Fiche n°5 : Opérateurs bitwises

Ces opérateurs réalisent des opérations sur les bits de la représentation binaire d'un entier.

Attention de ne pas les confondre avec les opérateurs Booléens **&&, || et !** qui interprètent les entiers comme des valeurs Booléennes.

opérateur	résultat
a & b	et Booléen entre bits de mêmes rangs
a | b	ou Booléen entre bits de mêmes rangs
a^b	ou exclusif entre bits de mêmes rangs
~a	négation des bits
a >> n	décalage de n bits à droite
a << n	décalage de n bits à gauche

Ces opérateurs ne modifient pas leurs opérandes mais retournent un résultat qui peut être ensuite exploité.

Par exemple :

c = a & b

modifie seulement la variable c

Exemples de résultats

	décimal	base 2							
a	78	0	1	0	0	1	1	1	0
b	103	0	1	1	0	0	1	1	1
a&b	70	0	1	0	0	0	1	1	0

	décimal	base 2							
a	78	0	1	0	0	1	1	1	0
~a	177	1	0	1	1	0	0	0	1

	décimal	base 2							
a	78	0	1	0	0	1	1	1	0
a>>2	19	0	0	0	1	0	0	1	1

Exemples d'utilisation

Isoler un bit dans un mot binaire

valeurs	représentations binaires							
masque	0	0	0	0	1	0	0	0
x	?	?	?	?	b	?	?	?
masque & x	0	0	0	0	b	0	0	0

Mettre à 1 un bit sans modifier les autres

valeurs	représentations binaires							
masque	0	0	0	0	1	0	0	0
x	b7	b6	b5	b4	b3	b2	b1	b0
masque | x	b7	b6	b5	b4	1	b2	b1	b0

Exercices

Comment mettre à 0 un bit particulier d'un mot binaire ?

Comment vérifier si les trois bits de poids faibles de deux mots binaires sont tous identiques ?

Comment réaliser une multiplication par 4 avec un opérateur bitwise ?

Fiche n°6 : Conversions de types

Un même mot binaire peut représenter plusieurs informations différentes. Par exemple **10000001** représente la valeur **129** en base **2** et la valeur **-127** en codage complément à deux.

Par ailleurs, des valeurs identiques peuvent être représentées de manière différentes. Par exemple **-12** est représenté par **11111010** avec le type **unsigned char** et par **11111111 11111111 11111111 11111010** avec le type **unsigned int**.

Le compilateur utilise la notion de **type** pour faire le lien une valeur et sa représentation

Parfois, une valeur d'un certain type est convertie implicitement dans un autre.

```
char z = -12;
int a = z;
unsigned int b = z;

printf("%d %u", a,b);
```

Affiche : -12 4294967284

z est converti en **int** de même valeur par la règle de promotion des entiers, et le mot binaire obtenu est recopié dans **b**, où il représente une valeur positive.

On peut aussi **forcer la conversion** avec la notation présentée sur cet exemple.

```
char z = -12;
int a = (unsigned char)z;

printf("%d", a);
```

z est converti en caractère non signé. Sa représentation binaire ne change pas, mais est interprétée comme la valeur positive **244**.

Cette valeur est ensuite convertie en int par la règle de promotion des entiers.

Affiche : 244

Exemple d'utilisation

On souhaite convertir une fraction de deux entiers en valeur de type **double**.

```
int n = 10, d = 8;
double f = n/d;

printf("%lf", f);
```

Le résultat de la division entière est l'entier 1. Il est converti implicitement en **double**.

Affiche : 1.0

Affiche : 1.25

```
int n = 10, d = 8;
double f = (double)n/d;

printf("%lf", f);
```

n est converti explicitement en **double**, n est promu implicitement en double, le résultat de la division est donc un **double**.

Remarques

La conversion explicite de type est utilisable avec les pointeurs, mais son utilisation peut conduire à des incohérences. A réserver à des situations très particulières qui seront introduites plus tard.

La fonction **printf** ne fait pas de conversion de type, juste une interprétation de la donnée binaire transmise dans le type spécifié dans la chaîne de formatage.

```
double f=1.25;

printf("%d", f);

printf("%d",(int)f);
```

Affiche : 0

Affiche : 1

Fiche n°7 : Tableaux à une dimension

Un tableau est une suite de données de même type dont les représentations sont contiguës en mémoire.

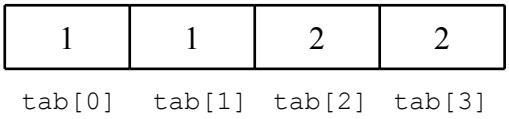
La zone mémoire contenant chaque donnée est appelée cellule. Les cellules sont numérotées à partir de 0.

Tableau de 4 cellules contenant chacune une valeur de type **int**.

Initialisation facultative, autorisée uniquement lors de la déclaration du tableau.

```
int tab[4] = {1,1,2,2};
```

Chacune des cellules du tableau peut être utilisée comme une variable.



Le nom du tableau (**tab** dans notre exemple) représente **l'adresse mémoire** de la première cellule est est donc de type **pointeur** (sur **int** dans notre exemple).

```
int* p = tab;
p[1]=0;
```

équivalent à **tab[1] = 0**

Arithmétique des pointeurs

On peut additionner un pointeur p et un entier i. Le résultat est l'adresse p + i*sizeof(*p), autrement dit p + i fois la taille en octets de la donnée pointée par p.

