

Chapitre 6:

Les opérateurs et les expressions en C

Notions d'opérateur et d'expression

- Une **expression** est une suite syntaxiquement correcte d'opérateurs et d'opérandes.
- Une expression ramène toujours une valeur, même si la valeur n'est pas utilisée.
- Une expression est fausse si son résultat est nul. Elle est vraie si son résultat est non nul.
- **Exemple**

$$x+2*y;$$

Notions d'opérateur et d'expression

- Une instruction simple est
 - ✓ soit une instruction de contrôle (voir chapitre suivant),
 - ✓ soit une expression suivie de `;`.
- Une instruction simple est toujours terminée par un `;`.
- D'autre part, des instructions pouvant éventuellement faire intervenir des expressions, comme, par exemple, l'instruction d'affectation :

`y = a * x + b ;`

ou encore l'instruction d'affichage :

`printf("valeur %d", n + 2*p) ;`

Les opérateurs

- Les opérateurs sont des commandes qui s'appliquent à des opérandes, qui sont des constantes, des variables ou des expressions (voir la section suivante).
- Le langage C dispose de plus de 40 opérateurs. Un critère de classification de ces derniers pourra être le nombre d'opérandes manipulés par un opérateur.
 - i. Les opérateurs **unaires** admettent un unique opérande. **Exemple:** `&x`.
 - ii. Les opérateurs **binaires** possèdent deux opérandes. **Exemple** `a+b`.
 - iii. Les opérateurs **ternaires** traitent trois opérandes. L'unique opérateur ternaire en C est l'opérateur conditionnel (voir plus loin).

Les opérateurs

- On peut aussi classer les opérateurs selon la nature des opérations qu'ils exécutent ou selon la nature de leurs opérandes.
 - i. Opérateurs arithmétiques.
 - ii. Opérateurs de comparaisons.
 - iii. Opérateurs logiques.

Opérateurs arithmétiques

- les opérateurs de cette catégorie procèdent des opérations arithmétiques sur leurs opérandes. Le tableau suivant affiche les opérateurs arithmétiques groupés selon le nombre de leurs opérandes.

Binaires		
Opérateur	Signification	Exemple
+	Addition	$X+Y$
-	Soustraction	$X-Y$
*	Multiplication	$X*Y$
/	Division	X/Y
%	Modulo	$X\%Y$
Unaire		
-	Négation	$-X$

Conversions implicites

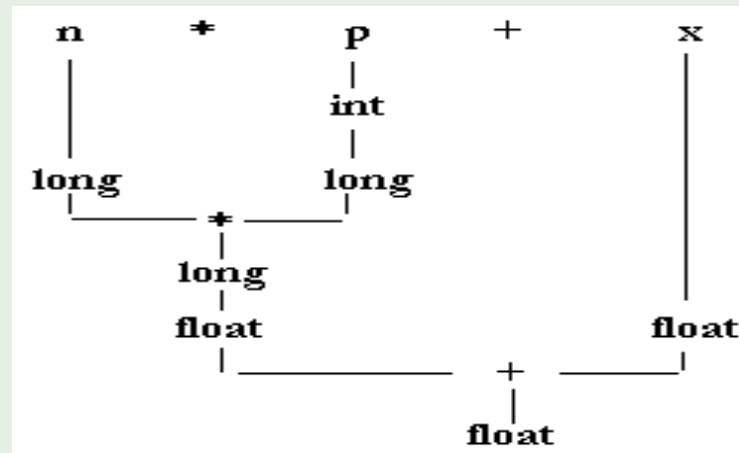
- les opérateurs arithmétiques ne sont définis que lorsque leurs deux opérandes sont de même type
- Dans le cas des expression de type mixte, le compilateur met en place des conversions implicites
- Une conversion telle que `int` \rightarrow `float` se nomme une « conversion d'ajustement de type ». Une telle conversion ne peut se faire que suivant une 'hiérarchie' qui permet de ne pas dénaturer la valeur initiale, à savoir:

