Programmation C++

Master GBM

Introduction

Les langages de programmation

- Le C++ est un langage de programmation : il sert donc à écrire des applications informatiques.

- Il s'agit d'ailleurs d'un des langages de programmation les plus utilisés aujourd'hui.

Un langage compilé

- Le C++ est un langage compilé : pour écrire un tel programme, il faut commencer par écrire un ou plusieurs fichiers source.

- Ensuite, il faut compiler ces fichiers source grâce à un programme appelé compilateur afin d'obtenir un programme exécutable.

- Les fichiers source sont des fichiers texte lisibles dont le nom se termine en général par .c, .cpp ou .h.

- Les fichiers exécutables portent en général l'extension .exe sous windows et ne portent pas d'extension sous Linux.

Introduction

Le C et le C++

Le langage C est un langage de programmation inventé par MM. Kernighan et Ritchie au début des années 70. Au début des années 90, Bjarne Stroustrup fait évoluer le langage vers le langage C++ en lui rajoutant notamment les notions orientées objet. Toutefois, bien que le C++ ait évolué à partir du C, et ait gardé un grand nombre de notions et de syntaxes du C.

Les compilateurs

Il existe de très nombreux compilateurs : on peut citer par exemple Visual C++ (de microsoft), C++ Builder (de Borland), ou encore gcc qui est un excellent compilateur libre.

Les GUI (Graphical User Interface) ou EDI (Environnements de Développement Intégrés)

On programme très souvent en utilisant un environnement de développement intégré : il s'agit d'un ensemble complet d'outils permettant d'éditer et de modifier des fichiers sources, de les compiler, de lancer l'exécutable, de "déboguer" le programme, etc... Visual C++ (version express disponible gratuitement), C++ Builder, Dev-cpp (disponible gratuitement et basé sur gcc) et Code::Blocks sont des environnements de développement intégrés.

Introduction

Compilateur seul :

- sous linux et Max, en général un compilateur comme GCC est déjà installé par défaut

- MinGW : pour installer GCC sous windows
  http://sourceforge.net/projects/mingw/files/MinGW/

Environnements de développement :

- Microsoft Visual Studio Express (spécifique à Windows)
- Digital Mars (spécifique à Windows)
- Xcode (spécifique Mac=> apple store)

- Dev-cpp (https://sourceforge.net/projects/orwelldevcpp)
- Code::Blocks (www.codeblocks.org - installer la version incluant MinGW de préférence)
- Scite (scintilla.org) (+ GCC à installer indépendamment)

Environnements online (en général assez basiques et nécessites une connexion à internet) :

- http://cpp.sh/
- https://www.onlinegdb.com/online_c++_compiler

Un premier exemple

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

int main()
{
   cout << "BONJOUR";
   return 0;
}
```

La directive #include

On place en général au début du programme un certain nombre d'instructions commençant par #include. Cette instruction permet d'inclure dans un programme la définition de certains objets, types ou fonctions. Le nom du fichier peut être soit à l'intérieur des < et >, soit entre guillemets :

• #include <nom_fichier> Inclut le fichier nom_fichier en le cherchant d'abord dans les chemins configurés, puis dans le même répertoire que le fichier source,

• #include "nom_fichier" Inclut le fichier nom_fichier en le cherchant d'abord dans le même répertoire que le fichier source, puis dans les chemins configurés.

Un premier exemple

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

int main()
{
   cout << "BONJOUR";
   return 0;
}
```

Le fichier iostream

Le fichier iostream contient un certain nombre de définitions d'objets intervenant dans les entrées/sorties du programme, c'est-à-dire dans l'affichage à l'écran ou dans des fichiers. La définition de cout se trouve dans ce fichier; pour utiliser cout dans notre programme
Ce fichier est fourni par l'éditeur du compilateur : il s'agit d'un fichier C++ standard.

using namespace std;

- Cette ligne indique l'utilisation de l'espace de nommage std. Un espace de nommage est un ensemble de classes dont cout fait partie. Étant donné que nous voulons utiliser l'objet cout, nous indiquons que l'on utilisera, par défaut, l'espace de nommage std.

- Dès que l'on veut utiliser cin ou cout, on doit écrire cette directive.

- Remarque par rapport au C : les fichiers d'en-tête standard ne sont plus nommés avec une extension .h (comme iostream.h).

Un premier exemple

cout

Il s'agit du flux de sortie du programme (Console Output : sortie console). Ce flux de sortie est envoyé par défaut vers l'écran. Il va nous servir à afficher des messages à l'écran en utilisant l'opérateur <<. Cet opérateur à la forme d'une flèche semblant indiquer le sens de transfert des données (écriture vers la console).

Exemple : `cout<<"BONJOUR";`

Cette instruction affiche BONJOUR à l'écran.

Un autre exemple :

`cout<<endl;`

Lorsqu'on envoie endl (End of Line : fin de la ligne) vers l'affichage, on passe à la ligne suivante.

