COURS et TP DE LANGAGE C++

Chapitre 3

Les tests et les boucles

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 3

LES TESTS ET LES BOUCLES

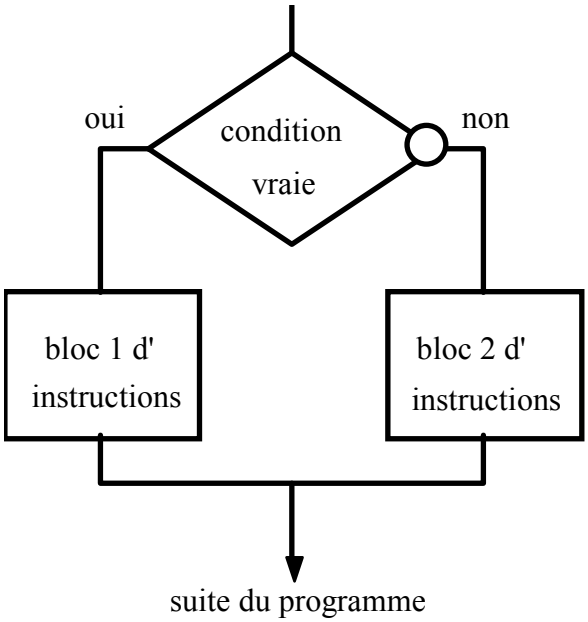
Un programme écrit en C++ s'exécute séquentiellement, c'est à dire instruction après instruction.
Ce chapitre explique comment, dans un programme, on pourra :

- ne pas exécuter une partie des instructions, c'est à dire faire un saut dans le programme,
- revenir en arrière, pour exécuter plusieurs fois de suite la même partie d'un programme.

L'INSTRUCTION SI ... ALORS ... SINON ...

Il s'agit de l'instruction: **si (condition vraie)**
alors {BLOC 1 D'INSTRUCTIONS}
sinon {BLOC 2 D'INSTRUCTIONS}

Organigramme:



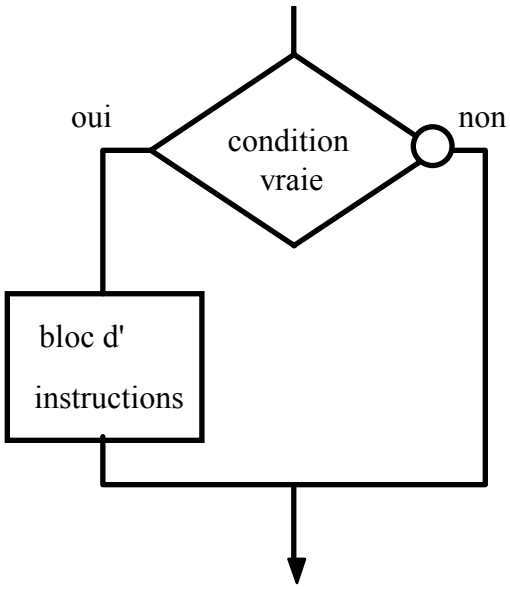
Syntaxe en C:

```
if (condition)
        {
        .....;          // bloc 1 d'instructions
        .....;
        .....;
        }
        else
```

```
{
.....;                    // bloc 2 d'instructions
.....;
.....;
}
suite du programme ...
```

Si la condition est vraie, seul le bloc1 d'instructions est exécuté, si elle est fausse, seul le bloc2 est exécuté.
Dans tous les cas, la suite du programme sera exécutée.

Le bloc "sinon" est optionnel: **si (condition)**
alors {BLOC D'INSTRUCTIONS}



Syntaxe en C:

```
if (condition)
        {
        .....;                    // bloc d'instructions
        .....;
        .....;
        }
suite du programme...
```

Si la condition est vraie, seul le bloc1 d'instructions est exécuté, si elle est fausse, on passe directement à la suite du programme.

Remarque: les {} ne sont pas nécessaires lorsque les blocs ne comportent qu'une seule instruction.

