Object Oriented Programming by C++

Selection & Repetition (2/2)

Conditional execution and Iteration (Loop)

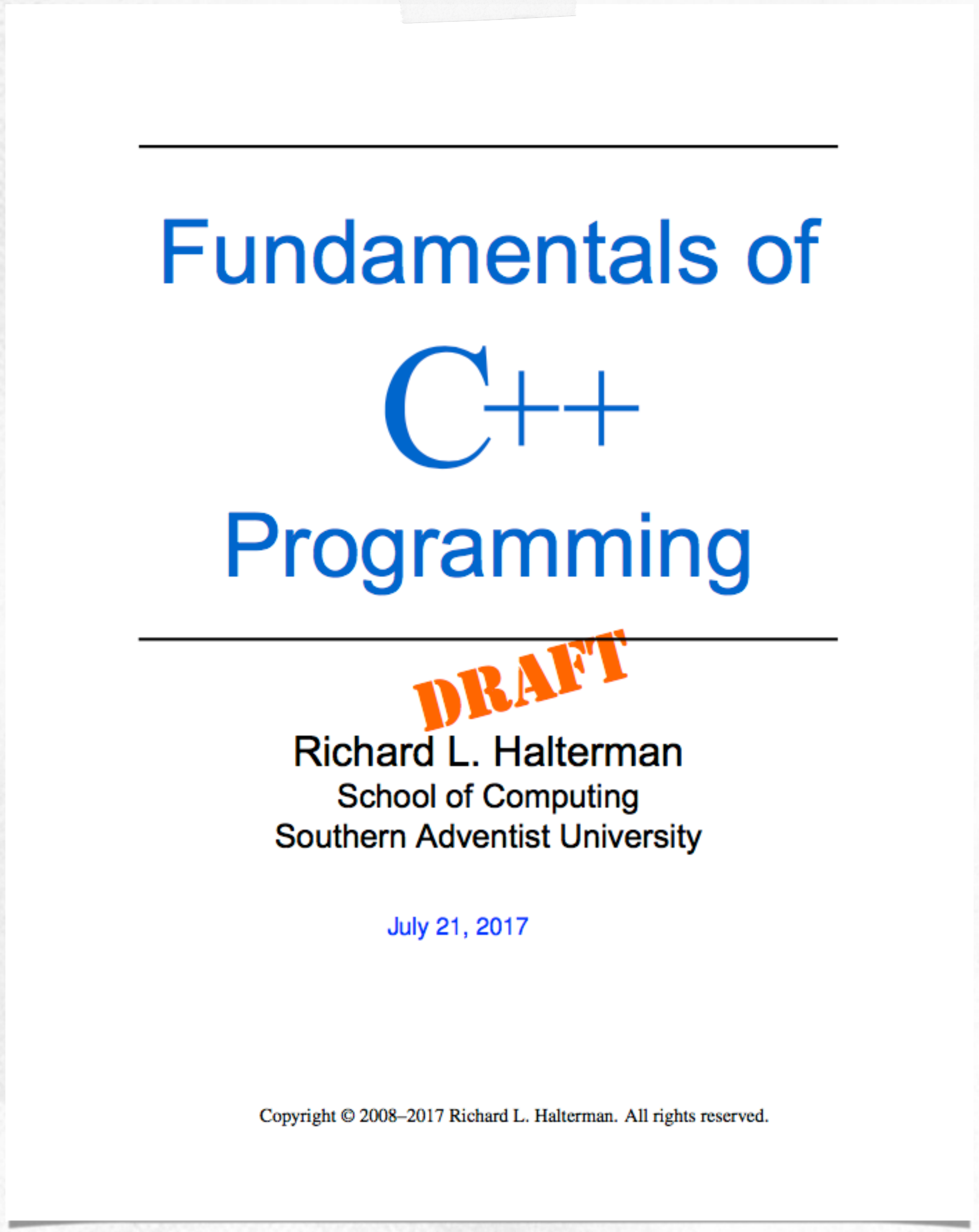
2017. 8.

Sungwon Lee / Professor

Email: drsungwon@khu.ac.kr
Web: http://mobilelab.khu.ac.kr/

Textbook & Copyright

- Textbook: http://python.cs.southern.edu/cppbook/progcpp.pdf
- Sample Codes: https://github.com/halterman/CppBook-SourceCode



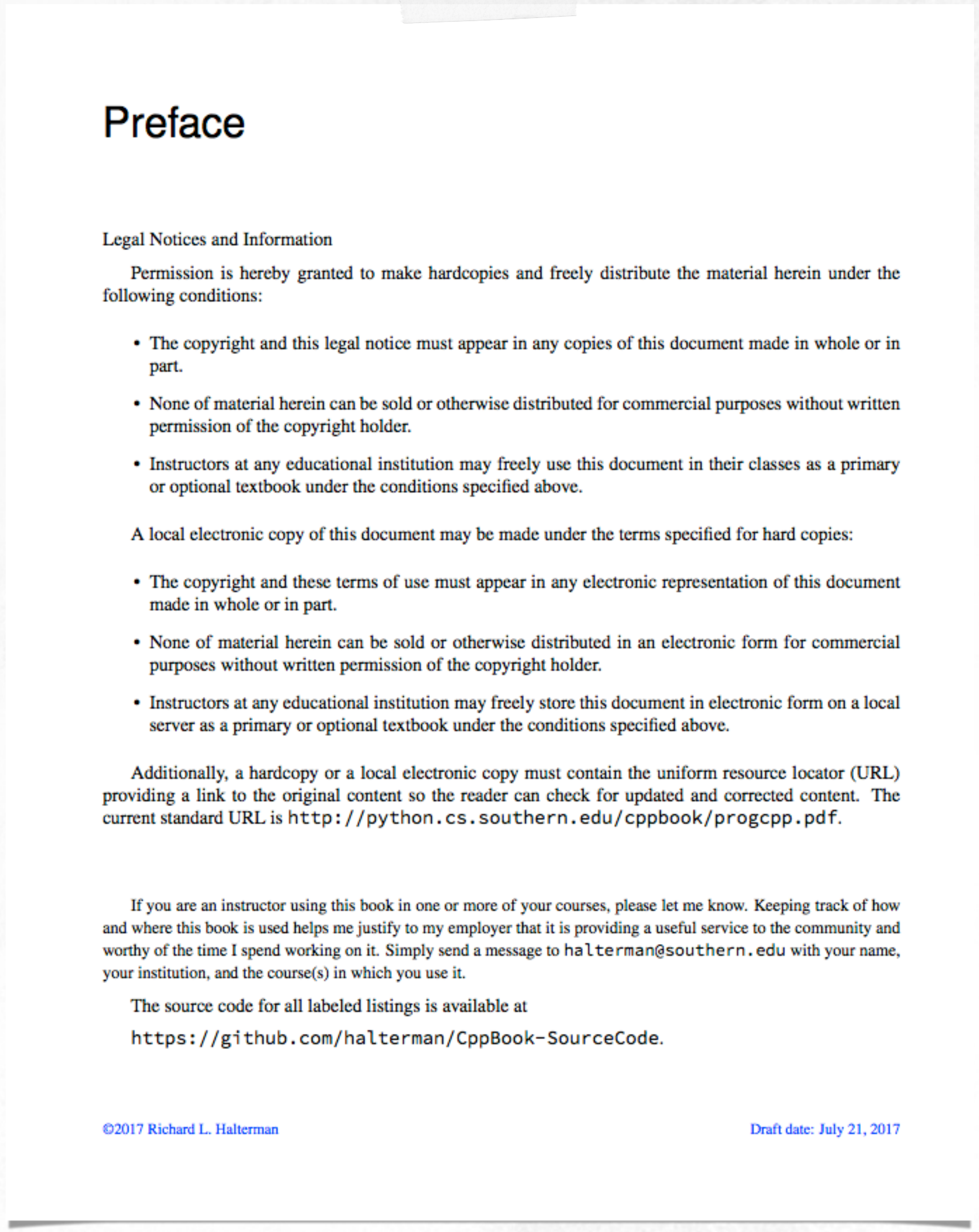
Fundamentals of C++ Programming

DRAFT

Richard L. Halterman
School of Computing
Southern Adventist University

July 21, 2017

Copyright © 2008–2017 Richard L. Halterman. All rights reserved.



