Chapitre 3 : Structure de contrôle et répétitive

Les structures de contrôle définissent la suite dans laquelle les instructions sont effectuées. Dans ce chapitre, nous allons voir comment les instructions de sélection connues fonctionnent en C et nous allons faire connaissance d'un couple d'opérateurs spécial qui nous permet de choisir entre deux valeurs à l'intérieur d'une expression.
Constatons déjà que la particularité la plus importante des instructions de contrôle en C est le fait que les 'conditions' en C peuvent être des *expressions quelconques qui fournissent un résultat numérique.* La valeur zéro correspond à la valeur logique faux et toute valeur différente de zéro est considérée comme vrai.

I) if – else :

La structure alternative en langage algorithmique :

```
si (<expression logique>)
 alors
   <bloc d'instructions 1>
 sinon
   <bloc d'instructions 2>
fsi
```

* *Si l'<expression logique> a la valeur logique* vrai, *alors le <bloc d'instructions 1> est exécuté*
* *Si l'<expression logique> a la valeur logique* faux, *alors le <bloc d'instructions 2> est exécuté*

La structure alternative en C :

```
if ( <expression> )
    <bloc d'instructions 1>
  else
    <bloc d'instructions 2>
```

* *Si l'<expression> fournit une valeur différente de zéro, alors le <bloc d'instructions 1> est exécuté*
* *Si l'<expression> fournit la valeur zéro, alors le <bloc d'instructions 2> est exécuté*

La partie **<expression>** peut désigner :

une variable d'un type numérique, une expression fournissant un résultat numérique.

La partie **<bloc d'instructions>** peut désigner :

un (vrai) bloc d'instructions compris entre accolades, une seule instruction terminée par un point virgule.

Exemple 1 :

```
if (a > b)
  max = a;
      else
        max = b;
```

Exemple 2 :

```
if (EGAL)
  printf("A est égal à B\n");
else
  printf("A est différent de B\n");
```

Exemple 3 :

```
if (A-B)
     printf("A est différent de
     B\n");
else
     printf("A est égal à B\n");
```

Exemple 4 :

```
if (A > B)
  {
    AIDE = A;
    A=C;
    C = AIDE;
  }
else
  {
    AIDE = B;
    B=C;
    C = AIDE;
  }
```

II) if sans else :

La partie **else** est facultative. On peut donc utiliser **if** de la façon suivante :

if sans else :

```
if ( <expression> )
    <bloc d'instructions>
```

Attention ! :

Comme la partie **else** est optionnelle, les expressions contenant plusieurs structures **if** et **if - else** peuvent mener à des confusions.

Exemple :

L'expression suivante peut être interprétée de deux façons :

```
if (N>0)
   if (A>B)
      MAX=A;
   else
      MAX=B;
if (N>0)
   if (A>B)
      MAX=A;
else
   MAX=B;
```

Pour N=0, A=1 et B=2,

* dans la première interprétation, MAX reste inchangé,

* dans la deuxième interprétation, MAX obtiendrait la valeur de B.

Sans règle supplémentaire, le résultat de cette expression serait donc imprévisible.

Convention :

En C une partie **else** est toujours liée au dernier **if** qui ne possède pas de partie **else**. Dans notre exemple, C utiliserait donc la première interprétation.

Solution :

Pour éviter des confusions et pour forcer une certaine interprétation d'une expression, il est recommandé d'utiliser des accolades **{ }** .

Exemple :

Pour forcer la deuxième interprétation de l'expression ci-dessus, nous pouvons écrire :

```
        if (N>0)
          {
    if (A>B)
      MAX=A;
          }
        else
            MAX=B;
```

I) if - else if - ... – else :

En combinant plusieurs structures **if** - **else** en une expression nous obtenons une structure qui est très courante pour prendre des décisions entre plusieurs alternatives :

if - else - ... - else :

```
        if ( <expr1> )
            <bloc1>
          else if (<expr2>)
            <bloc2>

           else if (<expr3>)
              <bloc3>
           else if (<exprN>)
              <blocN>
           else <blocN+1>
```

Les expressions <expr1> ... <exprN> sont évaluées du haut vers le bas jusqu'à ce que l'une d'elles soit différente de zéro. Le bloc d'instructions qui lui est lié est alors exécuté et le traitement de la commande est terminé.

