ECE 222 System Programming Concepts
Lecture 7 – Structures

What is a structure? A structure is a method to define a new data type (we have 5 so far: char, int, float, double, pointer) that refers to a group of variables using one name. Each part of a structure is called a field.

```c
struct person {
    char first[30]; /* first field of structure is array of char */
    char last[30]; /* second field of structure is array of char */
    int year; /* third field of structure is int */
    double ppg; /* fourth field is double */
};
/* ending ; means end of structure type definition */
```

In this case, “person” is the new data type. It consists of two arrays of char, an int, and a double. However, we have not yet declared a variable of the new type, just a template. In other words, we have defined something like “int”, not a variable of type “int”.

To declare a structure, we use the new type definition:

```
struct person   teacher;
```

We use the “.” symbol to access parts, called fields, of the structure:

```
teacher.year=2005;
teacher.ppg=10.4;
strcpy(teacher.first,"Adam");
strcpy(teacher.last,"Hoover");
```

What is the result of the following print statements?

```
printf("year: %d     points per game: %lf\n",teacher.year,teacher.ppg);
printf("%c\n",teacher.first[3]);
printf("%c %c\n",teacher.last[6],teacher.last[9]);
printf("%d
",teacher.last[6]);
printf("%c %c\n",teacher.first[32],teacher.first[33]);
```

To answer these questions, construct a memory map:

<table>
<thead>
<tr>
<th>label</th>
<th>address (byte)</th>
<th>value (ACSII symbol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>teacher.first[0]</td>
<td>400</td>
<td>65 ‘A’</td>
</tr>
<tr>
<td>[1]</td>
<td>401</td>
<td>100 ‘d’</td>
</tr>
<tr>
<td>[2]</td>
<td>402</td>
<td>97 ‘a’</td>
</tr>
</tbody>
</table>
[use struct2.c code example]

The individual structs in the array of structs are contiguous in memory, just as the fields of a single struct are contiguous in memory. The address of a struct is the same as the address of its first part; one need only think about a memory map to see this.

Struct variables can be global or local to a function, just like the other 5 data types.

[use struct3.c code example]

A struct template can be “throw away”, meaning it is used only once, when declaring variables of the given template. Without a template name, that structure template cannot be used to declare more variables of the same type (what would we call them?).

The second display of the paperback.title demonstrates a buffer overflow. We copied a name longer than the 20 chars defined for the field, so it overran the next bit of memory, which is the “cost” field. You can resolve these sorts of enigmas by reasoning about the memory map, and where a buffer overflow must have come from.

The last printf statements demonstrate global versus local structs work like any other data type.