CHARLES CARNUS
LYCÉE TECHNIQUE
ENSEIGNEMENT SUPÉRIEUR

Programmation

C++ : Tableaux et structures de contrôle

Prof : Kamal Boudjelaba

22 août 2022

B T S S N

LTP Charles Carnus
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR–EC

Table des matières

1 Les structures de contrôle **1**
1.1 Condition if . . . 1
1.2 Condition if, else . . . 1
1.3 Condition else if . . . 2
1.4 Condition switch . . . 2
2 Les boucles **3**
2.1 La boucle for . . . 3
2.2 La boucle while . . . 4
2.3 La boucle do ... while . . . 4
3 Opérateurs logiques et relationnels **5**
4 Conseils : Erreurs communes **6**
5 Les tableaux **8**
5.1 Initialisation d'un tableau . . . 8
5.2 Accès aux éléments d'un tableau . . . 9
6 Les vecteurs **10**
6.1 Manipulation de vecteurs . . . 10
7 Exercices **13**

Liste des tableaux

1 Opérateurs logiques en C++ . . . 5
2 Opérateurs relationnels en C++ . . . 5

1. Les structures de contrôle

1.1 Condition if

L'utilisation la plus élémentaire d'une instruction *if* consiste à exécuter une ou plusieurs instructions si, et seulement si, une condition donnée est remplie.

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a(10);

    if (a > 0)
    {
        cout << "Vous avez gagné" << endl;
    }

    cout << "Fin du programme" << endl;

    return 0;
}
```

1.2 Condition if, else

Il arrive souvent qu'on souhaite exécuter des instructions données si une condition spécifiée est remplie, et exécuter d'autres instructions dans le cas contraire. Cela peut être implémenté dans le code C++ en utilisant une instruction *if* conjointement avec une instruction *else*.

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a(0);

    if (a > 0)
    {
        cout << "Vous avez gagné" << endl;
    }
    else
    {
        cout << "Vous avez perdu" << endl;
    }
    cout << "Fin du programme" << endl;
    return 0;
}
```

```cpp
bool x = true;
if (x)
{
    std::cout << "Celui-ci sera affiché\n";
}
else
{
    // x est FAUX
    std::cout << "Cela ne sera pas affiché\n";
}
```

1.3 Condition else if

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a(2);

    if (a == 0)
    {
        cout << "Moyen" << endl;
    }
    else if (a == 1)
    {
        cout << "Assez bien" << endl;
    }
    else if (a == 2)
    {
        cout << "Bien" << endl;
    }
    else
    {
        cout << "Très bien" << endl;
    }
    cout << "Fin du programme" << endl;
    return 0;
}
```

1.4 Condition switch

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a(2);

    switch (a)
    {
        case 0:
            cout << "Moyen" << endl;
            break;

        case 1:
            cout << "Assez bien" << endl;
            break;

        case 2:
            cout << "Bien" << endl;
            break;

        default:
            cout << "Très bien" << endl;
            break;
    }
    return 0;
}
```

2. Les boucles

Il arrive souvent dans un programme qu'une même action (instruction) soit répétée plusieurs fois, avec éventuellement quelques variantes. Il est alors fastidieux d'écrire un algorithme qui contient de nombreuses fois la même instruction. Pour gérer ces cas, on fait appel à des instructions en boucle qui ont pour effet de répéter plusieurs fois une même instruction.

Deux formes existent : la première, si le nombre de répétitions est connu avant l'exécution de l'instruction de répétition (On utilise la boucle for), la seconde s'il n'est pas connu (on utilise la boucle while). L'exécution de la liste des instructions se nomme itération.

2.1 La boucle for

La boucle for permet de répéter un certain nombre de fois une partie d'un programme.

```
for (initialisation ; condition ; incrementation)
{
    instructions
}
```

La partie en-tête qui comprend des instructions en trois parties et contrôle l'exécution du corps de la boucle.

1. On exécute l'instruction initialisation
2. On teste la condition
   - si elle est vraie, on exécute instructions, puis l'instruction incrementation puis on revient au 2
   - si elle est fausse, on passe à la suite du programme (on sort de la boucle for).

