CS 112 Introduction to Programming

Lecture #8:

Simple Control Structures

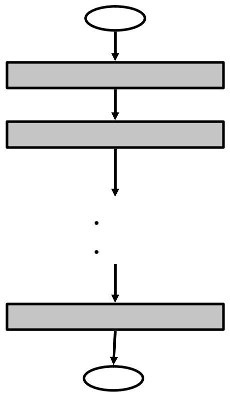
http://flint.cs.yale.edu/cs112/

Sequential Programming Revisited

```
//  Addition.cs
// An addition program.

using System;

class Addition
{
   static void Main( string[] args )
   {
      string firstNumber,   // first string entered by user
             secondNumber;  // second string entered by user

      int number1,          // first number to add
          number2,          // second number to add
          sum;              // sum of number1 and number2

      // prompt for and read first number from user as string
      Console.Write( "Please enter the first integer: " );
      firstNumber = Console.ReadLine();

      // read second number from user as string
      Console.Write( "\nPlease enter the second integer: " );
      secondNumber = Console.ReadLine();

      // convert numbers from type string to type int
      number1 = Int32.Parse( firstNumber );
      number2 = Int32.Parse( secondNumber );

      // add numbers
      sum = number1 + number2;

      // display results
      Console.WriteLine( "\nThe sum is {0}.", sum );

   } // end method Main

} // end class Addition
```

```
// prompt for and read first number from user as string
Console.Write( "Please enter the first integer: " );
firstNumber = Console.ReadLine();

// read second number from user as string
Console.Write( "\nPlease enter the second integer: " );
secondNumber = Console.ReadLine();

// convert numbers from type string to type int
number1 = Int32.Parse( firstNumber );
number2 = Int32.Parse( secondNumber );

// add numbers
sum = number1 + number2;

// display results
Console.WriteLine( "\nThe sum is {0}.", sum );
```

Sequence Structure (Flowchart)



Each of these statements could be:

- a variable declaration
- an assignment statement
- a method call (e.g., **`Console.WriteLine( …);`**)

or more complex statements (to be covered later)

3

More Interesting: Control Statements

❒ Selection (a.k.a., conditional statements): decide whether or not to execute a particular statement; these are also called the selection statements or decision statements

- **`if`** selection (one choice)
- **`if/else`** selection (two choices)
  - Also: the ternary conditional operator $\mathbf{e_1?e_2:e_3}$
- **`switch`** statement (multiple choices)

❒ Repetition (a.k.a., loop statements): repeatedly executing the same statements (for a certain number of times or until a test condition is satisfied).

- **`while`** structure
- **`do/while`** structure
- **`for`** structure
- **`foreach`** structure (Chapter 12)

4

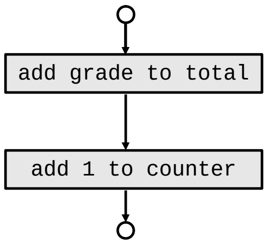
Why Control Statements ?

- Last few classes: a sequence of statements
  - *Sequential programming*
- Most programs need more flexibility in the order of statement execution
- The order of statement execution is called the flow of control

Pseudocode & Flowcharts to Represent Flow of Control

- Pseudocode
  - Artificial and informal language
  - Helps programmers to plan an algorithm
  - Similar to everyday English
  - Not an actual programming language
- Flowchart --- *a graphical way of writing pseudocode*
  - Rectangle used to show action ▭
  - Circles used as connectors ○
  - Diamonds used as decisions ◆

