CGT 215 Lecture 4

Control Statements Part I

Algorithms

- Any computing problem can be solved by executing a series of actions in a specific order.
- A procedure for solving a problem in terms of (a) the actions to execute and (b) the order in which these actions execute is called an ***algorithm***.

Algorithm

- Rise and shine algorithm
  1. Get out of bed
  2. Take off PJ's
  3. Take a shower
  4. Get dressed
  5. Eat breakfast
  6. Carpool to work
- Again… ***actions*** to perform and the ***order*** in which they are performed.

Program Control

- Specifying the order in which statements (actions) execute in an application is called ***program control***.

- In these notes, we examine program control using ***control statements***.

Pseudocode

- ***Pseudocode*** is an informal language that helps programmers develop algorithms without having to worry about the strict details of actual language syntax.

- ***Pseudocode*** is similar to everyday English – it is convenient and user-friendly, but it is not an actual computer programming language.

Control Structures

- Normally, statements in an application are executed one after the other in the order in which they're written.
  - This process is called ***sequential execution***.
- Many statements enable you to specify that the next statement to execute is not necessarily the next one in the sequence.
  - This is called ***transfer of control***.

Control Structures (Avoid This)

- ***Spaghetti code***
  - The **goto** statement allows programmers to specify a transfer of control to one of a wide range of possible destinations in an application. Jump to anywhere at any time.
  - *Avoid using the goto statement.*

Selection Structures

- Also called ***selection statements***
- if
  - Referred to as a ***single-selection statement***
- if...else
  - Referred to as a ***double-selection statement***
- if…else if…else
  - Referred to as a ***multiple-selection statement***
- Switch
  - Referred to as a ***multiple-selection statement***

if single-selection statement

```
if(condition)
{
	//body of if selection statement
	//remember that it is case-sensitive – lowercase if
}

if(userID == “rjglotzbach”)
{
	Console.WriteLine(“Welcome Ron!”);
}

if(grade >= 60)
{
	lblFinalGrade.Text = “D”;
}
```

if single-selection statement

- Only a single statement in the body of an if:
  - if(grade >= 70)
    lblFinalGrade.Text = “C”;

- Multiple statements in the body of an if:
  - if(grade >= 80)
    {
    //you must use curly braces
    lblFinalGrade.Text = “B”;
    lblResult.Text = “You passed!”;
    }

if…else double-selection statement

```
if(condition)
{
	//body of if statement
	//remember that it is case-sensitive – lowercase if
}
else
{
	//body of if…else statement
	//there is no condition after the else
	//else is a catch-all – if none of the others are true, this code is executed.
}
```

if…else double-selection statement

- When there is only one statement after the if…else:

```
if(grade >= 60)
    Console.WriteLine(“Passed”);
else
    Console.WriteLine(“Failed”);
```

if…else double-selection statement

- When there are multiple statements after the if…else:

```
if(grade >= 60)
{
    //you must use curly braces
    lblFinalGrade.Text = “D”;
    lblResult.Text = “Passed”;
}
else
{
    //you must use curly braces
    lblFinalGrade.Text = “F”;
    lblResult.Text = “Failed”;
}
```

Ternary Conditional Operator

- Yet another alternative is the ternary operator

- The ***ternary conditional operator*** can be used in place of an if…else

Ternary Conditional Operator

- Basic form:
  - <condition> **?** <value if true> **:** <value if false>

- Example
  - lblResult.Text = (grade >= 60 **?** “Passed” **:** “Failed”);

Nested if statements

- Placing an if…else inside another if…else statement creates a ***nested if statement:***

```
if(x > 5)
{
        if(y > 5)
        {
                lblFoo.Text = “x and y are both 5”;
        }
        else
        {
                lblFoo.Text = “x is 5 but y is less than 5”;
        }
}
else
{
        lblFoo.Text = “x is less than 5”;
}
```

- **Tip: *Always* use curly braces if you are creating nested if statements.**

Good Programming Practice

- **Your curly braces should always line up *vertically*:**

```
if(x > 5)
{
    if(y > 4)
    {
        lblFoo.Text = “x and y are both 5”;
    }
    else
    {
        lblFoo.Text = “x is 5 but y is less than 5”;
    }
}
else
{
    lblFoo.Text = “x is less than 5”;
}
```

if... else if

```
if(condition)
{
	//body of if statement
	//remember that it is case-sensitive – lowercase if
}
else if(different condition)
{
	//body of conditional
	//else if is two words
}
```

if…else if… else

```
if(condition)
{
        //body of if statement
        //remember that it is case-sensitive – lowercase if
}
else if(different condition)
{
        //body of second conditional clause
        //else if is two words
}
else if(different condition)
{
        //body of third conditional clause
}
else
{
        //there is no condition after the else
        //else is a catch-all – if none of the others are true, this code is executed.
}
```

if…else if… else multiple-selection statement

- When there is only one statement in the body of the if statement:

