Chapitre 3

Les structures de contrôle

Chapitre 3 : Les structures de contrôle

I L'instruction expression

Syntaxe : expression;

Interprétation : l'expression est évaluée et sa valeur est ignorée.

➢ Cette instruction n'a de sens que si l'expression réalise un effet de bord.

Exemple :

```
i +1 ;
```

- C'est une instruction valide, conforme à la syntaxe.
- Mais ne faisant aucun effet de bord, elle n'a aucune utilité.

Chapitre 3 : Les structures de contrôle

II L'instruction composée (ou bloc d'instructions)

Syntaxe :

```
{
[liste_de_déclarations]     /*optionnel*/
liste_d'instructions
}
```

Sémantique :

Son but est double car elle permet :

- de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction
- de déclarer des variables qui ne seront accessibles qu'à l'intérieur de l'instruction composée.

Chapitre 3 : Les structures de contrôle

Exemple :

```
{
 int a = 2, b;
 a+=b;
 ....
}
```

- Le bloc est défini entre { }
- Les deux variables a et b sont déclarées dans ce bloc.
- A l'extérieur du bloc (des deux accolades), elles n'existent plus.

Chapitre 3 : Les structures de contrôle

III Les instructions de branchement conditionnel

III.a La conditionnelle :

Syntaxe générale :

```
if (expression1)
    instructions 1
[else if (expression2)
    instructions 2]
[else if (expression3)
    instructions 3]
…
[else instructions]
```


Chapitre 3 : Les structures de contrôle

Remarque :

- Le nombre de **else if** peut être égal à 0
- Le **else** peut être optionnel
- instructions *i* peut être une instruction composée
- **expression doit être parenthésée**

Exemple 1 :

```
if (car == 'a')
{
  i++;
  nb--;
}
else nb++;
```

Chapitre 3 : Les structures de contrôle

Exemple 2 :

```
if (a > b)
   if (c< d) x = z;
   else i = j;
```

Dans cet exemple, le **else** est relatif au deuxième **if.**

Si l'on veut qu'il concerne le premier **if**, il faut écrire l'instruction comme suit :

```
if (a > b)
  {
    if (c< d) x = z;
  }
else i = j;
```


Chapitre 3 : Les structures de contrôle

III.b Le choix multiple :

Syntaxe générale :

```
switch (expression)
        {
         case   valeur_cste1 : liste_d'instructions1;
                               break;
         case   valeur_cste2 : liste_d'instructions2;
                               break;
         ....
         case   valeur_csteN : liste_d'instructionsN;
                               break;
        [ default : liste_d'instructions;
                    [break;]                    / * [  ] : entre crochets signifie optionnel */
        ]
        }
```

qui est un if généralisé.

Chapitre 3 : Les structures de contrôle

Sémantique :

✓ toute *valeur_cste* est connue à la compilation

✓ En fonction de la valeur de **expression**, on exécute l'une ou l'autre des listes d'instructions.

✓ Le **break** permet d'éviter que toutes les instructions ne soient exécutées en cascade.

✓ La clause **default** est optionnelle.

✓ 2 *valeur_cste* ne peuvent être identiques.

✓ ***expression doit être parenthésée***

Chapitre 3 : Les structures de contrôle

Exemple :

Soit le morceau de programme qui compte le nombre de caractères correspondant à un chiffre dans une suite de caractères lus :

```
switch (c)                          /* c est de type char    */
{
  case '0' :
  case '1' :
  case '2' :
  case '3' :
  case '4' : case '5' : case '6' : case '7' :
  case '8' : case '9' : nb_chiffres++;
                        break;
  default : nb_non_chiffres++;
}
```

Chapitre 3 : Les structures de contrôle

IV L'instruction d'itération

Syntaxe forme 1 :

```
while  (expression)                    /* signifie (expression ==1) */
          instructions;
```

Sémantique : tant que **expression** est vérifiée, c.à.d *non nulle*, instructions est exécutée. Cette dernière peut être une instruction composée (ou bloc d'instructions).

Exemple :

```
a =  10;
while (a > 0)
 {
      c+=a;                            /* nombre d'itérations : 10 */
      ....
      a--;
 }
```


Chapitre 3 : Les structures de contrôle

Syntaxe forme 2 :

```
          do
           instructions;
          while  (expression);          /* signifie (expression ==1) */
```

Sémantique : instructions est exécutée une fois. Et tant que **expression** est vérifiée, c.à.d *non nulle*, instructions sera exécutée. Instructions peut être une instruction composée (ou bloc d'instructions).

