Langage de programmation C :

Les pointeurs

LES POINTEURS

- Réflexion;

```
int i=5;
printf ("i=%d son adresse mémoire:%d", i,&i);
```

Va donner comme résultat aprés execution

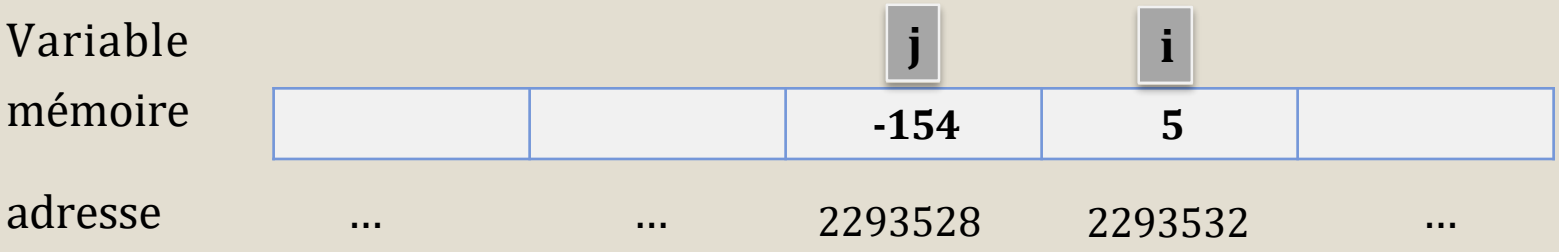
i=5 son adresse mémoire:2293532

```
int i=5,j=-154;
printf ("j=%d son adresse mémoire:%d\n", j,&j);
printf ("i=%d son adresse mémoire:%d", i,&i);
```

Va donner comme résultat aprés execution

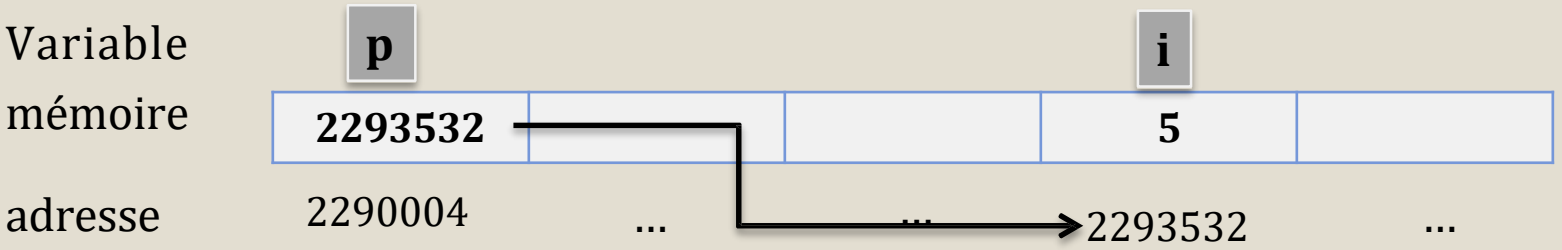
j=-154 son adresse mémoire:2293528

i=5 son adresse mémoire:2293532



LES POINTEURS

- Définition
  - Variable: sert à stocker les données manipulées par un programme. Lorsque l'on déclare une variable, un espace mémoire lui sera réservé pour y stocker sa valeur. L'emplacement de cet espace dans la mémoire est nommé adresse.
  - Pointeur: est une variable qui désigne un emplacement mémoire (une adresse) occupée par une donnée



On dit que p pointe sur l'adresse mémoire occupée par la variable i

LES POINTEURS

- Déclaration:

  un pointeur est déclaré au moyen de l'opération d'indirection «*».

  - Syntaxe :

  Type * <Nom_du_Pointeur>;

  →Nom_du_Pointeur ne peut (doit) recevoir que des adresses de variables du type Type

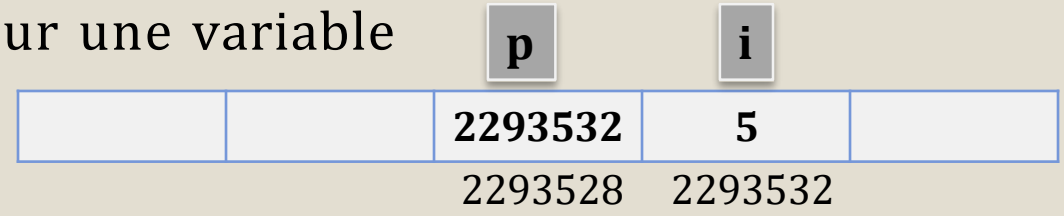
- Exemple:

  - int i, j; //déclaration de deux variables de type entier
  - int *p; //déclaration d'un pointeur sur une variable entière
  - float *ptr; //déclaration d'un pointeur sur une variable float
  - char nom, *ptr;

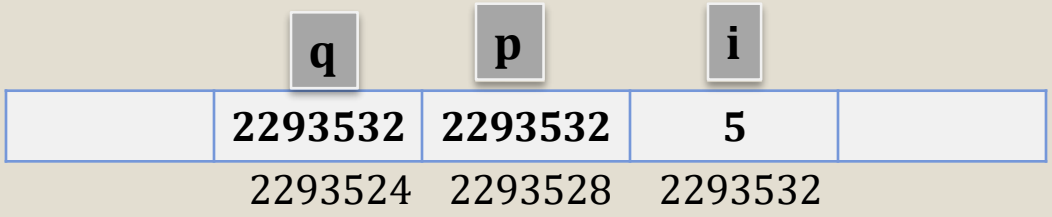
  //déclaration d'une variable de type char et un pointeur sur char.

Les pointeurs

- Initialisation:
  - Lorsqu'on déclare un pointeur sans l'initialiser, on ne sait pas sur quoi il pointe.
  - Pointeur qui ne pointe sur rien
    - int *ptr = NULL;
  - Pointeur qui pointe sur une variable
    - int i=5;
    - int *p=&i;



  - Pointeurs qui pointent sur la même adresse
    - int i=5;
    - int *p=&i;
    - int *q=p;



LES POINTEURS

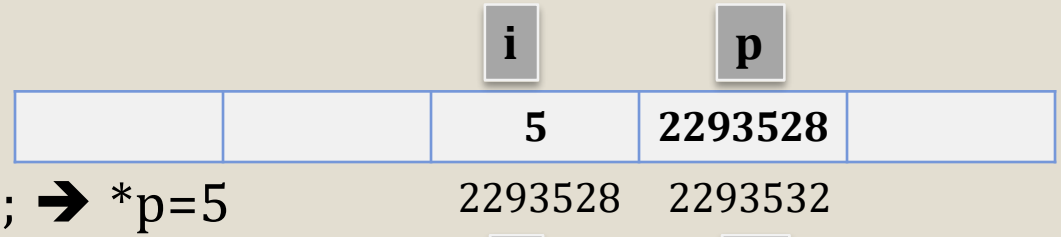
- Manipulation:
  - Exemple:
    int i=5;
    int *p=&i;
    - &i: l'adresse mémoire de la variable i
      - printf("%d\n",&i); ➔ 2293532
    - p: l'adresse mémoire sur laquelle pointe p
      printf("%d\n",p);➔ 2293532
    - *p: représente le contenu de la case mémoire sur laquelle p pointe:
      printf("%d\n",*p); ➔ 5
    - &p: l'adresse mémoire de la variable p:
      printf("%d\n",&p); ➔ 2293528

Les pointeurs

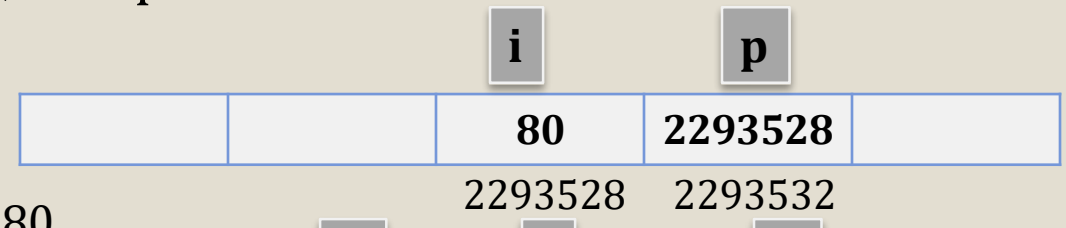
- Manipulation (suite):
  - Pointeur qui ne pointe sur rien

```
int *p = NULL;
```