LES OPERATEURS LOGIQUES

condition d'égalité: **if (a==b)** **" si a est égal à b "**

condition de non égalité: **if (a!=b)** **" si a est différent de b "**

conditions de relation d'ordre: **if (a<b)** **if (a<=b)** **if (a>b)** **if (a>=b)**

plusieurs conditions devant être vraies simultanément, ET LOGIQUE:
if ((expression1) && (expression2)) **" si l'expression1 ET l'expression2 sont vraies "**

une condition devant être vraie parmi plusieurs, OU LOGIQUE
if ((expression1) || (expression2)) **" si l'expression1 OU l'expression2 est vraie "**

condition fausse **if (!(expression1))** **" si l'expression1 est fausse "**

Toutes les combinaisons sont possibles entre ces tests.

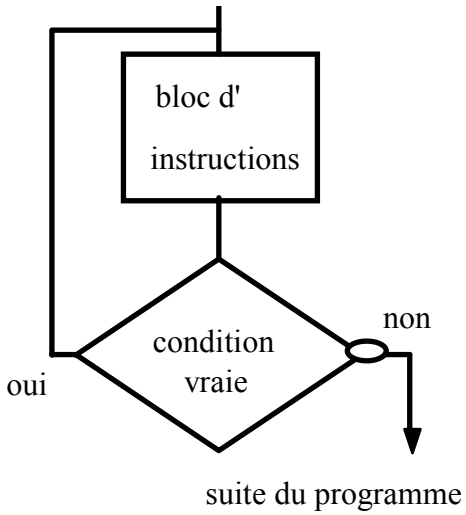
Exercice III-1: L'utilisateur saisit un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui il renvoie cette lettre en minuscule, sinon il renvoie un message d'erreur.

Exercice III-2: Dans une élection, I est le nombre d'inscrits, V le nombre de votants, Q le quorum, P = 100V/I le pourcentage de votants, M = V/2 + 1 le nombre de voix pour obtenir la majorité absolue.
Le quorum est le nombre minimum de votants pour que le vote soit déclaré valable.
Ecrire un programme qui
1- demande à l'utilisateur de saisir I, Q et V,
2- teste si le quorum est atteint,
3- si oui calcule et affiche P, M, sinon affiche un message d'avertissement.

L'INSTRUCTION REPETER ... TANT QUE ...

Il s'agit de l'instruction: **exécuter {BLOC D'INSTRUCTIONS}**
tant que (condition vraie)

Organigramme:



Syntaxe en C:

```
do
        {
        .....;          // bloc d'instructions
        .....;
        .....;
        }
while (condition);
suite du programme ...
```

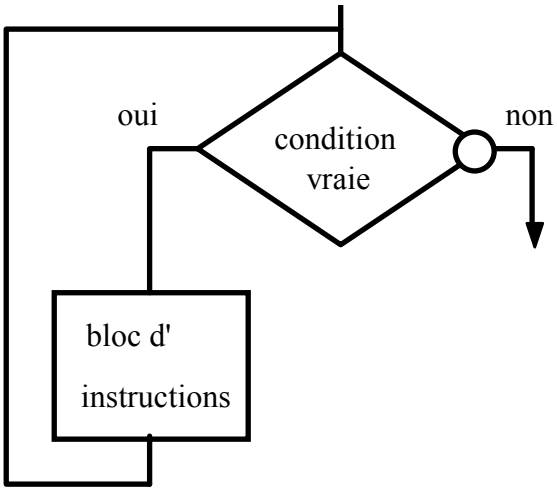
Le test se faisant **après,** le bloc est exécuté au moins une fois.

Remarque: les { } ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

LA BOUCLE TANT QUE ... FAIRE ...

Il s'agit de l'instruction: **tant que (condition vraie)**
Exécuter {BLOC D'INSTRUCTIONS}

Organigramme:



Syntaxe en C:

```
while (condition)
        {
        .....;              // bloc d'instructions
        .....;
        .....;
        }
suite du programme ...
```

Le test se fait **d'abord,** le bloc d'instructions n'est pas forcément exécuté.

