Programming Languages and Tools: Programming with C++ CS:3210:0003

Lecture/Lab #16

The One Definition Rule (ODR)

- Within the same scope, an entity (function/variable/type) can have only one definition
- This applies across multiple files if they are compiled together

- If violated, compiler error
- Harder to fix with included files
- Don’t `#include .cpp` files
  - Instead compile them

Header Guards

- Within the same scope, an entity (function/variable/type) can have only one definition
- ODR applies to class definitions
- Use header guards in header files to prevent class redefinition
- Best practice: add header guards to header files
- Syntax:

```
#ifndef IDENT
#define IDENT
//entire file here
#endif
```

Destructors

- Clean up: set of tasks that a class must perform before object is destroyed
- Destructor: special member function automatically called when object of (non-aggregate) class type is destroyed
  - Used to perform clean up
- A destructor
  1. Has same name as class preceded by ~
  2. Takes no arguments
  3. Has no return type
- A class can have only one destructor
- Ex: for `class Fraction`, `~Fraction()` is a destructor
- For a (non-aggregate) class type, compiler will generate an implicit destructor if an explicit one isn't defined

Scope vs Linkage

- Scope of a variable determines the part of the program from which it is accessible
  1. Local: begins at declaration, ends with the current block of code
  2. Global: begins at declaration, ends with file
- Linkage of a variable determines whether other declarations of the variable refer to it or not
  1. No linkage: a variable declaration shadows the declaration of another one with the same name
  2. Internal linkage: declaration is accessible within the file, but not from outside
  3. External linkage: declaration is accessible from outside the file *via forward declaration*

Static Variables

- Variables/functions declared static have internal linkage, and global scope

Variables	Scope	Linkage
Local	Local	None
Non-const global	Global	External
Static non-const global	Global	Internal
Const global	Global	Internal

`extern`

- The `extern` keyword is used either to
    1. Give a const global variable external linkage
    2. To forward declare a variable from a different file, so it can be used in the current file
- Forward declaration of functions don’t need the `extern` keyword
- Generally, non-const global variables are dangerous
    - Values can be changed by any function

Variables by Duration

- Static duration: created when program starts (before `main()`), destroyed when it ends
- Automatic duration: created at point of definition, destroyed at end of block
- `static` keyword gives local variable static duration (but not global scope!)

Variable Type	Scope	Duration	Linkage
Local	Local	Automatic	None
Global	Global	Static	External
Static Local	Local	Static	Internal
Static Global	Global	Static	Internal

Static Local Variables

- Static duration, does not go out of scope at end of block
- Conventionally, named with an `s_` prefix
- Zero-initialized at start of program,
  - Reinitialized first time definition is encountered, if necessary
- Compiler skips initialization in subsequent calls
- Commonly used to generate unique IDs:

```
int generateID(){
    static int s_id{0};
    return s_id++;
}
```

Static Member Variables

- Class:
  - Regular member variables: each object has a copy of variable
  - Static member variables: shared by all objects of class
- Lifetime of a static member variable is not bound to class object
- Static duration: created at start of program, destroyed at end
- Accessed using class name: `className::s_varName`
- Static member variables must be
  - Declared in the class
  - Defined/initialized outside it

Activity

- Use static member variables to create unique IDs for each object of a class (starting from 1, 2, 3, …)

Static Member Functions

- Can access other static members, but not non-static members
- Accessed using class name: `className::funcName()`
- Lifetime of a static member function is not bound to class object
- Static duration: created at start of program, destroyed at end
- Constructors can’t be static