STRUCTS, TYPEDEF, #DEFINE

THERE ONCE WAS A YOUNG MAN FROM LYME
WHO COULDN'T GET HIS LIMERICKS TO RHYME
WHEN ASKED, "WHY NOT?"
IT WAS SAID THAT HE THOUGHT…

THEY WERE PROBABLY TOO LONG AND
BADLY STRUCTURED AND NOT AT ALL VERY FUNNY

CSSE 120—Rose Hulman Institute of Technology

Exam 2

- Time for questions

Preamble: #define and typedef

- C allows us to define our own constants and type names so that our programs can more easily say what we mean.

```c
#define TERMS 3
#define FALL 0
#define WINTER 1
#define SPRING 2

typedef int coinValue;
coinValue quarter = 25, dime = 10;
```

For more info, see Kochan,
p. 299-303 (#define),
p. 325-327 (typedef)

How could we make our own boolean type?

Q1-Q2

Structures

- No objects or dictionaries in C. Structures (structs) are the closest thing that C has to offer.
- Two ways of grouping data in C:
  - **Array**: group several data elements of the **same type.**
    - Access individual elements by *position* : **student[i]**
  - **Structure**: group related data that may be of different types
    - Conceptually like how we used dictionaries to represent objects in Python
    - But syntax for element access is like objects in Python.
    - Access individual elements by *name*: **endPoint.x**
    - Not endPoint[“X”]

Q3-Q5

struct syntax

- struct <optional_tag_name> {

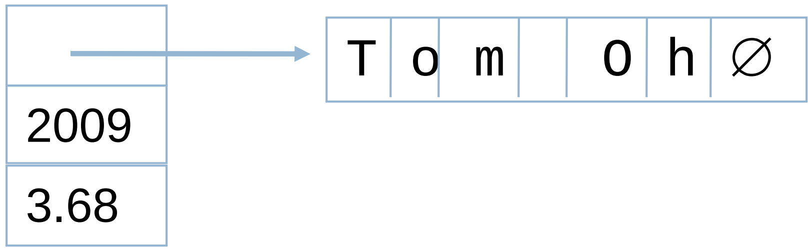
```
  <type_1>  <fieldname_1> ;
  <type_2>  <fieldname_2> ;
       . . .
  <type_n>  <fieldname_n> ;
};
```

- This says that each variable of this **struct** type has all these fields, with the specified types
- But structs are best declared in conjunction with **typedef**, as on on next slide…

Example: Student struct type

- Declare the type:

  -
    ```
    typedef struct {
        char *name;
        int year;
        double gpa;
    } Student;
    ```



- Function to print a student's info:

  -
    ```
    void printStudent(Student s) {
        printf("[%s  %d  %4.2lf]\n",
                  s.name, s.year, s.gpa);
    }
    ```
- Notice that once the type has been declared, it can be used in the same ways that a built-in type name is used.

Q6

Initializing a struct

Student juan;

juan.name = “Juan”

juan.year = 2008;

juan.gpa = 3.2;

Shorter:

Student juan = {“Juan”, 2008, 3.2};

(Only allowed when declaring and initializing variable together in a single statement.)

Just like our **makeCard** using Python dictionaries:

```
Student  makeStudent(char *name, int year, double gpa) {
  Student stu;
  stu.name = name;
  stu.year = year;
  stu.gpa = gpa;
  return stu;
}
```

```
typedef struct {
      char *name;
      int year;
      double gpa;
} Student;
```

Get the Point?

- Let’s define a **Point** struct type, similar to Zelle's Point class in Python (but without the GUI).
  - Make a new C Project
    - (Hello World ANSI C Project)
  - Create a typedef for the Point struct
  - Define makePoint()
  - Define a distance function that receives two points and returns a double for the distance between them.
  - In main(), make two Points from coordinates entered <u>by the user</u> and find the distance between them.

Add more struct types

- For a line segment, what should the fields be?
- Do the quiz question.
- Then type it in to create a new Segment struct
- In main(), use the 2 points created from coordinates entered by the user to create a segment
  - As time allows, write a **length** function for a segment. Hint: can you call the distance function we already wrote to avoid copy & paste?
- If you finish early, checkout and start reading HW23 RectangleStructs

A C Program in Multiple Files

- Check out ***Session23RectangleStructs*** from SVN.
- A large program can be organized by separating it into multiple files.
- Notice the three source files:
  - **rectangle.h** contains the struct definitions and function signatures used by the other files.
  - **rectangle.c** contains the definitions of the functions that comprise operations on point and Rectangle objects.
  - **Session23RectangleStructs.c** contains a main function to test the various functions of the rectangle module.
- Both of the **.c** files must include the **.h** file.