

# Programmation et généricité en C/C++

# Présentation

- Objectifs de l'option:
  - apprendre les bases du C/C++
  - apprendre à 'penser générique':
    - implémenter des algorithmes à l'aide de la STL
    - concevoir des algorithmes génériques.
- 26h de cours.
- 24h de TPs.
- TPs effectués sous Suse Linux 9.0.
- Utilisation de l'IDE KDevelop.

# Introduction C

- C:
  - inventé par Kernighan et Richie
  - à l'origine des systèmes de type Unix.
  - langage procédural
  - mélange haut-niveau / bas-niveau
  - large support des compilateurs
  - performant
  - sa syntaxe est reprise par de nombreux langages (C++, Java, Objective-C, C#, ...)
  - censé être portable.

# Introduction C (2)

- Inconvénients:
  - portabilité réelle limitée
  - 'error-prone'
  - maintenir des projets d'envergure écrits en C peut s'avérer cavalier
  - gestion des erreurs lourde.

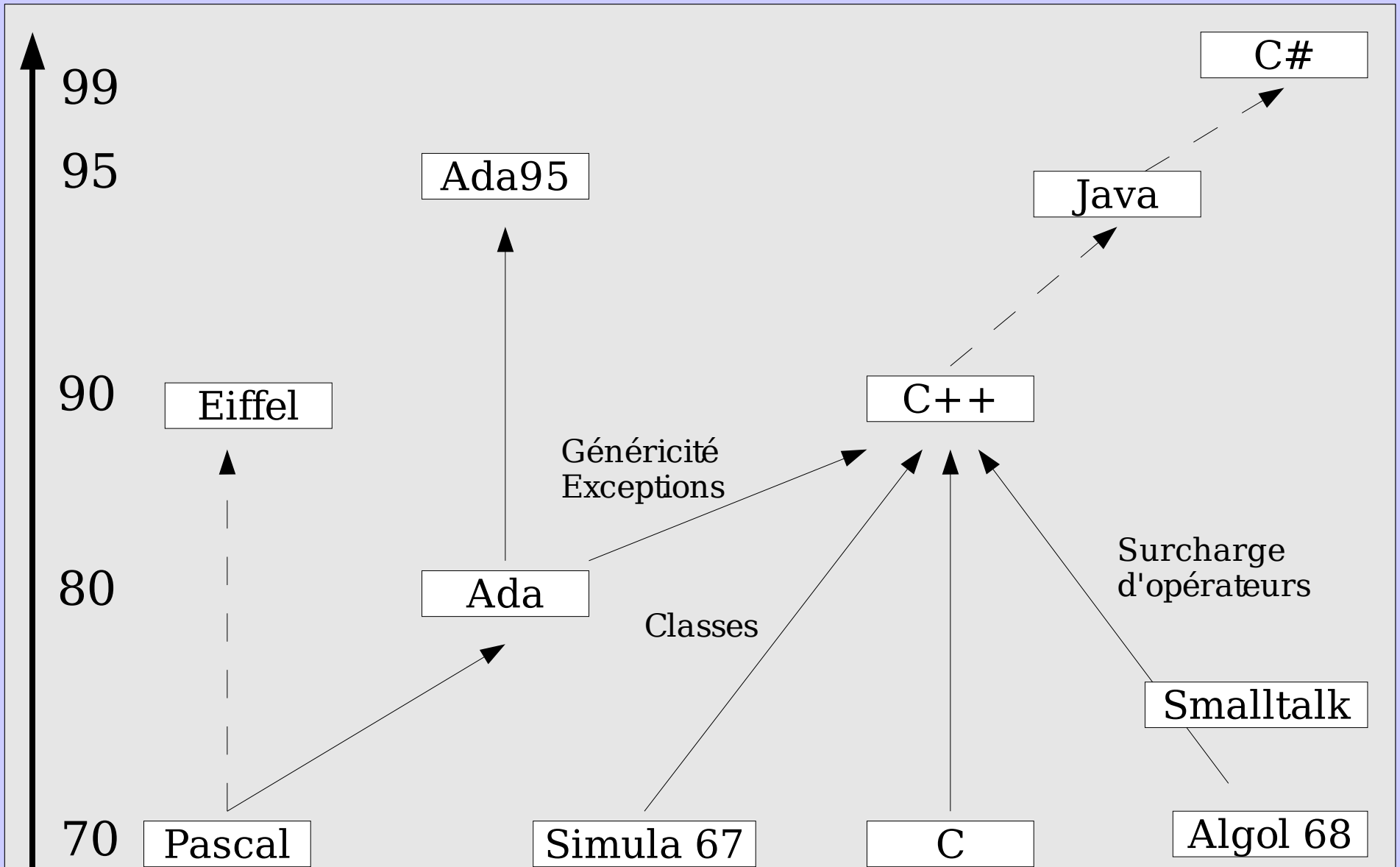
# Tendances actuelles autour de C

- Pour pallier aux inconvénients du C, plusieurs tendances se dégagent à l'heure actuelle:
  - utilisation de C pour des opérations de bas-niveau ou exigeantes en temps CPU
  - interfaçage avec un langage de plus haut-niveau (souvent de type scripting):
    - Python
    - Perl
    - Ruby
    - Visual Basic (pour les courageux).

# Nouveaux concepts

- De nouveaux concepts sont apparus avec d'autres langages:
  - généricité, exceptions (ADA)
  - classes (Simula 67)
  - surcharge d'opérateurs (Algol 68, Smalltalk).
- C++ étend C avec ces nouveaux concepts.

# Généalogie de C++



# Introduction C++

- Inventé par Bjarne Stroustrup dans les laboratoires AT&T Bell.
- Standard ISO depuis 1998 (ISO/IEC 98-14882).
- C++:
  - est un meilleur C
  - supporte l'abstraction de données
  - supporte la présentation orientée-objet
  - supporte la programmation générique.

# Introduction C++ (2)

- C++ n'est pas un langage objet !
- C++ est un langage multi-paradigme.
- Un programme C peut être compilé par un compilateur C++.
- C++ est plus portable que C.
- Compilateurs C++ majoritaires:
  - GNU GCC (libre)
  - Borland C++, MSVC++, Intel (propriétaires).