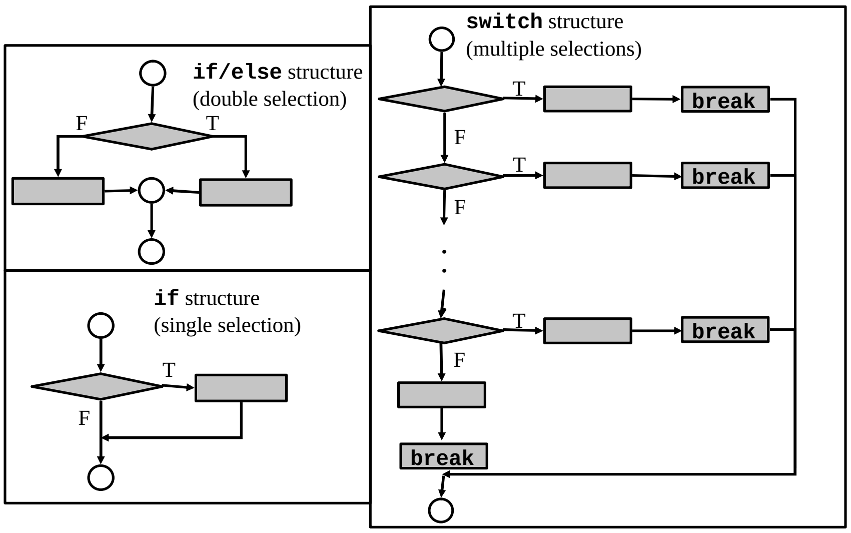
Sequence Structure



```
total = total + grade;

counter = counter + 1;
```

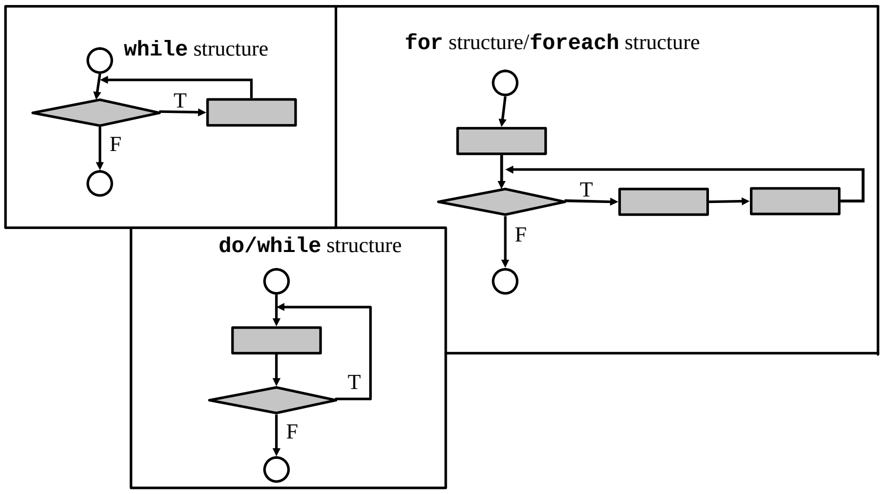
7

C# Control Structures: Selection



8

C# Control Structures: Repetition



9

if Statement

```
if ( <test> )
    <code executed if <test> is true> ;
```

- The **`if`** statement
  - Causes the program to make a selection
  - Chooses based on conditional
    - ***<test>:*** any expression that evaluates to a **`bool`** type
    - **True**: perform an action
    - **False**: skip the action
  - Single entry/exit point
  - *No semicolon after the condition*

10

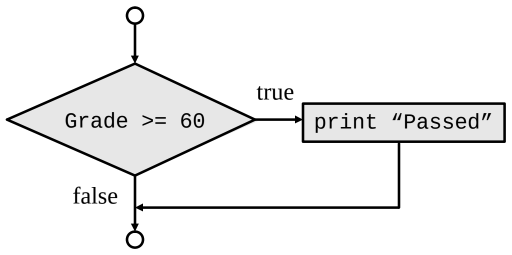
`if` Statement (cont'd)

```
if ( <test> )
{
   <code executed if <test> is true> ;
   ……
   <more code executed if <test> is true> ;
}
```

- The body of the branch can also be a block statement!
- *No semicolon after the condition*
- *No semicolon after the block statement*

11

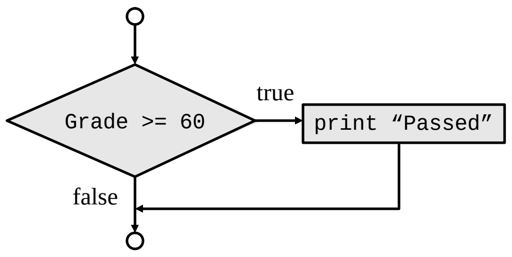
`if` Statement (cont'd)



```
if (studentGrade >= 60)
  Console.WriteLine ("Passed");

// beginning of the next statement
```

