

Chapitre 2

Structures de contrôle

Auteur : Marco Lavoie

Adaptation : Sébastien Bois, Hiver 2021



Langage C++

25908 IFM



Objectifs

2

- Utiliser les structures conditionnelles
- Utiliser les structures de répétition
- Comprendre les opérateurs d'incrémentement de décrémentation
- Utiliser les opérateurs d'affectation
- Utiliser les opérateurs logiques
 - Priorité de ces opérateurs dans les expressions conditionnelles
- Comprendre les instructions de contrôle de flux
break et **continue**



Aperçu

3

- Structures conditionnelles
 - `if`
 - `if / else`
 - `switch`
- Transtypage explicite et implicite
- Opérateurs d'affectation, d'incrémentation et de décrémentation
- Structures de répétition
 - `while`
 - `do / while`
 - `for`
 - Interruption d'itération : `break` et `continue`
- Opérateurs logiques et leur priorité



Structures `if` et `if / else`

4

- Syntaxe

```
if ( condition )  
    instruction ;
```

```
if ( condition )  
    instruction ;  
else  
    instruction ;
```

- Exemples

```
int note;  
std::cin >> note;  
if ( note >= 50 )  
    std::cout << "Réussite";
```

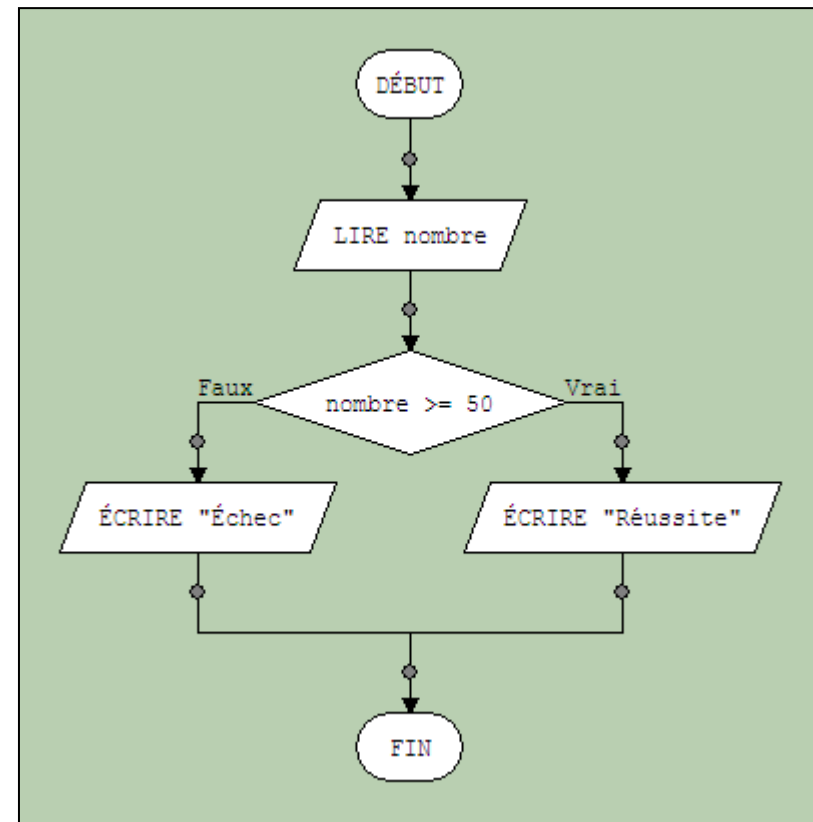
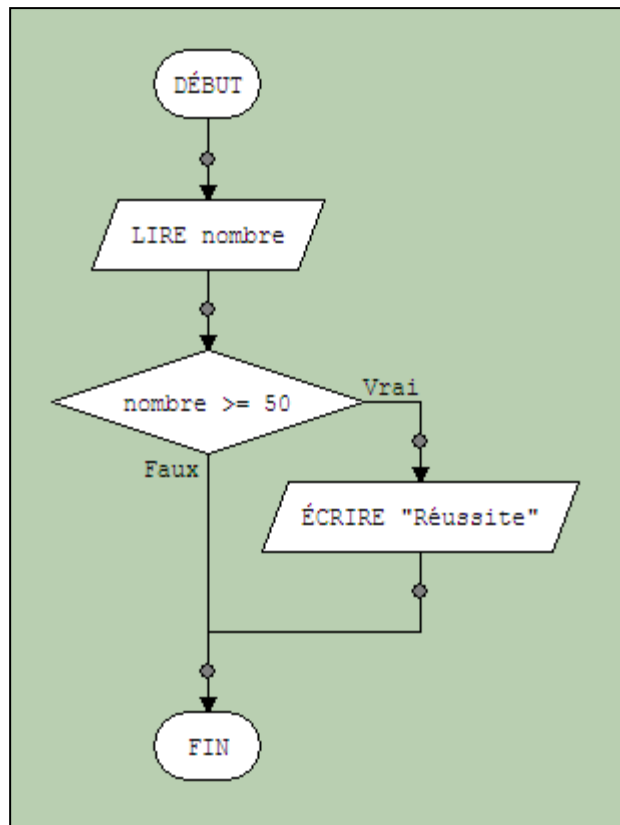
```
int note;  
std::cin >> note;  
if ( note >= 50 )  
    std::cout << "Réussite";  
else  
    std::cout << "Échec";
```

