CONTROL STRUCTURES: PART 2

Chapters 4 & 5

Deitel 10th Edition

TOPICS

Control Structures – A Quick Review

Repetition Overview

Repetition Statements

Repetition Types

Longer Repetition Example

break and continue Statements

Logical Operators

Equality and Assignment Operators

Control Structures – A Quick Review

Control Structure Definition

Block of code defining how a program proceeds through an algorithm

Sometimes called the “flow”

TYPES OF CONTROL STRUCTURES

All programs can be written using only **three control structures**

- **Sequence** structure
- **Selection** structure
- **Repetition** structure

Transfer of control

- The next statement to execute
- Not necessarily the next one in the code…

SEQUENCE CONTROL STRUCTURE

Sequential execution

- One after the other in the order in which they are written.
- Built into C++ (and all structured programming languages)

Unless directed otherwise, statements executed one after another in the order in which they're written

Block

- Series of statements enclosed within curly braces

Any single sequence statement can be replaced with a block

- Syntactically correct, but may not be algorithmically correct

Selection Control Structures

Allow for different paths through a program, depending on the value of a **condition**

Three **selection statements**:

- **if** statement
- **if…else** statement
- **switch** statement

Types of selection

- Single
- Double
- Multiple

Repetition Overview

Repetition Control Structures

Allow statements to be executed repeatedly

Types of repetition

- **Counter-controlled**
- **Sentinel-controlled**

Types of repetition statements (a.k.a. looping statements):

- **while** loop
- **do…while** loop
- **for** loop

Any repetition statement can be used to implement **any repetition type**

REPETITION STATEMENTS

while & for

- Perform the action(s) in their bodies **zero or more** times
- If the loop-continuation condition is initially false, the body will not execute

do…while

- Performs the action(s) in its body **one or more** times

Required Elements of All Repetition

Identification of a **loop control variable**

Initialization of LCV with real data, before it is tested for the first time

Test of LCV to determine if loop should continue

Update of LCV in each iteration of loop

Different loop statements put these steps in a different location, but if any part is missing, loop may not function as expected.

REPETITION STATEMENTS IN C++

Three Type of Repetition Statements

while

do…while

for

WHILE REPETITION STATEMENT

Repeats an action **while a condition remains true**

Body may be a single statement or a block

Loop condition is tested **before** loop starts

If the condition is false at the beginning, the loop will not run at all

WHILE REPETITION STATEMENT – GENERAL FORMAT

```
declaration & initialization of loop control variable
while (loop condition is true)
{
    body statement(s);
    update loop control variable;
}
```

Fig 4.10: ClassAverage.cpp

DO…WHILE REPETITION STATEMENT

Repeats an action **while a condition remains true**

Body may be a single statement or a block

Loop condition is tested **after** loop body executes once

So a do…while loop always executes **at least one time**

Don’t use a do…while when the loop may not have to execute

DO…WHILE REPETITION STATEMENT – GENERAL FORMAT

```
declaration & initialization of loop control variable
do
{
    body statement(s);
    update loop control variable;
} while (loop condition) ;
```

Fig. 5.9: DoWhileTest.cpp

FOR REPETITION STATEMENT

Repeats an action **while a condition remains true**

Specifies repetition details in a single line of code

- Called the for header

Body may be a single statement or a block

Loop condition is tested **before** loop starts

If the condition is false at the beginning, the loop will not run at all

FOR REPETITION STATEMENT – GENERAL FORMAT

```
for (initialization of LCV; test of LCV; update of LCV) {
        body statement(s);
}
```

OR

```
for (declaration and initialization of LCV; test of LCV; update of LCV) {
        body statement(s);
}
```

Fig. 5.2: ForCounter.cpp

All Three Parts of For Header are Optional

Omit **initialization** expression if control variable initialized before the loop.

If loop **condition test** omitted, the condition is always true, thus creating an infinite loop.

Omit **update** expression if handled in loop body or if no update needed.

Comma-Separated Lists of Expressions

Allowed within a **for** statement's header

Uses the comma operator

Used for multiple initializations or multiple increments

Fig. 5.5: Sum.cpp / Fig. 5.5: fig05_05_altsyntax.cpp

Comma-Separated Lists of Expressions – Comma Operators

Comma operators

- Lowest precedence
- Guarantee that lists of operations are evaluated left to right

Value and type of a comma-separated list of expressions is the value and type of the rightmost expression

Good rule of thumb:

- Use only expressions that involve the control variable in the initialize or increment sections

EXTRA SEMICOLON

What happens if you put a semicolon immediately to the right of the right parenthesis of a **for** header?

```
for ( int year = 1; year <= 10; year++ ) ;
  {
     //loop body
  }
```

Repetition Types

Two Types of Repetition

Counter-controlled

Sentinel-controlled

COUNTER-CONTROLLED REPETITION

Loop will execute based on the value of a **counter** variable

- So the loop control variable is a counter

Also called **definite repetition**

- Number of repetitions is known before the loop begins executing

Always use integers for counter variables

- Why?

Can be implemented using any of the three repetition statements

- Common to use for
- Not common to use do…while

COUNTER-CONTROLLED REPETITION – LOGIC

Set counter to initial value before loop starts

If the condition of the test is false, the counter is more than the final value, so the body of the for does not execute.

If the condition is true, the loop body executes.

