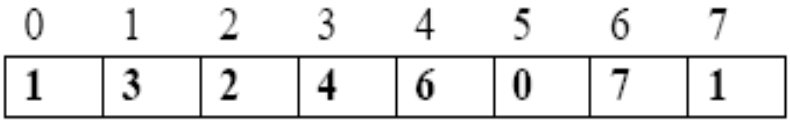
8- Les tableaux et les pointeurs

8.1- Les tableaux

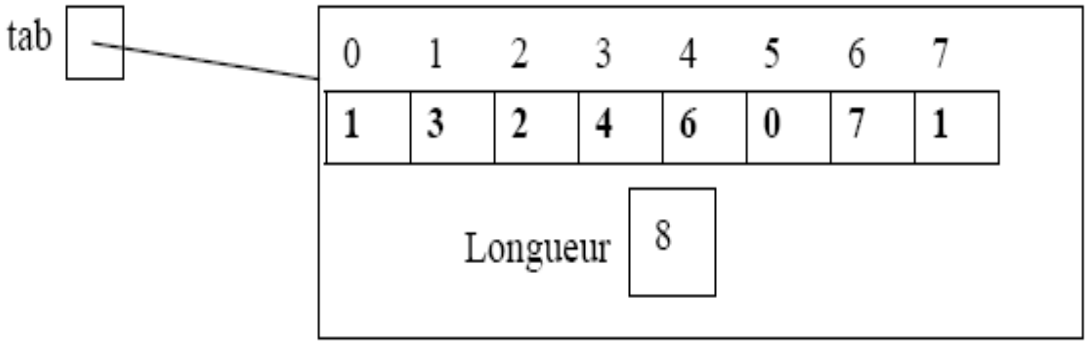
8.1.1- Les tableaux à un indice

- Tableau : Collection de variables (toutes de même type), appelées valeurs ou cases, accessibles directement à partir de leur position (indice).
- Longueur d'un tableau : nombre de cases défini lors de la définition du tableau et non modifiable après.
- Indices valides = entier de 0 jusqu'à "longueur du tableau -1"(dans d'autres compilateurs, l'ensemble des indices valides appartient à l'intervalle [1, longueur du tableau].

Représentation simplifiée et usuelle :



En mémoire :



Un tableau est placé à une adresse dans la mémoire de l'ordinateur. C'est cette adresse qui est stockée dans une variable de type tableau (i.e. tab).

Syntaxe :

type identificateur [Taille] ;

C'est un ensemble d'éléments de même type désignés par un identificateur unique. Chaque élément est repéré par un « indice » qui précise sa position dans l'ensemble.

Exemple :

```
int tab [ 10 ] ;
```

Cette déclaration réserve un emplacement de 10 éléments de type entier. En langage C, la numérotation des indices commence par 0. Le premier élément du tableau sera désigné par tab[0].

- **Règles générales des tableaux :**

  1- Un élément d'un tableau est une variable.

Exemple :

```
Tab[5] = 10 ;
Tab[5]++ ;
```

2- Un indice peut prendre la forme d'une expression arithmétique de type entier.

Exemple :

```
Tab[2*3]=6 ;
```

3- La dimension d'un tableau (nombre d'éléments) est une constante ou une expression constante.

Exemple :

```
#define N 20

int Tab_1[N] ;
float Tab_2[3*N-2] ;
```

- **Les opérations sur les tableaux**

L'accès aux éléments d'un tableau se fait en donnant le nom de la variable tableau suivi entre deux crochets ([]) de l'indice de l'élément. Le fait que les éléments constituant un tableau soient indicés permet de parcourir tous les éléments avec une boucle. Le **pour est généralement la boucle la plus adaptée,** puisque l'on connaît le nombre de fois qu'on doit effectuer le traitement (une fois par élément).

- Chaque élément du tableau est considéré comme une variable simple de type de base des éléments du tableau, on peut donc avoir les opérations suivantes :

  1. Accès en lecture (afficher):
  printf(" %d ", tab[i]) : le contenu du tableau à l'indice i est affiché à l'écran.

  Exemple : pour afficher le contenu du tableau en entier :

```
for(i = 0; i < 10; i++)
printf(" %d ", tab[i]);
```

  2. Accès en écriture (initialiser, modifier) :

  a. tab[i] = valeur : la valeur est placée dans le tableau à l'indice i.

  b. scanf("%d", &tab[i]) : la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice i.

