Control Structures

Outline

- Introduction
- Algorithms
- Control Structures
- The `if` Selection Structure
- The `if/else` Selection Structure
- The `switch` Multiple-Selection Structure
- Assignment Operators
- Increment and Decrement Operators

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

Fundamental Building Blocks of Programs

- THERE ARE TWO BASIC ASPECTS of programming:
  - data and instructions.
- To work with data
  - you need to understand **variables** and **types**
- To work with instructions
  - you need to understand **control structures** and **subroutines**.
  - You'll spend a large part of the course becoming familiar with these concepts

A few good programming practices

- **Be consistent with**
  - Use proper spac
  - There's nothing
- **Be consistent wit**
  - Chose one
- **Use global va**
  - Avoid ther as possible
- **Comment.** Ac nts t escribe both *what* the code is doing and w ide r way Ref:
- **Che**
- **Prov**
- **Rec**

http:/ es.php

A well formtated piece of code

```
int main()
{
    if ( num1 == num2 )
        cout << num1 << " is equal to " << num2 << endl;
    if ( num1 != num2 )
       cout << num1 << " is not equal to " << num2 << endl;
```

A poorly formtated piece of code

```
int main()
{ if (num1==num2)
cout<<num1<<" is equal to "<<num2<<endl;
if (num1!=num2)
cout<<num1<<" is not equal to "<<num2<<endl; }
```

Algorithms

- All computing problems
  - can be solved by executing a series of actions in a specific order
- Algorithm
  - A procedure determining the
    - Actions to be executed
    - Order in which these actions are to be executed
- Program control
  - Specifies the order in which statements are to executed

The Algorithm to Start the Car

1. Insert the key
2. Make sure car is in neutral gear
3. Press the gas pedal/ (Accelerator)
4. Turn the key to the start position
5. If the engine starts in 6 seconds ➔
    1. Release the key to the ignition position
6. Else if the engine does not start in 6 seconds ➔
    1. Release the key and gas pedal
    2. Wait for 10 seconds , and repeat the steps 3 – 6, but no more than 5 times
7. If the car does not start
    1. Call the workshop

6

2.3 Pseudocode

- Pseudocode
  - Artificial, informal language used to develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Allows us to “think out” a program before writing the code for it
  - Easy to convert into a corresponding C++ program
  - Consists only of executable statements

```
Example:
If student’s grade is greater than or equal to 60
    Print "Passed"
```

Control Structures

- Sequential execution
  - Statements executed one after the other in the order written
- Transfer of control
  - When the next statement executed is not the next one in sequence
- Bohm and Jacopini: all programs written in terms of 3 control structures
  - **Sequence structure**
    - Built into C++. Programs executed sequentially by default.
  - **Selection structures**
    - C++ has three types - **`if`**, **`if/else`**, and **`switch`**
  - **Repetition structures**
    - C++ has three types - **`while`**, **`do/while`**, and **`for`**

- **Flowchart**
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols connected by arrows called flowlines.
  - **Rectangle symbol** (action symbol)
    - Indicates any type of action.
  - **Oval symbol**
    - indicates beginning or end of a program, or a section of code (circles).
  - **Diamond symbol** (decision symbol)
    - indicates decision is to be made
- single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (control-structure stacking).
  - Makes programs easy to build.

The if Selection Structure

- Selection structure
  - used to 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 and program goes on to next statement
  - If the condition is **`false`**
    - print statement is ignored and the program goes onto the next statement
  - Indenting makes programs easier to read
    - C++ ignores whitespace characters