Preface

Legal Notices and Information

Permission is hereby granted to make hardcopies and freely distribute the material herein under the following conditions:

- The copyright and this legal notice must appear in any copies of this document made in whole or in part.
- None of material herein can be sold or otherwise distributed for commercial purposes without written permission of the copyright holder.
- Instructors at any educational institution may freely use this document in their classes as a primary or optional textbook under the conditions specified above.

A local electronic copy of this document may be made under the terms specified for hard copies:

- The copyright and these terms of use must appear in any electronic representation of this document made in whole or in part.
- None of material herein can be sold or otherwise distributed in an electronic form for commercial purposes without written permission of the copyright holder.
- Instructors at any educational institution may freely store this document in electronic form on a local server as a primary or optional textbook under the conditions specified above.

Additionally, a hardcopy or a local electronic copy must contain the uniform resource locator (URL) providing a link to the original content so the reader can check for updated and corrected content. The current standard URL is http://python.cs.southern.edu/cppbook/progcpp.pdf.

If you are an instructor using this book in one or more of your courses, please let me know. Keeping track of how and where this book is used helps me justify to my employer that it is providing a useful service to the community and worthy of the time I spend working on it. Simply send a message to halterman@southern.edu with your name, your institution, and the course(s) in which you use it.

The source code for all labeled listings is available at

https://github.com/halterman/CppBook-SourceCode.

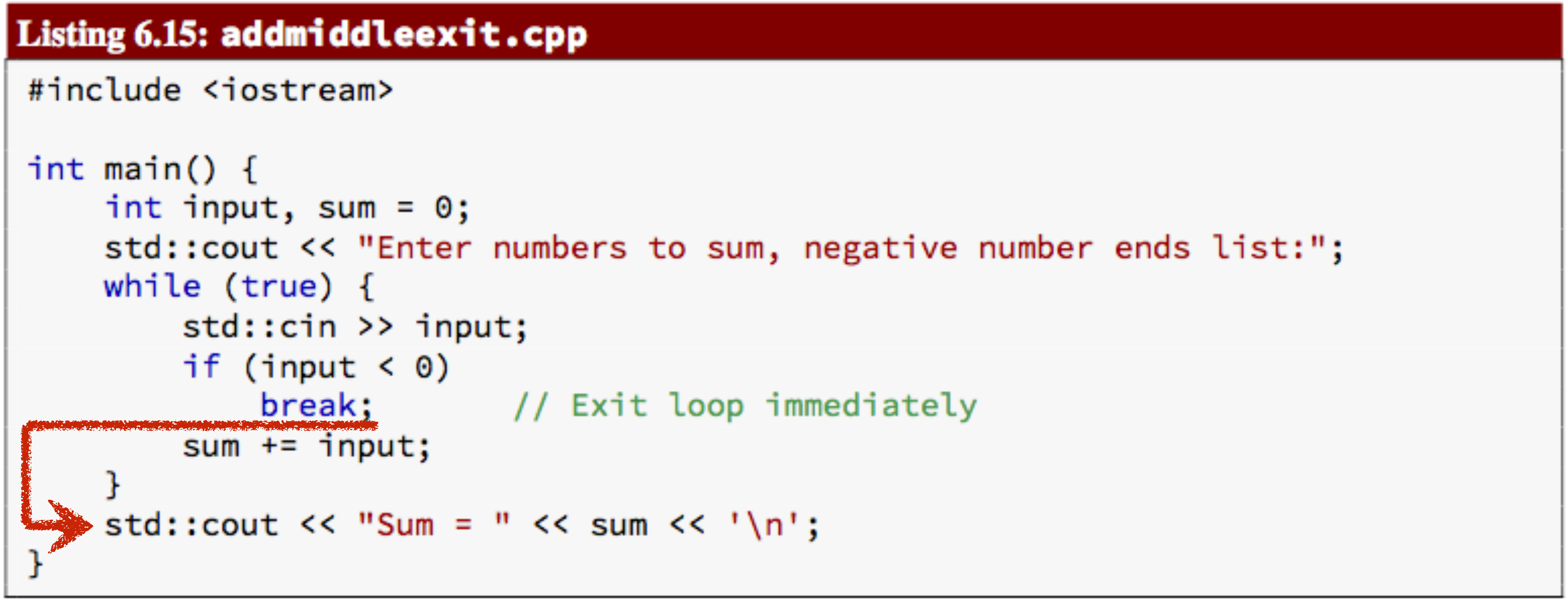
©2017 Richard L. Halterman Draft date: July 21, 2017

Contents

Abnormal Loop Termination

break statement

- *break*: causes the immediate exit from the body of the loop



Listing 6.15: addmiddleexit.cpp

```cpp
#include <iostream>

int main() {
    int input, sum = 0;
    std::cout << "Enter numbers to sum, negative number ends list:";
    while (true) {
        std::cin >> input;
        if (input < 0)
            break;        // Exit loop immediately
        sum += input;
    }
    std::cout << "Sum = " << sum << '\n';
}
```

if (input < 0) is true;
  Step.1: get out of while{…} statement,
  Step.2: go to next the line after while(){…} statement

Abnormal Loop Termination

continue statement

- *continue*: causes the immediate jump to the start of the loop

Listing 6.15: addmiddleexit.cpp

```
#include <iostream>

int main() {
    int input, sum = 0;
    std::cout << "Enter numbers to sum, negative number ends list:";
    while (true) {
        std::cin >> input;
        if (input < 0)
            continue:          // Exit loop immediately
        sum += input;
    }
    std::cout << "Sum = " << sum << '\n';
}
```

if (input < 0) is true;
  Step.1: get out of while{…} statement,
  Step.2: go to the *condition* of while(){…} statement

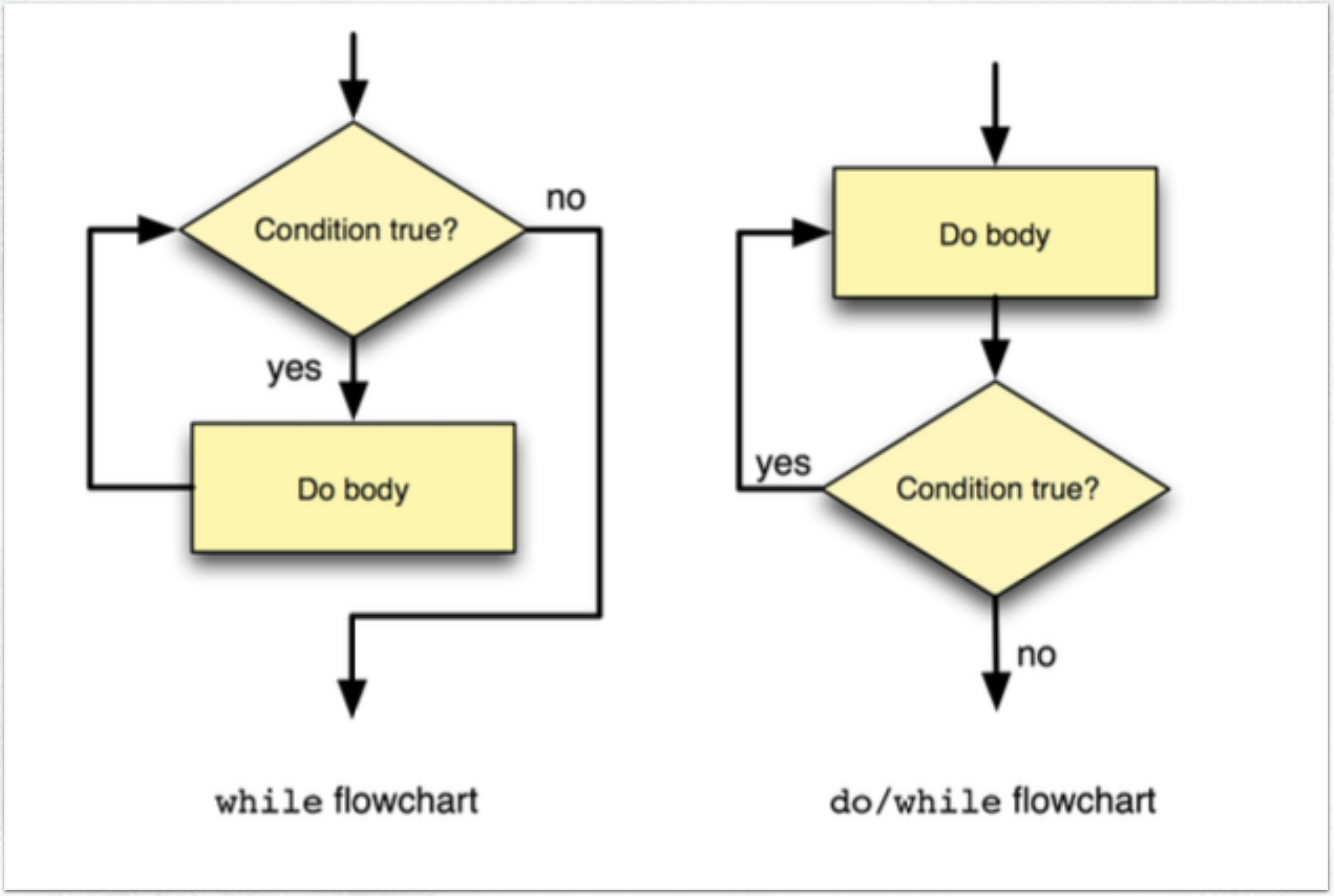
Abnormal Loop Termination

goto statement

Don’t use !!

