

# Standard Version of Starting Out with C++, 4th Edition

## Chapter 4 Making Decisions



# Topics

4.1 Relational Operators

4.2 The `if` Statement

4.3 Flags

4.4 Expanding the `if` Statement

4.5 The `if/else` Statement

4.6 The `if/else if` Statement

4.7 Using a Trailing `else`

4.8 Menus

4.9 Nested `if` Statements

# Topics

4.10 Logical Operators

4.11 Checking Numeric Ranges with Logical Operators

4.12 Validating User Input

4.13 More About Variable Definitions and Scope

4.14 Comparing Strings

4.15 The Conditional Operator

4.16 The `switch` Statement

4.17 Testing for File Open Errors

# 4.1 Relational Operators

- Used to compare numbers to determine relative order
- Operators:
  - > Greater than
  - < Less than
  - >= Greater than or equal to
  - <= Less than or equal to
  - == Equal to
  - != Not equal to

# Relational Expressions

- Boolean expressions – `true` or `false`
- Examples:

`12 > 5` **is** `true`

`7 <= 5` **is** `false`

**if** `x` **is** `10`, **then**

`x == 10` **is** `true`,

`x != 8` **is** `true`, **and**

`x == 8` **is** `false`

# Relational Expressions

- Can be assigned to a variable:  
`result = x <= y;`
- Assigns 0 for false, 1 for true
- Do not confuse `=` and `==`

## 4.2 The `if` Statement

- Allows statements to be conditionally executed or skipped over
- Models the way we mentally evaluate situations:
  - “If it is raining, take an umbrella.”

- Format:

```
if (expression)
    statement;
```

# `if` statement – what happens

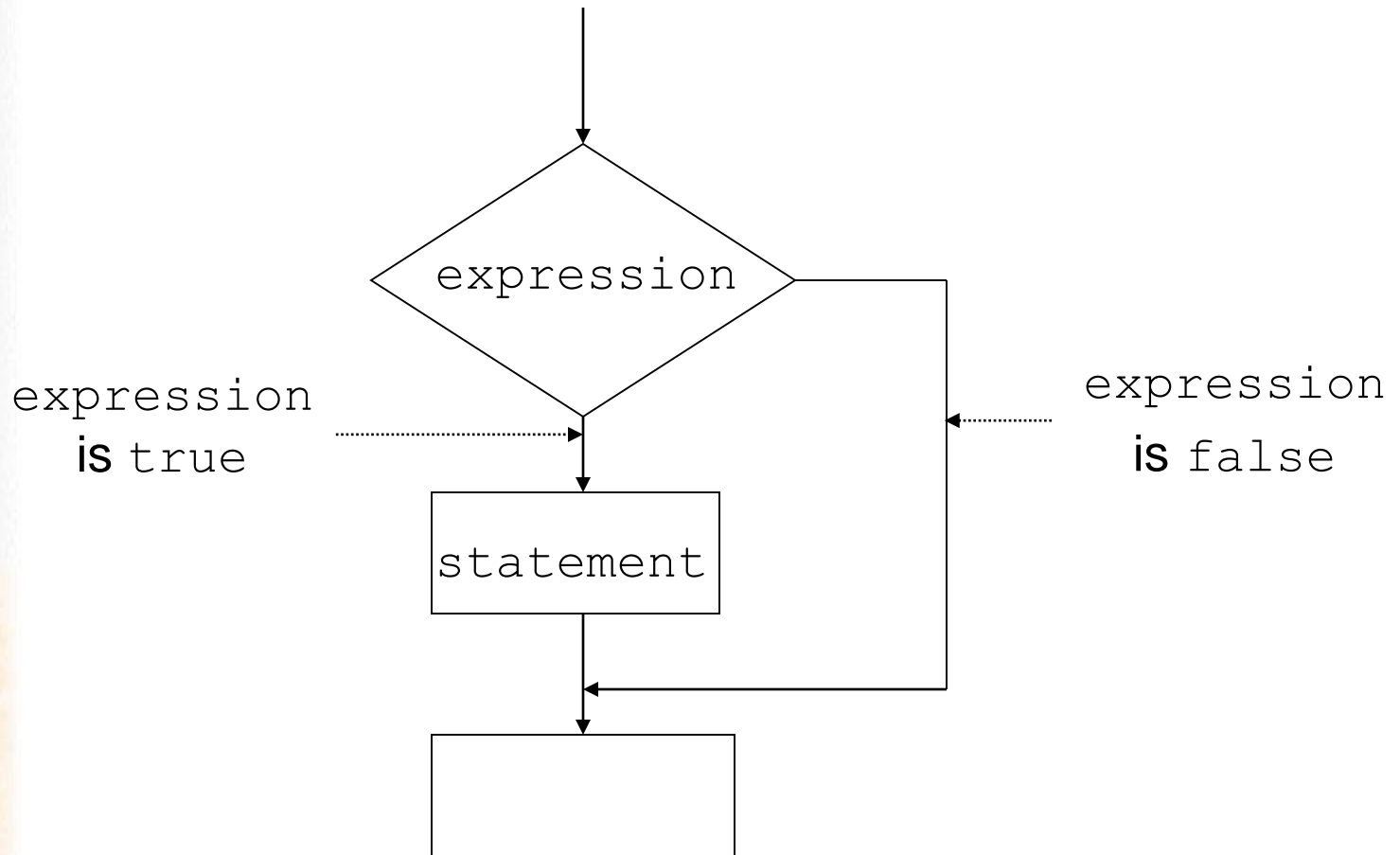
To evaluate:

```
if (expression)  
    statement;
```