Exemple 2.1

```
int main()
{
    int compteur(0);

    for (compteur = 0 ; compteur < 10 ; compteur++)
    {
        cout << compteur << endl;
    }
    return 0;
}
```

Exemple 2.2: for et *break* (sortir de la boucle lorsque i est égal à 4)

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  cout << i << "\n";
}
```

Exemple 2.3: for et *continue* (ignorer la valeur de 4 et et continuer avec l'itération suivante)

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

2.2 La boucle while

```
while (condition)
{
    Instructions
}
```

1. On teste condition
   - si elle est vraie, on exécute instructions puis on recommence au 1
   - si elle est fausse, on passe à la suite du programme (on sort de la boucle while).

Exemple 2.4: while et *break* (sortir de la boucle lorsque i est égal à 4)

```
int i = 0;
while (i < 10) {
  cout << i << "\n";
  i++;
  if (i == 4) {
    break;
  }
}
```

Exemple 2.5: while et *continue* (ignorer la valeur de 4 et et continuer avec l'itération suivante)

```
int i = 0;
while (i < 10) {
  if (i == 4) {
    i++;
    continue;
  }
  cout << i << "\n";
  i++;
}
```

2.3 La boucle do ... while

```
do
{
    Instructions
} while (condition);
```

L'instruction do...while exécute toujours la première itération.

1. On exécute instructions
2. On évalue condition
   - si elle est vraie, on recommence au 1
   - si elle est fausse, on passe à la suite du programme (on sort de la boucle do...while).

Exemple 2.6

```
int a(0);
do
{
    cout << "Taper un nombre" << endl;
    cin >> a;
} while (a < 0);
cout << "Vous avez tapé un nbre positif. Le nombre : " << a << endl;
```

3. Opérateurs logiques et relationnels

La combinaison d'opérateurs logiques et relationnels permet d'implémenter n'importe quelle condition raisonnable dans le code C++.
Une premiere utilité de combinaison d'opérateurs logiques et relationnels est de remplacer des instructions if imbriquées par une seule instruction if.

Opérateur	Symbole
AND	&&
OR	ll
NOT	!

Table 1. Opérateurs logiques en C++

Relation	Symbole
Egal à	== (mais pas = : affectation)
Différent de ($\neq$)	!=
Supérieur à	>
Supérieur ou égal à ($\geqslant$)	>=
Inférieur à	<
Inférieur ou égal à ($\leqslant$)	<=

Table 2. Opérateurs relationnels en C++

```
double x, z, p, q;
double y;
if ((x > z) && (p > q))
{
    // Les deux conditions sont remplies
    y = 10.0;
}
```

```
double p, q;
int i;
double y;
if ((p > q) || (i != 1))
{
    // Une ou les deux conditions sont remplies
    y = 10.0;
}
else
{
    // Aucune condition n'est remplie : p<=q et i==1
    y = -10.0;
}
```

4. Conseils : Erreurs communes

Erreur courante de codage : Ci-dessous se trouve un code qui a été écrit avec l'intention de doubler une variable x cinq fois.

```
1 double x = 2.0;
2 for (int i=0; i<5; i++);
3 {
4     x *= 2.0;
5 }
6 std::cout << "x = " << x << "\n";
```

On s'attendrait à ce que ce code produise la valeur $2 \times 2^5 = 64$. Cependant, la sortie réelle de ce code est : $x = 4$ La raison de la sortie surprenante est le point-virgule à la fin de la ligne 2. Il s'agit d'une erreur courante pour les programmeurs qui découvrent le langage. Après avoir vu que la plupart des lignes se terminent par un point-virgule, vous pourriez commencer à prendre l'habitude de terminer chaque ligne par un ";". Lorsque vous voyez la ligne de début d'une instruction *for*, *while* ou *if* sans point-virgule à la fin, il peut être tentant d'en coller une à la fin de cette ligne ! Vous pourriez demander "Si la boucle ne s'exécute pas comme prévu, pourquoi la réponse finale $x = 4$ et non $x = 2$? ». La réponse est que l'espace vide à la ligne 2 entre le ")" et le ";" est interprété comme le corps de la boucle — c'est l'espace vide qui est exécuté 5 fois. Le corps prévu de la boucle (lignes 3 à 5) est traité comme un bloc avec une portée spéciale. Ce bloc n'a aucun lien avec la boucle *for* et est exécuté une seule fois.

Égalité au lieu d'affectation : Lorsque nous avons introduit les opérateurs relationnels dans le tableau **??**, nous avons noté qu'il y a une différence entre un simple = et un double ==. L'opérateur = est un opérateur d'affectation qui prend la valeur de droite et l'affecte à la variable de gauche. L'opérateur d'égalité == renvoie vrai si et seulement si les valeurs à gauche et à droite sont égales.
Une erreur de programmation courante consiste à confondre l'un avec l'autre.

