While Loops and Increment Operators

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Increment and Decrement Operators
- While Loop Basics
- Loops and User Input

Shortcuts for Changing Variables

- Multiple ways to add 1 to a numeric variable:

```
int myVar = 1;
myVar = myVar + 1;
myVar += 1;
```

Now `myVar` is 2 → `myVar = myVar + 1;`

Now `myVar` is 3 → `myVar += 1;`

- Increment operator, ++, also adds 1 to a variable:

```
myVar++;
++myVar;
```

Now `myVar` is 4 → `myVar++;`

Now `myVar` is 5 → `++myVar;`

- **Postfix** increment: `myVar++` ; **Prefix** increment: `++myVar`

Decrement Operator

- Multiple ways to subtract 1 from a numeric variable:

```
                 int myVar = 10;
Now myVar is 9 → myVar = myVar – 1;
                 myVar -= 1;
Now myVar is 7 → myVar--;
Now myVar is 6 → --myVar;
```

- **Postfix** decrement: `myVar--` ; **Prefix** decrement: `--myVar`

	Increment	Decrement
Postfix	`myVar++`	`myVar--`
Prefix	`++myVar`	`--myVar`

Prefix vs. Postfix

- Both versions have same effect on variable: add/subtract 1
- Difference is which value is "returned" by the expression

Postfix Increment/Decrement

- Return value, *then* increment
- Value of expression is *original* value of variable, before increment

a is now 2

```
int a = 1;
Console.WriteLine(a++);
Console.WriteLine(a--);
```

```
1
2
```

Prefix Increment/Decrement

- Increment, *then* return value
- Value of expression is *new* value of variable, after increment

a is now 2

```
int a = 1;
Console.WriteLine(++a);
Console.WriteLine(--a);
```

```
2
1
```

Increment Operators in Expressions

- Increment/decrement operators have higher precedence than other math operators
- Value used in expression depends on prefix vs. postfix

```
int a = 1;
int b = a++;
int c = ++a * 2 + 4;
int d = a-- + 1;
```

b gets 1, then a is incremented to 2

$$++a * 2 + 4$$

$$\downarrow$$

$$\text{Now a} = 3 \longleftarrow \underbrace{3 \quad * 2 + 4}_{10 \longleftarrow c = 10}$$

$$\text{Now a} = 2 \longleftarrow \underbrace{3 \quad + 1}_{4 \longleftarrow d = 4}$$

AUGUSTA UNIVERSITY

Outline

- Increment and Decrement Operators
- **While Loop Basics**
- Loops and User Input

Repeating Code

- `while` statement: Execute code block repeatedly, as long as a condition is **true**
  - Or: Execute code repeatedly, *until* the condition is **false**

```
int counter = 0;
while(counter <= 3)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(counter);
  counter++;
}
Console.WriteLine("Done");
```

Condition

```
Hello again!
0
Hello again!
1
Hello again!
2
Hello again!
3
Done
```

While Loop Rules

- Condition is evaluated first to produce a `bool`
- If `false`, loop block is skipped
- If `true`, loop block is executed
- After executing loop block, go back to `while` statement, evaluate condition again
- Curly braces can be omitted if loop block is just **one** statement

```
while(<condition>)
{
  <statements>
}
```

```
while(<condition>)
  <statement>
```

While Loop in Detail

- First time: `counter` is `0`, so execute the loop block
- At end of loop block, evaluate `counter <= 3` again
  - `counter` is 1, so execute the loop block again
- Last iteration: after printing "3", increment `counter` to `4`
- Now `counter <= 3` is false, so skip the loop block

```
int counter = 0;
while(counter <= 3)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(counter);
  counter++;
}
Console.WriteLine("Done");
```

AUGUSTA UNIVERSITY

Initial Evaluation

- While loops may execute zero times!

```
int counter = 5;
while(counter <= 3)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(counter);
  counter++;
}
Console.WriteLine("Done");
```

```
Done
```