Un premier exemple

Il faut également connaître une écriture plus condensée. Au lieu d'écrire en 3 instructions :

```
cout << "BONJOUR";
cout << endl;
cout << "AU REVOIR";
```

On peut écrire en une seule instruction :

```
cout << "BONJOUR" << endl << "AU REVOIR";
```

Cependant, sur certaines implémentation, cette instruction condensée ne compile pas correctement, car l'implémentation du symbole endl ne permet pas d'utiliser l'opérateur << par la suite :

```
cout << "BONJOUR" << endl;
cout << "AU REVOIR";
```

Un premier exemple

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

int main()
{
   cout << "BONJOUR";
   return 0;
}
```

La fonction main()

Notre programme contient une fonction appelée "main" C'est à cet endroit que va commencer l'exécution du programme. Exécuter un programme en C++, c'est exécuter la fonction main de ce programme.
Tout programme en C++ doit donc comporter une fonction main.

Retour de la fonction

La dernière instruction de notre programme est return 0; Elle indique seulement que la fonction main s'est terminée correctement sans erreur particulière.

Remarque

L'ajout de l'instruction system("PAUSE"); sera parfois nécessaire pour que le programme ne s'arrête pas immédiatement après s'être ouvert. Cette instruction doit être ajouté avant l'instruction return 0;

Les variables

Déclaration des variables

- Une variable est un certain endroit en mémoire permettant de stocker une valeur.

- En C++, les variables sont obligatoirement typées

- Le type de la variable indique la nature des données que va contenir cette variable : un entier, un réel, un caractère,…

- Avant d'utiliser une variable, il faut la déclarer, c'est-à-dire fournir son nom et son type.

- Le nom de la variable s'appelle l'identificateur.

Syntaxe de la déclaration : type identificateur ;

Exemple : int a;

Cette déclaration déclare une variable a de type int.

Intérêt de la déclaration

- La déclaration des variables permet au programmeur d'indiquer la nature des données qui vont être stockées dans ces variables.

- La déclaration permet d'éviter des erreurs (bugs).

Les variables

Identificateurs valides

- Un identificateur est constitué d'une suite des lettres, de chiffres et _ (underscore).

- Un identificateur ne peut pas commencer par un chiffre. Il ne peut pas contenir d'espaces, ni contenir le caractères - (tiret) qui serait interprété comme un moins.

- Il doit être explicite c'est-à-dire qu'il doit être en rapport avec ce que contient la variable. Si une variable contient le prix d'un kg de tomates, on va appeler notre identificateur prix_tomate par exemple.

Le type int

- Il s'agit d'un type de base prédéfini dans le langage C++. Il permet de manipuler des entiers positifs ou négatifs. En général sur 32 bits, on peut représenter tous les entiers de -2^31 à 2^31-1.

- Attention, le nombre de bits et le système de représentation des données n'est pas déterminée en C++, ce qui pose des problèmes de portabilité des programmes

- La manipulation des entiers est exacte sans erreur de calcul !

L'affectation

- stock une valeur (ou le résultat d'une opération/expression) dans une variable

Syntaxe : identificateur= expression ;

- On commence par évaluer l’expression. On met le résultat dans la variable identificateur. L'écriture d'une valeur dans une variable remplace la valeur précédente qui est "écrasée".

- Il doit y avoir une correspondance des types.

Les variables

Utilisation d'une variable de type entier

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

int main()
{
    int a;
    a = 80 + 20;
    cout << "La valeur de a est : " << a << endl;
    return 0;
}
```

Dans cet exemple, nous déclarons une variable entière a grâce à la déclaration int a;

Nous utilisons ensuite l'affectation pour mettre dans a le résultat de l'expression 80+20, c'est-à-dire 100.

Nous affichons alors la valeur de a grâce à cout

Les variables

Incrémentation d'une variable

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

int main()
{
    int a;
    a = 80;
    a = a + 1;
    cout << "La valeur de a est : " << a << endl;
    return 0;
}
```

a vaut 80 avant l'exécution de cette instruction. a vaut 81 après cette exécution.

A la place de a=a+1; , on peut également écrire a++;

Les variables

Utilisation de plusieurs variables

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

int main()
{
    int a, bb = 9, c80;

    a = 80;
    a++;
    bb = bb + a;
    c80 = bb - 10;
    cout << "La valeur de a est : " << a << endl;
    cout << "La valeur de bb est : " << bb << endl;
    cout << "La valeur de c80 est : " << c80 << endl;

    return 0;
}
```

- Déclaration dans un premier temps de 3 variables a,bb et c80.
- Puis différentes affectations sur ces variables.
- Puis affichage du contenu final de ces variables.