while vs. do-while

- "Just do it! "



Example using *while* Statement

Listing 7.2: goodinputonly.cpp

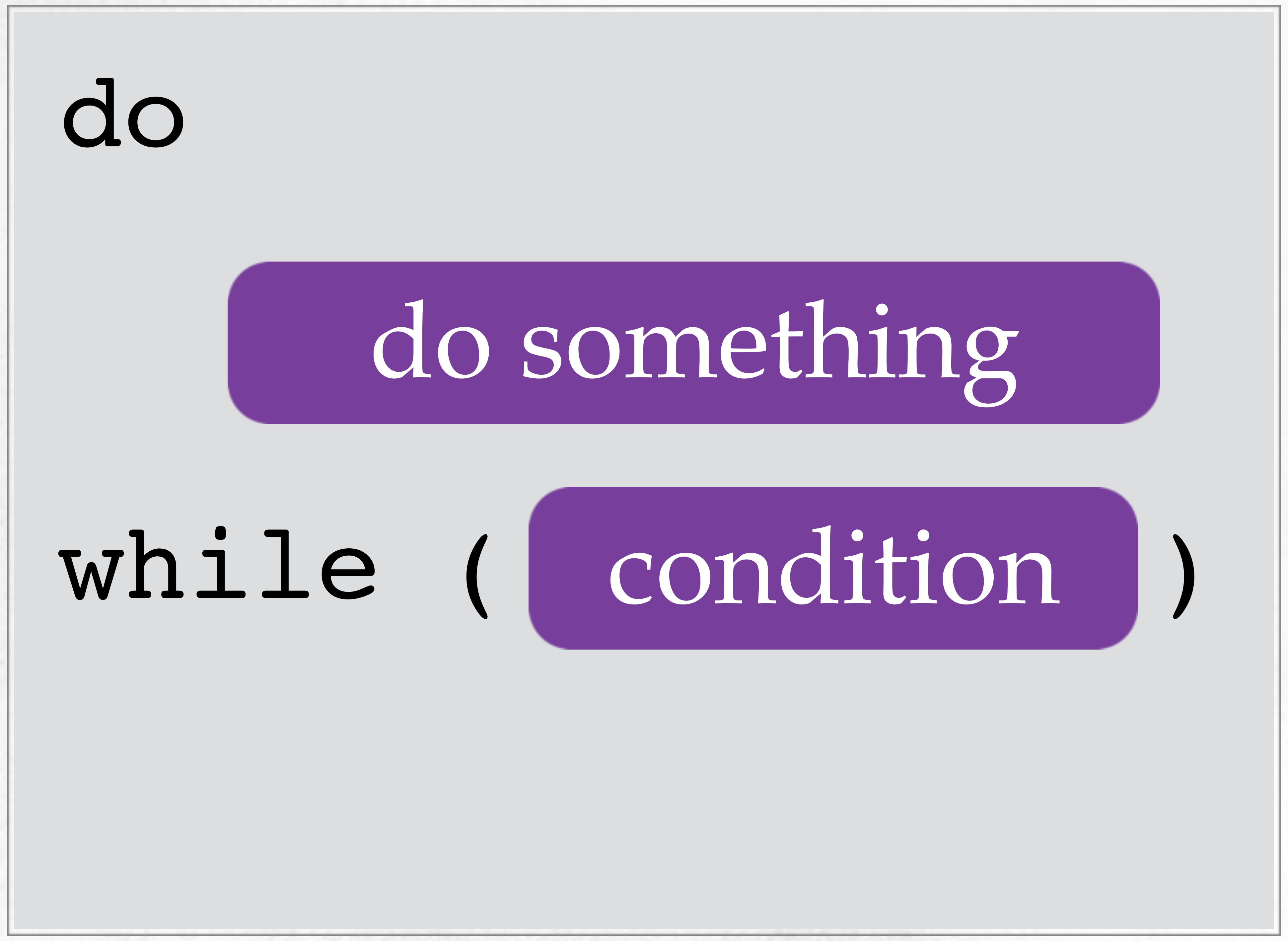
```cpp
#include <iostream>

int main() {
    int in_value = -1;
    std::cout << "Please enter an integer in the range 0-10: ";
    // Insist on values in the range 0...10
    while (in_value < 0 || in_value > 10)
        std::cin >> in_value;
    // in_value at this point is guaranteed to be within range
    std::cout << "Legal value entered was " << in_value << '\n';
}
```