Exemple :

```
a =  10;
do
{
   c+=a;                        /* nombre d'itérations : 10 */
   ....
   a--;
 }
while (a > 0);
```

Chapitre 3 : Les structures de contrôle

Syntaxe forme 3 :

for (expression1; expression2; expression3)

instructions;

où :

- expression1 : correspond à une initialisation
- expression2 : correspond à l'expression de conditions d'arrêt
- expression3 : correspond à des instructions

Les trois expressions peuvent être composées de plusieurs expressions : elles seront séparées par des virgules.

Utilité : *faire plusieurs initialisations à la fois par exemple*.

Chapitre 3 : Les structures de contrôle

Exemple : Soit le calcul fact de la factorielle d'un nombre ***nb*** entier positif donné :

```
fact = 1;
for (i = 1; i <= nb; i++)
        fact *= i; /* fact = fact*i; */
```

Que l'on peut également exprimer comme suit :

```
for (i = 1, fact = 1; i <= nb; fact *= i, i++);
```

Il est fortement conseillé d'éviter ce type d'écriture qui rend un programme rapidement illisible et qui n'apporte rien de plus à la lisibilité du programme surtout si on ne maîtrise pas le langage.

Chapitre 3 : Les structures de contrôle

Remarques :

1) `for ( ; ; );` /* *boucle infinie* */

2) On peut écrire :

```
char c;
for (c = 'a'; c <= 'z'; c++)
{
        ...
}
```


Chapitre 3 : Les structures de contrôle

V Instruction de branchement non conditionnel

Forme 1 :

- En plus de son utilisation dans l'instruction de branchement conditionnel **switch** l'instruction **break** peut être utilisée, de manière générale, dans toute forme d'itération. Elle a pour effet de faire sortir de l'itération et de passer à l'exécution de l'instruction suivante.

- Dans le cas de plusieurs itérations (boucles) imbriquées, elle fait sortir de la boucle la plus interne.

Chapitre 3 : Les structures de contrôle

Exemple :

```
int main ()
{
        int i, a = 0;
        for (i = 4; i > 0; i--)        /* i=4, a=8; i=3, a=14; i=2 */
        {
                a = a + 2*i;
                if (i = = 2)  break;
        }
}

/* valeur de i : 2, valeur de a : 14 */
```


Chapitre 3 : Les structures de contrôle

Forme 2 : Dans une itération, l'instruction **continue** permet de passer à l'étape suivante sans exécuter la suite d'instructions de la boucle.

Exemple :

```
int main ()
{
  int i, a = 0;                  /* i=4, a=8; i=3, a=14; i=2,1, a=16;i=0, a=16 */
  for (i = 4; i > 0; i--)
  {
    if (i = = 2)  continue;
    a = a + 2*i;
   }
}
/* valeur de i : 0, valeur de a : 16 */
```

Chapitre 3 : Les structures de contrôle

Remarque :

Il existe une 3° forme d'instruction de branchement conditionnel : l'instruction **goto**. Elle permet de faire un saut en avant ou en arrière à une instruction identifiée par une **étiquette**. Mais un programme **C** digne de ce nom se passe d'une telle instruction => Utilisation à proscrire.

Chapitre 3 : Les structures de contrôle

VI Opérations (ou fonctions) d'entrée-sortie usuelles

Les **entrées/sorties** ne font pas partie du langage **C** proprement dit. Néanmoins, il existe une bibliothèque standard (standard Input/Output Library : **stdio**) de fonctions d'affichage et de saisie utilisées avec les unités classiques que sont le clavier et l'écran.

Cette librairie est intégrée dans un programme à l'aide de la directive **include** :

#include <stdio.h>

/* fichier qui contient les entêtes des opérations contenues dans la librairie ***/**

Chapitre 3 : Les structures de contrôle

a) La fonction d'écriture : ***printf***

Syntaxe :

printf (" chaîne de contrôle ", liste_arguments);

✓ liste d'arguments comporte la liste des valeurs/variables à afficher, séparées par des virgules

✓ chaîne de contrôle contient le texte à afficher et les **spécifications** de **format** (ou **descripteurs**) relatif à chaque élément de la liste.

✓ Un **spécificateur** commence par le caractère **%** suivi du (ou des) caractère(s) désignant le format d'impression. Sa forme :

% [-] [taille] [.précision] [l] caractère_de_conversion

Chapitre 3 : Les structures de contrôle

- le tiret (-) indique que l'on veut aligner à gauche (l'alignement à droite est l'alignement par défaut)

- si la taille est indiquée c'est la taille minimale du champ

- la précision (*précédée d'un point*) indique le nombre maximal de caractères affichés ou le nombre de chiffres après la virgule dans le cas d'un nombre réel

- pour un entier, on peut préciser qu'il est de type ***entier long*** avec la lettre **l**

- le caractère de conversion indique le type de la valeur à écrire. Les plus usuels :