`int` \rightarrow `long` \rightarrow `float` \rightarrow `double` \rightarrow `long double`

Conversions d'ajustement de type

Exemple

Si n est de type **long**, p de type **int** et x de type **float**, l'expression : $n * p + x$ sera évaluée suivant ce schéma :

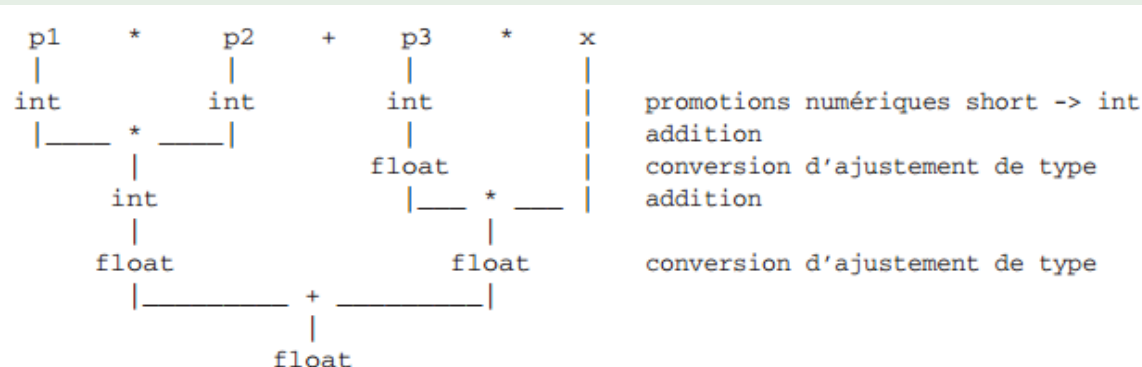


Les promotions numériques

- Les opérateurs numériques ne sont pas définis pour les types `char` et `short`.
- Le langage C prévoit tout simplement que toute valeur de l'un de ces deux types apparaissant dans une expression est d'abord convertie en `int`.

Exemple

Si `p1`, `p2` et `p3` sont de type `short` et `x` de type `float`, l'expression : `p1 * p2 + p3 * x` est évaluée comme l'indique le schéma ci-après :



Opérateurs de comparaison

- Les opérateurs de comparaison en C font partie des opérateurs binaires. Ils comparent les valeurs de leurs opérandes. Le résultat de la comparaison est une valeur booléenne, c'est à dire 0 ou 1.

Opérateur	Relation	Exemple
<code>==</code>	opérateur1 égal à opérateur 2 ?	<code>X == Y</code>
<code>!=</code>	opérateur1 différent de opérateur2 ?	<code>X != Y</code>
<code><=</code>	opérateur1 inférieur ou égal à opérateur2 ?	<code>X <= Y</code>
<code>>=</code>	opérateur1 supérieur ou égal à opérateur2 ?	<code>X >= Y</code>
<code><</code>	opérateur1 inférieur à opérateur2 ?	<code>X < Y</code>
<code>></code>	opérateur1 supérieur à opérateur2 ?	<code>X > Y</code>

Opérateurs de comparaison

Exemple

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    float X, Y;
    printf(" Ce programme compare les valeurs de deux nombres.\n");
    printf("Entrez deux nombres entiers :\n");
    printf(" X = ");
    scanf("%f",&X);
    printf(" Y = ");
    scanf("%f",&Y);

