Scrambled Cities

- SNA SEJO
- OIAPER
- ACHIWTI
- XNOIEPH
- AAPTM
- ULTAS
- GACOHIC
- NAS GOIDE
- THOUSNO
- PROTLDAN

- San Jose
- Peoria
- Wichita
- Phoenix
- Tampa
- Tulsa
- Chicago
- San Diego
- Houston
- Portland

Scope

Luke Sathrum – CSCI 20

Today's Class

- Scope

Local Variables and Scope

Local Variables

- Functions should be self-contained
- Should not interfere with other functions
- Usually we give functions their own variables
  - These are different from other variables we declare
- We call these variables local variables
  - They are local to their function

Local Variables

- Think about predefined function **`sqrt()`**
  - Did you know what variables were used?
  - Could you use those variables in your program?
- The same applies to functions we define
- We can declare variables with the same name in both **`main()`** and our programmer defined function
  - These variables are considered different variables

Scope

- When we declare a variable in a function we consider that function its scope

```
double TotalCost(int num_param, double price_param){
  double subtotal;
}
```

- **num_param**'s scope?
- **subtotal** is what kind of variable?

Class Scope

- The same applies for a class member function

```
double MyClass::TotalCost(int num_param, double price_param){
  double subtotal;
}
```

- Member Functions and Member Variables also have scope
- Their scope is all of the member functions of the class.
- This means you can access them from any other?
  - Member Function

Sample Code

- Local Variables and Scope
  - `local.cpp`

Globals and Scope

Global Constants

- Remember
  - We name constant values with the const modifier
  - `const double kTaxRate = 0.05;`
  - We do this so that the value cannot change
- These constants are local to a function
- What if we want to use the constant in multiple functions?

Global Constants

- We can use a constant globally by declaring them at the beginning of our program
  - By global we mean anywhere in our program
  - We think of their scope as the whole program
- We declare this constant after the
  - `#includes`
  - `using` statements

Global Constants Example

- Let's say we want to use the constant π
- We want to use the value for the following functions
  - area = π x (radius)^2
  - volume = (4/3) π x (radius)^3
- To do this we place a declaration for π after the includes in our program
- `const double kPi = 3.14159;`

Global Variables

- We can define variables globally much like we define global constants
- We avoid doing this
  - Can make a program harder to understand
  - Can make a program harder to maintain

Sample Code

- Using a Global Constant
  - `global.cpp`

More on Scope

Blocks

- Blocks are lines of code inside of braces **{ }**
  - Sometimes we call these compound statements
- Variables declared in a block have a scope of that block
- Notice that the body declaration of a function is a block

Nested Scopes

- We can nest blocks inside of blocks
- If we declare variables with the same name inside of each block then we have different variables
  - One variable only exists inside the inner block
    - Cannot be accessed outside of that block
  - One variable only exists in the outer block
    - Cannot be accessed by the inner block
- Note
  - You can access a variable of the outside block in an inside block before a variable with the same name is declared in the inside block

Nested Scope

```
{
 int number = 1;
 {
      int number = 2;
 }
}
```

- We have 2 variables both named number
- They have different values

Nested Scope Example

```
{
   Location A
   int number = 1;
   Location B
  {
      Location C
      int number = 2;
      Location D
  }
   Location E
}
```

Variables Declared in a `for` loop

- When we declare a variable in a `for` loop that variables scope is the body of the `for` loop

```
for (int i = 1; i <= 10; i++) {
     loop_body;
     // i only exists inside this loop body
}
```

Sample Code

- More Scope Ideas
  - `advanced_scope.cpp`