Introduction to C++

Language basics – variables: user defined types

Classes and objects

- **class keyword to define a class**
  - Trailing ; trips up C# devs
- **private and public sections**
  - Not line by line
  - Default is private
  - Best practice: no public member variables
- **Declare an instance with same syntax as fundamental types**
- **Access member variables and functions with .**
  - Static members and functions with classname and ::

Scope

- **Objects created like this have a lifetime**
  - Constructor called when control reaches the line they're declared
  - Memory is allocated for them on the stack
  - Object goes out of scope – usually at a }
  - Memory is freed and destructor runs
- **Resource Acquisition is Initialization**
  - Acquire resources in the constructor
  - Release them in the destructor
  - Eg open/close file, database connection, change Windows cursor, …

Odds and Ends

- **struct**
  - Generally used for “plain old data” with little or no business logic
  - Can have member functions, constructor, destructor
  - Only difference: default access is public
- **Inheritance**
  - Key to OO design
  - Derived classes can add or override member variables and functions
- **Namespaces**
  - Prevent name collisions
  - Separate from class name with :: (eg std::string)
- **Enum**
  - Give names to a set of constants
  - Names must be unique

PreProcessor

- **Lines that start # are pre-processor directives**
  - #include
- **Can use to compile slightly different code under different circumstances**
  - E.g., “a debug build”
- **Can also use for convenience**
  - Include guards with #ifndef / #endif and #define
  - #pragma once

Summary

- **Declare instances of objects or fundamental types on the stack:**
  - int i=3;
  - Person p1("Kate", "Gregory",123);
  - Status s = Pending;
- **When the instance goes out of scope, the object is cleaned up**
  - Memory released
  - Destructor runs
- **User defined types and fundamental types are equally real**
  - Classes in the std namespace are very useful
  - Your own classes can do whatever fundamental types can do