Exemple :

```
        #include <stdio.h>
        main()
        {
         int A,B;
```

```
        printf("Entrez deux nombres entiers :");
        scanf("%i %i", &A, &B);
        if (A > B)
         printf("%i est plus grand que %i\n", A, B);
        else if (A < B)
         printf("%i est plus petit que %i\n", A, B);
        else
            printf("%i est égal à %i\n", A, B);
        return (0);
       }
```

La dernière partie **else** traite le cas où aucune des conditions n'a été remplie. Elle est optionnelle, mais elle peut être utilisée très confortablement pour détecter des erreurs.

Exemple :

```
        ...
        printf("Continuer (O)ui / (N)on ?");
        getchar(C);
   if (C=='O')
        {
             ...
        }
        else if (C=='N')
             printf("Au revoir ...\n");
        else
             printf("\aErreur d'entrée !\n");
        ...
```

IV) Les opérateurs conditionnels :

Le langage C possède une paire d'opérateurs un peu exotiques qui peut être utilisée comme alternative à **if - else** et qui a l'avantage de pouvoir être intégrée dans une expression :

Les opérateurs conditionnels :

```
<expr1> ? <expr2> : <expr3>
```

* Si <expr1> fournit une valeur différente de zéro, alors la valeur de <expr2> est fournie comme résultat,
* Si <expr1> fournit la valeur zéro, alors la valeur de <expr3> est fournie comme résultat.

Exemple :

La suite d'instructions

```
if (A>B)
   MAX=A;
else
   MAX=B;
```

peut être remplacée par :

```
MAX=(A>B)?A:B;
```

Employés de façon irréfléchis, les opérateurs conditionnels peuvent nuire à la lisibilité d'un programme, mais si on les utilise avec précaution, ils fournissent des solutions très élégantes :

Exemple :

```
printf("Vous avez %i carte%c \n", N, (N==1) ? ' ' : 's');
```

Les ***règles de conversion de types*** s'appliquent aussi aux opérateurs conditionnels **? :** Ainsi, pour un entier N du type **int** et un rationnel F du type **float**, l'expression :

```
(N>0)?N:F
```

va ***toujours*** fournir un résultat du type **float**, que N soit plus grand ou plus petit que zéro !

V) Structure répétitive:

En C, nous disposons de trois structures qui nous permettent la définition de boucles conditionnelles :

1) la structure : **while**
2) la structure : **do - while**
3) la structure : **for**

Théoriquement, ces structures sont interchangeables, c'est-à-dire il serait possible de programmer toutes sortes de boucles conditionnelles en n'utilisant qu'une seule des trois structures.

VI) while :

La structure **while** correspond tout à fait à la structure tant que du langage algorithmique. (Si on néglige le fait qu'en C les conditions peuvent être formulées à l'aide d'expressions numériques.)

La structure tant que en langage algorithmique :

```
tant que (<expression
      logique>) faire <bloc
      d'instructions>
ftant
```

* Tant que l'<expression logique> fournit la valeur vrai, le <bloc d'instructions> est exécuté.
* Si l'<expression logique> fournit la valeur faux, l'exécution continue avec l'instruction qui suit ftant.
* Le <bloc d'instructions> est exécuté zéro ou plusieurs fois.

La structure while en C :

```
while ( <expression> )
      <bloc d'instructions>
```

* Tant que l'<expression> fournit une valeur différente de zéro, le <bloc d'instructions> est exécuté.
* Si l'<expression> fournit la valeur zéro, l'exécution continue avec l'instruction qui suit le bloc d'instructions.
* Le <bloc d'instructions> est exécuté zéro ou plusieurs fois.

La partie <expression> peut désigner : une variable d'un type numérique, une expression fournissant un résultat numérique.

La partie <bloc d'instructions> peut désigner : un (vrai) bloc d'instructions compris entre accolades, une seule instruction terminée par un point-virgule.

Exemple 1 :

```
/* Afficher les nombres de 0 à 9 */
int I = 0;
while (I<10)
  {
    printf("%i \n", I);
    I++;
  }
```

Exemple 2 :

```
int I;
/* Afficher les nombres de 0 à 9 */
I=0;
while (I<10)
  printf("%i \n", I++);
/* Afficher les nombres de 1 à 10 */
I=0;
while (I<10)
  printf("%i \n", ++I);
```

Exemple 3 :

```
/* Afficher les nombres de 10 à 1 */
int I=10;
while (I)
  printf("%i \n", I--);
     ;
```

VII) do – while :

La structure **do - while** est semblable à la structure **while**, avec la différence suivante :

* **while** évalue la condition ***avant*** d'exécuter le bloc d'instructions,
* **do - while** évalue la condition ***après*** avoir exécuté le bloc d'instructions. Ainsi le bloc d'instructions est exécuté au moins une fois.