- Just like `if`, code block is skipped if condition is false

Ending the Loop

- Statements in loop body **must** change a variable in the condition
- Otherwise the program will never end!

```
int counter = 0;
while(counter <= 3)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(counter);
}
Console.WriteLine("Done");
```

Loop condition uses counter

Loop body never changes counter

```
Hello again!
0
Hello again!
0
Hello again!
0
Hello again!
0
...
```

Loop continues forever because counter is always ≤ 3

Other Ways to Write Infinite Loops

- Changing a different variable, not the one in the condition

```
int num1 = 0, num2 = 0;
while(num1 <= 5)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(num1);
  num2++;
}
```

num2 isn't in the loop condition

- Changing the variable in the wrong "direction" for the condition

```
int number = 10;
while(number >= 0)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(number);
  number++;
}
```

Loop ends when number is negative

Need to decrement number, not increment

Writing a While Loop

- Questions to ask when writing a while loop:

1. When (under what condition) do I want the loop to continue?
2. When (under what condition) do I want the loop to stop?
3. How will the body of the loop bring it closer to its ending condition?

- Write a loop condition that will be `true` in circumstances described by (1), and `false` in circumstances described by (2)

Outline

- Increment and Decrement Operators
- While Loop Basics
- **Loops and User Input**

Ensuring Input is Valid

- Data in a program might have limited "good" or "valid" values
  - Example: `price` attribute of `Item` class should be positive
- What if the user provides a "bad" value as input?

```
Console.WriteLine("Enter the item's price.");
decimal price = decimal.Parse(Console.ReadLine());
Item myItem = new Item(desc, price);
```

```
public Item(string initDesc, decimal initPrice)
{
  description = initDesc;
  price = (initPrice >= 0) ? initPrice : 0;
}
```

Ignore invalid values

Ensuring Input is Valid

- Another approach: Ask user to re-enter data until it is valid

```
Console.WriteLine("Enter the item's price.");
decimal price = decimal.Parse(Console.ReadLine());
while(price < 0)
{
  Console.WriteLine("Invalid price. Please enter"
    + " a non-negative price.");
  price = decimal.Parse(Console.ReadLine());
}
Item myItem = new Item(desc, price);
```

Skips the block if `price` is already valid

By this point, `price < 0` must be false

String Parsing Errors

- When asked for a number, the user might not enter a number
- `int.Parse()` assumes the string is a valid number

```
Console.WriteLine("Guess a number.");
int guess = int.Parse(Console.ReadLine());
if(guess == favoriteNumber)
{
  Console.WriteLine("That's my favorite number!");
}
```

- Current behavior: Program crashes if user enters “hello”

AUGUSTA UNIVERSITY

The TryParse Method

- Indicates failure by returning `false`, not crashing
- Result of parsing is assigned to "out parameter," not method's return value

Keyword `out`: indicates a "parameter" that is used for **output**

```
string userInput = Console.ReadLine();
int intVar;
bool success = int.TryParse(userInput, out intVar);
```

Return value is `true` if parsing succeeded, `false` if it failed

Result of string conversion assigned to this variable

Using TryParse

```
Console.WriteLine("Please enter an integer");
string userInput = Console.ReadLine();
int intVar;
bool success = int.TryParse(userInput, out intVar);
if(success)
{
  Console.WriteLine($"The value entered was an integer: {intVar}");
}
else
{
  Console.WriteLine($"\"{userInput}\" was not an integer");
}
Console.WriteLine(intVar);
```

`intVar` is now the parsed integer

TryParse failed, so it returned `false`

Even if TryParse failed, `intVar` still has a value: `0`

Controlling a Loop with User Input

- Loops aren't always for validation
- User input can indicate when the loop should be done

```
Console.WriteLine("Enter a string.");
string input = Console.ReadLine();
while(input != "quit")
{
  Console.WriteLine($"Your string was: {input}");
  Console.WriteLine("Enter another string, "
    + "or enter \"quit\" to quit.");
  input = Console.ReadLine();
}
```

AUGUSTA UNIVERSITY

Summary

- Increment and Decrement Operators
- While Loop Basics
- Loops and User Input