C

Ch 3 – Les structures de contrôle

I. Introduction

A. Déroulement linéaire d'un programme

A partir de l'entrée dans la fonction « main », l'exécution des instructions va se dérouler séquentiellement. Les structures de contrôles vont permettre de définir des traitements plus complexes, en modifiant l'enchainement des instructions.

II. Structures de contrôle conditionnelles

A. Structure conditionnelle simple

La **STRUCTURE CONDITIONNELLE SIMPLE** permet l'exécution d'un bloc d'instructions seulement si une condition est remplie (= une expression logique a la valeur **true**), sinon aucune instruction n'est exécutée.

Syntaxe :

```
if (expression_logique) instruction ;
```

ou bien :

```
if (expression_logique)
    {
        ... bloc d'instructions ... ;
    }
```

- **expression_logique**

- représente l'expression logique à évaluer

- **instruction** représente une seule instruction à exécuter si expression_logique vaut true
- **bloc d'instructions** : représente le bloc d'instruction à exécuter si expression_logique vaut true
  - bloc d'instructions à exécuter si 'expression_logique' est vraie

Exemple :

```
if (age >= AGE_MAJORITE) printf("vous etes majeur !");
```

```
if (age >= AGE_MAJORITE)
   {
     printf("votre age est supérieur ou egal à %d" ,AGE_MAJORITE);
     printf(" : vous etes majeur !\n");
     }
```

B. Structure conditionnelle avec alternative

La **STRUCTURE CONDITIONNELLE avec ALTERNATIVE** permet l'exécution d'un bloc d'instructions si une expression logique a pour valeur **true** et d'un autre bloc d'instructions dans le cas contraire.
Elle offre le choix entre 2 possibilités d'action.

Syntaxes :

```
if (expression_logique)
        instruction_1;
    else instruction_2;
```

ou bien :

```
if (expression_logique)
    {
        ... bloc d'instructions 1... ;
    }
    else
    {
        ... bloc d'instructions 2... ;
    }
```

Ou bien encore toute combinaison des 2.

- **expression_logique** : représente l'expression logique à évaluer
- **instruction1** représente une seule instruction à exécuter si expression_logique est vraie
- **instruction2** représente une seule instruction à exécuter si expression_logique n'est pas vraie
- **bloc d'instructions 1** :
  - représente le bloc d'instructions à exécuter si 'expression_logique' est vraie
- **bloc d'instructions 2** :
  - représente le bloc d'instructions à exécuter si 'expression_logique' n'est pas vraie

Exemple :

```
if (age >= AGE_MAJORITE)
              printf("vous etes majeur !");
    else      printf("vous etes mineur !" ;
```

C. Imbrication de structures conditionnelles.

Syntaxe 1 :

```
if (expression_logique 1)
    { // alors
      if (expression_logique 2)
         {
            //... bloc d'instructions (1 ET 2)
         }
         else
         {
            //... bloc d'instructions (1 ET NON 2)
         }
    } // fin alors
    else
    { // sinon
      if (expression_logique 3)
         {
            //... bloc d'instructions (NON 1 ET 3)
         }
         else
         {
            //... bloc d'instructions (NON 1 ET NON 3)
         }
    } // fin sinon
```

D. Structures conditionnelle à alternatives successives.

Syntaxe 2 : structure simplifié (SI - ALORS - SINON SI . . .)

```
if (expression_logique 1)
         {
        //... bloc d'instructions (1)
         }
    else if (expression_logique 2)
         {
        //... bloc d'instructions (NON 1 ET 2)
         }
    else if (expression_logique 3)
         {
        //... bloc d'instructions (NON 1, NON 2 ET 3)
         }
    else
         {
        //... bloc d'instructions (NON 1, NON 2, NON 3)
         }
```

E. Structure conditionnelle à choix multiples

La **STRUCTURE CONDITIONNELLE A CHOIX MULTIPLE** permet de choisir entre plus de 2 possibilités de blocs d'actions à exécuter en fonction de valeurs différentes d'une variable (ou une expression numérique)

Syntaxe :

```
switch (valeur)
{
     case cas1 :
          ... bloc d'instructions 1... ;
          break ;
     case cas2 :
          ... bloc d'instructions 2... ;
          break ;
     . . .
     case casN :
          ... bloc d'instructions N... ;
          break ;
     default :
          ... bloc d'instructions sinon... ;
          break ;
}
```