The if Selection Structure

- Translation of pseudocode statement into C++:

```
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 the condition, follow appropriate path
- **`if`** structure is a single-entry/single-exit structure

The if Selection Structure

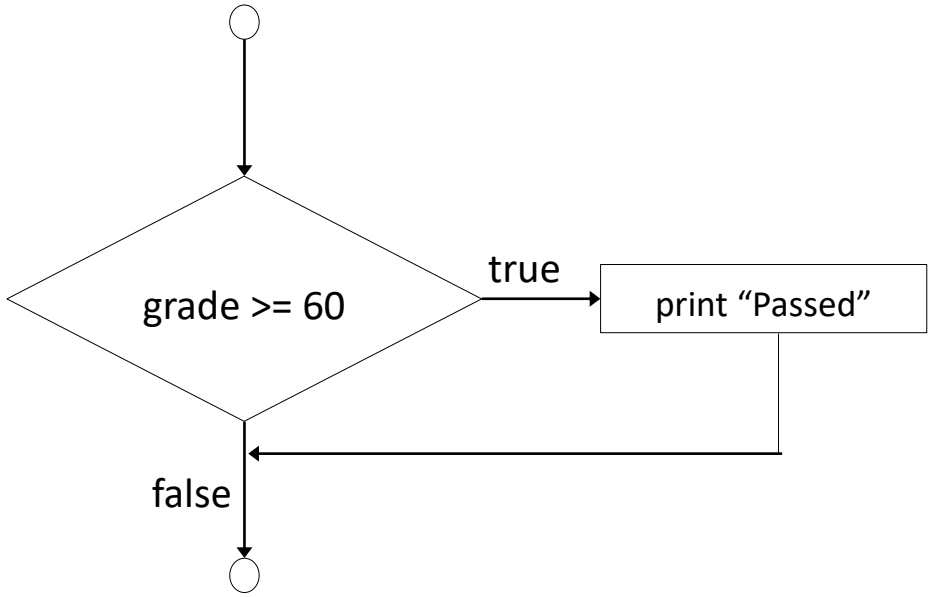
- Translation of pseudocode statement into C++:

```
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 the condition, follow appropriate path
- **`if`** structure is a single-entry/single-exit structure

The if Selection Structure

- Flowchart of pseudocode statement



A decision can be made on any expression.
zero - **`false`**
nonzero - **`true`**
Example:
`3 - 4` is **`true`**

The `if/else` Selection Structure

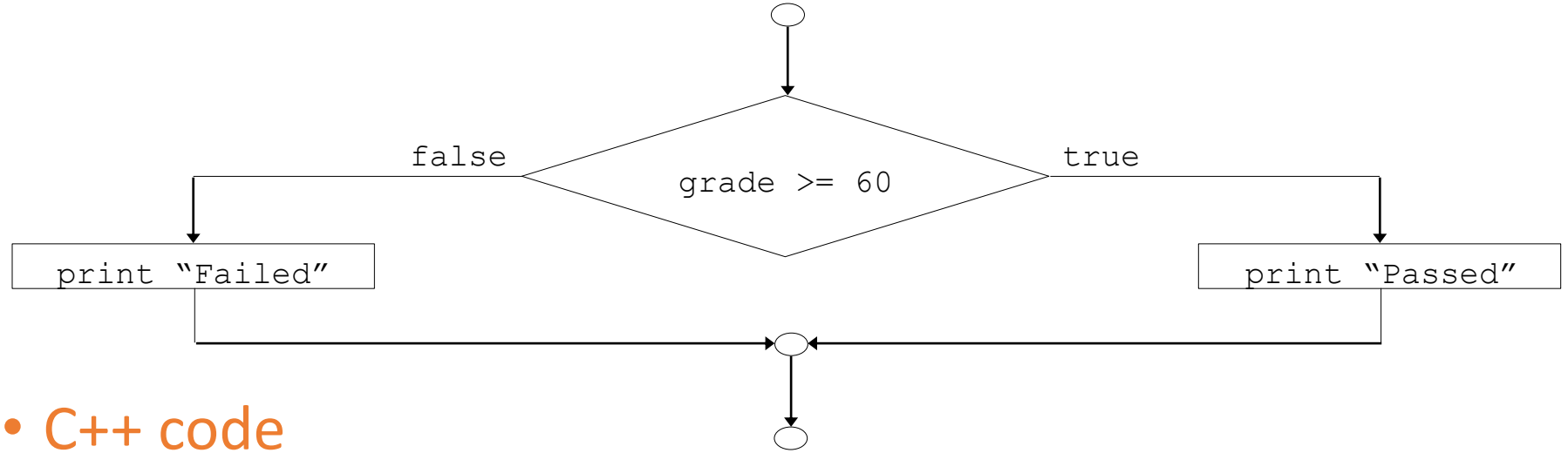
- **`if`**
  - Only performs an action if the condition is true
- **`if/else`**
  - A different action is performed when condition is true and when condition is false
- Psuedocode

```
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";
```

