Les pointeurs

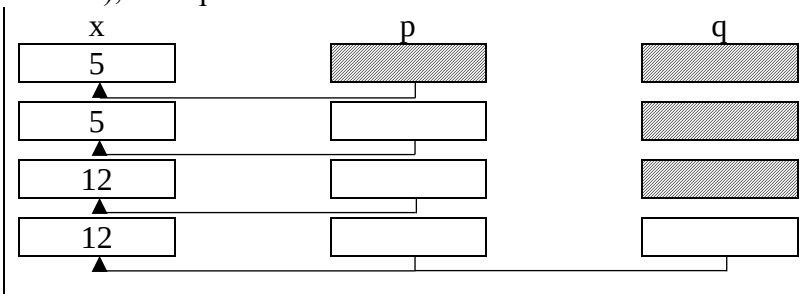
Un pointeur contient l'adresse mémoire d'un objet. Il permet de passer un objet par référence et non par valeur, l'écriture de code est plus compacte et plus efficace. Cependant le code devient moins lisible.
Pour déclarer un pointeur on utilise le caractère « * » (on peut le placer sur le type ou sur le nom de la variable). Un pointeur peut désigner n'importe quelle variable (statique, dynamique ou constante). Il est même possible de dénoter l'adresse d'une fonction.

L'opérateur « * » permet l'indirection (déréférencement), alors que l'opérateur « & » renvoie l'adresse de la variable (référencement), exemple :

```
int x = 5; p* = NULL;
q* = NULL;
p = &x;

*p = 12;

q = p;
```



Soit si on a deux pointeurs, *p* et *q*, qui pointent vers le même type :

&p == &q ⇒ p == q ⇒ *p ==*q

Pointeur sur des structures

```
struct{
  int a, b;
} x, *p = &x;
```

x et *p désigne la structure.
Accès à un champ, par exemple a : `x.a` ⇔ `(*p).a` ⇔ `p->a`

Opérations arithmétiques

Comparaison :	= =, !=, <, <=, >=, >	✔
Addition :	pointeur+*entier* ⟶ pointeur	✔
Soustraction :	pointeur-*entier* ⟶ pointeur	✔
Différence :	pointeur-pointeur ⟶ entier	⚠ Permet de calculer le nombre d'éléments de ce type entre les deux adresses

Efficacité des pointeurs

Plus efficace car pour accéder à une case d'un tableau cela revient à :

a[i] ⇔ *(a+i)
a[0] ⇔ *a (moins chère)
On va donc se servir directement du pointeur, et appeler chaque case comme si c'était la première, exemple :

Avant	Maintenant

`char *ptr = NULL;`
`*ptr = 'a';`
`*(ptr+1) = 'b';`
	`char *ptr = NULL;`
	`*ptr = 'a';`
	`ptr++;`
	`*ptr = 'b';`

Passage de chaines de caractères (tableaux)

Fonction void Printf(char *format, ...) qui se comporte comme printf et reconnait dans son format les séquences *%%*, *%c*, *%s*, *%x* et *%d*.

```
void afficherChiffre(int entier){
    if(entier>9)
      afficherChiffre(entier/10);
    putchar('0'+entier%10);
}
void afficherChiffreHex(int entier){
    if(entier>16)
      afficherChiffreHex(entier/16);
    int tempo = entier%16;
    if(tempo > 9){
        switch(tempo){
                case 10 : putchar('A'); break;
                case 11 : putchar('B'); break;
                case 12 : putchar('C'); break;
                case 13 : putchar('D'); break;
                case 14 : putchar('E'); break;
                case 15 : putchar('F'); break; }
    }else{ putchar('0'+entier%16); }
}
void Printf(char* s, ...){
    va_list ap; int index=0; char* argCh; int argI;
    va_start(ap,s);
    while(s[index]!='\0'){
        if(s[index]=='%'){
            switch(s[index+1]){
                case '%':
                putchar('%');
                break;
                case 'c':
                argI=va_arg(ap,int);
                putchar(argI);
                break;
                case 's':
                argCh=va_arg(ap,char*);
                int sousIndex=0;
                while(argCh[sousIndex]!='\0'){
                    putchar(argCh[sousIndex]);
                    sousIndex++;
                }
                break;
                case 'd':
                argI=va_arg(ap,int);
                afficherChiffre(argI);
                break;
                case 'x':
                argI=va_arg(ap,int);
                afficherChiffreHex(argI);
                break;
            }
            index+=2;
        }
        putchar(s[index]);
        index++;
    }
}
```

Passage d'un pointeur vers une fonction en paramètre avec *typedef*

```c
typedef int fct(int);

int carre(int entier){
    return entier*entier;
}

void map(fct f, int* tab, int taille){
    for(int i=0;i<taille;i++){
        tab[i]=f(tab[i]);
    }
}

int main(){
    int t[]={1,2,3,4,5};
    int taille = 5;
    for(int i=0;i<taille;i++){
        printf("%d\t",t[i]);
    }
    printf("\n");
    map(carre, t, taille);
    for(int i=0;i<taille;i++){
        printf("%d\t",t[i]);
    }
    printf("\n");
}
```