After the body executes, control returns to the header, where the counter is updated, and the test is checked again.

Fig. 5.1: WhileCounter.cpp

UNSIGNED INT DATA TYPE

Note the counter variable's data type (line 8)

Type	Size	Distinct Values	Min Value	Max Value
unsigned integer	32 bits	+4,294,967,295	0	+4,294,967,295
signed integer	32 bits	+4,294,967,295	-2,147,483,648	+2,147,483,647

Different schools of thought:

- Use unsigned whenever you have a value that can only be 0 or positive (e.g. counters)
- int is fine unless you have a good reason to use something else

Counter-Controlled Repetition – Off by One Errors

If LCV starts at:	Use relational operator:	Compare LCV to	Example
0	<	Its final value	x < 10
1	<=	Its final value	x <= 10

SENTINEL-CONTROLLED REPETITION

Use sentinel-controlled repetition when we don't know how many times the loop will have to be executed

Sentinel value used to indicate "end of data entry"

SENTINEL-CONTROLLED REPETITION – LOGIC

Read the first value before reaching the loop.

The value read should determine whether the loop is entered or not.

If the condition of the while is false, the first item of data was the sentinel value, so the body of the loop does not execute.

If the condition is true, the loop body executes.

The last step of the loop body is to get the next item of data.

Control returns to the test.

Fig. 4.12: ClassAverage.cpp

When to Use Each Type of Statement?

Typically:

while statements – **sentinel-controlled** repetition

do…while statements – **sentinel-controlled** repetition in response to a **menu**

for statements – **counter-controlled** repetition

But any type of statement can be used to implement any type of repetition

LONGER REPETITION EXAMPLE

COMPOUND INTEREST EXAMPLE

$1000 is invested in a savings account yielding 5% interest. Calculate balance at the end of each year for 10 years. Use the following formula:

$$a = p\,(1 + r)^{n}$$

Where:

p = the principal
r = annual interest rate as a decimal (e.g., use 0.05 for 5%)
n = years (exponent)
a = balance at end of year n

Fig. 5.6: Interest.cpp

STANDARD LIBRARY FUNCTION POW

Instead of an exponent operator, use **pow** function to raise a value to a power.

Need header **<cmath>**

pow(x, y) calculates the value of x raised to the y^{th} power

- Takes two doubles and returns a double.

Line 27: pow(1.0 + rate, year)

- What's inefficient about this statement?

Using Float or Double for Monetary Amounts

When dealing with monetary values, we usually only display 2 places after the decimal

But what is displayed isn't always the actual stored value

PrintingMoney.cpp

EOF

Platform-dependent keystroke that indicates end of data

- Evaluates as -1
- In <iostream>

Using EOF makes program more portable

Typed as <Ctrl> z or <Ctrl> d

while (cin >> grade) is true if user types an int, false otherwise

Fig. 5.11: LetterGrades.cpp

BREAK AND CONTINUE STATEMENTS

BREAK

Within a loop or switch **immediately exits the loop or switch**

Fig. 5.13: BreakTest.cpp

CONTINUE

Skips remaining statements in a loop body and proceeds with next iteration

In while or do…while, the condition is checked

In for, the control variable is incremented

Fig. 5.14: ContinueTest.cpp

LOGICAL OPERATORS

Combine Simple Conditions into Compound Conditions

Operator	Meaning	Compound expression is true when	Compound expression is false when
&&	conditional AND	Both simple conditions are true	Either simple condition is false
||	conditional OR	Either simple condition is true	Both simple conditions are false
!	Logical not	Simple condition is false	Simple condition is true

LOGICAL OPERATORS – ORDER OF CONDITIONS

When using &&, which condition should be in the left position – the one most likely to be true or false?

When using ||, which condition should be in the left position – the one most likely to be true or false?

LOGICAL OPERATORS EXAMPLE

Fig. 5.18: LogicalOperators.cpp

Produces truth tables for the operators discussed in this section

boolalpha stream manipulator

- Prints bool value as the word true or false (sticky)
- Can reset this behavior using noboolalpha

Equality and Assignment Operators

CONFUSING THE EQUALITY AND ASSIGNMENTS OPERATORS

= is not the same as **==**

Useful technique:

- Instead of writing x == 7, write 7 == x
- Get in the habit of writing equality expressions with the **constant on the left and variable on the right**
- If you accidentally write it with a single =, it will be a compile error

Equality.cpp

Structured Programming Summary

PROMOTES SIMPLICITY

Only three forms of control are needed to implement an algorithm (Bohm and Jacopini):

- Sequence
- Selection
- Repetition

Sequence structure is trivial.

- List statements to execute in the order in which they should execute.

IMPLEMENTING SELECTION

Selection is implemented in one of three ways:

- if statement (single selection)
- if…else statement (double selection)
- switch statement (multiple selection)

Simple if statement is really all you need

- If…else statement and switch statement logic can be implemented by combining if statements instead.

Implementing Repetition

Repetition is implemented in one of three ways:

- while statement
- do…while statement
- for statement

while statement is really all you need

- Everything that can be done with do…while and for can be done with the while.

COMBINING THEM ALL

Any form of control ever needed in a Java program can be expressed in terms of

- sequence
- if statement (selection)
- while statement (repetition)

and these can be combined in only two ways

- stacking
- nesting