- **If** (expression) **is true**, then statement **is executed**.
- **If** (expression) **is false**, then statement **is skipped**.



# if statement – what happens



# `if` statement notes

- Do not place `;` after `(expression)`
- Place `statement;` on a separate line after `(expression)`, indented:

```
if (score > 90)
    grade = 'A';
```

- Don't test `floats` for equality
- `0` is `false`; any other value is `true`

## 4.3 Flags

- Variable that signals a condition
- Often implemented as `bool`
- As with other variables in functions, must be assigned an initial value before it is used

## 4.4 Expanding the `if` Statement

- To execute > 1 statement as part of an `if` statement, enclose them in `{ }`:

```
if (score > 90)
{
    grade = 'A';
    cout << "Good Job!\n";
}
```

- `{ }` creates a block of code

## 4.5 The `if/else` Statement

- Allows choice between statements if (expression) **is** true **or** false
- Format:

```
if (expression)
    statement1;    // or block
else
    statement2;    // or block
```

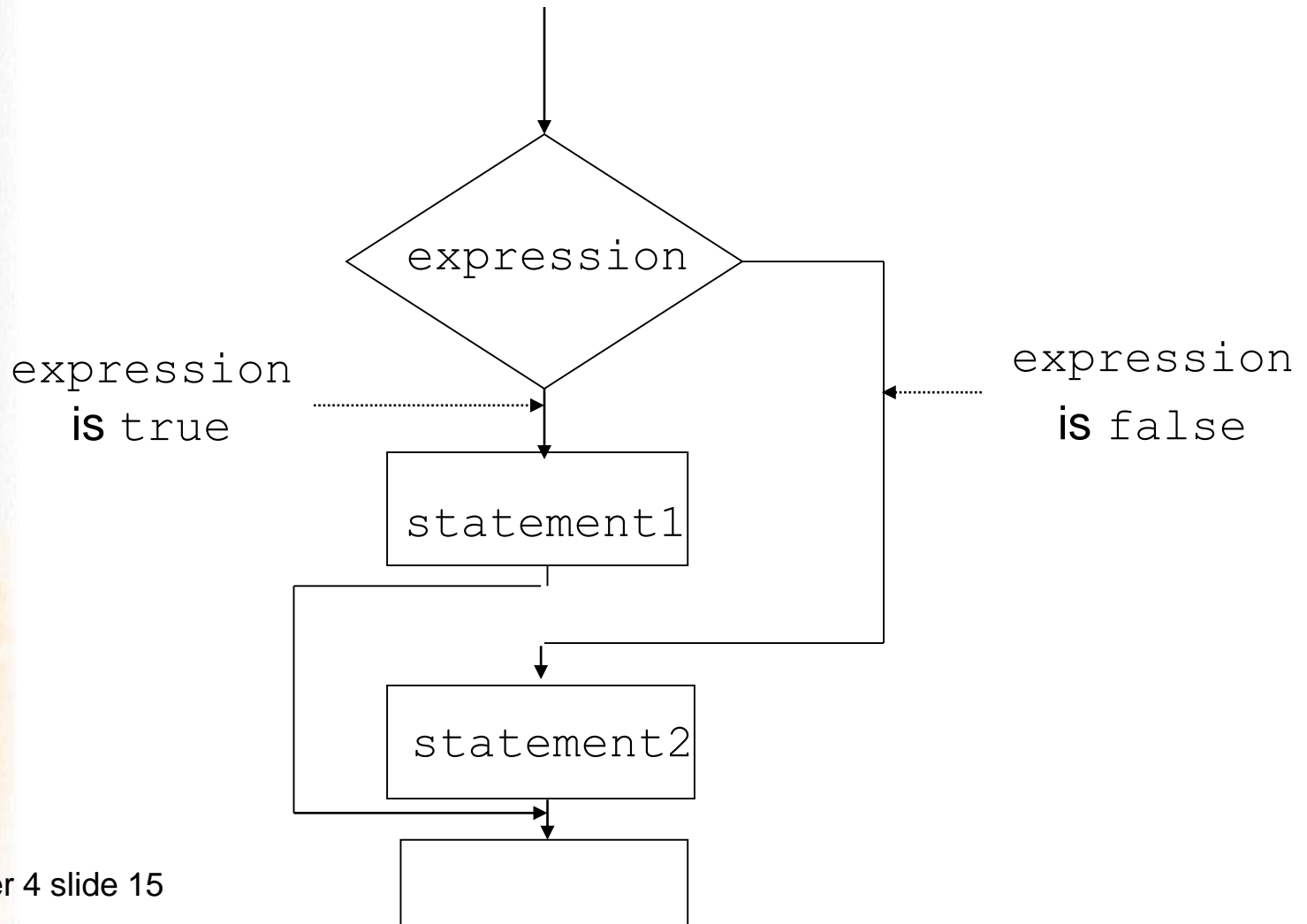
# if/else – what happens

To evaluate:

```
if (expression)
    statement1;
else
    statement2;
```

- If (expression) **is true, then statement1 is executed and statement2 is skipped.**
- If (expression) **is false, then statement1 is skipped and statement2 is executed.**

# if/else – what happens



## 4.6 The `if/else if` Statement

- Chain of `if` statements that test in order until one is found to be true
- Also models thought processes:
  - “If it is raining, take an umbrella, else, if it is windy, take a hat, else, take sunglasses”



# if/else if format

```
if (expression)
    statement1;    // or block
else if (expression)
    statement2;    // or block
.
. // other else ifs
.
else if (expression)
    statementn;    // or block
```

## 4.7 Using a Trailing `else`

- Used with `if/else if` statement when none of `(expression)` is true
- Provides default statement/action
- Used to catch invalid values, other exceptional situations

## 4.8 Menus

- Menu-driven program: program execution controlled by user selecting from a list of actions
- Menu: list of choices on the screen
- Can be implemented using `if/else if` statements