Attention aux séparateurs

Organigrammes

5

- Représentations algorithmiques





Opérateur conditionnel ? :

6

- L'opérateur conditionnel
?: permet d'écrire une
structure `if/else`
simple de façon plus succinct

```
if ( note >= 50 )  
    std::cout << "Réussite";  
else  
    std::cout << "Échec";
```

- Les deux instructions suivantes sont
équivalentes à la structure ci-contre

```
std::cout << ( note >= 50 ? "Réussite" : "Échec" );
```

ou

```
note >= 50 ? std::cout << "Réussite" : std::cout << "Échec";
```



Structures `if/else` imbriquées

7

- Les deux structures suivantes sont identiques
 - Seuls les retours de chariots et la mise en retrait diffèrent

```
if ( note >= 90 )
    std::cout << "A";
else
    if ( note >= 80 )
        std::cout << "B";
    else
        if ( note >= 70 )
            std::cout << "C";
        else
            if ( note >= 60 )
                std::cout << "D";
            else
                std::cout << "EC";
```

```
if ( note >= 90 )
    std::cout << "A";
else if ( note >= 80 )
    std::cout << "B";
else if ( note >= 70 )
    std::cout << "C";
else if ( note >= 60 )
    std::cout << "D";
else
    std::cout << "EC";
```

Note : les accolades ne sont pas requises car une structure `if/else` est considérée comme une seule instruction



Erreur de programmation courante

8

- Attention : l'ordre des conditions dans une structure `if/else` imbriquée est importante

- Qu'y a-t-il d'erroné dans la structure ci-contre ?

- Si la note est 60 ou plus, alors "D" est affiché quelque soit la valeur de la note

```
if ( note >= 60 )  
    std::cout << "D";  
else if ( note >= 70 )  
    std::cout << "C";  
else if ( note >= 80 )  
    std::cout << "B";  
else if ( note >= 90 )  
    std::cout << "A";  
else  
    std::cout << "EC";
```




Blocs d'instructions

9

- Attention à ne pas oublier les accolades délimitant les blocs d'instructions dans une structure `if/else`

```
if ( note >= 50 )  
    std::cout << "Réussite";  
else {  
    std::cout << "Échec";  
    std::cout << "Vous devez reprendre le cours";  
}
```

- La mise en retrait est ignorée par le compilateur
 - Sans les accolades, le code ci-dessus affiche toujours la *Vous devez reprendre le cours*, quelque soit la valeur de la note



Structure répétitive `while`

10

- Syntaxe

```
while ( condition )  
    instruction ;
```

- Exemples

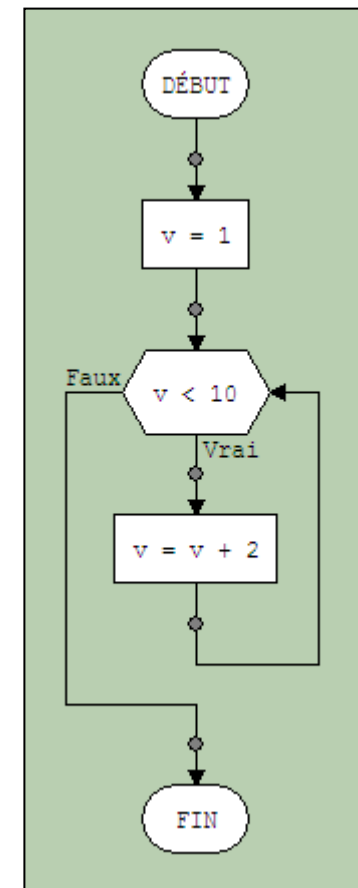
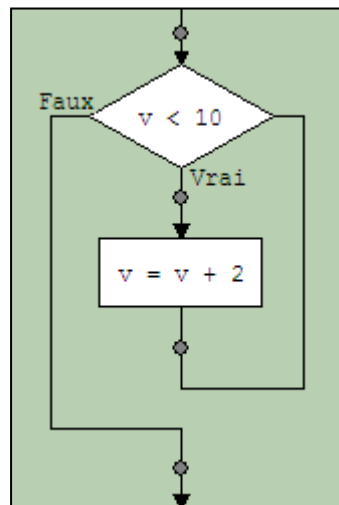
```
int v = 1;  
while ( v <= 10 )  
    v += 2;
```

```
int note;  
std::out << "Note? ";  
std::cin >> note;  
while ( note < 0 ) {  
    std::cout << "*** Erreur" << std::endl  
                << "Note? ";  
    std::cin >> note;  
}
```

