# M1102 - Amphi1 C++

Alain Casali  Marc Laporte

Aix Marseille Univ

# Langage de programmation

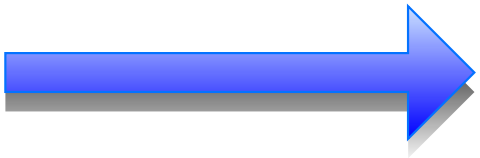
En informatique, un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est fait d'un alphabet, un vocabulaire, des règles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

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

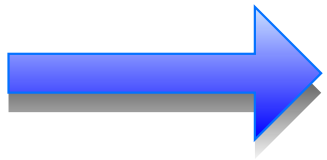
Fichiers


Executable

# Problème du développeur (1)



Cest une roue !



On ne redéveloppe **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

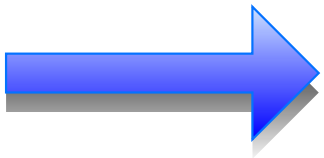
Elle se lit :

- 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 DUT : 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 2017.
Plusieurs compilateurs disponibles :

- g++ (support complet)
- clang (support complet)
- VS2014 (support partiel)

En C++ :

- Tout est appel de fonction ou de méthode (maintenant) ;
- Tout est flux (prochain module) ;
- 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 NomVariable : Type;
```

## Modèle général C++ :

```
Type VarIdent; //VarIdent (Variable Identifier)
```

## Exemple :

```
int i;
```

# Affectation

## Modèle général algorithmique :

```
NomVariable <- Valeur;
```

## Modèle général C++ :

```
VarIdent = Value;
```

## Exemple :

```
int i = 10;
```

# Cas des variables constantes

## Modèle général algorithmique :

```
declarer KNomVariable : 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 NomVariable : Type;
NomVariable <- 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 :

```
NomVariable 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 :

```
NomVariable 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 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++ :

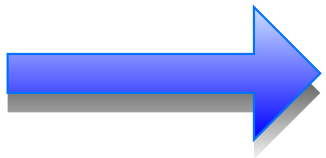
```
if (condition)
{
     instruction1;
     instruction2;
     ...
}
else
{
     instruction3;
     instruction4;
     ...
}
```

# Schéma alternatif avec condition sinon (2)

Exemple :

```
VerfifierVie;
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 (Expr_log_1)
    Sequ_1;
sinon_si (Expr_log_2)
    Sequ_2;
sinon_si (Expr_log_3)
    Sequ_3;
sinon //tous les autres cas
    Sequ_4;
fsi
```

## Modèle général C++ :

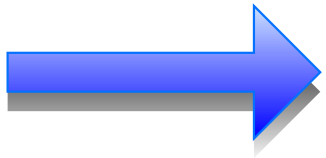
```
if (LogExp1)
{
    Seq1;
}
else if (LogExp2)
{
    Seq2;
}
else if (LogExp3)
{
    Seq3;
}
else
{
    Seq4;
}
```

# 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 << "c'est n'importe quoi !";
}
```

## Opérateur de négation

Produit un booleen

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 << "Racine de quatre vaut deux" << endl;
}
```

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

```
float Eps = 0.000001;
if (abs (2.0 - SquareOfFour) < Eps)
{
    cout << "Racine de quatre vaut deux" << endl;
}
```