Can you figure them out?

- 26 L of the A (26 Letters of the Alphabet)
- 5 T on a F
- 90 D in a R A
- 3 B M (S H T R)
- 23 P of C in the H B

Literals
Constants
Arithmetic Operators

Luke Sathrum – CSCI 20

Today’s Class

- Assignment Compatibility
- Literals
- Escape Sequences
- Naming Literals
- Arithmetic Operators
- Types and Arithmetic Operators
- Integer and Floating Point Division

Assignment Compatibility
Literals

Assignment Compatibility

- You cannot store one type in a variable of another type
  - `int int_variable = 2.99;`
- Called a **type mismatch** because **`2.99`** is?
- Compilers will assign the value **`2`** to **`int_variable`**
- You can assign an integer value to a floating-point type
  - `double double_variable = 2;`
  - Sets **`double_variable`** `to` **`2.0`**

Assignment Compatibility

- RULE:
  - You cannot place a value of one type in a variable of another type

Literals

- **Literal** is another name for one specific value
- These do not change values
- Example
    - `1`, `3`, `5`, `8.8`, `72`, `'A'`, `'B'`, `100.5`

Literals - **`double`**

- Can be written in scientific notation
  - `3.67 x 10`^{17} = 367000000000000000.0
    - Can write in C++ as **`3.67e17`**
  - 5.89 x 10^{-6}
    - In C++ **`5.89e-6`**

Literals – `char`, `string`, `bool`

- We express characters in single quotes
  - `char my_letter = 'a';`
- We express strings in double quotes
  - `string my_word = "This is a Word.";`
- We express booleans in true / false
  - `bool flag = false;`

Literal Summary

- Literal is a name for one specific value
- Literal values can only be stored in their associated types
- We denote characters with single quotes
- We denote strings with double quotes

Sample Code

- Type Compatibility and Mismatch
  - `compatibility.cpp`

Escape Sequences
Constants

Escape Sequences

- There are characters we need to use in strings and that aren’t easy to type
- Things like
  - New Line
  - Carriage Return
  - Tab
- There are also characters that already mean something
  - Double Quotes
  - Single Quotes

Escape Sequences

- We have **escape sequences** in order to use these characters
  - Use the backslash to start the sequence (****)

Sequence	Meaning
`\n`	New Line
`\r`	Carriage return
`\t`	(Horizontal) Tab
`\a`	Alert
`\\`	Backslash
`\'`	Single quote (used with **`char`**)
`\"`	Double quote (used with `string`)

Naming Literals

- We can name numbers so that if we need to change them across our whole program it will be easy
- We call these **constants**
- Example
  - Bank Branches and Bank Teller Windows

```
const int kBranchCount = 3;
const int kWindowsCount = 3;
```

- Can update either and recompile

Naming Constants – `const` Modifier

- `const` modifier makes it so we cannot change the value of the variable
- Given

```
const int kBranchCount = 10;
```

- This would fail

```
kBranchCount = 9;
```

- Notice how constant are named stating with a k
- Also don’t have an underscore and are mixed case
  - Part of our Naming Convention

Escape Sequences and Constants Summary

- We use escape sequences for certain characters
- We can name literals by creating constants

Sample Code

- Escape Sequences and Constants
  - `escape_constants.cpp`

Arithmetic Operators
Integer Division

Arithmetic Operators

- **Arithmetic Operators** allow us to do arithmetic (math) operations

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo	%

Arithmetic Operators

- The operators can be used with various types
  - Integers
    - `int + int = int`
  - Floating-Point
    - `double + double = double`
  - Intermixed
    - `double + int = double`

Arithmetic Expressions

- There is an order of Precedence
- PEMDAS
  - 4 + 6 * 3 = ?
- Can use parentheses to give different order
  - (4 + 6) * 3 = ???
- Good practice to use parentheses for readability
  - We write

    4 + 6 * 3
  - As

    4 + (6 * 3)

Division in C++

- Floating Point Division Acts as expected
  - 6.1 / 3.2 = 1.90625
- Integer Division
  - Only returns the integer part of the result
  - 10 / 3 = 3 (NOT 3.333)
  - 5 / 2 = 2 (NOT 2.5)
- We can achieve a floating point result if we use at least **a** floating point number on either side of the operator
  - 10 / 3.0 = 3.333
  - 5 / 2.0 = 2.5

Division in C++

- Mod %
  - Will give you the part that is lost in integer division
  - Think back to long division in grade school
    - 17 / 5 = ?
  - Integer division gives you
    - 3
  - Mod division gives you
    - 2
- Negative Integers
  - Don’t do it

Dividing Whole Numbers - Example

- Let’s say you are a landscaping architect
- You charge $5000 per mile
- You work on 2112 feet of highway
- There are 5280 feet in a mile
- Formula: `total_price = 5000 * (feet / 5280)`
- How much did you charge?

```
int feet;
double total_price;
feet = 2112;
total_price = 5000 * (feet / 5280);
```

Arithmetic Summary

- We have operators to do arithmetic operations
- There is an order of precedence to these operators
- Be careful of integer division

Sample Code

- Arithmetic Operators and Division
  - `arith_division.cpp`

Review

- Assignment Compatibility
- Literals
- Escape Sequences
- Constants
- Arithmetic Operators
  - Between Different Types
- Division in C++