12

`if` Statement (cont'd)



```
if (studentGrade >= 60)
{
  Console.WriteLine ("Passed");
}

// beginning of the next statement
```

13

`if/else` Statement

```
if ( <test> )
   <code executed if <test> is true> ;
else
   <code executed if <test> is false> ;
```

- The **`if/else`** structure
  - Alternate courses can be taken when the statement is false
  - Rather than one action there are two choices
  - Nested structures can test many cases
  - See `Wage.cs`

14

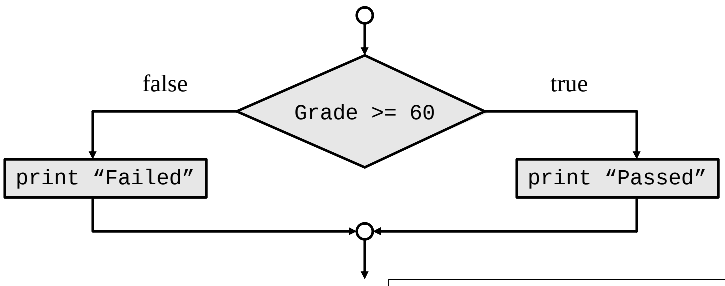
`if/else` Statement (cont'd)

```
if ( <test> )
{
   <code executed if <test> is true> ;
   ……
}
else
{
   <code executed if <test> is false> ;
   ……
}
```

❍ Can use the block statement inside either branch

15

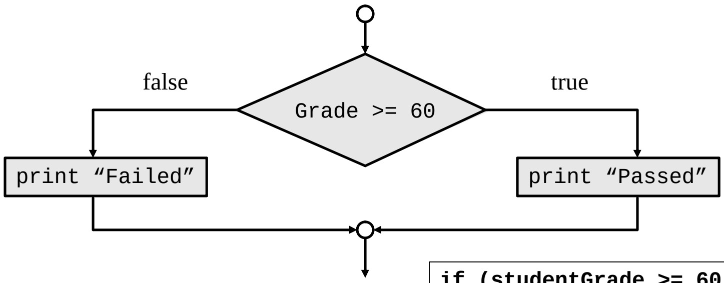
`if/else` Statement (cont'd)



```
if (studentGrade >= 60)
  Console.WriteLine ("Passed");
else
  Console.WriteLine ("Failed");
// beginning of the next statement
```

16

`if/else` Statement (cont'd)



```
if (studentGrade >= 60)
{
  Console.WriteLine ("Passed");
}
else
  Console.WriteLine ("Failed");
// beginning of the next statement
```

Nested `if/else` Statements

- The statement executed as a result of an if statement or else clause could be another if statement
- These are called *nested if /else statements*

```
if (studentGrade >= 90)
  Console.WriteLine("A");
else if (studentGrade >= 80)
  Console.WriteLine("B");
else if (studentGrade >= 70)
  Console.WriteLine("C");
else if (studentGrade >= 60)
  Console.WriteLine("D");
else
  Console.WriteLine("F");

// beginning of the next statement
```

Unbalanced `if-else` Statements

`if (favorite == “apple”)`
`   if (price <= 10 )`
`      Console.WriteLine(“10”);`
`   else`
`      Console.WriteLine(“1”);`
	`if (favorite == “apple”)`
	`   if (price <= 10 )`
	`      Console.WriteLine(“10”);`
	`else`
	`   Console.WriteLine(“not my favorite”);`

See IfElseMatch.cs

Rule: An else clause is matched to the last unmatched if (no matter what the indentation implies)

19

Ternary Conditional Operator (?:)

- Conditional Operator (**e_1?e_2:e_3**)
  - C#’s only ternary operator
  - Can be used to construct expressions
  - Similar to an **`if/else`** structure

```
string result;
int numQ;
…………
result = (numQ==1) ? “Quarter” : “Quarters”;

// beginning of the next statement
```

