CGT 215 Lecture 5

Control Statements Part II

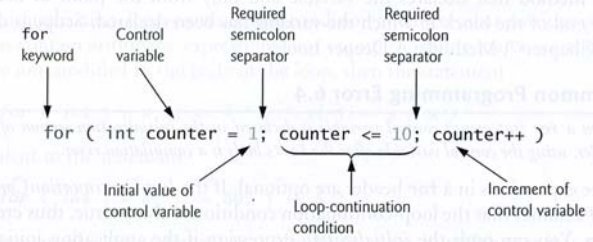
Counter-Controlled Repetition

- Essentials of counter-controlled repetition
  - A ***control variable*** (or loop counter)
  - The ***initial value*** of the control variable
  - The ***increment*** (or ***decrement***) by which the control variable is modified each time through the loop (also known as each iteration of the loop)
  - The ***loop-continuation condition*** that determines whether to continue looping.

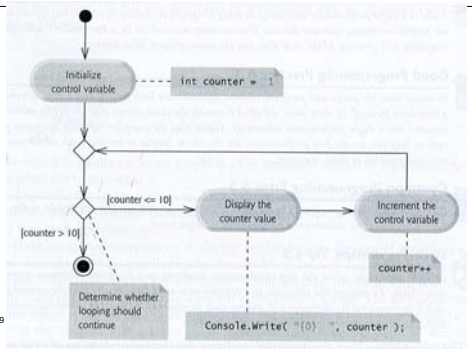
for

- for Repetition Statement
  - The for repetition statement specifies the elements of counter-controlled-repetition in a single line of code.
  - In general, counter-controlled repetition should be implemented with a for statementk

for header components



for statement – activity diagram



for – example 1

```
for ( int counter = 1; counter <= 10; counter++ )
      Console.Write( "{0}  ", counter );

//Or also written:

for ( int counter = 1; counter <= 10; counter++ )
{
      Console.Write( "{0}  ", counter );
}
```

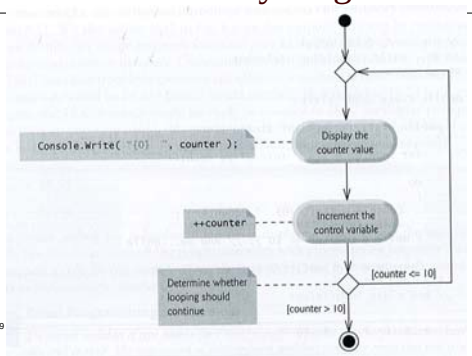
for – example 2

```
int total = 0;    // initialize total

// total even integers from 2 through 20
for ( int number = 2; number <= 20; number += 2 )
        total += number;

// display results
Console.WriteLine( "Sum is {0}", total );
```


do…while repetition statement

- The do…while repetition statement is similar to the while statement, however:
  - In the while statement, the loop-continuation condition is evaluated ***before*** the body of the loop executes.
  - In the do…while statement, the loop-continuation condition is evaluated ***after*** the body of the loop is executed.
  - Thus, the body of a do…while loop ***always executes at least one time***.

do…while – activity diagram



do…while – example

```
int counter = 1;    // initialize counter

do
{
        Console.Write( "{0}  ", counter );
        ++counter;
} while ( counter <= 10 );       // end do...while
```


switch statement

- Multiple-selection statement
- Performs different actions based on the possible values of an expression
- Each action is associated with the value of a ***constant integral expression*** or a ***constant string expression***.

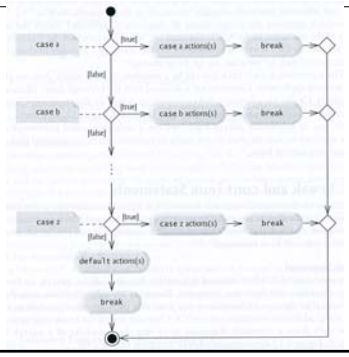
Constant integral expression

- Any expression involving character and integer constants that evaluates to an integer value
- i.e., values of type sbyte, byte, short, ushort, int, uint, long, ulong, char, or a constant from an enum type.

Constant string expression

- Any expression composed of string literals that always results in the same string

switch statement – activity diagram



switch statement

- switch… case is an alternative to using if…else

```
switch(find)
{
    case ‘a’:
        Console.WriteLine(“Regular Customer”);
        break;
    case ‘b’:
        Console.WriteLine(“Preferred Customer”);
        break;
    case ‘c’:
        Console.WriteLine(“Donor (monetary or organ... unsure which)”);
        break;
    default:
        Console.WriteLine(“We don’t want their business…”);
        break;
}
```