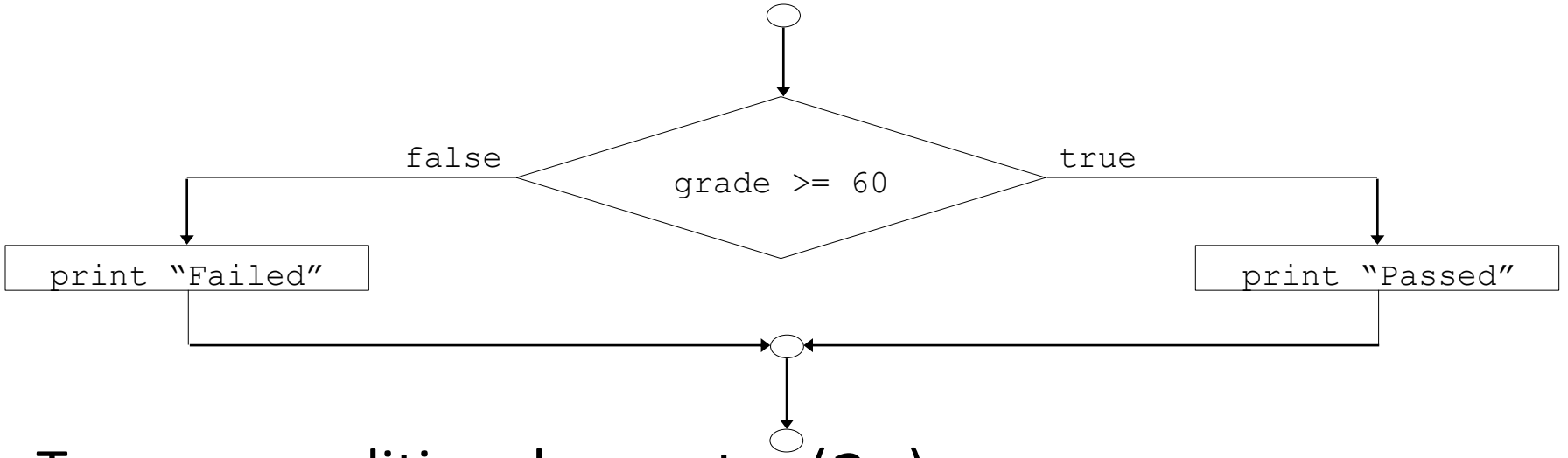
The `if/else` Selection Structure



- C++ code

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

The `if/else` Selection Structure



- Ternary conditional operator (**`?:`**)
  - Takes three arguments (condition, value if **`true`**, value if **`false`**)
- Our pseudocode could be written:

```
cout << ( grade >= 60 ? "Passed" : "Failed" );
```

- Nested **`if/else`** structures
  - Test for multiple cases by placing **`if/else`** selection structures inside **`if/else`** selection structures.

```
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"
```

  - Once a condition is met, the rest of the statements are skipped

The if/else Selection Structure

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if ( grade >= 60 )
     cout << "Passed.\n";
  else {
     cout << "Failed.\n";
     cout << "You must take this course again.\n";
  }
```

  - Without the braces,

