Chapter 2 - Control Structures

Outline
2.1 Introduction
2.2 Algorithms
2.3 Pseudocode
2.4 Control Structures
2.5 if Selection Structure
2.6 if/else Selection Structure
2.7 while Repetition Structure
2.8 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)
2.9 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)
2.10 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 3 (Nested Control Structures)
2.11 Assignment Operators
2.12 Increment and Decrement Operators
2.13 Essentials of Counter-Controlled Repetition
2.14 for Repetition Structure
2.15 Examples Using the for Structure
Chapter 2 - Control Structures

Outline

2.16 switch Multiple-Selection Structure
2.17 do/while Repetition Structure
2.18 break and continue Statements
2.19 Logical Operators
2.20 Confusing Equality (==) and Assignment (=) Operators
2.21 Structured-Programming Summary
2.1 Introduction

• Before writing a program
  – Have a thorough understanding of problem
  – Carefully plan your approach for solving it

• While writing a program
  – Know what “building blocks” are available
  – Use good programming principles
2.2 Algorithms

• Computing problems
  – Solved by executing a series of actions in a specific order

• Algorithm a procedure determining
  – Actions to be executed
  – Order to be executed
  – Example: recipe

• Program control
  – Specifies the order in which statements are executed
2.3 Pseudocode

• Pseudocode
  – Artificial, informal language used to develop algorithms
  – Similar to everyday English

• Not executed on computers
  – Used to think out program before coding
    • Easy to convert into C++ program
  – Only executable statements
    • No need to declare variables
2.4 Control Structures

• Sequential execution
  – Statements executed in order

• Transfer of control
  – Next statement executed *not* next one in sequence

• 3 control structures (Bohm and Jacopini)
  – Sequence structure
    • Programs executed sequentially by default
  – Selection structures
    • *if, if/else, switch*
  – Repetition structures
    • *while, do/while, for*
# 2.4 Control Structures

- **C++ keywords**
  - Cannot be used as identifiers or variable names

<table>
<thead>
<tr>
<th>C++ Keywords</th>
<th>Keywords common to the C and C++ programming languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>break, case, char, const</td>
</tr>
<tr>
<td>continue</td>
<td>default, do, double, else</td>
</tr>
<tr>
<td>enum</td>
<td>extern, float, for, goto</td>
</tr>
<tr>
<td>if</td>
<td>int, long, register, return</td>
</tr>
<tr>
<td>short</td>
<td>signed, sizeof, static, struct</td>
</tr>
<tr>
<td>switch</td>
<td>typedef, union, unsigned, void</td>
</tr>
<tr>
<td>volatile</td>
<td>while</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><em>C++ only keywords</em></td>
<td></td>
</tr>
<tr>
<td>asm</td>
<td>bool, catch, class, const_cast</td>
</tr>
<tr>
<td>delete</td>
<td>dynamic_cast, explicit, false</td>
</tr>
<tr>
<td>inline</td>
<td>mutable, namespace, new, operator</td>
</tr>
<tr>
<td>private</td>
<td>protected, public, reinterpret_cast</td>
</tr>
<tr>
<td>static_cast</td>
<td>template, this, throw, true</td>
</tr>
<tr>
<td>try</td>
<td>typeid, typename, using, virtual</td>
</tr>
<tr>
<td>wchar_t</td>
<td></td>
</tr>
</tbody>
</table>
2.4 Control Structures

• Flowchart
  – Graphical representation of an algorithm
  – Special-purpose symbols connected by arrows (flowlines)
  – Rectangle symbol (action symbol)
    • Any type of action
  – Oval symbol
    • Beginning or end of a program, or a section of code (circles)

• Single-entry/single-exit control structures
  – Connect exit point of one to entry point of the next
  – Control structure stacking
2.5 if Selection Structure

• Selection structure
  – Choose among alternative courses of action
  – Pseudocode example:
    
    *If student’s grade is greater than or equal to 60*
    
    *Print “Passed”*
  – If the condition is true
    *Print statement executed, program continues to next statement*
  – If the condition is false
    *Print statement ignored, program continues*
  – Indenting makes programs easier to read
    *C++ ignores whitespace characters (tabs, spaces, etc.)*
2.5 if Selection Structure

