CHAPTER 2.2
CONTROL STRUCTURES (ITERATION)

Dr. Shady Yehia Elmashad

Outline

1. C++ Iterative Constructs
2. The for Repetition Structure
3. Examples Using the for Structure
4. The while Repetition Structure
5. Examples Using the while Structure
6. Formulating Algorithms (Counter-Controlled Repetition)
7. Formulating Algorithms with Top-Down, Stepwise Refinement
8. Nested control structures
9. Essentials of Counter-Controlled Repetition
10. The do/while Repetition Structure
11. The break and continue Statements
12. Structured-Programming Summary

1. C++ Iterative Constructs

- There are three constructs:
  - ➢ while statement
  - ➢ for statement
  - ➢ do-while statement

2. The for Repetition Structure

The general format when using **`for`** loops is

```
for ( initialization;
  LoopContinuationTest; increment )

     statement
```

Example:

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

- Prints the integers from one to ten

No semicolon after last statement

2. The for Repetition Structure

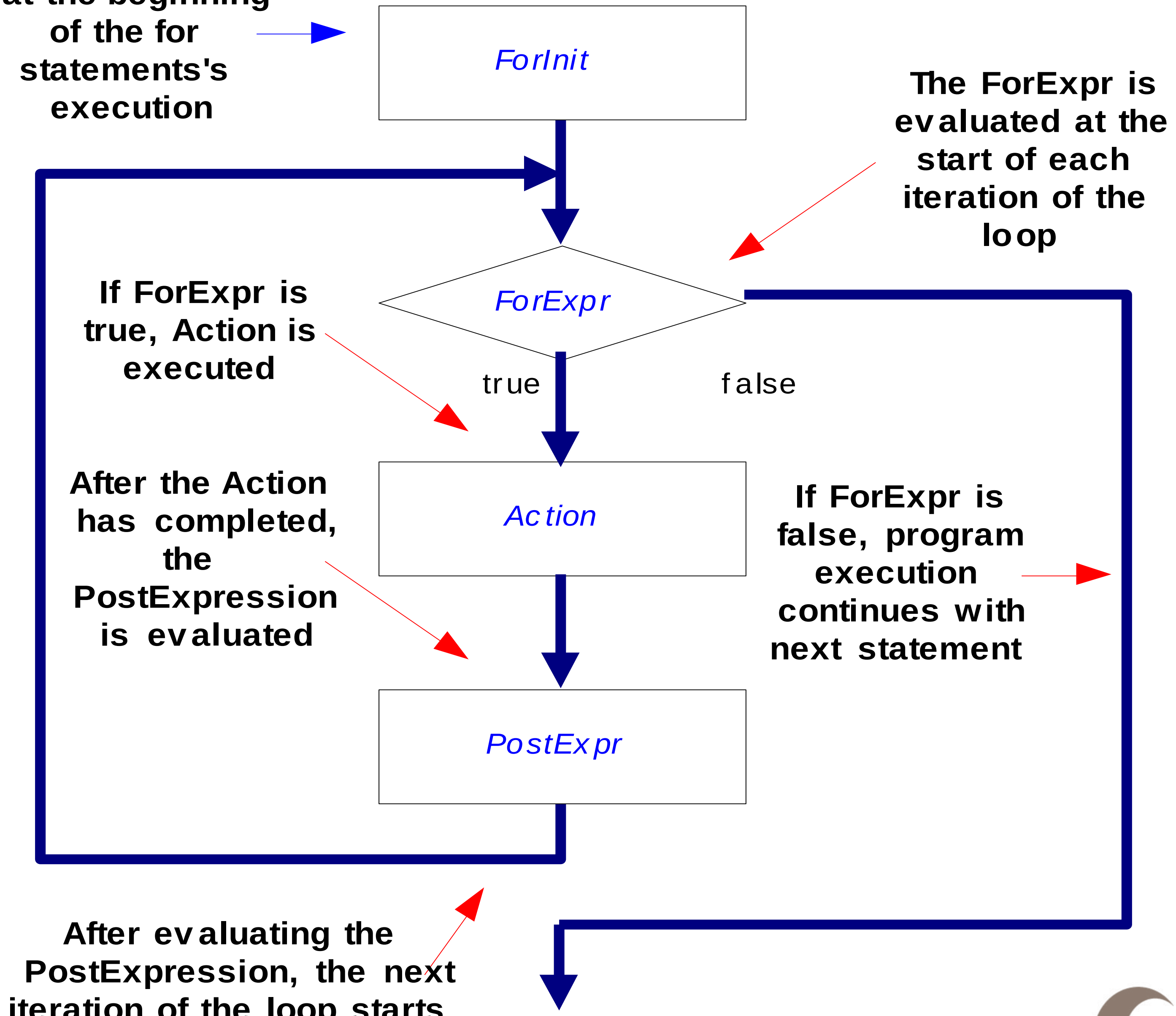
- Syntax

  `for` (*ForInit* ; *ForExpression*; *PostExpression*)

  Action

- Example

```
for (int i = 0; i < 3; ++i) {
  cout << "i is " << i << endl;
}
```



Evaluated once at the beginning of the for statements's execution

The ForExpr is evaluated at the start of each iteration of the loop

If ForExpr is true, Action is executed