Organigrammes

11

- Représentation algorithmique
 - Certains auteurs exploitent le symbole conditionnel pour représenter la condition de la boucle





Attention aux boucles infinies

12

- Exemples

```
unsigned int v;  
std::cin >> v;  
while ( v == 100 );  
    std::cin >> v;
```

- *Comportement ?*

- Boucle sans fin, sans lecture

- *Cause ?*

- Aucune instruction dans la boucle permettant de modifier le résultat de la condition
- Enlever le ; après la condition

```
unsigned int v;  
std::cin >> v;  
while ( v = 100 )  
    std::cin >> v;
```

- *Comportement ?*

- Lecture à répétition, sans fin

- *Cause ?*

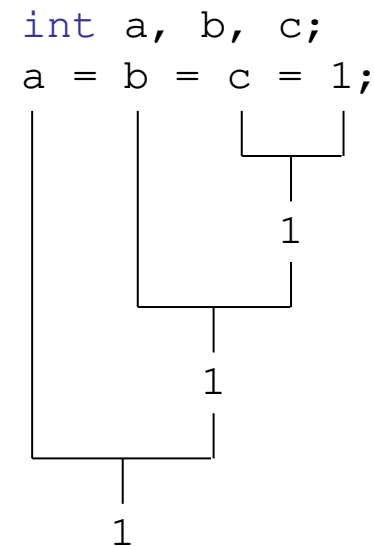
- L'affectation dans la condition est toujours vraie
- Devrait être (v == 100)



Erreur courante due à l'affectation

13

- L'affectation retourne une valeur résultante
 - Le résultat retourné par une affectation est la valeur assignée à la variable
 - C'est ce qui permet le chaînage d'affectations
 - Évaluée de droite à gauche
- Définition de `false` et `true`
 - `false` = 0
 - `true` = valeur autre que 0
 - Donc une condition est vraie si son résultat est autre que 0





Erreur courante due à l'affectation (suite)

14

- Retour sur l'exemple précédent

- La condition contient une affectation dont le résultat est 100 → toujours vraie

```
unsigned int v;  
std::cin >> v;  
while ( v = 100 )  
    std::cin >> v;
```

- L'affectation retourne un résultat car les programmeurs exploitent souvent cette fonctionnalité

```
char c;  
while ((c = cin.get()) != EOF) {  
    // traiter le caractère  
}
```

- Le fait de pouvoir exploiter l'affectation dans une condition résulte de cette fonctionnalité



Erreur due à la comparaison de flottants

15

- Attention lorsque vous comparez l'égalité de valeurs flottantes dans une condition
 - L'arithmétique à virgule flottante n'est pas exacte, donc deux nombres flottants théoriquement exactes peuvent ne pas l'être en pratique
 - Exemple : $(1.0 / 3.0) * 3.0 \neq 1.0$
 $0.999999999 \neq 1.0$
 - Exploitez plutôt une *valeur de tolérance* :

```
float v = 1.0 / 3.0;  
if ( abs(v * 3.0 - 1.0) < 0.0000001 )  
    std::cout << "v est 1";
```



Exercice #2.1

16

- Solutionnez l'exercice distribué par l'instructeur
 - Créer un nouveau projet console *Visual Studio C++*
 - Solutionner le problème tel que décrit, mais avec la variante suivante
 - *Le programme doit aussi afficher le total des salaires payés et des heures travaillées avant de fermer la console*
 - N'oubliez pas les conventions d'écriture
 - Soumettez votre projet selon les indications de l'instructeur
 - **Attention** : respectez à la lettre les instructions de l'instructeur sur la façon de soumettre vos travaux, *sinon la note EC sera attribuée à ceux-ci*



