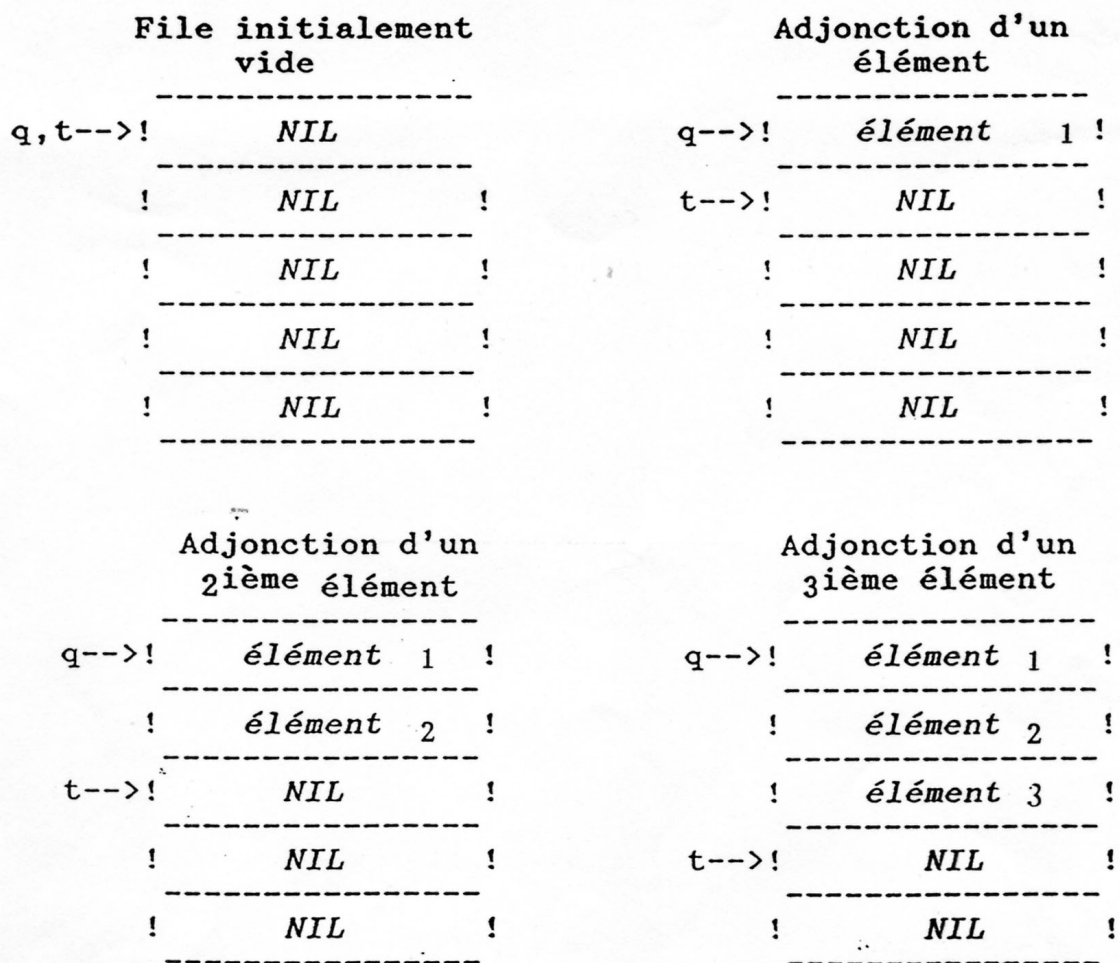
3 TP2 : Gestion d'une File semi-statique

3.1 Introduction

L'objectif est d'implanter une file semi-statique. Nous considérons qu'une file est un tableau de 5 entiers. Un élément de la file est vide quand sa valeur est égale à 0 (NIL sur les figures). Cette structure nécessite deux indices de tableau appelés t et q dans les figures, qui représentent la tête et queue de la file.

Le fonctionnement d'une telle structure est décrite de la manière suivante :

- ajout d'un élément en tête de la file,
- suppression d'un élément en queue de file
- lorsque la file est pleine, il est interdit d'ajouter un élément.