After the Action has completed, the PostExpression is evaluated

If ForExpr is false, program execution continues with next statement

After evaluating the PostExpression, the next iteration of the loop starts

2. The for Repetition Structure

- **`For`** loops can usually be rewritten as **`while`** loops:

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

- Initialization and increment as comma-separated lists

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
      cout << j + i << endl;
```

3. Examples Using the for Structure

Sum the numbers from 0 to 10

```
#include <iostram.h>
void main ( )
{
int sum = 0 ;
    for ( int i = 0; i < = 10; i++ )
   {
     sum = sum + i ;
     }
cout << " Summation = " << sum ;
}
```

```
Summation =
```

3. Examples Using the for Structure

Sum the even numbers from 0 to 100

```
#include <iostream.h>
void main ( )
{
int sum = 0 ;
    for ( int i = 0; i < = 100; i+=2 )
   {
     sum = sum + i ;
     }
cout << “ Summation = “ << sum ;
}
```

```
Summation =
```

3. Examples Using the for Structure

Sum the odd numbers from 0 to 100

```
#include <iostream.h>
void main ( )
{
int sum = 0 ;
    for ( int i = 1; i < = 100; i+=2 )
   {
     sum = sum + i ;
     }
cout << “ Summation = “ << sum ;
}
```

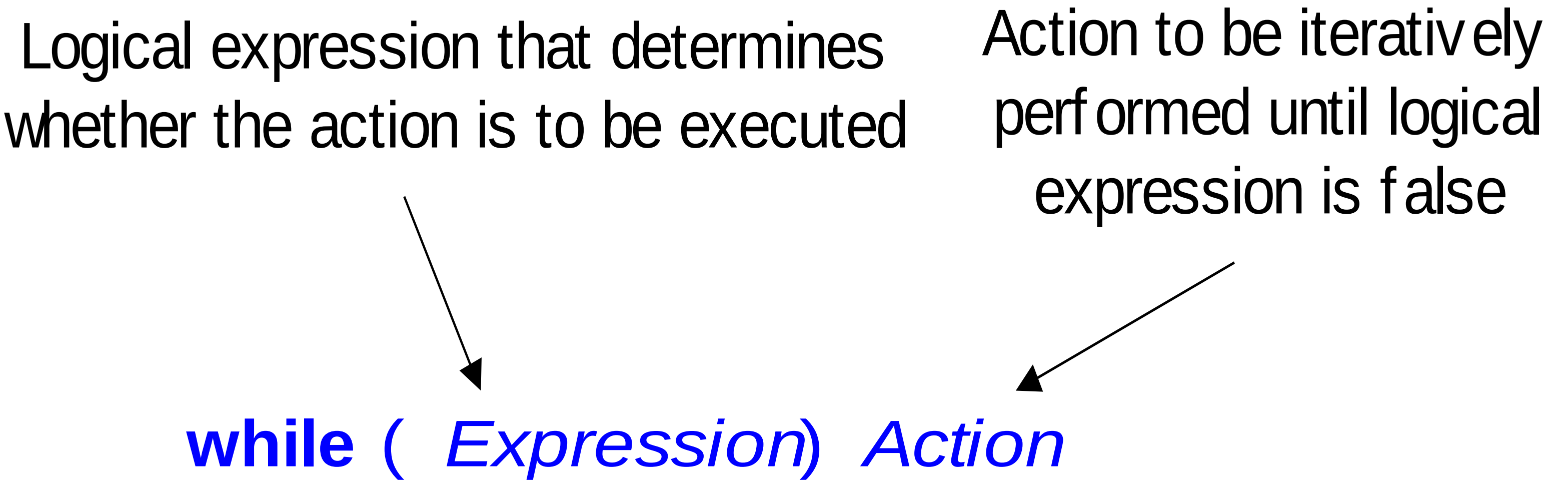
```
Summation =
```

3. Examples Using the for Structure

Printing characters depending on user entry