20

while Statement

- The *while statement* has the following syntax:

while is a reserved word

If the condition is true, the statement is executed. Then the condition is evaluated again.

```
while ( condition )
   statement;
```

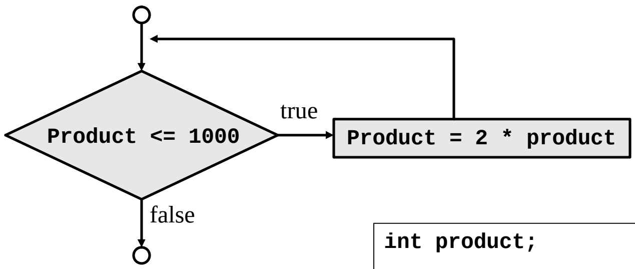
The statement (or a block of statements) is executed repetitively until the condition becomes false.

while Statement

```
while ( <test> )
{
   <code to be repeated if <test> is true>  ;
}
```

- Repetition Structure
  - An action is to be repeated
    - Continues while <test> is true
    - Ends when <test> is false
  - *Contains either a line or a body of code*

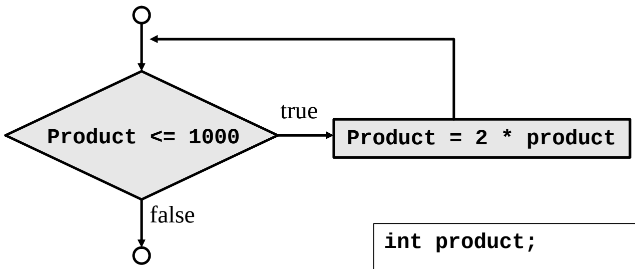
while Statement (cont'd)



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

// beginning of the next statement
```

23

while Statement (cont'd)



```
int product;
product = 2;
while (product <= 1000)
{
  product = 2 * product;
}
// beginning of the next statement
```

24

`while` Statement

- Note that if the condition of a while statement is false initially, the statement is never executed
- Therefore, the body of a while loop will execute zero or more times

25

Infinite Loops

- The body of a while loop must eventually make the condition false
- If not, it is an *infinite loop*, which will execute until the user interrupts the program
- See Forever.cs
- This is a common type of logical error
- You should always double check to ensure that your loops will terminate normally

26

Recap: Loop Statements

- *Loop statements* allow us to execute a statement multiple times
- They are often simply referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- C# has four kinds of loop statements: the *while loop*, the *do loop*, the *for loop, and* the *foreach loop*
- The programmer must choose the right kind of loop for the situation

27

Example 1: Counter Controlled `While` Loop

- Control variable
  - The variable used as a counter to determine whether or not the loop should continue
- Three components
  - Initial value of the counter
  - Check whether or not the counter has reached target
    - When the looping should continue
  - Incrementing/decrementing of the counter

28

Outline

WhileCounter.cs

```
// WhileCounter.cs
// Counter-controlled repetition.

using System;

class WhileCounter
{
   static void Main( string[] a
   {
      int counter = 1;          // init

      while ( counter <= 5 )    // repe
      {
         Console.WriteLine( counter );
         counter++;             // in

      } // end while

   } // end method Main

} // end class WhileCounter
```

This is where the counter variable is initialized. It is set to 1.

The loop will continue until counter is greater than five (it will stop once it gets to six)

The counter is incremented and 1 is added to it

Program Output

```
1
2
3
4
5
```

Computing Class Average

Problem: Get 10 grades from the user and compute the average.

