R101 - Amphi05 - C++

Alain Casali

Aix Marseille Univ

Langage de programmation

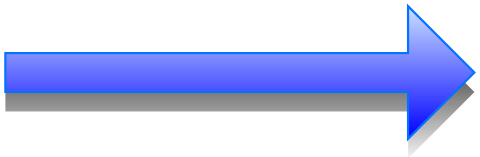
En informatique, un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est fait d'un alphabet, un vocabulaire, des règles de grammaire, et des significations.

Source : `http://fr.wikipedia.org/wiki/Langage_de_programmation`

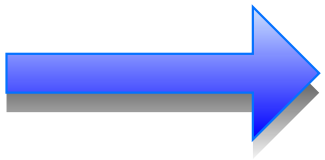
Fichiers

Executable

Problème du développeur (1)



Cest une roue !



On ne re-développe **JAMAIS** une fonctionnalité si elle est déjà fournie par le langage (sauf pour des raisons pédagogiques).

Problème du développeur (2)

Exemple : Trier un tableau d'entiers

- Il existe une trentaine d'algorithmes permettant de trier un tableau d'entiers avec des performances différentes.
- Un des algorithmes les plus efficaces est le tri rapide

Taille du tableau	Temps C++	Temps mon implémentation
1 000	$\sim$ 0s	$\sim$ 0s
10 000	$\sim$ 0s	$\sim$ 0s
100 000	$\sim$ 0s	$\sim$ 0s
1 000 000	$\sim$ 0s	$\sim$ 13s
10 000 000	$\sim$ 6s	$\sim$ 1200s

La documentation

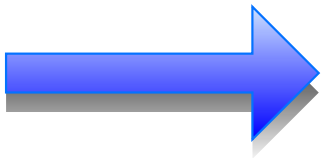
Elle se lit :

- En ligne (elle est à jour) ;
- En anglais (on évite les erreurs de traduction, si traduction il y a) ;

Elle comporte des exemples.

Objectifs

- Début janvier :
  - Être capable de lire une documentation en anglais ;
  - De la comprendre ;
  - D'adapter les exemples fournis à votre problématique.
- Fin du BUT : Être capable d'écrire une documentation technique en anglais ;



Tous les noms de fonctions, de variables, doivent être en anglais !

Le C++, cest quoi ?

1. Un langage de programmation ;
2. Une surcouche (partiellement) objet du C.

Dernière norme du C++ date de 2020.
Plusieurs compilateurs disponibles :

- g++ (support complet)
- clang (support complet)
- VS2017 (support partiel)

En C++ :

- Tout est appel de fonction ou de méthode (maintenant) ;
- Tout est flux ;
- Vive les références (maintenant) et les pointeurs (S3) ;

Un premier programme (1)

Quel que soit le langage de programmation utilisé, le premier programme est toujours d'afficher, soit :

- Sur un terminal (une console) ;
- Sur une page web ;
- Sur une interface graphique.

la chaine de caractères "`Hello world!`" (modulo quelques petits arrangements)

Un premier programme (2)

```
1  /**
2   *
3   * @file    Hello.cxx
4   *
5   * @author A. CAsali
6   *
7   * @date    12/09/2013
8   *
9  **/
10
11 #include <iostream>
12 using namespace std;
13
14 int main ()
15 {
16     cout << "Hello World!" << endl;
17     return 0;
18 }
```

Les commentaires

1. Commentaire sur une unique ligne : utilisation de //

```
//this is a one line remark
```

2. Commentaire sur plusieurs lignes : utilisation de /* */

```
/* this
   is a drawn out
   explanation
*/
```

3. Autre possibilité :

```
/*this is also a one line remark*/
```

Plus de détail : `http://www.stack.nl/~dimitri/doxygen/`

Déclaration de variables

Modèle général algorithmique :

```
declarer nomDeVariable : Type;
```

Modèle général C++ :

```
type varIdent; //varIdent (Variable Identifier)
```

Exemple :

```
int i;
```

Affectation

Modèle général algorithmique :

```
nomDeVariable <- valeur;
```

Modèle général C++ :

```
varIdent = value;
```

Exemple :

```
i = 10;
```

Cas des variables constantes

Modèle général algorithmique :

```
declarer KnomDeVariable : constante type <- valeur;
```

Modèle général C++ :

```
const type KvarIdent = value;
```

Exemple :

```
const int Ki = 10;
```

Tous les noms des constantes doivent commencer par la lettre 'K'

Déclaration et initialisation à la volée

Modèle général algorithmique :

```
declarer nomDeVariable : type <- Valeur;
```

Modèle général C++ :

1. `type varIdent = value;`
2. `type varIdent (value);`
3. `type varIdent {value(s)};`

Exemple :

```
int i = 10;
int j (5);
int k {3};
```

Seule la troisième forme permet la déclaration et initialisation à la volée des tableaux.