Les variables

Opérations arithmétiques

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

int main()
{
    int a = 10, b = 20, c, d, e, f;
    c = a + b;
    d = a * c;
    d = d - 80;
    e = d / 7;
    f = e % 4;

    cout << "La valeur de f est : " << f << endl;

    return 0;
}
```

Les variables

Utilisation de cin

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

int main()
{
    int a;
    cout << "Tapez la valeur de a : ";
    cin >> a;
    a = a + 10;
    cout << "La valeur de a est : " << a << endl;

    return 0;
}
```

Les variables

Le type double

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

int main()
{
    double a, b, moy;

    cout << "Tapez une valeur réelle : ";
    cin >> a;
    cout << "Tapez une valeur réelle : ";
    cin >> b;

    moy = (a + b) / 2;

    cout << "la moyenne des 2 réels est : " << moy <<endl;

    return 0;
}
```

- Le type double est un type prédéfini du langage C++. Il stocker un réel, en général sur 64bits.

- Taille et système de représentation n'est pas imposé par le langage. Chaque opération peut faire d'une petite erreur de calcul. La propagation de cette erreur peut devenir problématique

Les variables

Compatibilité int-double

On peut sans problème copier un int dans un double :
int a;
double b;
b=a;

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

int main()
{
    int a;
    double b;
    cout << "Tapez une valeur entière : ";
    cin >> a;

    b = a;

    cout << "La valeur de b vaut : " << b <<endl;

    return 0;
}
```

Pour mettre un double dans un int, il faut utiliser ce qu'on appelle un cast : on demande explicitement au compilateur de transformer le double en int et il y a alors troncature du double :

int a;

double b;

a=(int)b;

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

int main()
{
    double a;
    int b;

    cout << "Tapez une valeur réelle : ";
    cin >> a;

    b = (int)a;

    cout << "La valeur de b vaut : " << b <<endl;

    return 0;
}
```

Les commentaires

```
/*********************************************
  PROGRAMME TEST, AUTEUR : MOI
*********************************************/

#include <iostream>
using namespace std;

int main()
{
    double a; int b;

    // Saisie de la variable a
    cout << "Tapez une valeur réelle : ";
    cin >> a;

    // On met a dans un entier
    b = (int)a;

    /* Affichage */
    cout << "La valeur de b vaut : " << b <<endl;

    return 0;
}
```

- Le compilateur ne tient pas compte de tout ce qui est en commentaire

- Il est recommandé d'inclure dans tout programme des commentaires permettant de rendre le programme plus facilement compréhensible

- Un programme doit être compréhensible par un autre programmeur ou par vous même si vous le modifier plusieurs mois après par exemple

Les commentaires se présentent sous 2 formes:

Commentaires sur plusieurs lignes commençant par /* et finissant par */.

```
/*kkkkkkk JJJJJJJJJJJJJ*/
```

Commentaires sur une seule ligne commençant par //

```
// kkkkkkkkkkkkkkkkk
```

=> voir TD

Les structures conditionnelles

Le if

Cette structure de contrôle permet d'exécuter une instruction ou une suite d'instructions seulement si une condition est vraie.

Syntaxe :

if (condition) instruction;

On évalue la condition :

si elle est vraie on exécute l'instruction et on passe à l'instruction suivante,

si elle est fausse on passe directement à l'instruction suivante.

L'instruction peut être remplacée par une suite d 'instructions entre accolades

Les conditions

Les conditions habituelles sont utilisables :

if (a>b)… strictement supérieur à

if (a>=b)… supérieur ou égal à

if (a<b)… strictement inférieur à

if (a<=b)… inférieur ou égal à

if (a==b)… test d'égalité

if (a!=b)… différent de

Les structures conditionnelles

Exemple d'utilisation du if

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

int main()
{
  int a;
  cout << "Tapez la valeur de a : ";
  cin >> a;
  if (a > 10) cout << "Gagné !" << endl;
  cout << "Le programme est fini" << endl;
  return 0;
}
```

Les structures conditionnelles

Le ET logique

Syntaxe : condition1 && condition2

VRAI OU VRAI = VRAI
VRAI OU FAUX = FAUX
FAUX OU VRAI = FAUX
FAUX OU FAUX = FAUX

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

int main()
{
int a;
cout<<"Tapez une valeur entière : ";
cin>>a;
if(a>10 && a<20)cout<<"GAGNÉ"<<endl;
else cout<<"PERDU"<<endl;
return 0;
}
```

Le OU logique

Syntaxe : condition1 || condition2

VRAI OU VRAI = VRAI
VRAI OU FAUX = VRAI
FAUX OU VRAI = VRAI
FAUX OU FAUX = FAUX

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

int main()
{
int a;
cout<<"Tapez une valeur entière : ";
cin>>a;
if( a<3 || a>20 )cout<<"GAGNÉ"<<endl;
else cout<<"PERDU"<<endl;
return 0;
}
```

Les structures conditionnelles

Le NON logique

```
#include <iostream>
using namespace std;
int main()
{
int a;
cout<<"Tapez une valeur entière : ";
cin>>a;
if(!(a<3 || a>20))cout<<"GAGNÉ"<<endl;
else cout<<"PERDU"<<endl;
return 0;
}
```