```
#include <iostram.h>
void main ( )
{
int n ; char  ch;
cout << “ Please enter the character:  “ ;
cin >> ch ;
cout << “ Please enter the number of
repetition: “ ;
cin >> n ;
      for ( int i = 0; i <  n ; i++ )
      cout <<  ch;
 }
```

4. The while Repetition Structure



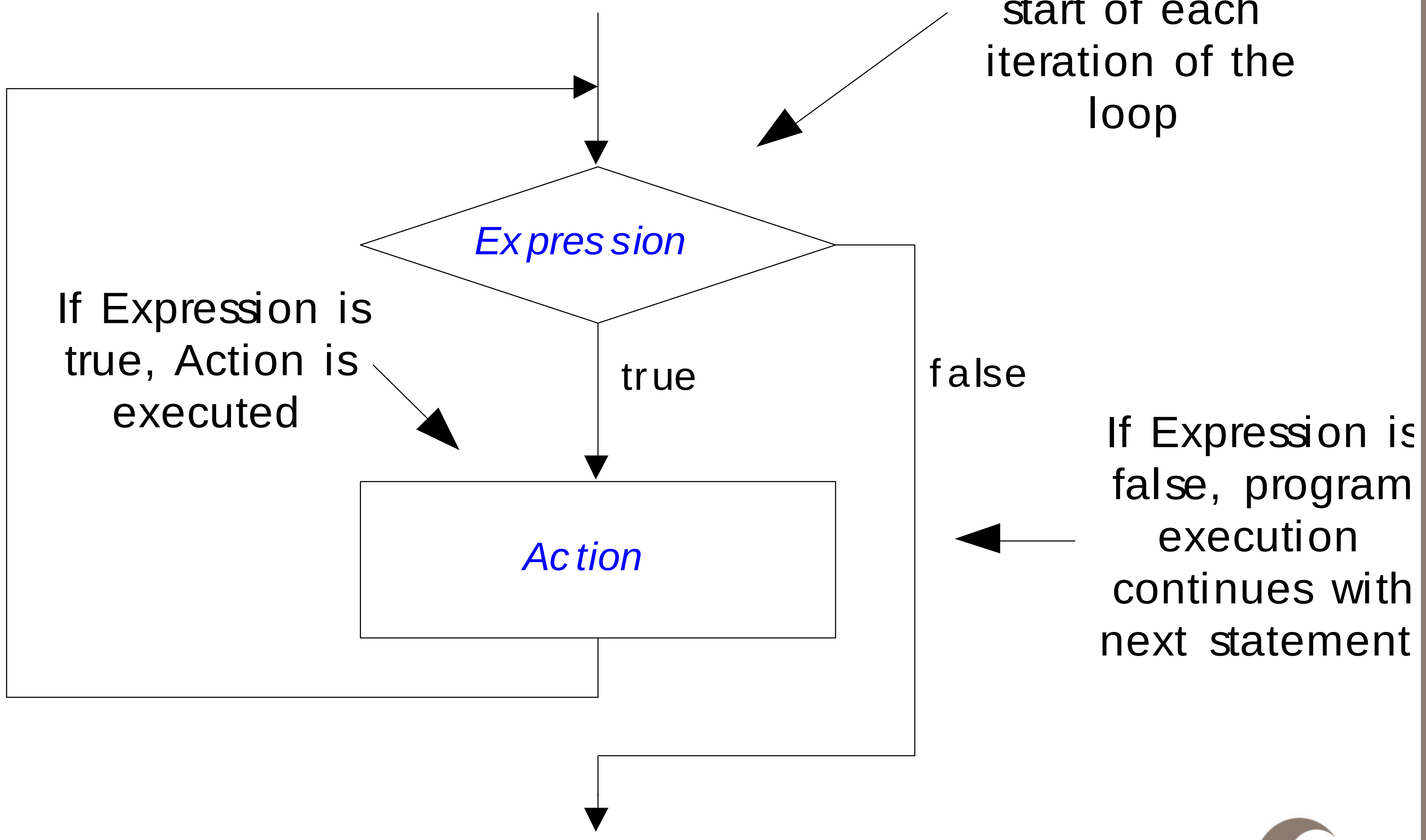
Logical expression that determines whether the action is to be executed

Action to be iteratively performed until logical expression is false

while (*Expression*) *Action*

4. The while Repetition Structure

While Semantics



Expression is evaluated at the start of each iteration of the loop

If Expression is true, Action is executed

If Expression is false, program execution continues with next statement

4. The while Repetition Structure

- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains true
  - Psuedocode