• Translation into C++

*If student’s grade is greater than or equal to 60*

*Print “Passed”*

```
if ( grade >= 60 )
    cout << "Passed";
```

• Diamond symbol (decision symbol)
  – Indicates decision is to be made
  – Contains an expression that can be true or false
    • Test condition, follow path

• **if** structure
  – Single-entry/single-exit
2.5 if Selection Structure

- Flowchart of pseudocode statement

```
grade >= 60
true
print "Passed"
false
```

A decision can be made on any expression.
zero - false
nonzero - true

Example:
3 - 4 is true
2.6 *if*/else Selection Structure

• *if*
  – Performs action if condition true

• *if*/else
  – Different actions if conditions true or false

• Pseudocode
  
  ```
  if student’s grade is greater than or equal to 60
    print “Passed”
  else
    print “Failed”
  ```

• C++ code
  
  ```
  if ( grade >= 60 )
    cout << "Passed";
  else
    cout << "Failed";
  ```
2.6 if/else Selection Structure

- Ternary conditional operator (? : )
  - Three arguments (condition, value if `true`, value if `false`)

- Code could be written:
  ```cpp
  cout << ( grade >= 60 ? "Passed" : "Failed" );
  ```

```
true

false

print "Failed"

print "Passed"

grade >= 60

Condition Value if true Value if false

false

true
```
2.6 if/else Selection Structure

- Nested if/else structures
  - One inside another, test for multiple cases
  - Once condition met, other statements skipped

```plaintext
if student’s grade is greater than or equal to 90
    Print “A”
else
    if student’s grade is greater than or equal to 80
        Print “B”
    else
        if student’s grade is greater than or equal to 70
            Print “C”
        else
            if student’s grade is greater than or equal to 60
                Print “D”
            else
                Print “F”
```
2.6 if/else Selection Structure

• Example

```cpp
if ( grade >= 90 )       // 90 and above
    cout << "A";
else if ( grade >= 80 )  // 80-89
    cout << "B";
else if ( grade >= 70 )  // 70-79
    cout << "C";
else if ( grade >= 60 )  // 60-69
    cout << "D";
else                     // less than 60
    cout << "F";
```
2.6 if/else Selection Structure

• Compound statement
  – Set of statements within a pair of braces
    
    ```cpp
    if ( grade >= 60 )
        cout << "Passed.\n";
    else {
        cout << "Failed.\n";
        cout << "You must take this course again.\n";
    }
    
    – Without braces,
    cout << "You must take this course again.\n";
    always executed
  ```

• Block
  – Set of statements within braces
2.7 while Repetition Structure

• Repetition structure
  – Action repeated while some condition remains true
  – Pseudocode
    
    ```
    while there are more items on my shopping list
      Purchase next item and cross it off my list
    ```
  – while loop repeated until condition becomes false

• Example

```c
int product = 2;
while ( product <= 1000 )
  product = 2 * product;
```
2.7 The while Repetition Structure

- Flowchart of `while` loop