```
1  // Cette ligne erronée n'a aucun effet
2  x == 2+2;
3  // Après avoir testé x par rapport à la valeur 4, le vrai/faux
4  // la réponse est rejetée.
5
6  x = 3;
7  // Cette ligne erronée sera après la valeur de x
8  if (x = 4)
9  {
10     x = 6;
11 }
```

Le code ci-dessus montre deux bogues imprévus courants dans le code C++. La ligne 2 de ce code testera si oui ou non la variable x est égale à 4, mais n'attribuera aucune valeur à x. Cette ligne n'a donc pas d'effet global. Votre compilateur peut vous donner un avertissement. Cependant, comme différents compilateurs donneront des avertissements différents, vous ne devriez pas vous fier à cela. À moins que des indicateurs de compilateur appropriés ne soient utilisés, le compilateur ne donnera aucune erreur car il s'agit d'une syntaxe valide. La deuxième erreur est indiquée aux lignes 8 à 11 du code. Dans ce cas, la ligne 8 du code utilise l'affectation (un seul signe égal) alors que le test d'égalité (un double signe égal) était prévu. Ce code aura pour effet de changer la valeur de x à la valeur 4 alors que ce n'était pas prévu. La condition réellement testée est obtenue à partir de la valeur de l'affectation. Une valeur différente de zéro (dans ce cas, la valeur 4) est interprétée comme True, et donc cette condition est remplie. Le code à l'intérieur des accolades sera donc exécuté, et donc la variable x prendra la valeur 6. Encore une fois, c'est une syntaxe valide donc le compilateur ne peut donner aucun avertissement ou erreur. Certains compilateurs peuvent signaler ces types de problèmes sous forme d'avertissements ou d'erreurs.

Boucle while infinie : Le code ci-dessous a été écrit pour trouver le maximum d'un tableau de quatre nombres positifs appelés positive_numbers. Pourquoi ce code ne quittera-t-il jamais la boucle *while* ?

```
1  double positive_numbers[4] = {1.0, .65, 42.0, 0.01};
2  double max = 0.0;
3  int count = 0;
4  while (count < 4)
5  {
6      if (positive_numbers[count] > max)
7      {
8          max = positive_numbers[count];
9      }
10 }
```

Le problème avec le code ci-dessus est que le nombre entier n'est pas incrémenté dans l'instruction *while*. La variable *count* prendra donc toujours la valeur 0, la condition count < 4 sera toujours satisfaite, et le code ne sortira jamais de la boucle *while*.

Comparer deux nombres à virgule flottante : Si i et j ont été déclarés comme des entiers, et que nous voulons mettre à zéro une autre variable entière k si ces variables prennent la même valeur, alors cela peut facilement être écrit en C++ en utilisant le code suivant :

```
1  int i, j, k;
2  if (i == j)
3  {
4      k = 0;
5  }
```

Supposons, à la place, que nous voulions mettre k à zéro si deux variables à virgule flottante double précision p et q prennent la même valeur. On peut penser qu'une modification très simple du code précédent suffira, où p et q sont déclarés comme des variables à virgule flottante double précision et la condition à la ligne 2 du code est modifiée pour tester l'égalité de p et q. Ceci, en revanche, n'est pas le cas. Les opérations entre nombres à virgule flottante induisent toutes des erreurs d'arrondi. Par conséquent, si la vraie valeur d'un calcul est 5, le nombre stocké peut être 5,000000000000186. Il est peu probable que le test d'égalité de deux variables à virgule flottante à double précision donne la réponse attendue, en raison d'erreurs d'arrondi, il est peu probable que deux de ces variables soient un jour égales. Au lieu de cela, nous devrions vérifier que les deux nombres diffèrent de moins d'un très petit nombre, comme indiqué ci-dessous :