```
while there are more items on my shopping list
   Purchase next item and cross it off my list
```

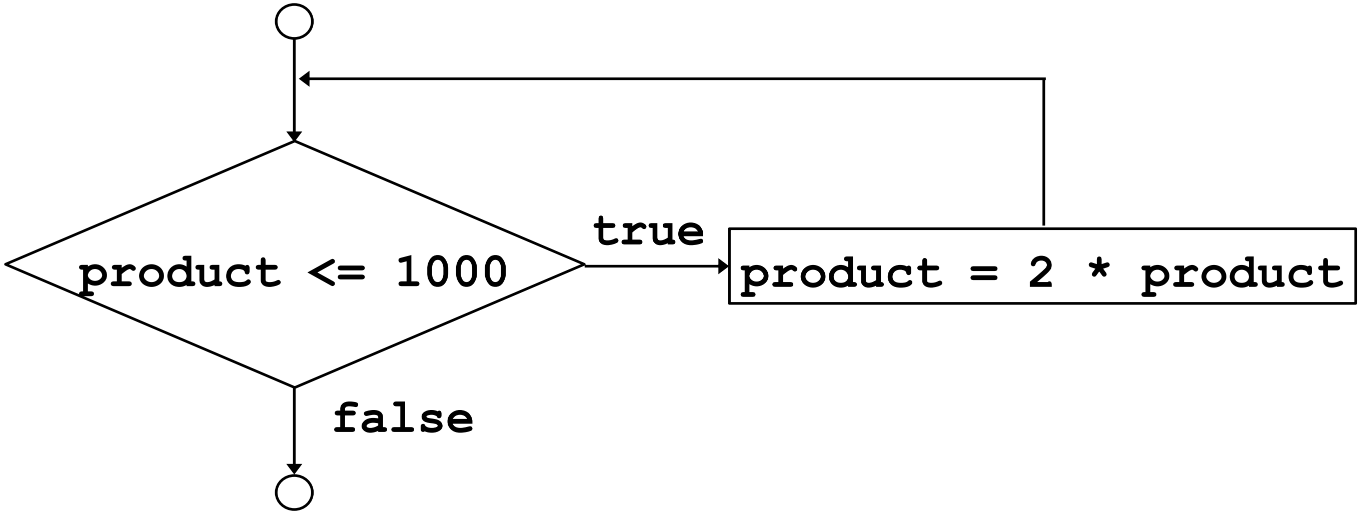
  - **`while`** loop repeated until condition becomes false.

- Example

