Structures

Discussion: C has primitive types like *int*, *double*, and *char*. Each holds a single "thing". C also has a concept called ***structures*** that allows the programmer to define her own aggregate types built from primitive types and other programmer-defined types.

For example, the programmer can define a `Dog` structure type that contains the Dog's age, weight and name.

Example: Here is a structure called `Dog` that has three components (called ***fields***): an *int* that is the Dog's age, a *float* that is the Dog's weight, and a *string* (character array) that is the Dog's name:

Defines a structure called `Dog` with three fields.

```
typedef struct {
    int age;
    float weight;
    char name[10];
} Dog;
```

Declares a structure variable called **dog1** whose type is **Dog**. Per the definition above of the Dog structure, this variable is a composite of 3 things: an *int*, a *float*, and a *string*.

Declares a variable called **dog2** whose type is **Dog** and ***initializes its three fields***.

```
int main() {
    Dog dog1;
    Dog dog2 = {12, 40.3, "lassie"};
```

Initializes the fields of **dog1**. Note the "***dot***" notation for ***referring to a structure variable's fields***.

```
    dog1.age = 4;
    dog1.weight = 90.5;
    strncpy(dog1.name, "marmaduke", 10);

    printf("Dog 1 is named %s. Its age is %i and weight is %f\n",
            dog1.name, dog1.age, dog1.weight);

    printf("Dog 2 is named %s. Its age is %i and weight is %f\n",
            dog2.name, dog2.age, dog2.weight);
}
```

A C structure variable is a weak version of an ***object*** in object-oriented (OO) languages. An object has *data* associated with it and *operations* it can do. A structure variable has only data associated with it; it cannot do any operations itself.

Structures

Defining a structure type:

Notation:

```
typedef struct {
    TYPE  FIELD_NAME;
       ...
    TYPE  FIELD_NAME;
} STRUCTURE_NAME;
```

Example:

```
typedef struct {
    int age;
    float weight;
    char name[10];
} Dog;
```

Note especially the placement of the *curly-braces* and the *concluding semi-colon*. Omitting the semi-colon creates many hard-to-decipher compile-time error messages!

Define the structure type at the top level of the file (i.e., not inside *main* or any other function), so that it can be used anywhere in the file.

The types of the fields can themselves be structures, as well as arrays (as in example above) or pointers (which can simulate arrays).

There are other notations for defining structures, but the above approach is clearest. Use it.

Declaring a structure variable (aka *instance*): Proceed just like for primitive types.

Examples:

```
Dog dog1;
Cat cat1, cat2, cat3;
Point points[30];
Dog* pointerToDog;
```

One `Dog` variable.

Three `Cat` variables.

One `Point` array, with 30 Point's.

One pointer to a `Dog`.

Accessing the *fields* of a structure variable: Use the "dot" notation, like this:

Example: Suppose you have:

```
typedef struct {
    int age;
    float weight;
    char name[10];
} Dog;
```

Then you could have statements like:

```
Dog dog1, dog2;

dog1.weight = 30.2;
scanf("%i", &(dog1.age));
dog2.age = dog1.age - 3;
strcpy(dog1.name, "buster");
```

That is, you put the structure variable name, then a dot, then the field name.