Scrambled Cities

- AKNSSA ITCY
- REEDVN
- ITSUAN
- TEEATSL
- LE OASP
- LUULOONH
- WNE ALROESN
- SNA TANNOOI
- KANSAS CITY
- DENVER
- AUSTIN
- SEATTLE
- EL PASO
- HONOLULU
- NEW ORLEANS
- SAN ANTONIO

CinReader
Code Style
Boolean Expressions

Luke Sathrum – CSCI 20

Today's Class

- Console Input with `CinReader`
- Code Style
- Define a Boolean Expression
- Comparison Operators
- Logical Operators

CinReader

Problems with Input

- Sometimes people input the wrong literal when we use **`cin`**
- Such as
  - `int num;`
  - `cin >> num;`
  - The user inputs `'Y'`
  - To handle this we have a class that we will use in all of our programs called `CinReader`
    - Created by Boyd Trolinger

CinReader

- There are two files for CinReader
  - CinReader.cpp
  - CinReader.h
- You will need to add these two files to your program

`CinReader` - Using

- After adding the files you need to include `CinReader`

```
#include "CinReader.h"
```

- You then need to create an instance of `CinReader` inside **`main()`**

```
CinReader reader;
```

Member Functions – `readInt()`

- **`readInt()`**
- This will get an Integer from the user.
- Example

```
int int_var;
int_var = reader.readInt();
```

- We can also tell it what range we want the numbers to be

```
int_var = reader.readInt(1, 10);
```

  - Number must be **greater than or equal to** 1 and **less than or equal to** 10

Member Functions – `readDouble()`

- Reads a Double from the user
- Example

```
double double_var;
double_var = reader.readDouble();
```

Member Functions – `readBool()`

- Read a Boolean from the user
- Example
  - `bool bool_var;`
  - `bool_var = reader.readBool();`
- It will accept the following inputs (case-insensitive)
  - `T`
  - `F`
  - `True`
  - `False`

Member Functions – `readChar()`

- Reads a single character from the user
- Example
  - `char char_var;`
  - `char_var = reader.readChar()`
- We can also limit our characters (case-sensitive)
  - `char_var = reader.readChar("abcdef")`
  - This limits it to only lowercase **a**, **b**, **c**, **d**, **e**, **f**

Member Functions – `readString()`

- Gets a string from the user
- Example
  - `string string_var = reader.readString();`
- We can limit by
  - Not allowing an empty string `""`
    - `string_var = reader.readString(false);`
  - Allowing only a certain length
    - `string_var = reader.readString(true, 5);`
    - This limits to 5 characters, also allowing the empty string `""`

CinReader Summary

- `CinReader` makes console input easy
- There are lots of member functions to get certain types
  - Integers
  - Floating-Point Numbers
  - Booleans
  - Characters
  - Strings

Sample Code

- Example of input using `CinReader`
  - `cinReader_example.cpp`

Code Style

Code Style

- It is important to be consistent in the way we write our programs
- Helps us to read our own code
- Helps us to read other’s code
- Makes it easy to read
- Makes it easy to modify
- Encompasses all that we write

Comments

- We use code comments to describe our code
- Use `//` to add single line comments
- Use `/*` to start multiple line comments
- Use `*/` to end multiple line comments
- We DO NOT need to comment every line of code
  - To describe a block of code
  - When something is important
  - When something is not obvious to other users

Code Style

- We will be following Google's C++ Style Guide
  - Found here: https://google.github.io/styleguide/cppguide.html
  - Has rules for
  - Indentation
  - Spacing
  - Naming Variables
  - Naming Constants
  - Line Length

Code Style - Indentation

- Blocks of code are always indented
- Blocks start with a { and end with a }
- Indented with 2 spaces
  - Do NOT use tabs
- Example

```
int main() {
  //This is indented
}
```

Code Style - Spacing

- Whitespace is important for readability

```
// Open braces should always have a space before
them.
void f(bool b) {
// Semicolons usually have no space before them.
int i = 0;
// Spaces inside braces for braced-init-list are
int x[] = { 0 };
// optional.  If you use them, put them on both
sides!
int x[] = {0};
```