  Exemple : pour remplir le tableau en entier :

```
for(i = 0; i < 10; i++)
scanf("%d", &tab[i]);
```

- **Exercice :**

Ecrire un programme qui permet de lire un tableau de 10 éléments entiers, puis de calculer et d'afficher la moyenne de ces éléments.

- **Exemples d'initialisation de tableaux**

```
int  tab [5] = {1,2,3,4,5} ;

int  tab [5] = {1,2} ; // tab[0]=1, tab[1]=2, tab[2] jusqu'à tab[4] contient la valeur 0

char  tab_ch [4] = {'a','b','c','d'} ;

char  tab_ch [8] = "exemple" ; // tab[7] contient le caractère null

 char  tab_ch [ ] = "tab aura 23 caractères" ;
```

- **Quelques algorithmes de base sur les tableaux**

1. Un algorithme de recherche

Ecrire un programme qui permet de lire un tableau de taille n, donnée par l'utilisateur, puis de chercher une valeur, donnée par l'utilisateur aussi, dans ce tableau ainsi que son indice.

- **Paramètres d'entrée : un tableau de n entiers et une valeur val.**
- **Paramètres de sortie : le booléen trouve et l'entier indice.**
- **Spécification : le booléen trouve doit être à 1 si val se trouve** dans le tableau. La valeur d'indice vaut alors le plus petit indice de la case contenant la valeur **val dans le tableau.**

Solution: faite au cours**.**

2. Un algorithme de tri

Trier un tableau c'est réorganiser ses éléments suivant un ordre déterminé croissant ou décroissant pour les nombres ou les caractères.

Le programme de tri consiste à trouver le plus petit élément et le mettre dans la première case et ainsi de suite pour les cases 2,3,…

Solution: faite au cours**.**

8.1.2- Les tableaux à deux indices (matrices)

Comme tout langage, le langage C autorise la déclaration des tableaux à deux indices.

Syntaxe :

type identificateur [nombre de ligne] [nombre de colonnes] ;

Exemple :

int matrice [2] [3] ; // déclare un tableau de (2 x 3) éléments.

8.2- Les pointeurs

Une valeur de type pointeur repère une variable, ce qui signifie qu'une valeur de type pointeur est l'adresse d'une variable.

Syntaxe :

type *nom_du_pointeur ;

Exemple :

int *p_entier ; // p_entier est un pointeur vers un int

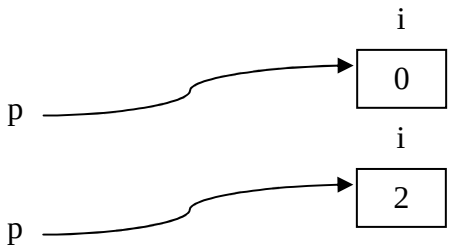
char *p_carac ; // p_carac est un pointeur vers un char

- Pour assigner une valeur à une variable de type de pointeur, deux démarches sont possibles :

1- Par l'utilisation de l'opérateur & :
L'opérateur & appliqué à une variable délivre son adresse. Donc, cette adresse peut être affectée à la variable de type pointeur.

Exemple :

```
int i ;
int *p ;
……
p = &i ;
……..
……..
*p = 2;
```

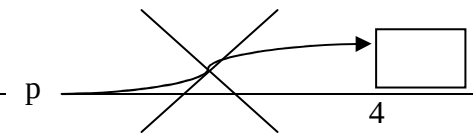


2- Par l'utilisation de la fonction **malloc** :
La fonction **malloc** permet de créer un emplacement pour une variable d'un type donné.

Exemple :

```
int *p ;
….
p = malloc(sizeof(int)) ;
```

- Pour libérer cet emplacement, nous utilisons la fonction **free**



```
free(p) ;
```

9- Les fonctions

Syntaxe : type identificateur (liste_arguments)

```
{
    liste_instructions ;
}
```

- **type :** est le type de la valeur rendue par la fonction, exemple : int, double, float, char,…
- **identificateur :** est le nom de la fonction.
- **liste_arguments :** est la liste des paramètres de la fonction (séparés par des virgules).
- **liste_instructions :** est l'ensemble des instructions qui seront exécutées lors de l'appel de la fonction à partir du programme principal. Parmi ces instructions, il doit y avoir au moins une instruction de type :

  return expression ;

  où « expression » est la valeur de retour de la fonction.

