– Algorithmique – TP N°7 – LES LISTES LINEAIRES CHAINEES –

I- Les enregistrements

I.1- Rappel de cours : Un enregistrement est une structure de données permettant de regrouper dans une seule entité un ensemble de données ***de types différents*** associées à un même et seul objet. Chaque donnée est appelée un champ. Chaque champ est identifié par ***un nom*** qui permet d'y accéder directement et ***un type***. Le type d'un champ peut être simple (***int, float, char, …***) ou structuré (***tableau, matrice, enregistrement***).

Exemple : Pour manipuler les informations d'un étudiant, on peut créer un type **Etudiant** comme suit :

Type Etudiant = **Enregistrement**

Nom : **Chaine**
Prenom : **Chaine**
Age : **Entier**
Moyenne : **Réel**

Fin

Var e1, e2 : Etudiant /* e1 et e2 deux variables de type **Etudiant**, elles ont chacune 4 champs : Nom, Prénom, … */

I.2- Les enregistrements en langage C

En langage C, le type enregistrement est déclaré avec le mot clé ***struct***.

Pour déclarer le type Etudiant en langage C, on écrit :

```
struct Etudiant {
      char Nom[40] ; /* une chaine de 39 caractères maximum */
      char Prenom[40] ; /* une chaine de 39 caractères maximum */
      int Age ;
      float Moyenne ;
} ;
struct Etudiant e1, e2 ; /* déclaration de deux variables de type struct Etudiant */
```

Remarques : ① En général, on utilise le mot clé ***typedef*** pour renommer le type **struct** Etudiant en Etudiant comme suit : `typdef struct Etudiant  Etudiant ;`

La déclaration des deux variables e1 et e2 sera simplifiée comme suit :

```
typdef struct Etudiant Etudiant ;
Etudiant e1, e2 ; /* Deux variables de type Etudiant */
```

② L'instruction ***typedef*** peut être placée avant ou après la déclaration du type `struct Etudiant { … } ;`

I.3- Accès aux champs d'un enregistrement

Pour accéder à un champ, il faut utiliser l'opérateur point (**.**).

Exemples : - Pour affecter la valeur 20 au champ Age de l'étudiant e1, on écrit : `e1.Age = 20 ;`

- Pour donner la valeur 11.5 au champ Moyenne de l'étudiant e1, on écrit : `e1.Moyenne = 11.5 ;`

II- Les pointeurs en langage C

Rappel : Un pointeur est une variable qui contient l'adresse mémoire d'une autre variable stockée en mémoire.

II.1- Déclaration : Pour déclarer une variable de type pointeur sur un type donné (simple ou structuré), il suffit de précéder sa déclaration par un astérisque (*).

Exemple1 : Pour déclarer un pointeur (appelé ptr) sur un entier (***int)***, on peut écrire : `int *ptr ;`

Exemple2 : Pour déclarer un pointeur (appelé p) sur un enregistrement de type Etudiant, on peut écrire :

```
Etudiant *p ;
```

II.2- Accès à la donnée : Pour accéder à la variable pointée, on écrit par exemple : `*ptr = 20 ;` ou `printf("%d", *ptr) ;`

Remarque : Les pointeurs sont utilisés pour le passage de paramètres par adresse (par variable) dans les procédures (et fonctions), comme déjà vu au **TP N°5 – PROCEDURES & FONCTIONS**.

II.3- Accès aux champs d'un enregistrement pointé

Pour accéder à un champ (le champ Age, par exemple) d'un enregistrement pointé (par la variable p), la notation `(*p).Age` reste valable. Cependant, le langage C, propose une notation plus simple selon la syntaxe suivante : `p->Age`. Pour affecter à l'âge la valeur 20, on écrit alors `p->Age = 20` ; au lieu de `(*p).Age = 20 ;`

II.4- Initialisation : La constante **NULL** (en majuscule) est utilisée pour indiquer que le pointeur ne pointe vers aucune donnée valide. On écrit alors : `ptr = NULL ;`

II.5- Allocation mémoire : Pour allouer un espace mémoire à la variable pointée, on utilise l'instruction `malloc` de la bibliothèque `stdlib.h`.

Exemple1 : pour réserver l'espace à un entier on écrit : `ptr = malloc( sizeof( int ) ) ;`

Exemple2 : pour réserver l'espace à un enregistrement de type Etudiant pointée par le pointeur `p`, on écrit :

```
p = malloc( sizeof( Etudiant ) ) ;
```

II.6- Libération de l'espace mémoire : Pour libérer l'espace mémoire (précédemment alloué par `malloc`) et pointé par le pointeur ***ptr***, on utilise l'instruction `free` de la bibliothèque `stdlib.h`.

On écrit alors : `free( ptr ) ;`

III- Les listes en langage C

III.1- Rappel : Une Liste Linéaire Chaînée (LLC) est un ensemble d'éléments (Cellule, Nœud, Maillon) alloués dynamiquement chaînés (reliés) entre eux. Chaque élément est un enregistrement qui contient au moins deux champs : ①- Un champ qui contient l'information (appelé Valeur, **Val**, Info ou Data).

②- Un champ qui est un pointeur sur l'élément suivant (appelé **Suiv** ou Suivant).