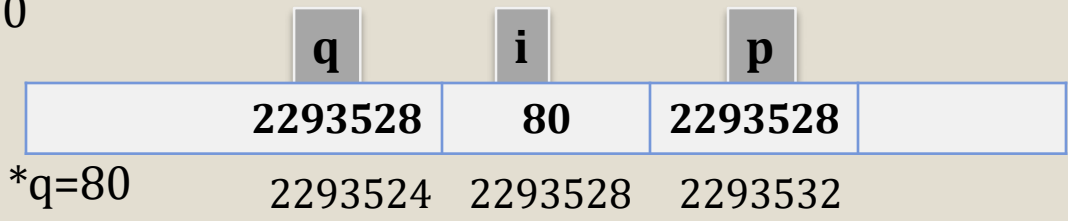

```
int i=5;
p=&i;
printf("*p=%d\n",*p); ➔ *p=5
```




```
*p=80;
printf("i=%d\n",i); ➔ i=80
```




```
int *q=p;
printf("*q=%d\n",*p); ➔ *q=80
(*p)++; //eq. à i++;
```



Les pointeurs

- Manipulation (suite):
  - Exercice: que produit le code suivant

```
int a=51;
int b=120;
int * ptr;
ptr = (a>b) ? &a : &b;
(*ptr)++;
printf("a=%d, b=%d, *pointeur=%d \n", a,b,*ptr);
```

Les pointeurs

- Manipulation (suite):
  - Exercice: que produit le code suivant

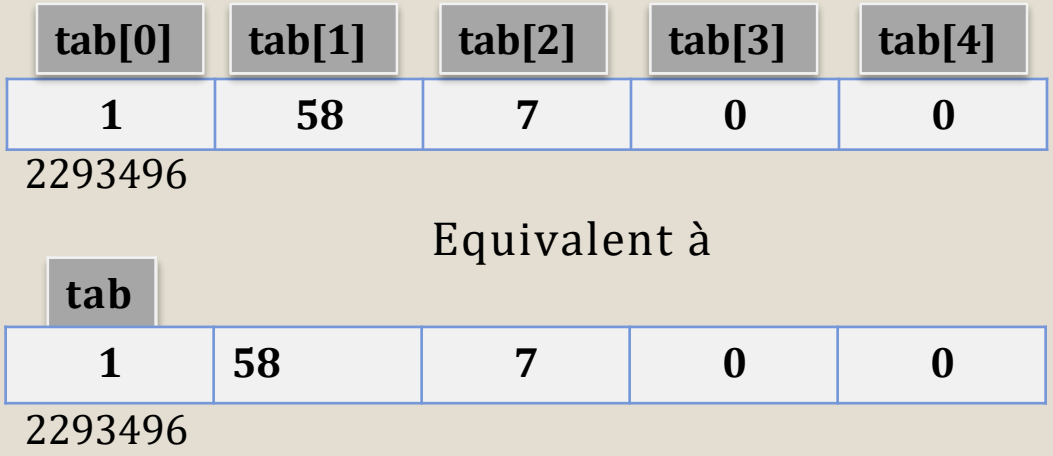
```
int a=51;
int b=120;
int * ptr;
ptr = (a>b) ? &a : &b; //ptr pointe sur la case b
(*ptr)++; //incrément ele contenu de la case pointée par ptr
printf("a=%d, b=%d, *pointeur=%d \n", a,b,*ptr);
```

  - Le résultat:

    a=51, b=121, *pointeur=121

Les pointeurs et tableaux

- Réflexion
  - int tab[5]={1,58, 7};



➔ Le nom d'un tableau représente l'adresse de son premier élément

➔ Équivalent à dire que le nom d'un tableau est un pointeur constant sur le premier élément du tableau

Les pointeurs et tableaux

- Pointeurs et tableaux sont en étroite relation:
  - Exemple 1:

```
int tab[10]={1,58, 7}, i;
printf ("%d\n",*(tab)); //1: tab pointe sur le 1er élément
printf ("%d\n",*(tab+1)); //58: tab+1 pointe sur le 2éme élément
printf ("%d",*(tab+i)); //7: tab+i pointe sur le iéme élément
```

  - Exemple 2:

```
int tab[10]={1,58, 7}, i;
int *ptr;
ptr=tab; //équivalent à ptr=&tabl[0];
printf ("%d\n",*(ptr)); //1: ptr pointe sur le 1er élément de tab
printf ("%d\n",*(ptr+1)); //58: ptr+1 point esur le 2éme élément de tab
i=2;
printf ("%d",*(ptr+i)); //7: ptr+i point sur l'iéme élément de tab
```

Les pointeurs et tableaux

- Relation entre pointeur et tableau (suite):
  - Exercice:
    - Déclarer un tableau de 5 éléments {15, 8, -8, 7, 9}
    - Afficher en utilisant une boucle **for** tous les éléments du tableau:
      1. Utilisant les indexes
      2. Utilisant les pointeurs