3.2 Elaboration du programme

Toutes les variables doivent être locales. L'utilisateur doit accéder à un menu affichant les différentes fonctions suivantes qui lui sont disponibles

- Adjonction

Cette fonction ajoute un élément en tête de file tout en assurant la mise à jour des indices de tête et de queue. Elle vérifie que la file ne soit pas pleine avant d'ajouter un élément.
- Suppression

Cette fonction supprime un élément en queue de file tout en assurant la mise à jour des indices de tête et de queue. Elle vérifie que la file ne soit pas vide avant de supprimer un élément.

**Suppression d'un
élément**

 ! *NIL* !

 q-->! *élément2* !

 ! *élément3* !

 t-->! *NIL* !

 ! *NIL* !

**Adjonction d'un
élément**

 ! *NIL* !

 q-->! *élément2* !

 ! *élément3* !

 ! *élément4* !

 t-->! *NIL* !

**Suppression d'un
élément**

 ! *NIL* !

 ! *NIL* !

 q-->! *élément3* !

 ! *élément4* !

 t-->! *NIL* !

**Adjonction d'un
élément**

 t-->! *NIL* !

 ! *NIL* !

 q-->! *élément3* !

 ! *élément4* !

 ! *élément5* !

**Adjonction d'un
élément**

 ! *élément6* !

 t-->! *NIL* !

 q-->! *élément3* !

 ! *élément4* !

 ! *élément5* !

**Adjonction d'un
élément**

 ! *élément6* !

 ! *élément7* !

 q,t-->! *élément3* !

 ! *élément4* !

 ! *élément5* !

Adjonction d'un élément		Adjonction d'un élément	
	----- ! <i>élément6</i> ! -----		----- ! <i>élément6</i> ! -----
	! <i>élément7</i> ! -----		! <i>élément7</i> ! -----
t-->!	! <i>NUL</i> ! -----		! <i>élément8</i> ! -----
q-->!	! <i>élément4</i> ! -----	q,t-->!	! <i>élément4</i> ! -----
	! <i>élément5</i> ! -----		! <i>élément5</i> ! -----

Le programme devra notamment être testé dans les cas suivants, adjonction d'un élément quand la file est pleine, suppression d'un élément quand la file est vide, adjonction de la valeur 0.

4 TP3 : Pointeurs et structure de données

4.1 Définition de la liste chaînée

Une liste chaînée est une série de noeuds reliés entre eux. Chaque noeud contient deux champs :

- un champ noté element contenant l'élément qui sera de type int,
- un champ noté suivant contenant l'adresse du noeud suivant.

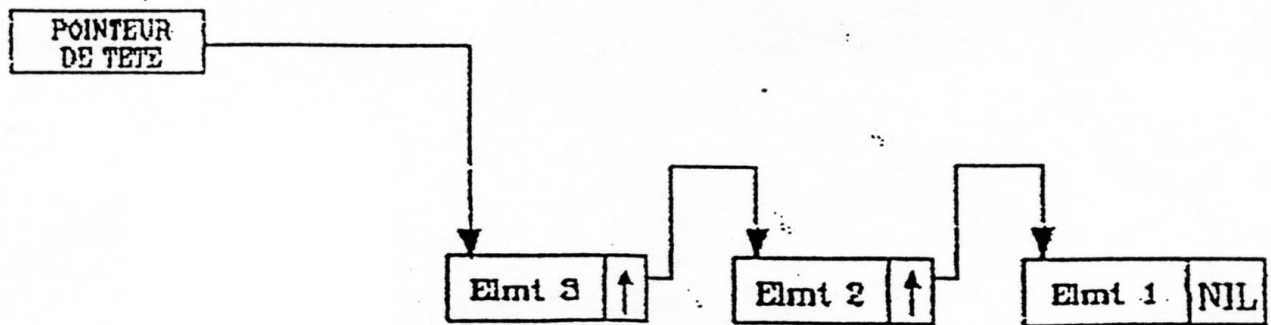


FIG. 1 – Liste chaînée

La liste chaînée est représentée par le pointeur de tête qui indique l'adresse où se trouve le premier noeud de la liste. Quand la liste est vide, le pointeur de tête pointe sur l'adresse 0 (NIL dans la figure 1). Quand la liste n'est pas vide, le champ suivant du dernier noeud de la liste pointe sur l'adresse 0.