```
1  double p, q;
2  int k;
3  if (fabs(p-q) < 1.0e-8)
4  {
5      k = 0;
6  }
```

Utiliser la sortie : Si vous avez besoin de savoir où votre programme plante et pourquoi, afficher quelques valeurs de variables à des points clés de l'exécution.

Utiliser un débogueur : Si tout le reste échoue, déboguez votre programme à l'aide d'un débogueur.

5. Les tableaux

Les tableaux correspondent aux vecteurs et matrices en mathématiques. Un tableau est caractérisé par sa taille et par le type de ses éléments.

Tableau à 1 dimension (vecteur) :

Déclaration : type nom[taille]
Cette instruction signifie que le compilateur réserve **taille** places en mémoire pour ranger les éléments du tableau.

- int vecteur[10] : le compilateur réserve des places en mémoire pour 10 entiers
- float nombre[5] : le compilateur réserve des places en mémoire pour 5 réels

Note 5.1.
Un élément du tableau est repéré par son indice. En langage C++ (C et PYTHON) les tableaux commencent à l'indice 0. L'indice maximum est donc $taille - 1$.

Tableau à 2 dimensions (matrice) :

Déclaration : type nom[taille1][taille2]

- int matrice[10][3] : tableau de nombres entiers de dimensions 10 lignes et 3 colonnes
- float nombre[2][5] : tableau de nombres réels de dimensions 2 lignes et 5 colonnes

5.1 Initialisation d'un tableau

Généralement, on initialise les tableaux au moment de leur déclaration.

```
int liste[10] = {1,2,4,8,16,3,6,12,25,52};

float nombre[6] = {2.7,5.8,-8.0,0.19,3.14,-2.16};

int y[2][3] = {{1,4,6},{3,7,5}}; // 2 lignes et 3 colonnes
```

Initialisation d'un tableau 2D en utilisant des boucles for imbriquées : Cet exemple montre l'initialisation d'une matrice à 2 dimensions (tableau 2D) à l'aide de boucles for imbriquées. Chaque niveau de boucle imbriquée traite une seule ligne de la matrice, on a donc deux niveaux dans cet exemple. On affiche également les éléments après l'affectation de l'entier aléatoire à la position donnée.

```
#include <iostream>
using namespace std;

int main()
{
    int nb_ligne = 4;
    int nb_col = 3;
    int matrice[nb_ligne][nb_col];

    for (int i = 0; i < nb_ligne; ++i)
    {
        for (int j = 0; j < nb_col; ++j)
        {
            matrice[nb_ligne][nb_col] = rand() % 100;
            cout << matrice[nb_ligne][nb_col] << "\t";
        }
        cout << endl;
    }

    return 0;
}
```

Initialisation d'un tableau 2D en utilisant des boucles while imbriquées :

```
#include <iostream>
using namespace std;

int main()
{
    int nb_ligne = 4;
    int nb_col = 3;
    int matrice[nb_ligne][nb_col];
    int j, i = 0;
    while (i < nb_ligne)
    {
        j = 0;
        while (j < nb_col)
        {
            matrice[nb_ligne][nb_col] = rand() % 100;
            cout << matrice[nb_ligne][nb_col] << "\t";
            j++;
        }
        i++;
        cout << endl;
    }
    return 0;
}
```

Initialisation d'un tableau 2D en utilisant des boucles do...while imbriquées :

```
#include <iostream>
using namespace std;

int main()
{
    int nb_ligne = 4;
    int nb_col = 3;
    int matrice[nb_ligne][nb_col];
    int j, i = 0;
    do {
        j = 0;
        do {
            matrice[nb_ligne][nb_col] = rand() % 100;
            cout << matrice[nb_ligne][nb_col] << "\t";
            j++;
        } while (j < nb_col);
        i++;
        cout << endl;
    } while (i < nb_ligne);
    return 0;
}
```

5.2 Accès aux éléments d'un tableau

Les éléments d'un tableau sont accessibles en plaçant les indices entre crochets, et on peut donc remplir complètement les tableaux `array1` et `array2` en utilisant le code suivant :

```
int array1[2];
double array2[2][3];

array1[0] = 1;
array1[1] = 10;

array2[0][0] = 12.6;
array2[0][1] = -5.1;
array2[0][2] = 5.0;
array2[1][0] = 3.0;
array2[1][1] = -10.2;
array2[1][2] = 5.4; // ou array2[1][2] = array2[0][1] + array2[1][0];
```

6. Les vecteurs

Les vecteurs :

— Un tableau dynamique est un tableau dont le nombre de cases (taille) peut varier au cours de l'exécution du programme (sa taille est ajustable).
— Les vecteurs (vectors) sont un type de tableaux dynamiques.

Pour utiliser les vecteurs en $C++$, il faut inclure dans l'en-tête :
`#include <vector>`

Pour déclarer un vecteur, il faut utiliser la syntaxe suivante :