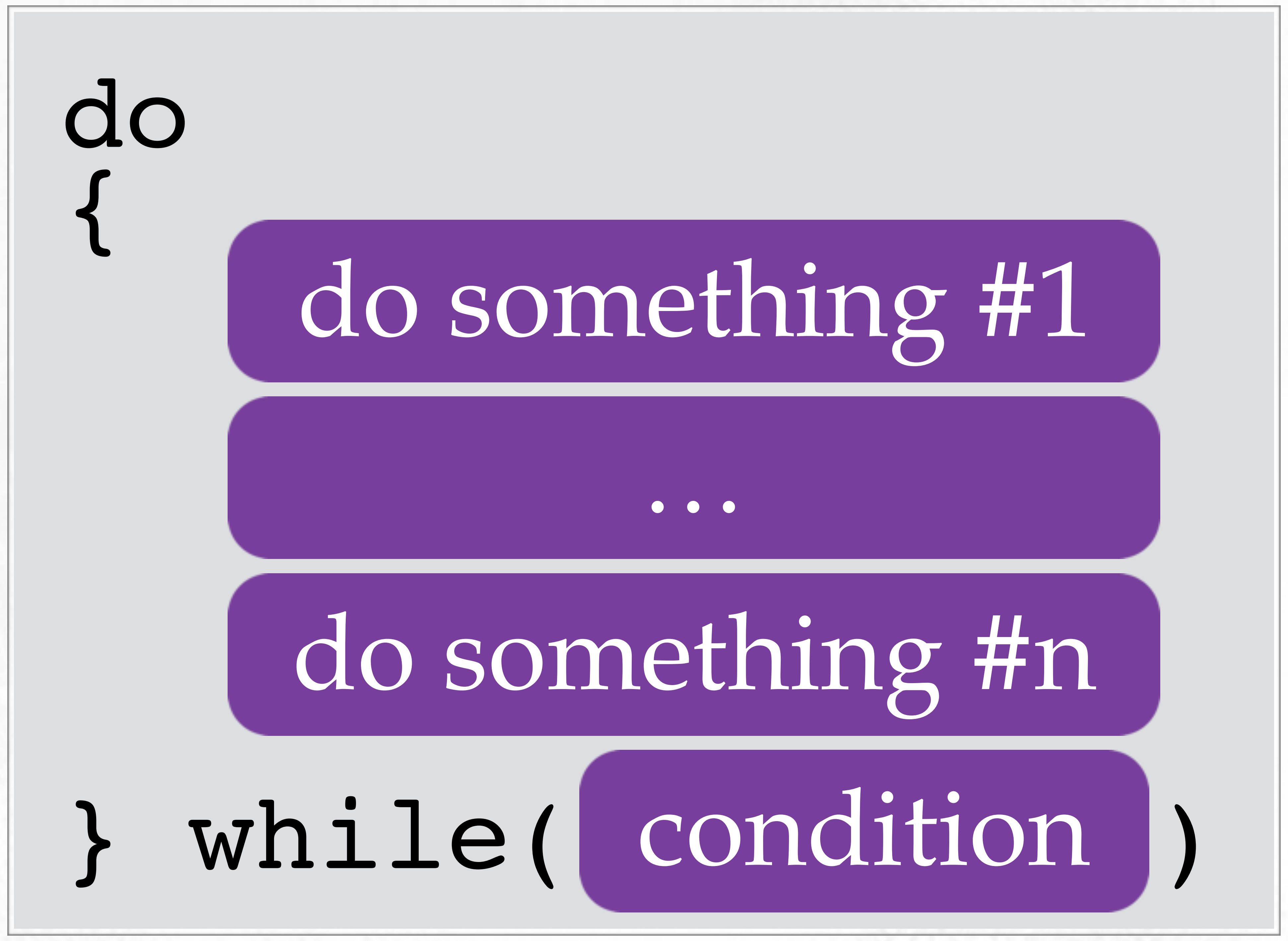
do-while Statement

Example using *while* Statement

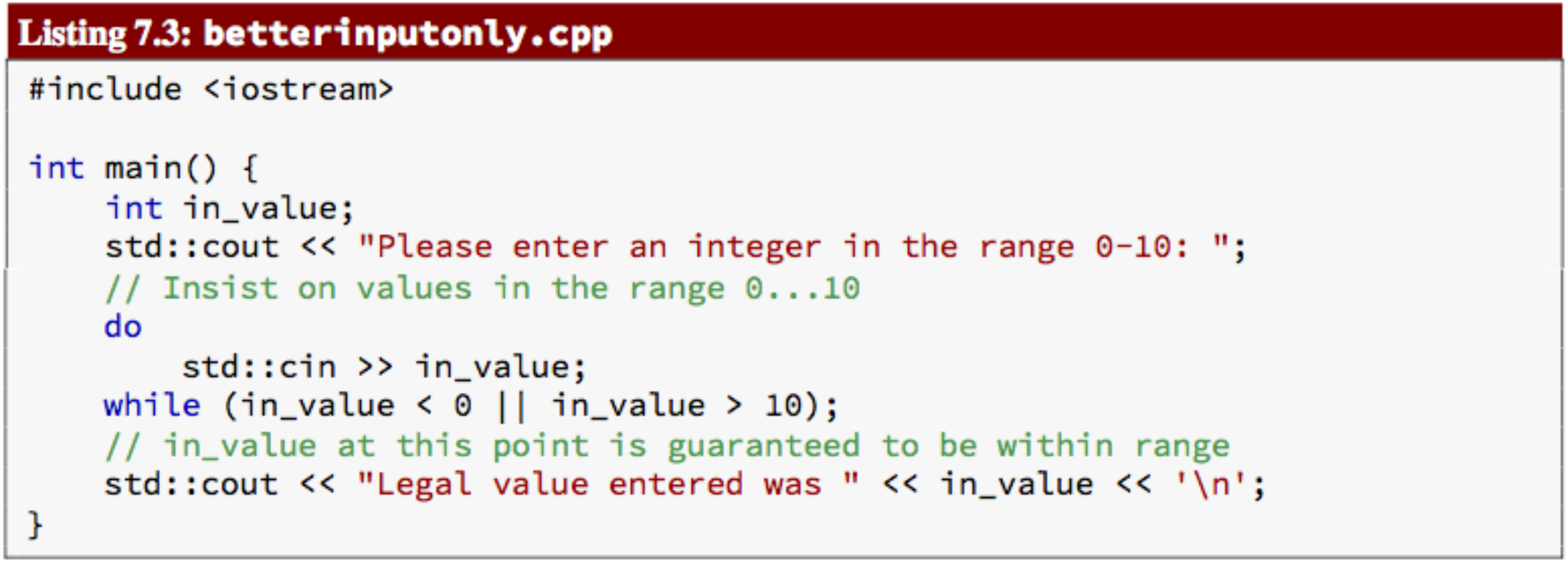
- *" Just do it! "*

- Iteration
  - Single statement Iteration
  - Multiple statement Iteration


```
do

      do something

while ( condition )
```


```
do
{
      do something #1
      ...
      do something #n
} while( condition )
```

