Structures

Section 4.3

Introduction

- C does not have classes and objects
- But still has a way to make a new data type composited of primitive data types
  - Not limited to just primitive types
- These custom types are called **structures**

struct Keyword

- Make a structure in C is a little like making a class in Java
- Groups related variables together
  - Called **structure members**
- Allows us to create a custom data type / template for such a grouping
- Cannot have methods / functions inside like classes can

- Use the **struct** keyword, followed by the name of the structure, and then a block of code specifying its variables
  - **struct *structureName* { ... };**
  - Note the closing brace is followed by a semicolon

```
struct point
{
    int x;
    int y;
};
```

```
struct person
{
    char first[32];
    char last[32];
    int year;
    double pay;
};
```

- ..doesn't allocate any memory for variables, just a template so far
- To actually make a structure variable, use the **struct** keyword, followed by the name of the structure, then a variable name
  - **struct *structureName variableName*;**
  - "**struct *structureName***" is like the data type

```
struct point p1;
struct point topLeft;

struct person pirate;
```

- To access members of a structure, say the structure variable's name, a dot, and the name of the member
  - ***variableName*.*memberName***
- When we do that, it can pretty much be treated like a regular variable
  - Note no public or private like with class members

```
p1.x = 10;
p1.y = 9;

printf("p1 = (%d, %d)", p1.x, p1.y);
```

```
pirate.year = 1567;
pirate.pay = 0.0;
strcpy(pirate.first, "Jack");
strcpy(pirate.last, "Sparrow");
```

Label(s)	Address	Value
p1, p1.x	400 – 403	10
p1.y	404 – 407	9

Label(s)	Address	Value
pirate, pirate.first	400 – 431	"Jack"
pirate.last	432 – 463	"Sparrow"
pirate.year	464 – 467	1567
pirate.pay	468 – 475	0.0

- If we try to assign a structure variable to another structure variable (of the same type), it copies it
  - ***variable1 = variable2;***
  - Copies *variable2* in *variable1*

- Can also use structures as parameter data types in functions

```
void printPoint(struct point p)
{
    printf("(%d, %d)", p.x, p.y);
}
```

- To use the function, we just pass in the variable name for the structure

```
printPoint(p1);
```

- Structures are passed by value into functions
  - Function parameter gets a copy of the structure passed in
- Need to keep this in mind, especially if the structure is rather large
  - See how to pass structures by reference next section