32 UNITE D'ENSEIGNEMENT 32 : STRUCTURE DE CONTROLE EN C

OBJECTIFS PEDAGOGIQUES :

✓Ecrire les structures de contrôle en langage C

CONTROLE DE PREREQUIS :

- ✓ Définir **programme informatique**
- ✓ Définir **programme source**

SITUATION DE VIE:

SIMON BOUBA veut écrire un programme informatique. Il rencontre quelques difficultés pour décrire l'enchaînement des instructions à utiliser. En plus il ne parvient pas à faire des traitements séquentiels, conditionnels ou répétitifs en langage de programmation C**.** On vous fait appel pour l'aider**.**

CONSIGNE:

1. Identifier les notions non comprises par **SIMON BOUBA**. (**Réponse attendue** : Il rencontre quelques difficultés pour décrire l'enchaînement des instructions à utiliser, aussi il ne parvient pas à faire des traitements séquentiels, conditionnels ou répétitifs).
2. Indiquer comment on appelle cette description de l'enchaînement des instructions en langage de programmation C. (**Réponse attendue** : Structures ou instructions de contrôle)
3. Enumérer un logiciel à utiliser pour saisir le programme à exécuter sur un ordinateur. (**Réponse attendue :** NotePad++)
4. Enumérer quelques logiciels qu'il peut utiliser pour compiler et exécuter son code. (**Réponse attendue**: CodeBlocks, Visual C++, Xcode, GCC)

RESUME

Les structures de contrôle décrivent l'enchaînement des instructions. Elles permettent de faire des traitements séquentiels, conditionnels ou répétitifs (itératifs). Il en existe plusieurs selon l'enchaînement qu'on veut donner à nos instructions.

L'INSTRUCTION *IF*

1. Blocs d'instructions

Un bloc est une suite d'instructions placées entre { et }. Les instructions figurant dans un bloc sont absolument quelconques. Il peut s'agir aussi bien d'instructions simples (terminées par un point-virgule) que d'instructions structurées (choix, boucles) lesquelles peuvent alors à leur tour renfermer d'autres blocs. N'oubliez pas que toute instruction simple est toujours terminée par un point-virgule.

2. Syntaxe de l'instruction if

Le mot ***else*** et l'instruction qu'il introduit sont facultatifs, de sorte que cette instruction ***if*** présente deux formes.

Première forme :

```
if (expression)
instruction_1
else
instruction_2
```

Deuxième forme :

```
if (expression)
 instruction_1
```

3. Exemples

```
i = i + 1 ;
if ( i < limite )
printf ("OK") ;
```

4. En cas d'imbrication des instructions *if*

La règle adoptée par le langage C pour lever l'ambiguïté est la suivante : Un **else** se rapporte toujours au dernier *if* rencontré auquel un **else** n'a pas encore été attribué**.**

```
if (a<=b)  if (b<=c) printf ("ordonné") ;
else printf ("non ordonné") ;
```

On affiche : **non ordonné** au cas où a est supérieur à b.

L'INSTRUCTION SWITCH

1. Exemples d'introduction de l'instruction Switch

Premier exemple

Voyez ce premier exemple de programme accompagné de trois exemples d'exécution.

```
main()
{
   int n ;
   printf ("donnez un entier : ") ;
   scanf ("%d", &n) ;

   switch (n)
   {  case 0 : printf ("nul\n") ;
               break ;

      case 1 : printf ("un\n") ;
               break ;
      case 2 : printf ("deux\n") ;
               break ;
   }

   printf ("au revoir\n") ;
}
```

```
donnez un entier : 0
nul
au revoir

_________

donnez un entier : 2
deux

 au revoir

_________

donnez un entier : 5
au revoir
```

L'instruction *Switch* s'étend ici sur huit lignes (elle commence au mot *Switch*). Son exécution se déroule comme suit. On commence tout d'abord par évaluer l'expression figurant après le mot *Switch* (ici n). Ensuite de quoi, on recherche dans le *bloc* qui suit s'il existe une "*étiquette*" de la forme "*case x*" correspondant à la valeur ainsi obtenue. Si c'est le cas, on "se branche" à l'instruction figurant après cette étiquette. Dans le cas contraire, on passe à l'instruction qui suit le bloc. Par exemple, quand n vaut 0, on trouve effectivement une étiquette "case 0" et l'on exécute l'instruction correspondante, c'est-à-dire : printf ("nul") ;

