

Programmation procédurale

3

- Introduction
- Bref historique du langage C++
- Structure d'un programme C++
- Compiler C++
- Premier programme
- Types, variables, constantes
- Affectation
- Les fonctions mathématiques
- Les entrées-sorties standards de C++
- Opérateurs et expressions
- Instructions de contrôle
- Instructions de branchement inconditionnel
- Q & A
- Exercices

Introduction

4

Algorithme et programmation

□ Définition: Algorithme

Un algorithme est une suite [finie] d'opérations élémentaires permettant d'obtenir le résultat final déterminé à un problème.

□ **Algorithme:** méthode pour résoudre un problème.

□ Propriété d'un algorithme

Un algorithme, dans des conditions d'exécution similaires (avec des données identiques) fournit toujours le même résultat.

Introduction

5

Algorithme et programmation

☐ Définition: Programme

Un programme informatique est un ensemble d'opérations destinées à être exécutées par un ordinateur.

☐ Programme: s'adresse à une machine !

Introduction

6

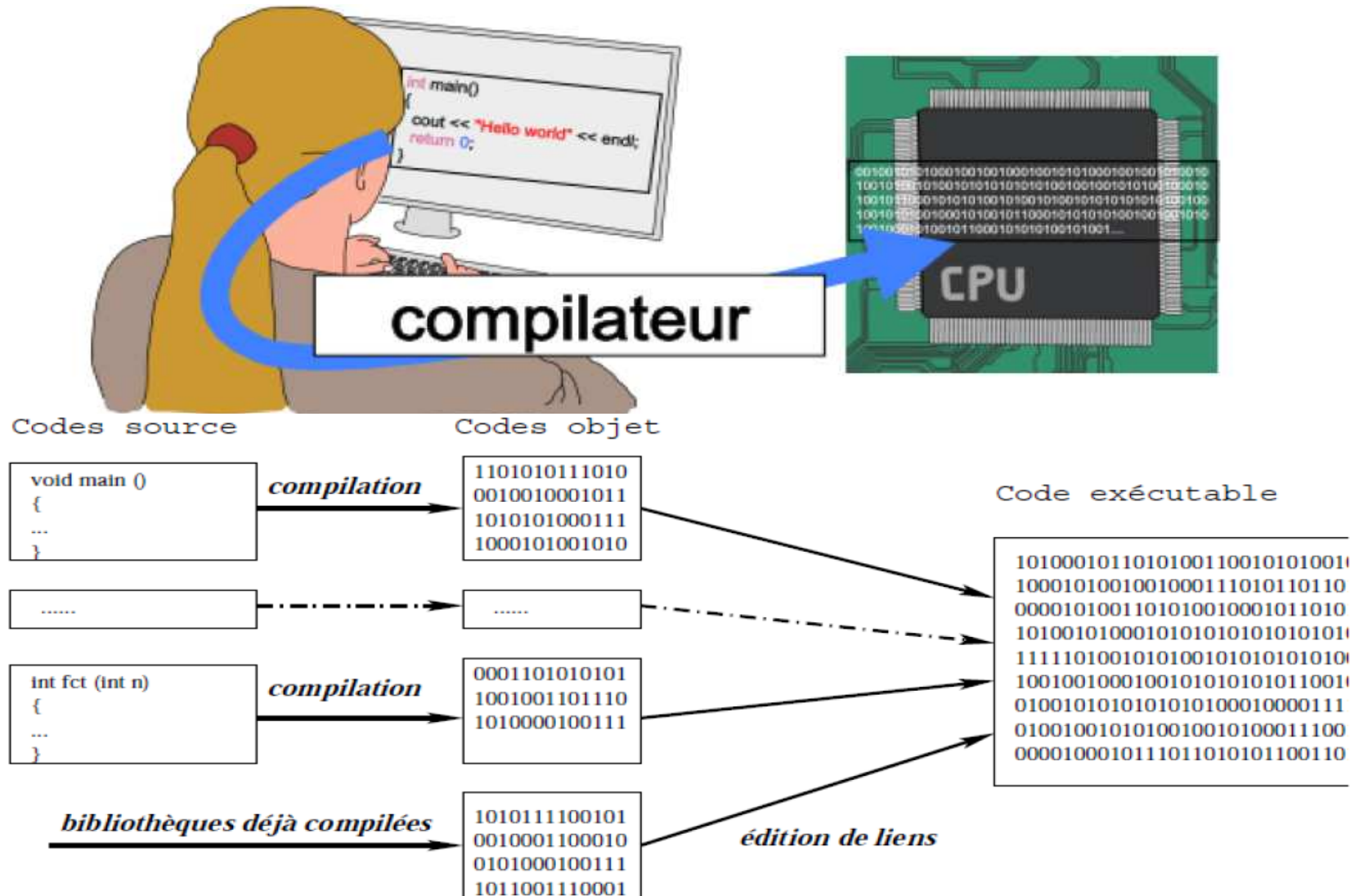
Qu'est-ce qu'un langage de programmation?

- ❑ Le **langage** est la capacité d'exprimer une pensée et de communiquer au moyen d'un système de signes doté d'une sémantique, et le plus souvent d'une syntaxe. Plus couramment, **le langage est un moyen de communication.**
- ❑ Un **langage de programmation** est un code de communication, permettant à un être humain de dicter des ordres (instructions) à une machine qu'elle devra exécuter.

Introduction

7

Le C++ est un langage compilé



Introduction

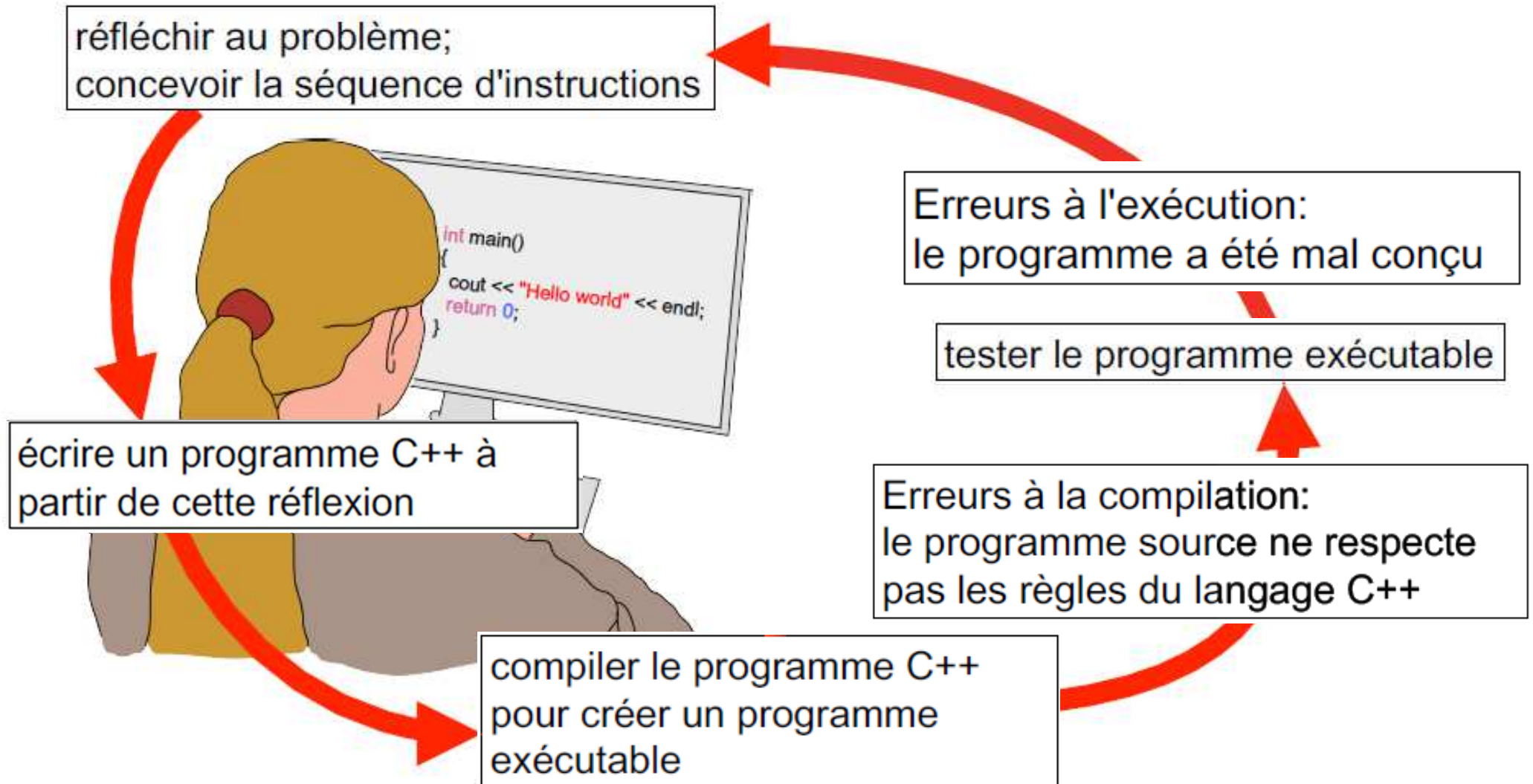
8

Étapes de réalisation d'un programme

1. Ecrire des instructions pour l'ordinateur dans un langage de programmation (par exemple le C++) ;
2. Les instructions sont traduites en binaire grâce à un **compilateur**;
3. L'ordinateur peut alors lire le binaire et faire ce que vous avez demandé !

Introduction

9



Bref historique du langage C++

10

- ❑ **1972 : Langage C**

AT&T Bell Labs.

- ❑ **1979: C with classes**

Bjarne Stroustrup développe le langage *C with classes*.

- ❑ **1985 : C++**

AT&T Bell Labs; **Extension Objet** par **Bjarne Stroustrup**.

- ❑ **1995 : Java**

Sun Microsystems puis Oracle; **Simplification du C++**, **purement objet**, également inspiré de Smalltalk, ADA, etc.

- ❑ **1998 : C++98: Normalisation du C++ par l'ISO**

(l'Organisation internationale de normalisation).

Bref historique du langage C++

11

- ❑ **2001: C#**

Microsoft; Originellement proche de **Java** pour **.NET**, également **inspiré de C++**, Delphi, etc.

- ❑ **2011: C++11**

Révision majeure du C++

- ❑ **2014: C++14**

Mise à jour mineure du langage C++11

- ❑ **2017: C++17**

Sortie de la dernière version

- ❑ **C++20: planifié depuis juillet 2017.**

C++ versus C

12

- ❑ Le langage **C** est inclus (à 99%) dans le langage **C++**
- ❑ Le **C++** rajoute des notions de programmation orientée objet (classe, héritage, ...), ...
- ❑ Un programme écrit en langage **C++** ressemble beaucoup à un programme écrit en langage **C**, à la différence près qu'il contient essentiellement des classes.
- ❑ **C++ = extension du langage C**
 - un compilateur **C++** peut **compiler du C** (avec qq restrictions)
 - un même programme peut **combinaison C, C++ et Objective C** (Apple) ou **C#** (Windows).

Caractéristiques principales

13

- ☐ Orientation Objet
- ☐ Grand nombre de fonctionnalités
- ☐ Performances du C
- ☐ Portabilité des fichiers sources
- ☐ Robustesse (typage fort, ...)
- ☐ Facilité de conversion des programmes C en C++, et, en particulier, possibilité d'utiliser toutes les fonctionnalités du langage C.
- ☐ Richesse des librairies (C++ Standard Library) et également les librairies du langage C
- ☐ Nombreuses bibliothèque de programmes dans des domaines très variés.

Caractéristiques principales

14

- ☐ Il intègre une interface graphique
- ☐ Multitâche intégré au langage (Multithreading)
- ☐ Bonne intégration des communications réseau
- ☐ Évolutivité (C++, C++11, C++14, C++17, C++20?)
- ☐ Des éléments de sécurité intégrés
- ☐ Sa gratuité
- ☐ Son adoption dans la formation (écoles, universités)
- ☐ Son nom ?

Choix de C++

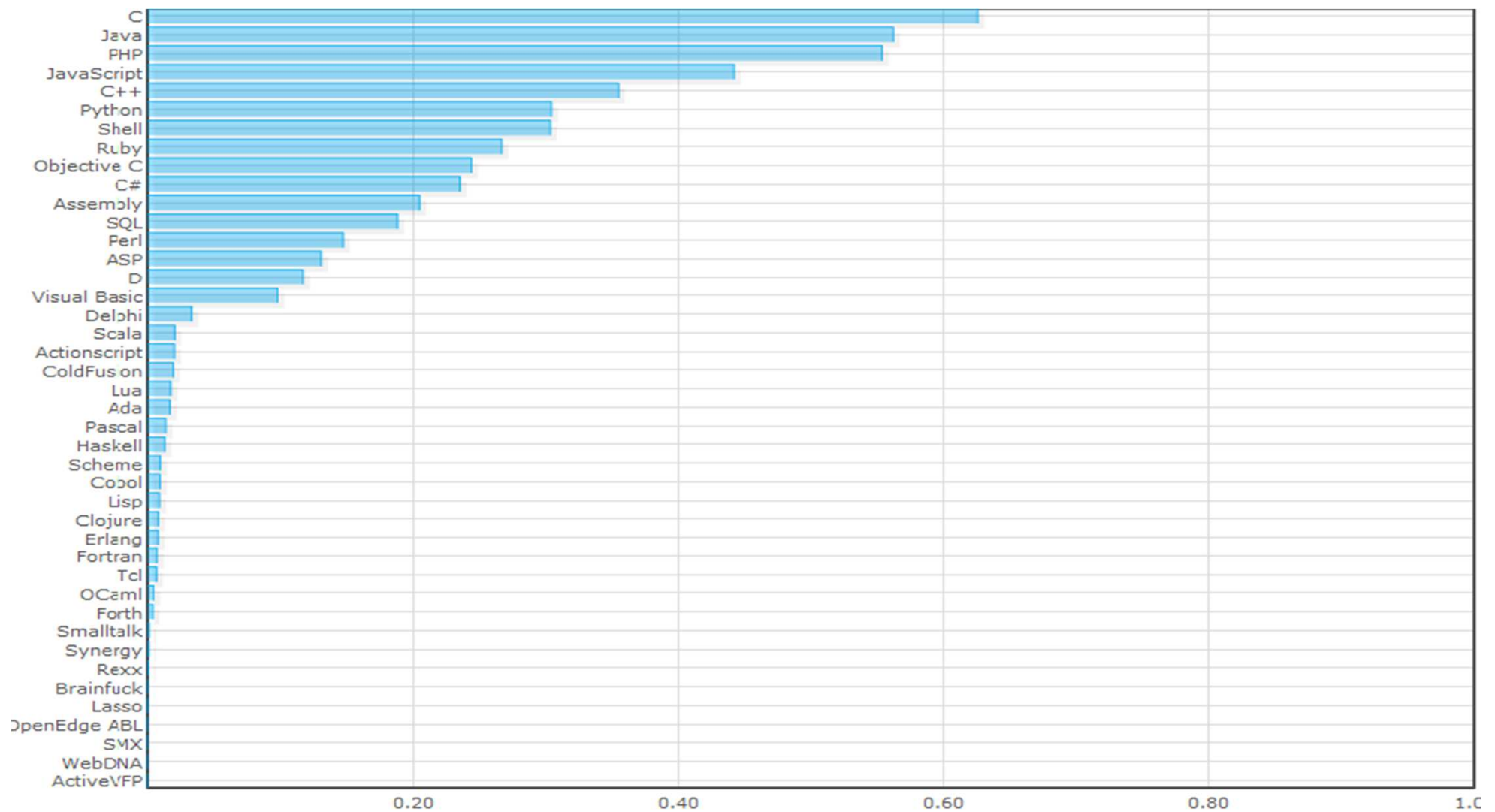
15

- ❑ Un langage de programmation procédurale
- ❑ Un langage de programmation orientée objet
- ❑ Un langage de programmation générique
- ❑ Un langage de programmation très populaire et très utilisé
- ❑ C'est un langage généraliste ayant un très vaste d'application
- ❑ (réseau, base de données, calcul scientifique, etc.)
- ❑ Il est rapide (calcul scientifique)
- ❑ PHP, JavaScript, C# et Java se sont fortement inspirés de C++
- ❑ ...