```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```

4. The while Repetition Structure

- Flowchart of **`while`** loop



5. Examples Using the while Structure

Printing characters depending on user entry

```
#include <iostream.h>
void main ( )
{
int n, i = 0 ; char  ch;
cout << “ Please enter the character: “ ;
cin >> ch ;
cout << “ Please enter the number of
repetition: “ ;
cin >> n ;
    while ( i <  n )    {
    cout <<  ch ;
    i ++ ;
    }
}
```

5. Examples Using the while Structure

The summation of the numbers squared from 0 to 10

```
#include <iostream.h>
void main ( )
{
int sq_sum = 0, x = 0, y ;
     while ( x < =  10 )    {
      y = x * x ;
      sq_sum = sq_sum + y ;
      x ++ ;
      }
cout << “The summation of the
numbers squared from 0 to 10 “ <<
sq_sum ;
}
```

5. Examples Using the while Structure

Factorial of a number

```
#include <iostram.h>
void main ( )
{
int n, fact = 1 ;
cout << “ Please enter a number “ << endl ;
cin >> n ;
      while ( n  > 0 )    {
      fact = fact * n ;
      n -- ;
      }
cout << “ The factorial of your number is “
<< fact ;
}
```

6. Formulating Algorithms (Counter-Controlled Repetition)

- Counter-controlled repetition
  - ➢ Loop repeated until counter reaches a certain value.
- Definite repetition
  - ➢ Number of repetitions is 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.

6. 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
```

- Following is the C++ code for this example

```cpp
// Fig. 2.7: fig02_07.cpp
// Class average program with counter-controlled repetition
#include <iostream>

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

int main()
{
   int total,        // sum of grades
       gradeCounter, // number of grades entered
       grade,        // one grade
       average;      // average of grades

   // initialization phase
   total = 0;                           // clear total
   gradeCounter = 1;                    // prepare to loop

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

   // termination phase
   average = total / 10;                // integer division
   cout << "Class average is " << average << endl;

   return 0;   // indicate program ended successfully
}
```

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

Outline

1. Initialize Variables

2. Execute Loop

3. Output results

Outline

Program Output

```
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
```

7. Formulating Algorithms with Top-Down, Stepwise Refinement (Sentinel-Controlled Repetition)

- Suppose the 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 the program know to end?
- Sentinel value
  - Indicates “end of data entry”
  - Loop ends when sentinel inputted
  - Sentinel value chosen so it cannot be confused with a regular input (such as -1 in this case)

7. Formulating Algorithms with Top-Down, Stepwise Refinement (Sentinel-Controlled Repetition)

- Top-down, stepwise refinement
  - begin with a pseudocode representation of the top:

    Determine the class average for the quiz
  - Divide top into smaller tasks and list them in order:

    Initialize variables

    Input, sum and count the quiz grades

    Calculate and print the class average

7. Formulating Algorithms with Top-Down, Stepwise Refinement

- Many programs can be divided into three phases:
  - ➢ Initialization

    - Initializes the program variables
  - ➢ Processing

    - Inputs data values and adjusts program variables accordingly
  - ➢ Termination

    - Calculates and prints the final results.

    - Helps the breakup of programs for top-down refinement.
- Refine the initialization phase from

Initialize variables

to

Initialize total to zero

Initialize counter to zero

7. Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine

```
Input, sum and count the quiz grades
            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)
```

- Refine

```
Calculate and print the class average
            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”
```

Outline

1. **Initialize Variables**

2. **Get user input**

2.1 Perform Loop

```
// Fig. 2.9: fig02_09.cpp
// Class average program with sentinel-controlled repetition.
#include <iostream>

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

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   int total,         // sum of grades
       gradeCounter,  // number of grades entered
       grade;         // one grade
   double average;    // number with decimal point for average

   // initialization phase
   total = 0;
   gradeCounter = 0;

   // processing phase
   cout << "Enter grade, -1 to end: ";
   cin >> grade;

   while ( grade != -1 ) {
```

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

Outline

3. Calculate Average

3.1 Print Results

put