Remarque: les { } ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

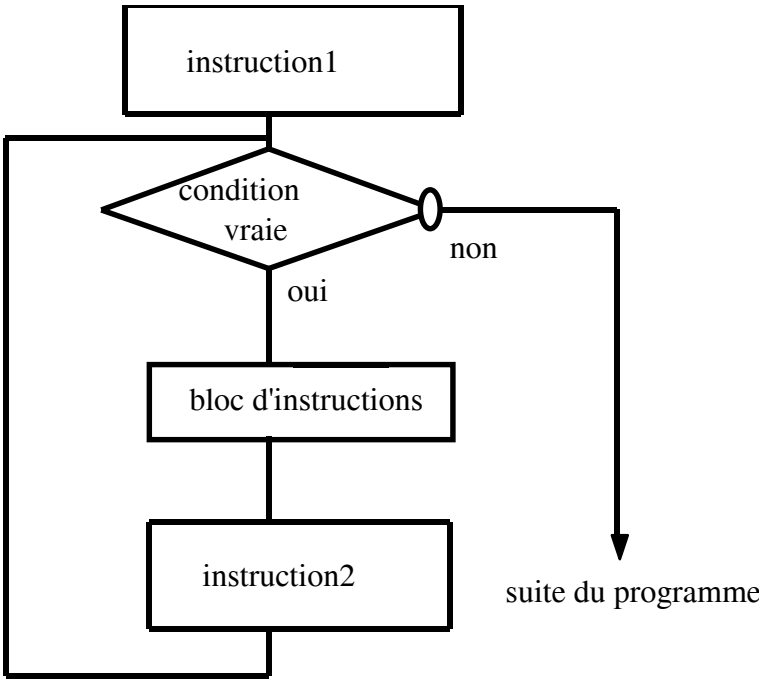
Remarque: On peut rencontrer la construction suivante: **while (expression);** terminée par un ; et sans la présence du bloc d'instructions. Cette construction signifie: **"tant que l'expression est vraie attendre". Ceci ne doit être exploité que par rapport à des événements externes au programme (attente de la frappe du clavier par exemple).**

L'INSTRUCTION POUR ...

Il s'agit de l'instruction:

pour (instruction1; condition; instruction2)
{BLOC D'INSTRUCTIONS}

Organigramme:



Syntaxe en C:

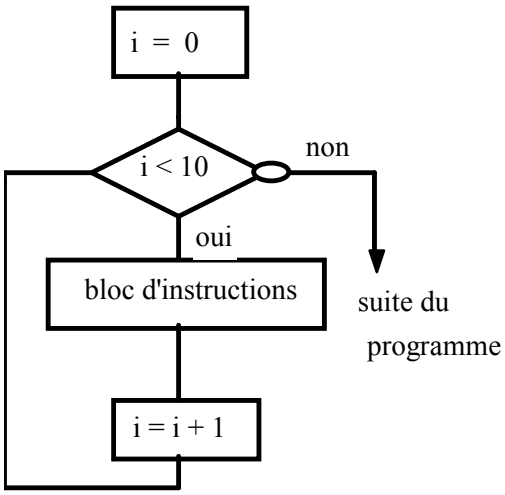
```
for(instruction1; condition; instruction2)
                    {
                    .....;          // bloc d'instructions
                    .....;
                    .....;
                    }
                    suite du programme ...
```

Remarques:
Il s'agit d'une version enrichie du « while ».
Les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.
Les 3 instructions du for ne portent pas forcément sur la même variable.
Une instruction peut être omise, mais pas les ;

Exemples:

```
for(int i = 0 ; i<10 ; i++)
{
.....;          // bloc d'instructions
.....;
.....;
}
suite du programme ...
```

correspond à l'organigramme suivant:



La boucle
```
for(;;)
{
.....;              // bloc d'instructions
.....;
.....;
}
```
est une boucle infinie (répétition infinie du bloc d'instructions).

Utilisation de variables différentes:

```
resultat = 0;
for(int i = 0 ; resultat<30 ; i++)
        {
        .....;              // bloc d'instructions
        .....;
        .....;
        resultat = resultat + 2*i;
        }
```

Exercice III-3:
Ecrire un programme permettant de saisir un entier n, de calculer n!, puis de l'afficher.
Utiliser une boucle do ...while puis while puis for.
Quelle est la plus grande valeur possible de n, si n est déclaré int, puis unsigned int?