Transtypage explicite

17

- Le **transtypage explicite** permet de transformer le type d'une expression
- Syntaxe

```
static_cast< type > ( expression )
```

- *expression* est une expression dont on veut changer le type du résultat
 - *type* est le nouveau type du résultat
- Le résultat du transtypage explicite est le résultat de l'expression selon le type spécifié
 - Peut parfois occasionner des pertes d'information



Transtypage explicite (suite)

18

- Exemple concret

```
// Calcul de moyenne de notes
#include <iostream>

int main() {
    int note, total = 0, compteNotes = 0;
    double moyenne;

    std::cout << "Entrez les notes (-1 pour terminer)";
    std::cout << " Note? ";
    std::cin >> note;

    while ( note != -1 ) {
        total += note;
        compteNotes++;
        std::cout << " Note? ";
        std::cin >> note;
    }

    moyenne = static_cast< double >( total ) / compteNotes;
    std::cout << "Moyenne = " << moyenne << std::endl;

    return 0;
}
```

Pourquoi le transtypage
est-il requis ?

Sans transtypage, la division
total/compteNotes serait
entière (perte de précision)



Transtypage explicite (suite)

19

- Autres exemples

Exemple	Résultat	Explication
<code>static_cast< int >('a');</code>	97	Code ASCII de 'a' est 97
<code>static_cast< char >(65);</code>	'A'	Code ASCII de 'A' est 65
<code>static_cast< int >(2.3);</code>	2	Perte d'information
<code>static_cast< float >(2);</code>	2.0	Aucune perte d'information

- Principales utilisations du transtypage explicite

- Pour éviter la division entière
- Pour adapter les arguments aux paramètres lors d'appel de fonctions
- Pour manipuler des pointeurs génériques

} *Étudiés plus tard*



Transtypage implicite

20

- Le compilateur peut effectuer certains transtypages "oubliés" par le programmeur
 - C'est du **transtypage implicite**

Exemple

```
float f = 5;
```

```
std::cout << 9 / 2.5;
```

```
std::cout.put(97);
```

Explication

La constante 5 étant de type `int`, le compilateur doit convertir en `float`

Les arguments de la division étant de types différents, l'entier 9 est **promu** en `float`, évitant ainsi la perte d'information (affiche 3.6)

La fonction `cin.put()` requiert un caractère (type `char`) en argument (affiche 'a')



Transtypage implicite (suite)

21

- Lorsque le transtypage implicite peut occasionner une perte d'information, le compilateur affiche un message d'avertissement mais effectue tout de même le transtypage

- Exemple dans *Visual Studio* :

```
int c;  
c = 2.1;
```

produit le message d'avertissement suivant

```
warning C4244: '=' : conversion de 'double' en  
'int', perte possible de données
```

- Aucun avertissement de ce genre pour les transtypages explicites
 - Le compilateur assume que le programmeur en est conscient



Transtypage invalide

22

- Certains transtypages sont invalides

```
double d = static_cast< double >( "abc" );
```

- Le transtypage est effectué lors de la compilation
 - Lorsque le compilateur ne peut faire le transtypage, il génère un message d'erreur
 - Exemple ci-dessus compilé par *Visual Studio* :

```
error C2440: 'static_cast' : impossible de convertir  
de 'const char [4]' en 'double'
```



Opérateurs d'affectation

23

- Le langage C++ offre les opérateurs d'affectation suivants

Opérateur	Exemple	Signification
=	<code>a = 5;</code>	$a \leftarrow 5$
+=	<code>a += 5;</code>	$a \leftarrow a + 5$
-=	<code>a -= 5;</code>	$a \leftarrow a - 5$
*=	<code>a *= 5;</code>	$a \leftarrow a * 5$
/=	<code>a /= 5;</code>	$a \leftarrow a / 5$
%=	<code>a %= 5;</code>	$a \leftarrow a \% 5$



Opérateurs d'incrémentation et décrémentation

24

- Opérateurs d'incrémentation et décrémentation disponibles en versions préfixe et suffixe

Opérateur	Exemple	Signification
++	<code>std::cout << ++a;</code>	<code>a = a + 1; std::cout << a;</code>
	<code>std::cout << a++;</code>	<code>std::cout << a; a = a + 1;</code>
--	<code>std::cout << --a;</code>	<code>a = a - 1; std::cout << a;</code>
	<code>std::cout << a--;</code>	<code>std::cout << a; a = a - 1;</code>