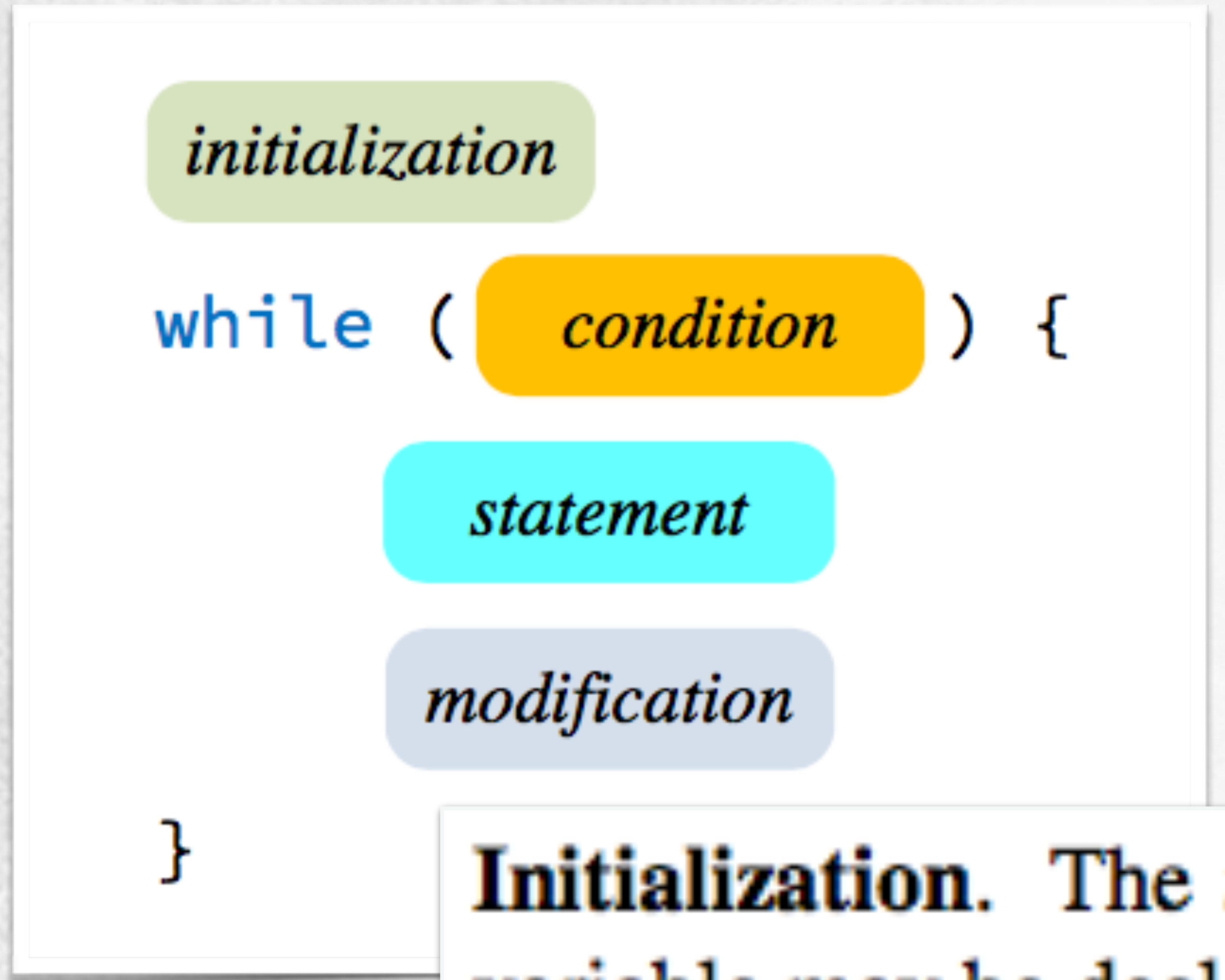
Example using *do-while* Statement



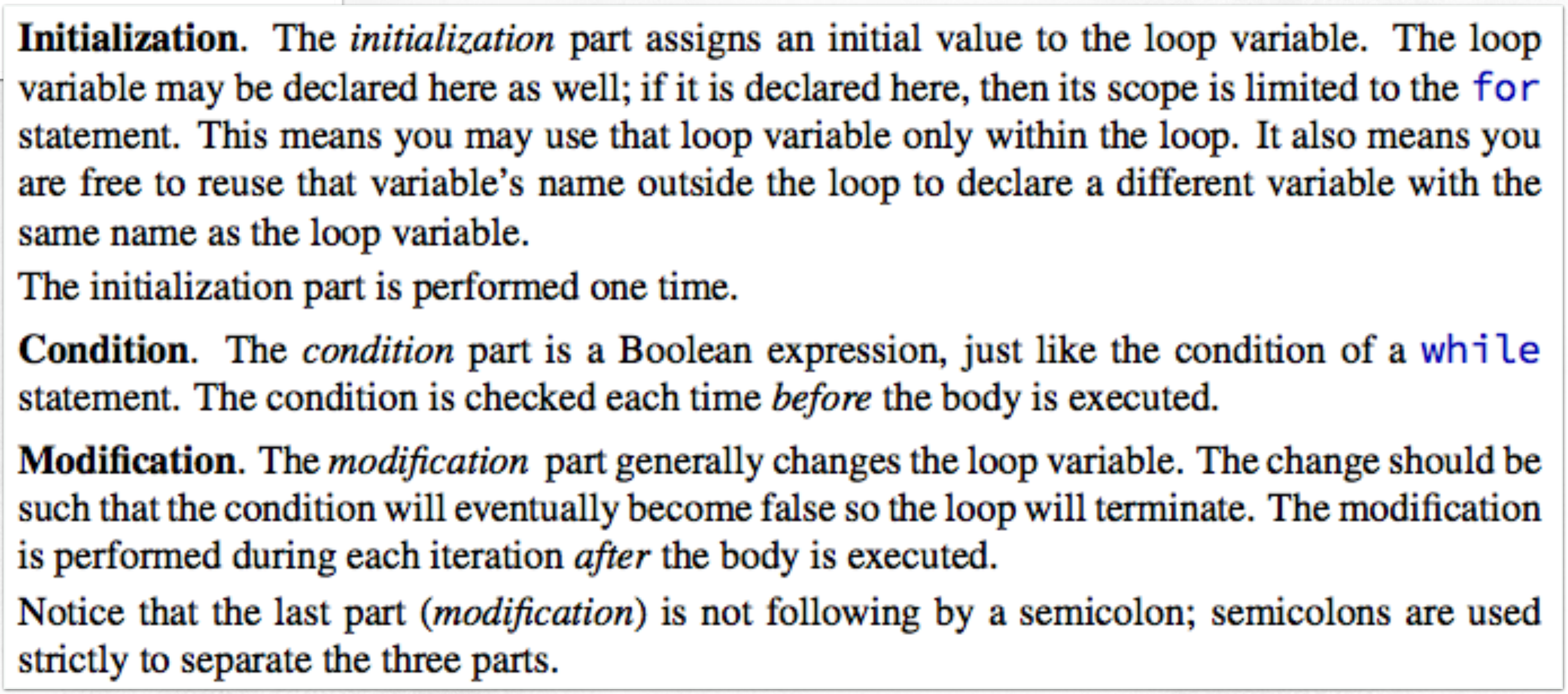
Listing 7.3: betterinputonly.cpp

```
#include <iostream>

int main() {
    int in_value;
    std::cout << "Please enter an integer in the range 0-10: ";
    // Insist on values in the range 0...10
    do
        std::cin >> in_value;
    while (in_value < 0 || in_value > 10);
    // in_value at this point is guaranteed to be within range
    std::cout << "Legal value entered was " << in_value << '\n';
}
```

Component of Loop Statement



```
initialization

while ( condition ) {

        statement

        modification

}
```



Initialization. The *initialization* part assigns an initial value to the loop variable. The loop variable may be declared here as well; if it is declared here, then its scope is limited to the `for` statement. This means you may use that loop variable only within the loop. It also means you are free to reuse that variable's name outside the loop to declare a different variable with the same name as the loop variable.

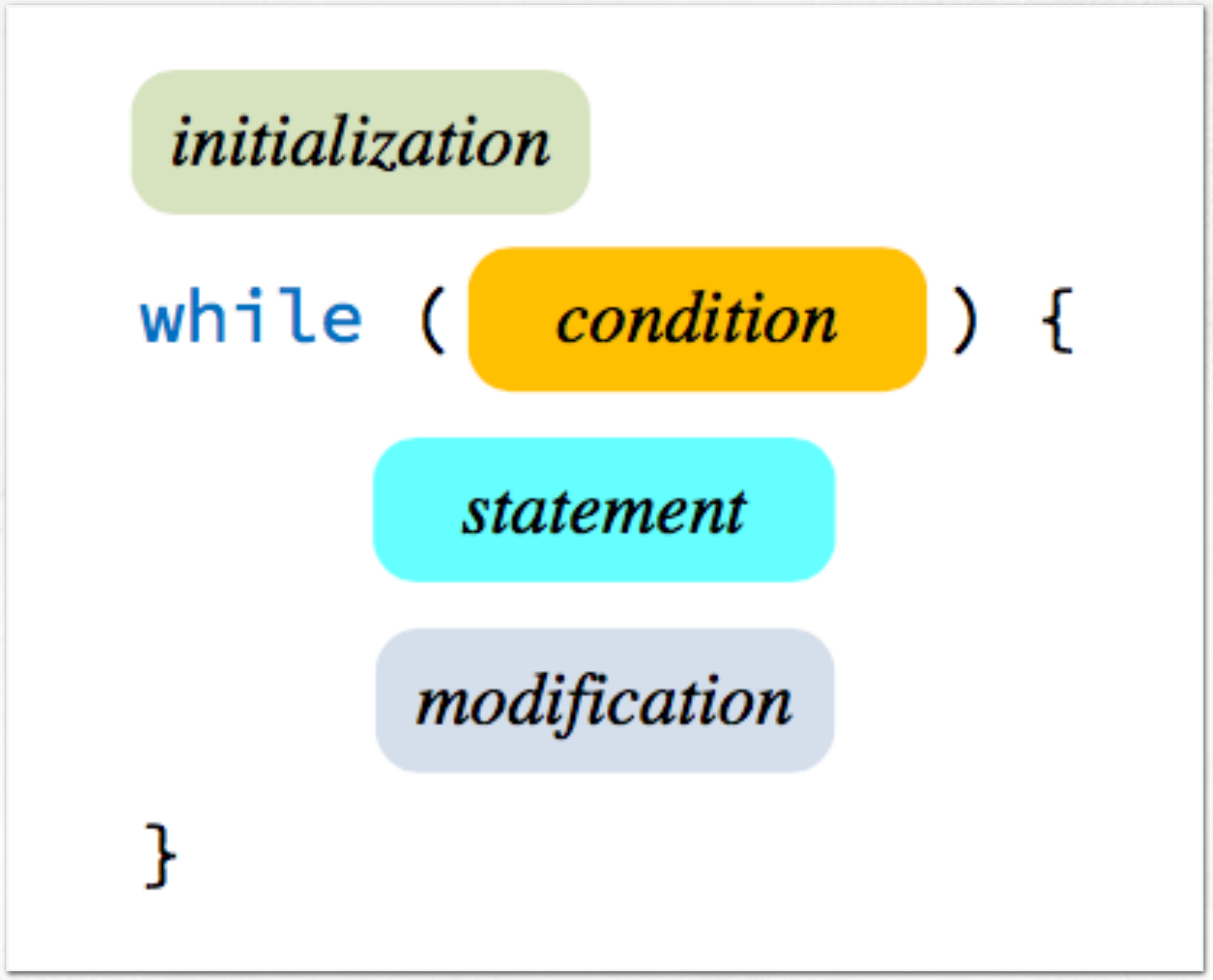
The initialization part is performed one time.

Condition. The *condition* part is a Boolean expression, just like the condition of a `while` statement. The condition is checked each time *before* the body is executed.

