Chapter 2 - Control Structures

Outline

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

```
cout << "Welcome ";
cout << "to ";
cout << "C++";
```

- **Transfer of control:** Next statement executed *not* next one in sequence

```
cout << "Welcome";
cout << " to";
if (course==105)
  cout << " C!";
else if (course==181)
  cout << " C++! ";
else
  { // Print error message and exit
       cerr << "Unknown course";
       exit(-1);
  }
```


2.4 Control Structures

- 3 control structures (Bohm and Jacopini) are sufficient to express any algorithm
  - Sequence structure
    - Programs executed sequentially by default
  - Selection structures
    - **`if`**, **`if/else`**, **`switch`**
  - Repetition structures
    - **`while`**, **`do/while`**, **`for`**

- "Go To Statement Considered Harmful"
  Edsger W. Dijkstra, CACM Mar 1968
  http://www.acm.org/classics/oct95/
  Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148

Example of "goto" statement

```
 cout << "Welcome";
 cout << " to";

 if (course == 181)
   goto greet181;

 if (course == 105)
   goto greet105;

 cerr << "\nError: Unknown Course" << endl;
 return -1; // exit with error status

greet181:
 cout << " C++!" << endl;
 goto endProgram;

greet105:
 cout << " C!" << endl;
 goto endProgram;

endProgram:
 cout << "Goodbye for now" << endl;

 return 0; // exit successfully.
```

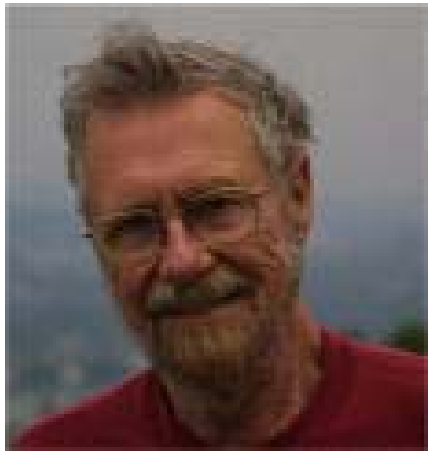

Dijkstra Letter Quotes

Edsger W. Dijkstra
1930-2002

- ...the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce.
- ...the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code).
- we should ... shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.
- The **go to** statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program

Go To Statement Considered Harmful, *Communications of the ACM*, Vol. 11, No. 3, March 1968, pp. 147-148

More Dijkstra Quotes



- "Computer Science is no more about computers than astronomy is about telescopes."
- "A Programming Language is a tool that has profound influence on our thinking habits."
- "The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague."
- "Progress is possible only if we train ourselves to think about programs without thinking of them as pieces of executable code."
- "Program testing can best show the presence of errors but never their absence."

2.4 C++ keywords

Cannot be used as identifiers or variable names

Keywords common to the** C and C++ programming **languages

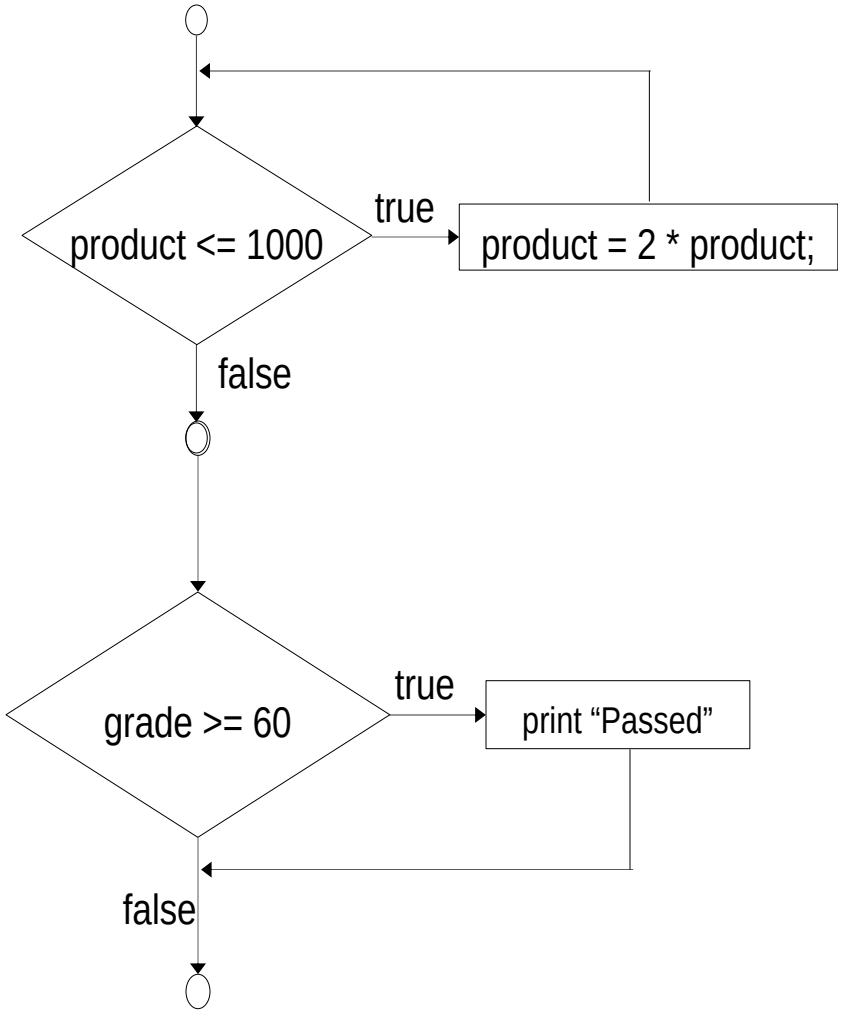
`auto`	`break`	`case`	`char`	`const`
`continue`	`default`	`do`	`double`	`else`
`enum`	`extern`	`float`	`for`	`goto`
`if`	`int`	`long`	`register`	`return`
`short`	`signed`	`sizeof`	`static`	`struct`
`switch`	`typedef`	`union`	`unsigmed`	`void`
`volatile`	`while`			

