Computer Science & Engineering 150A
Problem Solving Using Computers

Lecture 08 - Structures

Stephen Scott
(Adapted from Christopher M. Bourke)

Fall 2009

Introduction

- We've seen built-in simple data types: `int`, `double`, `char`, etc.
- *Simple* data types hold one value at any one time
- *Complex* data types can hold a *collection* of values

Structures

- C allows user-defined complex types through the use of *structures*
- Structures are data types that have *components*
- Components can either be simple data types or other structures

Structure I
Syntax

- Declare a structure using the `typedef struct` syntax:

```
1  typedef struct {
2    int anInt;
3    double aDbl;
4    int anotherInt;
5    char aString[20];
6  } aStructure;
```

- `aStructure` is the name of a new type
- Can now declare variables of type `aStructure` just like you'd declare a variable of type `int`, etc.
- Each variable of this type has 4 simple data type components

Structures
Motivation

- Certain data need to be logically grouped together
- *Records* in a database: data associated with a single person should be kept together
- A *person* may consist of first name, last name, NUID, birth date, etc.
- In turn, a birth date has a month, day, year
- Structures allow us to define such records

Structures
Usage

- Declare an instance of a structure like you would an atomic type:
```
aStructure anExampleOfAStructure;
aStructure anArrayOfStructures[10];
```
- To set or access the individual member variables we use the following syntax:
```
myStructure.memberVariable
```
- Known as *component selection operator*
- Examples: `anExampleOfAStructure.aDbl = 6.5`
`anArrayOfStructures[3].anotherInt = 4`

Structures
Example


```
1  typedef struct {
2    char month[15];
3    int day;
4    int year;
5  } date_t;
6
7  typedef struct {
8    char firstName[30];
9    char lastName[30];
10   int  NUID;
11   date_t birthDate;
12 } student;
13
14 int main(int argc, char *argv[])
15 {
16   ...
17
18   student aStudent;
19   strcpy(aStudent.firstName,"Tom");
20   strcpy(aStudent.lastName,"Waits");
21   aStudent.NUID = 12345678;
22   strcpy(aStudent.birthDate.month, "December");
23   aStudent.birthDate.day = 7;
24   aStudent.birthDate.year = 1949;
25
26   ...
27 }
```

Structures
As Function Arguments

- It is possible to pass and return structures as function arguments
- Same syntax as simple data types
- You can pass by value or by reference (address, array)

```
1 void printStudentRecordByValue(student s);
2 void printStudentRecordByRef(student *s);
```

Returning Structures

- As with arrays, we cannot return structures that are *local* in scope
- We must return a *pointer* to a dynamically allocated structure
- We haven't covered this in class, but it is in the textbook

Common Errors I

- Careful: structures must be declared before they are referenced or used
- Usually declared between preprocessor directives and function prototypes
- The size of a structure is not bounded: you can include as many components as you want
- Passing large structures *by value* is generally inefficient
- The entire structure is copied to the system stack on a by-value function call
- Pass large structures by reference (address) whenever possible

Exercise 1
Album Structure

Design a structure for album data. Include components for album title, artist, track titles, number of tracks, year, and any other relevant data.

Exercise 2
Complex Numbers

- A complex number consists of two real numbers: a real component and an imaginary component
- Complex numbers are able to handle roots of negative numbers: Examples:

$$\sqrt{-4} = 0 + 2i$$

$$\sqrt[4]{-4} = 1 + i$$

- Define a structure to handle complex numbers. Write a function to print them in a nice format.