Exemple :

```
int max( int a, int b )
{
    if( a > b)
    return a ;

    else
    return b ;
}
```

L'appel d'une fonction se fait par spécification du nom de la fonction et passage des valeurs à ses arguments.

Exemple :

```
int maximum ;
……
void main(void)
{
    maximum = max(5, 2) ;
}
```

A la différence des autres langages (comme Pascal), le langage C ne comporte pas le concept procédure. Cependant, on peut réaliser une procédure à l'aide d'une fonction qui ne retournera aucune valeur par l'utilisation du mot-clé : **void**. Ainsi, elle ne comporte pas l'instruction **return**.

Exemple :

```
void max( int a, int b )
{
    if( a > b)
    printf("Le max c'est %d", a);
```

```
        else
        printf(″Le max c'est %d″, b); ;
}
```

- L'appel dans le programme principal se fait comme suit :

```
……
void main(void)
{
        max(5, 2) ;
}
```

10- Les structures de données

10.1- Préliminaires

10.1.1- Définition de types

Il existe en C un moyen de donner un nom à un type. Il consiste à utiliser le mot clé : **typedef.** Il suffit de faire suivre ce mot clé par une déclaration d'une variable. Dans ce cas, l'identificateur (qui est le nom de la variable) devient le nom de type.
Exemple :

1)
```
typedef int tab[10] ;
tab t1, t2; // t1 et t2 deux tableaux de 10 entiers
```

2)
```
#define   VRAI   1
#define   FAUX   0
typedef int BOOLEAN ; //définition d'un type
BOOLEAN B ; // définition d'une variable de type BOOLEAN
…..
B = VRAI ;
```

10.1.2- Déclaration de structures

En C, la déclaration d'une structure se fait par l'utilisation du mot clé : **struct**.

Syntaxe :

```
struct nom_structure
{
//déclarer les champs de la structure
} ;
```

Exemple :

```
struct personne
{
        char nom[20] ;
        char prénom[20] ;
        int age;
} ;
```

- Pour déclarer des variables de ce type de structure :
  1- **struct personne** pers1, pers2 ;
  2-
```
struct {
        char nom[20] ;
        char prénom[20] ;
        int age;
```

```
} pers1, pers2 ;
```

Inconvénient : impossible de déclarer une autre variable du même type.

3-
```
struct personne
{
    char nom[20] ;
    char prénom[20] ;
    int age;
} pers1, pers2;
```

Initialisation d'une structure :

Une structure peut être initialisée à la manière d'initialisation d'un tableau.

Exemple :

```
struct personne pers1 = {"Ali","Kamel",25} ;
```

Accès aux champs d'une structure :

Pour désigner un champ d'une structure, la syntaxe est :

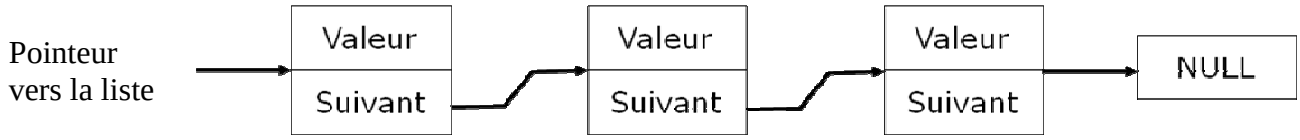
```
Variable_de_type_structure.nom_du_champ
```

Exemple :

```
pers1.nom[0]  c'est la lettre 'A' de la chaîne "Ali"
```

10.2- Les listes chaînées

Les liste chaînées est un système informatique qui permet la sauvegarde dynamique des données. Les éléments sont chaînés les uns avec les autres et l'accès se fait par un pointeur.



La déclaration d'une liste chaînée en C est comme suit :

```
typedef struct liste
{
    int valeur ;
    struct liste * suivant ;
} Maliste;
```

La déclaration d'une variable de type Maliste :

```
Maliste * liste_test = NULL ;
```

10.2.2- Insertion d'un élément dans la liste

L'insertion d'un élément dans la liste se fait généralement en deux étapes :

1- création et initialisation de l'élément : cette étape consiste à créer et initialiser les champs de l'élément à insérer.