4.2 Fonctions demandées

Vous devez réaliser un programme qui permet d'ajouter ou de supprimer un élément d'une liste chaînée de façon dynamique. Pour cela, les fonctions suivantes sont à implanter :

4.2.1 Adjonction

L'adjonction se fait en tête de liste. Un nouveau noeud (noeud contenant l'élément X dans la figure 2) doit être créé dynamiquement et l'élément contenu par ce nouveau noeud doit être saisi. Le champ suivant de ce nouveau noeud doit être affecté à l'adresse du premier noeud (à la place du noeud contenant l'élément n dans la figure 2). On modifie l'adresse contenue par le pointeur de tête qui doit alors pointer sur l'adresse du nouveau noeud (noeud contenant l'élément X dans la figure 2).

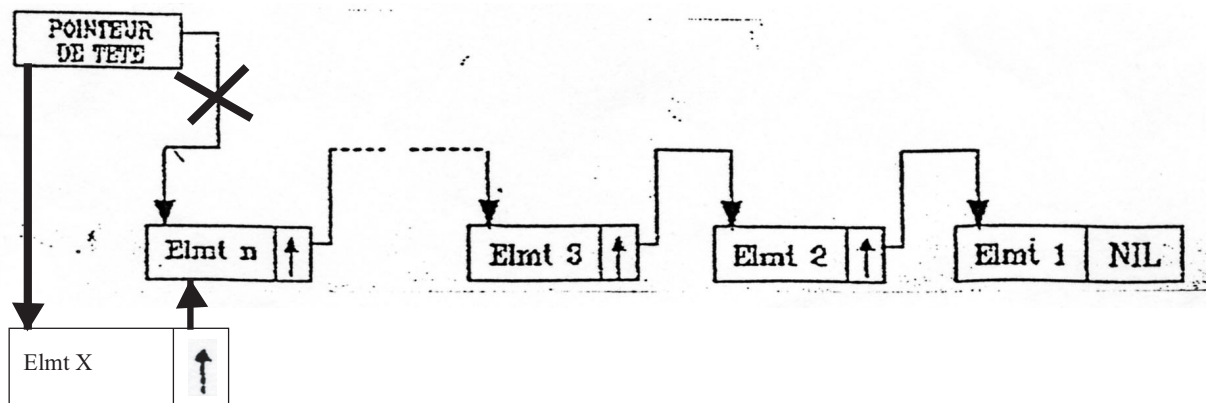


FIG. 2 – Adjonction d'un noeud en tête de liste

4.2.2 Lister

On accède à un noeud par lecture successive des noeuds de la liste et on affiche l'élément contenu dans chacun de ces noeuds. L'accès est donc séquentiel. Pour ce faire, il faudra utiliser un pointeur courant qui accédera successivement aux différents noeuds. Comme le pointeur de tête indique où se trouve le premier élément, le pointeur courant sera initialisé au pointeur de tête. Le parcours se termine lorsque le pointeur courant pointe sur l'adresse 0.

Les fonctions Adjonction et Lister vous sont fournies en annexe afin de vous aider.

4.2.3 Recherche d'un élément

L'utilisateur spécifie la valeur de l'élément correspondant au noeud à rechercher. Si aucun noeud ne contient l'élément considéré, le programme doit indiquer à l'utilisateur que l'élément n'existe pas. Sinon, le programme indique le nombre de fois que l'élément est présent dans la liste chaînée.

4.2.4 Insertion

Les insertions se font à l'intérieur de la liste chaînée. Il faut d'abord que l'utilisateur spécifie la valeur de l'élément correspondant au noeud après lequel il faut insérer un nouveau noeud. Si aucun noeud ne contient l'élément considéré, le programme doit indiquer à l'utilisateur que l'insertion est impossible. Si plusieurs noeuds contiennent le même élément, l'insertion se déroule après le premier noeud contenant l'élément.

