CSE 142
Programming I

Structures

Data Structures

- Data structures give us new ways to organize our data
  - ➤Store large amounts of data
  - ➤Store variable amounts of data
  - ➤Keep relevant data together
- Which ones did arrays help us with?

Structs

- A struct collects several variables, possibly of different types, together

driver's name	string
sex	char
age	int
height	double

Defining a struct

```
#define MAX_NAME 40
typedef struct{
  char name[MAX_NAME];
  char sex;
  int age;
  double height;
} driver;
```

YES!

What Does This Do?

- This `typedef struct {…} name;` construct only declares a new type—it creates NO storage.
- We can now make variables of the type driver

```
driver alice, bob;
```


Terminology

- Sometimes people will call a struct a "structure" or a "record"
- It's components are often called "records" or "fields"

Things You Can Do

- You CAN use = to assign entire structs!
- You CAN use structs are return types

Things you CAN'T do

- You CAN'T use == to compare structs
- You CAN'T use printf/scanf on entire structs
  - ➤You can scanf and printf the fields

Summary:

	arrays	strings	structs
use = to assign	no	no	yes
use == to compare	no	no	no
use printf/scanf	no	yes	no
use as parameters	yes	yes	yes
use as return vals.	no	no	yes

Accessing Struct Fields

- We can access the fields of a struct by using the . (dot) operator:

```
driver sara_lee;
sara_lee.sex = 'F';
sara_lee.height = '3.14';
```


Initializing Structs

- We could just assign all the fields by hand:

```
driver bob;
strcpy(bob.name, "Bob");
bob.sex = 'M';
bob.age = 112;
bob.height = 4.98;
```


Initializing Structs

- …or we could do it at the declaration:

```
driver bob = {"Bob",
              'M',
              112,
              4.98};
```

Assigning Structs

```
driver bob, opie;

/* initialize bob in here */

opie = bob;
    /* the same as copying all
       of the fields by hand */
```


Points as Structs

- We can make a struct to hold a point in three dimensional space:

```
typedef struct{
    double x, y;
} point_2d;
```


Structs and Functions

- We can write a midpoint function that returns the midpoint of a line in 2-space
- What is the right formula?

Midpoint Function

```
point_2d midpoint(point_2d a,
  point_2d b){




}
```


Midpoint With Pointers

```
void midpoint(point_2d a,
  point_2d b, point_2d *mid){




}
```


Why the -> ?

- The dot (.) operator says “look inside the struct”
- The -> operator says “follow the pointer, and look inside the struct you find there”

The -> Operator

- Assume the following situation

```
point_2d foo = {2.1, 5.4};
point_2d *fooPointer;

fooPointer = &foo;
```


The -> Operator

- In this setup,

```
fooPointer->x;
```

means the same thing as

```
(*fooPointer).x;
```


Testing Structs for Equality

- Remember: we can't use ==
- We can write a function:

```
int points_equal(point_2d pt1,
  point_2d pt2);
```


Struct I/O

- Likewise, we cannot scanf or printf structs, but we can easily write functions to do it for us.
- Example: How would we read in a point?

Hierarchical Structs

```
typedef struct{
  double x, y;
} point;

typedef struct{
  double width, height}
} dimensions;
```


Hierarchical Structs

```
typedef struct{
  dimensions size;
  point position;
  int line_color, fill_color;
} rectangle;
```

Accessing This Structure

```
rectangle a;

a.size.width = 3.0;
a.size.height = 4.0;
a.position.x = 10.0;
a.position.y = 20.0;
a.line_color = RED;
a.fill_color = MAUVE;
```


Calculating Types

- Given:

```
rectangle r;
rectangle *rp;
```

- What are the types?

```
r.size
r.size.width;
rp->size;
rp->position.x;
```