C++ only keywords

`asm`	`bool`	`catch`	`class`	`const_cast`
`delete`	`dynamic_cast`	`explicit`	`false`	`friend`
`inline`	`mutable`	`namespace`	`new`	`operator`
`private`	`protected`	`public`	`reinterpret_cast`	
`static_cast`	`template`	`this`	`throw`	`true`
`try`	`typeid`	`typename`	`using`	`virtual`
`wchar_t`				

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 (or circle) symbol
    - Beginning or end (of a program, or a section of code)
- Single-entry/single-exit control structures ("no gotos")
  - Connect exit point of one to entry point of the next
  - Control structure stacking



2.5 if Selection Structure

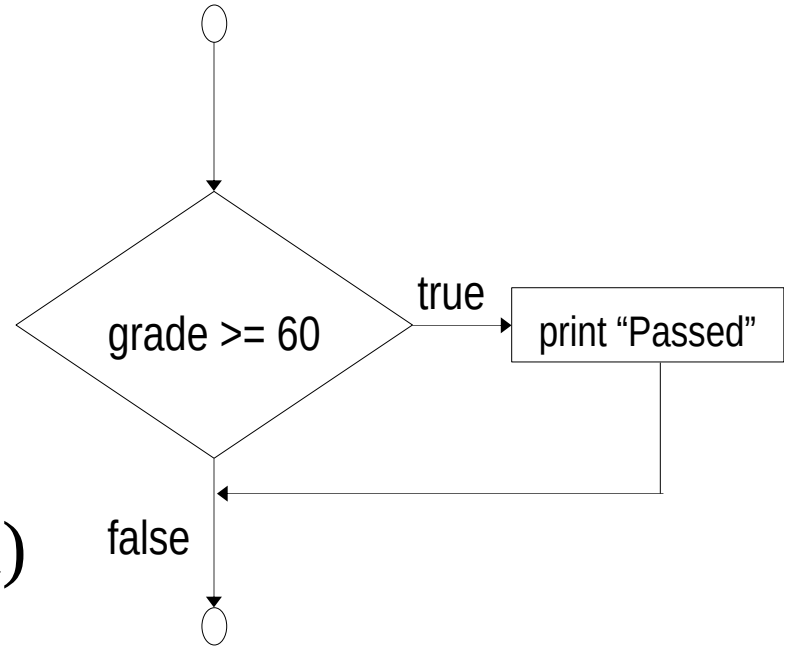
- Selection structure: Choose among alternative courses of action

  If student's grade is greater than or equal to 60

  Print "Passed"

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

  - Indenting makes programs easier to read
    - C++ ignores whitespace characters (tabs, spaces, etc.)