Exercice III-4:
La formule récurrente ci-dessous permet de calculer la racine du nombre 2 :
$U_0 = 1$
$U_i = (U_{i-1} + 2/U_{i-1}) / 2$
Ecrire un programme qui saisit le nombre d'itérations, puis calcule et affiche la racine de 2.

L'INSTRUCTION AU CAS OU ... FAIRE ...

L'instruction switch permet des choix multiples **uniquement sur des entiers (int) ou des caractères (char).**

Syntaxe:

```
switch(variable de type char ou int)        au cas où la variable vaut:
{
case  valeur1: ......;                      - cette valeur1: exécuter ce bloc d'instructions.
               .......;
               break;                       - se brancher à la fin du bloc case.
      valeur2:.......;                      - cette valeur2:  exécuter ce bloc d'instructions.
               ........;
               break;                             - se brancher à la fin du bloc case.
       .
       .                                          etc ...
       .
    default:  .......;                      - aucune des valeurs précédentes: exécuter ce bloc
              ........;                     d'instructions, pas de "break" ici.

}
```

le bloc "default" n'est pas obligatoire.

L'instruction switch correspond à une cascade d'instructions if ...else

Exemple:
Cette instruction est commode pour fabriquer des "menus":

```
char choix;
cout<<"LISTE PAR GROUPE TAPER 1\n";
cout<<"LISTE ALPHABETIQUE TAPER 2\n";
cout<<"POUR SORTIR TAPER S\n";
cout<<"\nVOTRE CHOIX : ";
cin >> choix;
switch(choix)
{
case '1':  .......;
           .......;
          break;

case '2':  ......;
           ......;
           break;

case 'S': cout<<"\nFIN DU PROGRAMME ....";
           break;
```

```
default;  cout<<"\nCE CHOIX N'EST PAS PREVU ";  // pas de break ici
}
```

COMPLEMENT SUR LES TESTS

En langage C++, **une expression nulle de type entier (int) est fausse, une expression non nulle de type entier (int) est vraie.**

Exemples:

```
int a,b,c,delta;                    est équivalent à    int a,b,c,delta;
delta = b*b-4*a*c;                                      delta = b*b-4*a*c;
if(delta != 0)                                          if(delta)
{ ..... }                                               { ..... }
```

```
int a,b,c,delta;                    est équivalent à    int a,b,c,delta;
delta = b*b-4*a*c;                                      delta = b*b-4*a*c;
if(delta == 0)                                          if(!delta)
{ ..... }                                               {.....}
```

En langage C++, le type booléen a été introduit. Il prend les valeurs TRUE ou FALSE.
Par exemple :

```
bool test ;
test = (x<45) ;
if ( test == TRUE)
{……..}
```

```
ou plus simplement                  ou aussi
if( (x<45) == TRUE) )               if (x<45) // !!!!!
```

EXERCICES RECAPITULATIFS

Exercice III_5: résoudre $ax^2 + bx + c = 0$.

Exercice III_6: La fonction kbhit appartient à la bibiothèque conio.h. Une fonction équivalente peut exister avec d'autres compilateurs. La fonction kbhit teste si un caractère a été frappé au clavier. Tant que ce n'est pas vrai kbhit renvoie 0 (ceci signifie que la valeur retournée par la fonction kbhit est 0).

Exemple: **while(kbhit() == 0)** // tant qu'aucun caractère n'a été frappé exécuter la boucle

{ ..... }

Cette écriture est équivalente à:
while(!kbhit()) // tant que kbhit est faux, exécuter la boucle
{.....} Ecrire un programme qui affiche le carré des entiers 1, 2, 3 ....., toutes les 500 ms tant qu'aucun caractère n'a été frappé au clavier. Générer la temporisation à l'aide d'une boucle for et d'un décompteur.

