Chapitre 2
Les contrôles des blocs d'instructions

1. Blocs d'instructions et conditions

1.1 Qu'est-ce qu'un bloc d'instructions ?

1.1.1 Définition

- Un bloc d'instructions est UNE instruction composée de plusieurs instructions qui se suivent.
- En C (et tous les langages dérivés) il est délimité avec les opérateurs { } (accolades ouvrante et fermante).
- Un bloc peut contenir d'autres blocs imbriqués.
- Dans un fichier source, il ne peut pas y avoir d'instruction en dehors d'un bloc (sauf directives macro-processeur et déclarations de variables globales ou de fonctions). Pour être valides, toutes les instructions doivent être localisées dans un bloc. Le bloc au sommet est celui de la fonction main() qui réunit in fine toutes les instructions du programme.

1.1.2 Exemple

```
#include <stdio.h>
#include <stdlib.h>

int main()
{  //------------------------- ouv bloc main B1
int x,pasx;    // déclarations de variables locales au bloc main
```

```
                // elles sont visibles (accessibles) dans ce bloc
                // et tous les sous-blocs

    {  //----------ouv B2
       int c;
       x=0;
       c=rand()%256;
       pasx=rand()%5;
    }  // ---------ferm B2
    x=640;

    {  // ---------ouv B3
       //c=10; // provoque erreur, c non visible dans ce bloc
       x/=2;
       pasx=15;

    }  // ---------ferm B3
    x+=pasx;
    printf("x vaut : %d\n",x);  // affichage ?

    Return 0;

}  //-------------------------ferm bloc main
```

Les blocs posent la question de la visibilité des variables. En fait la durée de vie des variables en générale est associée au bloc dans lequel elles sont déclarées. C'est pourquoi elles sont visibles dans tous les sous-blocs imbriqués et invisibles en revanche dans les blocs de niveau supérieur ou de même niveau mais séparés.

1.1.3 Utilité d'un bloc d'instructions

Dans l'exemple ci-dessus les blocs sont inutiles : ils n'influent en rien sur le déroulement linéaire du programme, il n'y a aucune modification de la succession des opérations et les supprimer revient au même sauf pour la déclaration de la variable c.

L'utilité des blocs est de permettre de rompre avec cette linéarité et d'introduire grâce à des instructions données par le langage :

- Des sauts de bloc, ce sont les trois instructions if, if-else, if-else if-else.
- Des branchements, c'est l'instruction switch.
- Et des répétitions, ce sont les trois instructions de boucles while, do-while et for.

Les blocs permettent également de généraliser des portions de code qui se répètent dans un programme avec éventuellement des valeurs différentes. C'est ce que nous appelons une fonction. Une fonction est un bloc d'instructions doté d'une entrée de valeur et d'une sortie de valeur.

© Editions ENI - All rights reserved

1.2 Définir une condition

1.2.1 Pourquoi une condition ?

Les sauts conditionnels permettent d'exécuter ou de ne pas exécuter un bloc d'instructions selon qu'une condition est remplie ou non. De même pour les boucles, un bloc d'instructions est répété tant qu'une condition fixée pour sa répétition reste vraie.

Par exemple pour l'instruction if, première forme du saut conditionnel, l'exécution du bloc est soumise à une condition de la façon suivante :

```
Si (condition vraie) Alors
    faire bloc d'instructions
FinSi
```

Si et seulement si la condition est remplie, c'est-à-dire est "vraie", alors les instructions du bloc qui en dépend sont exécutées.

Remarque

Une condition vraie, c'est une expression dont la valeur est différente de 0. À l'inverse une condition est considérée comme fausse si l'expression vaut 0.

1.2.2 Comment définir une condition ?

Définir une condition, c'est écrire une expression qui sera évaluée comme vraie ou fausse. La plupart des conditions sont élaborées à partir de comparaisons de variables. Les comparaisons bits à bits sont parfois utilisées également.

