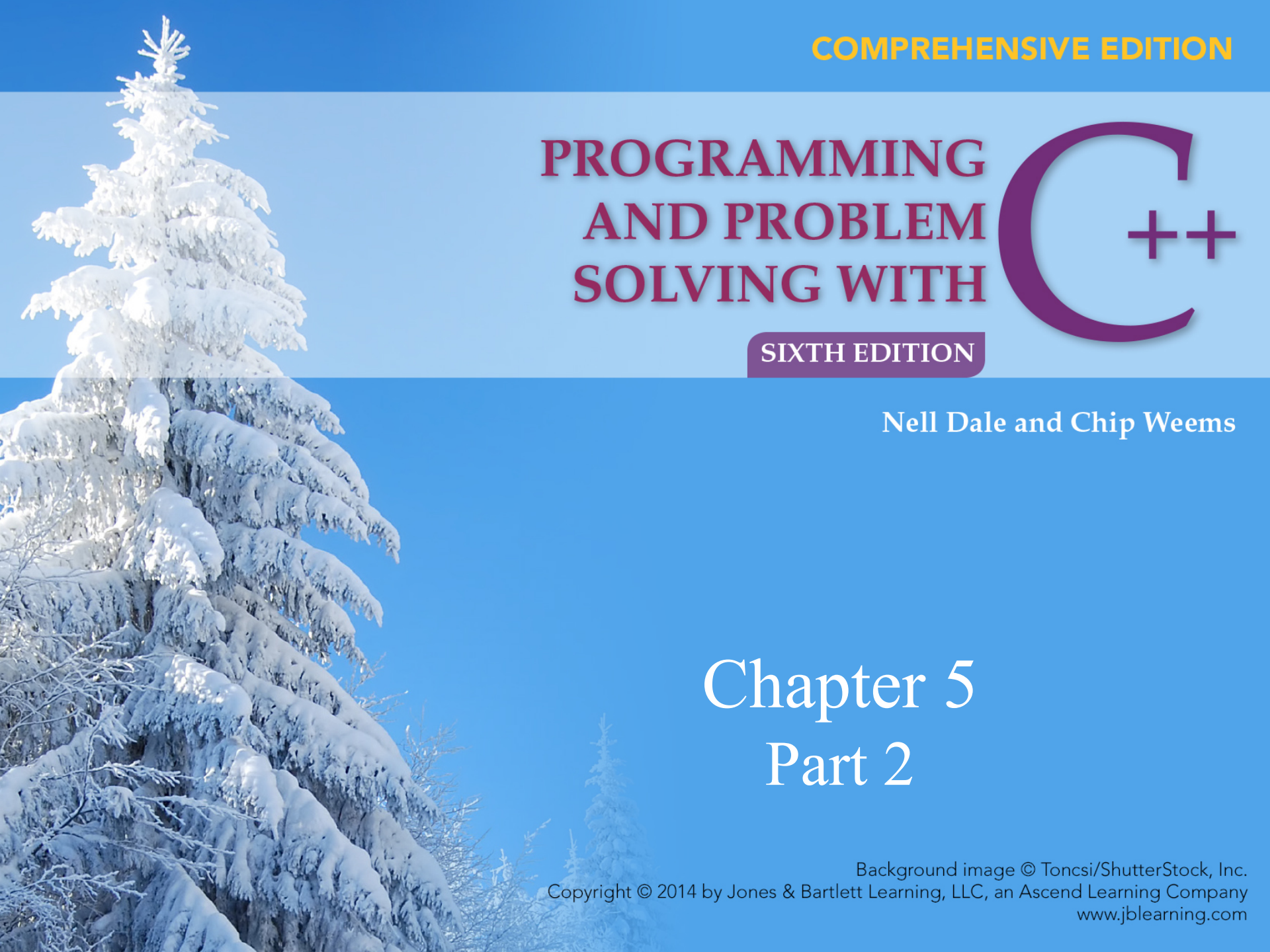


COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 5
Part 2

Exercise

- Write an algorithm and a flowchart for a season calculator.
- The calculator should take a month and day as input
- If the month is valid, then the output should be the season, otherwise the output should be an error message.
- Limit your data types, classes and objects to
  - basic data types
  - string
  - cin
  - cout
- Work in pairs or groups of three.