LES POINTEURS ET TABLEAUX

- Relation entre pointeur et tableau (suite):
  - Solution

```
#include <stdio.h>
main()
{
    int tab[5]={15, 8, -8, 7, 9}, i;
    for (i=0; i<5; i++)
    {
        printf ("%d\t",tab[i]);
    }
    printf("\n");
    for (i=0; i<5; i++)
    {
        printf ("%d\t",*(tab+i));
    }
}
```

Occupation mémoire

- Taille d'une variable
  - Pour toute variable créée, une zone mémoire lui sera associée, servant à stocker son contenu

  ➔ La taille dépend du type de la variable:

    - char : 1 octet
    - int : 2 ou 4 octets (selon l'architecture du système)
    - float : 4 octets
    - double : 8 octets
    - etc

Occupation mémoire

- Taille d'une variable
  - L'opérateur « sizeof()» retourne la taille en octets d'un type ou d'une variable passée en paramètre.
    sizeof(type) ***ou bien*** sizeof(nom_variable)
  - Exemple:

```
double x, tab[]={1,2,5,8};
printf("Sur mon système un 'double' fait %d octets", sizeof(x)); ➔
  taille de la variable x: 8 octets
//équivalent à
printf("Sur mon système un 'double' fait %d octets", sizeof(double));
            ➔ taille de la variable du type double: 8 octets
//Astuce: taille d'un tableau: sizeof(tab)/sizeof(type)
printf ("Taille du tableau est: %d", sizeof(tab)/sizeof(double));
```

ALLOCATION DYNAMIQUE

- Cas d'un pointeur:
  - Contrairement aux variables, un pointeur n'a pas d'existence tant qu'on ne l'a pas initialisé.
  - Il existe en C, des fonctions permettant d'allouer la mémoire à un pointeur.

  ➔ La fonction ***malloc*** de la bibliothèque « stdlib » permet de réserver de la mémoire au cours d'exécution d'un programme.
- Syntaxe:

  malloc(<NombreOctets>)
  - Elle renvoie l'adresse d'un bloc mémoire de taille indiquée en argument <NombreOctets>.
  - Elle renvoie 0 dans le cas d'échec.

ALLOCATION DYNAMIQUE

- Cas d'un pointeur: allocation (Réservation) de mémoire
  - Exemple 1: Sans allocation de mémoire

```
#include  <stdio.h>
#include<stdlib.h>
main()
{
   char *ptrChr;
   printf("Entrer un texte de 10 caractéres\n");
   scanf("%s",ptrChr);
   printf("Texte saisi: %s\n",ptrChr);
}
```

➔ Erreur d'exécution: après la saisi du texte par l'utilisateur, le programme va générer une erreur (Bug). Car le pointeur ptrChr n'a d'espace mémoire réservé

ALLOCATION DYNAMIQUE

- Cas d'un pointeur: allocation (Réservation) de mémoire
  - Exemple 2: Avec allocation de mémoire

```
#include  <stdio.h>
#include<stdlib.h>
main()
{
   char *ptrChr;
   //Réservation de la mémoire
   ptrChr = (char*)malloc(255); /* lui réservé 255 octets en mémoire */
   printf("Entrer un texte: \n");
   scanf("%s",ptrChr);
   printf("Texte saisi: %s\n",ptrChr);
}
```

➔ Le texte saisi sera affiché sans problème

ALLOCATION DYNAMIQUE

- Cas d'un pointeur: Libération de la mémoire réservée:
  - La fonction **free** de la biblio. « stdlib » permet la libération de l'emplacement mémoire réservé par malloc.

Syntaxe:

free(<Pointeur>);

Allocation dynamique

- Cas d'un pointeur: allocation/libération de mémoire
  - Exemple:

```
char *ptrChr;
int *ptrIn;
float *ptrNotes;
//Réservation de la mémoire
ptrChr = (char*)malloc(10); /* réserve 10 octets mémoire (10 caractères) */
ptrIn = (int*)malloc(20);     /*réserve  20 octets, soit la place pour 5 entiers*/
ptrNotes =(float*)malloc(16); /*réserve 16 octets, soit la place pour 4 réels */
//Libération de la mémoire
free(ptrChr);      /* libère l'espace mémoire réservé pour ptr*/
free(ptrIn);       /* libère l'espace mémoire réservé pour le pointeur notes */
free(ptrNotes); /* libère l'espace mémoire réservé pour le pointeur notes */
```