- **Valeur :** variable ou expression de type entier
- **cas1, cas2, casN :**:
  - représente chacune des valeurs évaluées
- **bloc d'instructions 1, 2, N :**:
  - chacun des blocs d'instructions
- **break** : **cette instruction permet de mettre fin à l'exécution des instructions d'un cas et reprend l'exécution après la fin du bloc switch ; sans cette instruction, l'exécution se poursuit aux instructions du cas suivant**
- **default :**
  - représentent le bloc d'instructions à exécuter dans le cas où aucun cas n'a été sélectionné (ou bien oubli de l'instruction de sortie d'un cas : break)

```
printf("entrez un numéro de couleur :");
scanf("%d",&couleur);
switch (couleur)
{
     case 1 :
          printf("rouge");
          break ;
     case 2 :
          printf("vert");
          break ;
     case 3 :
          printf("bleu");
          break ;
```

```
     default :
          printf("le numéro n'est pas un numéro de couleur connu");
          break ;
} // fin switch
```

F. L'opérateur conditionnel (ternaire) ? :

L'opérateur conditionnel « **?** » évalue **une expression logique** et renvoie l'une ou l'autre de 2 valeurs selon l'évaluation de l'expression.

Syntaxe :

```
(expression_condition) ? expression si vrai : expression si faux
```

- **Expression_condition :** expression conditionnelle évaluée
- **expression :**:expression évaluée

Exemple : plus grand de 2 nombres

```
c = (a > b) ? a : b;
```

→ Est équivalent à :

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

Exemple : valeur absolue de x

```
x = (x >= 0) ? x : -x;
```

→ Est équivalent à :

```
if (x >= 0) x = x ; else x = -x;
```

III.Structures de contrôles itératives

Les boucles, ou itérations, permettent la répétition d'un bloc d'instructions. Le nombre de répétitions est contrôlé par une condition de poursuite.

A. Généralités

Les **STRUCTURES ITERATIVES** permettent l'exécution répétée d'un bloc d'instructions.
Le nombre de répétitions est contrôlé par une **CONDITION DE POURSUITE DE LA REPETITION.**

La condition d'arrêt est exprimée selon une logique de poursuite : la boucle est répétée tant qu'une condition est vérifiée

LA CONDITION DE POURSUITE (OU D'ARRET) D'UNE BOUCLE DOIT ETRE PARFAITEMENT IDENTIFIEE.

B. Boucle TANT QUE : while

Dans la boucle **while**, l'instruction (ou le bloc d'instructions) est **EXECUTE SI UNE CONDITION EST VERIFIEE ET TANT QU'ELLE EST VERIFIEE**.
L'instruction, ou le bloc d'instruction, sera exécuté 0 à N fois.

Syntaxe :

```
// Initialiser les variables qui participent à la condition
```

```
puis :

while (condition) instruction ;
ou bien :
while (condition)
     {
       . . . bloc d'instructions . . . ;
       // modifier la/les variables participant à la condition
     } // fin while
```

- **condition = Expression_logique**
  - l'expression à évaluer : si VRAI, exécution du bloc d'instructions, et répétition de l'exécution TANT QUE cette expression logique a pour valeur VRAI
- **bloc d'instructions**:
  - bloc d'instructions à exécuter

```
nombre = 0 ; // initialisation
while (nombre <= 10) // test
      {
       nombre++; // modification avant le nouveau test
      }
```

```
printf("Entrez un nombre supérieur à 10 :") ;
scanf("%d",&nombre); // initialisation
while (nombre <= 10) // test
      {
       printf("Entrez un nombre correct (supérieur à 10) :");
       scanf("%d",&nombre); // modification avant le nouveau test
      }
```

ATTENTION : LES ELEMENTS CONSTITUANT L'EXPRESSION LOGIQUE DOIVENT IMPERATIVEMENT AVOIR ETE INITIALISEES AVANT LA BOUCLE ET DEVRONT ETRE A NOUVEAU MODIFIEES A L'INTERIEUR DU BLOC D'INSTRUCTIONS (SINON ON OBTIENT UNE BOUCLE INFINIE !)

C. Boucle REPETER … JUSQU'A : do – while

La structure REPETER JUSQU'A n'existe pas directement en C. La condition d'arrêt de la boucle doit être convertie en condition de poursuite pour le « do … while(condition de poursuite) ; »

La boucle **do … while** permet d'exécuter un bloc d'instructions **au moins une fois et tant qu'une expression logique est vérifiée**
Le bloc d'actions s'exécutera de 1 à N fois.

Syntaxe :

```
do instruction ;
    while (condition) ;
ou bien :
do
{
```

```
    . . .  Bloc d'instructions . . . ;
} while (condition) ;
```

- **Condition :**
  - l'expression à évaluer : exécution du bloc d'instructions 1 fois, et répétition éventuelle TANT QUE cette expression logique a pour valeur VRAI
- **instruction, bloc d'instructions**:
  - instruction ou bloc d'instructions à exécuter une ou plusieurs fois

```
do
{
       printf("Entrez un nombre supérieur à 10 :") ;
       scanf("%d",&nombre); // initialisation / modification
} while (nombre <= 10) ; // test
```

ATTENTION : LES ELEMENTS CONSTITUANT L'EXPRESSION LOGIQUE DOIVENT IMPERATIVEMENT ETRE MODIFIES DANS LE BLOC D'INSTRUCTIONS (SINON ON OBTIENT UNE BOUCLE INFINIE !)