III.2- Déclaration : Dans un programme C, le type Liste (d'entiers) peut être déclaré comme suit :

```
typedef struct Element Element;
typedef Element* Liste;
struct Element {
            int val; /* val Contient l'information utile */
            Liste suiv; /* suiv un pointeur vers l'élément suivant */
            };
Liste L, P, Q, Ptr ; /* L, P, Q, Ptr : des variables de type Liste ou pointeur sur Element */
```

III.3- Accès aux champs

✓ On accède au champ **val** par **L->val**. Exemples : `L->val = 25 ;` ou `printf("%d\n", L->val) ;`

✓ On accède au champ **suiv** par **L->suiv**. Exemples : `L->suiv = NULL ;` ou `if(L->suiv != NULL)`…

III.4- Allocation (création)

```
L = malloc( sizeof( Element) ) ;
```

✓ Si l'opération d'allocation s'est bien déroulée, l'instruction ***malloc*** retourne une adresse mémoire valide (valeur non nulle). Sinon, elle retourne NULL.

III.5- Libération (destruction)

✓ L'instruction `free(L) ;` permet de détruire (supprimer) un élément créé avec l'instruction ***malloc*** et pointé par le pointeur ***L***.

IV- Exercice

Objectif : Maîtriser les listes à travers la recherche des nombres premiers.
Dans ce TP, on s'intéresse à une liste d'entiers, mais les principes abordés restent valables pour des listes de n'importe quel type de données.

Le but de cet exercice est de trouver tous les nombres premiers inférieurs à un entier **Max** (par exemple $\mathbf{10^6}$).
Une méthode rapide pour vérifier si un nombre entier **N** est premier est de tester s'il accepte un diviseur parmi les nombres premiers (déjà connus) qui sont inférieurs à sa racine carrée.
Nous utilisons une liste linéaire chaînée pour stocker les nombres premiers. A chaque fois qu'un nombre premier est découvert (trouvé), il est inséré à la fin de cette liste.

Q1) Écrire une procédure ***InsererTete(L, N)*** qui permet d'insérer un entier ***N*** au début de la liste ***L*** (La liste ***L*** peut être vide).

Q2) Écrire une procédure ***AfficherListe(L)*** qui permet d'afficher les éléments d'une liste ***L***. Si ***L*** est vide, la procédure doit afficher le message '*** Liste vide ***'.

Q3) Tester les deux procédures écrites en **Q1** et **Q2** : Dans le programme principal (fonction ***int main()***), créer une liste vide (***L***) ; Insérer dans ***L***, les nombres suivants : 2, 3, 5, 7 puis afficher le contenu de cette liste. Dans quel ordre les nombres 2, 3, 5 et 7 sont affichés ?

Q4) Écrire une procédure ***InsererQueue(L, N)*** qui permet d'insérer un entier ***N*** à la fin de la liste ***L*** (***L*** peut être vide). **Remarque** : Il est possible d'utiliser la procédure ***InsererTete***, si nécessaire.

Q5) Dans le programme principal, remplacer les insertions en tête de la liste par des insertions à la fin. Cette fois-ci, dans quel ordre les nombres 2, 3, 5 et 7 sont affichés ?

Q6) Écrire une fonction ***Premier(L, N)*** où ***L*** est une liste qui contient les nombres premiers déjà découverts et ***N*** un entier. Cette fonction teste si ***N*** est un nombre premier ou non en vérifiant si ***N*** accepte un diviseur parmi les nombres premiers qui sont déjà dans la liste ***L*** et sont inférieurs à sa racine carrée.

Q7) Écrire une procédure ***Lister_Nombres_Premiers(L, Max)*** qui permet la construction de la liste ***L*** en cherchant tous les nombres premiers compris entre ***2*** et ***Max***.

Le principe de cette procédure est le suivant :

- On ne teste que les nombres impairs, car tous les nombres pairs sont divisibles par **2**.
- Au début la liste ***L*** est vide ***(L = { })***. - On rajoute 3 à la liste ***(L = { 3 })***.
- On commence la recherche par le nombre 5, le premier nombre dans la liste est 3, mais 3 est supérieur à la racine carrée de 5 alors 5 est premier et est ajouté à la liste ***(L = { 3, 5 })***.
- On passe à 7, le premier nombre dans la liste est 3, mais 3 est supérieur à la racine carrée de 7 alors 7 est premier et est ajouté à la liste ***(L = { 3, 5, 7 })***.
- On passe à 9, on le divise par 3, 3 est un diviseur de 9, alors 9 n'est pas premier.
- On passe à 11, on le divise par 3, il n'est pas divisible par 3. Le nombre premier suivant dans la liste est 5, mais 5 est supérieur à la racine carrée de 11 alors 11 est premier et est ajouté à la liste ***(L = { 3, 5, 7, 11 })***.
- On passe à 13, 15, 17, 19, … et on continue, avec le même principe, jusqu'à trouver **tous les nombres premiers impairs** inférieurs ou égaux à ***Max***.
- A la fin, on rajoute au début de la liste, le nombre **2** qui est le **seul nombre premier pair**.

Q8) Écrire une fonction ***Longueur(L)*** qui donne le nombre d'éléments de la liste ***L***.

Combien de nombre premier inférieur à 10^6 ?

Q9) Écrire une fonction ***Somme(L)*** qui donne la somme des éléments de la liste ***L*** (La somme de tous les nombres premiers inférieurs ou égaux à ***Max***).

Q10) Écrire une procédure ***SupprimerValeur(L, N)*** qui permet de supprimer la valeur ***N*** *de la liste* ***L***.

Q11) Un programme bien conçu, doit impérativement supprimer tous les éléments créés dynamiquement (avec l'instruction **malloc**) en utilisant l'instruction **free**. Écrire une procédure ***SupprimerListe(L)*** qui permet de supprimer tous les éléments de la liste ***L***. Appeler cette procédure à la fin du programme principal.

Q12) Donner une version récursive des fonctions/procédures suivantes : ***AfficherListe***, ***InsererQueue***, ***Longueur, Somme, SupprimerValeur, SupprimerListe.***