- Diamond symbol (decision symbol)
  - Indicates decision is to be made
  - Contains an expression that can be true or false
    - Note: in C/C++, every expression has true/false value: zero implies false, nonzero implies true. Example: if (3 - 4) is interpreted as "true".
  - `if` structure has single-entry/single-exit

2.6 `if/else` Selection Structure

- **`if`**
  - Performs action if condition true
- **`if/else`**
  - Different actions if conditions true or false

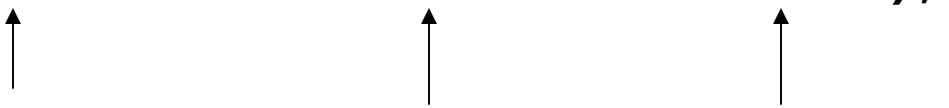
if student's grade is greater than or equal to 60
print "Passed"
else
print "Failed"

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


2.6 Ternary conditional operator (? :)

- Three arguments:
  (condition, value if **`true`**, value if **`false`**)

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



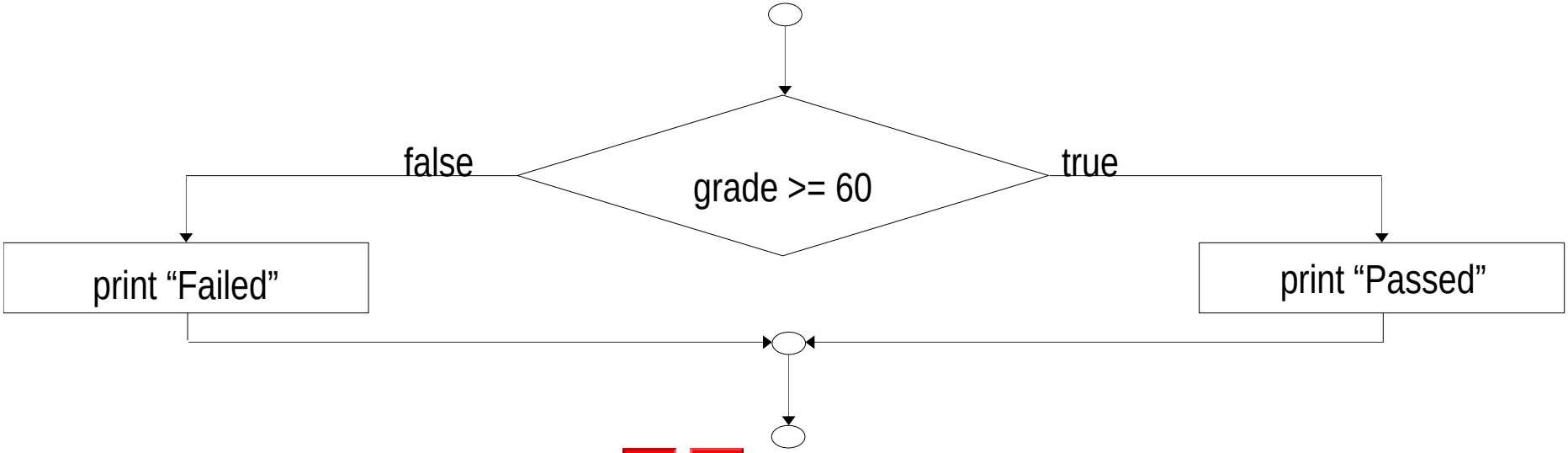
Comparison

if/else

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

vs. ternary operator

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



2.6 `if/else` Selection Structure

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

```
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

