

ISEN

ALL IS DIGITAL!

LILLE



yncréa

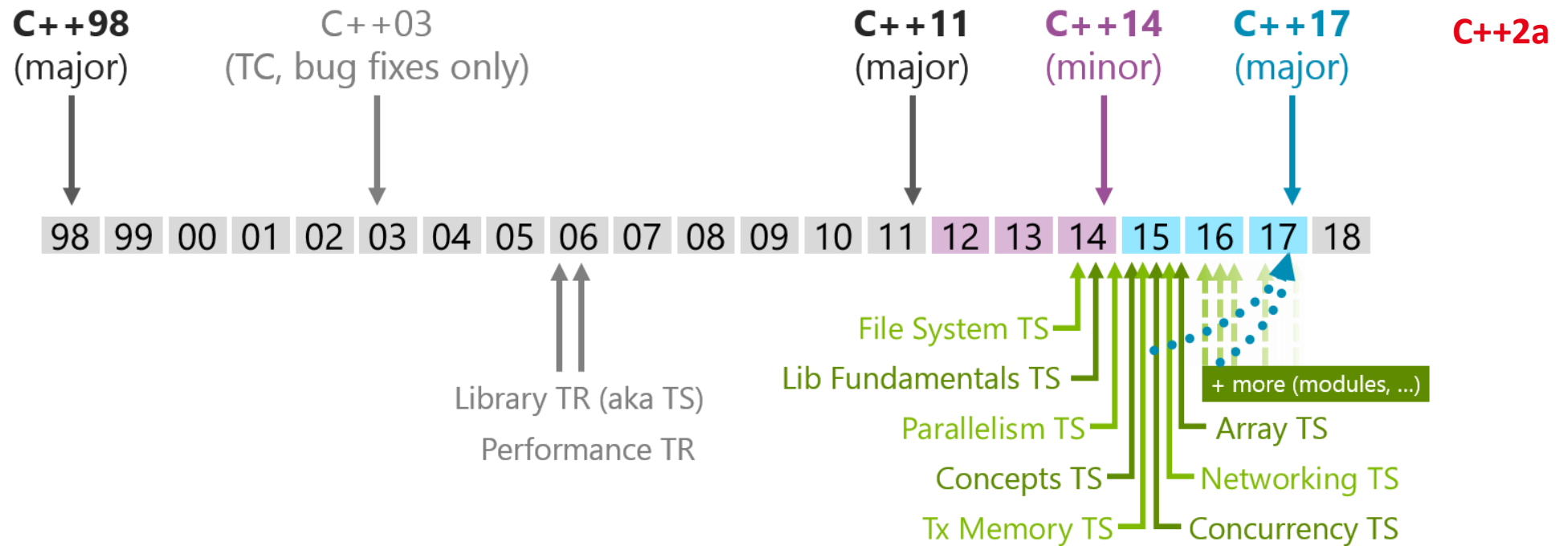


Le Langage C++

Introduction

- Première version début des années 80.
- Proposé par [Bjarne Stroustrup](#) (AT&T Lab).
- Présenté comme une extension du langage C, avec:
 - Des améliorations syntaxiques (nouveaux types, opérateurs et mots clés)
 - Introduction de la [Programmation Orienté Objet](#)
- Le nom original était "[C with classes](#)".

- Le C++ est régulièrement révisé.
- Certaines implémentations sont toujours en cours.