=> Inversion de la condition

Les structures conditionnelles

Plusieurs instructions (=bloc) dans un if … else

```
#include <iostream>
using namespace std;
int main()
{
int a;
cout<<"Tapez une valeur entière : ";
cin>>a;
if(a!=10){cout<<"GAGNÉ"<<endl;a=a+1;}
else {cout<<"PERDU"<<endl;a=a-1;}
cout<<"La valeur finale de a vaut "<<a<<endl;
return 0;
}
```

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

int main() {
	int a;
	cout<<"Tapez une valeur entière : ";
	cin>>a;
	if(a!=10) {
		cout<<"GAGNÉ"<<endl;a=a+1;}
	else {
		cout<<"PERDU"<<endl;a=a-1;}
	cout<<"La valeur finale de a vaut "<<a<<endl;
	return 0;
}
```

Exemple d'une mise en forme différente
Pas d'influence de l'indentation

Les structures conditionnelles

Utilisation du type bool (=booléen)

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

int main()
{
int a; bool c,d;
cout<<"Tapez une valeur entière : ";
cin>>a;
c=(a<3);
d=(a>20);
if(c||d)cout<<"GAGNÉ"<<endl;
else cout<<"PERDU"<<endl;
return 0;
}
```

Les structures conditionnelles

Le switch

Syntaxe :

```
switch(identificateur)
{
case c1:instruction1;break;
case c2:instruction2;break;
case c3:instruction3;break;
...
default: instruction;break;
}
```

- On veut teste la variable définie par l'identificateur.
- On la compare successivement aux constantes c1, c2, c3,…etc… Si la variable vaut c1 alors on exécute l'instruction1 et on passe à l'instruction suivante. Si elle vaut c2, on exécute l'instruction2 et on passe à l'instruction suivante. Idem s'il vaut c3. Si elle ne vaut ni c1, ni c2, ni c3 alors on exécute l'instruction après default et on passe à l'instruction suivante. Le default est facultatif.
- On peut remplacer les instructions instruction1, instruction2, instruction3 par des suites d'instructions sans mettre d'accolades. Les valeurs c1, c2,c3 .. sont obligatoirement des valeurs ou des constantes

Les structures conditionnelles

Le switch

```
#include <iostream>
using namespace std;
int main()
{
int i;
cout<<"Tapez un entier entre 1 et 3 bornes incluses :";
cin>>i;
switch(i)
{
case 1: cout<<"GAGNÉ"<<endl;
        i=i+99;
        break;
case 2: cout<<"PERDU n° 2"<<endl;
        i=0;
        break;
case 3: cout<<"PERDU n°3"<<endl;
        i=0;
        break;
}
cout<<"La valeur finale de i est "<<i<<endl;
return 0;
}
```

=> voir TD

```
#include <iostream>
using namespace std;
int main()
{
int i;
cout<<"Tapez un entier entre 1 et 3 :";
cin>>i;
switch(i)
{
case 1:
   cout<<"GAGNÉ"<<endl;
   i=i+99;
   break;

case 2 :
   cout<<"PERDU n° 2"<<endl;
   i=0;
   break;

case 3 :
   cout<<"PERDU n°3"<<endl;
   i=0;
   break;

default :
   cout<<"J'ai dit entre 1 et 3 !!!"<<endl;
   i=-1000;
   break;
}
cout<<"La valeur finale de i est "<<i<<endl;
return 0;
}
```

Les boucles

Le for

Le for est une structure de contrôle qui permet de répéter un certain nombre de fois une partie d'un programme.

Syntaxe :
for(instruction1 ; condition ; instruction2)
    bloc d'instructions ;

- on exécute l'instruction1
- on teste la condition :
- si elle est vraie, on exécute le bloc d'instructions, puis l'instrution2 puis on revient au début.
- si elle est fausse on passe sort de la boucle..

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

int main()
{
    int i;
    for (i=0; i<10; i=i+1)
        cout<<"La valeur de i est : "<<i<<endl;
    cout<<"La valeur finale de i est : "<<i<<endl;
    return 0;
}
```

Note : on peut remplacer i=i+1 par i++

Les boucles

Le for

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

int main()
{
    int i;
    for(i=10; i>3; i--)
        cout<<"La valeur de i est : "<<i<<endl;
    cout<<"La valeur finale de i est : "<<i<<endl;
    return 0;
}
```

for décroissant (avec i--)

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

int main()
{
    int i;
    for (i=10; i<20; i=i+2)
        cout<<"La valeur de i est : "<<i<<endl;
    cout<<"La valeur finale de i est : "<<i<<endl;

    return 0;
}
```

for de 2 en 2

Note : attention au boucles infinies (si la condition du for n'est jamais atteinte par exemple)

Les boucles

Le while

Syntaxe : while (condition) instruction(s);

- On teste la condition :
- si elle est vraie, on exécute l'instruction puis on recommence
- si elle est fausse, on passe à l'instruction suivante.

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

int main()
{
    int i=0;
    while(i<10)
    {
        cout<<"La valeur de i vaut : "<<i<<endl;
        i++;
    }
    cout<<"La valeur finale de i vaut : "<<i<<endl;
    return 0;
}
```