Expressions

Control structures use logical expressions to make choices, which may include:

6 Relational Operators

< <= > >= == !=

3 Logical Operators

! && ||

Operator	Meaning	Associativity
!	NOT	Right
*, / , %	Multiplication, Division, Modulus	Left
+ , -	Addition, Subtraction	Left
<	Less than	Left
<=	Less than or equal to	Left
>	Greater than	Left
>=	Greater than or equal to	Left
==	Is equal to	Left
!=	Is not equal to	Left
&&	AND	Left
||	OR	Left
=	Assignment	Right

Logical Expression	Meaning	Description
! p	NOT p	! p is false if p is true ! p is true if p is false
p && q	p AND q	p && q is true if both p and q are true. It is false otherwise.
p || q	p OR q	p || q is true if either p or q or both are true. It is false otherwise.

```
int   age;
bool  isSenior,  hasFever;
float temperature;

age = 20;
temperature = 102.0;
isSenior = (age >= 55); // isSenior is  false
hasFever = (temperature > 98.6);
// hasFever is  true
```

	Expression	Value
	isSenior && hasFever	
	isSenior || hasFever	
	! isSenior	
	! hasFever	

What is the value?

```
int age, height;
age = 25;
height = 70;
```

Expression	Value
!(age < 10)	?
!(height > 60)	?

"Short-Circuit" Evaluation

- C++ uses short circuit evaluation of logical expressions
- This means logical expressions are evaluated left to right and evaluation stops as soon as the final truth value can be determined

Short-Circuit Example

```
int age, height;

age = 25;
height = 70;
```

Expression

(age > 50) && (height > 60)

false

Evaluation can stop now because result of && is only true when both sides are true; thus it is already determined the expression will be false

More Short-Circuiting

```
int   age, height;

age = 25;
height = 70;
```

Expression

(height > 60) || (age > 40)

true

Evaluation can stop now because result of || is true if either side is true; thus it is already determined that the expression will be true

What happens?

```
int age, weight;
age = 25;
weight = 145;
```

Expression

(weight < 180) && (age >= 20)

true

Must still be evaluated because truth value of entire expression is not yet known

(Why?)

What happens?

```
int   age, height;

age = 25;
height = 70;
```

Expression

```
! (height > 60)  ||  (age > 50)
```

true

false

Does this part need to be evaluated?

Write an expression for each

- taxRate is over 25% and income is less than $20,000

```
(taxRate > .25) &&
  (income < 20000)
```

- temperature is less than or equal to 75° or humidity is less than 70%

```
(temperature <= 75) ||
  (humidity < .70)
```

Write an expression for each

- age is over 21 and age is less than 60

```
(age > 21) &&
  (age < 60)
```

- age is 21 or 22

```
(age == 21) ||
  (age == 22)
```

Use Precedence Chart

```
int    number;
float  x;
```

```
number != 0 && x < 1 / number
```

/ has highest priority

< next priority

!= next priority

&& next priority

What happens if Number has value 0?

Run Time Error (Division by zero) occurs

Short-Circuit Benefits

- One Boolean expression can be placed first to "guard" a potentially unsafe operation in a second Boolean expression
- Time is saved in evaluation of complex expressions using operators || and &&

Our Example Revisited

```
int    number;
float  x;

(number  !=  0)  &&  (x  <  1  /  number)
```

is evaluated first and has value false

Because operator is &&, the entire expression will have value false; because of short-circuiting, the right side is not evaluated in C++

Warning About Expression in C++

- "Boolean expression" means an expression whose value is true or false
- An expression is any valid combination of operators and operands

Warning About Expression in C++

- Each expression has a value, which can lead to unexpected results
- Construct your expressions carefully
  - use precedence chart to determine order
  - use parentheses for clarification (and safety)

What went wrong?

This is only supposed to display "HEALTHY AIR" if the air quality index is between 50 and 80.

But when you tested it, it displayed "HEALTHY AIR" when the index was 35.

```
int  AQIndex;
AQIndex = 35;

if (50 < AQIndex < 80)
        cout << "HEALTHY AIR";
```

Analysis of Situation

AQIndex = 35;

According to the precedence chart, the expression

(50 < AQIndex < 80) ***means***

(50 < AQIndex) < 80 ***because < is Left Associative***

(50 < AQIndex) is false ***(has value 0)***

(0 < 80) is true.