Choix de C++

16

Normalized Comparison 2014



Source: Programming Language Popularity (<http://65.39.133.14/>)

Choix de C++

17

Normalized Comparison 2015

Language Rank	Types
1. Java	  
2. C	  
3. C++	  
4. Python	 
5. C#	  
6. R	
7. PHP	
8. JavaScript	 
9. Ruby	 
10. Matlab	

Normalized Comparison 2016

Language Rank	Types
1. C	  
2. Java	  
3. Python	 
4. C++	  
5. R	
6. C#	  
7. PHP	
8. JavaScript	 
9. Ruby	 
10. Go	 

Choix de C++

18

Normalized Comparison 2017

Language Rank	Types
1. Python	 
2. C	  
3. Java	  
4. C++	  
5. C#	  
6. R	
7. JavaScript	 
8. PHP	
9. Go	 
10. Swift	 

Normalized Comparison 2018

Language Rank	Types
1. Python	  
2. C++	  
3. Java	  
4. C	  
5. C#	  
6. PHP	
7. R	
8. JavaScript	 
9. Go	 
10. Assembly	

Source: <https://spectrum.ieee.org/>

Choix de C++

19

TIOBE Index for August 2016

Aug 2016	Aug 2015	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	4		C#
5	5		Python
6	7	↑	PHP
7	9	↑	JavaScript
8	8		Visual Basic .NET
9	10	↑	Perl
10	12	↑	Assembly language

Choix de C++

20

TIOBE Index for September 2017

Sep 2017	Sep 2016	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	4		C#
5	5		Python
6	7	^	PHP
7	6	v	JavaScript
8	9	^	Visual Basic .NET
9	10	^	Perl
10	12	^	Ruby
11	18	^^	R
12	11	v	Delphi/Object Pascal
13	13		Swift

Choix de C++

21

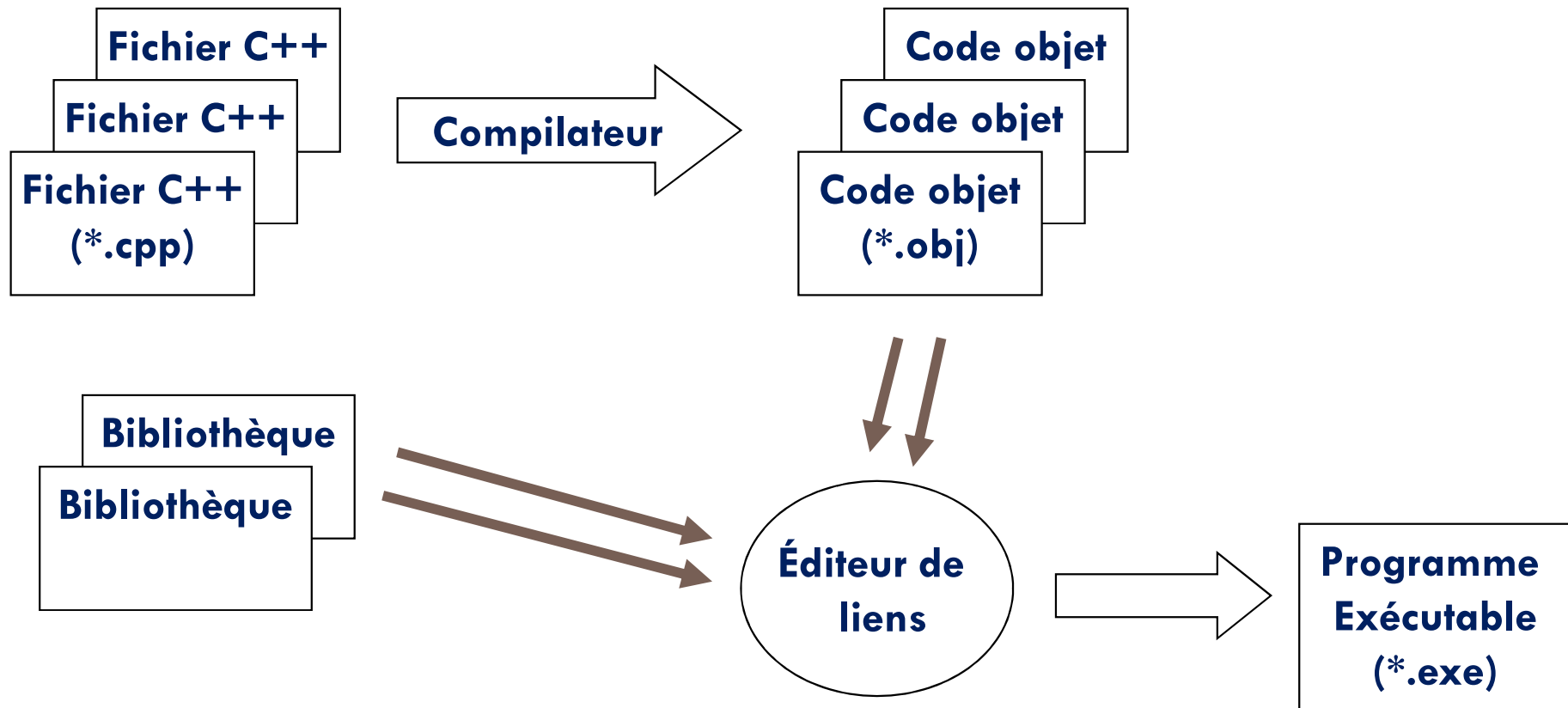
TIOBE Index for August 2018

Aug 2018	Aug 2017	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	5	⬆	Python
5	6	⬆	Visual Basic .NET
6	4	⬇	C#
7	7		PHP
8	8		JavaScript
9	-	⬆	SQL
10	14	⬆	Assembly language
11	11		Swift
12	12		Delphi/Object Pascal
13	17	⬆	MATLAB
14	18	⬆	Objective-C
15	10	⬇	Ruby

Structure d'un programme C++

22

- Un programme se présente comme une suite de fichiers.

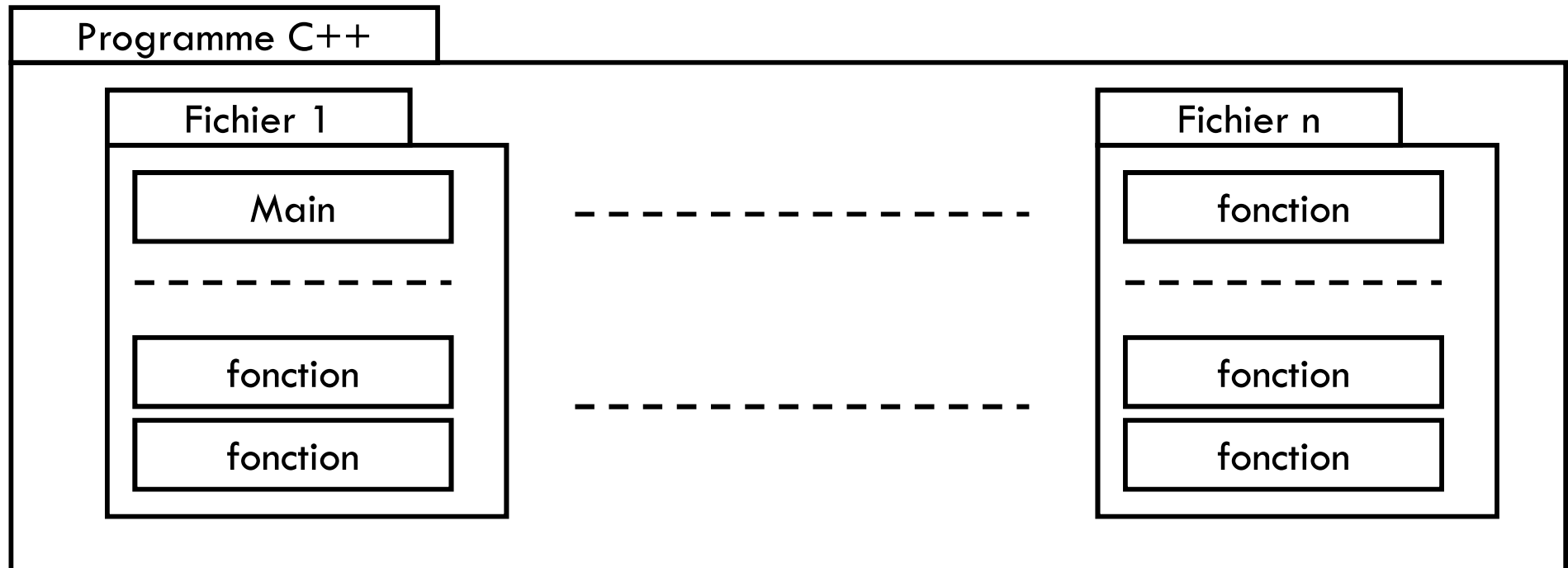


C++ est un langage compilé

Structure d'un programme C++

23

- ❑ Chaque fichier se présente comme une suite de fonctions.
- ❑ Une fonction est imposée: la fonction *main()*
 - Fonction principale.
 - Point d'entrée du programme.



Premier programme C++

24

- Un programme en C++ est un fichier text:

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

```
int main(){
```

```
    cout << "Hello World !" << endl;
```

```
    return 0;
```

```
}
```



une instruction

Output:

Hello World !

Premier programme C++

25

❑ `#include <iostream>`


- Les lignes qui commencent par # sont des instructions à destination du préprocesseur.
- Dans **iostream**: i=>input, o=>output.
- La librairie permet de gérer des flux d'entrées et de sorties.
- En effet **iostream** est une librairie de la norme C++ standard.

❑ `using namespace std;`

L'instruction sert à spécifier que l'on utilise l'espace de noms **std**.

Premier programme C++

26

- ❑ **int main()** : Tous les programmes possèdent une fonction dénommée « main » ce qui signifie « principale », elle représente le point d'entrée.
- ❑ Pour écrire à l'écran : **cout**
 - **cout << SomeString << endl;**  Pour afficher sur le console.
 - **endl** correspond à "retour à la ligne": ce qui sera affiché après le sera au début de la ligne suivante.

Exemple:

cout << "output";  Il affiche "output" sur le console

Compilation

27

- ❑ Un programme en langage C++ est un **fichier texte**, que l'on écrit à l'aide d'un **éditeur de texte**.
- ❑ Ce programme en langage C++ n'est pas exécutable directement par la machine: il doit être *compilé* pour pouvoir être exécuté par l'ordinateur.
- ❑ La compilation est réalisée par un programme appelé **compilateur**. Le compilateur crée un **fichier exécutable**.

Compilation et exécution

28

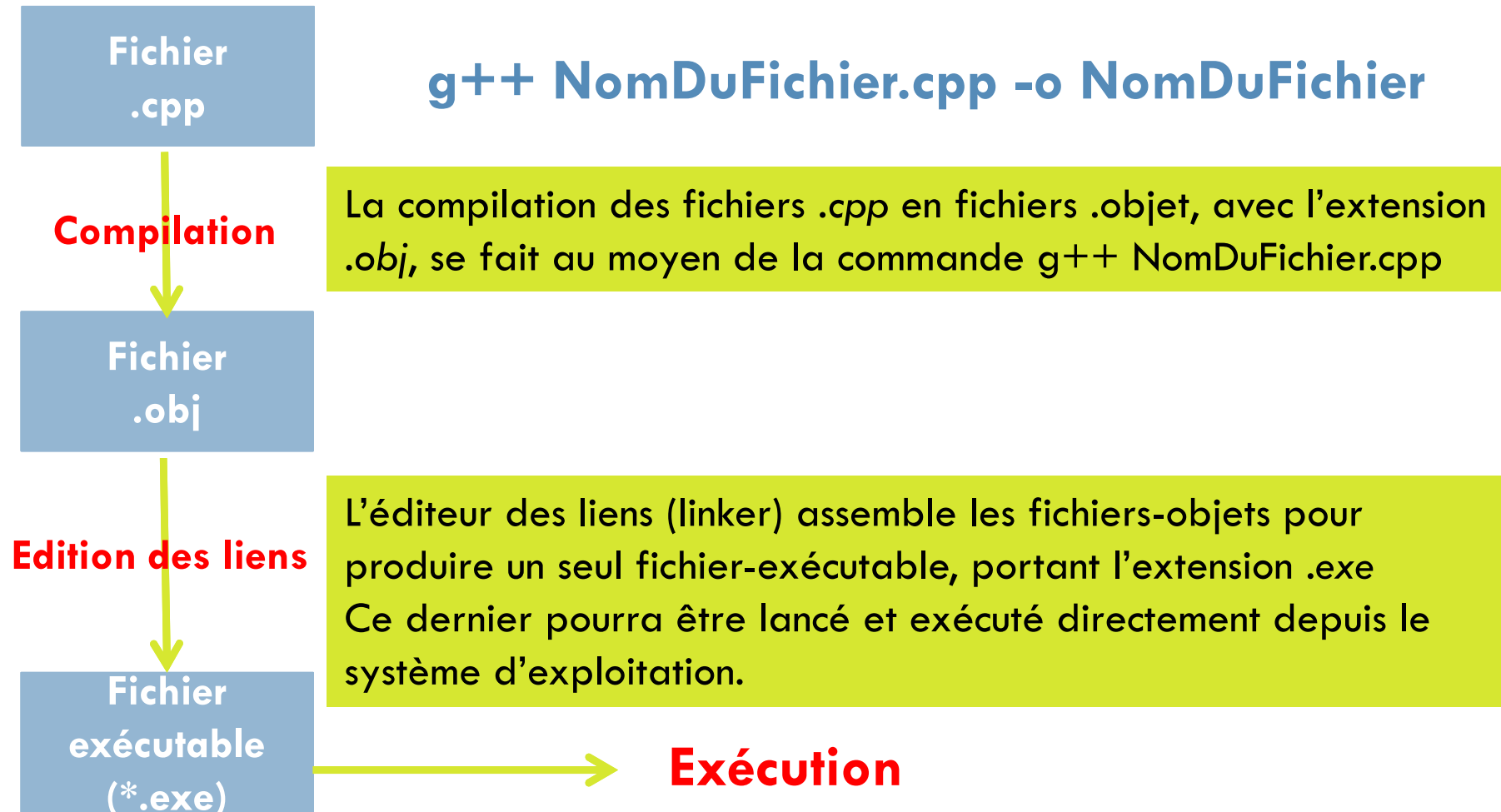


Programme en langage C++:
*Fichier texte compréhensible
par un programmeur*

*Fichier compréhensible par
l'ordinateur*

Compilation et exécution

29



Éléments du Langage

30

- **Commentaires**
- **Variables**
- **Constantes**
- **Types de base**
- **Les opérateurs et les expressions**
- **Les instructions de contrôles**
- **Les instructions de branchement inconditionnelles**

Commentaires

31

- formes de commentaires :

// ...Texte...

- ✓ Commence dès *//* et se termine à la fin de la ligne
- ✓ Sur une seule ligne
- ✓ A utiliser de préférence pour les commentaires généraux

/ ...Texte... */*

- ✓ Le texte entre */** et **/* est ignoré par le compilateur
- ✓ Peuvent s'étendre sur plusieurs lignes
- ✓ Ne peuvent pas être imbriqués
- ✓ Peuvent être utiles pour inactiver (temporairement) une zone de code

Exemples de commentaires

32

```
/*  
* Produit : Calcul du produit de deux nombres entiers  
*  
* Author Aziz DAROUICHI  
* Version 1.0 10.09.2018  
*/  
  
#include <iostream> // sorties standards  
  
int main() {  
    //--- Initialisation des variables  
    int a(3);  
    int b(4);  
    int total;           // Total pour stocker le produit  
    total = a * b;  
    //--- Affichage du résultat du calcul du produit-----  
    std::cout<<"Résultat = " << total;  
    return 0;  
}
```

Les variables

33

Une variable possède 3 caractéristiques:

- ❑ Son **identificateur**, qui est le nom par lequel la donnée est désignée;
- ❑ Son **type**, qui définit de quel « genre » est la donnée contenue dans la variable;
- ❑ Sa **valeur**. Par exemple, si la donnée est un nombre, sa valeur pourra être 12 ou 3.5

Exemple:

```
int n;
```

```
double x;
```

Les variables

34

Identificateurs

- ❑ Les **identificateurs** sont des **noms symboliques** permettant de référencer les programmes en C++ (variables, fonctions,...);
- ❑ Règles pour les **identificateurs**:
 - Doivent commencer par une lettre;
 - Suivi éventuellement d'un nombre quelconque de lettres et de chiffres (pas d'espace, ni de symboles !);
 - Distinction entre les majuscules et les minuscules.
 - Les accents ne sont pas autorisés;
 - Le caractère souligné `_` (*underscore*) est autorisé et considéré comme une lettre;
 - Les mot-clé réservés du langage C++ sont exclus;
 - Les identificateurs `x` et `X` désignent deux variables différentes.

Les variables

35

Identificateurs

Exemples des noms valides:

- n_carre
- Somme
- sousTotal98

Exemples des noms non valides:

- n_carré → Contient un accent;
- n carre → Contient des espaces;
- n-carre → Contient le symbole – (moins);
- 1element → Commence par un chiffre.