```
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

```
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

2.6 Indentation Rules

- Deitel/Deitel prefer:

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

- Conrad prefers:

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

- Whatever you do, be consistent.
  (choose one way or other way, otherwise may lose points...)

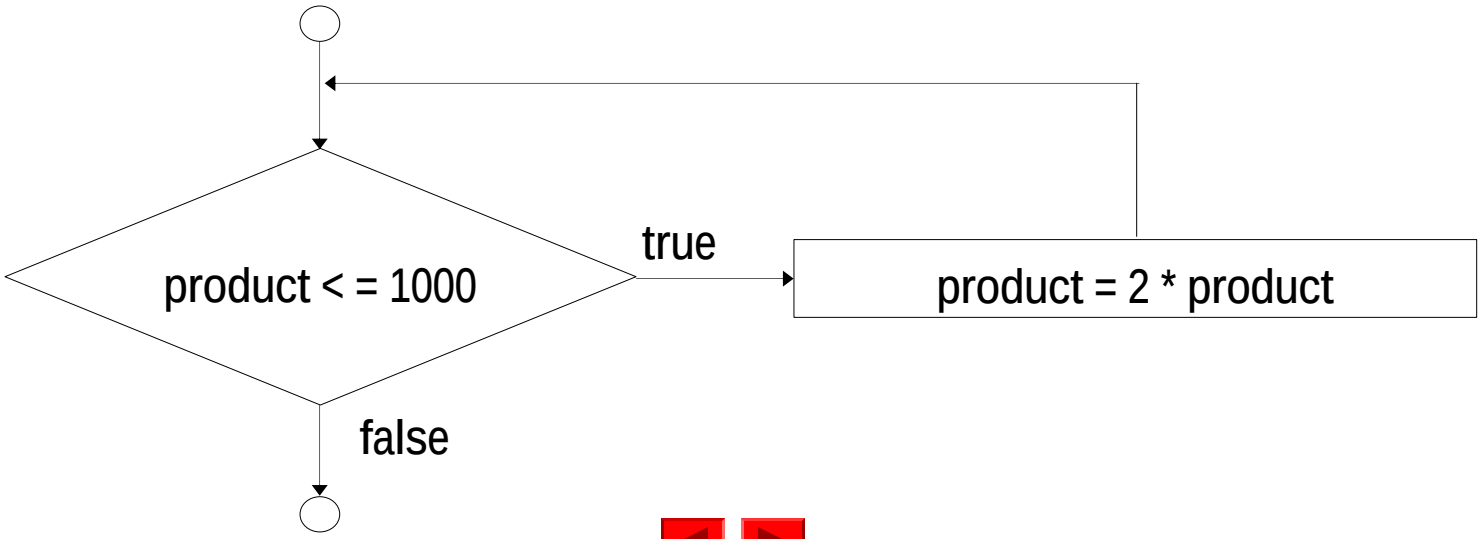
2.7 while Repetition Structure

- Repetition structure
  - Action repeated while some condition remains true

    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

```
int product = 2;
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.
  Determine the class average on the quiz.

  - We know there are *exactly* 10 scores.
  - So, we are going to need to repeat "something" *exactly* 10 times.
  - There may be some stuff to do first to set things up,
    then some stuff to do at the end to finish up, as well.

2.8 Pseudocode for finding class average (counter controlled repetition)

- Example

  A class of ten students took a quiz.
  The grades (integers in the range 0 to 100) for this quiz are available.
  Determine the class average on the quiz.

- Pseudocode for example:

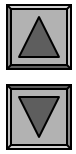
  Set total to zero

  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

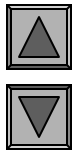
- Next: C++ code for this example



Outline

fig02_07.cpp
(1 of 2)

```
1  // Fig. 2.7: fig02_07.cpp
2  // Class average program with counter-controlled repetition.
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 total;        // sum of grades input by user
13    int gradeCounter; // number of grade to be entered next
14    int grade;        // grade value
15    int average;      // average of grades
16
17    // initialization phase
18    total = 0;          // initialize total
19    gradeCounter = 1;   // initialize loop counter
20
```



Outline

fig02_07.cpp (2 of 2)

fig02_07.cpp output (1 of 1)

```
21    // processing phase
22    while ( gradeCounter <= 10 ) {       // loop 10 times
23       cout << "Enter grade: ";          // prompt for input
24       cin >> grade;                     // read grade from user
25       total = total + grade;            // add grade to total
26       gradeCounter = gradeCounter + 1;  // increment counter
27    }
28
29    // termination phase
30    average = total / 10;                // integer division
31
32    // display result
33    cout << "Class average is
34
35    return 0;   // indicate p
36
37 } // end function main
```

The counter gets incremented each time the loop executes. Eventually, the counter causes the loop to end.

```
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
```