# Menu-driven program organization

- Display list of numbered or lettered choices for actions
- Prompt user to make selection
- Test user selection in `(expression)`
  - if a match, then execute code for action
  - if not, then go on to next `(expression)`

## 4.9 Nested `if` Statements

- An `if` statement that is part of the `if` or `else` part of another `if` statement
- Can be used to evaluate  $> 1$  data item or condition:

```
if (score < 100)
{
    if (score > 90)
        grade = 'A';
}
```

# Notes on coding nested `if`s

- An `else` matches the nearest `if` that does not have an `else`:

```
if (score < 100)
    if (score > 90)
        grade = 'A';
    else ...// goes with second if,
            // not first one
```

- Proper indentation helps greatly

## 4.10 Logical Operators

- Used to create relational expressions from other relational expressions
- Operators, meaning, and explanation:

& &	AND	New relational expression is true if both expressions are true
	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

# Logical Operators - examples

```
int x = 12, y = 5, z = -4;
```

<code>(x &gt; y) &amp;&amp; (y &gt; z)</code>	true
<code>(x &gt; y) &amp;&amp; (z &gt; y)</code>	false
<code>(x &lt;= z)    (y == z)</code>	false
<code>(x &lt;= z)    (y != z)</code>	true
<code>! (x &gt;= z)</code>	false



# Logical Operators - notes

- ! has highest precedence, followed by & &, then | |
- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)

## 4.11 Checking Numeric Ranges with Logical Operators

- Used to test to see if a value falls into a range:

```
if (grade >= 0 && grade <= 100)
    cout << "Valid grade";
```

- Can also test to see if value falls outside of range:

```
if (grade <= 0 || grade >= 100)
    cout << "Invalid grade";
```

- Cannot use mathematical notation:

```
if (0 <= grade <= 100) //doesn't work!
```

## 4.12 Validating User Input

- Input validation: inspecting data to a program to determine if it is acceptable
- Bad output will be produced from bad input
- Can perform various tests:
  - Range
  - Reasonableness
  - Valid menu choice
  - Divide by zero

## 4.13 More About Variable Definitions and Scope

- Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Usually defined at beginning of function
- May be defined close to first use

