Reading Input and Displaying Output

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Converting numbers to strings
  - The ToString method
- String concatenation
- Reading user input
- Parsing user input

String Interpolation

- Casting won't convert numbers to `strings`, interpolation will

```
int x = 47, y = 6;
double fraction = (double) x / y;
string text = $"{x} divided by {y} is {fraction}";
```

"47" "6" "7.8333333"

- Interpolation can convert any expression, not just a variable

"7.8333333"

```
Console.WriteLine($"{x} divided by {y} is {(double) x / y}");
Console.WriteLine($"{x} plus 7 is {x + 7}");
```

"54"

This does **not** change the value of x or y:

```
Console.WriteLine($"x={x}, y={y}");
```

Prints "x=47, y=6"

Behind the Scenes

- Interpolation doesn't know how to convert numbers to strings
- All data types in C# are objects, even `int`
- All objects in C# have a `ToString()` method – "convert this object to a `string`"

```
int num = 42;
double fraction = 33.5;
string intText = num.ToString();
string fracText = fraction.ToString();
```

Call method `ToString` on object `num`

Result: "42"

Result: "33.5"

Call method `ToString` on object `fraction`

Behind the Scenes

- Interpolation calls `ToString()` on the result of each expression

```
int num = 42;
string intText = num.ToString();
```

This:

```
Console.WriteLine($"num is {num}");
```

Is the same as:

```
Console.WriteLine($"num is {intText}");
```

Is the same as:

```
Console.WriteLine($"num is {num.ToString()}");
```

Outline

- Converting numbers to strings
  - The ToString method
- **String concatenation**
- Reading user input
- Parsing user input

String Concatenation

- The + operator is a different function for each operand type
- If the operand types are `string`, it performs **concatenation**

```
string greeting = "Hi there, " + "John";
```

Result: "Hi there, John"

`string + string` operator

- Works as expected for `string` variables:

```
string name = "Paul";
string greeting2 = "Hi there, " + name;
```

Result: "Hi there, Paul"

Mixed Types with +

- What if we use + operator with a string and a number?

```
int bananas = 42;
string text = "Bananas: " + bananas;
```

- Answer: Converts the other argument to a string with ToString()

```
string text = "Bananas: " + bananas;
```

bananas.ToString()

"Bananas: " + "42" → "Bananas: 42"

string + string, concatenation

Interpolation and Concatenation

- Can "print out" variables two equivalent ways:

```
int num = 42;
Console.WriteLine($"num is {num}");
Console.WriteLine("num is " + num);
```

- Interpolation is easier to write with many variables:

```
Console.WriteLine($"The variables are {a}, {b}, {c}, {d}, and {e}");
Console.WriteLine("The variables are " + a + ", " + b + ", "
     + c + ", " + d + ", and " + e);
```

Concatenation Puzzle

- Code executes left-to-right, binary operators (like +) are grouped left-to-right
- What does this produce?

int + int → int + string

```
int var1 = 6, var2 = 7;
Console.WriteLine(var1 + var2 + " is the result");
Console.WriteLine("The result is " + var1 + var2);
```

"13 is the result"

"The result is 67"

string + int → string + int

Parentheses Define Order

- Parentheses make order, grouping explicit
- Use to ensure compiler does what you mean

```
int var1 = 6, var2 = 7;
Console.WriteLine((var1 + var2) + " is the result");
Console.WriteLine("The result is " + (var1 + var2));
```

"13 is the result"

"The result is 13"

Evaluates second, concatenates string with result of parentheses

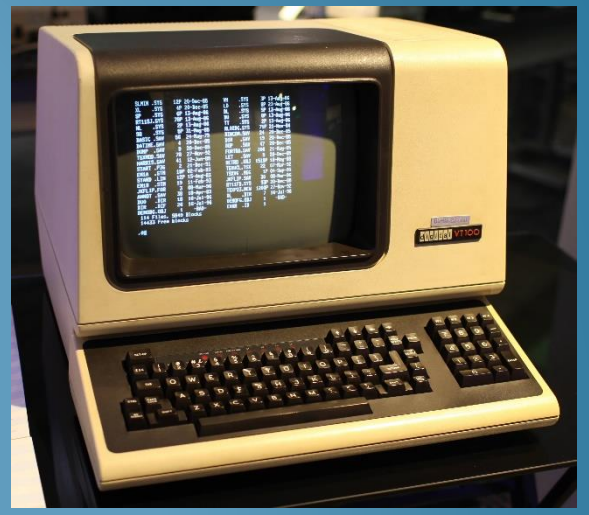
Evaluates first, so + operator sees `int + int`

Outline

- Converting numbers to strings
  - The ToString method
- String concatenation
- **Reading user input**
- Parsing user input

Input from the User

- With a CLI, output = print text to screen, input = read text from keyboard
- `Console` represents the "terminal" interface



Output:

Text to output

```
Console.WriteLine("Hi!");
```

Result: Terminal displays "Hi!"

Input:

Waits until user presses "Enter"

```
Console.ReadLine();
```

Result: A `string` value containing text typed in at terminal (one line)

Lab Recap

```
class PersonalizedWelcomeMessage
{
  static void Main()
  {
    string firstName;
    Console.WriteLine("Enter your first name:");.
    firstName = Console.ReadLine();
    Console.WriteLine($"Welcome, {firstName}!");
  }
}
```

Declare a string variable named firstName

Print text to the console

Wait for a line of text to be typed in

Insert the text from firstName into this string and print it

Assign the text received from the console to firstName

Outline

- Converting numbers to strings
  - The ToString method
- String concatenation
- Reading user input
- **Parsing user input**

The Opposite of ToString()

- Recall: Converting `int` to `string` requires a method

```
int myAge = 29;
string strAge = (string) myAge;
```

Error! Can't convert `int` to string

```
string strAge = myAge.ToString();
```

- Similarly, converting string to int uses a method:

```
string strAge = "29";
int myAge = (int) strAge;
```

Error! Can't convert `string` to `int`

```
int myAge = int.Parse(strAge);
```

Parse Methods

- Each built-in numeric type has a "Parse" method

```
int.Parse("42") ⟶ 42          double.Parse("3.65") ⟶ 3.65

long.Parse("42") ⟶ 42L        float.Parse("3.65") ⟶ 3.65f
```

- Useful for turning user input into data:

```
Console.WriteLine("Please enter the year.");
string userInput = Console.ReadLine();
int curYear = int.Parse(userInput);
Console.WriteLine("Next year it will be {curYear + 1}");
```

What Could Go Wrong?

- String is not a number: program crashes with `FormatException`

```
int badIdea = int.Parse("Hello!");
```

→ `FormatException!`

- String is a number that can't fit the desired type: program crashes with `OverflowException`

```
int fail1 = int.Parse("52.5");
int fail2 = int.Parse("3000000000");
```

→ `OverflowException!`

→ `OverflowException!`

Summary

- Converting numbers to strings
  - The ToString method
- String concatenation
- Reading user input
- Parsing user input