# Hello world C++

Structure type:

```
<inclusions>  
<déclarations globales>  
<implémentation>
```

```
#include <iostream>  
  
using namespace std;  
  
int main()  
{  
    cout << "Hello world !" << endl;  
    return 0;  
}
```

# Commentaires

```
// Ceci est un commentaire mono-ligne.  
// On peut m'insérer n'importe où:
```

```
x = y + z; // ajouter y+z à x ...
```

```
// Je suis une nouveauté de C++ (ne pas faire ce  
// type de commentaire en C !)
```

```
/*  
 * Ceci est un commentaire multi-ligne.  
 * Je puise mes origines dans le langage C.  
 */
```

```
/* On peut aussi m'employer comme ceci. */
```

# Variables

- C++ utilise un typage fort.
- Chaque variable possède un nom unique et doit être déclarée.
- Exemple de déclarations:

```
int x;           // Déclaration simple
int y = 2;       // Affectation à la déclaration
int w = z = 2;  // Déclaration + affectation en chaîne
int a, b = 1, c; // Déclarations multiples
```

# Nommage des variables

- Un nom de variable peut contenir n'importe quels caractères parmi:
  - 'a'...'z'
  - 'A'...'Z'
  - '0'...'9'
  - '\_'
- Un nom de variable ne peut pas commencer par un chiffre !
- Pas de lettres accentuées ('é', 'à', ...).
- Attention aux majuscules/minuscules !

# Blocs / portée des variables

- Un bloc est déclaré par des accolades.
- `{ ... }`  $\approx$  `begin ... end` du Pascal.
- La durée de vie d'une variable est celle du bloc où elle a été déclarée.
- Une variable déclarée en dehors d'un bloc (ou d'une fonction) est globale.
- Les blocs peuvent être imbriqués librement.
- **Les variables globales sont à éviter autant que possible.**

# Exemple de portées de variables

```
{
    int x;
    int y;
    {
        int a;
        // Ici x, y, a sont visibles
    }
    // Ici seuls x et y sont visibles
}
// Plus rien n'est visible
{
    // Pas de problèmes
    double x;
    double y;
}
```

# Types élémentaires

- Entiers:
  - char (8 bits)
  - short int (16 bits)
  - int (32 bits)
  - long int (64 bits).
- Modificateurs:
  - unsigned
  - long
  - short.
- Constantes: modificateur 'const'.