Notion de variable

36

- En informatique, la notion de **variable** est une **notion fondamentale** (un concept essentiel).
- Une **variable** définit une **case mémoire nommée et typée**.
 - Le **nom** est représenté par un identificateur
 - Le **type** définit le genre d'informations qui sera enregistré ainsi que les opérations qui pourront être effectuées sur ces informations
 - Il existe un certain nombre de types prédéfinis mais il est également possible de créer ses propres types

Déclaration de variables

37

- Avant de pouvoir être utilisée, une variable doit préalablement être **déclarée** (définition de l'identificateur et du type).
- Syntaxe :

Type *Identificateur* ;

```
float    note;  
Polygone triangle;
```

note 4.75

- Une déclaration de variable permet à l'ordinateur de **réserver de l'espace mémoire** (dont l'espace dépend du type) et donc de **créer la variable**.
- Une fois défini, le type de la variable ne peut plus changer.

Déclaration de variables

38

Exemple :

```
int n;      // déclaration d'une variable  
double x;  
string s;
```

- Ce sont des **déclarations de variables**.

Notion de variable

39

- La **valeur de la variable** peut être définie (initialisée), consultée, modifiée en utilisant le nom de la variable.
- Les **variables** ont une **visibilité** et une **durée de vie** qui dépendent du contexte dans lequel elles sont déclarées.
- Le **type d'une variable** peut être soit un **type primitif**, soit un **type référence** (c'est-à-dire le nom d'une classe ou d'un tableau)
- Exemples :

```
int      unNombreEntier; // Type primitif int
String   motDePasse;    // Classe String (prédéfinie)
Point    position;      // Classe Point
char[]   voyelles;      // Tableau de caractères
Point[]  nuage;          // Tableau de Point(s)
```

Représentation en mémoire

40

- De manière interne, la zone mémoire allouée à une variable est identifiée par une adresse.
- Cette adresse est complètement cachée au programmeur : l'accès à la zone mémoire s'effectue en utilisant le nom de la variable.

Code

```
. . .  
float note;  
. . .
```

```
. . .  
note = 5.2;  
. . .
```

Représentation interne

Variable	Adresse	Contenu
	1215	
note	1216	
	1217	

Variable	Adresse	Contenu
	1215	
note	1216	5.2
	1217	

Notation abstraite (à utiliser)

note

note

Initialisation des variables

41

- En même temps qu'elle est déclarée, une variable peut être initialisée, c'est-à-dire lui donner une valeur avant de l'utiliser.
- `int n(12);`
déclare donc une variable appelée n et lui donne la valeur 12.
- Si une variable n'est pas initialisée, elle peut contenir une valeur aléatoire!
- Pour cela, **il faut toujours initialiser les variables.**

Déclaration de variables avec initialisation

42

Exemples :

```
int n(1);           // déclaration et initialisation
int p(0), q(2);     //on peut déclarer plusieurs variables simultanément.
                   //Ne pas en abuser
double x(0.5), y;
string s("Hello");
```


Les constantes

43

- Il peut arriver que la valeur d'une variable ne doive pas changer après l'initialisation.
- Une constante est une valeur qui ne change pas au cours de l'exécution d'un programme.
- Dans ce cas, il faut ajouter le mot-clé **const** devant la déclaration:

const type identificateur(valeur_initiale);

- Le mot réservé « **const** » qualifie un **nom de variable** pour indiquer qu'*au travers de ce nom*, la valeur ne peut pas être modifiée.

Les constantes

44

Exemple :

```
const double LARGEUR_MAX (8.2); // déclaration d'une constante
```

```
const double PRIX_BARIL (44.62);
```

```
double km (437.3);
```

```
cout << km/100.0*LARGEUR_MAX;
```

- Ce sont des constantes:

LARGEUR_MAX

PRIX_BARIL

Les constantes

45

- Mot-clé **const** interdit la réaffectation de valeur

```
const double VITESSE_DE_LA_LUMIERE(299792.458);
```

- Dans ce cas, on ne peut plus modifier la variable:



```
VITESSE_DE_LA_LUMIERE = 200; // erreur!!!
```

Les constantes

46

Une valeur `const` n'est pas nécessairement connue à la Compilation

Supposons que l'on demande une valeur `i` à l'utilisateur de notre programme et que, une fois entrée, la valeur de `i` ne sera plus modifiée dans la suite. On pourrait bien sûr écrire :

```
int i;
```

```
cout << "Entrez une valeur : ";
```

```
cin >> i;
```

mais cela ne souligne pas, ne force pas, le fait que la valeur `i` ne soit plus être modifiée dans la suite.

Les constantes

47

Une valeur const n'est pas nécessairement connue à la Compilation

Pour bien marquer cela il serait préférable d'écrire :

```
int lue;
```

```
cout << "Entrez une valeur : ";
```

```
cin >> lue;
```

```
const int i(lue); // i est initialisé avec la valeur lue
```

On voit bien sur cet exemple que :

- une fois donnée, la valeur de i ne peut pas être modifiée ;
- pourtant la valeur que prend effectivement i n'est pas connue au moment de compiler le programme.

Les constantes en C++11

48

Constexpr

- Un nouveau mot réservé, *constexpr*, qui signifie justement « connu à la compilation et constant ».
- Une variable (ou plus largement une expression) peut être qualifiée de *constexpr* si justement ces deux conditions sont remplies :
 - on connaît sa valeur au moment de la compilation ;
 - cette valeur ne changera pas au cours du programme.

Exemple:

```
constexpr double pi(3.141592653589793238463);
```

Les types de base de C++

Types de base

50

Les types de base du langage C++ de valeurs qui peuvent être enregistrées et manipulées :

- ❑ **int**: pour les valeurs entières (pour **integer**, entiers en anglais);

Exemple : 0, 10, -47.

- ❑ **float ou double**: pour les nombres à virgule;

Exemple : 3.14, 1.0, -2.1.

- ❑ **char**: **Character**: pour les caractères (a...z, etc.);

Exemple : 'a', '+', '\$', '3'.

- ❑ **bool**: **Boolean**: pour les booléens;

Exemple : true or false.

Types entiers

51

■ Entiers signés (**short, int, long**)

- Le type `int` n'est pas le seul type qu'on peut utiliser pour déclarer des variables contenant des valeurs entières.
- Il existe aussi les types : `long int`, qu'on peut écrire aussi simplement `long`, et `short int`, qu'on peut écrire aussi simplement `short`.

Exemple:

```
int m;  
long int n;  
long n2;  
short int p;  
short p2;
```

Types entiers

52

▪ Entiers signés (**short**, **int**, **long**)

- Valeurs littérales (**int** par défaut) : notation habituelle
- Suffixe **l** ou **L** pour type **long**
- Préfixe **0** (zéro) pour valeur octale (base 8)
- Préfixe **0b** ou **0B** pour valeur binaire (base 2)
- Préfixe **0x** ou **0X** pour valeur hexadécimale (base 16)

• Exemples :

```
0          // valeur de type int
123        // valeur de type int
-56        // valeur de type int
0377       // 377 [octal] = 255 [décimal]
433L       // valeur de type long
0b1011     // valeur binaire de type int
0xff       // valeur hexadécimale de type int
0xA0B3L    // valeur hexadécimale de type long
```

Types entiers

53

- Entiers non-signés (**unsigned**)
 - On peut également ajouter le mot-clé **unsigned** devant chacun de ces 3 types pour obtenir 3 nouveaux types qui servent à déclarer des variables contenant des entiers positifs (ou nuls).

Exemple

```
unsigned int p;  
unsigned long q;  
unsigned short int r;
```

Types entiers

54

- Entiers non-signés (**unsigned**)
 - Suffixe **u** pour type **unsigned**
 - Suffixe **ul** pour type **unsigned long**
 - **Exemples:**
 - 2u //pour type unsigned int
 - 15u //pour type unsigned int
 - 3452ul //pour type unsigned long

Limitations des types entiers

55

Type	Contient	Valeurs (min à max)
int	Entier signé	-2 147 483 648 à +2 147 483 647
short	Entier signé court	-32 768 à +32 767
long	Entier signé long	environ -10^{18} à $+10^{18}$
unsigned short int	Entier positif court	0 à 65535
unsigned int	Entier positif	0 à 4 294 967 295
unsigned long int	Entier positif long	0 à $+10^{18}$

Type booléen ou logique

56

▪ Booléen (**bool**)

- Ne peuvent prendre que deux valeurs : *Vrai* ou *Faux*
- peuvent être interprétés comme des valeurs numériques [0, 1]
- Valeurs littérales
 - ✓ **true** *Vrai*
 - ✓ **false** *Faux*

Exemple 1:

```
bool CppEstFacile = true;
```

Type booléen

57

- ❑ Ce type est codé sur le même nombre de bits que le type **int**.
- ❑ Lorsque l'on convertit un type numéraire en **bool**, toute valeur non nulle est considérée comme **true**.

Exemple:

```
int a(1);
```

```
int b(2);
```

```
bool test1(a == b);
```

```
bool test2(a < b);
```

Types caractères

58

▪ Caractère (**char**)

- Caractères *Unicode* (codage normalisé sur 16 bits)
- Site de référence de la norme : www.unicode.org
- Peuvent être traités comme des entiers
- Valeurs littérales
 - ✓ Entre apostrophes : 'A' (attention : "A" n'est pas de type **char**)
 - ✓ Séquence d'échappement pour certains caractères spéciaux (voir table page suivante)

```
char c1, c2 ; //c1 et c2 sont deux variables de type caractère
```


Types caractères

59

- Séquences d'échappement (`char`)

Code	Signification
<code>\b</code>	Retour en arrière (<i>Backspace</i>)
<code>\t</code>	Tabulateur horizontal
<code>\n</code>	Saut de ligne (<i>Line-feed</i>)
<code>\f</code>	Saut de page (<i>Form-feed</i>)
<code>\r</code>	Retour de chariot (<i>Carriage-Return</i>)
<code>\"</code>	Guillemet
<code>\'</code>	Apostrophe
<code>\\</code>	Barre oblique arrière (<i>Backslash</i>)

Types caractères

60

❑ Il est possible d'utiliser directement le code du caractère en l'exprimant:

- soit sous **forme octale**, i.e `\nnn`.
- soit sous **forme hexadécimale** précédée de x, i.e `\xnnn`.

Exemple:

Caractère	En base octale	En base hexadécimale
'A'	'\101'	'\x41'
'!'	'\041'	'\x21'

Types flottants

61

- **Nombres en virgule flottante** (**float**, **double**)
 - Norme IEEE 754-1985
 - Précision
 - ✓ **float** : env. 6 chiffres significatifs
 - ✓ **double** : env. 15 chiffres significatifs
 - Valeurs littérales (**double** par défaut) : notation habituelle (***n.m***)
 - Suffixe **f** ou **F** pour type **float**
 - Suffixe **d** ou **D** pour type **double** (rarement nécessaire)
 - Notation exponentielle avec **e** ou **E** suivi de l'exposant

Attention : Les nombres en virgule flottante sont des **approximations** de nombres réels (tous les nombres réels ne peuvent pas être représentés de manière exacte).
Il faut en tenir compte dans les comparaisons et notamment dans les tests d'égalité (prendre en compte un "*epsilon*").

Types flottants

62

- Nombres en virgule flottante (**float, double**)

Exemples:

double x(1.); //x = 1. = 1.0

double y(1.4e3); //x = $1.4 \times 10^3 = 1.4 \times 1000 = 1400$

double z(-1.4e-3); //x = $-1.4 \times 10^{-3} = -1.4 \times 0.001 = 0.0014$

Types flottants

63

- Nombres en virgule flottante (**long double**)

- Suffixe **l** ou **L** pour type **long double**

- **Exemples:**

15.17891912345678L //pour type long double

-15.5e1200L //pour type long double

Types flottants

64

- Nombres en virgule flottante (**float**, **double**)

```
123.45      // valeur de type double (par défaut)
-4.032F     // valeur de type float
6.02e23     // 6.02 x 1023 de type double
-5.076E-2f  // -5.076 x 10-2 de type float
```

- Cas particuliers (problèmes numériques) :

```
1.2/0.0     // Infini
-5.1/0.0    // Moins l'infini
0.0/0.0     // Not a Number (NaN)
```

Limitations des types réels

65

Type	Contient	Taille	Valeurs (min à max)
float	Nombre en virgule flottante	4 octets	-3.4×10^{-38} à $+3.4 \times 10^{38}$
double	Nombre en virgule flottante	8 octets	-1.8×10^{-308} à $+1.8 \times 10^{308}$
long double	Nombre en virgule flottante	8 octets	-1.2×10^{4932} à $+1.2 \times 10^{4932}$

Types de base (bool-char)

66

Type	Contient	Taille	Valeurs
bool	Booléen	Même taille que le type int, parfois 1 sur quelques compilateurs	Deux valeurs : 'true' et 'false' Une conversion implicite (valant 0 ou 1) est faite par le compilateur lorsque l'on affecte un entier En réalité toute autre valeur que 0 est considérée comme égale à True.
char	Caractère	1 octet	-128 à +127

Les fonctions mathématiques

67

- La bibliothèque standard du C++ **cmath** fournit les fonctions mathématiques usuelles.
- **#include <cmath>**

Fonction	Description
sin(x)	sinus x (en radians)
cos(x)	cosinus x (en radians)
tan(x)	tangente x (en radians)
exp(x)	e^x
log(x)	$\ln(x)$, $x > 0$
ceil(x)	plus petit entier $\geq x$
floor(x)	plus grand entier $\leq x$
abs(x)	valeur absolue $ x $
pow(x,y)	x^y
sqrt(x)	racine carrée
...	...

<http://www.cplusplus.com/reference/cmath/>

Les fonctions mathématiques

68

Exemple:

```
#include <iostream> // sorties standards
#include <cmath>
using namespace std;

int main()
{
    double angle;
    double s;

    angle = 10 * 3.14159 / 180;
    s = sin(angle);
    cout << s;

    return 0;
}
```

Output:

0.173648

Les constantes mathématiques

69

- Les constantes de mathématique ne sont pas définies dans la norme C/C++. Pour les utiliser, vous devez d'abord définir `_USE_MATH_DEFINES` puis inclure `cmath`

Symbole	Expression	Valeur
M_PI	π	3.14159265358979323846
M_E	e	2.71828182845904523536
M_SQRT2	$\sqrt{2}$	1.41421356237309504880
M_PI_2	$\pi/2$	1.57079632679489661923
...

□ Syntaxe

```
#define _USE_MATH_DEFINES
```

```
#include <cmath>
```

Les constantes mathématiques

70

Exemple:

```
#include <iostream> // sorties standards
#define _USE_MATH_DEFINES
#include <cmath>
using namespace std;

int main()
{
    double angle_en_degrees;
    cout << "Entrez un angle en degrees: " << endl;
    cin >> angle_en_degrees;

    double angle_en_radians(M_PI * angle_en_degrees / 180);

    cout << "Sa valeur en radians vaut " << angle_en_radians << endl;
    cout << "Son cosinus vaut " << cos(angle_en_radians) << endl;

    return 0;
}
```

Les constantes mathématiques

71

Exemple: (suite)

```
Entrez un angle en degres:  
50  
Sa valeur en radians vaut 0.872665  
Son cosinus vaut 0.642788  
  
Process returned 0 (0x0)   execution time : 3.918 s  
Press any key to continue.
```

Conversions de Types

72

- Les conversions usuelles d'ajustement de type
 - `int -> long -> float -> double -> long double`
 - On peut bien sûr convertir directement un **int** en **double** ; par contre, on ne pourra pas convertir un **double** en **float** ou en **int**.