Si l'insertion doit s'effectuer, le nouveau noeud doit être créé dynamiquement et l'élément doit être saisi. L'insertion s'effectue en rompant la chaîne de noeud à l'endroit où l'on désire insérer un noeud. Pour cela, le champ suivant du noeud précédent (noeud contenant l'élément 3 sur la figure 3) doit pointer sur l'adresse du nouveau noeud (noeud contenant l'élément X sur la figure 3). Le champ suivant du nouveau noeud (noeud contenant l'élément X sur la figure 3) doit pointer sur l'adresse du noeud successeur (noeud contenant l'élément 2 sur la figure 3).

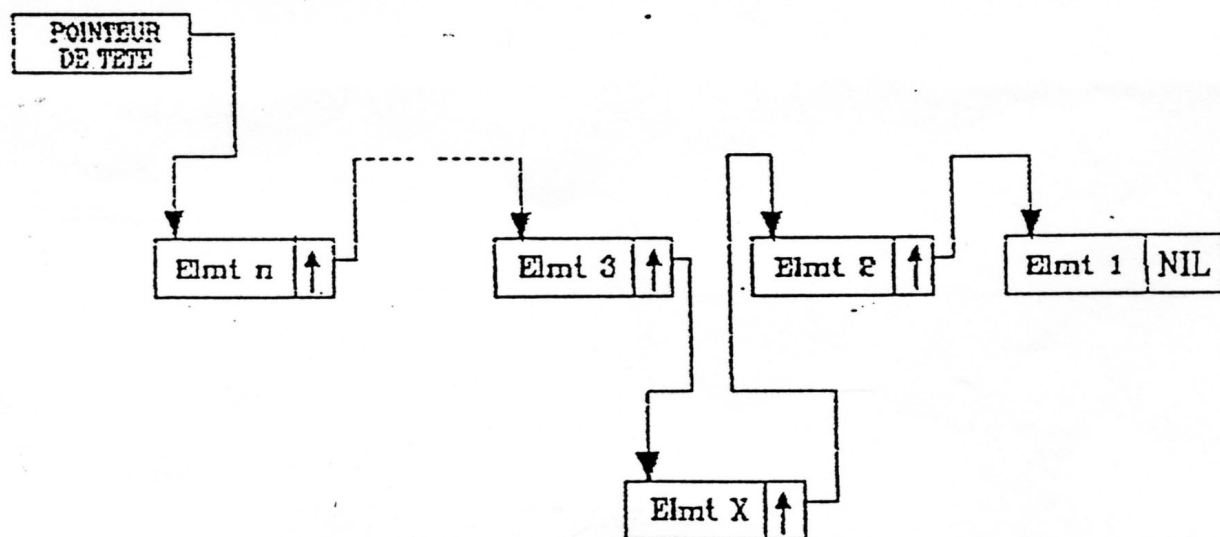


FIG. 3 – Insertion d'un noeud

4.2.5 suppression en tête de liste

La suppression en tête de liste consiste tout simplement à affecter le pointeur de tête sur l'adresse du second noeud de la liste. Il ne faut cependant pas oublier de supprimer en mémoire l'espace pris par le noeud contenant le premier élément.

4.3 Elaboration du programme

L'utilisateur devra accéder à un menu affichant les différentes fonctions disponibles.

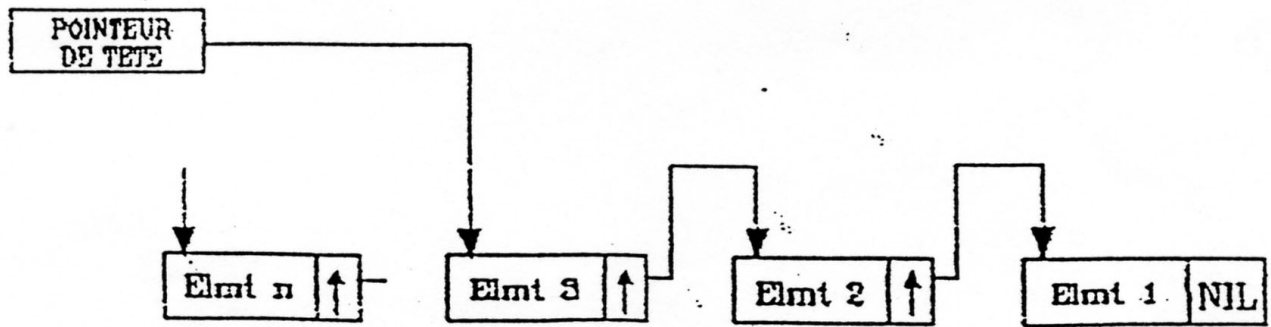


FIG. 4 – Suppression d'un noeud en tête de liste

Le menu se décompose de la manière suivante :