```
while product <= 1000
    product = 2 * product
```

2.8 Formulating Algorithms (Counter-Controlled Repetition)

- Counter-controlled repetition
  - Loop repeated until counter reaches certain value

- Definite repetition
  - Number of repetitions known

- Example

  A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.
2.8 Formulating Algorithms (Counter-Controlled Repetition)

• Pseudocode for example:

  Set total to zero
  Set grade counter to one
  While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
    Add one to the grade counter
  Set the class average to the total divided by ten
  Print the class average

• Next: C++ code for this example
// Fig. 2.7: fig02_07.cpp
// Class average program with counter-controlled repetition.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

// function main begins program execution
int main()
{
    int total;       // sum of grades input by user
    int gradeCounter; // number of grade to be entered next
    int grade;       // grade value
    int average;     // average of grades

    // initialization phase
    total = 0;       // initialize total
    gradeCounter = 1; // initialize loop counter
// processing phase
while ( gradeCounter <= 10 ) { // loop 10 times
cout << "Enter grade: "; // prompt for input
cin >> grade; // read grade from user
total = total + grade; // add grade to total
gradeCounter = gradeCounter + 1; // increment counter
}

// termination phase
average = total / 10; // integer division

// display result
cout << "Class average is " << average << endl;

return 0; // indicate program ended successfully

Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81

The counter gets incremented each time the loop executes. Eventually, the counter causes the loop to end.
2.9 Formulating Algorithms (Sentinel-Controlled Repetition)

• Suppose problem becomes:
  
  _Develop a class-averaging program that will process an arbitrary number of grades each time the program is run_
  
  – Unknown number of students
  – How will program know when to end?

• Sentinel value
  
  – Indicates “end of data entry”
  – Loop ends when sentinel input
  – Sentinel chosen so it cannot be confused with regular input
    • -1 in this case
2.9 Formulating Algorithms (Sentinel-Controlled Repetition)

- Top-down, stepwise refinement
  - Begin with pseudocode representation of top
    *Determine the class average for the quiz*
  - Divide top into smaller tasks, list in order
    *Initialize variables*
    *Input, sum and count the quiz grades*
    *Calculate and print the class average*
2.9 Formulating Algorithms (Sentinel-Controlled Repetition)

• Many programs have three phases
  – Initialization
    • Initializes the program variables
  – Processing
    • Input data, adjusts program variables
  – Termination
    • Calculate and print the final results
  – Helps break up programs for top-down refinement
2.9 Formulating Algorithms (Sentinel-Controlled Repetition)

• Refine the initialization phase

  *Initialize variables*

  goes to

  *Initialize total to zero*

  *Initialize counter to zero*

• Processing

  *Input, sum and count the quiz grades*

  goes to

  *Input the first grade (possibly the sentinel)*

  *While the user has not as yet entered the sentinel*

  *Add this grade into the running total*

  *Add one to the grade counter*

  *Input the next grade (possibly the sentinel)*
2.9 Formulating Algorithms (Sentinel-Controlled Repetition)

• Termination

  *Calculate and print the class average*

  *goes to*

  *If the counter is not equal to zero*

  *Set the average to the total divided by the counter*

  *Print the average*

  *Else*

  *Print “No grades were entered”*

• Next: C++ program
// Class average program with sentinel-controlled repetition.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;
using std::fixed;

#include <iomanip> // parameterized stream manipulators
using std::setprecision; // sets numeric output precision

// function main begins program execution
int main()
{
    int total; // sum of grades
    int gradeCounter; // number of grades entered
    int grade; // grade value

    double average; // number with decimal point for average

    // initialization phase
    total = 0; // initialize total
    gradeCounter = 0; // initialize loop counter

    // Data type double used to represent decimal numbers.
// processing phase
// get first grade from user
cout << "Enter grade, -1 to end: "; // prompt for input
cin >> grade; // read grade from user

// loop until sentinel value read from user
while ( grade != -1 ) {
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter
    cout << "Enter grade, -1 to end: "; // prompt for input
    cin >> grade; // read next grade
}
// end while

// termination phase
// if user entered at least one grade ...
if ( gradeCounter != 0 ) {
    // calculate average of all grades entered
    average = static_cast< double >( total ) / gradeCounter;
}

// static_cast<double>() treats total as a double temporarily (casting).
// Required because dividing two integers truncates the remainder.

// gradeCounter is an int, but it gets promoted to double.
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50

// display average with two digits of precision
cout << "Class average is " << setprecision( 2 )
    << fixed << average << endl;

} // end if part of if/else

else // if no grades were entered, output appropriate message
    cout << "No grades were entered" << endl;

return 0; // indicate program ended successfully

} // end function main

fixed forces output to print in fixed point format (not scientific notation). Also, forces trailing zeros and decimal point to print.

Include <iomanip>

Include <iostream>
2.10 Nested Control Structures

• Problem statement

A college has a list of test results (1 = pass, 2 = fail) for 10 students. Write a program that analyzes the results. If more than 8 students pass, print "Raise Tuition".

