

Computer Science & Engineering 150A

Problem Solving Using Computers

Lecture 11 - Structures

Christopher M. Bourke

Spring 2009

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

- 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

- 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 structure's name
- It has 4 simple data type components

- 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

- 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*

Structures

Example

CSCE150A

Introduction

Structures

Pointers

Function Args

Pitfalls

Exercises

```
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 }
```

- Pointers to structures can also be used just as with simple data types
- Syntax: `student *myStudent = NULL;`
- You can reference ordinary structures as well:
`myStudent = &aStudent;`

- Dynamically allocate memory for structures using `malloc`
- Structure components are fixed at compile time, so C knows “how big” they are; use the `sizeof` function
- **Pitfall**: remember to cast your pointer!

```
1 student *newStudent = NULL;  
2 newStudent=(student *)malloc(sizeof(student));
```

- If a structure is referenced via a pointer, there is different syntax for accessing its members
- Instead of a period, we use a *pointer*: ->
- Known as *indirect component selection operator*
- “Equivalent” to `(*newStudent).NUID` (dereference, then component selection)

```
1 student *newStudent = NULL;
2 newStudent=(student *)malloc(sizeof(student));
3
4 strcpy(newStudent->lastName, "Jones");
5 newStudent->NUID = 24681012;
6 newStudent->birthDate.year = 1940;
```

Structures

As Function Arguments

CSCE150A

Introduction

Structures

Pointers

Function Args

Pitfalls

Exercises

- 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

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

- As with arrays, we cannot return structures that are *local* in scope
- We must return a *pointer* to a dynamically allocated structure

```
1  student * readInStudent ()
2  {
3      student *newStudent = NULL;
4      newStudent=(student *)malloc(sizeof(student));
5      printf("Enter First Name>");
6      scanf("%s", newStudent->firstName);
7      printf("Enter NUID>");
8      scanf("%d", &newStudent->NUID);
9      return newStudent;
10
11 }
```

- **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 structures *by value* is generally a bad idea
- The entire structure is copied to the system stack on a by-value function call: very inefficient
- Pass structures by reference whenever possible
- Know when to use the `.memberVariable` syntax and when to use the `->memberVariable` syntax

Exercise 1

Album Structure

CSCE150A

Introduction

Structures

Pointers

Function Args

Pitfalls

Exercises

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

CSCE150A

Introduction

Structures

Pointers

Function Args

Pitfalls

Exercises

- 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. Also write a function to compute the multiplication of two complex numbers, define as:

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

- Rewrite the `quadraticRoots` function to compute the roots of a quadratic equation as `complex` types

CSCE150A

Introduction

Structures

Pointers

Function Args

Pitfalls

Exercises

Do Programming Project 1 in Chapter 11 (p601)