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

int main()
{
    int i;
    cout <<"Tapez une valeur entre 0 et 20 bornes
incluses : ";
    cin>>i;
    while (i<0 || i>20)
    {
        cout <<"ERREUR ! ";
        cout <<"Tapez une valeur entre 0 et 20 bornes
incluses : ";
        cin >> i;
    }
    return 0;
}
```

Les boucles

Le do … while

Syntaxe :

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

- on exécute l'instruction ou le boc d'instructions.
- on évalue la condition.
- si elle est vraie, on recommence au début.
- si elle est fausse, on passe à la suite

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

int main()
{
    int i=0;
    do {
        cout<<"La valeur de i vaut : "<<i<<endl;
        i=i+1;
    } while (i<10);
    cout<<"La valeur finale de i est "<<i<<endl;
    return 0;
}
```

Les types de base

Les types de bases sont les types prédéfinis existants dans le langage C++

Les types int et double
Les type int et double servent à stocker respectivement un entier et un réel.

Le type char
- Il permet de stocker un caractère. Il est codé en général sur 8 bits (un octet donc).
- On peut faire des tests sur le caractère :
if(c>='A' && c<='Z') permet par exemple de tester si un caractère est une lettre majuscule.
On peut transférer un char dans un int. On récupère alors le code ASCII du caractère.
- l'assignation de la valeur se fait avec des « ou des '

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

int main()
{
  char c; int i;

  do {
     cout<<"Saisir une lettre : "; cin>>c;
     i=c;
     cout <<"Le caractère ASCII de "<<c<<" est : "<<i<<endl;
  } while (c!='Q');

  cout << "Pressez une touche pour continuer ..." << endl;
  cin.ignore();
  cin.get();

  return 0;
}
```

Les types de base

Comparaison sur les char

Lorsqu'on effectue des comparaisons sur les char, on compare en fait les codes ascii.
On peut tester si un caractère c est une majuscule par le test (x>='A' && x<='Z').
On peut tester si un caractère c est une minuscule par le test (x>='a' && x<='z').
On peut tester si un caractère c est un chiffre par le test (x>='0' && x<='9').

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

int main()
{
  char a;
  cout<<"Tapez un caractere : "; cin>>a;

  if (a>='A' && a<='Z') cout<<"Vous avez tapé une majuscule."<<endl;
  else if (a>='a' && a<='z') cout<<"Vous avez tapé une minuscule."<<endl;
  else if (a>='0' && a<='9') cout<<"Vous avez tapé un chiffre."<<endl;
  else cout<<"Vous n'avez tapé ni une majuscule, ni une minuscule, ni un chiffre."<<endl;

  return 0;
}
```

Les types de base

Transformation char-int

On peut effectuer des opérations de base sur les char : addition et soustraction.
Ces opérations sont en fait réalisée sur les codes ascii.
Si c est un chiffre c-'0' est la valeur de l'entier correspondant à ce chiffre.

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

int main()
{
  char a;
  int x;
  cout <<"Tapez un caractere : "; cin>>a;

  if (a>='0' && a<='9')
  {
    cout <<"Vous avez tapé un chiffre." <<endl;
    x = a-'0';
    cout <<"Ce chiffre est : "<< x <<endl;
  }
  else cout <<"Vous n'avez pas tapé un chiffre." <<endl;
  return 0;
}
```

Les types de base

Transformation majuscule-minuscule

Pour transformer un caractère c qui est une majuscule en la minuscule correspondante, il suffit de lui ajouter ('a'-'A').
Pour transformer un caractère c qui est une minuscule en la majuscule correspondante, il suffit de lui ajouter ('A'-'a').

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

int main()
{
  char a, b;
  cout<<"Tapez un caractere : "; cin>>a;
  if (a>='A' && a<='Z') {
    cout<<"Vous avez tapé une majuscule."<<endl;
    b = a + ('a'-'A');
    cout<<"La minuscule correspondante est "<< b <<endl;
  }
  else if (a>='a' && a<='z') {
    cout<<"Vous avez tapé une minuscule."<<endl;
    b = a + ('A'-'a');
    cout<<"La majuscule correspondante est "<< b <<endl;
  }
  else cout<<"Vous n'avez pas tapé une lettre."<<endl;
  return 0;
}
```

=> voir TD

Les tableaux de char

string ou tableaux de char

- plusieurs façons de représenter les chaînes de caractères :

* utiliser la classe prédéfinie string
* utiliser des tableaux de char (= chaîne de caractères de style C).

- Une telle chaîne de caractères est contenue dans un tableau de char.

- Chaque caractère sera dans une case du tableau.

- A la fin d'une chaîne de caractères (qui n'est pas forcément à la dernière case du tableau) doit se trouver le caractère spécial noté '\0' qui indique la fin de la chaîne.