    /* Affichage des résultats */
    printf("%LG egal a %LG ?--> %d\n",X,Y,X==Y);
    printf("%LG different de %LG ? --> %d\n",X,Y,X!=Y);
    printf("%LG inferieur ou egal a %LG ? --> %d\n",X,Y,X<=Y);
    printf("%LG supsrieur ou egal a %LG ? --> %d\n",X,Y,X>=Y);
    printf("%LG inferieur a %LG ? --> %d\n",X,Y,X<Y);
    printf("%LG supsrieur a %LG ? --> %d\n",X,Y,X>Y);

    system("PAUSE");
    return 0;
}
```

Opérateurs de comparaison

Exemple

```

C:\Documents and Settings\lamnii\Bureau\TPC\tp16.exe
Ce programme compare les valeurs de deux nombres.
Entrez deux nombres entiers :
X = -5
Y = 3
-5 egal a 3 ? --> 0
-5 different de 3 ? --> 1
-5 inferieur ou egal a 3 ? --> 1
-5 supsrieur ou egal a 3 ? --> 0
-5 inferieur a 3 ? --> 1
-5 supsrieur a 3 ? --> 0
Appuyez sur une touche pour continuer...

```

```

C:\Documents and Settings\lamnii\Bureau\TPC\tp16....
Ce programme compare les valeurs de deux nombres.
Entrez deux nombres entiers :
X = 7
Y = 7
7 egal a 7 ? --> 1
7 different de 7 ? --> 0
7 inferieur ou egal a 7 ? --> 1
7 supsrieur ou egal a 7 ? --> 1
7 inferieur a 7 ? --> 0
7 supsrieur a 7 ? --> 0
Appuyez sur une touche pour continuer...

```

Opérateurs logiques

- Les opérateurs logiques en C effectuent les opérations classiques de la logique. ET (AND), OU (OR) et NON (NOT). Grâce à eux, il est possible de relier logiquement des instructions.
- Il y a trois opérateurs logiques. Deux d'entre eux sont binaires et l'un est unaire.

Opérateur		Relation	Exemple
binaire	&&	ET	X && Y
	 	OU	X Y
unaire	!	NON	!X

Opérateurs logiques

Exemple : && (Et logique)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int a=2, b=7, c=-5;
    int test1, test2, test3, test4;

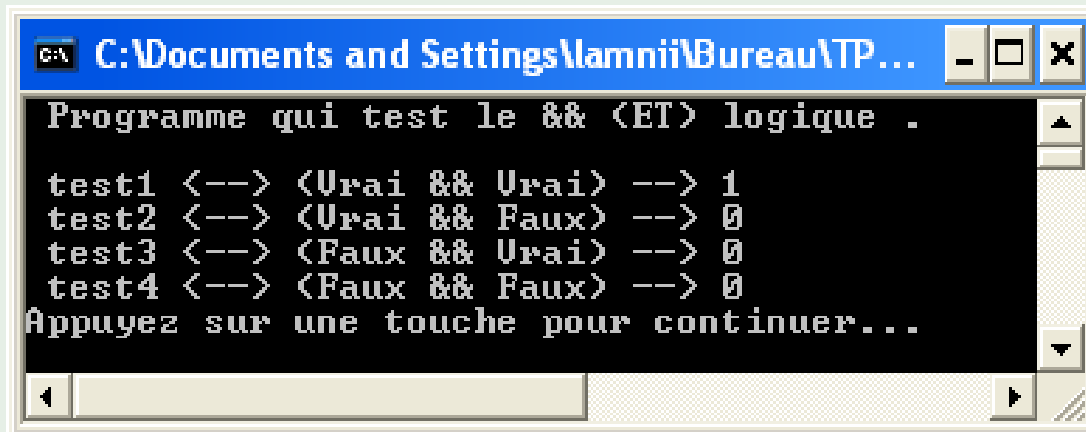
    test1=(a>=c) && (b>c);
    test2=(a>=c) && (b<c);
    test3=(a<=c) && (b>c);
    test4=(a<=c) && (b<c);

    printf(" Programme qui test le && (ET) logique .\n\n");
    printf(" test1 <--> (Vrai && Vrai) --> %d\n",test1);
    printf(" test2 <--> (Vrai && Faux) --> %d\n",test2);
    printf(" test3 <--> (Faux && Vrai) --> %d\n",test3);
    printf(" test4 <--> (Faux && Faux) --> %d\n",test4);

    system("PAUSE");
    return 0;
}
```