- **Version préfixe** : le résultat retourné est la valeur APRÈS l'application de l'opérateur
- **Version suffixe** : le résultat retourné est la valeur AVANT l'application de l'opérateur



Structure répétitive `for`

25

- Syntaxe

```
for ( initialisation; condition; incrément )  
    instruction ;
```

- Exemple

```
for ( int i = 0; i < 10; i++ )  
    std::cout << "Bonjour";
```

– Correspond à

```
initialisation;  
while ( condition ) {  
    instruction;  
    incrément;  
}
```

```
int i = 0;  
while ( i < 10 ) {  
    std::cout << "Bonjour";  
    i++;  
}
```



Variable d'itération

26

- Variable d'itération

```
for ( int i = 0; i < 10; i++ )  
    std::cout << "Bonjour";
```

- Si la variable d'itération (**i** dans l'exemple) est déclarée dans la structure `for`, sa portée est limitée à l'exécution de la structure `for`

```
for ( int i = 0; i < 10; i++ )  
    std::cout << i;  
i = 0; ← i non déclarée
```



```
int i;  
for ( i = 0; i < 10; i++ )  
    std::cout << i;  
i = 0;
```





Plusieurs variables d'itération

27

- Un `for` peut exploiter plusieurs variables d'itération

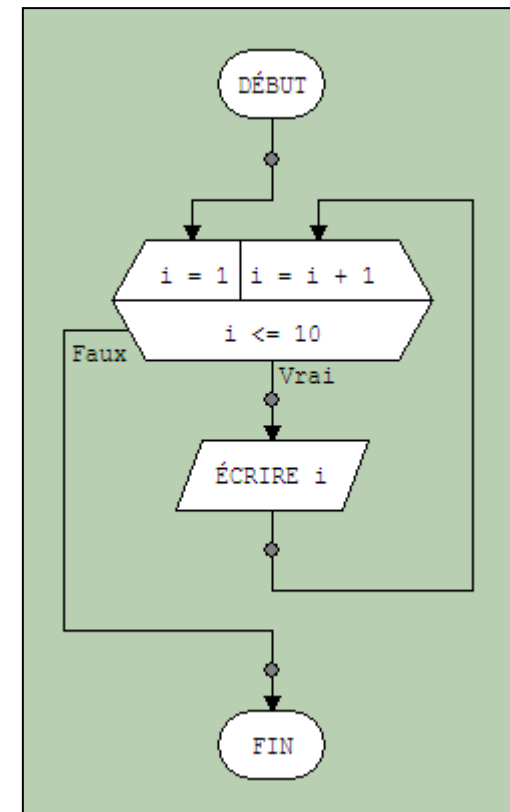
```
      initialisation                                incrément  
      └───┬──────────┘                                └───┬──────────┘  
for ( int i = 0, j = 10; i * j > 0; i++, j-- )  
    std::cout << i * j << std::endl;
```

- Si elles sont déclarées dans la structure, elles doivent toutes être du même type
 - Sinon il faut déclarer les variables à l'extérieur de la structure
- La virgule (,) sert de séparateur dans l'initialisation ainsi que dans l'incrément

Organigrammes

28

- Représentation algorithmique
 - Le symbole utilisé à l'entrée de la structure peut varier selon l'auteur
 - Pas de standard





Exercice #2.2

29

- Solutionnez l'exercice distribué par l'instructeur
 - Créer un nouveau projet console *Visual Studio C++*
 - Solutionner le problème tel que décrit, avec les spécifications supplémentaires suivantes
 - Un seul programme devant produire les quatre figures successivement avec une pause (style *Pressez une touche pour continuer...*) entre chacune
 - Chaque figure est produite par une seule structure `for`. Celle-ci peut cependant contenir des `for` imbriquées.
 - N'oubliez pas les conventions d'écriture
 - Soumettez votre projet selon les indications de l'instructeur