Bloc et portée de variable

Définition : bloc

Un bloc est une suite d'instructions entre { }

Propriété : portée de variable

Une variable n'existe que dans le bloc dans laquelle est déclarée.

Exemple :

```
{
    type i;
    i = value1;
    ...
    {
        type j (value2);
        //i et j existent
        i = value3;
        j = value4;
    }
    //seul i existe
    i = value5;
    j = value6; //<- erreur de compilation
}
```

Bloc externe

Bloc interne

Saisie clavier

Modèle général algorithmique :

```
saisir (nomDeVariable);
```

Modèle général C++ :

```
cin >> varIdent;
```

cin signifie console input
Le symbole » est appelé un extracteur.

Exemple :

```
int i;
cin >> i; //10
```

Prérequis pour utiliser le clavier :

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

Utilisation de la bibliothèque qui gère les flux (stream) dentrées / sorties (input / output)

Affichage écran

Modèle général algorithmique :

```
afficher (unLitteral);
afficher (nomDeVariable);
```

Modèle général C++ :

```
cout << literalPattern;
cout << varIdent;
```

cout signifie console output
Le symbole « est appelé un injecteur.

Exemple :

```
cout << 10;
cout << "une jolie chaine";
int i;
i = 10;
cout << i;
```

Littéral entier	10
Littéral chaine de caractères	une jolie chaine
Variable	10

Prérequis pour utiliser la console :

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

Utilisation de la bibliothèque qui gère les flux (stream) dentrées / sorties (input / output)

Passage à la ligne lors d'un affichage console

Modèle général algorithmique :

```
ligne_suivante;
```

Modèle général C++ :

```
cout << endl;
```

endl signifie end line
Le symbole `endl` est appelé un identificateur.
Exemple :

```
cout << i;
cout << endl;
```

10

Compression des sorties écran

Il est possible d'injecter plusieurs informations de type différents et donc de compacter l'écriture.

Exemple :

- ```
  cout << "Valeur de i : ";
  cout << i;
  cout << endl;
  ```

- ```
  cout << "Valeur de i : "
       << i
       << endl;
  ```

1. Absence du caractère ';' à la fin de la ligne ;
2. Tous les injecteurs sont alignés.

- ```
  cout << "Valeur de i  : " << i << endl;
  ```
  ```

Opérateur identité

Modèle général algorithmique :

```
nomDeVariable vaut Valeur
```

Modèle général C++ :

```
varIdent == value
```

Exemple :

```
i == 10
```

Ou mieux

```
10 == i
```

On ne compare que des variables et / ou des littéraux du même type.

Opérateur différence

Modèle général algorithmique :

```
nomDeVariable ne_vaut_pas Valeur
```

Modèle général C++ :

```
varIdent != value
```

Exemple :

```
i != 10
```

1 Avant Propos

2 Un premier programme

3 Les commentaires

4 Déclaration, affectation, bloc

5 Entrées / Sorties

6 Opérateur identité et différence

7 Schéma alternatif simple
- Schéma alternatif sans condition sinon
- Schéma alternatif avec condition sinon
- Schéma alternatif en cascade

8 Schéma alternatif complexe

9 Le type booléen

10 Les types entiers

11 Les types réels

Schéma alternatif sans condition sinon

Modèle général algorithmique :

```
si (condition)
    instruction1;
    instruction2;
    ...
fsi
```

Modèle général C++ :

```
if (condition)
{
    instruction1;
    instruction2;
    ...
}
```

- `condition` est une expression booléenne ;
- '{' marque le début d'un bloc d'instruction(s) ;
- '}' marque la fin de ce bloc.

Toutes les instructions à l'intérieur d'un même bloc sont alignées

Schéma alternatif avec condition sinon (1)

Modèle général algorithmique :

```
si (condition)
     instruction1;
     instruction2;
     ...
sinon
     instruction3;
     instruction4;
     ...
fsi
```

Modèle général C++ :

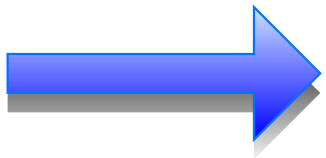
```
if (condition)
{
     instruction1;
     instruction2;
     ...
}
else
{
     instruction3;
     instruction4;
     ...
}
```

Schéma alternatif avec condition sinon (2)

Exemple :

```
verifierVie;
verifierArmes;

si (toutEstOK)
    attaquerBoss;
     ...
sinon
     prendrePotion;
     ...
fsi
partir;
```



```
checkLife ();
checkWeapons ();

if (everythingIsOK)
{
    attackBoss ();
     ...
}
else
{
     takePotion ();
     ...
}
leave ();
```

Schéma alternatif en cascade (1)

Modèle général algorithmique :

```
si (exprLog1)
    sequ1;
sinon_si (exprLog2)
    sequ2;
sinon_si (exprLog3)
    sequ3;
sinon //tous les autres cas
    sequ4;
fsi
```

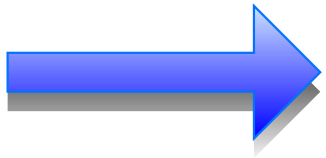
Modèle général C++ :