Opérateurs logiques

Exemple : && (Et logique)



```
C:\Documents and Settings\lamnii\Bureau\TP...
Programme qui test le && (ET) logique .
test1 <--> <Urai && Urai> --> 1
test2 <--> <Urai && Faux> --> 0
test3 <--> <Faux && Urai> --> 0
test4 <--> <Faux && Faux> --> 0
Appuyez sur une touche pour continuer...
```

Opérateurs logiques

Exemple : || (Ou logique)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int a=2, b=7, c=-5;
    int test1, test2, test3, test4;

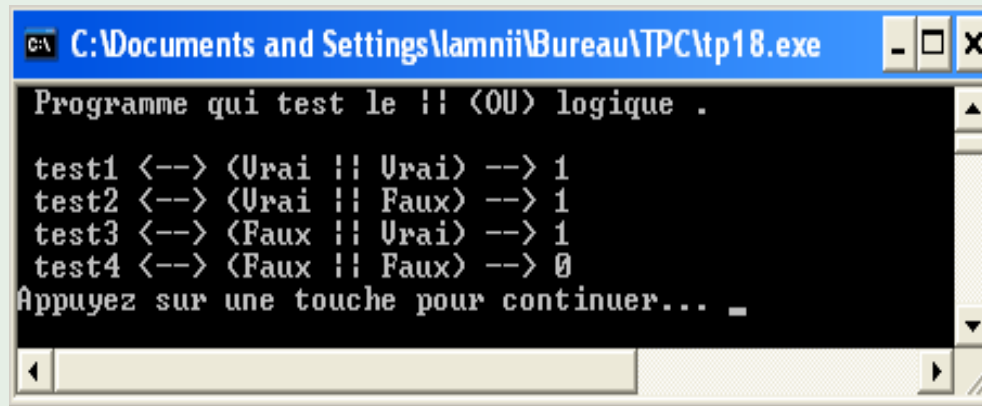
    test1=(a>=c) || (b>c);
    test2=(a>=c) || (b<c);
    test3=(a<=c) || (b>c);
    test4=(a<=c) || (b<c);

    printf(" Programme qui test le || (OU) logique .\n\n");
    printf(" test1 <--> (Vrai || Vrai) --> %d\n",test1);
    printf(" test2 <--> (Vrai || Faux) --> %d\n",test2);
    printf(" test3 <--> (Faux || Vrai) --> %d\n",test3);
    printf(" test4 <--> (Faux || Faux) --> %d\n",test4);

    system("PAUSE");
    return 0;
}
```


Opérateurs logiques

Exemple : || (Ou logique)



```
C:\Documents and Settings\lamnii\Bureau\TPC\tp18.exe
Programme qui test le || (OU) logique .
test1 <--> (Vrai || Vrai) --> 1
test2 <--> (Vrai || Faux) --> 1
test3 <--> (Faux || Vrai) --> 1
test4 <--> (Faux || Faux) --> 0
Appuyez sur une touche pour continuer... _
```

Opérateurs logiques

Exemple : ! (Non logique)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int a=2, b=-1;
    int test1, test2;

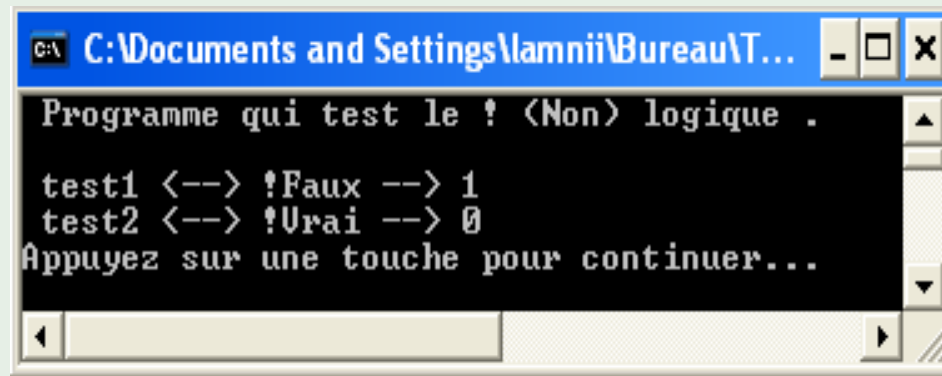
    test1=(a==b); /* Faux (0) */
    test2=(a!=b); /* Vrai (1) */

    printf(" Programme qui test le ! (Non) logique .\n\n");
    printf(" test1 <--> !Faux --> %d\n", !test1);
    printf(" test2 <--> !Vrai --> %d\n", !test2);

    system("PAUSE");
    return 0;
}
```