  - ***without **break** statements, **every** case will execute

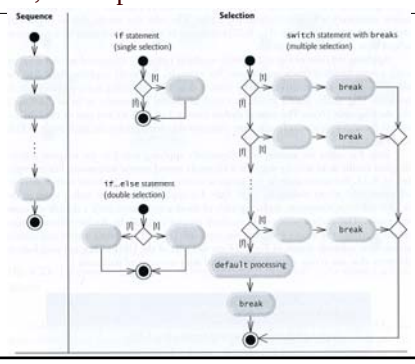
switch statement – example 1

```
// determine which grade was entered
switch ( grade / 10 )
{
         case 9:        // grade was in the 90s
         case 10:       // grade was 100
            ++aCount;   // increment aCount
            break;      // necessary to exit switch
         case 8:        // grade was between 80 and 89
            ++bCount;   // increment bCount
            break;      // exit switch
         case 7:        // grade was between 70 and 79
            ++cCount;   // increment cCount
            break;      // exit switch
         case 6:        // grade was between 60 and 69
            ++dCount;   // increment dCount
            break;      // exit switch
         default:       // grade was less than 60
            ++fCount;   // increment fCount
            break;      // exit switch
} // end switch
```

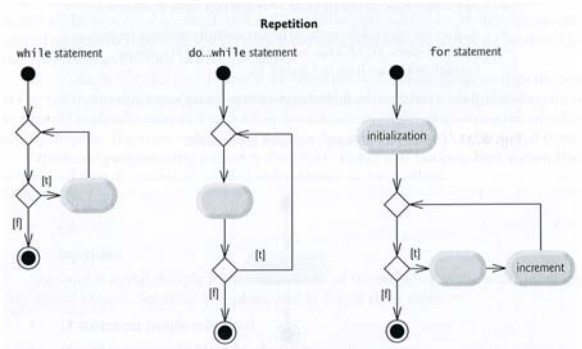

switch statement – example 2

```
test = "foo";

switch (test)
{
   case "apple":
      tb1.Text = "it's an apple!";
      break;
   case "orange":
      tb1.Text = "it's an orange!";
      break;
   case "foo":
      tb1.Text = "it's a foo!";
      break;
   default:
      tb1.Text = "it's not a $%#& thing!";
      break;
}
```


C#’s single-entry / single-exit sequence, selection, and repetition statements



C#'s single-entry / single-exit sequence, selection, and repetition statements



CGT 215 Lecture 5

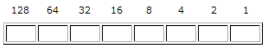
Logic

Binary numbers

- Binary numbers are made up of 0 and 1.
- An example of a binary number would look like: 10010111
  - This is an example of an 8-bit binary number.
- A 16-bit binary number would look like: 1001001011011001
- The decimal value of 10010111 is not 10,010,111.
  - It is actually 151 as a base 10 numeric value.

How is it calculated?

- Binary numbers count from ***right to left***.
- Each digit to the left is twice the value of its digit to the right.
- A graphical representation of this would look like this:



- Hence, a 1 in the 128 box gives the binary number a decimal value of at least 128.

Some examples

Binary Number		Decimal Value
10000000	=	128
10000001	=	129
00000000	=	0
00000001	=	1
00000010	=	2
00000011	=	3
00000100	=	4
00000101	=	5
11111111	=	255

- Thus, 0 to 255 offers 256 values within an 8-bit binary number.

Logical Operators

- Enables you to form more complex conditions by combining simple conditions
- The logical operators are:
  - `&&` (conditional AND)
  - `||` (conditional OR)
  - `&` (boolean logical AND)
  - `|` (boolean logical inclusive OR)
  - `^` (boolean logical exclusive OR)
  - `!` (logical negation)

Conditional AND &&

- if((gender == “F”) && (age >= 65))
  seniorFemales++;

Expression 1	Expression 2	Expression 1 && Expression 2
false	false	false
false	true	false
true	false	false
true	true	true

Conditional OR ||

- if((semesterAvg >= 90) || (finalExam >= 90))
  Console.WriteLine(“Student got an A”);

Expression 1	Expression 2	Expression 1 && Expression 2
false	false	false
false	true	true
true	false	true
true	true	true

Boolean logical AND &

- Works identically to the && operator, with one exception – the & always evaluates both of the operands. For example:
  - (gender == “F”) & (age >= 65)
  - Evaluates (age >= 65) regardless of whether gender is equal to “F”

Boolean logical inclusive OR |

- Works identically to the || operator, with one exception – the | always evaluates both of the operands. For example:
  - (birthday == true) | (++age >= 65)
  - Evaluates (++age >= 65) even if birthday is true, ensuring that age would be incremented.

Boolean logical exclusive OR ^

- Also called the ***logical XOR***
- is true *if and only if one of its operands is true and the other is false*.

Expression 1	Expression 2	Expression 1 && Expression 2
false	false	false
false	true	true
true	false	true
true	true	false

Logical negation !

- Enables you to reverse the meaning of a condition.
- Logical negation is a unary operator (only has one operand)
- Placed before a condition
- if(!(grade == -1))
  Console.WriteLine(“The next value is: ”);

Logical negation

- !true is the same as writing false
- !false is the same as writing true
- if(!(grade >= 60))
  Console.WriteLine(“Get a tutor”);