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

if…else if… else multiple-selection statement

- When there are multiple statements in the body of the if statement:

```
if(grade >= 90)
{
        //you must use curly braces
        lblFinalGrade.Text = “A”;
        lblResult.Text = “Passed”;
}
else if(grade >= 80)
{
        lblFinalGrade.Text = “B”;
        lblResult.Text = “Passed”;
}
else if(grade >= 70)
{
        lblFinalGrade.Text = “C”;
        lblResult.Text = “Passed”;
}
else if(grade >= 60)
{
        lblFinalGrade.Text = “D”;
        lblResult.Text = “Passed”;
}
else
{
        lblFinalGrade.Text = “F”;
        lblResult.Text = “Failed”;
}
```

Dangling else problem

- If you write:

```
if(x > 5)
    if(y > 5)
        Console.WriteLine("x and y are both 5");
else
    Console.WriteLine("x is less than 5");
```

- Which **if** does the **else** belong to??
  - Beware: the **else** actually belongs to the second **if** *in this case*
- Again: *Always* use curly braces if you are creating nested if statements.

Errors

- Logic error
  - A ***logic error*** has its effect at execution (also called runtime)
- Fatal logic error
  - A ***fatal logic error*** causes an application to fail and terminate prematurely.
- Nonfatal logic error
  - A ***nonfatal logic error*** allows an application to continue executing, but causes it to produce incorrect results.

Repetition Control Structures

- Also referred to as ***repetition statements***
  - Enable applications to perform statements repeatedly, depending on the value of a ***loop-continuation condition***.
- while
- do...while
- for
- foreach

while repetition

- Executes repeatedly until a condition is met

```
int product = 3;

while(product <= 100)
    product = 3 * product;
```

- The result would be:
  - 9, 27, 81, 243
  - The loop body executes 4 times
  - The loop terminates when the product equals 243

counter-controlled repetition

- Use a counter to control when to exit the loop

```
numStudents = 25;
counter = 1;

while(counter <= numStudents)
{
    tbProgress.Text += “You are on student number ” + counter.ToString() + “\r\n”;
    counter++; //if you forget this, infinite loop
}
```

- Outputs:
  - You are on student number 1
    You are on student number 2
    …
    You are on student number 25

sentinel-controlled repetition

- Loop until the user enters a specific value

```
while(grade != -1)
{
    Console.Write(“Enter grade: ”);
    grade = Convert.ToInt32(Console.ReadLine());
}
```

- When the user types -1 the loop will stop, otherwise it will loop infinitely.

nested control repetition

```
numStudents = 25;
counter = 1;

while(counter <= numStudents)
{
        if(grade >= 90)
        {
                lblFinalGrade.Text = “A”;
                lblResult.Text = “Passed”;
        }
        else if(grade >= 80)
        {
                lblFinalGrade.Text = “B”;
                lblResult.Text = “Passed”;
        }
        else if(grade >= 70)
        {
                lblFinalGrade.Text = “C”;
                lblResult.Text = “Passed”;
        }
        else if(grade >= 60)
        {
                lblFinalGrade.Text = “D”;
                lblResult.Text = “Passed”;
        }
        else
        {
                lblFinalGrade.Text = “F”;
                lblResult.Text = “Failed”;
        }

        tbProgress.Text += “You are on student number ” + counter.ToString() + “\r\n”;
        counter++; //if you forget this, infinite loop
}
```

Compound Assignment Operators

Assignment Operator	Sample Expression	Explanation	Assigns…
Assume: int c=3, d=5, e=4, f=6, g=12;			
+=	`c += 7`	`c = c + 7`	10 to c
-=	`d -= 4`	`d = d - 4`	1 to d
*=	`e *= 5`	`e = e * 5`	20 to e
/=	`f /= 3`	`f = f / 3`	2 to f
%=	`g %= 9`	`g = g % 9`	3 to g

Conversions

- Explicit conversion
  - Using a ***cast*** operator
  - string num = “3”;
    x = (int)num;
    //this is called ***casting*** a string to an int
- Implicit conversion (or promotion)
  - C# performs promotion is *selected* cases
  - If you have a double and an int in an equation, C# will implicitly promote the int to a double so that the operation can be performed.
    int x = 3;
    double y = 4.2;
    double avg;
    avg = (x + y) / 2;