La structure do - while en C :

```
do
      <bloc d'instructions>
while ( <expression> );
```

Le <bloc d'instructions> est exécuté au moins une fois et aussi longtemps que l'<expression> fournit une valeur différente de zéro.

Exemple 1 :

```
float N;
do
  {
    printf("Introduisez un nombre entre 1 et 10 :");
    scanf("%f", &N);
  }
while (N<1 || N>10);
```

Exemple 2 :

```
int n, div;
printf("Entrez le nombre à diviser : ");
scanf("%i", &n);
do
   {
    printf("Entrez le diviseur ( 0) : ");
    scanf("%i", &div);
  }
while (!div);
printf("%i / %i = %f\n", n, div, (float)n/div);
```

Exemple 3 :

Le programme de calcul de la racine carrée :

```
programme RACINE_CARREE
   réel N
   répéter
       écrire "Entrer un nombre (>=0) : "
       lire N
   jusqu'à (N >= 0)
   écrire "La racine carrée de ",N ,"vaut ", N
fprogramme (* fin RACINE_CARRE *)
```

se traduit en C par :

```
#include <stdio.h>
#include <math.h>
main()
{
float N;
do
  {
    printf("Entrer un nombre (>= 0) : ");
    scanf("%f", &N)
  }
while (N < 0);
printf("La racine carrée de %.2f est %.2f\n", N, sqrt(N));
return (0);
}
```

VIII) for :

La structure **for** est utilisée pour faciliter la programmation de boucles de comptage.

La structure for en C :

```
for ( <expr1> ; <expr2> ; <expr3> )
    <bloc d'instructions>
```

est équivalent à :

```
<expr1>;
while ( <expr2> )
  {
    <bloc d'instructions>
    <expr3>;
  }
```

<**expr1**> est évaluée une fois avant le passage de la boucle. Elle est utilisée pour initialiser les données de la boucle.

<**expr2**> est évaluée avant chaque passage de la boucle. Elle est utilisée pour décider si la boucle est répétée ou non.

<**expr3**> est évaluée à la fin de chaque passage de la boucle. Elle est utilisée pour réinitialiser les données de la boucle.

Le plus souvent, **for** est utilisé comme boucle de comptage :

```
for ( <init.> ; <cond. répétition> ; <compteur> )
    <bloc d'instructions>
```

Exemple :

```
int I;
for (I=0 ; I<=20 ; I++)
    printf("Le carré de %d est %d \n", I, I*I);
```

En pratique, les parties <expr1> et <expr2> contiennent souvent plusieurs initialisations ou réinitialisations, *séparées par des virgules*.

Exemple :

```
int n, temp;
for (temp=0, n=1 ; n<101 ; n++)
    temp +=n;
printf("La somme des nombres de 1 à 100 est %d\n", temp);
```

Exemple :

Cet exemple nous présente différentes variations pour réaliser le même traitement et nous montre la puissance de la structure **for**. Les expressions suivantes lisent un caractère au clavier et affichent son code numérique en notation binaire :

```
/* a */
/* notation utilisant la structure while */
int C, I;
C=getchar();
I=128;
while (I>=1)
```

```
  {
    printf("%i ", C/I);
    C%=I;
    I/=2;
  }

/* b */
/* notation utilisant for  - très lisible - */
int C, I;
C=getchar();
for (I=128 ; I>=1 ; I/=2)
  {
    printf("%i ", C/I);
    C%=I;
  }

/* c */
/* notation utilisant for  - plus compacte - */

int C, I;
C=getchar();

for (I=128 ; I>=1 ; C%=I, I/=2)
    printf("%i ", C/I);

/* d */
/*   notation   utilisant   for   -   à
déconseiller - */ /* surcharge et mauvais
emploi de la structure */

int C, I;
for(C=getchar(),I=128; I>=1 ;printf("%i ",C/I),C%=i,i/=2);
```