Introduction to Software Development – CS 6010
Lecture 7 – Structs

Master of Software Development (MSD) Program

Varun Shankar

Fall 2022

Notes from Yesterday…

```
for( x = 0; x < numbers.size(); x++ ) {
    numbers[x] = numbers[x+1];  // What does this do?
} // Also, why might it cause a problem?
```

- Finding the largest # (and many other tasks)

```
largest = numbers[ 0 ]; // Grab the 0th item.
for( x = 1; x < numbers.size(); x++ ) {
    // The for loop starts at one!
}
```

- #include “” vs #include <>
  - “” – files in the local directory. <> - system files
- Compiling from the command line vs XCode
- Link errors are hard to find in XCode.

Lecture 7 – Structs

- Topics
  - Struct(ure)s
  - Lab – Structs
  - Homework – Deck of Cards

Create Our Own Type…

...to represent some complicated data object.

- What information would we need to represent a student?
  - The type(s) of information is comprised of *simpler* types of data…
    - name string
    - ID int/string
    - GPA double
    - address Address
    - Could contain many other additional pieces of data…

How to organize this information?

- vector< string > student_names;
- vector< int > student_ids;
- vector< double > student_gpas;
- vector< Address > student_addresses;

- Information lives across 4 vectors. Messy.

Structs

- Structs are:
  - User defined datatypes
    - Note, there are system defined structs too (Like *time* information)
  - Comprised of any subset of other “simpler” datatypes.
  - Each piece of data in a struct must be named.
  - Represents a single object, for example: a car or a student or a building.

Student Struct(ure)

```
// Declare the Student struct in Student.h
struct Student { // Creating a new type (Name it anything you like)
    string name;  // Each field can be a different type
        int id;
        double gpa;
        Address addresses;     // What type is address? Another struct!
}; // end struct Student (notice the ';' at the end)

// Now create an object of type Student:
Student student; // Same as any other var:   type name;
vector< Student > students; // or a list of multiple students
```

Working with Structs

```
Student student;
cout << student << “\n”; // ERROR, this does not work. (yet)
cout << student.name << “\n”; // .method() for functions, .field for
fields.
vector< Student > roster;
for( Student s : roster ) { // Can use the foreach loop on a vector.
        cout << s.name;
}
for( int i = 0; I < roster.size(); i++ ) {
        cout << roster[ i ].name; // roster is just a vector
}
```

What is the *type* of *roster*? Of *roster[i]*?

Initializing a Student

```
Student ben; // How to set the name, id, and gpa of the ben Student?
ben.name = “Ben”;
ben.id = 12345;
ben.gpa = 4.0;
ben.address = … some address ….

Student ben { “Ben”, 12345, 4.0, some_address }; // Magic syntax to initialize struct
int i = 0;  // Similar to above line.
```

A Struct (Object) is NOT a String

- Declare a function that takes in a list of students and returns the students with at least a 3.0 GPA
- Incorrect examples:

  Student getHonorStudents()

  - Why is this incorrect?
    - This function does not take in any data (parameters) and thus can't process any data to give us a result.
    - This function returns a **single** Student - the function should return a list of Students.

  vector<string> getHonorStudents(vector<string> students)

  - Why is this function incorrect?
    - The function takes in (and returns) a list (vector) of strings. We were asked to return a list of Students. A *string* is not a *Student.*
    - "John Doe" is not { "John Doe", 12345, 3.6, address }
    - It is common among beginning programmers to think that the student name **is** the student, **but it is only a part of the student structure**.

Declare getHonorStudent()

vector<Student> getHonorStudents(vector<Student> allStudents);

- This is the correct declaration of the *getHonorStudents()* function. Takes in a list of students (ie: the *allStudents* vector) and returns (what will be a new) list of students.

- And just like any other function declaration, it is written:

<return type>   functionName(parameter(s)   );

Define (Implement) getHonorStudents()

```
vector<Student>   getHonorStudents( vector<Student> allTheStudents )
{
        // We need a place to store the information that we are returning
        vector<Student> honorStudents;

        // Do the processing… in this case add students to the someStudents vector.
        for(Student student: allTheStudents){
                if(isHonorStudent(student))
                        honorStudents.push_back( student);
        }

        // Return the information to who ever called us.
        return honorStudents;
}
```

Use getHonorStudent()

```
int main()
  {
   // call the function
    vector<Student> honorStudents = getHonorStudents( aListOfStudents );
    Student theFirstHonorStudent = honorStudents[0];

    // Display the 1st student's name
    cout << theFirstStudent.name;
    }
```

Include Guards

- To avoid including a header file twice. If the compiler has seen this .h file before, don't include it again (just skip it the 2nd+ times).