```
cout << "You must take this course again.\n";
```

    would be automatically executed

- Block
  - Compound statements with declarations

The if/else Selection Structure

- Syntax errors
  - Errors caught by compiler
- Logic errors
  - Errors which have their effect at execution time
    - Non-fatal logic errors
      - program runs, but has incorrect output
    - Fatal logic errors
      - program exits prematurely

The `switch` Multiple-Selection Statement

- `switch`
  - Useful when a variable or expression is tested for all the values it can assume and different actions are taken
- Format
  - Series of `case` labels and an optional `default` case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
    }
```

  - **break; // exits from statement**

Same actions taken for two cases

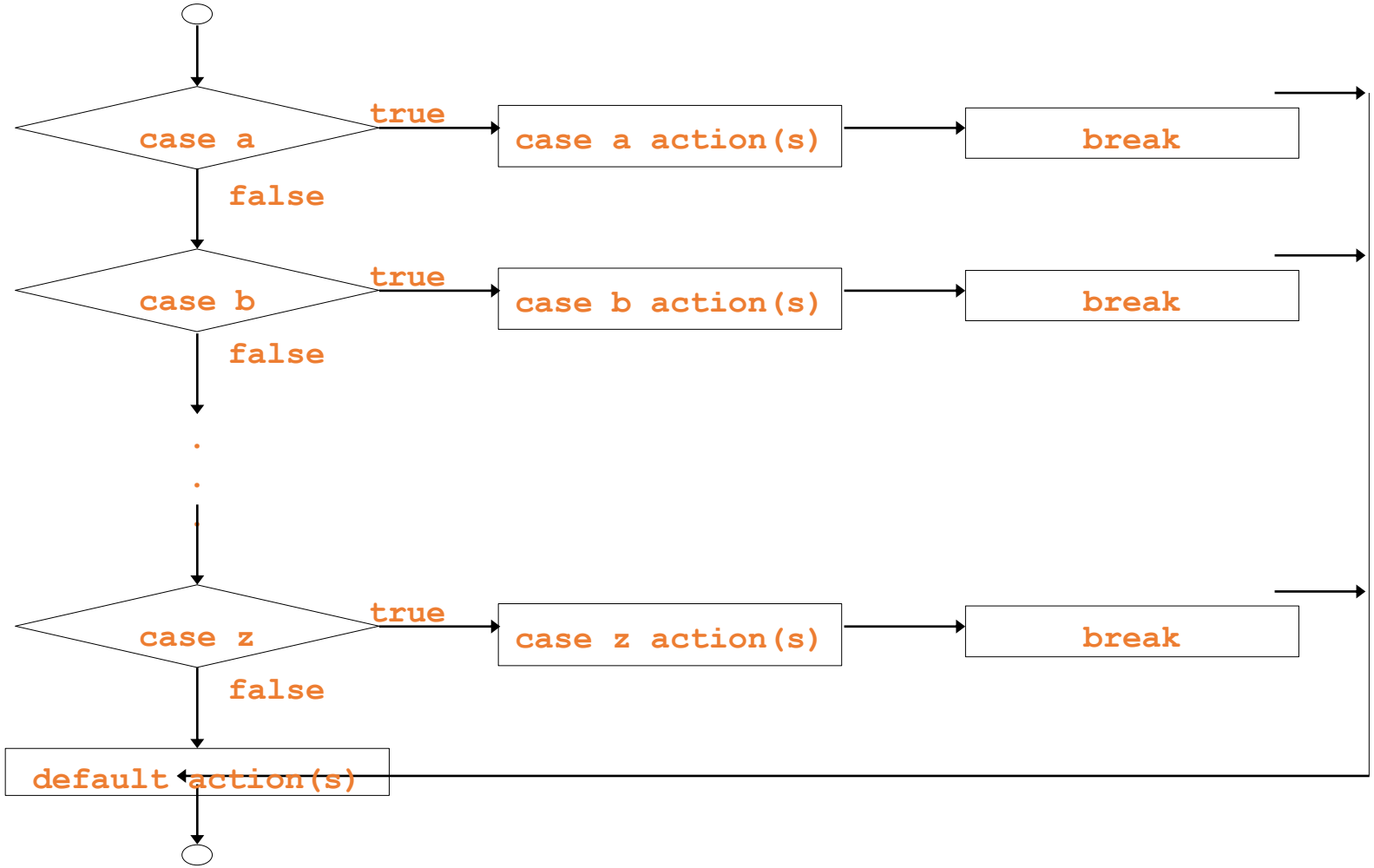
```
switch ( value ){
   case '1':
   case '2':
      actions /* one or more statements */
      break;

   case '3':
   case '4':
      actions /* one or more statements */
      break;

   default:
      actions  /* one or more statements */
      break;
   }
```