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

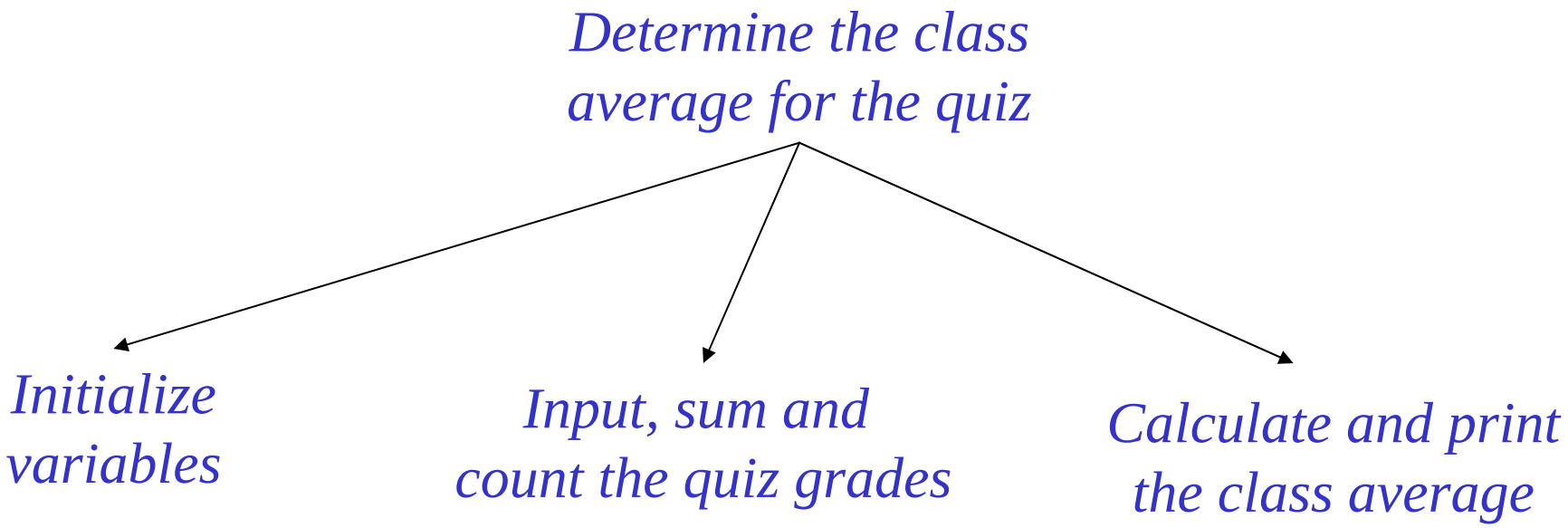
  – Sentinel chosen so it cannot be confused with regular input

    - -1 in this case

2.9 Formulating Algorithms (Sentinel-Controlled Repetition)

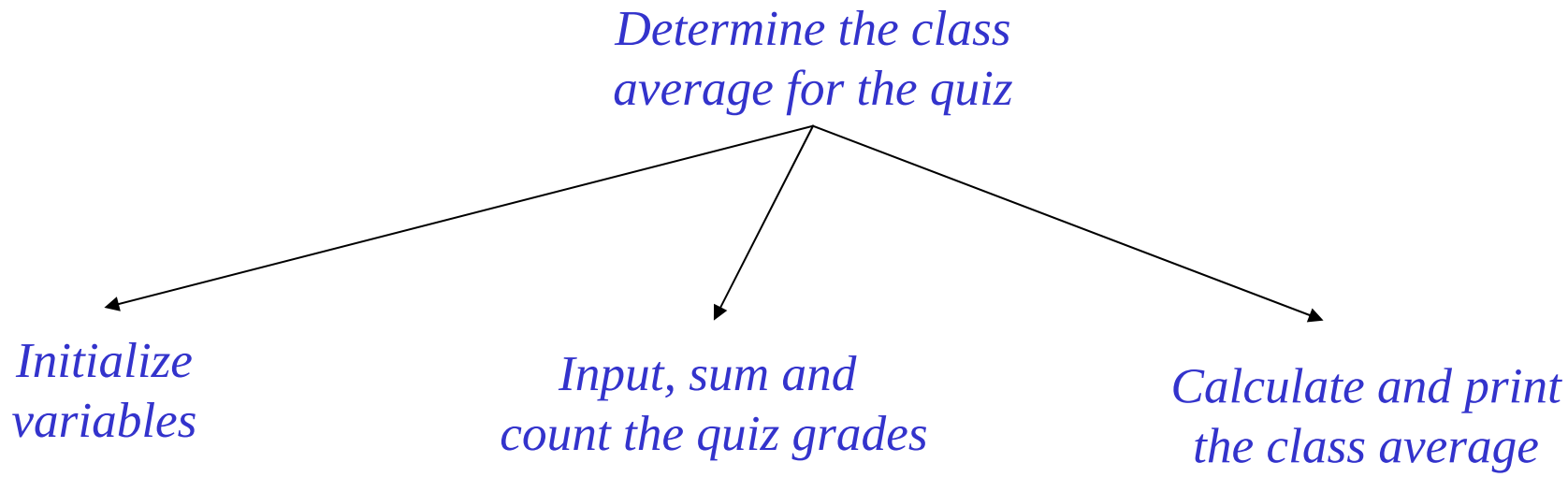
- Top-down, stepwise refinement
  - Begin with pseudocode representation of "top" (whole problem)

