C/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 est vraie
- **bloc d'instructions** :
  - bloc d'instructions à exécuter si 'expression_logique' est vraie

Exemple :

```
if (Vage >= AGE_MAJORITE) cout << "vous etes majeur !" ;
```

```
if (Vage >= AGE_MAJORITE)
   {
     cout << "votre age est supérieur ou egal à " << AGE_MAJORITE;
     cout << " : vous etes majeur !" << endl;
     }
```

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... ;
    }
```

- **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 (Vage >= AGE_MAJORITE)
                cout << "vous etes majeur !" ;
    else        cout << "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
```

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)
         }
```

D. 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** : **OBLIGATOIRE, SINON POURSUITE DANS LE BLOC DU CAS SUIVANT**
  - instruction qui, une fois le bloc d'instruction exécuté, quitte la structure **switch**
- **default :**
  - représentent le bloc d'instructions à exécuter dans le cas où aucun cas ne correspond : : UN SEUL BLOC SERA EXECUTE

```
cout << "entrez un nombre POSITIF inférieur à 10 :" ;
cin >> Vnombre ;
switch (Vnombre)
{
    case 2 :
    case 4 :
    case 6 :
    case 8 :
        cout << "nombre pair";
        break ;
    case 1 :
    case 3 :
    case 5 :
    case 7 :
```

```
     case 9 :
          cout << "nombre impair";
          break ;
     default :
          cout << "le nombre n'est par correct" ;
          break ;
} // fin switch
```

E. L'expression conditionnelle et opérateur ternaire

L'expression conditionnelle permet l'évaluation de l'une ou l'autre de 2 expressions en fonction du caractère VRAI ou FAUX d'une expression conditionnelle, grâce à l'utilisation d'un opérateur ternaire (3 éléments constitutifs).

Syntaxe :

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

- **condition :** expression conditionnelle évaluée
- **expression :**:
  - expression évaluée

Exemple : plus grand de 2 nombres

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

Exemple : valeur absolue de x

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

Exemple : valeur absolue de x (avec mémorisation)

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

III.Structures de contrôles répétitives

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

A. Généralités

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

La condition d'arrêt peut être exprimée de 2 manières logiques :

- Logique de poursuite : quelle condition permet à la boucle de continuer ?
- Logique d'arrêt : Quand la boucle s'arrête-elle ?

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

B. Boucle TANT QUE - FAIRE : while

Dans la boucle **while**, l'instruction (ou le bloc d'instructions) est exécuté tant qu'une condition est vraie.
L'instruction, ou le bloc d'instruction, sera exécuté 0 à N fois.

Syntaxe :

```
// initialiser les éléments de l'expression 'test'
puis :

while (test) instruction ; // avec modification de 'test'
ou bien :
while (test)
     {
        . . . bloc d'instructions . . . ;
         // modifier les éléments de l'expression 'test'
     } // fin while
```

- **Test = 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

```
cout << "Entrez un nombre supérieur à 10 :" ;
cin >> Vnombre ;
while (Vnombre <= 10)
      {
       cout << "Entrez un nombre supérieur à 10 :" ;
       cin >> Vnombre ;
      }
```

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

C. Boucle FAIRE … TANT QUE : do - while

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

Syntaxe :

```
do instruction ; // avec modification de 'test'
    while (test) ;
ou bien :
do
{
    . . .  Bloc d'instructions . . . ;
         // modifier les éléments de l'expression 'test'
} while (test) ;
```

- **Expression_logique**
  - 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
{
       cout << "Entrez un nombre supérieur à 10 :" ;
       cin >> Vnombre ;
} while (Vnombre <= 10) ;
```

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

- **Initialisation** : initialisation de la variable de boucle
  - **Particularité C++ : il est possible de déclarer la variable de boucle dans la phase d'initialisation : la variable déclarée ne sera alors définie qu'à l'intérieur de la boucle : ceci est déconseillé[1]**
- **test** : condition déterminant la poursuite de la boucle (test sur la variable de boucle)
- **incrémentation** : incrémentation (ou modification) de la variable de boucle

```
int i;
for(i=0 ;i<=10 ;i=i++) cout << i ; //

for (int i=0; i<=10; i=i++) cout << i; // déconseillé
```

UTILISER UNE VARIABLE DE TYPE ENTIER COMME COMPTEUR DE BOUCLE.

Syntaxe équivalente avec while :

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

Syntaxe équivalente avec do while :

[1] Certains compilateurs ont une règle différente : une fois la variable déclarée dans la boucle 'for', elle reste déclarée après l'instruction. On a donc des problèmes d'incompatibilité potentiels en fonction du compilateur.

```
Initialisation ;
If (test)
  {
   do
    {
    instruction;
    incrémentation ;
    } while (test);
  }
```

IV. Ruptures de séquences

Le langage C++ offre la possibilité de sortir d'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 DU 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 être utilisée **AVEC PRECAUTION** dans les structures répétitives..

Exemple 1 : voir l'instruction **switch**
Exemple 2, dans une répétitive : **EVITER**

```
for (;;) // boucle sans fin (= anglais « forever », pour toujours)
{
   cout << "Entrez un nombre supérieur à 10 :"
   cin >> Vnombre ;
   if (Vnombre > 10) break ;
}
// instruction réalisée à la sortie du break
```

PREFERER L'EXEMPLE Ci-DESSOUS :

```
do
{
   cout << "Entrez un nombre supérieur à 10 :"
   cin >> Vnombre ;
} while (Vnombre <= 10)
```

B. continue

L'instruction **continue** saute les instructions de la boucle en cours et poursuit à l'itération suivante.

```
/* Calcul de la somme des 1000 premiers entiers pairs */
Vsomme = 0;
for (i=1; i<=1000; i++)
```

```
{
   if (i % 2 == 1) continue; // i non pair, incrément suivant
   Vsomme = Vsomme  + i;
}
```

C. return

L'instruction **return** quitte la fonction en cours d'exécution et revoie une valeur à l'appelant. (cf. fonctions)

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