```
vector</* type des éléments du tableau */> identifiant {};
```

Exemple 6.1

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    vector<double> const tableau_de_double {};
    vector<string> const tableau_de_string {};
    return 0;
}
```

6.1 Manipulation de vecteurs

La fonction at()

Pour accéder à un élément d'un vecteur, on peut utiliser l'opérateur [ ]
Par exemple, pour accéder au 3ème élément du vecteur tableau_de_double, on procède comme suit :

```
vector<int> vecteur = { 11, 12, 13, 14, 15 };
cout << vecteur[2] << endl;
```

Si on met entre les [ ], une valeur qui dépasse la "taille du vecteur -1", le programme peut planter ou avoir un comportement indéterminé. Pour pallier ce problème, on préfère utiliser la fonction `at()` :

```
vector<int> vecteur = { 11, 12, 13, 14, 15 };
cout << vecteur.at(2) << endl;
```

Les fonctions front() et back()

— La fonction **front()** permet d'accéder au premier élément
— La fonction **back()** permet d'accéder au dernier élément

Les fonctions empty() et clear()

— La fonction **empty()** permet de savoir si le vecteur est vide ou non.
— La fonction **clear()** permet de vider tout le vecteur.

Exemple 6.2

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> vecteur;
  // Remplissage du vecteur
  for (int i = 20; i < 29; ++i)
    vecteur.push_back(i);
  cout << "Le vecteur contient " << vecteur.size() << " éléments :" << endl;
  for (int i = 0; i < vecteur.size(); ++i)
  {
    cout << "\t- Valeur " << i+1 <<" est : " << vecteur[i] << endl;
  }
  // On ajoute 3 éléments
  for (int i = 0; i < 3; ++i)
  {
    vecteur.push_back(100);
  }
  cout << endl << "La nouvelle taille est : " << vecteur.size() << " éléments :" << endl;
  for (int i = 0; i < vecteur.size(); ++i)
  {
    cout << "\t- N-Val " << i+1 <<" est : " << vecteur[i] << endl;
  }
}
```

Exemple 6.3

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> mon_tableau { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; // Dépend de la version de C++
    // Affichage du 1er et dernier élément
    cout << "Le premier élément du tableau : " << mon_tableau.front() << "." << endl;
    cout << "Le dernier élément du tableau : " << mon_tableau.back() << "." << endl;
    // Ajout d'éléments au tableau
    mon_tableau.push_back(11);
    mon_tableau.push_back(12);
    // Modification du 2ème élément.
    mon_tableau[1] = 20;
    // Affichage de la taille du nouveau tableau
    cout << "Taille du nouveau tableau : " << mon_tableau.size() << endl;
    // Retirer le dernier élément
    mon_tableau.pop_back();
    cout << "Affichage des éléments du nouveau tableau:\n";
    for (int element : mon_tableau) // Dépend de la version de C++
    {
        cout << element << endl;
    }
    // Vider le tableau.
    mon_tableau.clear();
    cout << "Taille après avoir vidé le tableau : " << mon_tableau.size() << endl;

    // Remplacement du tableau par un tableau de taille en assignant 4 fois la valeur 33
    mon_tableau.assign(4, 33);
    cout << "Taille actuelle de mon tableau : " << mon_tableau.size() << "\n";
    cout << "Les valeurs actuelles sont : " << "\n";
    for (int valeur : mon_tableau)
    {
        cout << valeur << endl;
    }
    cout << '\n';
```

```
37
38          return 0;
39      }
```

7. Exercices

Exercice 1 :

Saisir une matrice d'entiers de dimensions 2×2, calculer et afficher son déterminant.