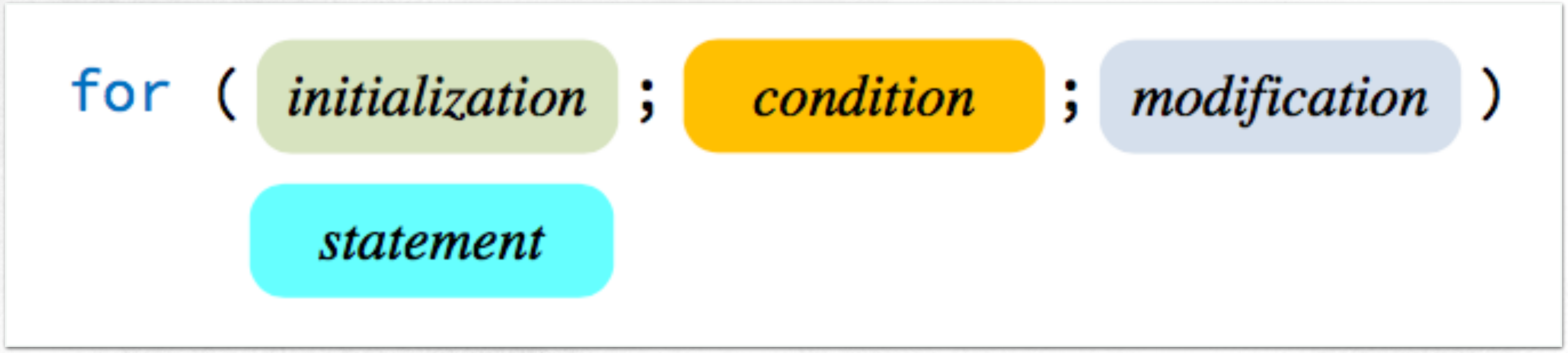
Modification. The *modification* part generally changes the loop variable. The change should be such that the condition will eventually become false so the loop will terminate. The modification is performed during each iteration *after* the body is executed.

Notice that the last part (*modification*) is not following by a semicolon; semicolons are used strictly to separate the three parts.

for Statement Description

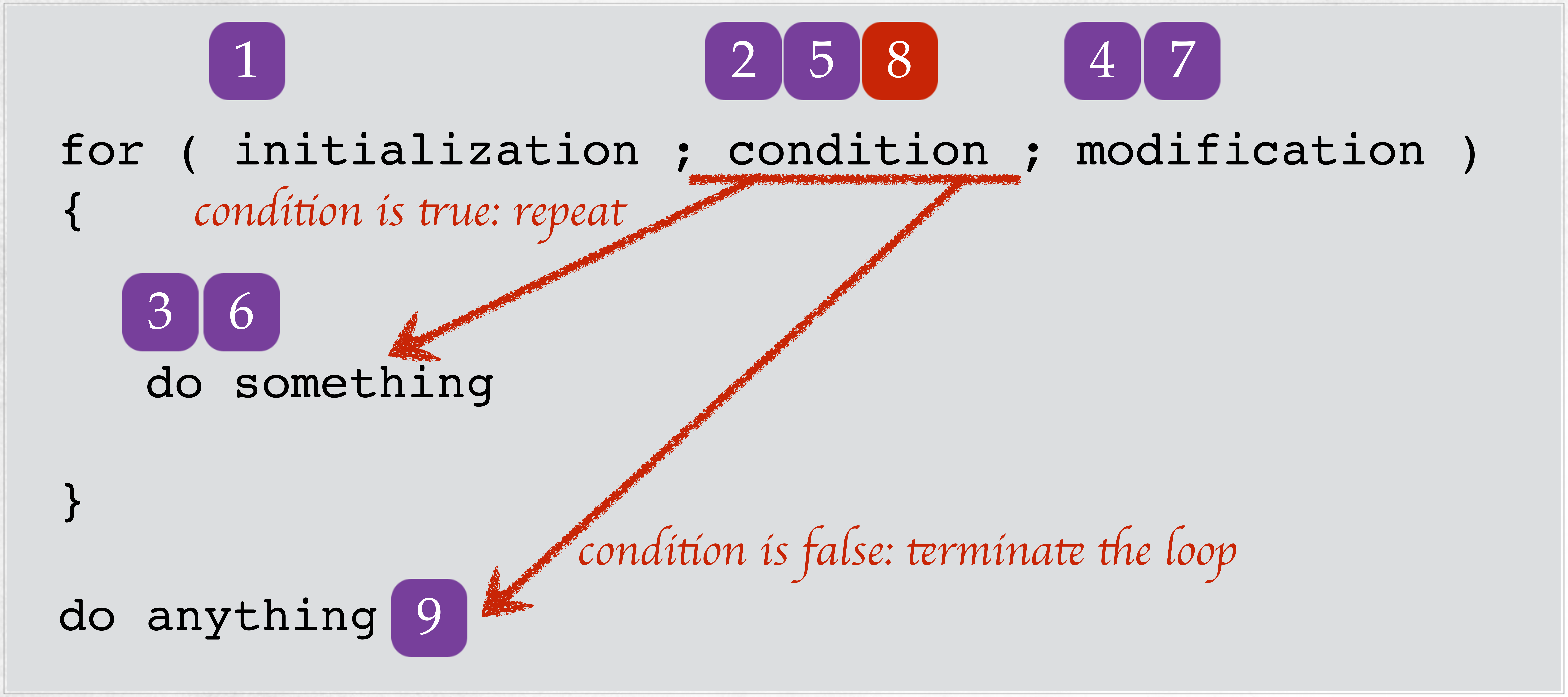


```
initialization
while ( condition ) {
        statement
        modification
}
```



```
for ( initialization ; condition ; modification )
      statement
```

Loop Sequence in *for* Statement



```
for ( initialization ; condition ; modification )
{
    do something
}
do anything
```

1
2
5
8
4
7
condition is true: repeat
3
6
condition is false: terminate the loop
9

For example: if (condition) is false at sequence ‘8’ then
 Terminate the *for* loop statement,
 Execute the next line (of *for* loop statement).

Example using *for* Statement

Listing 7.4: forcounttofive.cpp

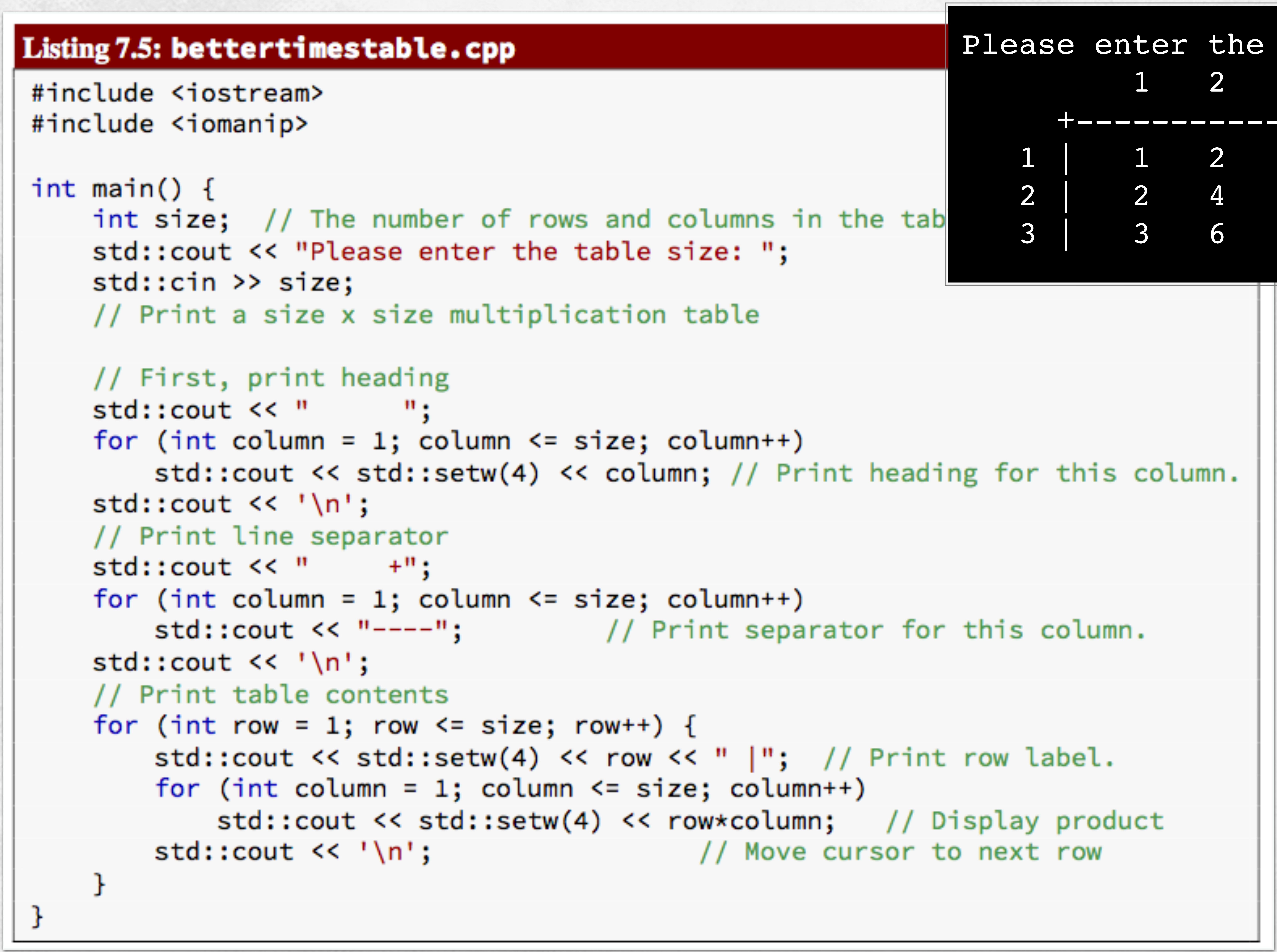
```
#include <iostream>

int main() {
    for (int count = 1; count <= 5; count++)
        std::cout << count << '\n';  // Display counter
}
```

