Les structures dynamiques

1. Les Pointeurs

> Définition : un pointeur est une variable qui contient l'addresse mémoire d'une autre variable.

Le Stype du pointeur est le même que la variable pointé.

Déclaration :

En aglo :

```
  nomPointeur : pointeur de type
```

En C :

```
type * nomPointeur;
```

Deux opérateurs sont utilisés pour manipuler les addresses mémoire où les contenus de ces addresses :

- & : récupérer l'addresse mémoire d'une variable
- * : récupérer la valeur de l'espace mémoire pointé par un pointeur

Exemple :

```
int main() {
  int x;
  x = 10;
  int * ptr;

  ptr = & x; // reçoit l'addresse mémoire de x

  printf("La valeur de x est de %d :", *ptr);
  printf('x habite à l'addresse %u), ptr); // unisgned

  return 0;
}
```

À retenir :

- addresse & x ----------> ptr
- contenu x ----------> *ptr

2. Allocation et libération de mémoire

On peut utiliser les pointeurs sans passer par une variable déjà existante en faisant une allocation dynamique de la mémoire centrale. Puis après utilisation, on libère l'espace alloué.

Syntaxe des fonctions :

Opération | Algo | C
Allocation | allouer | malloc
Libération | liberer | free

Exemples

```
int main() {
  int * ptr;

  // allocation dynamique de la mémoire centrale
  ptr = (int *) mailoc (sizeof(int));
  // affectation d'une valeur à l'espace mémoire pointé par ptr
```

```
  *ptr = 10;
  // multiplier le contenu de l'espace pointé par ptr = 2
  *ptr *= 2; // Ne pas confondre le * de multiplication et le * d'allocation de valeur
  printf('La valeur de l'espace mémoire pointé : %d', *ptr);
  // Libération de l'espace mémoire
  free(ptr);

  return 0
}
```

Les Applications des pointeurs

Les tableaux dynamiques

En C, tout tableau est un pointeur qui garde l'addresse mémoire de la première case(élément) qu'il soit statique ou dynamique.

Tableau statique :

La taille est fixée dès la déclaration

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

tab garde l'addresse du premier élément : `&tab[10]`

tableaux dynamiques :

la taille est fixée pendant l'execution du programme

```
int main() {
  int *tab;
  int nb;

  printf("Donner le nombre d'éléments : ");
  scanf("%d", &nb);
  // allocation dynqmique de la MC pour un tableau
  tab = (int*) malloc (sizeof(int) * nb);

  // parcours par indices
  int i;
  for (i=0; i<nb; i++) {
    printf("Donner un element : ");
    scanf("%d", &tab[i]);
  }

  // la nouveauté : le parcour par pointeurs
  int *ptr;
  for(ptr = tab; ptr < tab + nb; ptr ++) {
    scanf("%d", ptr)
  }

  // affichage de éléments par indices :
  for(i=0; i<nb; i++) {
    printf("Element %d : %d", i, tab[i]);
  }

  // la nouveauté : le parcour par pointeur
  for (ptr = tab; ptr < tab + nb; ptr ++) {
    printf(" element %d : %d", ptr-tab, *ptr );
  }

  return 0;
}
```

Les chaines de caractères

les chaines de caractères en langage C sont des tableaux de caractères et donc des pointeurs :

```
int main() {
```

```
  char ville[] = "paris";
  char ptr = ville;

  printf("%s", ville); // ---> paris
  printf("%s", ptr); // -----> paris
  printf("%c", *ptr); // ----> p
  printf("%c", *(ptr+2)); // > r
  printf("%c", *(++ptr)); // > a
  printf("%c", *(ptr++)); // > a
  printf("%s", ptr); // -----> ris
  printf("%c", ville[ptr-ville+1]); // --> i

  return 0;
}
```

Passage de parametres par addresse

Dans les procédures en C, le passage de paramètres se fait par copie ou par addresse.

- Passage par Copie

```
void ajouter(int x) {
  x = x + 5;
}
int main() {
  int a = 10;
  ajouter(a);
  printf("la valeur de x est de %d" : a);

  return 0;
}
```

Donc avec ce passage, a vaut 10 car le modifications de la procédure ajouter se font sur la copie de paramètre a. Elles ne sont pas effectives sur a.

- passage par addresse :
  - On envoie l'addresse de la variable a au pointeur x

```
void adjouter(int *x) {  //x pointe sur a
  *x = *x + 5; // manipulation du contenu de a
}
int main() {
  int a = 10;
  ajouter(&a); //envoie de l'addresse de a
  printf("la valeur de a est de %d", a);

  return 0;
}
```

Les modifications sont faites directment sur la valeur a en manipulant son addresse.