• Notice that
  – Program processes 10 results
    • Fixed number, use counter-controlled loop
  – Two counters can be used
    • One counts number that passed
    • Another counts number that fail
  – Each test result is 1 or 2
    • If not 1, assume 2
2.10 Nested Control Structures

• Top level outline

  Analyze exam results and decide if tuition should be raised

• First refinement

  Initialize variables
  Input the ten quiz grades and count passes and failures
  Print a summary of the exam results and decide if tuition should be raised

• Refine

  Initialize variables
to
  Initialize passes to zero
  Initialize failures to zero
  Initialize student counter to one
2.10 Nested Control Structures

• Refine

Input the ten quiz grades and count passes and failures to

While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
    Add one to passes
  Else
    Add one to failures
  Add one to student counter
2.10 Nested Control Structures

• Refine

  *Print a summary of the exam results and decide if tuition should be raised*

  to

  *Print the number of passes*
  *Print the number of failures*
  *If more than eight students passed Print “Raise tuition”*

• Program next
// Fig. 2.11: fig02_11.cpp
// Analysis of examination results.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

// function main begins program execution
int main()
{
    // initialize variables in declarations
    int passes = 0;   // number of passes
    int failures = 0; // number of failures
    int studentCounter = 1;  // student counter
    int result; // one exam result

    // process 10 students using counter-controlled loop
    while (studentCounter <= 10) {
        // prompt user for input and obtain value from user
        cout << "Enter result (1 = pass, 2 = fail): ";
        cin >> result;
// if result 1, increment passes; if/else nested in while
if ( result == 1 ) // if/else nested in while
    passes = passes + 1;

else // if result not 1, increment failures
    failures = failures + 1;

// increment studentCounter so loop eventually terminates
studentCounter = studentCounter + 1;

} // end while

// termination phase; display number of passes and failures
cout << "Passed " << passes << endl;
cout << "Failed " << failures << endl;

// if more than eight students passed, print "raise tuition"
if ( passes > 8 )
    cout << "Raise tuition " << endl;

return 0; // successful termination

} // end function main
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed 6
Failed 4

Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed 9
Failed 1

Raise tuition
2.11 Assignment Operators

- Assignment expression abbreviations
  - Addition assignment operator
    
    \[ c = c + 3; \] abbreviated to
    
    \[ c += 3; \]

- Statements of the form
  
  \[ \text{variable} = \text{variable operator expression}; \]

  can be rewritten as

  \[ \text{variable operator= expression;} \]

- Other assignment operators
  
  \[ \begin{align*}
    d &= 4 & (d &= d - 4) \\
    e &= 5 & (e &= e * 5) \\
    f &= 3 & (f &= f / 3) \\
    g &= 9 & (g &= g \% 9)
  \end{align*} \]
2.12 Increment and Decrement Operators

- Increment operator (++) - can be used instead of c += 1
- Decrement operator (--) - can be used instead of c -= 1
  - Preincrement
    - When the operator is used before the variable (++c or --c)
    - Variable is changed, then the expression it is in is evaluated.
  - Posincrement
    - When the operator is used after the variable (c++ or c--)
    - Expression the variable is in executes, then the variable is changed.
2.12 Increment and Decrement Operators

• Increment operator (++)
  – Increment variable by one
  – c++
    • Same as c += 1

• Decrement operator (--) similar
  – Decrement variable by one
  – c--
2.12 Increment and Decrement Operators

- **Preincrement**
  - Variable changed before used in expression
    - Operator before variable \((++c\text{ or } --c)\)

- **Postincrement**
  - Incremented changed after expression
    - Operator after variable \((c++, c--)\)
2.12 Increment and Decrement Operators

- **If c = 5, then**
  - `cout << ++c;`
    - c is changed to 6, then printed out
  - `cout << c++;`
    - Prints out 5 (*cout* is executed before the increment.)
    - c then becomes 6
2.12 Increment and Decrement Operators

• When variable not in expression
  – Preincrementing and postincrementing have same effect
    ```
    ++c;
    cout << c;
    ```
    and
    ```
    c++;  
    cout << c;
    ```
    are the same
// Fig. 2.14: fig02_14.cpp
// Preincrementing and postincrementing.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
    int c; // declare variable

    // demonstrate postincrement
    c = 5; // assign 5 to c
    cout << c << endl; // print 5
    cout << c++ << endl; // print 5 then postincrement
    cout << c << endl << endl; // print 6

    // demonstrate preincrement
    c = 5; // assign 5 to c
    cout << c << endl; // print 5
    cout << ++c << endl; // preincrement then print 6
    cout << c << endl; // print 6
return 0; // indicate successful termination

} // end function main
2.13 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - Name of control variable/loop counter
  - Initial value of control variable
  - Condition to test for final value
  - Increment/decrement to modify control variable when looping
// Fig. 2.16: fig02_16.cpp
// Counter-controlled repetition.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
    int counter = 1; // initialization

    while ( counter <= 10 ) { // repetition condition
        cout << counter << endl; // display counter
        ++counter; // increment
    } // end while

    return 0; // indicate successful termination
} // end function main
2.13 Essentials of Counter-Controlled Repetition

• The declaration