Example using *nested-for* Statement

Listing 7.5: bettertimestable.cpp

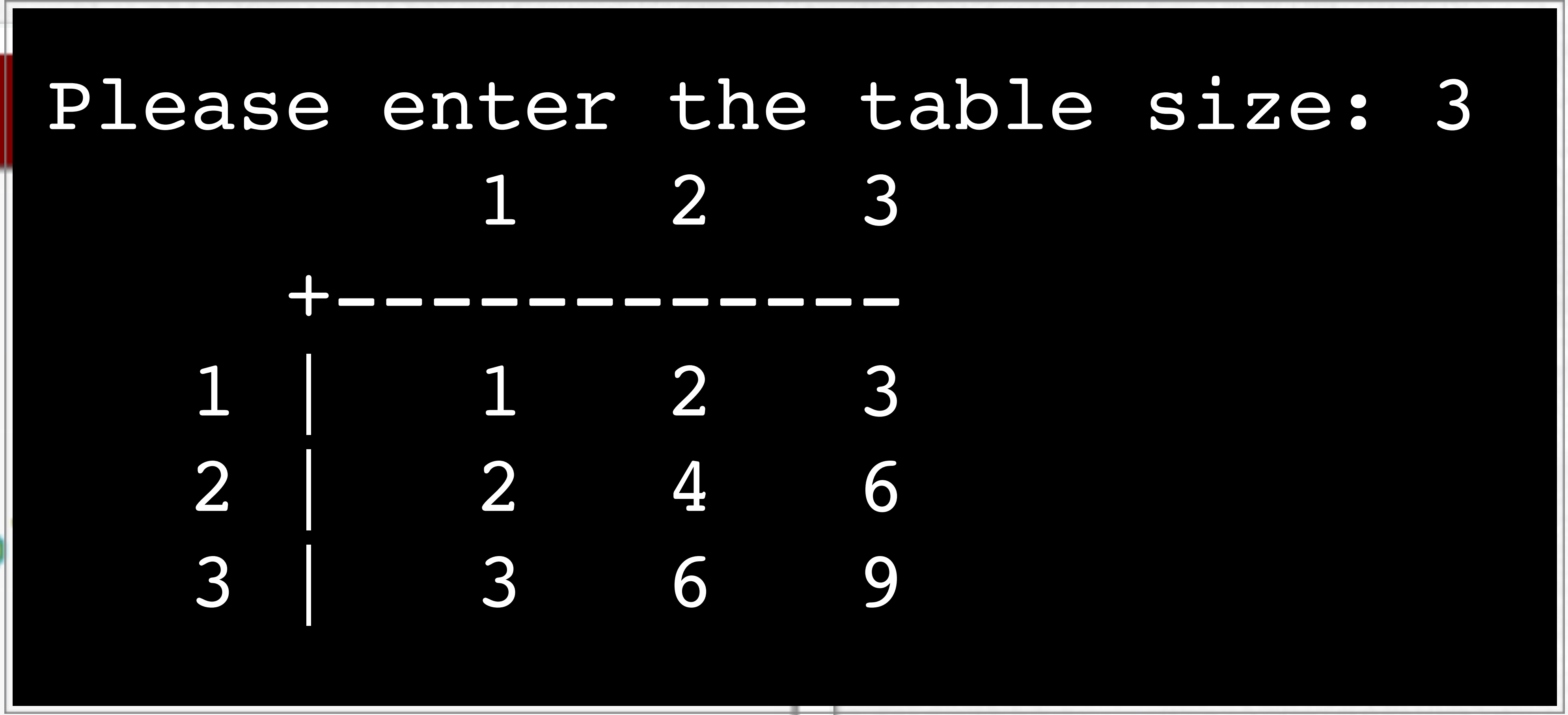
```cpp
#include <iostream>
#include <iomanip>

int main() {
    int size;  // The number of rows and columns in the table
    std::cout << "Please enter the table size: ";
    std::cin >> size;
    // Print a size x size multiplication table

    // First, print heading
    std::cout << "      ";
    for (int column = 1; column <= size; column++)
        std::cout << std::setw(4) << column; // Print heading for this column.
    std::cout << '\n';
    // Print line separator
    std::cout << "     +";
    for (int column = 1; column <= size; column++)
        std::cout << "----";         // Print separator for this column.
    std::cout << '\n';
    // Print table contents
    for (int row = 1; row <= size; row++) {
        std::cout << std::setw(4) << row << " |";  // Print row label.
        for (int column = 1; column <= size; column++)
            std::cout << std::setw(4) << row*column;   // Display product
        std::cout << '\n';                 // Move cursor to next row
    }
}
```

Example using *nested-for* Statement



Listing 7.5: bettertimestable.cpp

```
#include <iostream>
#include <iomanip>

int main() {
    int size;  // The number of rows and columns in the tab
    std::cout << "Please enter the table size: ";
    std::cin >> size;
    // Print a size x size multiplication table

    // First, print heading
    std::cout << "      ";
    for (int column = 1; column <= size; column++)
        std::cout << std::setw(4) << column; // Print heading for this column.
    std::cout << '\n';
    // Print line separator
    std::cout << "     +";
    for (int column = 1; column <= size; column++)
        std::cout << "----";          // Print separator for this column.
    std::cout << '\n';
    // Print table contents
    for (int row = 1; row <= size; row++) {
        std::cout << std::setw(4) << row << " |";  // Print row label.
        for (int column = 1; column <= size; column++)
            std::cout << std::setw(4) << row*column;   // Display product
        std::cout << '\n';                          // Move cursor to next row
    }
}
```