On passe ensuite, naturellement, à l'instruction suivante, à savoir, ici : **break** ; Celle-ci demande en fait de sortir du bloc. Notez bien que le rôle de cette instruction est fondamental. Voyez, à titre d'exemple, ce que produirait ce même programme en l'absence d'instructions ***break*** :

***Quand on oublie les* breaks on obtient :**

```
main()
{
   int n ;
   printf ("donnez un entier : ") ;
   scanf ("%d", &n) ;

   switch (n)
   {  case 0 : printf ("nul\n") ;
      case 1 : printf ("un\n") ;
      case 2 : printf ("deux\n") ;
   }

   printf ("au revoir\n") ;
}
```

```
donnez un entier : 0
nul
un
deux
au revoir

_______

donnez un entier : 2
deux
au revoir
```

L'étiquette "default"

Il est possible d'utiliser le mot clé "**default**" comme étiquette à laquelle le programme se "branchera" dans le cas où aucune valeur satisfaisante n'aura été rencontrée auparavant. En voici un exemple :

```
{
   int n ;
   printf ("donnez un entier : ") ;
   scanf ("%d", &n) ;

   switch (n)
   {  case 0 :  printf ("nul\n") ;
                break ;
      case 1 :  printf ("un\n") ;
                break ;
      case 2 :  printf ("deux\n") ;
                break ;
      default : printf ("grand\n") ;
   }
   printf ("au revoir\n") ;
}
```

```
donnez un entier : 2
deux
au revoir

__________

donnez un entier : 25
grand
au revoir
```

Un exemple plus général

D'une manière générale, on peut trouver :

➔ plusieurs instructions à la suite d'une étiquette,

➔ des étiquettes sans instructions, c'est-à-dire, en définitive, plusieurs étiquettes successives (accompagnées de leurs deux points).

Voyez cet exemple, dans lequel nous avons volontairement omis certain **break**.

```
main()
{
   int n ;                                          donnez un entier : 1
   printf ("donnez un entier : ") ;                 petit
   scanf ("%d", &n) ;                               moyen


  switch (n)                                              ___________
   {  case 0  : printf ("nul\n") ;
              break ;                                   donnez un entier : 4
      case 1  :                                         moyen
      case 2  : printf ("petit\n") ;
      case 3  :                                               ___________
      case 4  :
      case 5  : printf ("moyen\n") ;                    donnez un entier : 25
              break ;                                   grand
      default : printf ("grand\n") ;
   }
}
```

2. **Syntaxe de l'instruction *Switch***

```
switch (expression)
    { case constante_1 : [ suite_d'instructions_1 ]

      case constante_2 : [ suite_d'instructions_2 ]
                  ..............
      case constante_n : [ suite_d'instructions_n ]
      [ default        :   suite_d'instructions   ]
    }
```

3. **L'INSTRUCTION DO ... WHILE**

Abordons maintenant la première façon de réaliser une boucle en C, à savoir l'instruction ***do... while*.**

L'instruction: **do{.....}while(n<=0);** répète l'instruction qu'elle "contient" (ici un bloc) tant que la condition mentionnée (n<=0) est vraie (c'est-à-dire, en C, non nulle). Autrement dit, ici, elle demande un nombre à l'utilisateur (en affichant la valeur lue) tant qu'il ne fournit pas une valeur positive. On ne sait pas a priori combien de fois une telle boucle sera répétée. Toutefois, de par sa nature même, elle est toujours parcourue au moins une fois. En effet, la condition qui régit cette boucle n'est examinée qu'à la fin de chaque répétition (comme le suggère d'ailleurs le fait que la "partie while" figure en fin). Notez bien que la "sortie de boucle" ne se fait qu'après un parcours complet de ses instructions et non dès que la condition mentionnée devient

fausse. Ainsi, ici, même après que l'utilisateur a fourni une réponse convenable, il y a exécution de l'instruction d'affichage : printf ("vous avez fourni %d", n) ;

Exemple

```
main()
{
   int n ;                                          donnez un nb >0 : -3
   do                                               vous avez fourni -3
      { printf ("donnez un nb >0 : ") ;             donnez un nb >0 : -9
        scanf ("%d", &n) ;                          vous avez fourni -9
        printf ("vous avez fourni %d\n", n) ;       donnez un nb >0 : 12
      }                                             vous avez fourni 12
   while (n<=0) ;                                   réponse correcte
   printf ("réponse correcte") ;
}
```

Syntaxe de l'instruction do... while

```
do      instruction

        while  (expression) ;
```

4. L'instruction *WHILE*

Voyons maintenant la deuxième façon de réaliser une boucle conditionnelle, à savoir l'instruction ***While***.

