Pointeurs.

Adresses comme données.

valeurs gauches (ang. lvalues)

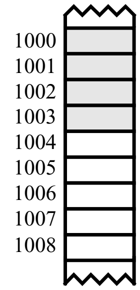
x = 1.0;

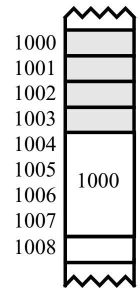
intarray[2] = 17;

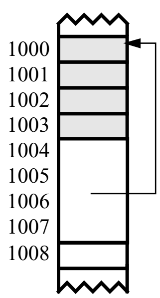
1. Chaque valeur gauche est sauvegardée quelque part dans la mémoire, donc elle possède une adresse.
2. Une fois déclarée, l'adresse d'une valeur gauche ne change jamais, bien que le contenu de la valeur gauche peut changer.
3. Dépendement du type de valeur qu'elles contiennent, différentes valeurs gauches peuvent être stockées dans les zones de mémoire de différentes tailles.
4. La valeur gauche est en soi une valeur, qui peut être manipulée et stockée dans la mémoire.

Example.

int i;







Déclaration de variables pointeurs en C.

int *p;

char *cptr;

La syntaxe de la déclaration du pointeur.

type-de-base * *variable-pointeur;*

où:
type-de-base dénote le type de la valeur vers laquelle le pointeur pointe,
variable-pointeur dénote le nom de la variable déclarée

int *p1, *p2;

int *p1, p2;

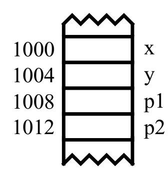
Opérations fondamentales sur les pointeurs.

& - adresse de

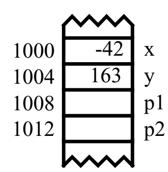
* - valeur pointée par (opérateur de déréférencement)

Example.

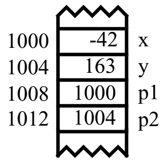
int x, y;
int *p1, *p2;

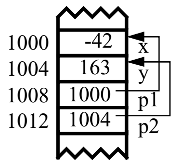


x = -42;
y = 163;

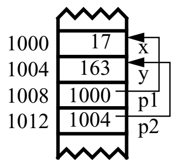


p1 = &x;
p2 = &y;

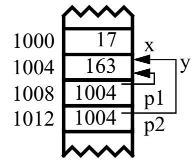




*p1 = 17;



p1 = p2;



p1 = p2;

*p1 = *p2;

Pointeur NULL.

Example.

```
main ()
{
int i, j;
int * p, * q;

i = 7;
p = &i;
          printf ("apres p = &i;\n");
          printf ("*p = %d\n\n", *p);
```



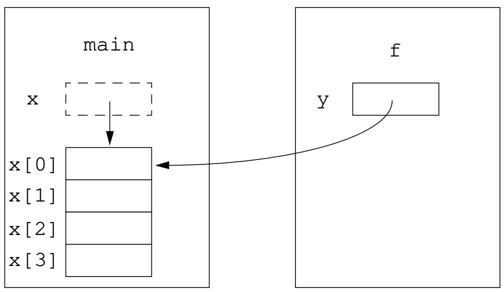
```
i = 8;

j = *p;
          printf ("apres j = *p;\n");
          printf ("j = %d\n", j);
          printf ("*p = %d\n\n", *p);
*p = 9;
          printf ("apres *p = 9;\n");
          printf ("*p = %d\n", *p);
          printf ("j = %d\n\n", j);
q = &j;
          printf ("apres q = &j;\n");
          printf ("*q = %d\n\n", *q);
*p = *q;
          printf ("apres *p = *q;\n");
          printf ("*p = %d\n", *p);
          printf ("*q = %d\n", *q);
          printf ("p = %p\n", p);
          printf ("q = %p\n\n", q);
*p = 10;

p = q;
          printf ("apres p = q;\n");
          printf ("*p = %d\n", *p);
          printf ("*q = %d\n", *q);
          printf ("p = %p\n", p);
          printf ("q = %p\n\n", q);

/* a la fin */
          printf ("a la fin\n");
          printf ("&i = %p\n", &i);
          printf ("&j = %p\n", &j);
          printf ("&p = %p\n", &p);
          printf ("&q = %p\n", &q);
          printf ("p = %p\n", p);
          printf ("q = %p\n\n", q);
}
```



```
apres p = &i;
*p = 7

apres j = *p;
j = 8
*p = 8

apres *p = 9;
*p = 9
j = 8

apres q = &j;
*q = 8

apres *p = *q;
*p = 8
*q = 8
p = dffff27c
q = dffff278

apres p = q;
*p = 8
*q = 8
p = dffff278
q = dffff278

a la fin
&i = dffff27c
&j = dffff278
&p = dffff274
&q = dffff270
p = dffff278
q = dffff278
```

Les tableaux passées en paramètres à des fonctions.

```
main()
{
   int x[4];

   f(x);
}

f(y);
int y[ ];
{
   . . .
}
```



Le passage de paramètres par référence.