# Types élémentaires (2)

- Sans le modificateur 'unsigned', un entier peut être positif ou négatif.
- Par exemple:
  - char: [-127, 127]
  - unsigned char: [0, 255].
- 'char' s'emploie aussi pour désigner un caractère.

```
const char c1 = 'a';  
const char c2 = 97;  
// c1 et c2 sont égaux
```

# Types élémentaires (3)

- Nombres en virgule flottante:
  - float
  - double (plus précis que float)
  - long double.

```
// Sans qualification de type:  
float bankAccount = 10000.0;  
double income = 20345.87;
```

```
// Avec qualification de type:  
float bankAccount = 10000.0f;  
double income = 20345.87lf;
```

```
double z = 10; // A éviter ! Utiliser 10.0 ou 10.0lf
```

# Types élémentaires (4)

- 'bool' est le type pour les booléens. Les valeurs admises sont 'true' ou 'false'.
- 'void' est le type pour ... 'rien'. Il est principalement employé dans les déclarations de fonctions.
- Les énumérations permettent de nommer un ensemble de valeurs.
- Les énumérations portent sur des types entiers.

# Types élémentaires (5)

```
// Utilisation courante:  
enum couleur { ROUGE, VERT, BLEU };  
// -> ROUGE = 0, VERT = 1, BLEU = 2  
enum couleur ma_couleur = ROUGE;
```

```
// Affectation de valeurs explicites:  
enum couleur { ROUGE = 100, VERT = 010, BLEU = 001 };  
enum couleur ma_couleur = ROUGE;
```

# Types élémentaires (6)

- Il est possible de 'construire' de nouveaux types avec l'opérateur 'typedef'.
- On peut faire des conversions de type.

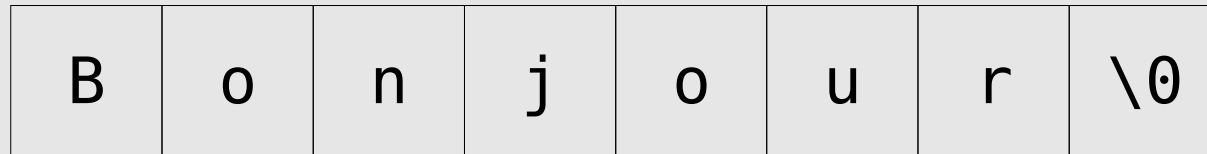
```
typedef enum { ROUGE, VERT, BLEU } couleur;  
couleur ma_couleur = ROUGE;  
  
typedef int entier; // Autre exemple  
entier x = 1;  
  
int x;  
double y = 2.5;  
x = (int)y; // Conversion de type (casting)
```

# Les chaînes de caractères (en C)

- Il n'existe pas de type pour les chaînes de caractères comme en Pascal, Python, VB, ...
- Une chaîne de caractères est un tableau de caractères.
- Nous verrons le fonctionnement des tableaux et pointeurs dans un autre cours.
- Une chaîne de caractères se termine par le caractère nul, noté '\0'.

# Exemple de chaîne de caractères

```
// Déclaration d'une chaîne de caractères:  
char* str = "Bonjour";
```



# Chaines de caractères (en C) (2)

- On ne peut pas effectuer d'affectation entre chaînes de caractères (il faut faire une copie en mémoire, mais cela dépasse le cadre de ce cours).
- La comparaison lexicographique nécessite des fonctions spécifiques.

```
// Erreurs classiques du débutant en C:
```

```
char* s1 = "Plop";  
char* s2 = s1; // A ne jamais faire sur des chaînes !!!  
  
char* s3 = "Bonjour";  
cout << s1 == s3 << endl; // s1 est différent de s3 !!!
```

# Chaines de caractères (en C++)

- La bibliothèque standard du C++ fournit un type 'string' de haut-niveau.
- Il intègre les opérations de comparaison (ordre lexicographique).
- L'emploi de ce type est simple.
- Défini dans l'en-tête 'string'.
- Appartient à l'espace de nommage 'std' (celui de la bibliothèque standard).

# Chaines de caractères (en C++) (2)

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

using namespace std;

int main()
{
    string str = "Hello world !";
    cout << str << endl;
    return 0;
}
```

# Opérateurs de base du C++

- Affectation: '='
- Test d'égalité: '=='
- Tests de comparaison: '<', '>', '!=', '>=', '<='
- Arithmétique: '+', '-', '\*', '/', '%'
- Logique: '==', '&&', '||', '!'