- **Exemple:**

```
int n; long p; float x;
```

```
n*p+x;    //conversion de n en long
```

```
           //le résultat de * est de type long
```

```
           //il est converti en float pour être additionné à x
```

```
           //ce qui fournit un résultat de type float
```

Conversions de Types

73

- **Les promotions numériques**
 - short -> int
 - bool -> int
 - char -> int

Les promotions numériques

74

■ Exemples:

1. **short** p1, p2, p3; **float** x;

p1*p2+p3*x; // promotion numérique short -> int

 // conversion d'ajustement de type int -> float

2. **char** c; **int** n;

c+1; // promotion numérique char -> int

c+n; // promotion numérique char -> int puis résultat en int

3. **bool** b1 = **true**, b2 = **false**;

cout << b1 + 3 ; // affiche 4

cout << b2 + 3 ; // affiche 3

Transtypage (Casting)

75

- Conversion de type explicite (au risque du programmeur)

- Syntaxe : **(*Type_de_destination*) Valeur_à_convertir**

Transtypage (Casting)

76

- **Exemples:**

```
int p1, p2;
```

```
double x, y;
```

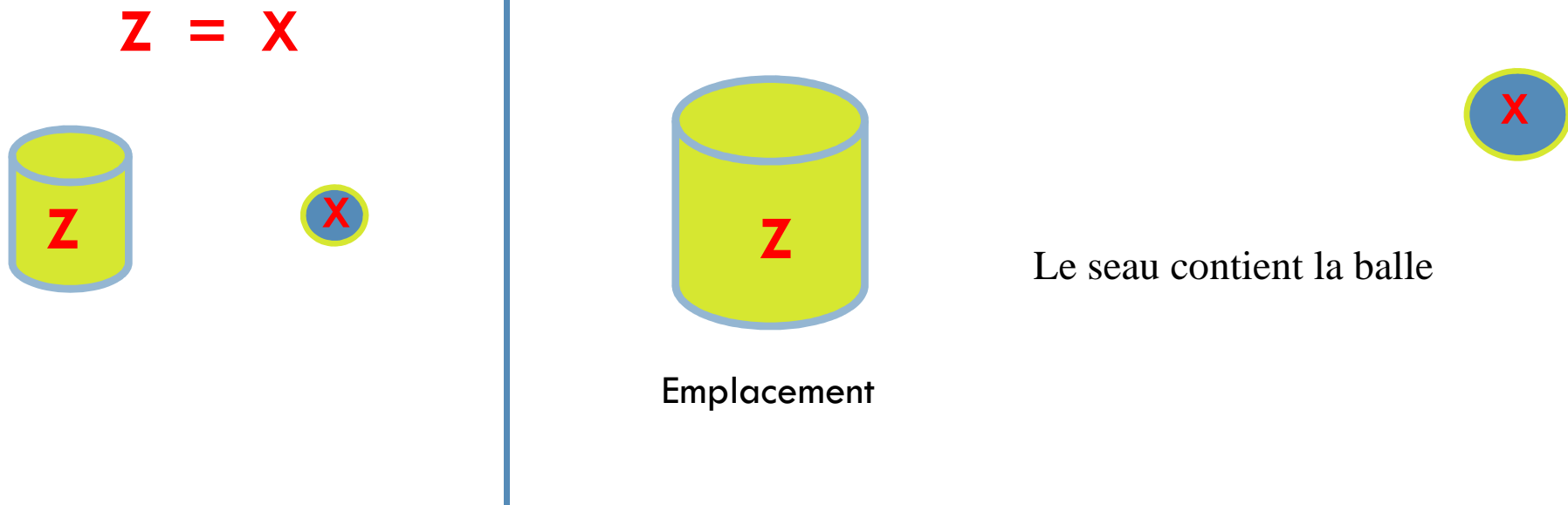
```
x = (double) (p1/p2);    // x aura comme valeur celle de l'expression  
                          // entière p1/p2 convertie en double.
```

```
y = (double) p1/p2;    // Si p1 = 5 et p2 = 2 alors y = 2.5
```

Affectation/Assignment

77

□ On utilise $=$ pour affecter des valeurs aux variables.



Affectation/Assignment

78

- De façon plus générale, une affectation suit le schéma:

`nom_de_variable = expression;`

- À droite du signe `=` dans une affectation se trouve une **expression**.
- Une **expression** calcule une valeur, qui doit être de même type que la variable.

- **Exemples d'expression:**

- 3
- $n * n$
- $n/4$
- $n * (n + 2) + 4 * n - 1$

Affectation/Assignment

79

- Enregistre dans l'opérande de gauche (variable) la valeur de l'opérande de droite (valeur de l'expression)
- Le type de la variable (opérande de gauche) doit être compatible avec le type de l'expression de droite (si nécessaire, conversion élargissante automatique)
- Le **type et valeur de retour** d'une expression d'affectation correspondent au type de la variable et, respectivement, à sa valeur après affectation
- **Attention** : Ne pas confondre l'opérateur d'affectation (=) avec celui d'égalité (==)

"prend pour valeur" et non pas "est égal à"

```
a = b + c;      // La variable a prend pour valeur le résultat de la somme (b+c)
a = b = c;      // Associativité à droite  $\Rightarrow$  a = (b = c)
t[i] = circle.center();
```

Affectation/Assignment

80

Exemple d'affectation:

```
string st;  
st = "IAP 6.092";
```

Exemple d'affectation combinée avec une déclaration:

Une affectation peut être combinée à une déclaration :

```
double badPi (3.14);  
bool isSeptember (true);
```

Affectation/Assignment

81

Exemple:

```
#include <iostream> // sorties standards

using namespace std;

int main()
{
    string s = "Déclaration et affectation";
    cout << s << endl;
    s = "Une Autre Affectation";
    cout << s << endl;

    return 0;
}
```

Output:

Déclaration et affectation
Une Autre Affectation

Affectation en C++11

82

- Noter bien que depuis C++11 une troisième syntaxe d'affectation est possible:

```
double x{1.5};
```


Les entrées-sorties standards de C++

Pour écrire à l'écran

84

Le flot **cout**

`cout` représente la
fenêtre Terminal

affiche la *valeur* de la variable `n`
(et non pas la lettre `n`)

ce qui est entre guillemets
(`"`) est affiché littéralement

les différents éléments à
afficher doivent être
séparés par le symbole
`<<`

```
cout << "La variable n contient " << n << "." << endl;
```

affiche:

La variable `n` contient 4.

fait un "retour à la ligne": le
prochain affichage se fera sur
la ligne suivante de la fenêtre
Terminal

Pour écrire à l'écran

85

Le flot **cout**

- ❑ Le flot **cout** est un flot de sortie prédéfini, connecté à la sortie standard stdout (l'écran).
- ❑ L'opérateur **<<** permet d'envoyer de l'information sur le flot **cout**, correspondant à l'écran.
- ❑ En générale, l'opérateur **<<** permet d'envoyer sur le flot **cout** la valeur d'une expression d'un type de base quelconque.
- ❑ On peut aussi utiliser **cout** pour afficher la valeur d'une **expression**:

```
cout << "Le carré de " << n << " est " << n * n << "." << endl;
```

Pour écrire à l'écran

86

Le flot **cout**

Exemple 1:

Considérons ces instructions :

```
int n = 20 ;
```

```
cout << "Valeur : " ;
```

```
cout << n ;
```

Elles affichent le résultat suivant :

```
Valeur : 20
```

Pour écrire à l'écran

87

Le flot **cout**

Exemple 1: (suite)

Les deux instructions :

```
cout << "Valeur : " ;
```

```
cout << n ;
```

peuvent se condenser en une seule :

```
cout << "Valeur : " << n ;
```

Pour écrire à l'écran

88

Remarque:

- ❑ `cout` et `endl` sont des mots réservés de la bibliothèque standard `std` (`using namespace std`), nous pouvons les nommer `std::cout` et `std::endl`.

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

Déroulement du programme pas-à-pas

89

Exemple :

```
int n(4);  
int n_carre;  
  
n_carre = n * n;  
  
cout << "La variable n contient " << n << "." << endl;  
cout << "Le carre de " << n << " est " << n_carre << "." << endl;  
cout << "Le double de n est " << 2 * n << "." << endl;
```

Output :

La variable n contient 4.

Le carre de 4 est 16.

Le double de n est 8.

Pour écrire à l'écran

90

Exercice 1:

❑ Qu'affiche ce programme?

```
int a(2);  
int b(1);  
  
b = a * (b + 2);  
  
cout << a << ", " << b << endl;
```

A: a, b

B: 1, 2

C: 2, 1

D: 2, 6

?

Pour écrire à l'écran

91

Exercice 2:

☐ Qu'affiche ce programme?

```
int a(1);  
int b(2);  
  
a = b;  
b = a;  
  
cout << a << ", " << b << endl;
```

A: 1, 1

B: 1, 2

C: 2, 2

D: 2, 1

?

Pour écrire à l'écran

92

Exercice 3:

☐ Qu'affiche ce programme?

```
int a(5);  
int b(a + 3);  
  
a = 1;  
  
cout << a << ", " << b << endl;
```

A: 5, 4

B: 1, 1

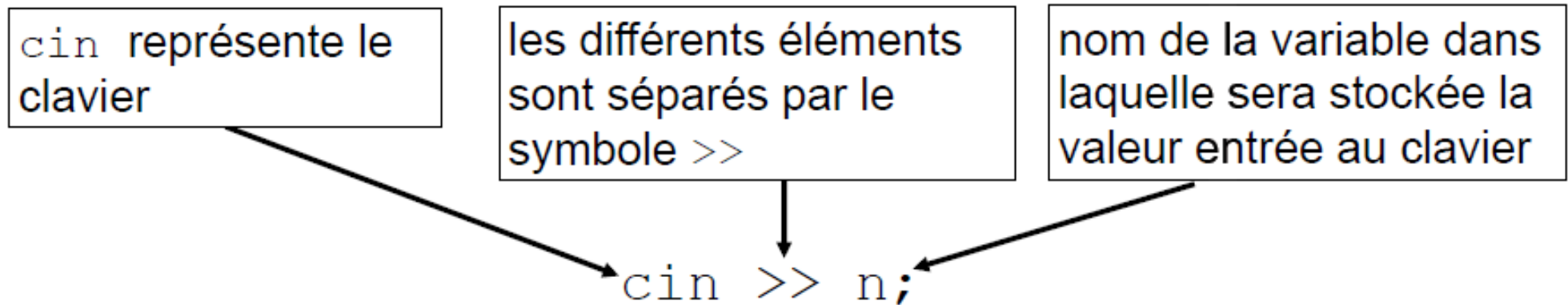
C: 1, 4

D: 1, 8

?

Lire une valeur au clavier

93



- ❑ **Attention**, uniquement des noms de variables peuvent figurer à droite du symbole `>>`.
- ❑ Le **flot `cin`** est un flot d'entrée prédéfini, connecté à l'entrée standard `stdin` (le clavier).

Lire une valeur au clavier

94

❑ On ne peut pas faire :

```
cin >> "Entrez un nombre" >> n;
```

Il faut faire:

```
cout << "Entrez un nombre" << endl;
```

```
cin >> n;
```

Lire une valeur au clavier

95

- On peut lire plusieurs valeurs à la suite :

`cin >> n1 >> n2 >> n3;`

Exemple:

```
int main() {  
    int n1 = 0, n2 = 0;  
    cout << "Saisir deux entiers" << endl;  
    cin >> n1 >> n2 ;  
    cout << n1 << ", " << n2 << endl;  
}
```

Déroulement du programme pas-à-pas

96

```
int n;  
  
cout << "Entrez une valeur pour n:";  
cin >> n;  
  
cout << "La variable n contient " << n << "." << endl;
```

Output:

Entrez une valeur pour n:

2

La variable n contient 2.

Opérateurs et expressions

Affectation avec opérateur

98

- Combinaison de l'affectation avec un opérateur (arithmétique ou orienté bits)
- Le type de l'opérande de gauche (variable) doit être compatible avec le type de l'expression de droite

`var op = expr`

est équivalent à

`var = var op (expr)`

- Opérateurs combinés : `+=` `-=` `*=` `/=` `%=`

Affectation avec opérateur

99

Exemple:

- `a += 3;` `// a = a + 3;`
- `b *= a;` `// b = b * a;`
- `x /= 3;` `// x=x/3`

Affectation d'une valeur décimale à une variable entière

100

❑ Quand on affecte une valeur décimale à une variable de type `int`, la partie fractionnaire est perdue.

❑ **Exemple:**

```
double x(1.5);
```

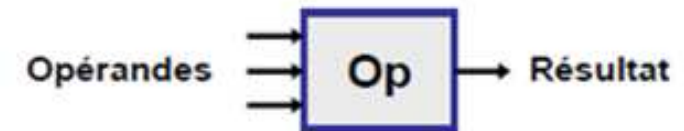
```
int n;
```

```
n = 3 * x;    //Le type de la valeur 4.5 est converti de double vers int.  
              //donc n=4
```

Les opérateurs et les expressions

101

- Les **opérateurs** sont des éléments syntaxiques qui effectuent certaines opérations en utilisant les valeurs de leurs **opérandes** (paramètres de l'opération).



- Opérateurs unaires** (monadiques) *1 opérande*
 - `-b` Opérateur monadique moins
 - `a++` Opérateur monadique de post-incrémentation
- Opérateurs binaires** (dyadiques) *2 opérandes*
 - `a * b` L'opérateur de multiplication est binaire
 - `y = z` L'affectation est également binaire
 - `(int)y` Transtypage (casting)
- Opérateurs ternaires** *3 opérandes*
 - Un seul opérateur ternaire
 - `x > y ? x : y` Retourne la valeur maximale entre x et y

Les opérateurs arithmétiques

102

Opérateurs arithmétiques à deux opérandes :

- addition $a + b$
- soustraction $a - b$
- multiplication $a * b$
- division a / b
- modulo $a \% b$

Remarque:

La division des entiers fournit un résultat tronqué et non arrondi.

Les opérateurs arithmétiques

103

Opérateurs arithmétiques à deux opérandes :

Exemples:

- $1/3 = 0$
- $7/3 = 2$
- $1/2.0 = 0.5$
- `double x; x = 1/2;` //l'expression $1 / 2$ est d'abord évaluée, elle vaut 0
//la valeur 0 est affectée à x donc $x = 0.0$

Les opérateurs arithmétiques

104

L'ordre des opérateurs suit les standards mathématiques

1. Parenthèses ()
2. Opérateurs unaires + et - ont la priorité la plus élevée
3. Multiplication (*), division (/) et modulo (%)
4. Addition (+) et soustraction (-)

Exemple:

■ $x + y * z$	équivalent à	$x + (y * z)$
■ $x * y + z \% t$	équivalent à	$(x * y) + (z \% t)$
■ $- x \% y$	équivalent à	$(- x) \% y$
■ $- x + y \% z$	équivalent à	$(- x) + (y \% z)$
■ $- x / - y + z$	équivalent à	$((- x) / (- y)) + z$
■ $- x / - (y + z)$	équivalent à	$(- x) / (- (y + z))$

Les opérateurs et les expressions

105

Opérateurs à un opérande :

- ☐ Opposé: $-X$
- ☐ Pré-incrémentation: $++X$
- ☐ Post-incrémentation: $X++$
- ☐ Pré-décrémentation: $--X$
- ☐ Post-décrémentation: $X--$

Les opérateurs et les expressions

106

Opérateurs à un opérande :

- Si l'on prend l'exemple suivant :

```
int i(3);
```

```
int j(i);      // i et j ont la même valeur
```

```
int k(0);
```

```
int l(0);
```

```
k = ++i;      // opérateur préfixé
```

```
l = j++;      // opérateur postfixé
```

- A l'issue de ce bout de code:

i et j auront tous les deux la valeur 4, mais k aura la valeur 4 alors que l aura la valeur 3.

Les opérateurs et les expressions

107

Exemple 1:

<pre>int i = 0; i++;</pre>		<pre>int i = 0; i = i + 1;</pre>
<pre>int t = i++;</pre>		<pre>int t = i; i = i + 1;</pre>
<pre>int s = ++i;</pre>	équivalent à	<pre>i = i + 1; int s = i;</pre>
<pre>int r = --i;</pre>		<pre>i = i - 1; int r = i;</pre>
<pre>r += i;</pre>		<pre>r = r + i;</pre>
<pre>s *= i; i /= 2;</pre>		<pre>s = s * i; i = i / 2;</pre>

Les opérateurs et les expressions

108

Exemple 2:

La richesse de la notion d'expression en C++ fait que l'expression régissant le choix peut réaliser certaines actions. Ainsi :

```
if (++i < limite) cout << "OK";
```

est équivalent à :

```
i = i + 1 ;
```

```
if (i < limite) cout << "OK";
```

Par ailleurs :

```
if ( i++ < limite ) .....
```

est équivalent à :

```
i = i + 1 ;
```

```
if ( i-1 < limite ) .....
```

Les opérateurs relationnels

109

Opérateurs relationnels ou opérateur de comparaison :

Math	C++	Description
$>$	<code>></code>	Supérieur à
\geq	<code>>=</code>	Supérieur ou égal à
$<$	<code><</code>	Inférieur à
\leq	<code><=</code>	Inférieur ou égal à
$=$	<code>==</code>	Egalité
\neq	<code>!=</code>	Inégalité

Attention :

Pour les types références (objets et tableaux) les opérateurs d'égalité et d'inégalité comparent les références et non les valeurs référencées.

