STRUCTS, TYPEDEF, #DEFINE

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.

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

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

typedef int creditHours[TERMS];
creditHours susanHours;
susanHours[WINTER] = 15;
```

The Object of structures

- No classes and objects in C. Structures (structs) are the closest thing that C has to offer.
  - Can't encapsulate data and operations together, as Python does in class definitions.
  - No equivalent of Python's **self**.
- 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
    - Access individual elements by *name*: **endPoint.x**

Q3-Q5

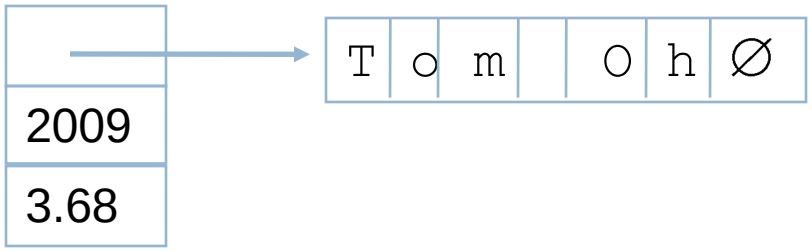
struct syntax

- 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, of the specified types.
- Structs are best declared in conjunction with typedef.

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.

Initializing a struct

```
student juan;
juan.name = "Juan"
juan.year = 2008;
juan.gpa = 3.2;
```

Shorter:

```
student juan = {"Juan", 2008, 3.2};
```

(Only when declaring and initializing variable together, like arrays.)

No constructors in C, but we can fake one:

```
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 (**PointStruct**)
    - (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 2 points from coordinates entered by the user and find the distance between them.

Add more struct types

- For a segment, what should the fields be?
  - segment – a line segment
- Write the code together for this type declaration
- Implement code to create a new segment
- 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 segments. Hint: can you call the distance function we already wrote to avoid copy & paste?

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.

Add functions for homework

```
/* Makes a rectangle from the given coordinates.  A rectangle is
 * made up of two points. */
Rectangle makeRect(int x1, int y1, int x2, int y2);

/* Returns the x-coordinate of the left-most edge of the given
 * rectangle. */
int getLeft(Rectangle r);

/* Returns the x-coordinate of the right-most edge of the given
 * rectangle. */
int getRight(Rectangle r);

/* Returns the y-coordinate of the top-most edge of the given
 * rectangle. */
int getTop(Rectangle r);
```

Add additional functions

```
/* Returns the y-coordinate of the bottom-most edge of the given
 * rectangle. */
int getBottom(Rectangle r);

/* Returns TRUE if the given rectangles touch and FALSE if they do
 * not touch. */
boolean areIntersecting(Rectangle q, Rectangle r);

/* Returns a new rectangle representing the overlapping area of
 * the two given rectangles.  */
Rectangle intersect(Rectangle q, Rectangle r);
```