D. Boucle POUR : for

La boucle **POUR** permet de répéter l'exécution d'un bloc d'instructions en faisant varier une variable de boucle : cette dernière permet de compter le nombre d'itérations et sert à déterminer la condition de poursuite de la boucle.
LE NOMBRE D'ITERATIONS EST CONNU.

Syntaxe :

```
for (initialisation ; condition ; incrémentation) instruction;
ou bien :
for (initialisation ; condition ; incrémentation)
     {
        . . . bloc d'instruction . . .;
     }
```

- **Initialisation** : initialisation de la variable de boucle
- **condition** : condition de poursuite de la boucle
- **incrémentation** : incrémentation (ou modification) de la variable de boucle

Exemple pour afficher les nombres de 0 à 10 :

```
int i;
for(i=0 ;i<=10 ;i++) printf("%d", i) ; //
```

→ Equivalent à :

```
int i;
i = 0; // initialisation
while (i<=10) // test
    {
      printf("%d", i) ;
      i++; // incrémentation
    }
```

UTILISER UNE VARIABLE DE TYPE ENTIER COMME COMPTEUR DE BOUCLE.
NE PAS MODIFIER LA VARIABLE DE BOUCLE DANS LE CORPS DE LA STRUCTURE.

Syntaxe équivalente avec while :

```
Initialisation ;
while (condition)
    {
    instruction;
    incrémentation ;
    }
```

Syntaxe équivalente avec do while :

```
Initialisation ;
if (condition)
  {
   do
    {
    instruction;
    incrémentation ;
    } while (condition);
  }
```

IV. Ruptures de séquences

Le langage C offre la possibilité de quitter un bloc d'instructions au sein d'une structure répétitive au bien d'une structure conditionnelle à choix multiple (switch).

CES INSTRUCTIONS (MIS A PART L'INSTRUCTION BREAK POUR LE SWITCH) DOIVENT ETRE UTILISEES SEULEMENT QUAND ELLES SIMPLIFIENT LE CODE, ET AMELIORENT LA LISIBILITE ET LA MAINTENANCE DU PROGRAMME.

A. break

L'instruction **break** permet de **QUITTER LA STRUCTURE DE CONTROLE** en cours d'exécution et poursuit l'exécution à l'instruction qui suit.
Elle **DOIT ETRE UTILISEE** dans la structure conditionnelle à choix multiples **switch.**
Elle PEUT ETRE UTILISEE **AVEC PRECAUTION** dans les structures répétitives..

Exemple 1 : voir l'instruction **switch**

Exemple 2, dans une répétitive : (à éviter…)

```
for (;;) // boucle sans fin (= anglais « forever », pour toujours)
{
   printf("Entrez un nombre supérieur à 10 :") ;
   scanf("%d",&nombre);
   if (nombre > 10) break ;
}
```

```
// suite des instructions
```

PREFERER L'EXEMPLE Ci-DESSOUS :

```
do
{
   printf("Entrez un nombre supérieur à 10 :") ;
   scanf("%d",&nombre);
} while (nombre <= 10)
```

B. continue

L'instruction **continue** SAUTE LES INSTRUCTIONS DU BLOC EN COURS DANS UNE ITERATION ET PASSE A L'ITERATION SUIVANTE.

Exemple 1 : calcul de la somme des 1000 premiers entiers pairs : OK

```
somme = 0;
for (i=1; i<=1000; i++)
{
   if (i % 2 != 0) continue; // i non pair, itération suivante
   somme = somme  + i;
}
```

Dans l'exemple ci-dessus, « continue » saute les instructions de la boucle puis passe par la phase d'incrémentation de la variable de boucle : OK

Exemple 2 : calcul de la somme des 1000 premiers entiers pairs : **ATTENTION : BOUCLE INFINIE (l'incrémentation n'est pas réalisée au premier nombre pair !)**

```
somme = 0;
i=1;
while (i<=1000)
{
   if (i % 2 != 0) continue; // i non pair, itération suivante
   somme = somme  + i;
   i++;
}
```

Dans l'exemple ci-dessus, « continue » saute les instructions de la boucle et revient au test, sans que la variable de boucle n'ait été incrémentée : BOUCLE INFINIE !!!

C. return

L'instruction **return** QUITTE IMMEDIATEMENT LA FONCTION EN COURS D'EXECUTION, renvoie une valeur à l'appelant et rend le contrôle de l'exécution à l'endroit où elle avait été appelée. (cf. fonctions)

```
/* exemple de la fonction main */
int main ()
{
  . . .
     return 0;
  . . .
}
```