Enseignant
Manuel clergue

2

Introduction au langage C
Les instructions

2022

esiea

1 Eléments d'un programme

Expressions

- Une expression est :
  - une valeur littérale
  - une variable
  - le résultat de l'appel d'une fonction
  - la composition de une, deux ou trois expressions par un opérateur
- Une expression à une valeur (en général)

Opérations :

symbole	opération
+	addition
−	soustraction
*	multiplication
/	division (réelle ou euclidienne)
%	reste de la division euclidienne

Exemple :

`9 / 4` vaut 2 `9 % 4` vaut 1 `9. / 4.` vaut 2.25

Fonctions mathématiques : définies dans la “library” math.h

mot-clé	fonction	mot-clé	fonction
sqrt	racine carrée	sin	sinus
pow	puissance	cos	cosinus
		atan	arc tangente
log	logarithme népérien	floor	troncature
exp	exponentielle		(réel → entier)
abs	valeur absolue	ceil	entier supérieur
fabs	valeur absolue (réel)		(réel → entier)

Opérateurs relationnels :

symbole	opérations
==	égal à
! =	différent de
<	inférieur à (strictement)
>	supérieur à (strictement)
<=	inférieur ou égal à
>=	supérieur ou égal à

Opérateurs logiques :

symbole	opérations
&&	et (conjonction)
||	ou (disjonction)
!	non (négation)

Exemple :

```
(x<3) && ((y!=(x-1)) || (y>2))
```

est vrai quand x vaut 1 et y vaut 7 ; c'est faux quand x vaut 2 et y vaut 1

Attention :

Ne pas confondre les opérateurs logiques et les opérateurs sur la représentation des nombres

`x && y` est différent de `x & y`

Affectations

symbole	opérations
$=$	affectation
$+=, *=, \ldots$	combinaison opération puis affectation
$i++$	post-incrémentation
$i--$	post-decrémentation
$++i$	pré-incrémentation
$--i$	pré-decrémentation

Manipulation de bits (bitwise operators

symbole	opérations
<<	décalage vers la gauche
>>	décalage vers la droite
&	et bit à bit
|	ou bit à bit
~	négation bit à bit
^	ou exclusif bit à bit

Opérateur conditionnel ternaire

- Il existe en C un opérateur ternaire (à trois opérandes)
- `a?b:c`, avec a,b et c trois expressions
- si a s'évalue à vrai, `a?b:c` prend la valeur de b
- sinon, `a?b:c` prend la valeur de c

Une instruction est une chaîne d'expression se terminant par un `;`

```
1 int a=6, b=7; // Instruction composée de 2 expressions
```

Un bloc d'instructions est une séquence d'instructions délimitée par une paire d'accolades

```
1 {
2 // instruction 1
3 // instruction 2
4 // etc ...
5 }
```

Instruction conditionnelle (if/else)

- conditionnelle simple

```
1  if(cond){
2          // ...
3  }
```

- conditionnelle avec alternative

```
1  if(cond){
2          // ...
3  }
4  else{   // if(cond)
5          // ...
6  }
```

Un exemple

```
1  if (age < 3){ // Test de l'instruction
2          printf ("On n'est pas des imbeciles!\n");
3  }
4  else{ // Sinon (if (age < 3))
5          if (age <= 30){
6                  printf ("Jeunot!\n");
7          }
8          else{ // (if (age <= 30)
9                  if (age < 50){
10                         printf ("Ca passe encore\n");
11                 }
12                 else { // (if (age < 50))
13                         printf ("Desole\n");
14                 }
15 }
```

. . .ou bien

```
1  if (age < 3){ // Test de l'instruction
2          printf ("On n'est pas des imbeciles!\n");
3  }
4  if (age >= 3 && age <= 30){
5          printf ("Jeunot!\n");
6  }
7  if (age > 30 && age < 50){
8          printf ("Ca passe encore\n");
9  }
10 if (age >= 50){
11         printf ("Desole\n");
12 }
```