1.2.3 Les opérateurs de comparaison

Il est possible de comparer entre elles des variables ou n'importe quelles expressions du point de vue de leurs valeurs. Soient deux expressions a et b (des variables ou des opérations), elles ont chacune une valeur et les comparaisons suivantes sont possibles :

```
a  >   b    a strictement supérieur à b
a  <   b    a strictement inférieur à b
a  >=  b    a supérieur ou égal à b
a  <=  b    a inférieur ou égal à b
a = =  b    a égal b (test d'égalité)
a  !=  b    a différent de b (test d'inégalité)
```

Le résultat de chaque expression vaut :

```
0 si c'est faux
1 si c'est vrai
```

À votre avis, qu'imprime le programme suivant ?

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int a=10, b=5;
    printf("%d ",a < b);
    printf("%d ",a+b > 12);

    a = 5;
    printf("%d ",a >= b);
    printf("%d ",a == b);

    b = 4;
    printf("%d ",a <= b);
    printf("%d ",a == b);
    printf("%d ",a+3 != b*2);
    return 0;
}
```

Réponse :

```
0 1 1 1 0 0 0
```

Autre exemple :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int a,b;
    printf("%d ",56 < rand() );   //  1 si vrai, 0 si faux

    printf("%d ", (a=rand()) + (b=rand()) >= 1000 );
                                 // 1 si vrai, 0 si faux
    printf("%d ",a*b == 45);     // 1 si oui, 0 si non
        return 0;
}
```

Les résultats dépendent des retours de la fonction rand().

1.2.4 L'opérateur unaire NON : !

L'opérateur ! placé à gauche d'une expression donne vrai si l'expression vaut 0 (faux) et faux si l'expression est différente de 0 (vraie). Par exemple :

```
#include <stdio.h>
#include <stdlib.h>
```

© Editions ENI - All rights reserved

```
int main()
{
int a=10, b=20;
    printf("%d ", !(a < b)); // imprime 0
    printf("%d ", !(a > b)); // imprime 1
    return 0;
}
```

Il est souvent utilisé dans des conditions à la place d'une égalité à 0, par exemple :

```
int a=10;

     printf("%d ", a==0);// imprime 0

     // et peut être remplacé par l'expression :

     printf("%d ", !a);   // qui imprime également 0
```

1.2.5 Priorités des opérateurs NON et comparaison

Toutes les opérations arithmétiques ainsi que décalages, conversion de type, sizeof, NON et complément à 1 sont prioritaires par rapport aux comparaisons. En revanche les comparaisons sont prioritaires sur ET, OU inclusif et exclusif et affectations combinées (Annexe Priorité et associativité des opérateurs).

Par exemple :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int a=10, b=0, c=2;
    printf("a+b < c*10 vaut %d\n", a+b < c*10);
    printf("c*10 >= 66 vaut %d\n", c*10 >= 66);
    printf("a+b < c*10 >= 66 == 80 vaut %d\n",
           a+b < c*10 >= 66 == 80);
    return 0;
}
```

```
La première ligne donne 1
la seconde  0
la troisième 0
```

1.3 Mise en pratique : opérateurs de comparaison et NON

Exercice 1

Faire un programme qui affiche les résultats de ce qui suit :

```
int a, b, c;
    srand(5);
    Imprimez les résultats de :
    a=rand()%256;
    b=rand()%256;
    c=rand()%256;

    a<128
    b>128
    c==223
    a < b >= rand()%2 == 1
    a= b !=c +rand()%50;
    b= a==c
    c= rand()%10 < rand()%10 >= rand()%10 != rand()%10 ==rand()%10
```

Exercice 2

Qu'imprime le programme suivant ?

```
#include <stdio.h>

int main()
{
int a=10, b=0, c=2;
    printf("a+b < c*10 vaut %d\n", a+b < c*10);
    printf("c*10 >= 15 vaut %d\n", c*10 >= 15);
    printf("a+b < c*10 >= 15 == 1 vaut %d\n",
           a+b < c*10 >= 15 == 1);
    printf("a+b < c*10 <= 15 == 1 vaut %d\n",
           a+b < c*10 <= 15 == 1);
    return 0;
}
```

2. Sauts conditionnels

2.1 L'instruction if

SI et seulement si l'expression est vraie ALORS le bloc des instructions associées au if est exécuté. Il ne peut y avoir qu'un seul bloc associé au if.

```
if ( expression1 vraie){
```

© Editions ENI - All rights reserved