Les opérateurs logiques

110

- **&&** : ET logique (**and**)
- **||** : OU logique (**or**)
- **!** : Négation (**not**)

Les opérateurs logiques

111

Ordre de priorité

- ▣ ! précède && précède ||.
- ▣ L'opérateur ! a une priorité supérieure à celle de tous les opérateurs arithmétiques binaires et aux opérateurs relationnels.
- ▣ && et || sont de priorité inférieure aux opérateurs arithmétiques ou relationnels.
- ▣ () peuvent rendre l'expression plus claire.

Exemple

- | | | |
|----------------------------|--------------|------------------------------|
| ▪ $a < b \ \&\& \ c < d$ | équivalent à | $(a < b) \ \&\& \ (c < d)$ |
| ▪ $a < b \ \ c < d$ | équivalent à | $(a < b) \ \ (c < d)$ |
| ▪ $a \ \ !b \ \&\& \ c$ | équivalent à | $a \ \ ((!b) \ \&\& \ c)$ |

Les opérateurs et les expressions

112

L'opérateur à trois opérandes:

condition ? vrai : faux

Exemple:

$y < 5 ? 4 * y : 2 * y$

Si *condition* est vrai, alors on retourne l'évaluation de l'expression *vrai*, sinon on retourne celle de *faux*.

Les opérateurs et les expressions

113

Exemple :

```
#include <iostream> // sorties standards
using namespace std;
int main() {

    double score(1.0 + 2.0 * 3.0);
    cout << score << endl;

    double copy(score);
    copy = copy / 2.0;

    cout << copy << endl;
    cout << score << endl;

    return 0;
}
```

Output:

7

3.5

7

Affichage sur la console

114

- Pour afficher une valeur littérale ou le contenu d'une variable sur la console de sortie, on utilise les lignes de code suivantes :

- **cout <<** // Affichage (reste sur la même ligne)
- **cout << + endl** // Affichage et retour à la ligne

- Exemples :

```
cout << "Résultats" << endl;  
cout << "-----" << endl;
```

Résultats

```
int size(123);  
char unit ('m');  
cout << "Longueur : " ;  
cout << size;  
cout << " " ;  
cout << unit << endl;
```

Longueur : 123 m

Même affichage mais en utilisant l'opérateur << :

```
int size(123);  
char unit ('m');  
cout << "Longueur : " << size << " " << unit << endl;
```

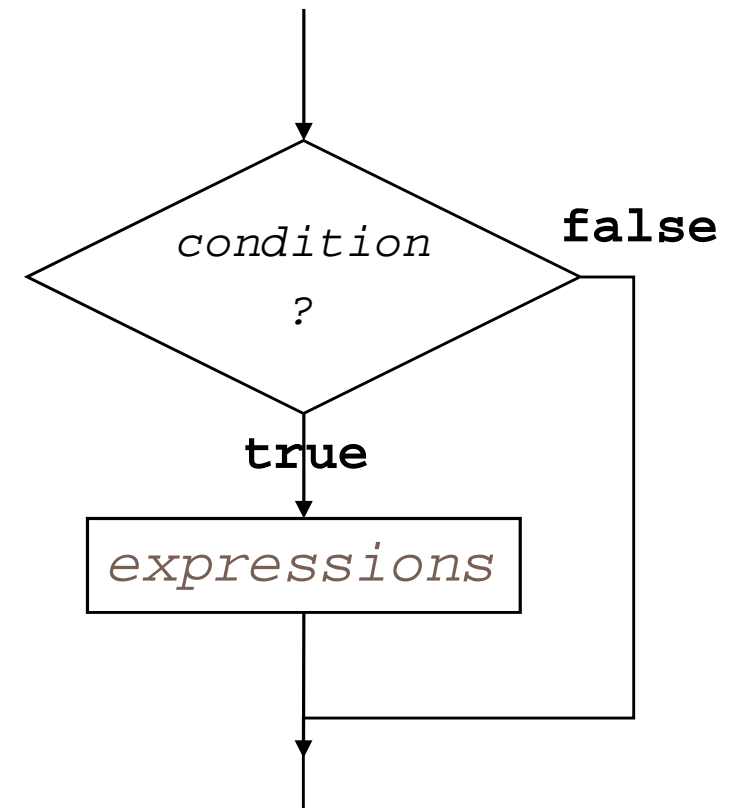

Les instructions de contrôle

Les instructions de contrôle en C++

116

Décisions

- Interruption de la progression linéaire du programme suite à une décision
 - ▣ Basée sur une **condition** qui peut être
 - **true**
 - **false**



Les instructions de contrôle en C++

117

Décisions

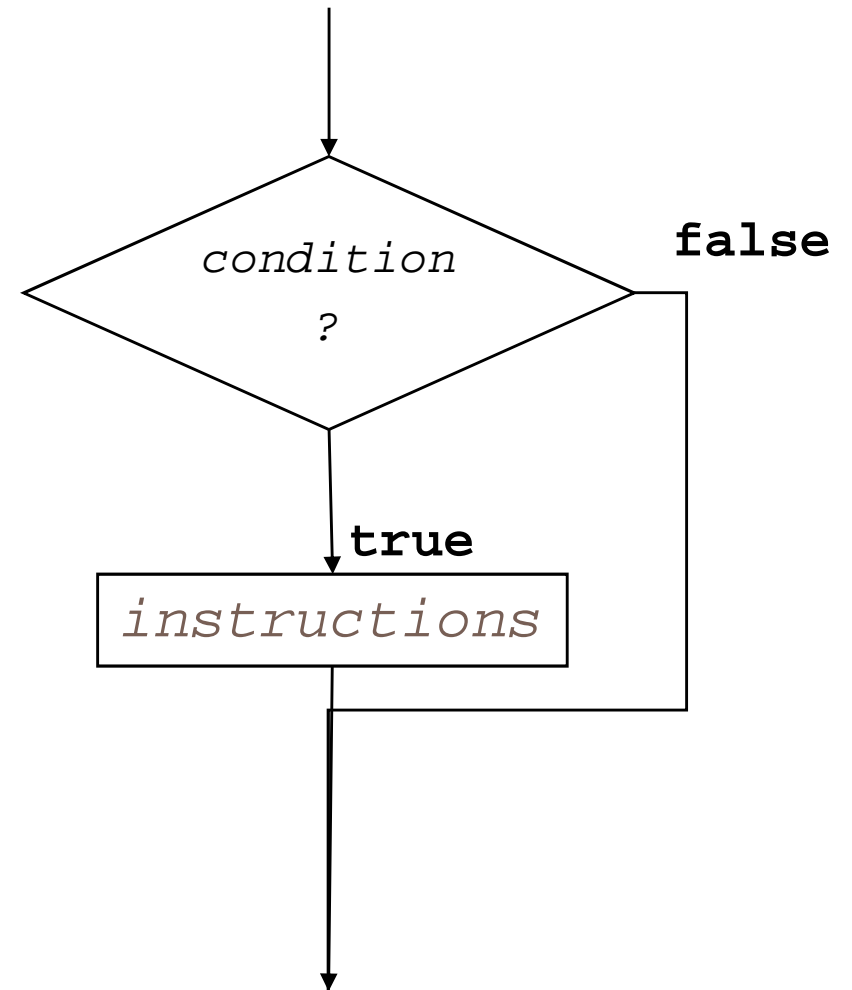
- Conditions basées sur des opérateurs relationnels
 - ▣ e.g. « égal à », « supérieur à »...
- Opérateurs relationnels combinées avec opérations logiques
 - ▣ e.g. « ET », « OU »...
- Traduction en code par des structures:
 - ▣ **if**
 - ▣ **while**
 - ▣ **for**
 - ▣ **do-while**

Les instructions de contrôle en C++

118

L'instruction conditionnelle if

```
if (condition) {  
    Bloc d'instructions;  
}
```



- L'instruction **if** fait apparaître une **condition** entre parenthèses.
- **Attention:** la condition est toujours entourée de parenthèses.

Les instructions de contrôle en C++

119

L'instruction conditionnelle if

Exemple :

```
if (age >= 18) status = "Adult";
```

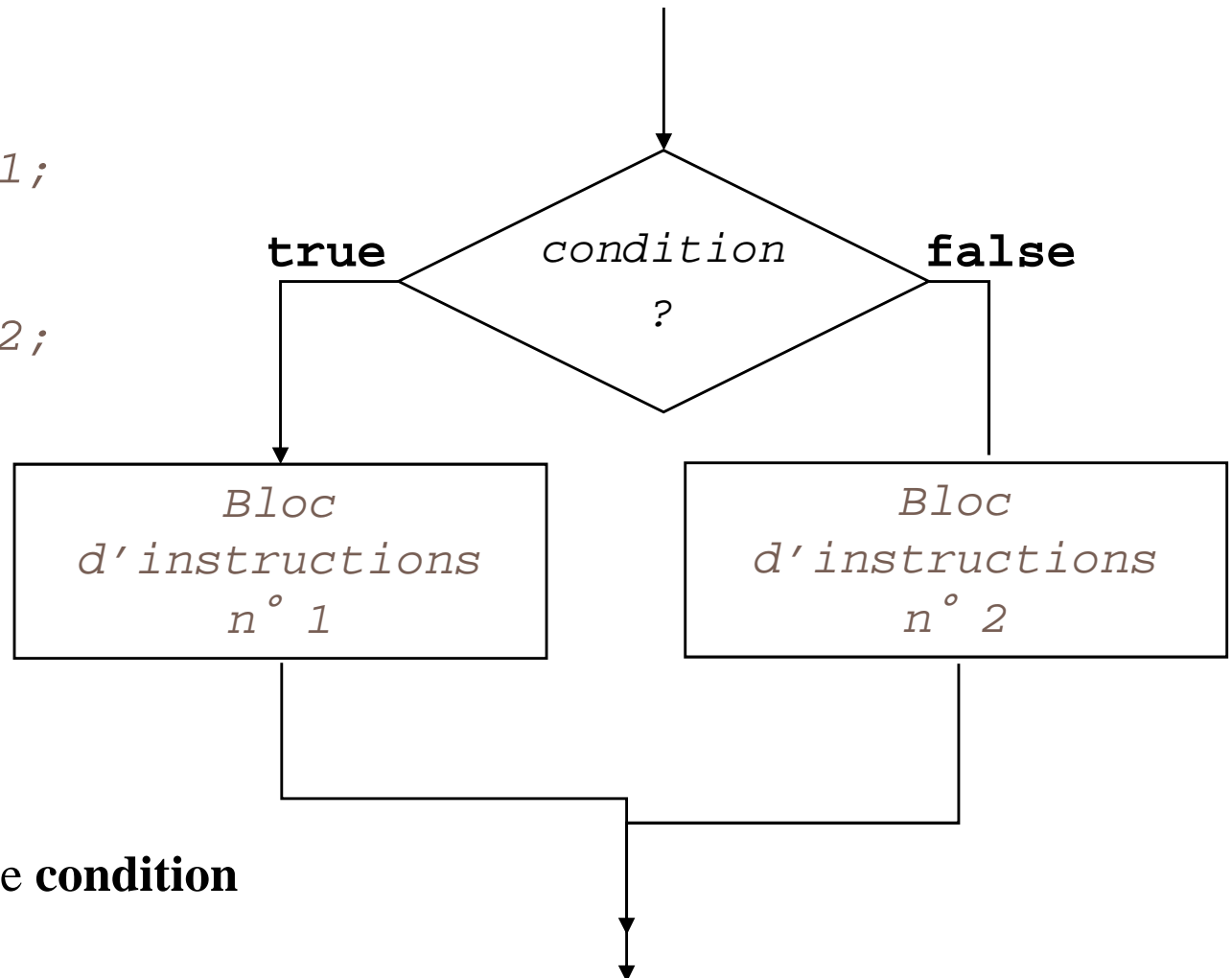
```
if ((age < 16) || isStudent) {  
    canHaveAllocations = true;  
    amount = 400;  
}
```

Les instructions de contrôle en C++

120

L'instruction d'alternative if...else

```
if (condition) {  
    Bloc d'instructions n°1;  
} else {  
    Bloc d'instructions n°2;  
}
```




- L'instruction **if** fait apparaître une **condition** entre parenthèses.
- **Attention:** la condition est toujours entourée de parenthèses.

Les instructions de contrôle en C++

121

L'instruction d'alternative if...else

Exemple 1:



```
if (r >= 0) {  
    carre = sqrt(r);    //Cette instruction sera exécutée si la condition est vraie.  
} else {  
    cout<<"Erreur";    //Cette instruction sera exécutée si la condition est fausse.  
}
```

Les instructions de contrôle en C++

122

L'instruction d'alternative if...else

Exemple 2:

```
if (average >= 10)
    cout << "Pass"; //Cette instruction sera exécutée si la condition est vraie.
else {
    cout << "Fail";
    repeat = true;
}
```

N. B: Quand un bloc contient une seule instruction, il n'est pas obligatoire d'utiliser des accolades.

Les instructions de contrôle en C++

123

Instruction conditionnelle if

□ Lequel est correct?

```
if (vitesse = 160) {  
    points == -8;  
}
```

```
if (vitesse == 130) {  
    points = -6;  
}
```

```
if (vitesse = 110) {  
    points = -4;  
}
```

□ Evidemment...

```
if (vitesse == 130) {  
    points = -6;  
}
```

□ Ne pas confondre

□ Affectation =

□ Égalité ==

Les instructions de contrôle en C++

124

Instruction conditionnelle if

```
int x(1);  
int y(2);  
  
if (x == y) {  
    cout << "Cas 1" << endl;  
} else {  
    cout << "Cas 2" << endl;  
}  
if (2 * x == y) {  
    cout << "y est égal au double de x." << endl;  
}
```

Output:

Cas 2

y est égal au double de x.

Instruction conditionnelle if

125

Exemple avec l'opérateur logique: and (ou &&)

```
cout << "Entrez un nombre entre 1 et 10 : " << endl;  
cin >> n;  
if ((n >= 1) and (n <= 10))  
    cout << "correct" << endl;  
else  
    cout << "incorrect" << endl;
```

Instruction conditionnelle if

126

□ Exemple avec l'opérateur logique: and (ou &&)

```
string type = "Ouragan";  
int vent;  
  
cout << "Donner la vitesse du vent Km/h" << endl;  
cin >> vent;  
if (vent >= 0 && vent < 64)  
    type = "Petite Tempête";  
else if (vent < 118)  
    type = "Tempête";  
else if (vent < 153)  
    type = "Violente tempête";  
else  
    type = "Ouragan";  
cout << "Le type du vent est " << type << endl;
```

Instruction conditionnelle if

127

Exemple avec l'opérateur logique: or (ou ||)

```
cout << "Entrez deux valeurs : " << endl;
cin >> m >> n;
if ((m >= 0) or (n >= 0))
    cout << "Au moins une valeur est positive" << endl;
else
    cout << "Les deux valeurs sont négatives" << endl;
```

Instruction conditionnelle if

128

Exemple avec l'opérateur logique: or (ou ||)

```
int beaufort;  
cin >> beaufort;  
if (beaufort < 0 || beaufort > 12)  
    cout << "Entrée erronée";  
else  
    fait quelque chose
```

Instruction conditionnelle if

129

Opérateurs logiques:

Rappel:

- Pour le ET logique (and):
les deux conditions doivent être vraies;
- Pour le OU logique (or):
au moins l'une des conditions doit être vraie.

Instruction conditionnelle if

130

Exemple avec l'opérateur : not (ou !)

a	!a
true	false
false	true

```
int beaufort;  
cin >> beaufort;  
  
if (!(beaufort > 9))  
    cout<< "Sortons";  
else  
    cout<< "Au plumard";
```


Les instructions de contrôle en C++

131

Ces deux codes sont équivalents:

1.

```
if (a < b)
```

```
    min = a;
```

```
else
```

```
    min = b;
```

2.

```
min = a < b ? a : b;
```

Emboitement d'instructions if...else

132

- L'instruction **if** / **else** peut contenir dans chacune de ses deux branches une autre instruction **if** / **else** qui peut elle-même contenir une autre instruction ...
- On peut ainsi avoir plusieurs niveaux d'emboîtement formant une logique plus ou moins complexe.
- En l'absence de blocs d'instructions (**{...}**), la **clause else se rapporte toujours au if précédent** (sans **else**) le plus proche (l'indentation ne change pas la logique du code !).

```
if (x >= 2)
    if (x <= 20)
        status = 1;
else
    if (x <= 10)
        status = 2;
    else
        status = 3;
```

```
if (x >= 2)
    if (x <= 20)
        status = 1;
    else
        if (x <= 10)
            status = 2;
        else
            status = 3;
```

```
if (x >= 2) {
    if (x <= 20)
        status = 1;
}
else {
    if (x < 0)
        status = 0;
    else
        status = 2;
}
```

L'indentation est importante pour la lisibilité du code !