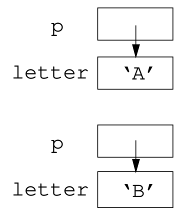
Paramètres de retour.

- les variables tableaux

- variables globales

- pointeurs

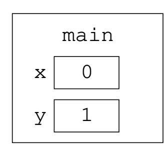
```
/* 1 */    char letter,  *p;
/* 2 */    p = &letter;
/* 3 */    letter = 'A';
```



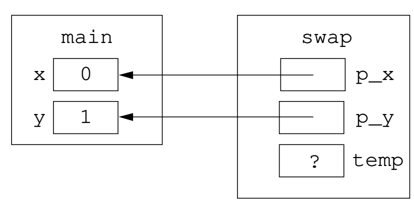

```
/* 4 */    *p = 'B';

scanf ("%d", &x);
```


Example.

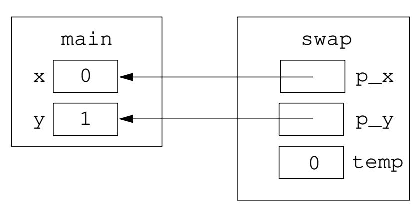
```
main ()
{
   int x, y;

   x = 0;
   y = 1;
   printf ("Dans main avant swap; x = %d, y = %d\n", x, y );
   swap (&x, &y);
   printf ("Dans main apres swap; x = %d, y = %d\n", x, y );
   exit (0);
}

void swap (int *p_x, int *p_y)
{
   int temp;
   temp = *p_x;
   *p_x = *p_y;
   *p_y = temp;
}
```

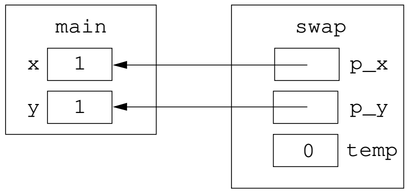
à l'appel de swap:



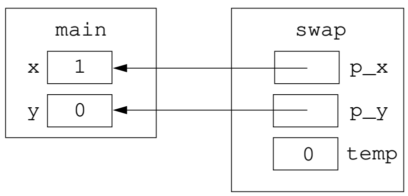
temp = *p_x;



*p_x = *p_y;



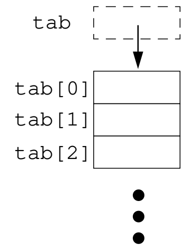
*p_y = temp;



Pointeurs et tableaux.

La relation entre pointeurs et tableaux.

tab *est défini identique à* &tab[0]



int tab[ ];

int *tab; /* une definition equivalente */

void TrierTableauEntier (int tab[], int n);

void TrierTableauEntier (int *tab, int n);

int tab[5]; /* allocation de mémoire pour stocker 5 entiers */

int *p; /* allocation de mémoire pour stocker une adresse*/

p = tab; /* p et tab sont synonymes à partir de ce moment */

Pointeurs et structures.

```
main()
{
   struct {
      int i;
      float f;
   } s, *p_s;

   p_s = &s;
   p_s->i = 5;  /* ou (*p_s).i  = 5*/
   p_s->f = 1.1;
}
```

Allocation dynamique.

- allocation statique

- allocation automatique

- allocation dynamique

malloc(10)

Le type void *

int *ip;

char *cp;

void *vp;

void *malloc (int nBytes);

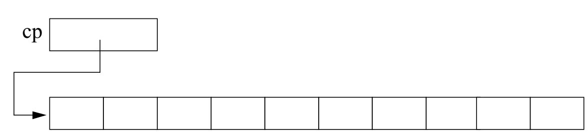
char *cp;

cp = malloc(10);

cp = (char*) malloc(10);

Tableaux dynamiques.

cp = (char*) malloc(10);



1. Déclarer une variable pointeur au type de base.
2. Allouer la mémoire dynamique (à l'aide de malloc).
3. Affecter le resultat de malloc à la variable pointeur.

int *arr;

```
arr = malloc (10 * sizeof(int) );

arr = malloc (10 * sizeof(int) );
if (arr == NULL) Error (“Il n’y a plus de memoire”);

arr = GetBlock (10 * sizeof(int) ); /* GetBlock de genlib.h */


free (arr);
```