# Still More About Variable Definitions and Scope

- Variables defined inside { } have local or block scope
- When inside a block within another block, can define variables with the same name as in the outer block.
  - When in inner block, outer definition is not available
  - Not a good idea

## 4.14 Comparing Strings

- Can not use relational operators with character strings
- Must use the `strcmp` function to compare C-strings
- `strcmp` compares the ASCII codes of the characters in the strings.  
Comparison is character-by-character

# Comparing Strings

`strcmp(str1, str2)`: compares strings `str1` and `str2`

- returns 0 if the strings are the same, negative number if `str1 < str2`, and positive number if `str1 > str2`

```
char myName[10] = "George";
```

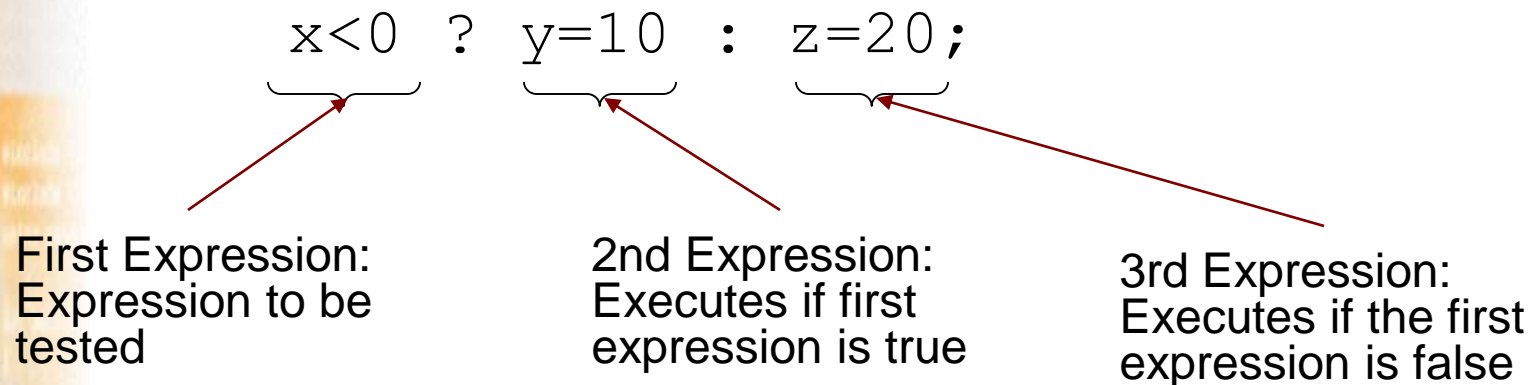
```
char yourName[10] = "Georgia";
```

```
if (strcmp(myName, yourName) < 0)
```

```
    cout << myName << " comes before "  
    << yourName << " in the alphabet";
```

# 4.15 The Conditional Operator

- Can use to create short `if/else` statements
- **Format:** `expr ? expr : expr;`





# The Conditional Operator

- The value of a conditional expression is
  - The value of the second expression if the first expression is true
  - The value of the third expression if the first expression is false
- Parentheses ( ) may be needed in an expression due to precedence of conditional operator

## 4.16 The `switch` Statement

- Used to select among statements from several alternatives
- May be used instead of `if/else if` statements

# switch statement format

```
switch (expression) //integer
{
    case exp1: statement1;
    case exp2: statement2;
    ...
    case expn: statementn;
    default:    statementn+1;
}
```

# switch statement requirements

- 1) `expression` must be an integer variable or an expression that evaluates to an integer value
- 2) `exp $1$`  through `exp $n$`  must be constant integer expressions or literals, and must be unique in the `switch` statement
- 3) `default` is optional but recommended

# switch statement – how it works

- 1) `expression` is evaluated
- 2) The value of `expression` is compared against `exp1` through `expn`.
- 3) If `expression` matches value `expi`, the program branches to the statement following `expi` and continues to the end of the `switch`
- 4) If no matching value is found, the program branches to the statement after `default`:

# break statement

- Used to stop execution in the current block
- Also used to exit a switch statement
- Useful to execute a single `case` statement without executing the statements following it

# Using `switch` with a menu

- `switch` statement is a natural choice for menu-driven program:
  - display menu
  - get user input
  - use user input as `expression` in `switch` statement
  - use menu choices as `expr` in `case` statements

## 4.17 Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");  
if (!infile)  
{  
    cout << "File open failure!";  
}
```

- Can also use the `fail` member function



# Standard Version of Starting Out with C++, 4th Edition

## Chapter 4 Making Decisions