```
bool b = !(3 == 2) || (x < y);  
int x = 2, y = 3;  
int z = (x + y) / (x % y);
```

# Opérateurs de base du C++ (2)

- Les priorités des opérateurs sont 'classiques'.
- L'utilisation de parenthèses permet de les expliciter.
- Il est conseillé d'utiliser les parenthèses autant que possible.

# Opérateurs de base du C++ (3)

- Il existe des opérateurs condensés:
  - +=, -=, /=, %=
  - ++, -- .
- Affectation conditionnelle:
  - <expr> ? <val\_true> (: <val\_false>)
- '++' et '--' servent à faire des incréments ou décréments:
  - ++x: post-incrément
  - x++: post-incrément.

# Opérateurs de base du C++ (4)

```
int x = 2;
x += 3; // x == 5

int y = 1;
cout << ++y << endl; // Affiche 2

int z = 1;
cout << y++ << endl; // Affiche 1

int w = 5;
int i = (w >= 3) ? 3 : w; // i vaudra 3
int j = (w > 3) ? w : 3; // i vaudra 5
```

# Instructions

- Une instruction atomique est terminée par ';'.  
• Un bloc est constitué d'instructions atomiques.

```
if (condition)
{
    // instructions
}
else
{
    // instructions
}
```

# Instructions (2)

```
switch (condition)
{
case const1: instructions; break;
case const2: instructions; break;
case const3:
case const4: instructions;
case const5: instructions; break;
default: instructions; break;
}
```

```
switch (state)
{
case 1: x += 3;
       y -= 5; break;
case 2: y -= 10; break;
default: x = y = 0; break;
}
```

# Instructions (3)

```
while (condition)
{
    // instructions
}
```

```
do
{
    // instructions
} while (condition)
```

```
for (expr; cond; expr)
{
    // instructions
}
```

# Instructions (4)

- Dans une boucle 'for', on peut mettre plusieurs éléments dans les expressions ou la condition. Ceci n'est pas valable en C !
- Il est possible (et pratique) de déclarer une variable dans l'instruction de boucle 'for'.

# Exemples de boucles 'for'

```
int n = 0;
for (int i = 0; i < MAX; ++i)
{
    n += i;
}
```

```
for (int i = 0; i < MAX; i += 2)
{
    cout << i << endl;
}
```

```
for (int i = 0, int j = 10;
     i < MAX, j > MIN; ++i, --j)
{
    // ...
}
```

# Instructions (5)

- 'break' provoque la sortie immédiate d'une boucle ou de l'instruction 'switch'.
- 'continue' passe directement à l'itération suivante de la boucle.

```
for (int i = 0; i < MAX; ++i)
{
    if (i % 2 != 0) continue;
    cout << i << endl;
}
```

# Entrées / sorties

- Tout programme dispose de 3 flux d'entrées / sorties:
  - le flux de sortie standard
  - le flux d'erreurs
  - le flux d'entrée standard.
- Certains systèmes d'exploitation (moins évolués ?) mélangent les 2 premiers flux.

# Entrées / sorties (2)

- La bibliothèque standard du C++ nous fournit ces flux (en-tête 'iostream'):
  - cout (sortie standard)
  - cerr (sortie d'erreurs)
  - cin (entrée standard).
- Pour les flux de sortie, on dispose de 'endl' qui correspond à un retour à la ligne.
- Un retour à la ligne varie selon le système d'exploitation ('\n', '\r', '\r\n').

# Entrées / sorties (3)

- Pour travailler avec les flux, on utilise les opérateurs de flux '<<' et '>>'.
- On peut chaîner les opérateurs de flux.
- **La lecture d'une chaîne de caractères se fait avec séparation sur les espaces.**

```
#include <iostream>

using namespace std;

// (...)
int a, b;
cin >> a >> b;
cout << "a + b = " << a + b << endl;
```

# Fonction principale

- Tout programme doit posséder un point-d'entrée, c'est à dire une fonction qui sera appelée en premier.
- En C/C++, le point d'entrée est la fonction 'main' (nous verrons les fonctions dans un autre cours).
- La fonction main doit renvoyer un entier. Il est nul quand le programme termine normalement.

# Fonction principale, variantes

```
// Minimaliste
void main() {
    // ...
}

// Pas de gestion des paramètres
int main()
{
    // ...
    return 0;
}

// Gestion des paramètres
int main(int argc, char** argv)
{
    // ...
    return 0;
}
```

# Styles de code

```
// Style ANSI
void foo()
{
    int x = 0;
    if (x == 1)
    {
        // Panic !
    }
    else
    {
        // ok
    }
}
```

```
// Style K&R
void foo() {
    int x = 0;
    if (x == 1) {
        // Panic !
    } else {
        // ok
    }
}
```

```
// Style GNU
void foo()
{
    int x = 0;
    if (x == 1)
    {
        // Panic !
    }
}
```