Opérateurs logiques

Exemple : || (Ou logique)



```
C:\Documents and Settings\lamnii\Bureau\T...
Programme qui test le ! <Non> logique .
test1 <--> !Faux --> 1
test2 <--> !Vrai --> 0
Appuyez sur une touche pour continuer...
```

Les opérateurs ++ et --

- Dans des programmes écrits en C ou en un autre langage, on rencontre souvent des expressions (instructions) telles que:

`i=i+1;`

`n=n-1;`

Qui « incrémentent » ou qui « décrémentent » de **1** la valeur d'une « variable ».

- En C, ces actions peuvent être réalisées par des opérateurs « unaires ». Ainsi, l'expression

`++i;`

a pour effet d'incrémenter de 1 la valeur de i, et sa valeur est celle de i après *incrémentat*ion.

Les opérateurs ++ et --

- Ainsi, si la valeur de *i* est 7, l'expression:
$$n=++i-7;$$
 affectera à *i* la valeur 8 et à *n* la valeur 1.
- En revanche, lorsque cet opérateur est placé après *i*, la valeur de l'expression correspondante est de la variable avant incrémentation. Ainsi, si *i* vaut 7, l'expression
$$n=i++-7;$$
 affectera à *i* la valeur 8 et à *n* la valeur 0 (car ici la valeur de *i*++ est 7).

Les opérateurs ++ et --

- On dit que ++ est:
 - ✓ Un opérateur de **pré-incrémentation** lorsqu'il est placé **à gauche** de la « lvalue » sur laquelle est portée.
 - ✓ Un opérateur de **post-incrémentation** lorsqu'il est placé **à droite** de la « lvalue » sur laquelle est portée.
- De la même manière il existe un opérateur de décrémentation noté --

Les opérateurs ++ et --

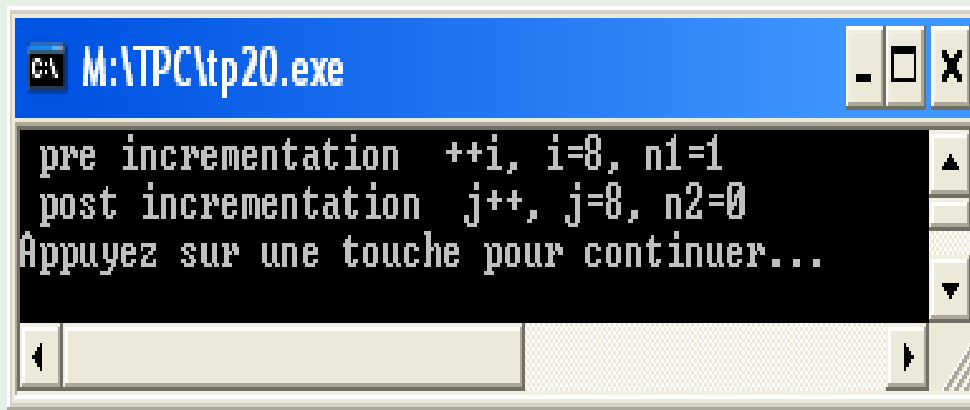
Exemple : i++ et ++i

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int i, j, n1, n2;
    i=7;
    j=7;
    /* pré incrémentation  ++i */
    n1 = ++i - 7;
    printf(" pré incrémentation  ++i, i=%d, n1=%d\n",i,n1);
    /* post incrémentation  j++ */
    n2 = j++ - 7;
    printf(" post incrémentation  j++, j=%d, n2=%d\n",j,n2);