Corrected Version

```
int  AQIndex;
AQIndex = 35;

if ((50 < AQIndex) && (AQIndex < 80))

    cout << “HEALTHY AIR“;
```

Comparing Real Values

Do not compare floating point values for equality, compare them for near-equality.

```
float myNumber;
float yourNumber;

cin >> myNumber;
cin >> yourNumber;

if (fabs (myNumber - yourNumber) <
0.00001)
    cout << "They are close enough!"
         << endl;
```

Comparing Strings

- We can use the relational operators to compare strings because the relational operators are **overloaded**
- The `string` class contains a function called `compare` to determine if one string comes before another
- A call to `name1.compare(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

Comparing Strings

```
if (name1.compare(name2) < 0)
   cout << name1 << " comes first.";
else
   if (name1.compare(name2) == 0)
      cout << "Same name";
   else
      cout << name2  << " comes first."<< endl;
```

- Because comparing characters and strings is based on a character set, it is called a *lexicographic ordering*

Lexicographic Ordering

- Lexicographic ordering is not strictly alphabetical when uppercase and lowercase characters are mixed
- For example, the string `"Great"` comes before the string `"fantastic"` because all of the uppercase letters come before all of the lowercase letters in Unicode
- Also, short strings come before longer strings with the same prefix (lexicographically)
- Therefore `"book"` comes before `"bookcase"`

== operator for string

- == is an operator function that takes two arguments as follows
- inline bool operator==(const string& lhs, const string& rhs){
  /* do actual comparison */
  return (lhs.compare(rhs) == 0);
  }

What can go wrong here?

```
float   average;
float   total;
int     howMany;
   .
   .
   .
average = total / howMany;
```

Improved Version

```
float  average,
float  total;
int     howMany;

if (howMany >  0)
{
    average = total  / howMany;
    cout <<  average;
}
else
    cout << "No prices were entered";
```

Example

```
// Where is first 'A' found in a string?
string   myString;
string::size_type   pos;
   .  .  .
pos  =  myString.find('A');

if  (pos == string::npos)
    cout  <<  "No 'A' was found" <<  endl;
else
    cout  <<  "An 'A' was found in position "
          <<  pos   <<  endl;
```

These are equivalent. *Why?*

```
if  (number == 0)
{
        .
        .
        .
}
```

```
if  (! number )
{
        .
        .
        .
 }
```

Each expression is only true when number has value 0

In the absence of braces,

- an `else` is always paired with the closest preceding `if` that doesn't already have an `else` paired with it

Example

```
float average;

average = 100.0;

if   (average >= 60.0)
     if (average < 70.0)
        cout << "Marginal PASS";
else
     cout << "FAIL";
```

FAIL is printed; WHY? **The compiler ignores indentation and pairs the else with the second if**

Use Braces to Correct Problem

```
float average;

average = 100.0;

if  (average >= 60.0)
{
    if (average < 70.0)
        cout << "Marginal PASS";
}
else
    cout << "FAIL";
```

Each I/O stream has a state (condition)

- An **input stream** enters fail state when you
  - try to read invalid input data
  - try to open a file which does not exist
  - try to read beyond the end of the file

- An **output stream** enters fail state when you
  - try to create a file with an invalid name
  - try to create a file on a write-protected disk
  - try to create a file on a full disk

Determining the Stream State

- The stream identifier can be used as if it were a Boolean variable that has value false when the last I/O operation on that stream failed and has value true when it did not fail
- After you use a file stream, you should check on its state

Checking the State

```
ofstream  myOutfile;

myOutfile.open ("myOut.dat");

if  (! myOutfile)
{
  cout  <<  "File opening  error.  "
        <<  "Program terminated."  <<  endl;
  return 1;
}
// Otherwise send output to myOutfile
```

Testing Selection Control Structures

- To test a program with branches, use enough data sets to ensure that every branch is executed at least once
- This strategy is called **minimum complete coverage**

Testing Often Combines Two Approaches

WHITE BOX TESTING

Code Coverage

Allows us to see the program code while designing the tests, so that data values at the boundaries, and possibly middle values, can be tested.

