Pointeur : Notions de Base

Déclaration : type * nom_pointeur | int *p ;
Initialisation : nom_pointeur = & nom_variable-pointe
Int n ; int *p =NULL ; p=&n;
Affichage : printf("\n%p",p);//contenu pointeur (adresse)
printf("\n%d",*p); valeur de variable pointé

Pointeur et Fonction

```
void echange ( int *a , int *b){
        int aux ;
        aux=*a ;
        *a=*b;
        *b=aux;}
```

Pointeur et Tableaux

- Le nom du tableau est un pointeur qui contient l'adresse de son premier élément .
- **P++** pointe sur l'élément suivant du tableau .
- **Tab** : l'adresse de Tab[0]
- **Tab+i** : l'adresse de T [i]
- ***(Tab+i)** : le contenu de Tab [i]<=> T [i]
- **Sizeof()**: fonction la taille (en octets) du type ou variable

Addition Pointeur :
P=T ; P=P+3 <=> &T[0+3] <=> P=P+3*sizeof(int)
Soustraction Pointeur :
Q =&T[3] ; Q-2<=> &T[3-2]<=>Q-2*sizeof()
Remplir un Tableau :

```
void remplir (int *T , int n){
        int *p;
        p=T;
        for(p=T;p<T+n;p++) {
                printf("T[%d] = ",p-T);
                scanf("%d",p);
        }
}
```

Afficher un Tableau :

```
void afficher(int*T , int n){
        int *p;
        p=T;
        for(p=T;p<T+n;p++) {
                printf("%d|",*p);
        }
}
```

La somme élément Tableau :

```
s=0 ;
For (P=T;P<T+6 ;P++){
            s=s + *P;
}
```

Minimum d'un Tableau :

```
min *T ;
For (P=T;P<T+6 ;P++){
          If(min >*P){min =*P;}
}
```

Pointeur et Structure

```
struct complexe {
        float reel;
        float imaginaire;
};
void main(){
   struct complexe comp ;
  // déclaration pointeur de type structure nombre complexe
  struct complexe *pcomp;
  //initialisation pointeur vers le variable complexe comp
  pcomp=&comp;
  pcomp -> reel =1; //<=>(*pcomp).reel
}
```

Allocation Dynamique

<stdlib.h>
malloc():
Void * malloc (taille_octet) //retourner NULL ou pointeur.
P=malloc (sizeof (int)); //allocation dynamique entier.
Int * p =malloc(4*sizeof(int))//tableau entier 4 cases.
*(p+i) // accéder a l'élément i
free():
Void free (pointeur) ; free (T);//libérer allocation
Calloc():
Void * calloc (nbr_element , taille_element)
Int *p =malloc (4*sizeof(int));
<=> int * p =calloc(4,sizeof(int));
//L'espace alloué est initialisé a zéro pas vide
Realloc():
Void *realloc (pointeur,Taille_octet);
//Redimensionner un espace déjà alloué
Int *p =malloc(3*sizeof(int));
p= realloc(p,5*sizeof(int));

Liste Chainé

```
 struct element{
        int information;
        struct element *suivant;

};
typedef struct element element;
```

```
// ajout élément par début
struct element * AjoutD(struct element *tete, int x){
        element *p;
        p=malloc(sizeof(element));
        p->information=x;
        p->suivant=tete;
        return p;
}
```

```
//afficher la liste chaine
void Afficher(struct element *tete){
        element * index;
        index =tete;
        while(index !=NULL){
                printf("%d |",index->information);
                index =index->suivant;
        }
}
```

```
struct element *Dernier( struct element *tete){
        //retourner le dernier élément de la chaine
        struct element *index ;
        struct element *dernier;
        index=tete ;
        dernier=tete;
        while(index !=NULL){
                dernier=index;
                index=index->suivant;
        };
        return dernier;
}
struct element * AjoutF(struct element *tete, int x){
        //ajouter element a la fin de liste
        element *p;
        element * dernier ;
        p=malloc(sizeof(element));

        if(tete==NULL){
                p->information=x;
                p->suivant=NULL;
                tete=p;
        }else{
                dernier=Dernier(tete);
                p->information=x;
                p->suivant=NULL;
                dernier->suivant=p;
        }
        return tete;
}
```