Source : <https://isocpp.org/std/status>

- Le type `bool`
 - Les mots clés `auto` et `decltype`
 - Le mot clé `const`
 - Les Références
 - La surcharge des fonctions
 - Valeurs d'arguments par défaut
 - Manipulation des flux avec `<<` et `>>`
 - Désallocation mémoire avec `new` et `delete`
 - Les *Namespaces* et l'opérateur de portée `::`
- ... Avant d'aborder la **P**rogrammation **O**rientée **O**bjet

- Extensions :
 - `.h` (headers) pour les fichiers entêtes
 - `.cpp` pour le code source
- La fonction `main()` doit retourner un `int`
- Compilateur : `g++` (sous Linux ou environnements minGW ou cygwin).
L'option `-std=c++17` doit être ajoutée pour le standard C++17 :
`g++ -std=c++17 filename.cpp`
- GCC (dans ses versions récentes) implémente partiellement la version standard ISO C++2a
- Pour suivre le développement ?? Aller sur
<https://gcc.gnu.org/projects/cxx-status.html>

En plus des commentaires `/* */`, le C++ permet l'usage des commentaires de ligne.
Les commentaires de ligne vont du symbole `//` à la fin de la ligne.

```
// variables initialization  
i=0; // row counter  
j=0; // column counter
```

Ils peuvent aussi être utilisés dans les directives de compilation

```
#define sq(x) (x*x) // computes the square of x
```

ISEN

ALL IS DIGITAL!

LILLE



yncréa



Ajouts syntaxiques C++

Les variables

- C < 99
 - Pas de support des booléens
- C > 99
 - Définis dans `stdbool.h`
 - Basé sur un type `int` et des macros `TRUE` et `FALSE`
- C++
 - Mots clés `bool`, `true` et `false`
 - Conversion automatique à partir des autres types
 - Généralement stocké sur un octet (en fonction des systèmes)


```
bool b;  
int i;  
  
b = true;  
i = b;           // i = 1  
  
b = false;  
i = b;           // i = 0  
  
i = 3;  
b = i;           // b = true  
  
i = -2;  
b = i;           // b = true  
  
i = 0;  
b = i;           // b = false
```

- Les variables locales peuvent être déclarées n'importe où dans un bloc.
- Un bloc est un ensemble d'instructions entouré de { et }.
- Les blocs anonymes (non liés à une fonction ou une boucle) sont syntaxiquement corrects
- La portée (et la vie) de la variable s'arrête à la fin du bloc.

```
#include <stdio.h> // for printf
```

```
int main(){  
  int start = 4;
```

```
  start++;
```

```
  int end = 10;
```

```
  for(int i = start; i < end; i++){  
    int k = i/2;  
    printf("%d\n", k);  
  }
```

```
  int j = k; ←
```

```
  printf("%d\n", j);
```

```
  return 0;  
}
```

La variable k est utilisée en dehors de sa portée
(compilation error : 'k' was not declared in this scope)

- Le mot clé **const** a plusieurs utilisations. L'une d'entre elles est la déclaration de constantes (ou variables protégées contre la modification) .
- La valeur d'une variable constante est donnée à la déclaration de cette variable.
- Les tentatives de modification génèrent des erreurs de compilation.
- Exemple :

```
const int i = 1;  // OK
```

```
i++;  /* compilation error  
      *error: increment of read-only variable 'i' */
```

- Une référence peut être vu comme un alias (d'une autre variable).
- Le symbole **&** est utilisé (placé entre le type et le nom de la référence).
- Une référence est initialisée à sa déclaration (référence à quel variable ??).
- La variable originale et sa référence pointent la même zone mémoire.
- Une amélioration syntaxique pour réduire l'usage des pointeurs.

```
variable_type original = value;
```

```
variable_type& reference = original;
```

Adresse	Contenu	Nom
	...	
42	value	original, reference
	...	

```
#include <stdio.h>

int main(){
    int i = 1;
    int& ref = i;

    printf("&i: %x, &ref: %x\n", &i, &ref);

    i++;
    printf("i: %d, ref: %d\n", i, ref);

    ref++;
    printf("i: %d, ref: %d\n", i, ref);

    return 0;
}
```

Ecriture équivalente
avec les pointeurs

```
#include <stdio.h>

int main(){
    int i = 1;
    int* ref = &i;

    printf("&i: %x, ref: %x\n", &i, ref);

    i++;
    printf("i: %d, *ref: %d\n", i, *ref);

    (*ref)++;
    printf("i: %d, *ref: %d\n", i, *ref);

    return 0;
}
```