Affichage et saisie d'une chaîne

- On peut afficher une chaîne de caractères par cout : le tableau de caractères sera alors affiché jusqu'au caractère de fin de chaîne.

- On peut saisir une chaîne par cin : le caractère de fin de chaîne est alors rajouté automatiquement.

- On peut accéder au caractère numéro i d'une chaîne t en indexant le tableau t[i].

Les tableaux de char

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

int main()
{
    char tt[20];
    tt[0] = 'B';
    tt[1] = 'O';
    tt[2] = 'N';
    tt[3] = 'J';
    tt[4] = 'O';
    tt[5] = 'U';
    tt[6] = 'R';
    tt[7] = '\0';
    cout << tt;
    return 0;
}
```

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

int main()
{
    char tt[20];
    cout << "Tapez une chaîne SVP : ";
    cin >> tt;
    cout << "Vous avez tapé la chaîne : ";
    cout << tt << endl;
    return 0;
}
```

Les tableaux de char

Comparaison de chaînes

Les chaînes de caractères sont contenues dans des tableaux de char, il existe de nombreuses fonctions prédéfinies dans le fichier cstring qui permettent de manipuler simplement ces tableaux.

On ne teste pas l'égalité de 2 chaînes par ==, on utilise strcmp(chaine1,chaine2) qui renvoie :
0 si les 2 chaînes sont égales.
un nombre <0 si chaîne1 est avant chaîne2 dans l'ordre lexicographique.
un nombre >0 si chaîne1 est après chaîne2 dans l'ordre lexicographique.
Pour utiliser strcmp, il faut inclure cstring.

```
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
    char tt[20];
    cout << "Tapez une chaîne : "; cin >> tt;
    if(strcmp(tt, "BONJOUR")==0)
        cout<<"GAGNE"<<endl;
    else cout<<"PERDU"<<endl;
    return 0;
}
```

Les tableaux de char

```
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
    char a[20];
    cout << "Tapez une chaîne : "; cin >> a;
    if (strcmp(a,"BONJOUR")>0)
        cout << a << " est après BONJOUR" << endl;
    else
        cout << a << " est avant BONJOUR" << endl;
    return 0;
}
```

```
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
    char a[20], b[20];
    strcpy(a, "BONJOUR");
    strcpy(b, a);
    cout << "La chaîne b vaut : " << b << endl;
    return 0;
}
```

Les tableaux de char

Longueur d'une chaîne

- La fonction strlen permet de connaître le nombre de caractères d'une chaîne.
- On ne tient pas compte du caractère de fin de chaîne lorsqu'on compte les caractères.

```
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
    char a[20];
    int b;
    cout << "Tapez une chaîne : "; cin >> a;
    b = strlen(a);
    cout << "Taille de la chaine = " << b << endl;

    return 0;
}
```

```
#include <iostream>

using namespace std;

int main()
{
    char tt[10];
    cout << "Tapez une chaîne SVP : ";
cin.getline(tt,10);
    cout << "Vous avez tapé la chaîne : ";
    cout << tt << endl;
    return 0;
}
```

Les tableaux de char

```
#include <iostream>

using namespace std;

bool valide(char t[])
{
bool r=true;
int h,m;
    if( t[0]<'0' ||t[0]>'9')
    r=false;
    else if( t[1]<'0' ||t[1]>'9')
        r=false;
        else if( t[3]<'0' ||t[3]>'9')
            r=false;
            else if( t[4]<'0' ||t[4]>'9')
                r=false;
                else if(t[2]!='h')
                    r=false;
                    else if(t[5]!='\0')
                        r=false;
                        else { h=(t[0]-'0')*10+(t[1]-'0');
      if(h>23) r=false; m=(t[3]-'0')*10+(t[4]-'0');
      if(m>59) r=false; }

return r;
}
```

```
int main()
{
    char a[20];
    do {
        cout << "Tapez une heure sous le
format ..h.. : ";
        cin >> a;
    } while (!valide(a));
    return 0;
}
```

Les tableaux statiques

Syntaxe : type identificateur[taille];

Un tableau sera constitué d'un ensemble de cases (=éléments). Chaque case comportera une valeur dont le type sera type. Le nom du tableau sera identificateur. Le nombre total de cases du tableau sera taille. Cette variable sera obligatoirement une constante.
Le premier indice (première case) est 0, donc le dernier est à « longueur du tableau -1 »

- On peut déclarer et initialiser un tableau de la manière suivante : int t[]={8,7,6,4,8};
Cette syntaxe évite d'écrire : int t[5]; t[0]=8; t[1]=7; t[2]=6; t[3]=4; t[4]=8;