```c
int counter = 1;
```

– Names `counter`
– Declares `counter` to be an integer
– Reserves space for `counter` in memory
– Sets `counter` to an initial value of 1
2.14 for Repetition Structure

• General format when using **for** loops

```cpp
for ( initialization; LoopContinuationTest; increment )
    statement
```

• Example

```cpp
for( int counter = 1; counter <= 10; counter++ )
    cout << counter << endl;
```

− Prints integers from one to ten

No semicolon after last statement
// Fig. 2.17: fig02_17.cpp
// Counter-controlled repetition with the for structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
    // Initialization, repetition condition and incrementing
    // are all included in the for structure header.
    for ( int counter = 1; counter <= 10; counter++ )
        cout << counter << endl;

    return 0;  // indicate successful termination
}

// end function main
2.14 for Repetition Structure

• **for** loops can usually be rewritten as **while** loops

```
initialization;
while ( loopContinuationTest){
    statement
    increment;
}
```

• Initialization and increment
  
  – For multiple variables, use comma-separated lists
  
  ```
  for (int i = 0, j = 0; j + i <= 10; j++, i++)
      cout << j + i << endl;
  ```
// Fig. 2.20: fig02_20.cpp
// Summation with for.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
    int sum = 0; // initialize sum
    // sum even integers from 2 through 100
    for (int number = 2; number <= 100; number += 2)
    {
        sum += number; // add number to sum
    }
    cout << "Sum is " << sum << endl; // output sum
    return 0; // successful termination
}
// end function main

Sum is 2550
2.15 Examples Using the for Structure

• Program to calculate compound interest

• A person invests $1000.00 in a savings account yielding 5 percent interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts:

\[ a = p(1+r)^n \]

• \( p \) is the original amount invested (i.e., the principal),

\( r \) is the annual interest rate,

\( n \) is the number of years and

\( a \) is the amount on deposit at the end of the \( n \)th year
// Fig. 2.21: fig02_21.cpp
// Calculating compound interest.
#include <iostream>

using namespace std;

# include <iomanip>

// enables program to use function pow

int main()
{
    double amount; // amount on deposit
    double principal = 1000.0; // starting principal
    double rate = .05; // interest rate
}
// output table column heads
cout << "Year" << setw(21) << "Amount on deposit" << endl;

// set floating-point number format
cout << fixed << setprecision(2);

// calculate amount on deposit for each of ten years
for (int year = 1; year <= 10; year++) {
    // calculate new amount for specified year
    amount = principal * pow(1.0 + rate, year);

    // output one table row
    cout << setw(4) << year
         << setw(21) << amount << endl;
}

return 0;  // indicate successful termination
} // end function main

pow(x, y) = x raised to the yth power.
Sets the field width to at least 21 characters. If output less than 21, it is right-justified.
<table>
<thead>
<tr>
<th>Year</th>
<th>Amount on deposit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1050.00</td>
</tr>
<tr>
<td>2</td>
<td>1102.50</td>
</tr>
<tr>
<td>3</td>
<td>1157.63</td>
</tr>
<tr>
<td>4</td>
<td>1215.51</td>
</tr>
<tr>
<td>5</td>
<td>1276.28</td>
</tr>
<tr>
<td>6</td>
<td>1340.10</td>
</tr>
<tr>
<td>7</td>
<td>1407.10</td>
</tr>
<tr>
<td>8</td>
<td>1477.46</td>
</tr>
<tr>
<td>9</td>
<td>1551.33</td>
</tr>
<tr>
<td>10</td>
<td>1628.89</td>
</tr>
</tbody>
</table>

Numbers are right-justified due to `setw` statements (at positions 4 and 21).
2.16 switch Multiple-Selection Structure

- **switch**
  - Test variable for multiple values
  - Series of *case* labels and optional *default* case