Les instructions de contrôle en C++

133

L'instruction d'alternative if...else if...else...

```
if (condition1) {  
    Bloc d'instructions n°1;  
} else if (condition2) {  
    Bloc d'instructions n°2;  
}  
else {  
    Bloc d'instructions n°3;  
}
```

- L'instruction **if** fait apparaître une **condition** entre parenthèses.
- **Attention:** la condition est toujours entourée de parenthèses.

Imbrication if...else...

134

- Lorsque l'on doit choisir une instruction (ou un bloc d'instructions) parmi plusieurs (groupes d'instructions mutuellement exclusifs) il faut imbriquer les clauses `if else if` sur plusieurs niveaux.

```
if (n==1) {  
    ...                // Code executed if n is 1  
}  
else {  
    if (n==2) {  
        ...            // Code executed if n is 2  
    }  
    else {  
        if (n==3) {  
            ...        // Code executed if n is 3  
        }  
        else {  
            if (n==4) {  
                ...    // Code executed if n is 4  
            }  
            else {  
                ...    // Code executed otherwise  
            }  
        }  
    }  
}
```

Imbrication if...else if...

135

- Si le nombre d'alternatives est grand, l'indentation provoque un décalage du code vers la droite qui peut devenir gênant.
- C'est pourquoi l'on préférera, dans ce cas, la disposition suivante :

```
if      (n==1) {  
    ...           // Code executed if n is 1  
}  
else if (n==2) {  
    ...           // Code executed if n is 2  
}  
else if (n==3) {  
    ...           // Code executed if n is 3  
}  
else {  
    ...           // Code executed otherwise  
}
```

Remarque : Il ne s'agit pas d'une nouvelle syntaxe mais simplement d'une mise en page plus lisible (sans cumul de l'indentation).

Les instructions de contrôle en C++

136

Imbrication des instructions *if*

- Un *else* se rapporte toujours au dernier *if* rencontré auquel un *else* n'a pas encore été attribué.

Instruction switch

137

```
switch ( expression ) {  
    case const1 : suite_instructions_1  
    case const2 : suite_instructions_2  
    ...  
    default : suite_instructions_default  
}
```

- Instruction de sélection multiple parmi une liste d'instructions (indexées) et dont le **point d'entrée** est déterminé par la valeur d'une expression (constante) de type `int`, `char` ou d'un type énuméré (`enum`)
- A partir du point d'entrée, l'exécution se poursuit ensuite dans les instructions suivantes (celles des autres clauses `case`) jusqu'à la fin de l'instruction `switch` ou jusqu'à l'instruction `break` qui interrompt l'instruction `switch`

Instruction switch

138

```
n = ...           // Variable of type           int, char or enum
switch (n) {
  case 1 :
    Instruction;  // Start here if n is 1
    Instruction;
    ...
  case 5 :
  case 6 :
    Instruction;  // Start here if n is 5 or 6
    Instruction;
    ...
  case 8 :
    Instruction;  // Start here if n is 8
    Instruction;
    ...
  default :
    Instruction;  // Start here if n ≠ {1, 5, 6, 8}
    Instruction;
    ...
}
```

● Point d'entrée

A partir du point d'entrée,
toutes les instructions qui
suivent sont exécutées jusqu'à
la fin de l'instruction switch
ou jusqu'à l'instruction break

Instruction switch

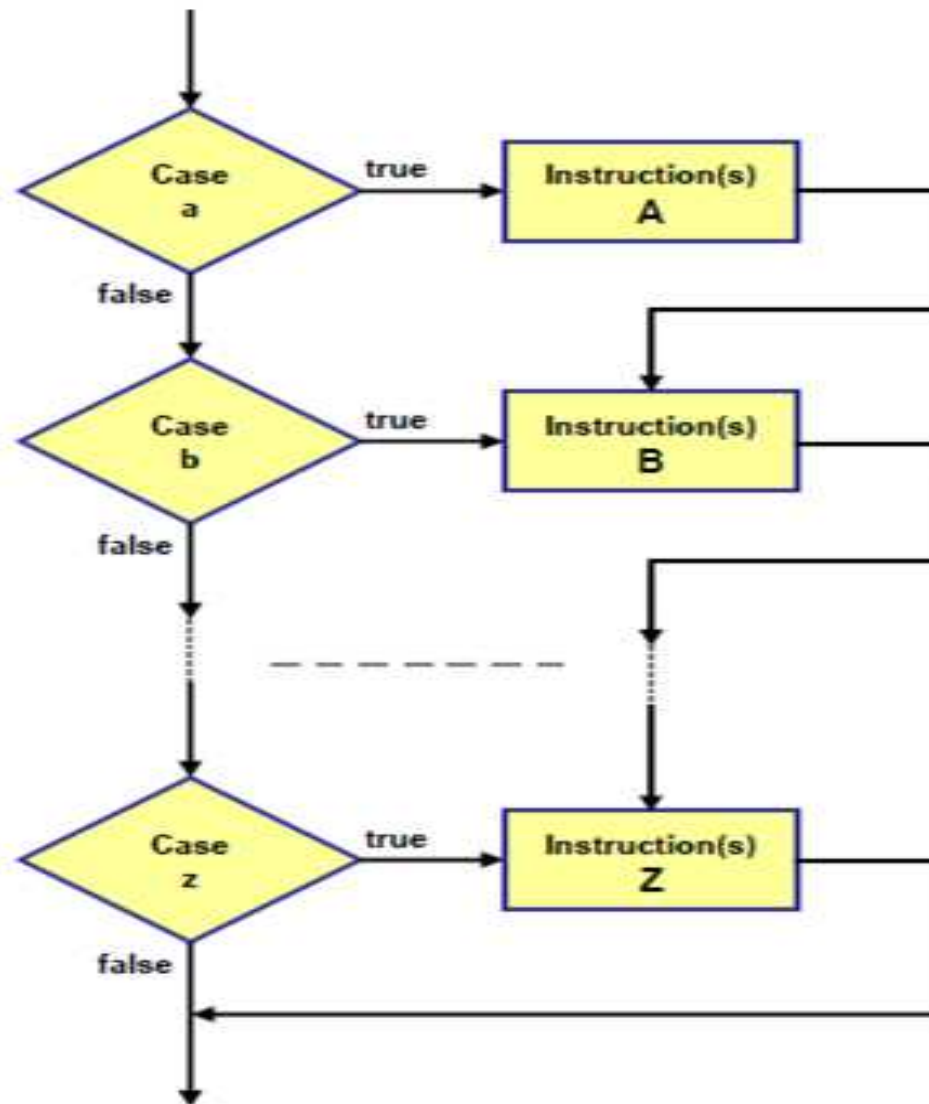
139

- Le mot clé `default` définit un point d'entrée qui est sélectionné si la valeur de l'expression ne correspond à aucune valeur dans la liste des clauses `case`. Le mot clé `default` est optionnel et est généralement placé comme dernière clause (ce n'est pas imposé par le langage mais fortement recommandé).
- Plusieurs clauses `case` peuvent étiqueter la même suite d'instructions (même point d'entrée pour plusieurs valeurs différentes).
- Les valeurs associées aux clauses `case` doivent être des valeurs ou expressions constantes (évaluables par le compilateur).
- Toutes les valeurs des clauses `case` doivent être différentes.

L'instruction `switch` est une instruction de bas niveau qui peut être remplacée par un enchaînement d'instructions `if...else`. Les instructions `if...else` offrent, en outre, davantage de flexibilité (expressions de types quelconques, utilisation de variables et d'expressions relationnelles et logiques dans les clauses de sélection) et n'imposent pas l'utilisation quasi systématique d'instructions d'interruption (`break`, ...).

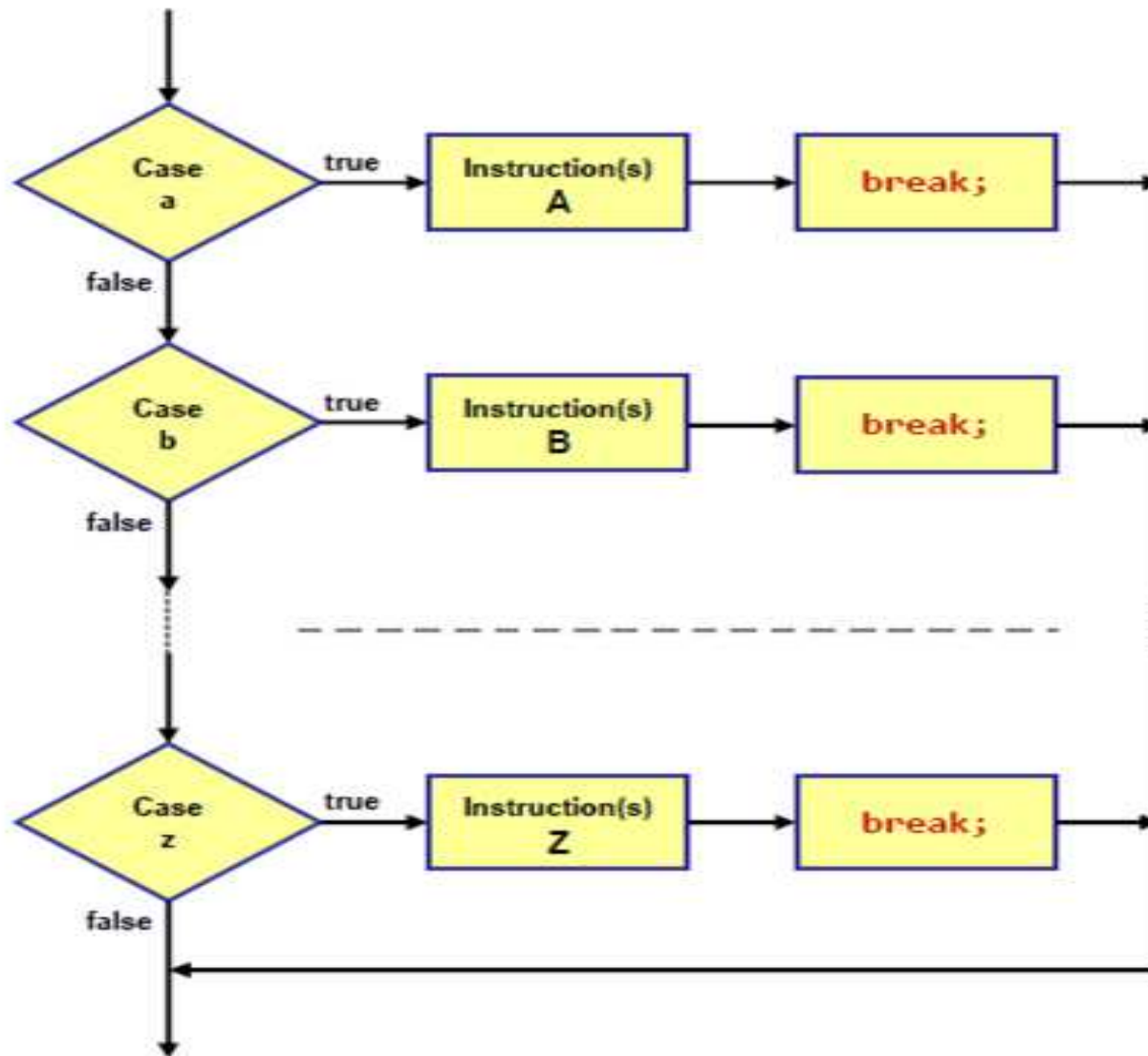
Instruction switch

140



Instruction switch

141



Instruction switch

142

```
int month, nbDays;
month = . . .
switch (month) {
    case 4 :
    case 6 :
    case 9 :
    case 11 :
        nbDays = 30; break;
    case 2 :
        nbDays = 28; break; // Années bissextiles non considérées
    default :
        nbDays = 31; break;
}
```

Conseil : Sauf cas exceptionnels, placez systématiquement une instruction **break** à la fin de chaque clause **case** car il est rare que l'on souhaite l'exécution des instructions des clauses **case** qui suivent celles du point d'entrée.

Instruction switch

143

□ Switch avec const

```
const int LIMITE = 10;
```

```
.....
```

```
switch (n){
```

```
case LIMITE-1 : .....
```

```
case LIMITE : .....
```

```
case LIMITE+1 : .....
```

```
}
```

Instruction switch

144

Exemple:

```
char grade = 'D';
switch(grade) {
    case 'A' :
        cout << "Excellent!" << endl;
        break;
    case 'B' :
    case 'C' :
        cout << "Bien fait" << endl;
        break;
    case 'D' :
        cout << "You passed" << endl;
        break;
    case 'F' :
        cout << "Better try again" << endl;
        break;
    default :
        cout << "Grade Invalide" << endl;
}
cout << "Your grade is " << grade << endl;
```

Instruction switch

145

Exemple: (suite)

Output:

You passed

Your grade is D

Itérations/Boucles en C++

146

- On appelle **boucle** ou **itération** l'exécution multiple d'une ou de plusieurs instructions **en fonction de la situation** (exprimée sous la forme d'une condition) prévalant à l'instant considéré.
- On trouve **trois formes** de réalisation:
 - L'instruction **while** qui permet de répéter un nombre quelconque de fois (0 ou plus) l'exécution d'une instruction ou d'un bloc d'instructions. Le nombre de répétitions n'est pas forcément connu au moment d'entrer dans la boucle.
 - L'instruction **do ... while** qui permet de répéter un nombre quelconque de fois (1 ou plus) l'exécution d'une instruction ou d'un bloc d'instructions. Le nombre de répétitions n'est pas forcément connu au moment d'entrer dans la boucle.
 - L'instruction **for** qui permet de répéter un nombre quelconque de fois (0 ou plus) l'exécution d'une instruction ou d'un bloc d'instructions. Le nombre de répétitions est généralement connu au moment d'entrer dans la boucle (mais ce n'est pas toujours le cas).

Les instructions de contrôle en C++

147

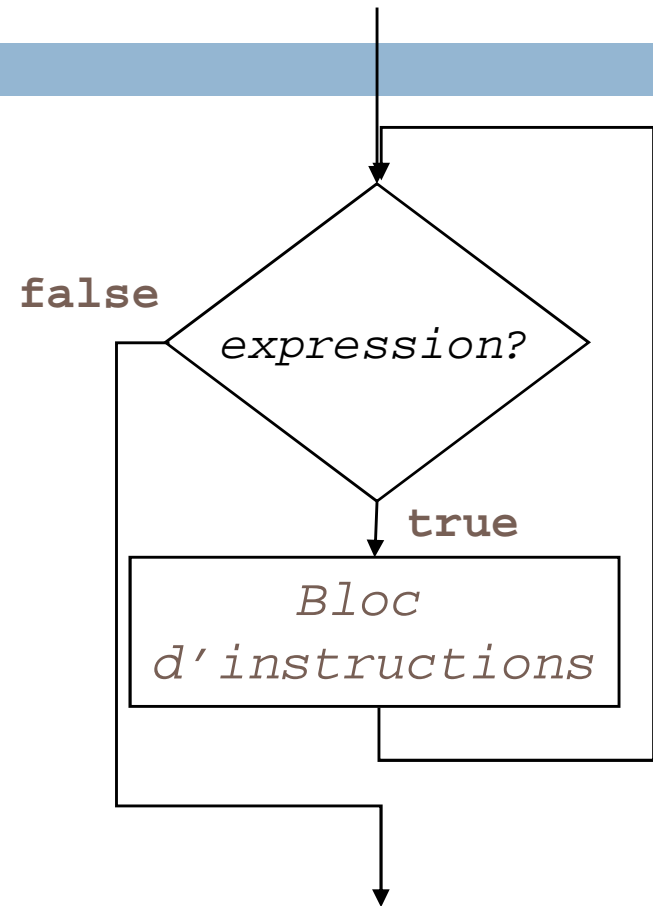
L'instruction *while*

□ Syntaxe de la boucle *while*

```
while (expression) {  
    bloc d'instructions;  
}
```

■ Instruction itérative

- L'expression booléenne est évaluée
- Si elle est vraie, l'instruction est exécutée
- Puis l'expression booléenne est évaluée à nouveau
- *etc...*
- L'instruction *while* se termine si l'expression booléenne est fausse



Les instructions de contrôle en C++

148

L'instruction *while*

- La boucle la plus souvent rencontrée

```
i = valeurInitiale; // Initialisation
while (i <= valeurFinale)
{
    bloc d'instructions;
    i++; //Incréméntation
}
```