- adjonction d'un élément (fait par le programme en annexe)
- listing de la liste (fait par le programme en annexe)
- recherche d'un élément dans la liste
- insertion d'un élément dans la liste
- suppression d'un élément en tête de liste
- quitter

A noter que pour des raisons de simplicité, le pointeur de tête peut être implanté sous la forme d'une variable globale. Quand l'utilisateur quitte le programme, il est nécessaire de supprimer de la mémoire tous les noeuds de liste chaînée.

Il faut partir du programme mis en annexe ci-joint.

```

/*
BASE DU TP3
GESTION D'UNE LISTE CHAINEE
*/

#include<stdio.h>
#include<conio.h>
#include<malloc.h>

typedef struct champs{int valeur;struct champs *suivant;}element;

void menu();
void init(element *);
void adjonction(element *);
void insertion(element *);
void suptete(element *);
void listing(element *);
void recherche(element *);

void main()
{
    menu();
}

void init(element *ptete)
{
    ptete->suivant=NULL;
}

void menu()
{
    element tete;
    char choix;
    init(&tete);
    do
    {
        printf("\n\n\n\t\t\tPOINTEURS ET STRUCTURES DE DONNEES");
        printf("\n\n\n\t\t1 - Adjonction d'un élément");
        printf("\n\n\t\t2 - Insertion d'un élément");
        printf("\n\n\t\t3 - Suppression d'un élément en tête de liste");
        printf("\n\n\t\t5 - Recherche d'un élément");
        printf("\n\n\t\t6 - Listing de la liste");
        printf("\n\n\t\t7 - Fin du programme\n\n");
        fflush();
        choix=getchar();
        fflush();
        switch(choix)
        {
            case'1':adjonction(&tete);break;
            case'2':insertion(&tete);break;
            case'3':suptete(&tete);break;
            case'5':recherche(&tete);break;
            case'6':listing(&tete);break;
            case'7':break;
            default:break;
        }
    }
    while(choix!='7');
}

//ADJONCTION:->permet l'ajout d'un élément dans la liste
//          Paramètres: ptete->pointeur d'element

```

```

void adjonction(element *ptete)
{
    element *elt;
    elt=(element *)malloc(sizeof(element));          /*alloue un emplacement
mémoire de la taille d'une structure "element"*/
    printf("\n\nIndiquez l'élément à ajouter: ");
    scanf("%d",&elt->valeur);
    fflush();
    elt->suivant=ptete->suivant;
    ptete->suivant=elt;
}

//INSERTION:->permet d'insérer un élément dans la liste
//          Paramètres: ptete->pointeur d'element

void insertion(element *ptete)
{
}

//LISTING:->permet d'afficher la liste
//          Paramètres: ptete->pointeur d'element

void listing(element *ptete)
{
    element *elt;
    elt=ptete->suivant;
    printf("\t\t\tContenu de la file :\n\n");
    if(elt==NULL)
        printf("\t\t\tFile vide");
    else
        while(elt!=NULL)                //affiche tous les
éléments de la file tant qu'on ne pointe pas le pointeur de fin
        {
            printf("\t\t\t%-10d\n",elt->valeur);
            elt=elt->suivant;
        }
    printf("\n\nAppuyez sur une touche pour continuer");
    getchar();
    fflush();
}

//RECHERCHE:->permet de vérifier si un élément est bien présent dans la liste
//          Paramètres: ptete->pointeur d'element

void recherche(element *ptete)
{
}

//SUPTEETE:->permet de supprimer l'élément en tête de liste
//          Paramètres: ptete->pointeur d'element

void supptete(element *ptete)
{
}

```