Structure répétitive do/while

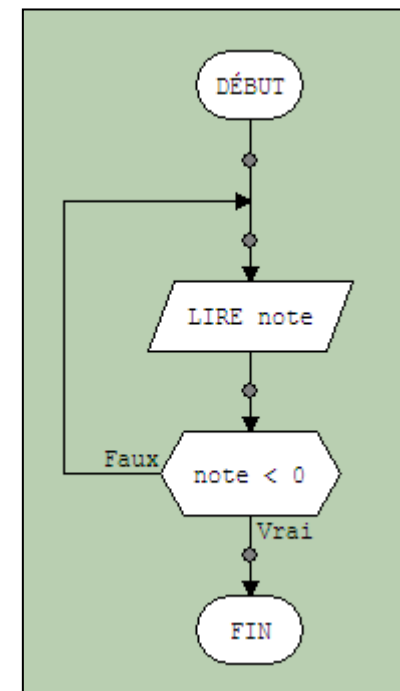
30

- Syntaxe

```
do  
    instruction ;  
while ( condition );
```

- Utilisée lorsqu'au moins une itération est requise
 - Exemple

```
int note;  
do {  
    std::cout << "Note? ";  
    std::cin >> note;  
} while (note < 0);
```





Structure de sélection `switch`

31

- Syntaxe

- Alternative aux `if/else` imbriqués lorsque les conditions aiguillent l'exécution en fonction de la valeur d'une variable

Les instructions `break` sont optionnelles

La partie `default` est optionnelle {

```
switch ( variable )  
    case constante :  
        instruction;  
        break;  
    case constante :  
        instruction;  
        break;  
    ...  
    default :  
        instruction;  
}
```

- Couramment utilisée gérer le choix de l'utilisateur à un menu



Exemple de switch

32

```
char noteLettre;

// Lire note alphabétique
std::cout << "Note? ";
std::cin >> noteLettre;

// Convertir en note numérique
float noteNum;
switch ( noteLettre ) {
    case 'A':
    case 'a':
        noteNum = 4.0;
        break; ← break requis pour
                sortir du switch

    case 'B':
    case 'b':
        noteNum = 3.0;
        break; ←
```

```
    case 'C':
    case 'c':
        noteNum = 2.0;
        break; ←

    case 'D':
    case 'd':
        noteNum = 1.0;
        break; ←

    case 'F':
    case 'f':
        noteNum = 0.0;
        break; ←

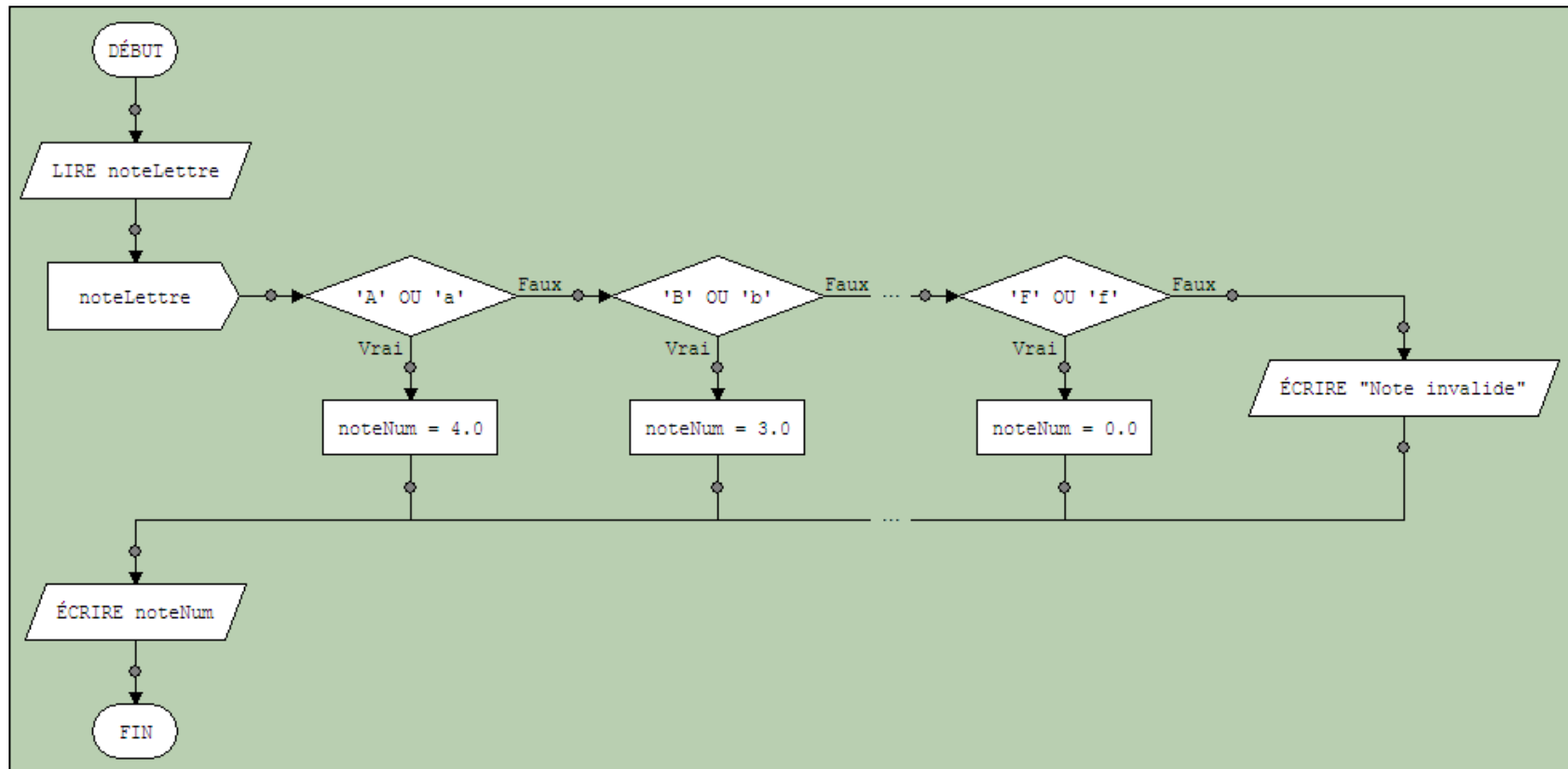
    default:
        std::cout << "Note invalide.\n"; }
}
```

