ECE 131 – Programming Fundamentals

Dr. Daryl Lee

University of New Mexico

Structures

- What do arrays allow you to do? Create a collection of variables, all of the same type, and group them together under one name. To access individual elements of the array, you index into it.
- What if we'd like to group together a collection of variables, but they don't all have the same type?
  Ex: In a student record, you may wish to store variables related to:
  - name
  - address
  - phone number
  - email
  - student id
  - list of classes

  The group as a whole is called a "student record", and it has the "fields" or "members" shown in the list. A student database would be composed of many such student records, one for each student.

THE UNIVERSITY of NEW MEXICO

Structures

- In C, structures are used to implement the concept we have been describing.
- In other programming languages, structures are sometimes called records.
- In C the keyword struct is used to declare a structure, here's the syntax:

```
struct <structure_name> {
    member_list
} <structure_tag>;
```

where the *member_list* consists of any collection of valid C variable declarations. I.e., the variable declarations in the member list may include "regular" variables, pointers, arrays, or even other structure declarations.
We'll talk about the *structure_name* and *structure_tag* parts of this declaration through the use of examples.

Structures

Ex: You may wish to group together the x and y coordinates of point in two dimensional space, and call such a point a *vector*. Here's how you'd do that using a C structure:

```
struct vector {
   double x;
   double y;
};
```

- This creates a "template" that you can now use in order to declare variables of type `vector`.
  E.g., `struct vector pt1, pt2;`
- The first declaration, of `vector`, does *not* cause the compiler to allocate any memory. However, by using the *structure_name*, `vector`, the second declaration will cause the compiler to allocate memory for (i.e., instantiate) the variables `pt1` and `pt2`.
- Notice that the keyword `struct` must be used when declaring `pt1` and `pt2`.

Structures

- In summary, a structure defines a type that may be instantiated using the *structure_name*.
- Using the *structure_tag*, we can define the structure, and instantiate instances of it as well.

E.g.,

```
struct {
    double x;
    double y;
} pt;
```

- This instantiates a structure variable called pt.
- However, because the *structure_name* was left out, there is no way to declare additional structure variables of this type. I.e., this is a one-time instantiation.

Structures

- We can use both the *structure_name* and the *structure_tag* in a declaration.

E.g.,

```
struct vector {
   double x;
   double y;
} pt;
```

  - This instantiates a structure variable called `pt`.
  - As before, additional structure variables may be instantiated using: `struct vector`

Structures

- The syntax for initializing a structure is similar to what we used for initiatlizing arrays—a comma separated list of elements enclosed in curly braces.

  Ex: `struct vector a={1.1,2.2}, b={3,2};`

  The `x` and `y` members in variable `a` will be assigned `1.1` and `2.2`, and the `x` and `y` members in `b` will be assigned `3` and `2` (but these will first be converted to doubles).
- Notice that the values are assigned to the members in the order they are listed.
- Using the following syntax, you can initialize the members in any order.

  Ex:

  ```
  struct vector a={.y=2.2, .x=1.1}, b={.x=3, .y=2};
  ```

The Typedef Statement

- The typedef statement in C can be used to give an alternative name to a data type.
- The syntax involves the keyword `typedef`, followed by an existing type, then the name of the new type you want to declare:

`typedef` *known-type new-type*;

Ex: `typedef int number;`

This creates a new name for `int` called `number`. You can then use `number` in place of `int`.

E.g., `number a,b;` // compiler converts this to: `int a, b;`

Structures

The typedef statement is commonly used with structures.

E.g.,

```
typedef struct {
   double x;
   double y;
} vector;
```

- Here the “known-type” is the struct, and the “new-type” is “vector”.
- Given the typedef statement above, we can declare variables as follows:

```
    vector a, b;
```

Now we don’t need to use the keyword struct.

Structures - Copying

Copying structures is easy.

E.g.,

```
vector v1, v2;
...
v2 = v1;
```

Copies the entire struct v1 into v2.

Structures

In order to use a structure variable, we need to be able to access its members.

- The membership operator "." is used to access a member directly through the structure variable.

Ex: Given the previous typedef statement, consider the variable declarations:

```
vector a = {1.1, 2.2};
```

Then `a.x` evaluates to `1.1`, and `a.y` evaluates to `2.2`.

Structures

The following function will add two `vector` variables, provided as input, and return the result:

```
vector add(vector c1, vector c2)
{
   vector result; // variable for the result
   result.x = c1.x + c2.x;
   result.y = c1.y + c2.y;
   return result;
}
```

Here's how you could use it:

```
vector a={1,2}, b={2,4}, c;
c = add(a,b);
```

Arrays and Structures

Arrays can hold structures as elements:

```
vector model[1000];
...
model[3].x = 1.78;
```

Structures can include arrays:

```
typedef struct {
   int studentID;
   int courseID[10];
} courseList;
courseList advisory[10];
...
advisory[2].studentID = 105793268;
advisory[2].courseID[0] = 819462;
```