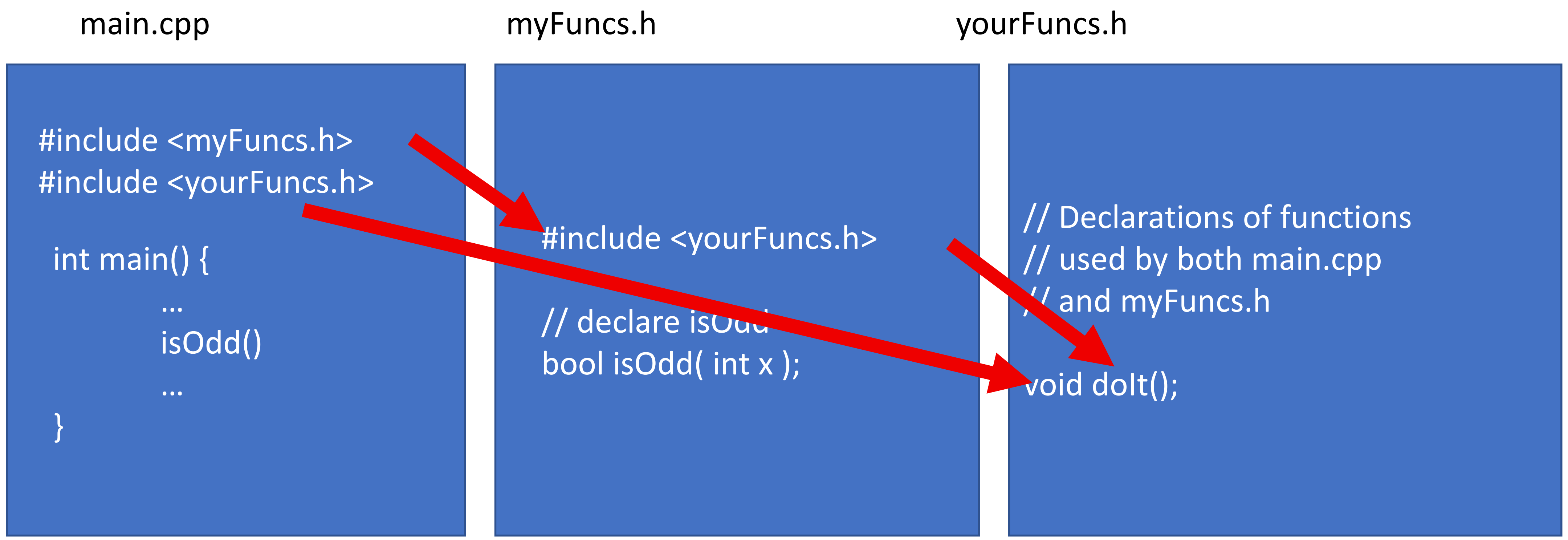
```
#ifndef SOMETHING_H
#define SOMETHING_H

// All the declarations of functions in this file

#endif
```

- Note, the new way to do this is:

```
#pragma once  // at the top of the file.
```

- Note, the "#" invokes the *preprocessor*

Include Guard Purpose



main.cpp

```
#include <myFuncs.h>
#include <yourFuncs.h>

 int main() {
          …
          isOdd()
          …
 }
```

myFuncs.h

```
#include <yourFuncs.h>

// declare isOdd
bool isOdd( int x );
```

yourFuncs.h

```
// Declarations of functions
// used by both main.cpp
// and myFuncs.h

void doIt();
```

Without include guards, because *#include* literally just copies the header file into the file including it, we would end up with two declarations of *"void doit()" in main.cpp (because main will have two copies of yourFuncs.h. This causes "redefinition" errors during compilation.*

Program Design

- First question is:
  - What type of data do I need to model in my program.
  - Most data represents complicated objects, thus we use structs to encapsulate that information.

In Class Example – Car

```
// Car.h
#pragma once
#include <string>
using namespace std; // Don't do this in a .h
file!

struct CarType {
  string make;
  string model;
};

struct Car {
  CarType type;
  int     numDoors;
  float   mpg; // miles per gallon
  string  color;
};
```

```
// main
#include <string>
#include <iostream>
#include "Car.h"

using namespace std;

int main() {
  Car c;
  c.type.make  = "Honda";
  c.type.model = "Civic";

  c.numDoors = 4;
  c.mpg      = 32.4;
  c.color    = "Blue";

  cout    << "The " << c.color << " " << c.type.make << " " << c.type.model
          << " has " << c.numDoors << " doors.\n";
```

Today’s Assignment(s)

- Lab – Structs
- Homework – Deck of Cards