*Exécuté si noteLettre ne
correspond pas à une des
constantes énumérées*

Organigrammes

33

- Représentation algorithmique





Importance du break

34

- Sans le break, l'exécution continue dans le switch

```
int choix;  
std::cin >> choix;  
  
switch ( choix ) {  
    case 1:  
        std::cout << "Un\n";  
        break;  
    case 2:  
        std::cout << "Deux\n";  
        break;  
    case 3:  
        std::cout << "Trois\n";  
        break;  
    case 4:  
        std::cout << "Quatre\n";  
        break;  
    default:  
        std::cout << "Invalide\n";  
}
```

- Si l'utilisateur entre **2** :
Le code affiche Deux
- Si on enlève les trois premiers break et l'utilisateur entre **2** :
Le code affiche Deux
Trois
Quatre
- Si on enlève le dernier break et l'utilisateur entre **2** :
Le code affiche Deux
Trois
Quatre
Invalide



Restrictions d'utilisation du `switch`

35

- L'utilisation du `switch` impose quelques restrictions
 - *variable* doit être une expression entière (i.e. de type `int`, `short`, `long` ou `char`)
 - *constante* doit être une constante de type entier (comme *variable*)

```
switch ( variable )  
    case constante :  
        instruction;  
        break;  
    case constante :  
        instruction;  
        break;  
    ...  
    default :  
        instruction;  
}
```



Instructions `break` et `continue`

36

- Modifient le flot d'exécution dans une structure répétitive ou de sélection
 - Structures `while`, `do/while` et `for`
 - `break` permet de quitter immédiatement la structure de boucle
 - `continue` termine l'itération en cours pour passer à la prochaine
 - Structure `switch`
 - `break` permet de quitter la structure de sélection
 - `continue` n'est pas applicable dans un `switch`

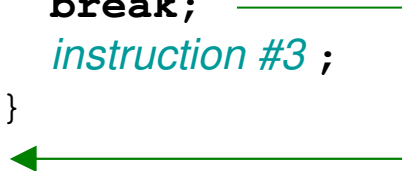


Redirection du flot d'exécution

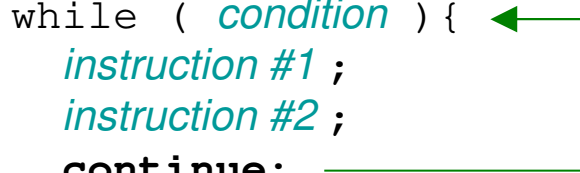
37

- Nous avons déjà vu des exemples d'utilisation du `break` dans un `switch`
- Le `break` et le `continue` dans une boucle :

```
while ( condition ) {  
    instruction #1 ;  
    instruction #2 ;  
    break ;  
    instruction #3 ;  
}
```



```
while ( condition ) {  
    instruction #1 ;  
    instruction #2 ;  
    continue ;  
    instruction #3 ;  
}
```



– Même chose dans le `do/while` et le `for`

Exemple

38

- Utilisation du `break` et du `continue`

```
for (int i = 0; i < 10; i++) {  
    std::cout << i;  
  
    if (i == 8)  
        break;  
  
    std::cout << "x";  
  
    if (i < 4)  
        continue;  
  
    std::cout << "y";  
}
```

— Qu'affiche la boucle ci-contre ?