```
         total = total + grade;
         gradeCounter = gradeCounter + 1;
         cout << "Enter grade, -1 to end: ";
         cin >> grade;
      }

      // termination phase
      if ( gradeCounter != 0 ) {
         average = static_cast< double >( total ) / gradeCounter;
         cout << "Class average is " << setprecision( 2 )
              << setiosflags( ios::fixed | ios::showpoint )
              << average << endl;
      }
```

static_cast<double>() - treats **total** as a **double** temporarily.

Required because dividing two integers truncates the remainder.

gradeCounter is an **int**, but it gets *promoted* to **double**.

setiosflags(ios::fixed | ios::showpoint) - stream

ers with a fixed number of decimal

decimal point and trailing zeros, even if

| – separates multip

cision(2) - prints only two digits

mal point.

Programs that use this must include **<iomanip>**

```
Ent
Ent
Enter grade, -1 to end:
Enter grade, -1 to end:
Enter grade, -1 to end:
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
```

8. Nested Control Structures

- Problem:

  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".

- We can see that
  - ➢ The program must process 10 test results. A counter-controlled loop will be used.
  - ➢ Two counters can be used—one to count the number of students who passed the exam and one to count the number of students who failed the exam.
  - ➢ Each test result is a number—either a 1 or a 2. If the number is not a 1, we assume that it is a 2.

- Top level outline:

  Analyze exam results and decide if tuition should be raised

8. Nested Control Structures

- 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
```

8. 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
```

- 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"
```

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

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

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

   // process 10 students; counter-controlled loop
   while ( studentCounter <= 10 ) {
      cout << "Enter result (1=pass,2=fail): ";
      cin >> result;

      if ( result == 1 )        // if/else nested in while
         passes = passes + 1;
```

Outline

1. Initialize variables

2. Input data and count passes/failures

```
      else
         failures = failures + 1;

      studentCounter = studentCounter + 1;
   }

   // termination phase
   cout << "Passed " << passes << endl;
   cout << "Failed " << failures << endl;

   if ( passes > 8 )
      cout << "Raise tuition " << endl;

   return 0;   // successful termination
}
```

Outline

3. Print results

```
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
```

Program Output

8. Nested Control Structures

Accept 10 numbers from the user & print the max. one

```
#include <iostram.h>
void main ( )
{
int num, largest = 0 ;
   for ( int i = 0; i < 10; i ++ )  {
   cout << “ Enter a number: “ ;
   cin >> num ;
                 if ( num > largest) {
                 largest = num ;
                 }
    }
cout << “ The largest number is “ << largest
<< endl ;
}
```

8. Nested Control Structures

Multiplication Table of 5

```
#include <iostream.h>
void main ( )
{
cout << “ \ t  1   \ t  2   \ t  3   \ t  4   \ t  5 “
; << endl ;
        for ( int i = 1 ; i < = 5 ; i ++ )    {
        cout << i ;
        cout << “ \ t “ ;
                for ( int j = 1 ; j < = 5 ; j ++ )    {
                cout << i * j << “ \ t “ << “ | “ ;
                }
        cout << endl;
        }
}
```

8. Nested Control Structures

Multiplication Table of n

```
#include <iostram.h>
void main ( )   {
cout << “ Please enter a number: “ ;
cin >> n ;
      for ( int i = 1 ; i < = n ; i ++ )    {
       cout << i ;
       cout << “ \ t “ ;
       }
cout << endl ;
              for ( int j = 1 ; j < = n ; j ++ )    {
              cout << i ;
              cout << “ \ t “ ;
                     for ( int k = 1 ; k < = n ; k ++ )    {
                     cout << j * k << “ \ t “ << “ | “ ;
                     }
              cout << endl;
       }
}
```

9. Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires:
  - The name of a control variable (or loop counter).
  - The initial value of the control variable.
  - The condition that tests for the final value of the control variable (i.e., whether looping should continue).
  - The increment (or decrement) by which the control variable is modified each time through the loop.
- Example:

```
int counter =1;              //initialization
while (counter <= 10){       //repetition
condition
   cout << counter << endl;
   ++counter;                //increment
 }
```

9. Essentials of Counter-Controlled Repetition

- The declaration

```
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`**

10. The do/while Repetition Structure

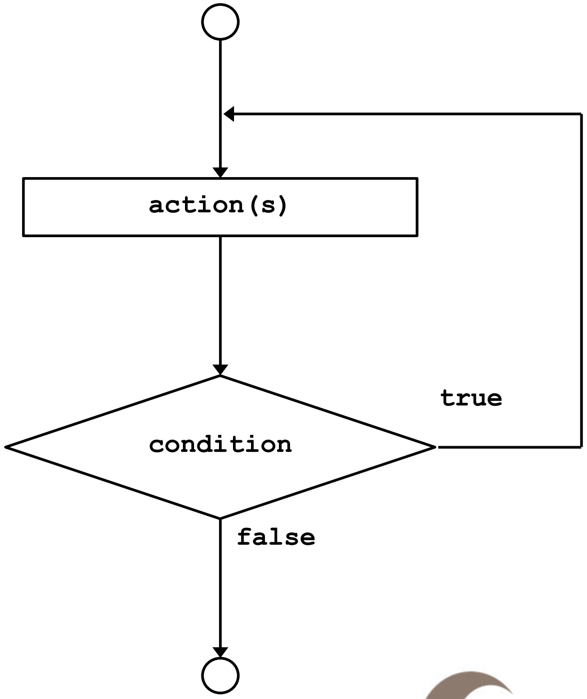
- The **`do/while`** repetition structure is similar to the **`while`** structure,
  - ➢ Condition for repetition tested after the body of the loop is executed
- Syntax:

```
do {
    statement(s)
} while ( condition );
```

- Example (letting counter = 1):

```
do {
  cout << counter << " ";
} while (++counter <= 10);
```

  - ➢ This prints the integers from **1** to **10**
- All actions are performed at least once.



11. The break and continue Statements

- **`Break`**
  - ➢ Causes immediate exit from a **`while`**, **`for`**, **`do/while`** or **`switch`** structure
  - ➢ Program execution continues with the first statement after the structure
  - ➢ Common uses of the **`break`** statement:

    - Escape early from a loop

    - Skip the remainder of a **`switch`** structure

11. The break and continue Statements

- **Continue**
  - Skips the remaining statements in the body of a **while**, **for** or **do/while** structure and proceeds with the next iteration of the loop
  - In **while** and **do/while**, the loop-continuation test is evaluated immediately after the **continue** statement is executed
  - In the **for** structure, the increment expression is executed, then the loop-continuation test is evaluated

11. The break and continue Statements

```
#include <iostream.h>
Void main()
{
    int sum = 0, num;

     // Allow the user to enter up to 10 numbers
    for (int count=0; count < 10; ++count) {
        cout << "Enter a number to add, or 0 to exit: ";
        cin >> num;

        // exit loop if user enters 0
        if (num == 0)
            break;

        // otherwise add number to our sum
        sum += num;
    }
     cout << "The sum of all the numbers you entered is " << sum << "\n";
}
```

11. The break and continue Statements

```
#include <iostream.h>
 void main ( )
{
    while (true)                  // infinite loop
    {
        cout << "Enter 0 to exit or anything else to continue: ";
        int num;
        cin >> num;

        // exit loop if user enters 0
        if (num == 0)
            break;
    }

    cout << "We're out!\n";
}
```

11. The break and continue Statements

```
#include <iostream.h>
 void main ( )
{
        for (int count=0; count  < =20; ++count) {
    // if the number is divisible by 4, skip this iteration
    if ((count % 4) == 0)
        continue;

    // If the number is not divisible by 4, keep going
    cout << count << endl;
}
}
```