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

int main()
{
    double t[4];
    int i;
    for(i=0; i<4; i++)
        t[i] = 1.0 / (i+1);
    for(i=0; i<4; i++) cout<<"La valeur numéro "<<i<<"
est : " << t[i] <<endl;
    return 0;
}
```

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

int main()          // calcul de moyenne
{
    int t[4], i;
    double s=0;
    for(i=0; i<4; i++)
    {
        cout << "Tapez la valeur numéro " << i << " : ";
        cin >> t[i];
       s = s + t[i];
    }
    s = s/4;
    cout << "La moyenne est : " << s << endl;
    return 0;
}
```

Les tableaux statiques

```
#include <iostream>
using namespace std;
const int N = 2;
const int M = 3;

int main()
{
    int i, j;
    int t[N][M];

    for(i=0; i<N; i++)
      for(j=0; j<M; j++)
      {
        cout<<"Tapez t["<< i <<"]["<< j <<"] :";
        cin >> t[i][j];
      }

    cout<<"Voici le tableau :"<<endl;
    for(i=0; i<N; i++)
    {
      for(j=0; j<M; j++) cout<< t[i][j] <<" ";
      cout<<endl;
    }
    return 0;
}
```

Tableau à 2 dimensions

Les fonctions

Syntaxe : type identificateur(paramètres) { Corps de la fonction }

- A chaque appel de la fonction on exécute le corps de la fonction.
- L'identificateur est le nom de la fonction.
- La fonction peut avoir des paramètres.
- La fonction peut renvoyer une valeur de type type.

- Lors de l'appel de la fonction, le programme exécute la totalité des instructions du corps de la fonction, puis reprend le programme juste après l'appel de la fonction.

- notions de « variables locales »

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

void b(int i)   // ← une procédure !
{
    int j;
    for(j=0; j<i; j++) cout<<"Bonjour"<<endl;
}

int main()  // ← une fonction
{
    cout<<"COUCOU1"<<endl;
    b(2);
    cout<<"COUCOU2"<<endl;
    b(3);
    return 0;
}
```

Les fonctions

Avec renvoi de valeur :
return variables(s)/valeur(s)

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

int b(int i, int j)
{
    int k;
    k = i*i + j*j;
    return k;
}

int main()
{
    int a;
    a = b(3,4);
    cout<<"Le resultat vaut : "<<a<<endl;
    return 0;
}
```

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

const int n=4;

void saisir(int t[n])
{
    int i;
    for(i=0; i<n; i++)
    {
        cout<<"Tapez la valeur numero "<<i<<" : ";
        cin >> t[i];
    }
}

void affiche(int t[n])
{
    int i;
    for(i=0; i<n; i++) cout<<"La valeur numero "<<i<<" est :
"<<t[i]<<endl;
}

int main()
{
    int a[n];
    saisir(a);
    affiche(a);
    return 0;
}
```

Les fonctions

Passage de paramètres par référence (et pas par copie)

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


void minmax (int i, int j, int & min, int & max)
{
    if(i<j) {min=i; max=j;} else {min=j; max=i;};
}

int main()
{
    int a,b,w, x;

    cout << "Tapez la valeur de a : "; cin >> a;
    cout << "Tapez la valeur de b : "; cin >> b;
    minmax(a,b,w,x);
    cout << "Le plus petit vaut : " << w << endl;
    cout << "Le plus grand vaut : " << x << endl;
    return 0;
}
```

Notions d'objets / la classe « string »

- « Classe » qui simplifie l'utilisation des chaînes de caractères
→ elle « encapsule » les données et les fonctions sur les chaînes (l'utilisateur n'a pas besoin de savoir comment marchent les fonctions, mais doit juste savoir comment les utiliser)

- Pour utiliser cette classe, il faut rajouter : #include <string>

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

int main () {

        string s1, s2, s3;
        cout << "Tapez une chaine : "; cin >> s1;
        cout << "Tapez une chaine : "; cin >> s2;
        s3 = s1 + s2;
        cout << "Voici la concatenation des 2 chaines :" << endl;
        cout << s3 << endl;
        return 0;

      }
```

Notions d'objets / la classe « string »

- dans le dernier exemple il n'est pas possible de mettre un espace car il est considéré comme un séparateur (donc 2 arguments pour « cin »).
=> il faut utiliser la fonction getline() : getline(iostream, string)

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

int main ()

{
string s1, s2, s3;
cout << "Tapez une chaine : "; getline (cin, s1);
cout << "Tapez une chaine : "; getline (cin, s2);
s3 = s1 + s2;
cout << "Voici la concatenation des 2 chaines :" << endl;
cout << s3 << endl;
return 0;

}
```

Notions d'objets / la classe « string »

size() :
Renvoie le nombre de caractères d'une chaîne (sa longueur)

chaine.at(int i) :
Renvoie le caractère de la variable chaine (de type string) à la position i (de type entier). *Commence à l'indice 0*

```
#include <iostream>
#include<string>

using namespace std;
int main (void)
{
string s= "BONJOUR";
int i, taille = s.size ();
cout << "La chaine comporte " << taille << " caracteres." << endl;
for (i = 0 ; i < taille ; i++) {
        cout << "caractère " << i << " = " << s.at(i) << endl; }
return 0;
}
```

