STRUCTS IN C

THE SHORTEST DISTANCE BETWEEN TWO POINTS
IS UNDER CON**STRUCT**ION
-- NOELIE ALTITO

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.

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 structs

- No classes and objects in C. 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 **self**.
- Two ways of grouping data:
  - Array: group several data elements of the same type.
    - Access individual elements by *position*: **student[i]**
  - Struct: group related data that may be of different types
    - Access individual elements by *name*: **endPoint.x**

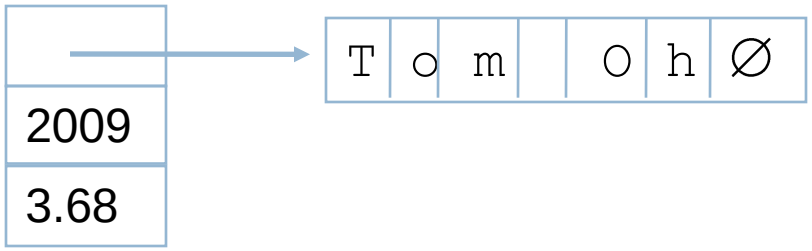
struct syntax

- 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 of 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  %1.2lf]\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.

Using the new type and function

```
student tomas;
tomas.name = "Tomas";
tomas.year = 2007;
tomas.gpa = 3.4;
printStudent(tomas);

student jose = {"Jose", 2006, 2.8};
printStudent(jose);

if (jose.year < tomas.year)
   printf("Jose is older.\n");
else
   printf("Jose is not older.\n");
```

Declare, initialize and print a variable of type **student**.

A shorter way!

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

OUTPUT:

```
[Tomas  2007  3.40]
[Jose  2006  2.80]
Jose is older.
```

```
void printStudent(student s) {
  printf("[%s  %d  %1.2lf]\n",
            s.name, s.year, s.gpa);
}
```

Get the Point?

- We define a **point** struct type, similar to Zelle's Point class in Python (but without the GUI).
- "Constructor" is not a C concept, but we can do something similar with a **makePoint** function.

```
typedef struct {
   double xCoord;
   double yCoord;
 } point;

point makePoint(double xx, double yy)  {
  point p;
  p.xCoord = xx;
  p.yCoord = yy;
  return p;
}
```

Go the Distance!

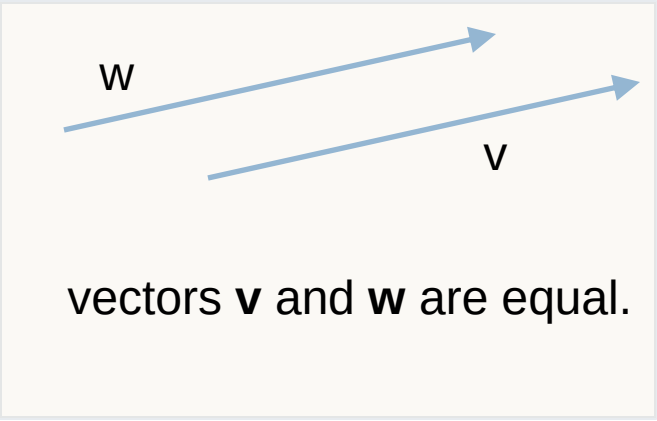
- In your C programming workspace, check out the **StructsIntro** project from your repository.
- Write a function **distance** that takes two point structs as its arguments, and returns the distance between them. Your code can call the following function:, which is provided for you.

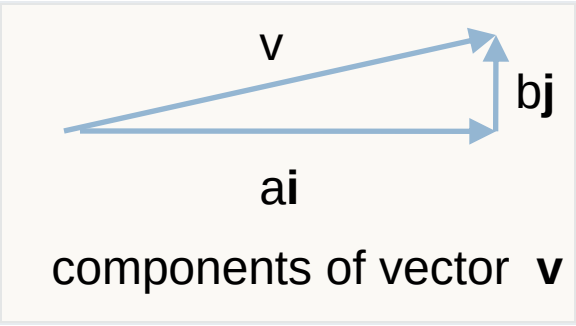
```
double square(double x) {
      return x * x;
}
```

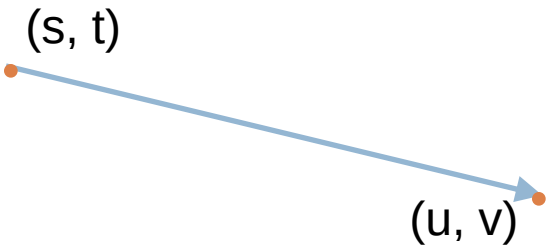
Ask for help as needed!

Quick review of vectors

- A vector has magnitude and direction.
- Two vectors are equal if they have the same magnitude and direction.
- A vector can also be expressed in terms of its x-component and y component:
  v = a**i** + b**j**, where a and b are its components.
- The vector from point (s, t) to point (u,v) is (u-s)**i** + (v-t)**j** .

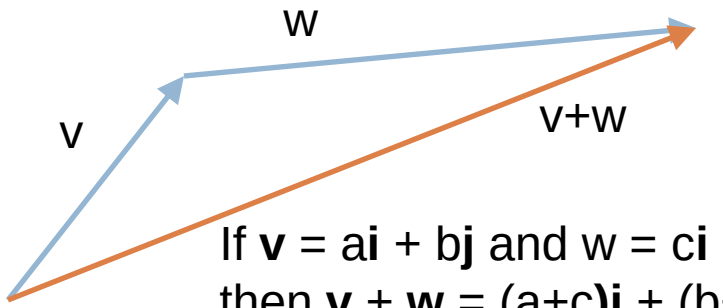




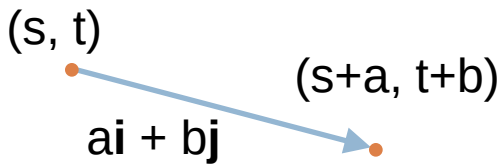


Some Vector Operations

- Vector addition
- Translating a point by a vector
- Length of $a\mathbf{i} + b\mathbf{j}$:
  $|v| = \text{sqrt}(a^2 + b^2)$
- dot product:
  $(a\mathbf{i} + b\mathbf{j}) \bullet (c\mathbf{i} + d\mathbf{j}) = ac + bd$



If $\mathbf{v} = a\mathbf{i} + b\mathbf{j}$ and $w = c\mathbf{i} + d\mathbf{j}$,
then $\mathbf{v} + \mathbf{w} = (a+c)\mathbf{i} + (b+d)\mathbf{j}$



Add more struct types

- for each type, what should the fields be?
  - segment – a line segment
  - vector – there are two reasonable choices of how we could represent a two-dimensional vector
  - line (no endpoints)
- Write the code together for these type declarations

Add additional functions together

```
/* print the coordinates of p in fixed format.  */
void printPoint(point p);

/* return the point from first pointCount elements
 * of pList that is closest to the origin. */
point closestToOrigin(point pList [], int pointCount);

/* Construct the vector from p1 to p2.          */
vector makeVecFromPoints(point p1, point p2);

/* Construct a line from a point and a vector. */
line makeLine(point p, vector v);

/* Construct a line segment from its endpoints.*/
segment makeSegment(point p1, point p2);
```

A C Program in Multiple Files

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