- Détection automatique du type de la variable (**Depuis C++11**)

```
auto i = 0;           // i is int
auto j = 2 + 3;        // j is int
auto d = 3.14;         // d is a double
auto f = 0.0f;         // f is a float
```


- Introduit dans la norme C++11
- Ce mot clé signifie : le même type que ...
- Le type est déduit automatiquement à partir d'un autre type ou d'une expression
- Exemple :

```
int d = 5;
```

```
float pi = 3.14;
```

```
//x is the type you get when you multiply
```

```
decltype(d*pi) x = d*pi;
```

Les autres exemples d'usage seront discutés ultérieurement.

ISEN

ALL IS DIGITAL!

LILLE



yncréa



Les nouveautés syntaxiques

Les fonctions

- En C, les fonctions doivent avoir des noms différents.
- En C++:
 - Les fonctions peuvent avoir le même nom (dans le même espace) à conditions d'avoir des paramètres qui diffèrent en **types** ou en **nombre**
 - Lors de l'exécution, le compilateur sélectionne la fonction adéquate en comparant la liste d'arguments passés avec la liste des paramètres de chaque fonction surchargée.

- Exemple : Déclaration des fonctions - [overloaded_print.h](#)

```
void print_params(int i1, int i2);  
void print_params(int i, float f);  
void print_params(float f);
```

```
int print_params(int i1, int i2);
```

Ambiguïté / Erreur de compilation

- Définition des fonctions - [overloaded_print.cpp](#)

```
#include <stdio.h>

void print_params(int i1, int i2){
    printf("Parameters: integer %d and integer %d\n", i1, i2);
}

void print_params(int i, float f){
    printf("Parameters: integer %d and float %f\n", i, f);
}

void print_params(float f){
    printf("Parameter: float %f\n", f)
}
```

- Exemple d'appels – [test.cpp](#)

```
#include "overloaded_print.h"

int main(){
    int i1 = 1;
    int i2 = 2;
    float f = 3.0;

    print_params(i1, i2);

    print_params(i1, f);

    print_params(f);

    return 0;
}
```

- Un argument par défaut : une valeur spécifique utilisée par défaut si la valeur d'un paramètre n'est pas fournie lors de l'appel.
- Les arguments par défauts sont obligatoirement placés en dernier dans la liste des paramètres de la fonction.
- Ils peuvent être spécifiés dans la déclaration de la fonction ou dans sa définition, mais pas dans les deux.

```
int increment(int i, int step = 1);
```

 ✓

```
int increment(int i = 0, int step = 1);
```

 ✓

```
int increment(int i = 0, int step);
```

 ✗



- L'utilisation des arguments par défaut dans les fonctions surchargées peut générer des ambiguïtés.
- La règle est simple : toutes les versions induites par les arguments par défaut ainsi que les différentes surcharges fonction ne doivent pas être en conflit.
- Exemple.

- En plus du passage par copie et du passage par adresse, le C++ permet de passer des paramètres par référence (en utilisant le symbole `&`).
- Un paramètre passé par référence peut modifier la variable originale (référéncée).
- Placer le mot clé `const` devant le paramètre pour empêcher sa modification.
- L'usage des référence permet de lier les performances du passage par adresse au confort et la facilité d'écriture du passage par copie
- Il est aussi possible de retourner des variables par référence.

```
// declaration
```

```
return_type function_name(arg1_type&, arg1_type&, ...);
```

```
// appel
```

```
function_name(arg1, arg2, ...); // aucun symbole lors de l'appel
```

ISEN

ALL IS DIGITAL!

LILLE



yncréa



Ajouts syntaxiques C++

Les entrées sorties standards

- Les entrées/sorties standards du C peuvent toujours être utilisées.