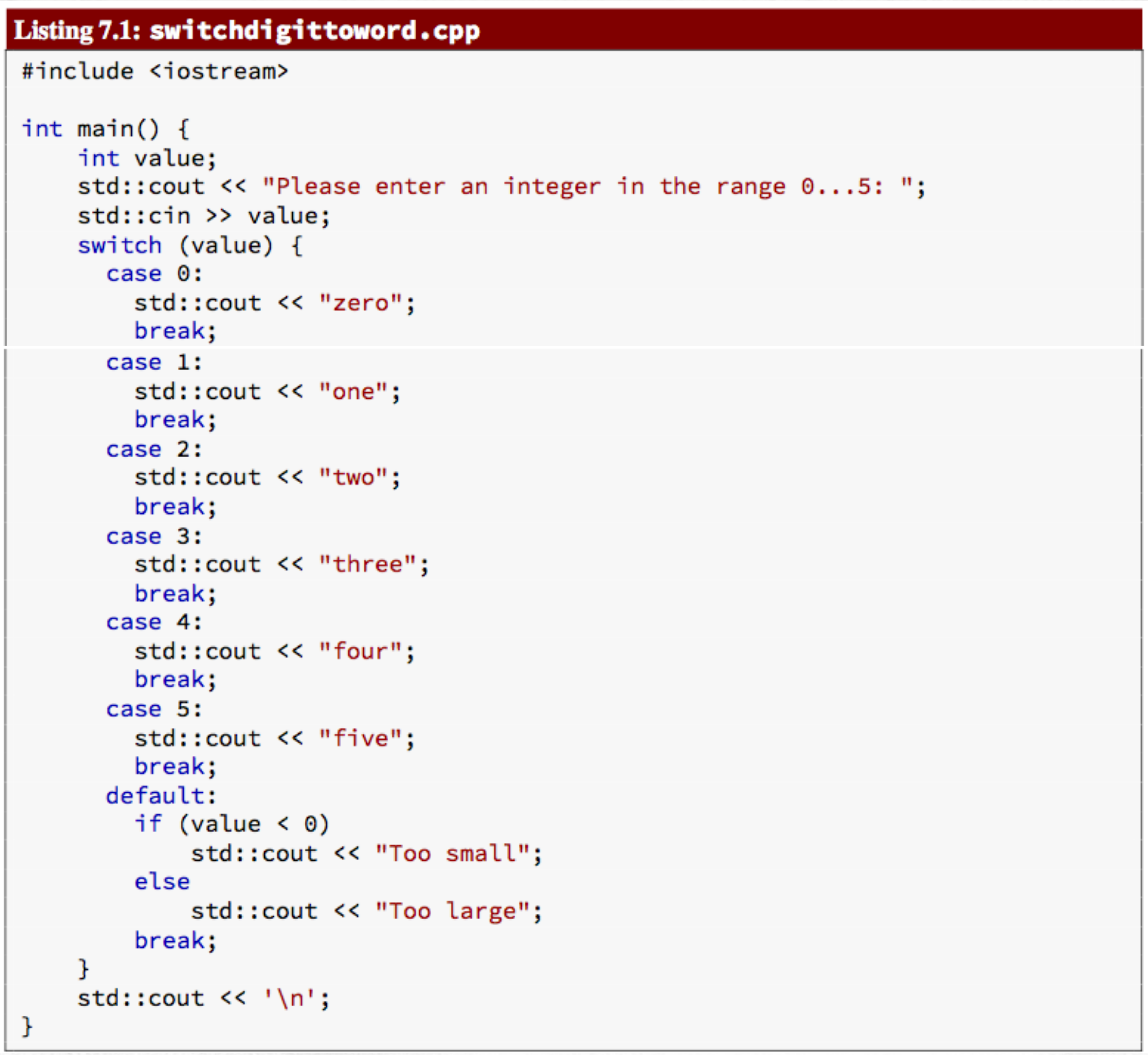
```
Please enter the table size: 3
         1   2   3
     +------------
   1 |   1   2   3
   2 |   2   4   6
   3 |   3   6   9
```

Solution for Nested if-else Statements

```
switch ( integral expression ) {
    case integral constant 1 :
        statement sequence 1
        break;
    case integral constant 2 :
        statement sequence 2
        break;
    case integral constant 3 :
        statement sequence 3
        break;
          ⋮
    case integral constant n :
        statement sequence n
        break;
    default:
        default statement sequence
}
```

```
if ( integral-expression is integral-constant-1 ) then:
    execute statement-sequence-1;
else if ( integral-expression is integral-constant-2 ) then:
    execute statement-sequence-2;
else if ( integral-expression is integral-constant-3 ) then:
    execute statement-sequence-3;
… // skip statements
else if ( integral-expression is integral-constant-n ) then:
    execute statement-sequence-n;
else
    execute default-statement-sequence;
```

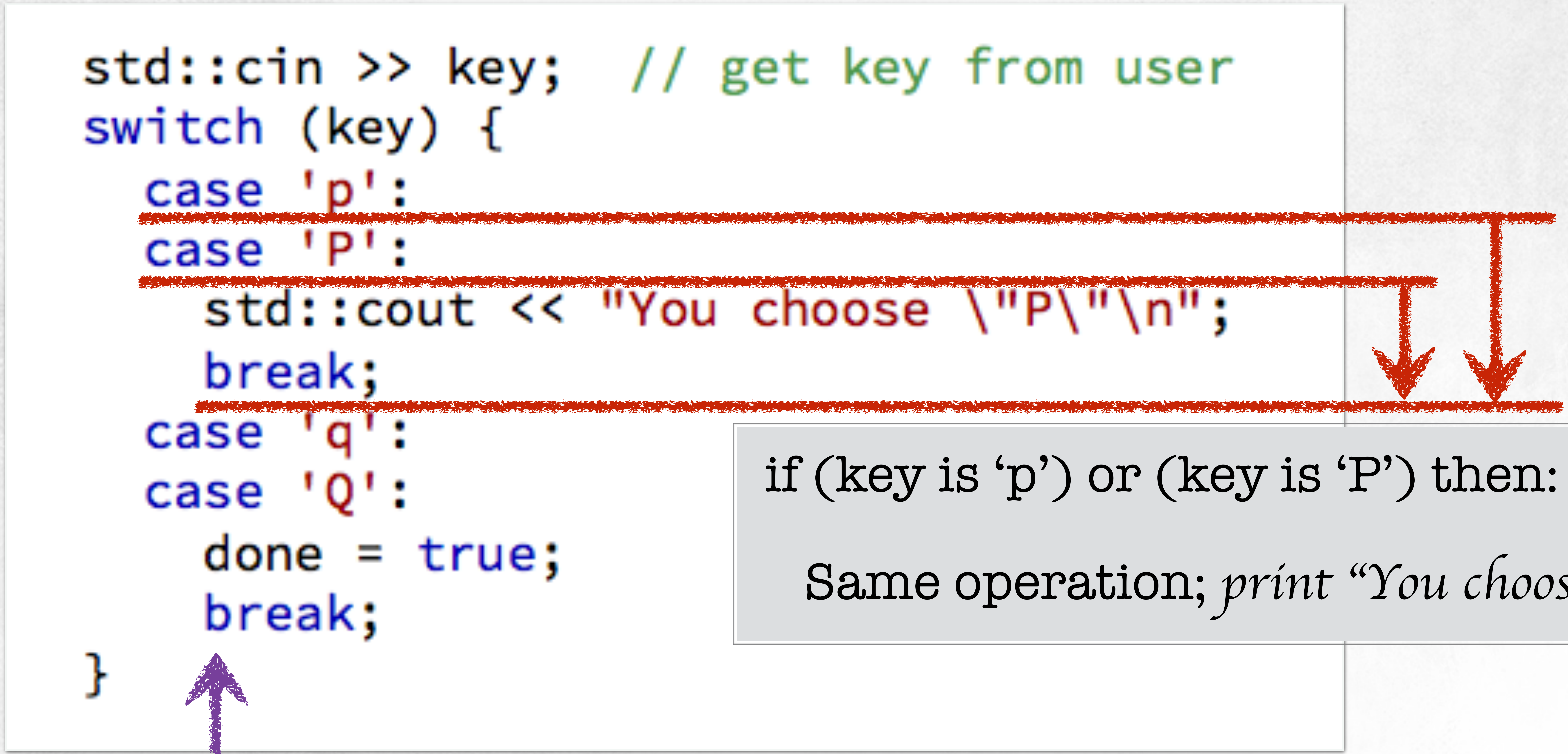
Example for *switch* Statements



Listing 7.1: switchdigittoword.cpp

```
#include <iostream>

int main() {
    int value;
    std::cout << "Please enter an integer in the range 0...5: ";
    std::cin >> value;
    switch (value) {
      case 0:
        std::cout << "zero";
        break;
      case 1:
        std::cout << "one";
        break;
      case 2:
        std::cout << "two";
        break;
      case 3:
        std::cout << "three";
        break;
      case 4:
        std::cout << "four";
        break;
      case 5:
        std::cout << "five";
        break;
      default:
        if (value < 0)
            std::cout << "Too small";
        else
            std::cout << "Too large";
        break;
    }
    std::cout << '\n';
}
```

Role of *break* in *switch* Statements



```
std::cin >> key;  // get key from user
switch (key) {
  case 'p':
  case 'P':
    std::cout << "You choose \"P\"\n";
    break;
  case 'q':
  case 'Q':
    done = true;
    break;
}
```

if (key is ‘p’) or (key is ‘P’) then:

Same operation; *print “You choose \”P”\n”;*

default: is not mandatory

Nested Statements Example

- Code Review: Listing 7.6 in Textbook

Read, Estimate, Execute !!

Object Oriented Programming by C++

Sungwon Lee / Professor

Email: drsungwon@khu.ac.kr

Web: http://mobilelab.khu.ac.kr/

…

…

- …