```
Set total to zero
Set counter to one

While counter is less than or equal to ten
        Input the next grade
        Add the grade into the total
        Add one to counter

Set the class average to the total divided by ten
Print the class average
```

Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

Outline

Average1.cs

```
// Average1.cs
// Class average with counter-controlled repetition.

using System;

class Average1
{
   static void Main( string[] args )
   {
      int total,          // sum of grades
          gradeCounter,   // numbe                          d
          gradeValue,     // grade
          average;        // average of all grades

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

      // processing phase
      while ( gradeCounter <= 10 )  // loop 10 times
      {
         // prompt for input and read grade from user
         Console.Write( "Enter integer grade: " );

         // read input and convert to integer
         gradeValue = Int32.Parse( Consc

         // add gradeValue to total
         total = total + gradeValue;

         // add 1 to gradeCounter
         gradeCounter = gradeCounter + 1;
      }
```

Initialize total to 0

Initialize gradeCounter to 1

The **`while`** loop will loop through 10 times to get the grades of the 10 students

Prompt the user to enter a grade

Accumulate the total of the 10 grades

Add 1 to the counter so the loop will eventually end

Outline

Average1.cs

```

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

      // display average of exam grades
      Console.WriteLine( "\nClass average is {0}", average );

   } // end Main

} // end class Average1
```

Divide the total by ten to get the average of the ten grades

Display the results

Program Output

```
Enter integer grade: 100
Enter integer grade: 88
Enter integer grade: 93
Enter integer grade: 55
Enter integer grade: 68
Enter integer grade: 77
Enter integer grade: 83
Enter integer grade: 95
Enter integer grade: 73
Enter integer grade: 62

Class average is 79
```

Example 2: Sentinel Controlled `while` Loops

- ❒ This is typical of an input-driven program
- ❒ Continues an arbitrary amount of times
- ❒ Sentinel value
  - Causes loop to break
  - Avoid collisions
    - When flag value = user entered value

33

Sentinel Controlled Loop

```
Initialize total to zero
Initialize counter to zero

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)

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"
```

34

Outline

Average2.cs

```
// Average2.cs
// Class average with sentinel-controlled repetition.

using System;

class Average2
{
   static void Main( string[] args )
   {
      int total,           // sum of grades
          gradeCounter,    // number of grades ent
          gradeValue;      // grade value

      double average;      // average of all grades

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

      // processing phase
      // prompt for input and convert to integer
      Console.Write( "Enter Integer Grade, -1 to Quit: " );
      gradeValue = Int32.Parse( Console.ReadLine() );
```

The variable average is set to a double so that it can be more exact and have an answer with decimals

Variables gradeCounter and total are set to zero at the beginning

Get a value from the user and store it in gradeValue

Outline

```
      // loop until a -1 is entered by user
      while ( gradeValue != -1 )
      {
         // add gradeValue to total
         total = total + gradeValue;

         // add 1 to gradeCounter
         gradeCounter = gradeCounter + 1;

         // prompt for input and read grade from use
         // convert grade from string to integer
         Console.Write( "Enter Integer Grade, -1 to Quit: " );
         gradeValue = Int32.Parse( Console.ReadLine() );

      } // end while

      // termination phase
      if ( gradeCounter != 0 )
      {
         average = ( double ) total / gradeCounter;

         // display average of exam grades
         Console.WriteLine( "\nClass average is {0}", average
      }
      else
      {
         Console.WriteLine( "\nNo grades were entered" );
      }

   } // end method Main

} // end class Average2
```

Have the program loop as long

Accumulate the total of the grades

Add 1 to the counter in order to know the student count

Prompt the user for another grade, this time it is in the loop so it can happen repeatedly

Divide the total by the number of times the program looped to find the average

Display the average

Inform user if no grades were entered

```
Enter Integer Grade, -1 to Quit: 97
Enter Integer Grade, -1 to Quit: 88
Enter Integer Grade, -1 to Quit: 72
Enter Integer Grade, -1 to Quit: -1

Class average is 85.6666666666667
```

Outline

Average2.cs
Program Output

Loop-and-a-Half Idiom

```
Initialize total to zero
Initialize counter to zero

While (true)
    Input the first grade (possibly the sentinel)
    If  ( the user has entered the sentinel)
        break;

    Add this grade into the running total
    Add one to the grade counter

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"
```

38

Example 3: Program Condition

- ❒ Continuously change the value of a variable until some conditions are met.
- ❒ Example: ReverseNumber.cs

39