En C

```
#include <stdio.h>  
stdin, stdout, stderr  
printf, scanf, ...
```

En C++

```
#include <iostream>  
std::cin, std::cout, std::cerr,  
les opérateurs << et >>
```

C

```
#include <stdio.h>

int main(){
    int i = 1;
    printf("i is %d\n", i);
    return 0;
}
```

C++

```
#include <iostream>

int main(){
    int i = 1;
    std::cout << "i is " << i << std::endl;
    return 0;
}
```

C

```
#include <stdio.h>

int main(){
    int i = 1;
    printf("i is %d\n", i);
    return 0;
}
```

C++

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

int main(){
    int i = 1;
    cout << "i is " << i << endl;
    return 0;
}
```

C

```
#include <stdio.h>

int main(){
    int i = 1;
    scanf("%d\n", &i);
    return 0;
}
```

C++

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

int main(){
    int i = 1;
    cin >> i;
    return 0;
}
```

- `#include <fstream>`
- Types prédéfinis : `fstream`, `ifstream`, `ofstream`, ...
- Des modes d'accès (qui peuvent être combinés avec `|`):
 - `ios::in`
 - `ios::out`
 - `ios::app`
 - `ios::trunc`
 - `ios::binary`
- Des méthodes pour manipuler les fichiers :
`Open()`, `close()`, `is_open()`, `eof()`, `read()`, `write()`, `getline()`, ...
- Les flux fichiers sont compatibles avec les opérateurs `<<` et `>>`.

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

int main(){
    ofstream file;
    file.open("example.txt");

    if(file.is_open()){
        file << "First line" << endl;
        file << "Second line" << endl;
    }

    file.close();

    return 0;
}
```



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

int main(){
    char buffer[100];
    ifstream file;
    file.open("example.txt");

    if(file.is_open()){
        while(!file.eof()){
            file.getline(buffer, 100);
            cout << buffer << endl;
        }
    }

    file.close();

    return 0;
}
```

ISEN

ALL IS DIGITAL!

LILLE



yncréa



Ajouts syntaxiques C++

La mémoire

En C

```
#include <stdlib.h>  
malloc(), calloc(), realloc(), free()
```

En C++

Rien à inclure
Les opérateurs **new** et **delete**



Ne pas mixer les fonctions et les opérateurs C/C++

- `int* i = new int;` `/* allocates 1 int, left uninitialized */`
- `int* j = new int();` `/* equivalent syntax */`
- `int* k = new int(3);` `/* allocates 1 int, and initializes it to 3 */`
- `float* f1 = new float[2];` `/* allocates 2 floats, left uninitialized */`
- `delete i, j, k;`
- `delete[] f1;`

ISEN

ALL IS DIGITAL!

LILLE



yncréa



Nouveautés syntaxiques C++

Les espaces de noms (Namespaces)

- Syntaxe :

`scope::variable`

- **Scope** peut être
 - Vide (variables globales)
 - Un espace de nom
 - Le nom d'une classe
 - ...
- Exemples

- Définissent un contexte
- Réduisent les problèmes de conflits de noms
- Similaires au *packages* dans d'autres langages
- Les mots clés `namespace` et `using` sont utilisés pour déclarer des espaces de noms ou de les utiliser dans un programme.

Déclaration

```
namespace mynamespace{  
    /* identifiers here */  
  
}
```

Utilisation

```
mynamespace::identifiant
```

Utilisation du contexte

```
using namespace mynamespace;
```

```
namespace f2{
    int factor = 2;

    int mult(int i){ return i*factor; }
}

namespace f4{
    float factor = 4.0;

    float mult(int i){ return i*factor; }
}
```

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

```
int main(){
    int i = 3;

    i = f2::mult(i);
    cout << "i = " << i << endl;

    i = f4::mult(i);
    cout << "i = " << i << endl;

    i = f2::factor;
    cout << "i = " << i << endl;
}
```