Les instructions de contrôle en C++

149

L'instruction *while*

□ Exemple

```
int i=0;
while (i<=9){
cout << i << endl ;
i++;
}
```

Les instructions de contrôle en C++

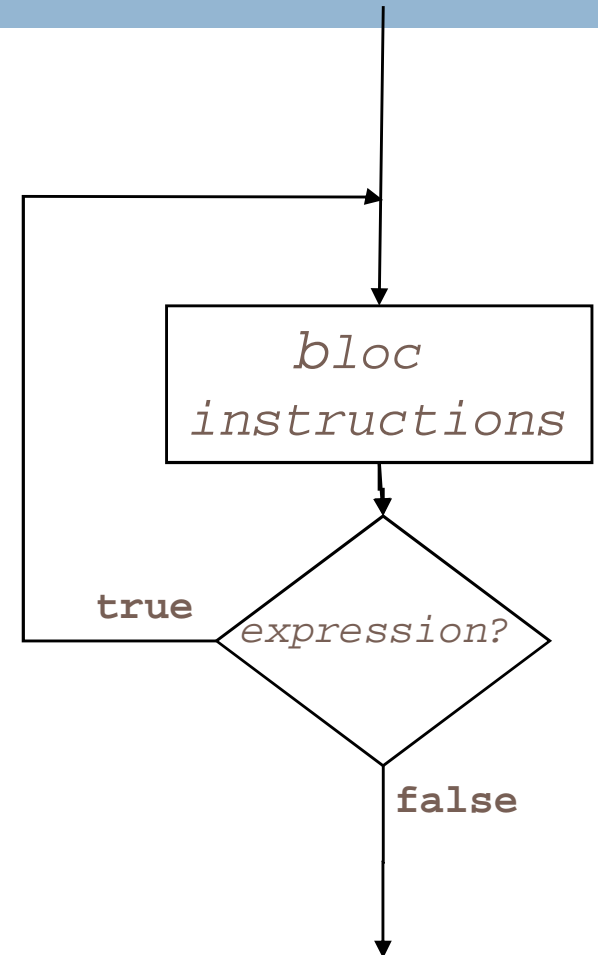
150

Syntaxe de l'instruction *do...while*

```
do{  
    bloc d'instructions;  
}while (expression);
```

■ Instruction itérative

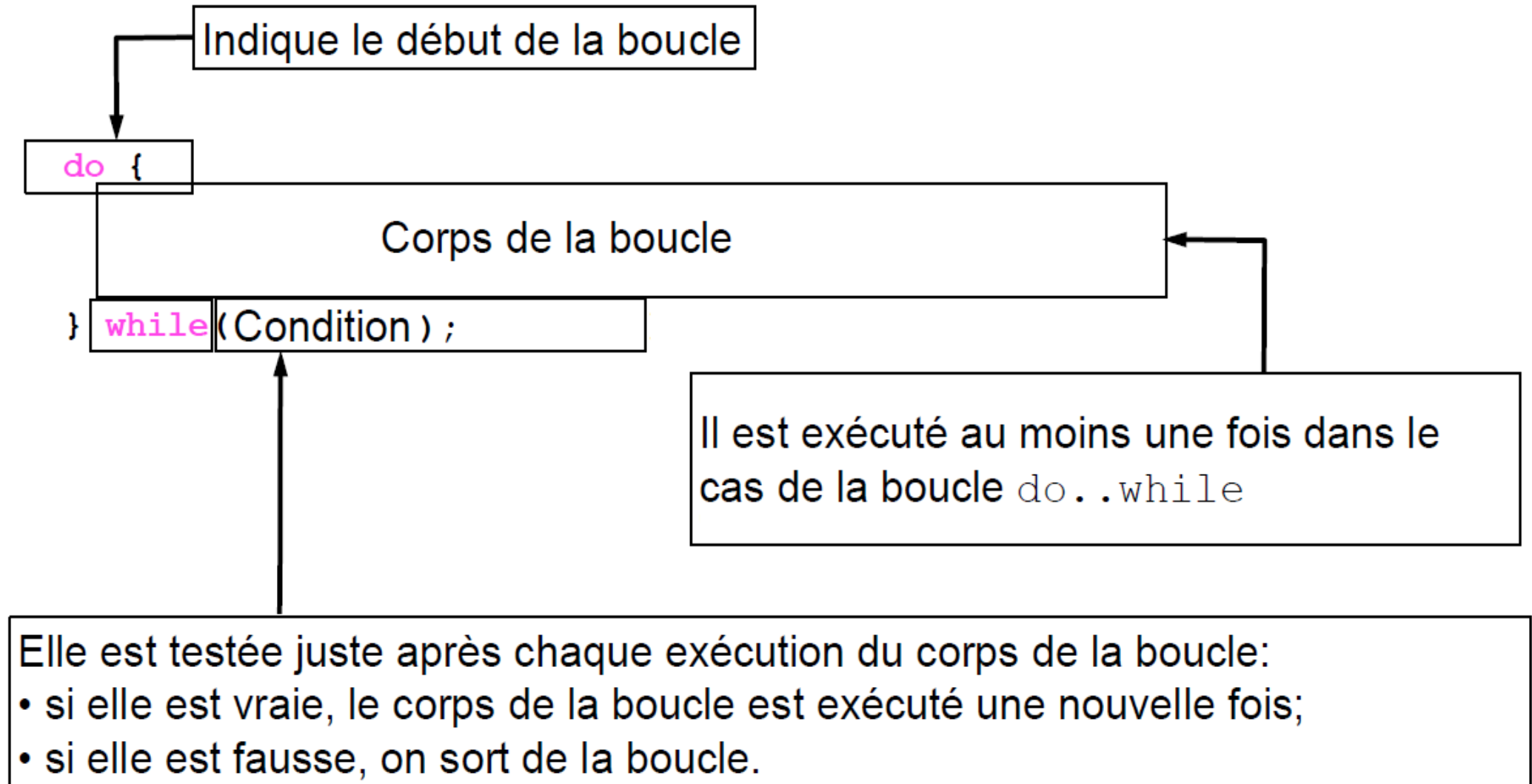
- Exécute l'instruction
- Évalue l'expression booléenne
- Si elle est vraie, l'instruction est exécutée à nouveau
- Évalue à nouveau l'expression booléenne
- etc...
- L'instruction **do** se termine si l'expression booléenne est fausse



Les instructions de contrôle en C++

151

Explication de l'instruction `do...while`



Les instructions de contrôle en C++

152

L'instruction *do...while*

Exemple

```
int nbre ;  
do{  
    cout << "Donnez un nombre positif (nbre >0) : " << endl;  
    cin >> nbre ;  
    cout << "Vous avez fourni : " << nbre << "\n" ;  
}while (nbre <= 0) ;  
cout << "Réponse correcte" ;
```

Les instructions de contrôle en C++

153

L'instruction *do...while*

Exemple

```
int nbre ;  
do{  
    cout << "Donnez un nombre positif (nbre >0) : " << endl ;  
    cin >> nbre ;  
}while (cout << "Vous avez fourni : " << nbre << "\n" , nbre <= 0) ;  
cout << "Réponse correcte" ;
```

Les instructions de contrôle en C++

154

L'instruction *do...while*

Exemple

```
int nbre ;  
do{  
    cout << "Donnez un nombre positif (nbre >0) : " << endl ;  
}  
while (cin >> nbre ,cout << "Vous avez fourni : " << nbre << "\n",nbre <= 0);  
cout << "Réponse correcte" ;
```


Les instructions de contrôle en C++

155

L'instruction *do...while*

Exemple

```
int nbre ;  
do{  
}  
while (cout << "Donnez un nombre positif (nbre >0) : " , cin >> nbre ,  
       cout << "Vous avez fourni : " << nbre << "\n", nbre <= 0);  
cout << "Réponse correcte" ;
```

Les instructions de contrôle en C++

156

Comparaison entre *while* et *do...while*

Exemple :

```
int i(10);  
do {  
    cout << "Hi" << endl;  
} while (i < 1);  
affichera une fois Hi.
```

```
int i(10);  
while (i < 1) {  
    cout << "Hi" << endl;  
}  
n'affichera rien.
```

Dans les 2 cas: la condition $i < 1$ est fausse.

Les instructions de contrôle en C++

157

Erreurs classiques

Remarque:

❑ Il n'y a pas de ; à la fin du **while...**:

```
while (i < 1); // !!
```

```
++i;
```

sera interprété comme

```
while(i < 1)
```

```
;
```

```
++i;
```

❑ En revanche, il y a un point-virgule à la fin du do..while:

```
do {
```

```
++i;
```

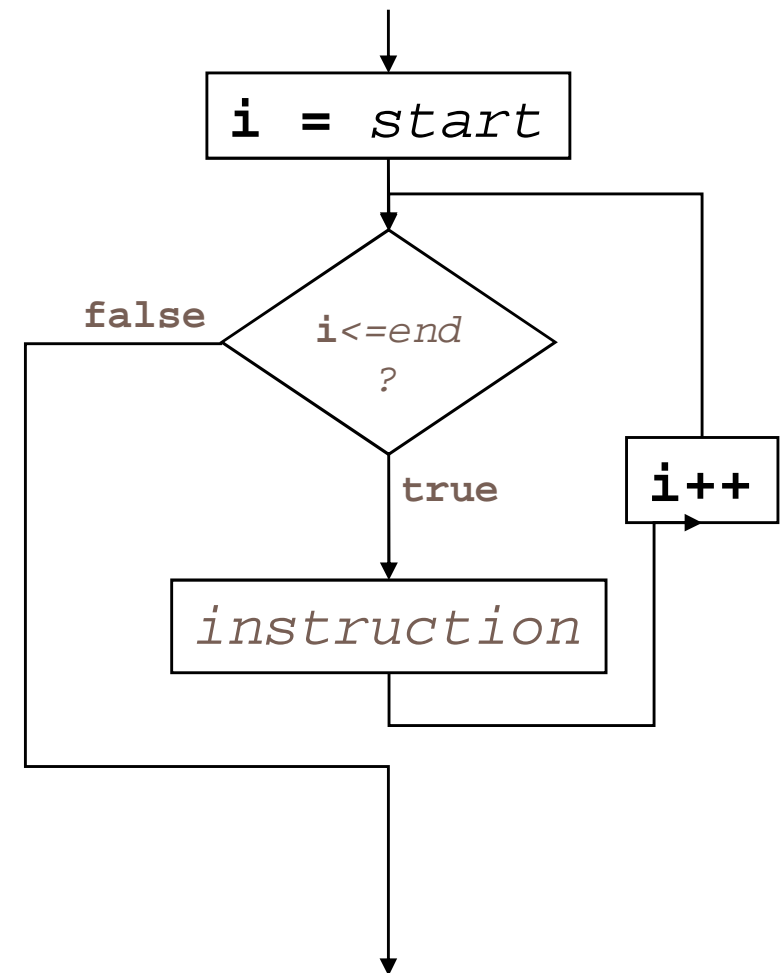
```
} while(i < 1);
```

Les instructions de contrôle en C++

158

Syntaxe de l'instruction *for*

```
for(déclaration_et_initialisation; condition; incrément)  
{  
    bloc d'instructions;  
}
```



L'instruction for

159

Remarque:

- ❑ L'initialisation, la condition, et l'incrémentation sont séparées par des points-virgules.
- ❑ Si la condition ne devient jamais fausse, les instructions dans la boucle sont répétées indéfiniment !
- ❑ Comme pour le `if`, les accolades ne sont obligatoires que si plusieurs instructions doivent être répétées.

Les instructions de contrôle en C++

160

L'instruction for

- La première expression correspond à l'initialisation d'un compteur;
 - Elle est évaluée (une seule fois) avant d'entrer dans la boucle.
- La deuxième expression correspond à la condition de poursuite:
 - Elle conditionne la poursuite de la boucle. Elle est évaluée avant chaque parcours.
- La troisième expression correspond à l'incrémentation du compteur.
 - Elle est évaluée à la fin de chaque parcours.
- Lorsque la condition (l'expression booléenne) est absente, elle est considérée comme vraie.

Les instructions de contrôle en C++

161

L'instruction for

□ Exemple 1:

//Affiche les carrés des 5 premiers entiers.

```
for ( int i(0) ; i < 5 ; ++i ){  
    cout << "Le carre de " << i << " vaut " << i * i << endl;  
}
```

Le carre de 0 vaut 0

Le carre de 1 vaut 1

Le carre de 2 vaut 4

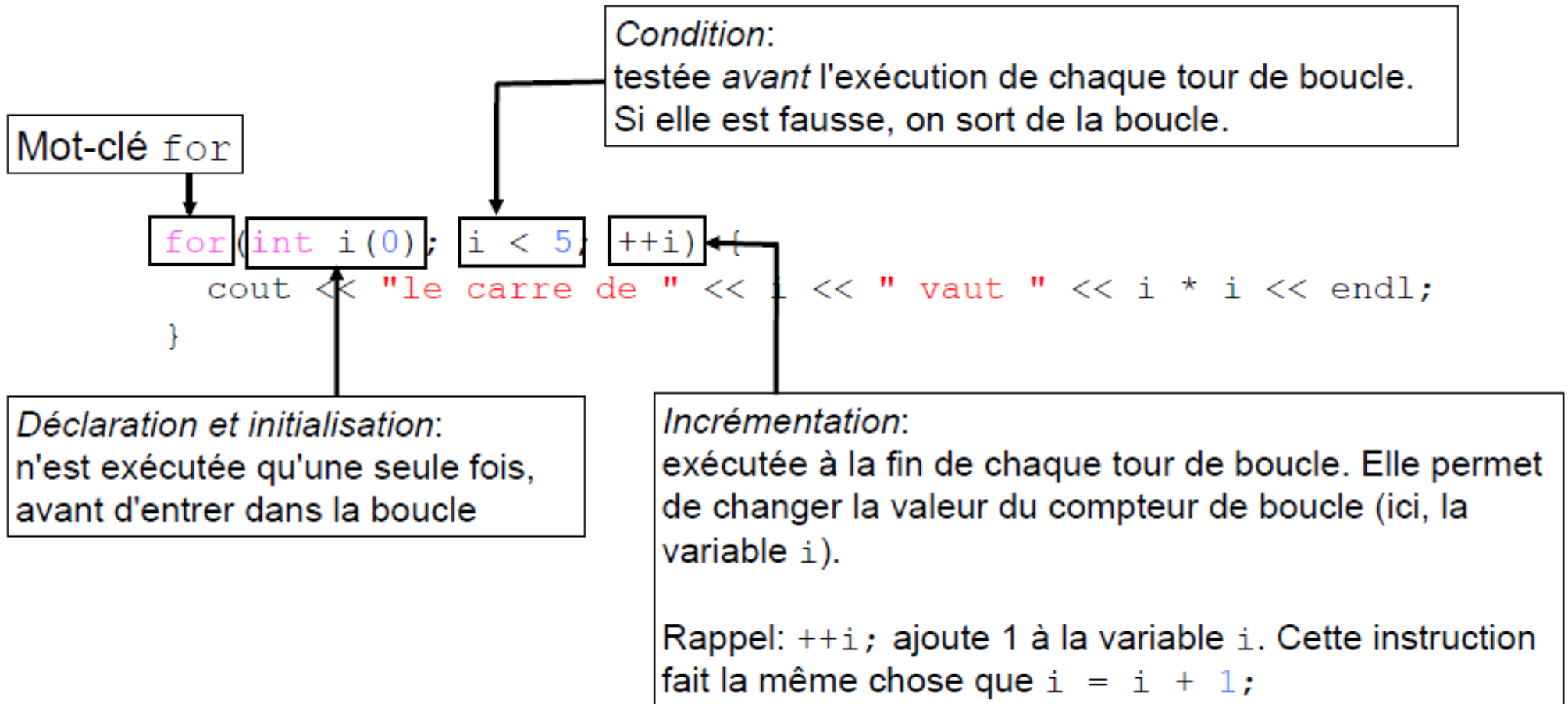
Le carre de 3 vaut 9

Le carre de 4 vaut 16

L'instruction for

162

□ Exemple 1: (suite)



Les instructions de contrôle en C++

163

L'instruction for

□ Exemple 2:

```
for ( int i(0), j(2) ; i<=4 ; i++, j+=i ){  
    cout << "i = " << i << " j = " << j << endl ;  
}
```

Output:

```
i = 0 j = 2  
i = 1 j = 3  
i = 2 j = 5  
i = 3 j = 8  
i = 4 j = 12
```

Les instructions de contrôle en C++

164

L'instruction for

□ Exemple 3:

```
for (double x(0.) ; x<=1. ; x+=0.1 ){  
    cout << "x = " << x << endl ;  
}
```

Output:

x = 0

x = 0.1

x = 0.2

...

x = 0.8

x = 0.9

x = 1

Les instructions de contrôle en C++

165

L'instruction for

- Equivalence entre les trois ensembles d'instructions suivants :

```
/*-----cas 1-----*/
for (i=1 ; i<=5 ; i++) {
    cout << "Salam tout le monde " << endl ;
    cout << i << " fois" << endl ;
}

/*-----cas 2-----*/
i=1 ;
for ( ; i<=5 ; i++) { // ne pas oublier le premier point-virgule
    cout << " Salam tout le monde " << endl ;
    cout << i << " fois" << endl ;
}

/*-----cas 3-----*/
i=1 ;
for (; i<=5 ; ) {
    cout << " Salam tout le monde " << endl ;
    cout << i << " fois" << endl ;
    i++ ;
}
```

Les instructions de contrôle en C++

166

L'instruction `for`

□ Remarques

- 1) Si la *condition* est absente, elle est considérée comme vraie.
On pourrait penser que, dans ce cas, on aboutit à une boucle infinie. En fait, on verra qu'il est possible qu'une telle boucle renferme une instruction *break* permettant d'y mettre fin.
- 2) Notez bien que dans *déclaration_et_initialisation*, on n'a droit qu'à un « déclarateur ».
Ceci serait illégal :
`for (int i=1, j=0, float x=1.5 ; ... ; ...) // erreur`
- 3) Une déclaration n'est permise que dans la première « expression » de l'instruction `for`.