    system("PAUSE");
    return 0;
}
```

Les opérateurs ++ et --

Exemple : i++ et ++i



```
M:\TPC\tp20.exe  
pre incrementation ++i, i=8, n1=1  
post incrementation j++, j=8, n2=0  
Appuyez sur une touche pour continuer...
```


Les opérateurs ++ et --

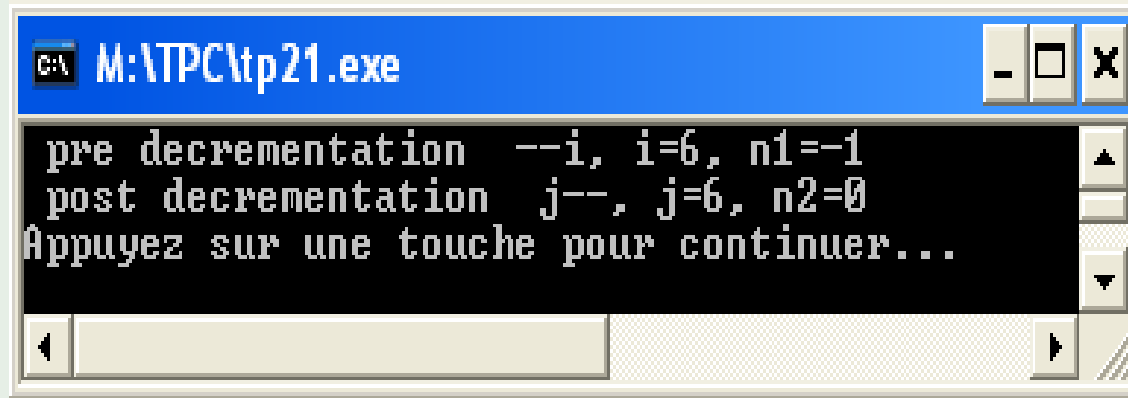
Exemple : i-- et --i

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int i, j, n1, n2;
    i=7;
    j=7;
    /* pré décrémentation  --i */
    n1 = --i - 7;
    printf(" pre decrementation  --i, i=%d, n1=%d\n", i, n1);
    /* post décrémentation  j-- */
    n2 = j-- - 7;
    printf(" post decrementation  j--, j=%d, n2=%d\n", j, n2);

    system("PAUSE");
    return 0;
}
```

Les opérateurs ++ et --

Exemple : i++ et ++i



```
M:\TPC\tp21.exe  
pre decrementation --i, i=6, n1=-1  
post decrementation j--, j=6, n2=0  
Appuyez sur une touche pour continuer...
```

Les opérateurs d'affectation élargie

- D'une manière générale, C permet de condenser les affectations de la forme :

$lvalue = lvalue \text{ opérateur } expression$

en :

$lvalue \text{ opérateur} = expression$

- Exemple: Les affectations

$i = i + k; \text{ et } a = a * b;$

peuvent être remplacées respectivement par :

$i += k; \text{ et } a *= b;$

L'opérateur de cast

- L'opérateur de cast permet au programmeur de forcer la conversion d'une expression quelconque dans un type de son choix.
- Par exemple, n et p sont des variables entières, l'expression :

`(double)(n/p);`

aura comme valeur celle de l'expression entière n/p convertie en double.

- La notation `(double)` correspond en fait à un opérateur unaire dont le rôle est d'effectuer la conversion dans le type double de l'expression sur laquelle il porte.