0x1x2x3x4xy5xy6xy7xy8

Le `continue` cesse d'être appliqué

Le `break` est appliqué



Opérateurs logiques

39

- Opérateurs permettant de combiner des conditions simples
 - Rappel : les *opérateurs relationnels* sont <, <=, >, >=, == et !=
 - Les **opérateurs logiques** sont
 - && : ET logique
 - || : OU logique
 - ! : Négation

- Exemples

```
if ( !( note == 'A' || note == 'B' ) )  
    std::cout << "Éprouve des difficultés";
```

```
if ( ( sexe == 'M' && age < 20 ) || ( sexe == 'F' && age < 18 ) )  
    std::cout << "Manque de maturité!";
```



Exercice #2.3

40

- Solutionnez l'exercice distribué par l'instructeur
 - Créer un nouveau projet console *Visual Studio C++*
 - Solutionner le problème tel que décrit, avec les spécifications supplémentaires suivantes
 - Votre programme doit traiter au minimum une paire de nombre (i.e. utilisez un `do/while`)
 - Votre programme doit afficher le total de toutes les ventes de la journée
 - N'oubliez pas les conventions d'écriture
 - Soumettez votre projet selon les indications de l'instructeur



Priorité des opérateurs

41

- Incluant tous les opérateurs vus à date, en ordre décroissant de priorité

Opérateurs	Associativité
()	De gauche à droite
<code>static_cast<type>()</code> <code>++</code> <code>--</code> (<i>versions suffixe</i>)	De droite à gauche
<code>++</code> <code>--</code> <code>+</code> <code>-</code> (<i>versions préfixe</i>)	De gauche à droite
!	De gauche à droite
* / %	De gauche à droite
+ -	De gauche à droite
<< >>	De gauche à droite
< <= > >=	De gauche à droite
== !=	De gauche à droite
&&	De gauche à droite
	De gauche à droite
?:	De droite à gauche
= += -= *= /= %=	De droite à gauche



Erreurs de programmation

42

- Ne pas oublier les accolades `{ }` lorsque requises par une structure conditionnelle ou répétitive
 - Et éviter le `;` après la condition de ces structures
- Ne pas comparer l'égalité de deux valeurs flottantes avec `==` (utiliser une tolérance)
- Attention à l'opérateur relationnel utilisé dans la condition du `for`
 - Pour itérer n fois : `(i = 0; i < n; i++)` ou `(i = 1; i <= n; i++)`
- Ne pas confondre l'affectation (`=`) avec l'opérateur d'égalité (`==`) dans les conditions
- Ne pas oublier les `break` lorsque requis dans un `switch`
- Ne pas écrire `(3 < x < 7)` mais `(3 < x && x < 7)`



Bonnes pratiques de programmation

43

- Mettre en retrait adéquatement les instructions dans les structures
- Comparer des flottants en exploitant une tolérance
- Éviter le transtypage implicite car il peut occasionner des résultats inattendus
 - Exploiter le transtypage explicite (`static_cast<type>()`)
- Insérer une ligne vide avant et après chaque structure
- Attention aux erreurs de décalage de un dans un `for` (en confondant `<=` avec `<`)
- La négation de `(a && b)` n'est pas `!a && !b`, mais `!(a && b)` ou `!a || !b` (lois de Morgan)
- Si une boucle doit itérer au moins une fois, exploiter un `do/while` plutôt qu'un `while`
- Toujours mettre un `default` : dans un `switch`, même s'il ne contient aucune instruction
 - Et porter une attention particulière aux `case` : sans `break`
- Attention lorsqu'une boucle contient des `break` ou `continue`



Devoir #1

44

- Solutionnez le problème distribué par l'instructeur
 - Créer un nouveau projet console *Visual Studio C++*
 - Solutionner le problème tel que décrit
 - N'oubliez pas les conventions d'écriture
 - Respectez l'échéance imposée par l'instructeur
 - Soumettez votre projet selon les indications de l'instructeur
 - **Attention** : respectez à la lettre les instructions de l'instructeur sur la façon de soumettre vos travaux, *sinon la note EC sera attribuée à ceux-ci*



Pour la semaine prochaine

45

- Vous devez relire le contenu de la présentation du chapitre 2
 - Il y aura un **quiz** sur ce contenu au prochain cours
 - À livres et ordinateurs fermés