```
if (logExp1)
{
    sequ1;
}
else if (logExp2)
{
    sequ2;
}
else if (logExp3)
{
    sequ3;
}
else
{
    sequ4;
}
```

Schéma alternatif en cascade (2)

Exemple :

```
si (peuDeVie)
    prendreUnePotion;
sinon_si (jeSuisUnGuerrier)
    attaquerAvecEpee;
sinon_si (jeSuisUnMage)
    lancerUnSort;
sinon
    partir;
fsi
```



```
if (fewOfLife)
{
    takeAPotion ();
}
else if (IAmAWarrior)
{
    attackWithSword ();
}
else if (IAmAWizard)
{
    castASpell ();
}
else
{
     leave ();
}
```

Schéma alternatif complexe (1)

Condition logique en algo.	Equivalence en C++
ET	&
OU	|
ET_ALORS	&&
OU_SINON	||

Exemple :

```
si (beaucoupDeVie OU_SINON jeSuisUnGuerrier)
    Attaquer;
fsi
```

se traduit par :

```
if (fullOfLife || IAmAWarrior)
{
    attack ();
}
```

Schéma alternatif complexe (2)

Exemple :

```
si (peuDeVie ET_ALORS jeSuisUnGuerrier)
    attaquer;
fsi
```

se traduit par :

```
if (fewOfLife && IAmAWarrior)
{
    attack ();
}
```

- Les règles de propagation de l'opérateur de négation sont les mêmes qu'en algorithmique (amphi2 T11 -> T14) ;
- Les tableaux de vérité sont les mêmes qu'en algorithmique (amphi2 T15 & T16).

Identificateur

```
bool
```

Valeurs

```
true false
```

Opérations (opérateurs booléens)

Produisent un booleen

Voir transparents précédents

```
bool toBe;
bool notToBe;
bool question;
...

toBe     = false;
notToBe  = !toBe;            // true
question = toBe | notToBe; // always true!
```

Opérations (d'identité)

Produisent un booleen

== !=

Exemple :

```
if (toBe == notToBe)
{
    cout << "WTF ! ";
}
```

Opérateur de négation

Produit un booléen

Modèle général algorithmique :

```
si (NON condition)
    instruction1;
    instruction2;
    ...
fsi
```

Modèle général C++ :

```
if (!condition)
{
    instruction1;
    instruction2;
    ...
}
```

Identificateur

`int`

Valeurs

Sous-ensemble des entiers mathématiques

Opérations

Produisent un entier

`+ - * / %`

- Troncature du reste de la division ;
- L'opérateur modulo (%) renvoie le reste de la division entière.

Exemple :

```
int x;
cout << x;          0
x = 1;
cout << x;          1
```

```
x = (x + 2) / 4 * 3;        0
cout << x;
x = 1;
x = (x + 2) * 3 % 4;
cout << x;                  9
```

Opérations (de comparaison)

Produisent un booleen

< <= > >=

Opérations (d'identité)

Produit un booleen

== !=

Exemple :

```
int nb1;
nb1 = 12;
int nb2;
nb2 = 5;
int nb3;
nb3 = nb1 / nb2;      // nb3 vaut 2

if (4 == nb3)
{
    cout << "nb3 vaut 4";
}
else if (nb3 >= 2)
{
    cout << ("nb3 >= 2");
}
```

Les différents types d'entiers

Type	Min	Max	#octets
`int`	-2 147 483 648	2 147 483 647	4
`unsigned [int]`	0	4 294 967 295	4
`short [int]`	-32 768	32 767	2
`unsigned short [int]`	0	65 535	2
`long long [int]`	-9 223 372 036 854 770 000	9 223 372 036 854 770 000	8
`unsigned long long [int]`	0	18 446 744 073 709 551 615	8

Identificateur

`float`

Valeurs

Sous-ensemble des réels mathématiques

Opérations

Produisent un réel

`+ - * /`

L'opérateur division (/) renvoie la division réelle.

Produisent un booleen

`< <= > >=`

Opérations (d'identité)

Produit un booleen

`== !=`

Les différents types d'entiers

Type	#octets
`float`	4
`double`	8
`double double`	12

Plus il y a doctets, plus la précision est grande (voir cours darchitecture).

Égalité entre de deux réels

On ne compare jamais deux réels avec l'opérateur d'égalité, même si mathématiquement le résultat est juste !

Exemple : La comparaison suivante a de fortes chances d'être fausse :

```
float four;
four = 4.0;
float squareOfFour;
squareOfFour = ... // Computation of four
if (2.0 == squareOfFour)
{
   cout << "the value of the square root of 4 is 2" << endl;
}
```

Il est préférable de remplacer le test d'égalité par :

```
float eps = 0.000001;
if (abs (2.0 - squareOfFour) < Eps)
{
    cout << "the value of the square root of 4 is 2" << endl;
}
```