    Determine the class average for the quiz
  - Then, divide top into smaller tasks, list in order



2.9 Many programs have three phases

- Helps break up programs for top-down refinement
  - Initialization
    - Initializes the program variables
  - Processing
    - Input data, adjusts program variables
  - Termination
    - Calculate and print the final results



2.9 Refinement of the first two phases...

- 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 Refinement of the termination phase

- Termination

  Calculate and print the class average

  goes to

  If the counter is not equal to zero

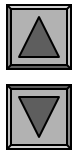
  Set the average to the total divided by the counter

  Print the average

  Else

  Print “No grades were entered”

- Next: C++ program

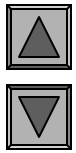


Outline

fig02_09.cpp
(1 of 3)

```
1  // Fig. 2.9: fig02_09.cpp
2  // Class average program with sentinel-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>        // parameterized stream manipulators
11
12 using std::setprecision;  // sets numeric output precision
13
14 // function main begins program execution
15 int main()
16 {
17    int total;         // sum of grades
18    int gradeCounter;  // number of grades entered
19    int grade;         // grade value
20
21    double average;    // number with decimal point for average
22
23    // initialization phase
24    total = 0;         // initialize total
25    gradeCounter = 0;  // initialize loop counter
```

Data type **`double`** used to represent decimal numbers.



Outline

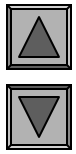
fig02_09.cpp
(2 of 3)

```
26
27    // processing phase
28    // get first grade from user
29    cout << "Enter grade, -1 to end: ";  // prompt for input
30    cin >> grade;                        // read grade from user
31
32    // loop until sentinel value read from user
33    while ( grade != -1
34       total = total + g
35       gradeCounter = gr
36
37       cout << "Enter gr
38       cin >> grade;
39
40    } // end while
41
42    // termination phase
43    // if user entered at least one grade ...
44    if ( gradeCounter != 0 ) {
45
46       // calculate average of all grades entered
47       average = static_cast< double >( total ) / gradeCounter;
48
```

`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`**.



Outline

```
49        // display average with two digits of precision
50        cout << "Class average is " << setprecision( 2 )
51             << fixed << average << endl;
52
53     } // end if part of if/else
54
55     else // if no grades were entered, output appropriate message
56        cout << "No grades were entered" << endl;
57
58     return 0;   // indicate program ended successfully
59
60  } // end function main
```

fig02_09.cpp
(3 of 3)

fig02_09.cpp
output (1 of 1)

```
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
```

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

Include **`<iostream>`**

`sion(2)` prints two digits past
t (rounded to fit precision).

t use this must include **`<iomanip>`**

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

Conrad Comments: Probably better to do error checking on input!

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

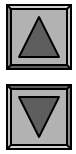
  to

  Print the number of passes

  Print the number of failures

  If more than eight students passed
  Print "Raise tuition"

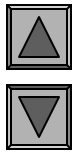
- Next: C++ program



Outline

fig02_11.cpp
(1 of 2)

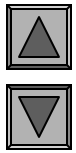
```
1  // Fig. 2.11: fig02_11.cpp
2  // Analysis of examination results.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12    // initialize variables in declarations
13    int passes = 0;           // number of passes
14    int failures = 0;         // number of failures
15    int studentCounter = 1;   // student counter
16    int result;               // one exam result
17
18    // process 10 students using counter-controlled loop
19    while ( studentCounter <= 10 ) {
20
21       // prompt user for input and obtain value from user
22       cout << "Enter result (1 = pass, 2 = fail): ";
23       cin >> result;
24
```



Outline

fig02_11.cpp
(2 of 2)

```
25       // if result 1, increment passes; if/else nested in while
26       if ( result == 1 )        // if/else nested in while
27          passes = passes + 1;
28
29       else  // if result not 1, increment failures
30          failures = failures + 1;
31
32       // increment studentCounter so loop eventually terminates
33       studentCounter = studentCounter + 1;
34
35    } // end while
36
37    // termination phase; display number of passes and failures
38    cout << "Passed " << passes << endl;
39    cout << "Failed " << failures << endl;
40
41    // if more than eight students passed, print "raise tuition"
42    if ( passes > 8 )
43       cout << "Raise tuition " << endl;
44
45    return 0;   // successful termination
46
47 } // end function main
```



Outline

fig02_11.cpp
output (1 of 1)

```
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
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
```