Chapter Summary

C++ provides a limited number of statements. Most of these affect the flow of control within a program:

- `while`, `for`, and `do while` statements, which provide iterative execution.
- `if` and `switch`, which provide conditional execution.
- `continue`, which stops the current iteration of a loop.
- `break`, which exits a loop or `switch` statement.
- `goto`, which transfers control to a labeled statement.
- `try` and `catch`, which define a `try` block enclosing a sequence of statements that might throw an exception. The `catch` clause(s) are intended to handle the exception(s) that the enclosed code might throw.
- `throw` expression statements, which exit a block of code, transferring control to an associated `catch` clause.
- `return`, which stops execution of a function. (We'll cover `return` statements in Chapter 6.)

In addition, there are expression statements and declaration statements. An expression statement causes the subject expression to be evaluated. Declarations and definitions of variables were described in Chapter 2.

Defined Terms

block Sequence of zero or more statements enclosed in curly braces. A block is a statement, so it can appear anywhere a statement is expected.

break statement Terminates the nearest enclosing loop or `switch` statement. Execution transfers to the first statement following the terminated loop or `switch`.

case label Constant expression (§ 2.4.4, p. 65) that follows the keyword `case` in a `switch` statement. No two `case` labels in the same `switch` statement may have the same value.

catch clause The `catch` keyword, an exception declaration in parentheses, and a block of statements. The code inside a `catch` clause does whatever is necessary to handle an exception of the type defined in its exception declaration.

compound statement Synonym for block.

continue statement Terminates the current iteration of the nearest enclosing loop. Execution transfers to the loop condition in a `while` or `do`, to the next iteration in a range `for`, or to the expression in the header of a traditional `for` loop.

dangling else Colloquial term used to refer to the problem of how to process nested `if` statements in which there are more `if`s than `else`s. In C++, an `else` is always paired with the closest preceding unmatched `if`. Note that curly braces can be used to effectively hide an inner `if` so that the programmer can control which `if` a given `else` should match.

default label `case` label that matches any otherwise unmatched value computed in the `switch` expression.

do while statement Like a `while`, except that the condition is tested at the end of the loop, not the beginning. The statement inside the `do` is executed at least once.

exception classes Set of classes defined by the standard library to be used to represent errors. Table 5.1 (p. 197) lists the general-purpose exception classes.

exception declaration The declaration in a `catch` clause. This declaration specifies the type of exceptions the `catch` can handle.

exception handler Code that deals with an exception raised in another part of the program. Synonym for `catch` clause.

exception safe Term used to describe programs that behave correctly when exceptions are thrown.

expression statement An expression followed by a semicolon. An expression statement causes the expression to be evaluated.

flow of control Execution path through a program.

for statement Iteration statement that provides iterative execution. Ordinarily used to step through a container or to repeat a calculation a given number of times.

goto statement Statement that causes an unconditional transfer of control to a specified labeled statement elsewhere in the same function. `goto`s obfuscate the flow of control within a program and should be avoided.

if else statement Conditional execution of code following the `if` or the `else`, depending on the truth value of the condition.

if statement Conditional execution based on the value of the specified condition. If the condition is `true`, then the `if` body is executed. If not, control flows to the statement following the `if`.

labeled statement Statement preceded by a label. A label is an identifier followed by a colon. Label identifiers are independent of other uses of the same identifier.

null statement An empty statement. Indicated by a single semicolon.

raise Often used as a synonym for throw. C++ programmers speak of "throwing" or "raising" an exception interchangeably.

range for statement Statement that iterates through a sequence.

switch statement A conditional statement that starts by evaluating the expression that follows the `switch` keyword. Control passes to the labeled statement with a `case` label that matches the value of the expression. If there is no matching label, execution either continues at the `default` label, if there is one, or falls out of the `switch` if there is no `default` label.

terminate Library function that is called if an exception is not caught. `terminate` aborts the program.

throw expression Expression that interrupts the current execution path. Each `throw` throws an object and transfers control to the nearest enclosing `catch` clause that can handle the type of exception that is thrown.

try block Block enclosed by the keyword `try` and one or more `catch` clauses. If the code inside a `try` block raises an exception and one of the `catch` clauses matches the type of the exception, then the exception is handled by that `catch`. Otherwise, the exception is handled by an enclosing `try` block or the program terminates.

while statement Iteration statement that executes its target statement as long as a specified condition is `true`. The statement is executed zero or more times, depending on the truth value of the condition.