La construction : **while (som<100)** répète l'instruction qui suit (ici un bloc) tant que la condition mentionnée est vraie (différente de zéro), comme le ferait ***Do... While***. Par contre, cette fois, la condition de poursuite est examinée **avant** chaque parcours de la boucle et non après. Ainsi, contrairement à ce qui se passait avec ***Do... While***, une telle boucle peut très bien n'être parcourue aucune fois si la condition est fausse dès qu'on l'aborde (ce qui n'est pas le cas ici).

Exemple

```
main()
{
   int n, som ;                                     donnez un nombre : 15
   som = 0 ;                                        donnez un nombre : 25
   while (som<100)                                  donnez un nombre : 12
      { printf ("donnez un nombre : ") ;            donnez un nombre : 60
        scanf ("%d", &n) ;                          somme obtenue : 112
        som += n ;
      }
   printf ("somme obtenue : %d", som) ;
}
```

Syntaxe de l'instruction While

```
while   (expression)
        instruction
```

5. L'INSTRUCTION *FOR*

Etudions maintenant la dernière instruction permettant de réaliser des boucles, à savoir l'instruction ***For***.

Exemple d'introduction de l'instruction For

Considérez ce programme :

```
main()
{
   int i ;
   for ( i=1 ; i<=5 ; i++ )
      {  printf ("bonjour ") ;
         printf ("%d fois\n", i) ;
      }
}
```

```
bonjour 1 fois
bonjour 2 fois
bonjour 3 fois
bonjour 4 fois
bonjour 5 fois
```

La ligne : for (i=1 ; i<=5 ; i++) comporte en fait trois expressions. La première est évaluée (une seule fois) avant d'entrer dans la boucle. La deuxième conditionne la poursuite de la boucle. Elle est évaluée **avant** chaque parcours. La troisième, enfin, est évaluée à la fin de chaque parcours. Le programme précédent est équivalent au suivant :

Pour remplacer une boucle for par une boucle While :

```
main()
{
   int i ;
   i = 1 ;
   while (i<=5)
      {  printf ("bonjour ") ;
         printf ("%d fois\n", i) ;
         i++ ;
      }
}
```

```
bonjour 1 fois
bonjour 2 fois
bonjour 3 fois
bonjour 4 fois
bonjour 5 fois
```

Là encore, la généralité de la notion d'expression en C fait que ce qui était expression dans la première formulation (for) devient instruction dans la seconde (while).

Syntaxe de l'instruction for

```
for  ( [ expression_1 ] ; [ expression_2 ] ; [ expression_3 ] )
                instruction
```

SITUATION D'INTEGRATION :

Situation1

Soit le petit programme suivant :

```
#include <stdio.h>
main()
{
int i, n, som ;
som = 0 ;
for (i=0 ; i<4 ; i++)
{ printf ("donnez un entier ") ;
scanf ("%d", &n) ;
som += n ;
}
printf ("Somme : %d\n", som) ;
}
```

Ecrire un programme réalisant exactement la même chose, en employant, à la place de l'instruction *for* :

a) une instruction *while*,

b) une instruction *do... while*.

Situation2

Calculer la moyenne de notes fournies au clavier avec un "dialogue" de ce type:

Note 1 : 12

Note 2 : 15.25

Note 3 : 13.5

Note 4 : 8.75

Note 5 : -1

Moyenne de ces 4 notes : 12.37

Le nombre de notes n'est pas connu a priori et l'utilisateur peut en fournir autant qu'il le désire. Pour signaler qu'il a terminé, on convient qu'il fournira une note fictive négative. Celle-ci ne devra naturellement pas être prise en compte dans le calcul de la moyenne.

REINVESTISSEMENT :

1. Définir : **structure de contrôle**.
2. En utilisant la structure de contrôle **for** (en français pour), proposer un programme en C qui prend en entrée **n** notes (n veut dire qu'on généralise pour quelqu'un qui veut entrer 5 notes, 6 notes…) des matières numérotées de **1 à n** (utilisez un compteur **i** pour désigner ces matières), qui prend aussi en entrée les coefficients respectifs de ces matières puis calcule et affiche la moyenne de l'élève **NOUTTE Ghislain** du lycée de Bagangté.