Notions d'objets / la classe « string »

Transformation string <=> char :

- char => string : affectation d'un char dans une string

- string => char : utilisation de la fonction (=méthode) c_str()

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

int main (void) {

string s1, s2;

char c1 []= "BONJOUR";
char c2 [];

s1 = c1;
cout << s1 << endl;
s2 = "AU REVOIR";
c2 = s2.c_str();
cout << c2 << endl;

return 0;

}
```

=> voir TD

Manipulation des fichiers

Généralités

- Pour créer/modifier un fichier :
    * il faut l'ouvrir en écriture.
    * on écrit des données dans le fichier.
    * on ferme le fichier.
- Pour lire des données écrites dans un fichier :
    * on l'ouvre en lecture.
    * on lit les données en provenance du fichier.
    * on ferme le fichier.

Fichiers textes et binaires

- les fichiers textes qui sont des fichiers lisibles par un simple éditeur de texte (codés en ASCII)
- Le reste, les fichiers codés en binaires.

2 bibliothèques pour manipuler les fichiers en C++

- 'cstdio' qui provient du C (utilisable en C++). il faut inclure <cstdio>

- 'fstream' qui est spécifique du C++. Il faut inclure <fstream>

Manipulation des fichiers

- Inclure **<fstream>**
- Puis pour les opérations de lectures la classe '**ifstream**' (Input File stream)
- Pour les opérations d'écriture '**ofstream**' (Output File stream)
- ou pour lire et écrire à la fois '**fstream**'

ofstream monFichier; *//monFichier est un type particulier, un flux*
monFichier.open("c:\\essai.txt");

ou

ifstream monFichier;
monFichier.open("c:\\essai.txt");

- Ou peut aussi créer le flux directement sans open :
ofstream MonFichier("c:\\essai.txt");

- Pour fermer un fichier ***MonFichier.close()***

- Pour tester si le flux est ok (fichier bien ouvert) → le flux != 0

ifstream fichier("test.txt");
if(! fichier) { cout << « Erreur d'ouverture ou fichier introuvable »; }

Manipulation des fichiers

Lors de la création du flux, on peut ajouter un 2eme argument : le mode d'ouverture

ifstream fichier("test.txt", ios::in); // « ios:: » est la classe 'ios'

ios::in (pour input) : ouvre le fichier en lecture.
Par défaut quand on utilise un objet ifstream. Ecrase les données du fichier par les nouvelles

ios::out (pour output) : ouvre le fichier en écriture.
Par défaut quand on utilise un objet ofstream. Ecrase les données du fichier par les nouvelles

ios::app (pour append = ajouter à la suite) : ouvre le fichier en écriture et écrit les nouvelles données à la fin du fichier sans effacer le contenu, s'il y en a un.

ios::trunc (pour truncate = tronquer) : ouvre le fichier en écriture et efface l'ancien si existe déja, sinon en crée un nouveau vide

ios::ate (pour at end) : ouvre le fichier en écriture et positionne le curseur à la fin de celui-ci. La différence avec ios::app est que si on se repositionne dans le fichier, l'écriture ne se fera pas forcément à la fin du fichier, contrairement à ios::app.

On peut combiner ces options en les séparant avec un « **|** »

Manipulation des fichiers

Le flux du fichier est équivalent au flux de CIN ou COUT, donc pour lire/écrire dans le fichier :

```
int n = 4;
double x = 3.7;
char c = 'y';
ofstream monFichier("essai.txt");
if (monFichier) {    // l'ouverture a réussi, on peut donc écrire
  monFichier << n << '\t' << x << endl;
  monFichier << c << '\n';
  }
monFichier.close();
```

```
int entier;
double decimal;
char caractere;
ifstream leFichier("essai.txt");
If (leFichier)              // l'ouverture a réussi, on peut donc lire
  {
  leFichier >> entier;
  leFichier >> decimal;
  leFichier >> caractere;
  }
leFichier.close();
```

Manipulation des fichiers

Lecture

- getline(flux, chaineDeCaractères) : pour lire une ligne complète ;
- flux.get(caractère) : pour lire un caractère ;
- flux >> variable : pour récupérer à partir du fichier jusqu'à un délimiteur (espace, saut à la ligne, ...).

```
if (fichier)
{
	string ligne;
	while(getline(fichier, ligne))  // tant que l'on peut mettre la ligne dans "contenu"
	{
		cout << ligne << endl;  // on l'affiche
	}
}
```

Remarque : pour lire tout le contenu d'un fichier, utiliser par exemple une string vide, puis lire les lignes du fichier (comme au dessus) et les concater dans cette string. La string contiendra tout le fichier (pour les fichiers ascii uniquement).

Autres ...

- Les pointeurs

- Utilisation des classes / objets / héritages

- Les structures

- Les interfaces graphiques (.net, WxWidgets, Qt, Gtk, etc...)

=> voir TD