The switch Multiple-Selection Structure

- **`switch`**
  - Useful when variable or expression is tested for multiple values
  - Consists of a series of **`case`** labels and an optional **`default`** case



The `switch` Multiple-Selection Statement

- `switch`
  - Useful when a variable or expression is tested for all the values it can assume and different actions are taken
- Format
  - Series of `case` labels and an optional `default` case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
    }
```

  - `break;` exits from statement

Same actions taken for two cases

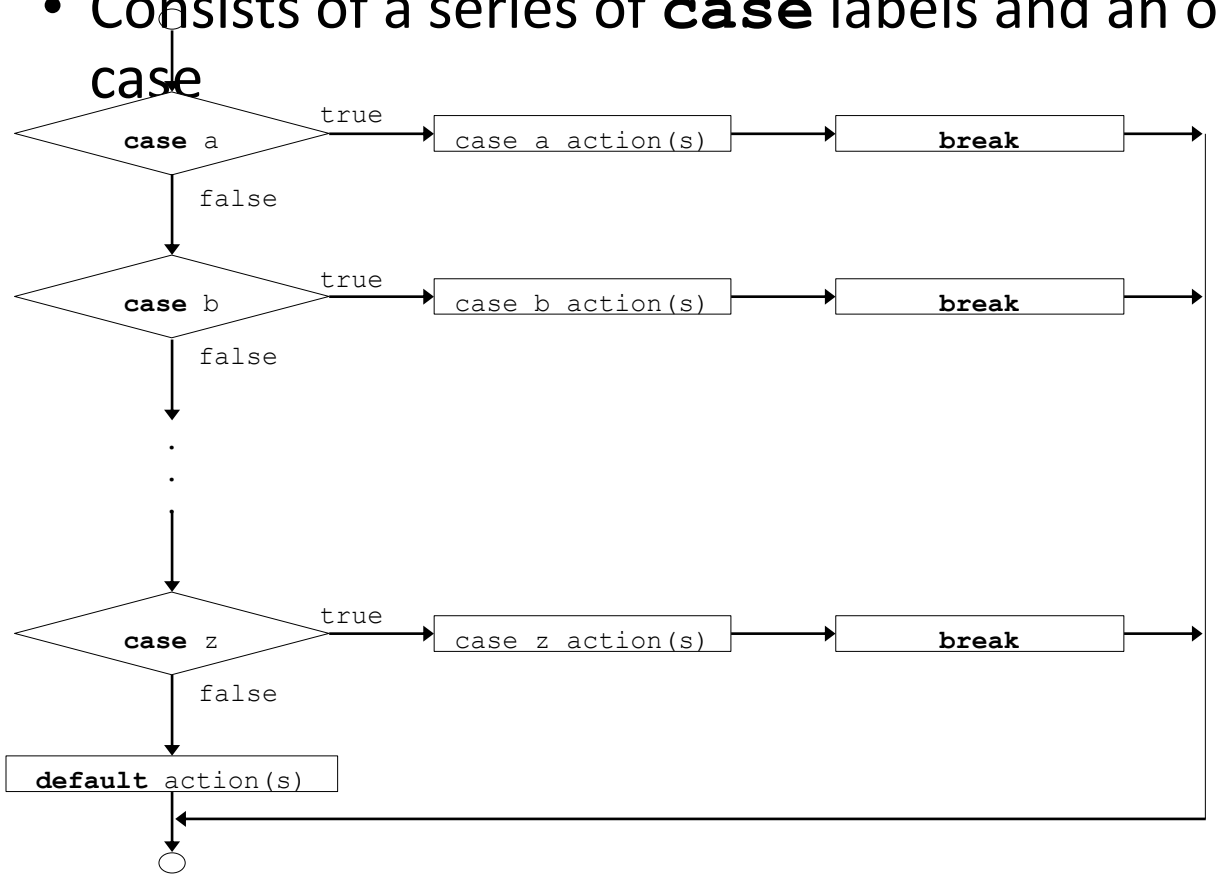
```
switch ( value ){
    case '1':
    case '2':
        actions /* one or more statements */
        break;