BLACK BOX TESTING

Data Coverage

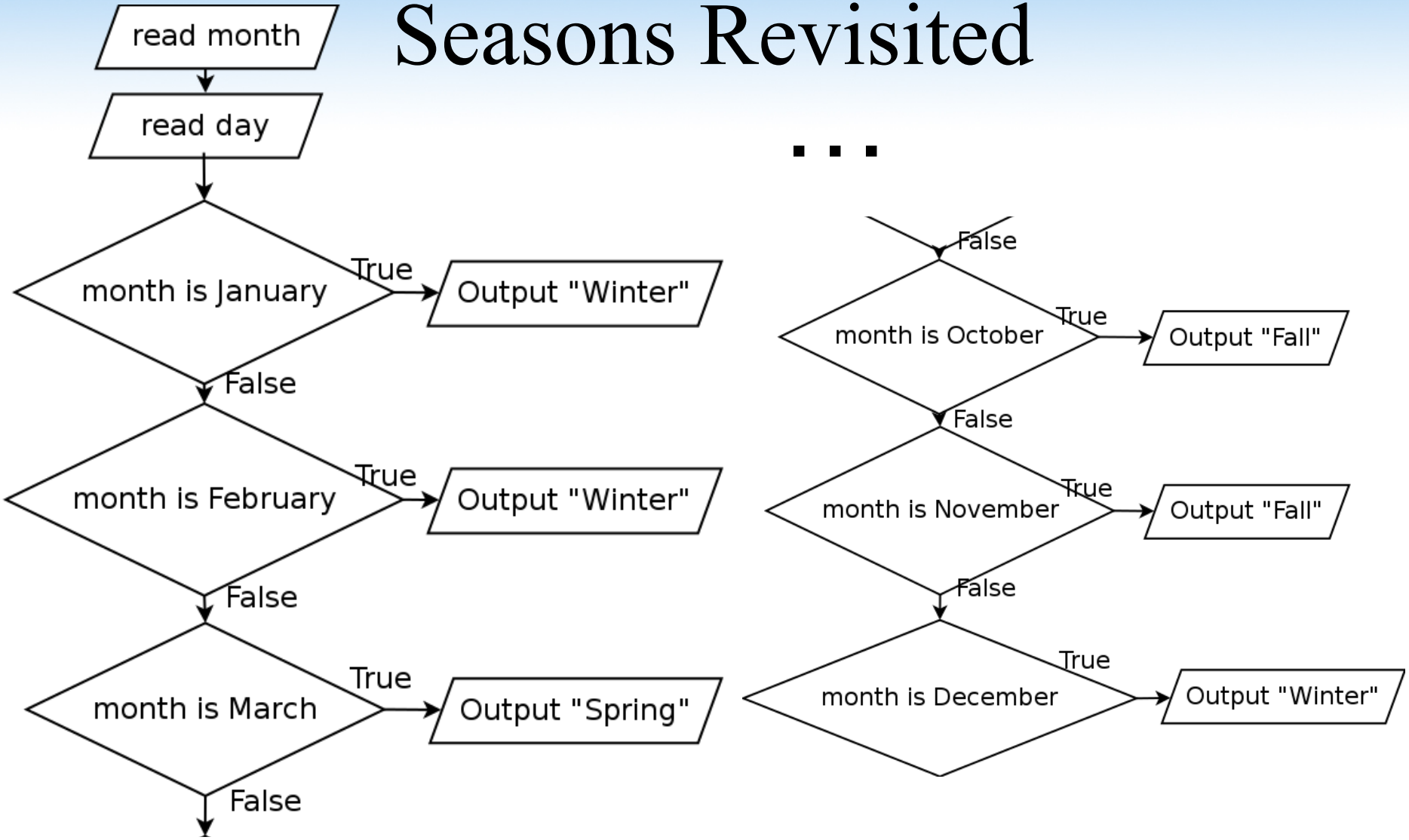
Tries to test as many allowable data values as possible without regard to program code.

Testing

- Design and implement a test plan
- A test plan is a document that specifies the test cases to try, the reason for each, and the expected output
- Implement the test plan by verifying that the program outputs the predicted results

PHASE	RESULT	TESTING TECHNIQUE
Problem solving	Algorithm	Algorithm walk-through
Implementation	Coded program	Code walk-through, Trace
Compilation	Object program	Compiler messages
Execution	Output	Implement test plan

Seasons Revisited



Checking Day