On peut soustraire un pointeur q à un pointeur p de même type. Le résultat est l'entier (p-q)/sizeof(*p).

conséquences

notations équivalentes	
p[i]	*(p+i)
&(p[i])	p+i
p[0]	*p
p	&(p[0])

Passage de tableaux en paramètre

Pour passer un tableau en paramètre à une fonction, il faut lui transmettre **l'adresse de sa première cellule** (le nom du tableau représente cette adresse) **et la taille** du tableau, car sinon la fonction ne pourrait pas connaître le nombre de cellules.

Grâce à cette adresse, la fonction peut accéder en lecture et écriture aux cellules du **tableau original** (et non à une copie comme lorsqu'on passe en paramètre une valeur de type primitif).

Exemple

```
double moyenne(double t[], int n)
{
  int i;
  double sum=0.0;
  for(i=0; i<n; i++)
  {
    sum = sum+t[i];
  }
  return sum/n;
}
```

la notation **double* t** est aussi possible

```
int main()
{
  double tab[] = {10.0,12.0,17.0};

  printf("%lf",moyenne(tab,3));

  return 0;
}
```

Affiche 13.0

Remarque

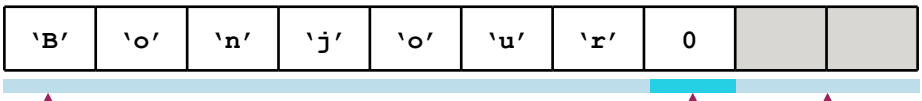
Dans les anciennes versions des compilateurs C, la taille d'un tableau doit être une constante. Dans les versions les plus modernes (à partir de C99) la taille initiale peut être une variable, mais ne peut être modifiée après la création du tableau.

Fiche n°8 : Chaînes de caractères

Une chaîne de caractères est une suite de caractères pouvant former par exemple un mot, une phrase.

En langage C, une chaîne est représentée par un tableau de caractères de longueur au moins égale à 1 plus le nombre de caractères de la chaîne.

La partie utile est terminée par la valeur 0, à ne pas confondre avec la caractère '0'.




```
char s[10] = "Bonjour";
```

codes ascii

marqueur de fin de chaîne

partie non utilisée du tableau

Quelques fonctions standards

signature	usage
`int strlen(char* s)`	Retourne la longueur de s.
`char* strcpy(char* dest, char* src)`	Copie src dans dest. Écrase l'ancienne valeur.
`char* strcat(char* dest, char* src)`	Concaténation de src à dest.
`int strcmp(char* s1, char* s2)`	Comparaison lexicographique. Retourne un entier négatif si s1 < s2, positif si s2 > s1, 0 si les chaînes sont identiques.

Exemple

```
char s1[10] = "aaa";
char s2[10] = "bbb";

strcat(s1,s2);

printf("%s %s",s1,s2);
```

Attention : le tableau de destination doit avoir une capacité suffisante pour recevoir le résultat.

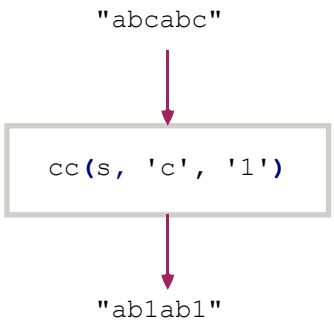
Affiche **aaabbb bbb**

Passer une chaîne en paramètre

Exemple

Cette fonction remplace toutes les occurrences d'un caractère par un autre caractère.

```
void cc(char* s, char a, char b)
{
  int n = strlen(s);
  int i;
  for(i=0; i<n; i++)
  {
    if(s[i]==a) s[i]=b;
  }
}
```



Attention. Il faut garder à l'esprit que l'identificateur d'une chaîne représente son adresse mémoire (comme c'est le cas pour tout tableau). Une des conséquences est que l'opérateur == ne compare pas le contenu de deux chaînes, mais leurs adresses. Pour comparer des chaînes, il faut utiliser la fonction **strcmp**.

Fiche de préparation n°2

Vous devez indiquez les valeurs affichées lors de l'exécution du programme suivant, **sans utiliser d'ordinateur !** Le but est de vérifier que comprenez suffisamment bien les notions concernées pour être capable de prévoir le résultat d'un programme qui les utilise.

```
void f(int* t1, int* t2, int n)
    {
        int i;
        for(i=0; i<n; i++)
        {
            if(t1[i]<t2[i]) t1[i]=t2[i];
        }
    }
```

```
void affiche(int tab[], int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        printf("%d ",tab[i]);
    }
    printf("\n");
}
```

Cette fonction permet l'affichage du contenu d'un tableau.

```
char* g(char s[])
{
    int i=0;
    while(s[i])
    {
        if((s[i]>='a')&&(s[i]<='z'))
            return s+i;
        else
            i++;
    }
    return NULL;
}
```

```
int main()
{
    printf("%x\n",(1<<3)^255);

    int tab1[] = {1,2,3,4,5,6};
    int tab2[] = {6,5,4,3,2,1};
    f(tab1+1,tab2,4);

    affiche(tab1,6);

    char s[] = "123soleil";
    char* p = g(s);
    *p = *p + ('A'-'a');

    printf("%s\n",s);

    return 0;
}
```

Le format d'affichage **%s** permet à **printf** d'afficher une chaîne de caractère.