Code Style - Naming

- Give descriptive names
- Good ones

```
// No abbreviation.
int price_count_reader;
// "num" is a widespread convention.
int num_errors;
// Most people know what "DNS" stands for.
int num_dns_connections;
```

Code Style - Naming

- Bad Ones

```
// Meaningless.
int n;
// Ambiguous abbreviation.
int nerr;
// Ambiguous abbreviation.
int n_comp_conns;
// Only your group knows what this stands for.
int wgc_connections;
// Lots of things can be abbreviated "pc".
int pc_reader;
```

Code Style - Naming

- Filenames
  - Use underscores between words
  - `my_useful_program.cpp`
- Constants
  - Use a k followed by mixed case
  - `kDaysInAWeek`

Code Style - Naming

- Variables
  - All lowercase
  - Underscores between words
  - `my_exciting_local_variable`
  - `google_url`

Code Style - Formatting

- Line Length
  - Each line should be at most 80 characters long
- Spaces vs. Tabs
  - Use 2 spaces instead of 1 tab
- Return Values
  - Do not needlessly surround the `return` expression with parentheses

Summary

- We have code style standards
- These help us be consistent in the way we write
- Have standards for everything that we write
- Located here: https://google.github.io/styleguide/cppguide.html

Boolean Expressions

Boolean Expression

- A **Boolean Expression** is an expression that is either
  - True (`true`)
  - False (`false`)
- Simplest form is comparing numbers or variables
  - `1 > 5`
  - `cost < value`
  - `cost > 5`

Comparison Operators - Equality

Math Symbol	English	C++ Notation
=	Equal to	==
≠	Not Equal to	!=

- Examples

```
(x + 7) == (2 * y)
ans != 5
```

Comparison Operators – Less Than

Math Symbol	English	C++ Notation
<	Less Than	<
≤	Less Than or Equal to	<=

- Examples

```
count < (m + 3)
time <= limit
```

Comparison Operators – Greater Than

Math Symbol	English	C++ Notation
>	Greater Than	>
≥	Greater Than or Equal to	>=

- Examples

```
time > limit
age >= 21
```

What do they evaluate to?

- Assume `int num = 2;`

```
num == 2
2 > num
num < 2
8 == num
2 <= num
num = 2
```

Boolean Expression Summary

- Boolean Expressions evaluate to either `true` or `false`
- We have operators to handle our expressions
  - Don’t forget equality is ==

Building Boolean Expressions

Building Boolean Expressions

- We can combine 2 (or more) different Boolean expressions using 3 operators
  - AND (**&&**)
  - OR (**||**)
  - NOT (**!**)

BBE – AND (&&)

- **`true`** if both comparisons are **`true`**
- **`(2 < x) && (x < 7)`**
- **`true`** if ***both*** **`2 < x`** AND **`x < 7`**
- **`false`** if ***either*** part is **`false`**

BBE – AND (&&) - Example

```
(score > 0) && (score < 10)
```

- What does this evaluate to if score is 5?
- What if score is 10?

Strings of Inequalities

- In math we can write x < y < z
- In C++ we write as
  - `(x < y) && (y < z)`

BBE - OR (||)

- **`true`** if either comparison is **`true`**
- **`(x < 2) || (x > 7)`**
  - **`true`** if ***either*** **`x < 2`** ***or*** **`x > 7`**
  - **`false`** if ***both*** are **`false`**

BBE - OR (||) - Example

```
(score > 3) || (score <= 1)
```

- What does this evaluate to if score is 5?
- What if score is 1?
- What if score is 2?

BBE – NOT (!)

- Changes `true` to `false`
- Changes `false` to `true`

BBE – NOT (!) - Example

```
!(score < 4)
```

- What does this evaluate to if score is 5?
- What if score = 1?

Building Boolean Expressions Summary

- Boolean Expressions can be combined
- Combined with the logical operators
  - **&&**
  - **||**
  - **!**

Review

- CinReader
- Code Style
- Boolean Expressions