Chapitre 3 : Les structures de contrôle

- **d** : entier en notation décimale
- **o** : entier en notation octale
- **x** : entier en notation hexadécimale
- **u** : entier non signé en notation décimale
- **e** : réel en notation exponentielle
- **f** : réel en notation avec virgule
- **g** : réel en notation e ou f (la plus courte des 2)

(combinables avec le format **l** (long))

- **c** : caractère
- **s** : chaîne de caractères

Chapitre 3 : Les structures de contrôle

Exemple : l'instruction

printf (**"**Nous sommes le **% -.**3**s** **%**2**d** % **-.**10**s**, **%**4**.**4**d** **.\n"**, nomjour, nojour, mois, annee);

affichera : Nous sommes le lun 14 octobre , 2024.

- le nom du jour est affiché sur 3 caractères maxi avec justification à gauche
- le n° du jour est affiché sur 2 positions (justifié à droite par défaut)
- le nom du mois est affiché sur 10 caractères maxi avec justification à gauche
- l'année est affichée sur 4 positions (au moins et au plus)

Chapitre 3 : Les structures de contrôle

Autres exemples **:**

❖printf(" %5.2f ", **nb**) ; ⇒ on réserve 5 caractères (incluant le **.**) pour afficher le nombre réel **nb** dont 2 pour les chiffres après la virgule.

❖printf(" %.10f ", **nb**) ; ⇒ le nombre réel **nb** sera affiché avec 10 chiffres après la virgule.

Dans le cas du type réel , si la précision n'est pas indiquée, elle est égale par défaut à 6 chiffres après la virgule.

❖printf(" %8.4s ", **ch**) ; ⇒ on réserve un champ de 8 caractères pour afficher la chaîne **ch** mais seuls les 4 premiers caractères sont affichés. Le reste est complété par des espaces.

❖printf(" %5d ", **ent**); ⇒ on réserve 5 caractères pour afficher l'entier **ent** avec justification à droite (par défaut).

❖printf(" %-5d ", **ent**); ⇒ on réserve 5 caractères pour afficher l'entier **ent** avec justification à gauche.

Chapitre 3 : Les structures de contrôle

b) La fonction de saisie : ***scanf***

Syntaxe **:**

scanf ("chaîne de contrôle**",** liste_arguments**);**

✓ liste d'arguments comporte la liste des variables de saisie, séparées par des virgules.

✓ argument = **&***nom_variable : adresse de la variable + variable de saisie*

✓ chaîne de contrôle : elle indique le format dans lequel les données sont converties, à l'exclusion de tout autre caractère (pas de \n par exemple).

✓ Les données à saisir sont séparées par des espaces ou <CR> (touche entrée), sauf dans le cas de la saisie de caractères.

✓ On peut fixer le nombre de caractères à saisir à l'aide d'un format : %4s pour une chaîne de 4 caractères, %4d pour un entier qui peut comporter 4 chiffres, signe inclus.

Chapitre 3 : Les structures de contrôle

Exemple :

```
int main ()
{
   int nb;
   printf ("Saisir un entier sous forme hexadecimale = ");
   scanf (" %x", &nb);
   printf ("valeur en décimale : %d, valeur en octale : %o \n", nb, nb);
}
```

Valeur saisie = **2b** => *Valeurs affichées* : **43** (en *décimal*), **53** (en *octal*).

Remarque : Avec certains compilateurs, il n'est pas nécessaire d'inclure stdio.h pour utiliser **printf** et **scanf**.

Chapitre 3 : Les structures de contrôle

c) Lecture et affichage de caractères

Le type caractère possède des fonctions spécifiques : **getchar** et **putchar**.

✓ Ce sont des fonctions d'entrée/sortie non formatées.

✓ On peut stocker le caractère lu dans une variable de type char.

✓ Lecture d'un caractère avec stockage : car = **getchar()**;

- C'est une fonction sans paramètre qui " capte " la valeur du caractère lu à la position courante de l'entrée standard (le clavier) et le retourne (elle renvoie un ***int*** *correspondant au caractère lu)*. Quand un programme exécute cette fonction, le curseur dans la fenêtre d'exécution est « en attente », l'utilisateur doit alors saisir au moins un caractère au clavier.

Chapitre 3 : Les structures de contrôle

✓ Affichage d'un caractère : **putchar(car)**;

- Elle affiche le caractère **car** sur la sortie standard (elle retourne un ***int*** correspondant au caractère affiché).

Exemple d'utilisation des 2 fonctions :

```
int main ()
{
  char c;
  while ((c = getchar()) != '\n')
          putchar(c);
}
```

Cette itération permet de récupérer les caractères saisis (au clavier) jusqu'au caractère \n (touche entrée) et de les afficher au fur et à mesure.