CORRIGE DES EXERCICES

Exercice III-1:

```
#include <iostream.h>
#include <conio.h>

    void main()
   {
      char c,d;
      cout <<"ENTRER UNE LETTRE MAJUSCULE :";
      cin>>c;

      d = c + 'a' - 'A';

      if((c>='A') && (c<='Z')) cout<<"LETTRE EN MINUSCULE : "<<d<<"\n";
      else cout<<"CE N'EST PAS UNE LETTRE MAJUSCULE\n";

      cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice III-2:

```
#include <iostream.h>
#include <conio.h>

    void main()
   {
      int I, V, M, P, Q;
      cout<<"Entrer le nombre d'inscrits : "; cin>>I;
      cout<<"Entrer le nombre de votants : "; cin>>V;
      cout<<"Entrer le quorum : "; cin>>Q;
      P = V*100/I;
      M = V/2 +1;
      if(P > Q)
      {
         cout<<"Quorum atteint - vote valable\n";
         cout<<"Participation: "<<P<<"% - Majorite obtenue pour : ";
         cout<< M <<" bulletins\n";
      }
      else
      {
         cout<<"Quorum non atteint - vote non valable\n";
      }
      cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice III_3:

```
#include <iostream.h>
#include <conio.h>

    void main()
   {
      int n, fac= 1;
      cout<<"ENTRER UN ENTIER : ";cin>>n;
      for (int i=1;i<=n;i++) fac= fac * i;
      cout<<"\nn ="<<n<<" n!= "<<fac;
      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Les entiers sont des nombres de 32 bits:
n int : n! maximum= 12!
n unsigned int : n! maximum = 12!

Exercice III_4:

```
#include <iostream.h>
#include <conio.h>

    void main()
   {
      int n,;
      float U0=1.0, Ui;
      cout <<"ENTRER LE NOMBRE D'ITERATIONS: ";cin >> n;
      for (int i=0;i<n;i++)
      {
         Ui = (U0 + 2.0/U0)/2.0;
         U0 = Ui;
      }

      cout <<"RESULTAT = "<< Ui <<"\n";
      cout <<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";

      getch();
   }
```

Exercice III_5:

```
#include <iostream.h>
#include <conio.h>
#include <math.h>// contient la fonction racine

    void main()
   {
      float a,b,c,x1,x2;
   // saisie de A, B et C
      cout<<"\t\t\tRESOLUTION DE L'EQUATION DU SECOND DEGRE\n";
      cout<<"\t\t\t                2\n";
      cout<<"\t\t\t             AX +BX+C=0\n\n\n";
      cout<<"SAISIR A B C SEPARES PAR RETURN\n";
      cout<<"A = "; cin>>a;
      cout<<"B = "; cin>>b;
      cout<<"C = "; cin>>c;

   // début du calcul : cas particuliers
      if((a==0)&&(b==0)&&(c==0))cout<<"INFINITE DE SOLUTIONS\n";
      else if((a==0)&&(b==0)&&(c!=0))cout<<"PAS DE SOLUTIONS\n";
      else if((a==0)&&(b!=0))cout<<"UNE SOLUTION: X= "<<-c/b<<"\n";
      else //cas général
      {_// delta : variable "brouillon"
         float delta = b*b-4*a*c;  // déclarée localement
         cout<<"DELTA="<<delta<<"\n";
         if(delta<0)cout<<"DELTA NEGATIF PAS DE SOLUTION\n";
         else
         {
            if(delta==0)cout<<"DELTA NUL, UNE SOLUTION X="<<-b/2/a<<"\n";
            else
            {
               x1= (-b+sqrt(delta))/2/a;
               x2= (-b-sqrt(delta))/2/a;
               cout<<"DELTA POSITIF DEUX SOLUTIONS\n";
               cout<<"X1= "<<x1<<"  X2= "<<x2<<"\n";
            }
         }
      }

   // calculs terminés
      cout<<"\n\nPOUR CONTINUER FRAPPER UNE TOUCHE";
      getch();
   }
```

Exercice III_6:

```
#include <iostream.h>
#include <conio.h>

    void main()
   {
      int i = 0;
   // le type float pour générer des temporisations suffisamment longues
      float x, tempo=5000000.0;
      cout<<"POUR SORTIR DE CE PROGRAMME FRAPPER UNE TOUCHE ...\n";
      do
      {
         cout<<"i = "<< i <<"  i*i="<< i*i <<"\n";
         for(x=tempo; x>0; x--);
         i++;
      }
      while(kbhit()==0); // on peut aussi écrire while(!kbhit());
   }
```