Les instructions de contrôle en C++

167

L'instruction for

□ Remarques (suite)

- 4) A deux exceptions près¹⁾, l'instruction `for` est équivalente à la boucle `while` suivante :

```
initialisation;  
while (expression_booléenne) {  
    instruction;  
    conclusion;  
}
```

- 1) - S'il y a un `continue`, *conclusion* sera exécuté dans l'instruction `for` mais pas dans `while`
- *Initialisation* et *conclusion* ne peuvent pas être des instructions séparées par des virgules dans la boucle `while`

Les instructions de contrôle en C++

168

L'instruction for

□ Exemple - pour calculer

$$n! = 1 \times 2 \times 3 \times \dots \times n$$

```
int n, facteur, produit = 1;
for (facteur = n; facteur > 0; facteur--) {
    produit *= facteur;
}
cout << produit;
```

Les boucles en C++

169

Choisir la boucle **while**/la boucle **do-while**/la boucle **for**?

- Quand le nombre d'itérations (de répétitions) est connu avant d'entrer dans la boucle, utiliser **for**:

```
for ( int i (0); i < nombre_d_iterations; ++i) { }
```

- Sinon, utiliser **while**:

- Quand les instructions doivent être effectuées au moins une fois, utiliser

do...while:

```
do {
```

```
    instructions;
```

```
    } while (condition);
```

- Sinon, utiliser la forme **while**...

```
while (condition) {
```

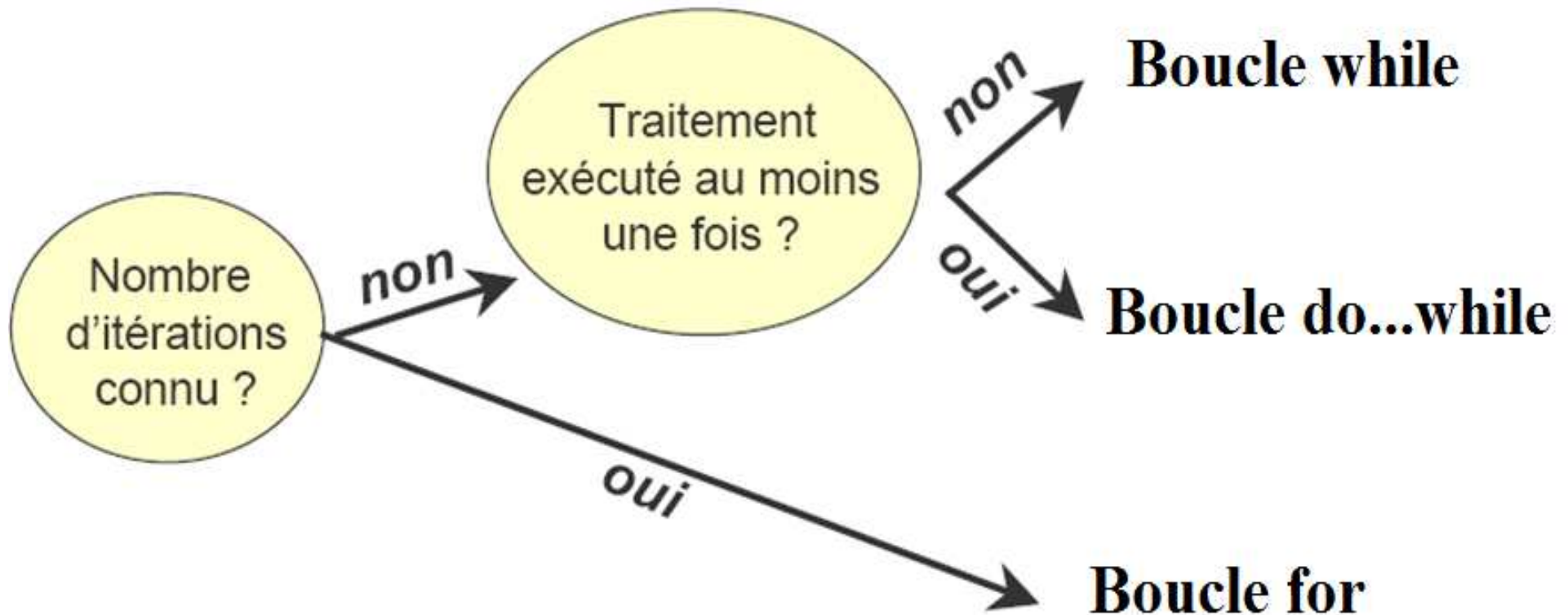
```
    instructions;
```

```
}
```

Les boucles en C++

170

Choisir la boucle while/la boucle do-while/la boucle for?

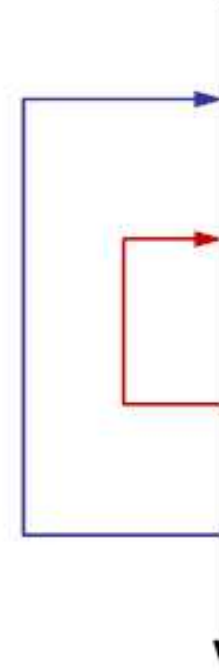


Imbrication de boucles

171

- On peut placer n'importe quelle(s) instruction(s) à l'intérieur d'une instruction d'itération `while`, `do` ou `for`, y compris une nouvelle instruction d'itération.
- Les instructions itératives peuvent donc être imbriquées.
- Une telle construction est très fréquente.

```
for ( int i=1; i<=5; i++ ){  
    cout << "i> " << i << endl;  
    for ( int k=0; k<=2; k++ ){  
        cout << "k> " << k << endl;  
    }  
  
    cout << "-----" << endl;  
}
```



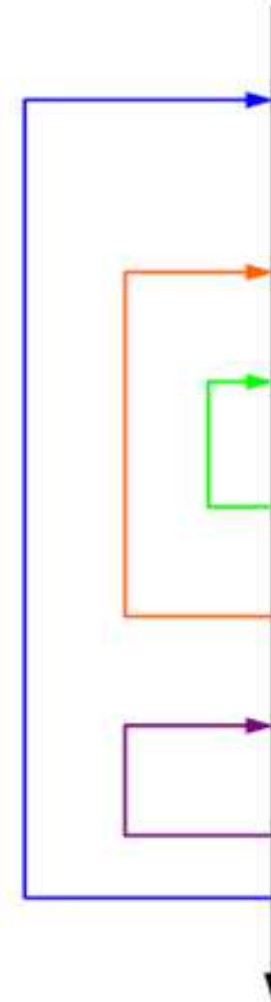
```
i> 1  
k> 0  
k> 1  
k> 2  
-----  
i> 2  
k> 0  
k> 1  
k> 2  
-----  
i> 3  
k> 0  
k> 1  
k> 2  
-----  
i> 4  
k> 0  
k> 1  
k> 2  
-----  
i> 5  
k> 0  
k> 1  
k> 2  
-----
```

Imbrication de boucles

172

- Autre exemple d'imbrication de boucles :

```
for ( int i=0; i<=3; i++ ){  
    cout << "i> " << i << endl;  
    int j = 2*i;  
    while (j>0){  
        cout << "j> " << j << endl;  
        for ( int k=6; k>0; k = k-2 ){  
            cout << "k> " << k << endl;  
        }  
        j = j/2;  
    }  
    int m = 4-i;  
    do {  
        cout << "m> " << m << endl;  
    } while (--m>0);  
}
```



```
i> 0  
m> 4  
m> 3  
m> 2  
m> 1  
i> 1  
j> 2  
k> 6  
k> 4  
k> 2  
j> 1  
k> 6  
k> 4  
k> 2  
m> 3  
m> 2  
m> 1  
i> 2  
j> 4  
k> 6  
k> 4  
k> 2  
j> 2  
k> 6  
k> 4  
k> 2  
j> 1  
k> 6  
k> 4  
k> 2  
m> 2  
m> 1
```

Boucles: quiz

173

Que s'affiche-t-il quand on exécute le code :

```
for(int i(0); i < 3; ++i) {  
    for(int j(0); j < 4; ++j) {  
        if (i == j) {  
            cout << "*";  
        } else {  
            cout << j;  
        }  
    }  
    cout << endl;  
}
```

A:

*123
*123
*123

C:

B:

012*
012*
012*

D:

*123
0*23
01*3

?

Boucles: quiz

174

Que s'affiche-t-il quand on exécute le code :

```
for(int i(0); i < 3; ++i) {  
    for(int j(0); j < i; ++j) {  
        cout << j;  
    }  
    cout << endl;  
}
```

A:

0
01

B:

0
01
012

C:

rien

D:

0123
0123
0123

?

Les instructions de branchement inconditionnel

Instructions étiquetées

176

- On peut **donner un nom** (étiquette, label) **à une instruction** en la précédant d'un identificateur et d'un caractère deux points (':') :

***nom** : instruction ;*

- Les étiquettes ne sont utilisées qu'en relation avec **goto**.

Instruction break

177

Exemple

```
int i ;  
for (i=1 ; i<=10 ; i++) {  
    cout << "Début tour " << i << endl ;  
    cout << "Salam" << endl;  
    if (i==3) break ;  
    cout << "Fin tour " << i << endl;  
}  
cout << "Après la boucle for" ;
```

Instruction break

178

Résultat

Début tour 1

Salam

Fin tour 1

Début tour 2

Salam

Fin tour 2

Début tour 3

Salam

Après la boucle for

Instruction break

179

```
break;
```

- L'instruction `break` force le compilateur à passer immédiatement à la fin de l'instruction englobante `switch`, `while`, `do` ou `for` la plus interne

Instruction break

180

Remarques :

- ❑ L'instruction *break* ne peut apparaître que dans une boucle ou dans une instruction *switch*.
- ❑ En cas de boucles imbriquées, *break* fait sortir de la boucle la plus interne. De même si *break* apparaît dans un *switch* imbriqué dans une boucle, elle ne fait sortir que du *switch*.

Instruction continue

181

```
continue;
```

- L'instruction `continue` force la boucle englobante la plus interne à démarrer une **nouvelle itération**
- L'instruction `continue` ne peut être utilisée qu'au sein d'une boucle `while`, `do` ou `for`
- Effets :
 - Avec `while` et `do` : évalue l'expression booléenne et, si elle est vraie, exécute une nouvelle fois le corps de la boucle
 - Avec `for` : effectue les instructions de conclusion puis évalue l'expression booléenne et, si elle est vraie, exécute une nouvelle fois le corps de la boucle

Instruction continue

182

Exemple 1:

```
int i ;  
for ( i=1 ; i<=5 ; i++ ){  
    printf ("Début tour %d\n", i) ;  
    if (i<4) continue ;  
    printf ("Salam\n") ;  
}
```

Output:

```
Début tour 1  
Début tour 2  
Début tour 3  
Début tour 4  
Salam  
Début tour 5  
Salam
```

Instruction continue

183

Exemple 2:

```
int nbre ;
do
{
    cout << "Saisissez un nombre >0 : " ;
    cin >> nbre ;
    if (nbre<0) {
        cout << "SVP un nombre >0\n" ;
        continue ;
    }
    cout << "Son carré est : " << nbre*nbre << "\n" ;
} while(nbre) ;
```

Instruction continue

184

Exemple 2: (suite)

Output:

```
Saisissez un nombre >0 : 9
Son carré est : 81
Saisissez un nombre >0 : -4
SVP un nombre >0
Saisissez un nombre >0 : 0
Son carré est : 0
```

Instruction continue

185

Exemple 3:

```
for ( int i=1 ; i<=5 ; i++ ){
    printf ( "-----Tour i=%d-----\n",i) ;
    for ( int j=1 ; j<=5 ; j++ ){
        if (j == 3) continue ;           //Prochaine itération
                                         //de la boucle interne

        if (i == 4) continue;           //Prochaine itération
                                         //de la boucle externe

        cout<<"i*j = " << i*j << endl;
    }
}
```

Instruction continue

186

Exemple 3: (suite)

Output:

```
-----Tour i=1-----  
i*j = 1  
i*j = 2  
i*j = 4  
i*j = 5  
-----Tour i=2-----  
i*j = 2  
i*j = 4  
i*j = 8  
i*j = 10  
-----Tour i=3-----  
i*j = 3  
i*j = 6  
i*j = 12  
i*j = 15  
-----Tour i=4-----  
-----Tour i=5-----  
i*j = 5  
i*j = 10  
i*j = 20  
i*j = 25
```


Instruction continue

187

Remarques:

- ❑ Quand l'instruction *continue* est utilisée dans une boucle for, elle effectue bien un branchement sur l'évaluation de l'expression de fin de parcours de boucle , et non après.
- ❑ En cas de boucles imbriquées, l'instruction *continue* ne concerne que la boucle la plus interne.

Instruction goto

188

❑ L'instruction *goto* permet le branchement en un emplacement quelconque du programme.

Exemple 1:

```
for ( int i=1 ; i<=5 ; i++ ){  
    cout << "Début tour " << i << endl ;  
    cout << "Salam" << endl ;  
    if ( i==3 ) goto sortie;  
    cout << "Fin tour " << i << endl ;  
}  
sortie : cout << "Après la boucle" ;
```

Instruction goto

189

Exemple 1:

Output:

```
Début tour 1  
Salam  
Fin tour 1  
Début tour 2  
Salam  
Fin tour 2  
Début tour 3  
Salam  
Après la boucle
```

Instruction goto

190

Exemple 2:

Il est fortement recommandé de n'utiliser l'instruction *goto* que dans des circonstances exceptionnelles et d'éviter tout branchement vers l'intérieur d'un bloc, comme dans cet exemple:

```
int i ;  
goto ici ;  
for (i=0 ; i<5 ; i++){  
    cout << "Salam" << endl ;  
    ici : cout << i << endl ;  
}
```

Output:

4201083

Instruction goto

191

Exemple 3:

```
int n=0;
goto ici :
for (int i=0 ; i<5; i++){
    cout << "Salam" << endl;
    ici : cout << i << endl;
}
```

Output:

Build Failed. 1 errors, 1 warnings.

Instruction return

192

```
return [ expression ] ;
```

- L'instruction **return** termine l'exécution d'une méthode en cours d'exécution et rend le contrôle à la méthode appelante en retournant (restituant) éventuellement une valeur (expression) qui doit être compatible avec le type de la méthode.
- Si une méthode déclare un type de retour, l'instruction **return** (avec une expression compatible) doit obligatoirement figurer dans le corps de la méthode (dernière instruction exécutable).

```
double square(double x) {  
    return x * x;  
}
```

Travaux pratiques

193

1. Développer un convertisseur € / MAD ou \$ / MAD avec la possibilité de mettre à jour le taux de change.
2. Développer un menu pour choisir la devise à convertir :

1- Conversion € / MAD

2- Conversion \$ / MAD

3. Demander à l'utilisateur de mettre à jour le taux de change / la somme en euro :

Merci de mettre à jour le taux de change € :

Merci d'entrer la somme à convertir :

4. Fournir le résultat de la conversion :

L'équivalent de XXX € en MAD est : YYYY DH

Q & A

194



Références

195

- 1) <http://www.cplusplus.com>
- 2) <https://isocpp.org/>
- 3) <https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c>
- 4) Programmer en C++, Claude Delannoy, éditions Eyrolles, 2014.
- 5) Initiation à la programmation (en C++), Jean-Cédric Chappelier, & Jamila Sam, coursera, 2018.
- 6) Introduction à la programmation orientée objet (en C++), Jamila Sam & Jean-Cédric Chappelier, coursera, 2018.