```java
switch (variable) {
    case value1:        // taken if variable == value1
        statements
        break;            // necessary to exit switch

    case value2:

    case value3:        // taken if variable == value2 or == value3
        statements
        break;

    default:            // taken if variable matches no other cases
        statements
        break;
}
```
2.16 switch Multiple-Selection Structure

case a
  true
  case a action(s) → break
  false
  case a

  false

  case b
  true
  case b action(s) → break
  false

  .

  .

  case z
  true
  case z action(s) → break
  false

default action(s)
2.16 switch Multiple-Selection Structure

• Example upcoming
  – Program to read grades (A-F)
  – Display number of each grade entered

• Details about characters
  – Single characters typically stored in a char data type
    • char a 1-byte integer, so chars can be stored as ints
  – Can treat character as int or char
    • 97 is the numerical representation of lowercase ‘a’ (ASCII)
    • Use single quotes to get numerical representation of character
      cout << "The character (" << 'a' << ") has the value " << static_cast< int > ( 'a' ) << endl;

Prints

The character (a) has the value 97
1      // Fig. 2.22: fig02_22.cpp
2      // Counting letter grades.
3      #include <iostream>
4
5      using std::cout;
6      using std::cin;
7      using std::endl;
8
9      // function main begins program execution
10     int main()
11     {
12         int grade;     // one grade
13         int aCount = 0;  // number of As
14         int bCount = 0;  // number of Bs
15         int cCount = 0;  // number of Cs
16         int dCount = 0;  // number of Ds
17         int fCount = 0;  // number of Fs
18
19         cout << "Enter the letter grades.\n" << endl
20             << "Enter the EOF character to end input.\n" << endl;
// loop until user types end-of-file key sequence
while (( grade = cin.get() ) != EOF ) {

    // determine which grade was input
switch ( grade ) { // switch structure

        case 'A':  // grade was uppercase A
        case 'a':  // or lowercase a
            ++aCount;  // increment aCount
            break;  // exit switch

        case 'B':  // grade was uppercase B
        case 'b':  // or lowercase b
            ++bCount;  // increment bCount
            break;  // exit switch

        case 'C':  // grade was uppercase C
        case 'c':  // or lowercase c
            ++cCount;  // increment cCount
            break;  // exit switch

    // cin.get() uses dot notation (explained chapter 6). This
    // function gets 1 character from the keyboard (after Enter pressed), and
    // it is assigned to grade.
    cin.get() returns EOF (end-of-file) after the EOF character is
    input, to indicate the end of data. EOF may be ctrl-d or ctrl-z,
    depending on your OS.

    Assignment statements have a value, which is the same as
    the variable on the left of the =. The value of this statement
    is the same as the value returned by cin.get().

    This can also be used to initialize multiple variables:
a = b = c = 0;
case 'D': // grade was uppercase D
  ++dCount; // increment dCount
  break; // exit switch

case 'd': // or lowercase d
  ++dCount; // increment dCount
  break; // exit switch

case 'F': // grade was uppercase F
  ++fCount; // increment fCount
  break; // exit switch

case 'f': // or lowercase f
  ++fCount; // increment fCount
  break; // exit switch

case '\n': // ignore newline,
  case '\t': // tabs,
  case ' ': // and spaces in input
    break; // exit switch

default: // catch all other characters
  cout << "Incorrect letter grade entered."
    << " Enter a new grade." << endl;
  break; // optional; will exit switch anyway

} // end switch

} // end while
cout << "\n\nTotals for each letter grade are:"
    << "\nA: " << aCount  // display number of A grades
    << "\nB: " << bCount  // display number of B grades
    << "\nC: " << cCount  // display number of C grades
    << "\nD: " << dCount  // display number of D grades
    << "\nF: " << fCount  // display number of F grades
    << endl;

return 0; // indicate successful termination
} // end function main
Enter the letter grades.
Enter the EOF character to end input.

a
B
c
C
A
d
f
C
E

Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
2.17 do/while Repetition Structure

• Similar to **while** structure
  – Makes loop continuation test at end, not beginning
  – Loop body executes at least once

• Format