Exemple :

// Création d'un élément qui va être insérer dans la liste liste_test :

```
Maliste * element ;
…
element = malloc(sizeof(Maliste)) ;
```

// initialisation des champs de l'élément :

```
element -> valeur = 10 ; // element -> valeur est équivalente à : (*element).valeur
```

2- l'ajout de l'élément à la liste : cette étape consiste à rajouter l'élément à la liste (l'ajout se fait par défaut à l'en-tête de la liste).

Exemple :

// l'ajout de « element » à la liste liste_test

```
element -> suivant = liste_test ;
liste_test = element ;
```

10.2.3- Suppression d'un élément de la liste

Pour supprimer un élément de la liste, il faut créer un élément qui permet de pointer sur le reste de la liste afin de garder le chemin des éléments restants de la liste.

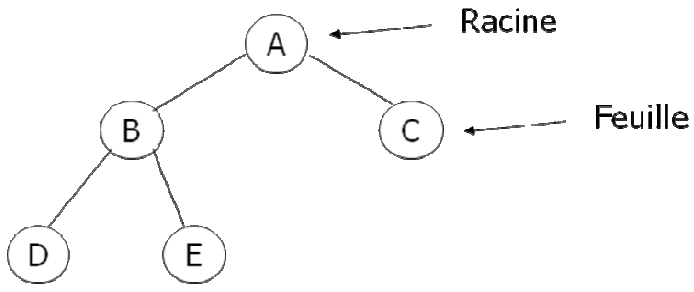
Exemple :

// suppression du premier élément de la liste

```
Maliste * element_supprime ;
….
element_supprime = liste_test -> suivant ; // pointer sur le reste de la liste
free (liste_test) ; // libérer l'espace occupé par le 1er élément de la liste
liste_test = element_supprime ; /* liste_test pointe maintenant sur le reste des éléments de la liste */
```

10.3- Les arbres

Un arbre est une structure composée de nœuds et de feuilles reliés par des arrêtes. On les représente généralement en mettant la racine en haut et les feuilles en bas (contrairement à un arbre réel).



10.3.1- Les arbres binaires

Un arbre binaire est un arbre où les nœuds ont au plus deux fils (gauche et droit).

- ***Déclaration d'un arbre binaire***

```
typedef struct Arbre
{
int Valeur;
struct Arbre * Nœud_Gauche;
struct Arbre * Nœud_Droit;
} Arbre_Binaire;
```

- ***Création d'un élément***

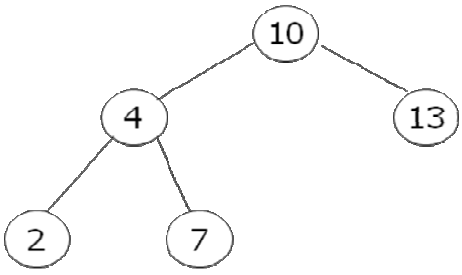
```
Arbre_Binaire *Racine;
Racine = malloc(sizeof(Arbre_Binaire));
Racine->Valeur = 5;
Racine->Nœud_Gauche=NULL;
Racine->Nœud_Droit=NULL;
```

- ***Ajouter un nœud à l'arbre***

Pour faciliter cette opération on utilise les arbres binaires de recherche. Un arbre binaire de recherche est un arbre où :

- Tous les nœuds du sous-arbre de gauche d'un nœud de l'arbre ont une valeur inférieure ou égale à la sienne ;
- Tous les nœuds du sous-arbre de droite d'un nœud de l'arbre ont une valeur supérieure ou égale à la sienne ;

Exemple :



Dans ce cas, l'insertion, de chaque élément, se fait comme suit :

```
Arbre_Binaire *p ; // pointeur qui va parcourir l'arbre
Arbre_Binaire *element ; // l'élément qui va être rajouté à l'arbre
p = Racine ;
…..
while ( p != NULL )
{
        if( val > p -> Valeur )
        p = p -> Nœud_Droit ; // aller à droite
        else
        p = p -> Nœud_Gauche ;  // aller à gauche
}

element = malloc(sizeof(Arbre_Binaire)) ;
element -> Valeur = val ;
```

```
element -> Nœud_Droit = NULL ;
element -> Nœud_Gauche = NULL ;

p = element ;
```