Instruction conditionnelle - Switch/case

- utile pour faire des tests (d'égalité] successifs sur une valeur entière
- dès qu'un *case* est validé, éxécution de toutes les instructions qui suivent
- dès qu'un *break* est rencontré, l'éxécution du *switch* est stoppée
- le cas par défaut *default* est toujours valide

```
switch(valeur){
case val1 :
        // ...
case val2 :
        // ...
        break;
default :
        // ...
}
```

```
switch (var%5){ // Test sur le modulo 5 de var
case 0: // Si var%5 == 0
        var /= 5;
        var ++;
        break;
case 1: // Si var%5 == 1
        var *= 10;
        break;
case 3: // Si var%5 == 3
        var ++;
        break;
case 2 : // Autres
case 4 :
        var = 0;
        break;
}
```

Instruction de boucle - while

```
while(expression_conditionnelle){
        // ...
}
```

- expression_conditionnelle : évaluée au début de chaque itération
  - si vrai, on réalise l'itération
  - si faux, on sort de la boucle
- Incrément à effectuer dans la boucle (sinon boucle infinie
- accolades optionnelles si une seule instruction (dangereux)
- à partir de C99 (?), il est possible de déclarer des variables dans la boucle

```
1 printf("Entrez un nombre entre 1 et 5"\n);
2 scanf("%d",&a);
3 while(a  < 1 || a > 5){
4         printf("On a dit entre 1 et 5 !!!\n");
5         scanf("%d",&a);
6 }
```

Instruction de boucle - do . . .while

```
1 do{
2         // ...
3 }
4 while(expression_conditionnelle);
```

- expression_conditionnelle : évaluée à la fin de chaque itération
  - si vrai, on réalise l'itération suivante
  - si faux, on sort de la boucle
- Incrément à effectuer dans la boucle (sinon boucle infinie
- accolades optionnelles si une seule instruction (dangereux)
- à partir de C99 (?), il est possible de déclarer des variables dans la boucle

```c
int resultat, essai = 0;
resulat = rand()%5 + 1;  // Génération de nombre aléatoire entre 1 et 5
do{
        printf("Trouvez le nombre entre 1 et 5\n");
        scanf("%d", &essai);
} while (resulat != essai);
printf("Félicitations\n");
```

```c
int resultat, essai = 0;
resulat = rand()%5 + 1;  // Génération de nombre aléatoire entre 1 et 5
do{
        printf("Trouvez le nombre entre 1 et 5\n");
        scanf("%d", &essai);
        while(essai  < 1 || essai > 5){
                printf("On a dit entre 1 et 5 !!!\n");
                scanf("%d",&essai);
        }
} while (resulat != essai);
printf("Félicitations\n");
```

Instruction de boucle - for

```
for(initialisation; expression_conditionnelle ; increment){
        // ...
}
```

- initialisation : instruction exécutée avant le début de la boucle
- expression_conditionnelle : évaluée au début de chaque itération
  - si vrai, on réalise l'itération
  - si faux, on sort de la boucle
- increment : instruction éxécutée à la fin de chaque itération
- accolades optionnelles si une seule instruction (dangereux)
- à partir de C99 (?), il est possible de déclarer des variables dans initialisation

```c
int i, j = 0; // Variables de boucle
for (i = 0; i < 10; i++) // Boucle sur 10 tours
        printf("i=%d\n", i);
for (i = 10 ; i > 0; i--) // Boucle sur 10 tours descendante
        printf("i=%d\n", i);
for (i = 0; i <= 10; i+=2) // Boucle sur 10 tours, incrément de 2.
        printf("i=%d\n", i);
for (i = 0 , j = 0; i <= 5; i++, j+=2) // Boucle sur 5 tours, 2 incréments.
        printf("i=%d, j=%d\n", i, j);
```

Contôle de boucle

(a.k.a *Paresse algorithmique*)

- `break` : la boucle est immédiatement arrêtée
- `continue` : l'itération courante est immédiatement arrêtée et on passe à l'itération suivante
- `goto` : on sait que ça existe, mais on ne l'utilise pas ...