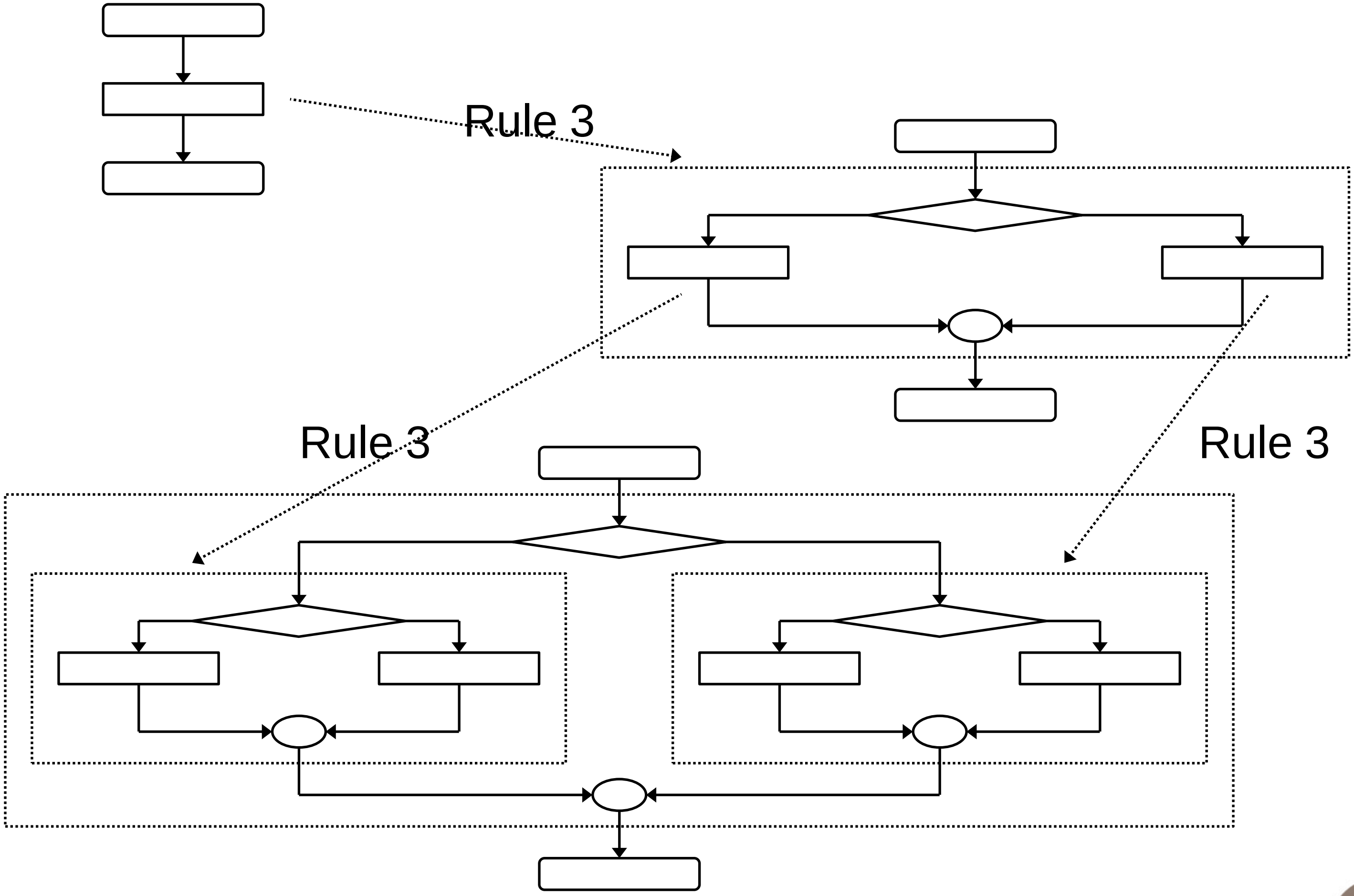
- This program prints all of the numbers from 0 to 20 that aren’t divisible by 4.

12. Structured-Programming Summary

- Structured programming
  - Programs are easier to understand, test, debug and, modify.
- Rules for structured programming
  - Only single-entry/single-exit control structures are used
  - 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.

12. Structured-Programming Summary

Representation of Rule 3 (replacing any rectangle with a control structure)



12. Structured-Programming Summary

- All programs can be 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