    case '3':
    case '4':
        actions /* one or more statements */
        break;

    default:
        actions  /* one or more statements */
        break;
    }
```

The switch Multiple-Selection Structure

- **`switch`**
  - Useful when variable or expression is tested for multiple values
  - Consists of a series of **`case`** labels and an optional **`default`** case



```
1  // Fig. 2.22: fig02_22.cpp
2  // Counting letter grades
3  #include <iostream.h>
4
5
6
7
8
9  int main()
10 {
11    int grade,        // one grade
12        aCount = 0,   // number of A's
13        bCount = 0,   // number of B's
14        cCount = 0,   // number of C's
15        dCount = 0,   // number of D's
16        fCount = 0;   // number of F's
17
18    cout << "Enter the letter grades." << endl
19         << "Enter the E character to end input." << endl;
20
21    while ( ( grade = cin.get() ) != 'E' ) {
22
23       switch ( grade ) {
24
25          case 'A':  // grade was uppercase A
26          case 'a':  // or lowercase a
27             ++aCount;
28             break;  // necessary to exit switch
29
30          case 'B':  // grade was uppercase B
31          case 'b':  // or lowercase b
32             ++bCount;
33             break;
34
```

Notice how the **case** statement is used

1. **Initialize variables**

2. **Input data**

2.1 Use switch loop to update `count`

```
35          case 'C':  // grade was uppercase C
36          case 'c':  // or lowercase c
37             ++cCount;
38             break;
39
40          case 'D':  // grade was upper
41          case 'd':  // or lowercase d
42             ++dCount;
43             break;
44
45          case 'F':  // grade was upper
46          case 'f':  // or lowercase f
47             ++fCount;
48             break;
49
50          case '\n': // ignore newlines,
51          case '\t': // tabs,
52          case ' ':  // and spaces in
53             break;
54
55          default:   // catch all other characters
56             cout << "Incorrect letter grade entered."
57                  << " Enter a new grade." << endl;
58             break;  // optional
59       }
60    }
61
62    cout << "\n\nTotals for each letter grade are:"
63         << "\nA: " << aCount
64         << "\nB: " << bCount
65         << "\nC: " << cCount
66         << "\nD: " << dCount
67         << "\nF: " << fCount << endl;
68
69    return 0;
70 }
```

2.1 Use switch loop to
e count

nt results

`break` causes **`switch`** to end and the program continues with the first statement after the **`switch`** structure.

Notice the **`default`** statement.

```
Enter the letter grades.
Enter the E character to end input.
a
B
c
C
A
d
f
C
G
Incorrect letter grade entered. Enter a new grade.
D
A
b

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

Program Output

Assignment Operators

- Assignment expression abbreviations

  `c = c + 3;` can be abbreviated as `c += 3;` using the addition assignment operator

- Statements of the form

  ```
  variable = variable operator expression;
  ```

  can be rewritten as

  ```
  variable operator= expression;
  ```

- Examples of other assignment operators include:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

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.
- If `c = 5`, then
  - `cout << ++c;` prints out `6` (`c` is changed before `cout` is executed)
  - `cout << c++;` prints out `5` (`cout` is executed before the increment. `c` now has the value of `6`)