Exercice 2 :

Un programme contient l'instruction (la déclaration) suivante :

```
1 int tab[20] = {15,-6,-2,-31,24,3,-7,81,-65,32,91,17,-33,68,7,8,-54,72,34,49};
```

Compléter ce programme pour afficher les éléments du tableau sous la forme suivante :

15	−6	−2	−31	24
3	−7	81	−65	32
91	17	−33	68	7
8	−54	72	34	49

Remarque 7.1 :

Pour copier (imprimer ou capturer) le résultat d'une exécution de programme : dans le terminal d'exécution, appuyez sur les touches Fn et Imp écr (variable selon le type de clavier). Vous pouvez alors indiquer le chemin de stockage de votre fichier et lui donner un nom (extension png par défaut).

Exercice 3 :

Écrire 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* ?

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Exercice 4 :

La formule de récurrence suivante permet de calculer la racine carrée du nombre 2 :

$$U_0 = 1$$
$$U_i = \frac{U_{i-1} + \frac{2}{U_{i-1}}}{2}$$

Écrire un programme qui saisit le nombre d'itérations, puis calcule et affiche la racine carrée de 2.

Exercice 5 :

Écrire un programme pour résoudre l'équation $ax^2 + bx + c = 0$

Exercice 6 :

Écrire un programme pour saisir 6 réels, les ranger dans un tableau.
Calculer et afficher leur moyenne $\overline{x}$ et leur écart-type σ.

Avec $\sigma = \sqrt{\dfrac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}$.

- x_i désigne une valeur du tableau
- $\overline{x}$ est la moyenne arithmétique des valeurs du tableau
- n est la taille du tableau

Exercice 7 :

Soit le programme suivant :

```
#include <iostream>
#include <math.h> // #include <cmath>
using namespace std;
int main()
{
float rayon, surface, circonference;
cout<< " Rentrez la valeur du rayon : "<<endl;
cin>>rayon;
if (rayon <= 0)
    {
    cout<<"Erreur de saisie : rayon inférieur à 0 !"<<endl;
    cout<<"Sortie du programme ..."<<endl;
    return EXIT_FAILURE;
    }
else
    {
    surface = M_PI * pow(rayon ,2);
    circonference = M_PI * 2 * rayon;
    cout<<"La surface du cercle de rayon "<<rayon<<" cm est "<<surface<<" cm^2"<<endl;
    cout<<"La circonférence du cercle de rayon "<<rayon<<"cm est "<<circonference<<" cm"<<endl;
    return EXIT_SUCCESS;
    }
}
```

Essayer de comprendre le programme proposé : il s'agit de réaliser les calculs d'aire et de circonférence d'un cercle à partir de son rayon.
Le programme demande la saisie par l'utilisateur de la valeur du rayon. Si le rayon saisi est inférieur ou égal à 0, le programme se termine en renvoyant un code d'erreur au système d'exploitation. Par contre, si le rayon a une valeur positive, les différents calculs sont effectués et affichés.

Travail demandé

1. Tapez le programme dans l'éditeur de texte de votre environnement de développement.
2. Compilez votre programme. Corrigez les éventuelles erreurs de saisie.
3. Exécutez votre programme afin de le tester.
4. Décrivez l'action faite à chaque ligne du programme par un commentaire.
5. Á quel endroit peut-on trouver la valeur de la constante M_PI ? Quelle est cette valeur ?
   Les réponses seront fournies dans votre programme (en commentaire).
6. Á votre avis, à quoi correspond l'instruction pow(rayon,2) ? Donnez votre explication en commentaire dans votre programme.
7. Faites une trace de l'exécution de votre programme à la main. Pour cela, utilisez le débogueur. Le débogueur est une partie logicielle de votre IDE permettant de repérer les conditions dans lesquelles se produit une erreur de fonctionnement d'un logiciel informatique, ou de vérifier le bon fonctionnement de ce logiciel.
   — Á l'aide de ce débogueur, visualiser les variables : rayon, surface et circonference.
   — Faites une exécution pas à pas pour voir l'évolution des variables et la séquence des instructions effectuées.
   — Vous ferez deux essais : un essai avec un rayon de 9 cm et un autre essai avec un rayon de 0 cm.

Exercice 8 :

Un appareil de mesure (oscilloscope) à mémoire connecté à un PC renvoie l'information suivante sous forme d'une chaîne de caractères terminée par '0' au PC : "CHANNELA 0 10 20 30 40 30 20 10 0 -10 -20 -30 -40 -30 -20 -10 "
Afficher sur l'écran la valeur des points vus comme des entiers.
On simulera la présence de l'oscilloscope en initialisant une chaîne de caractères : `char mesures [100];`