```
do {
    statement
} while ( condition );
```
// Fig. 2.24: fig02_24.cpp
// Using the do/while repetition structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
    int counter = 1;  // initialize counter

    do {
        cout << counter << " ";  // display counter
    } while ( ++counter <= 10 );  // end do/while

    cout << endl;
    return 0;  // indicate successful termination
} // end function main

Notice the preincrement in loop-continuation test.
2.18 break and continue Statements

• **break** statement
  – Immediate exit from `while`, `for`, `do/while`, `switch`
  – Program continues with first statement after structure

• **Common uses**
  – Escape early from a loop
  – Skip the remainder of `switch`
// Fig. 2.26: fig02_26.cpp
// Using the break statement in a for structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{

    int x; // x declared here so it can be used after the loop

    // loop 10 times
    for ( x = 1; x <= 10; x++ ) {

        // if x is 5, terminate loop
        if ( x == 5 )
            break; // break loop only if x is 5

        cout << x << " "; // display value of x
    }

    cout << "\nBroke out of loop when x became " << x << endl;
return 0;  // indicate successful termination

} // end function main

Broke out of loop when x became 5
2.18 break and continue Statements

• **continue** statement
  – Used in **while**, **for**, **do/while**
  – Skips remainder of loop body
  – Proceeds with next iteration of loop

• **while** and **do/while** structure
  – Loop-continuation test evaluated immediately after the **continue** statement

• **for** structure
  – Increment expression executed
  – Next, loop-continuation test evaluated
// Fig. 2.27: fig02_27.cpp
// Using the continue statement in a for structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
    // loop 10 times
    for ( int x = 1; x <= 10; x++ ) {
        // if x is 5, continue with next iteration of loop
        if ( x == 5 )
            continue; // skip remaining code in loop body
        cout << x << " "; // display value of x
    } // end for structure
    cout << "\nUsed continue to skip printing the value 5" << endl;
    return 0; // indicate successful termination
// end function main

Used continue to skip printing the value 5
2.19 Logical Operators

• Used as conditions in loops, if statements

• && (logical AND)
  - true if both conditions are true
    if ( gender == 1 && age >= 65 )
    ++seniorFemales;

• || (logical OR)
  - true if either of condition is true
    if ( semesterAverage >= 90 || finalExam >= 90 )
    cout << "Student grade is A" << endl;
2.19 Logical Operators

• ! (logical **NOT**, logical negation)
  – Returns **true** when its condition is **false**, & vice versa
    
    ```
    if ( !( grade == sentinelValue ) )
    cout << "The next grade is " << grade << endl;
    ```

    Alternative:
    
    ```
    if ( grade != sentinelValue )
    cout << "The next grade is " << grade << endl;
    ```
2.20 Confusing Equality (==) and Assignment (=) Operators

- **Common error**
  - Does not typically cause syntax errors

- **Aspects of problem**
  - Expressions that have a value can be used for decision
    - Zero = false, nonzero = true
  - Assignment statements produce a value (the value to be assigned)
2.20 Confusing Equality (==) and Assignment (=) Operators

- Example

```cpp
if ( payCode == 4 )
    cout << "You get a bonus!" << endl;
```
- If payCode is 4, bonus given

- If == was replaced with =

```cpp
if ( payCode = 4 )
    cout << "You get a bonus!" << endl;
```
- Paycode set to 4 (no matter what it was before)
- Statement is true (since 4 is non-zero)
- Bonus given in every case
2.20 Confusing Equality (==) and Assignment (=) Operators

- **Lvalues**
  - Expressions that can appear on left side of equation
  - Can be changed (i.e., variables)
    - \( x = 4; \)

- **Rvalues**
  - Only appear on right side of equation
  - Constants, such as numbers (i.e. cannot write \( 4 = x; \))

- Lvalues can be used as rvalues, but not vice versa
2.21 Structured-Programming Summary

• Structured programming
  – Programs easier to understand, test, debug and modify

• Rules for structured programming
  – Only use single-entry/single-exit control structures
  – Rules
    1) Begin with the “simplest flowchart”
    2) Any rectangle (action) can be replaced by two rectangles (actions) in sequence
    3) Any rectangle (action) can be replaced by any control structure (sequence, if, if/else, switch, while, do/while or for)
    4) Rules 2 and 3 can be applied in any order and multiple times
2.21 Structured-Programming Summary

Representation of Rule 3 (replacing any rectangle with a control structure)
2.21 Structured-Programming Summary

• All programs broken down into
  – Sequence
  – Selection
    • if, if/else, or switch
    • Any selection can be rewritten as an if statement
  – Repetition
    • while, do/while or for
    • Any repetition structure can be rewritten as a while statement