EECS 22: Advanced C Programming

Lecture 11

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 11: Overview

- Course Administration
  - Midterm course evaluation: Results
  - Midterm exam: Review and Discussion
- Data Structures
  - Structures
  - Type definitions

Course Administration

- Midterm Course Evaluation: Results
  - Participation
    - 12 out of 19 students (63.16%)
    - Thank you!
  - Specific Feedback
    - Overall very positive
    - Overlap and discrepancy to EECS 20
    - Some interesting suggestions
  - MidtermEvaluation_Report.pdf
  - Discussion…

Course Administration

- Midterm Exam: Review and Discussion
  - Overall satisfactory results
    - Most show good understanding
    - Some questions appear to be “harder”
      - Q1, Q4, Q7, Q16, Q17, Q18
    - “Free” programming appears to be a harder (new?) topic
      - Contents of header files
      - Makefile!
  - MidtermExam_Solution.pdf
  - Discussion…

Data Structures

- Basic Data Types
  - Non-composite types with built-in operators
    - Integral types
    - Floating point types
- Static Data Structures
  - Composite user-defined types with built-in operators
    - Arrays
    - Structures, unions, enumerators
- Dynamic Data Structures
  - Composite user-defined types with user-defined operations
    - Lists, queues, stacks
    - Trees, graphs
    - Dictionaries, etc.
    - ➢ *Pointers*!

Data Structures

- Structures (aka. *records*): **`struct`**
  - User-defined, composite data type
    - Type is a composition of (different) sub-types
  - Fixed set of members
    - Names and types of members are fixed at structure definition
  - Member access by name
    - Member-access operator: ***`structure_name.member_name`***
- Example:

```
struct S { int i; float f;} s1, s2;

s1.i = 42;     /* access to members */
s1.f = 3.1415;
s2 = s1;       /* assignment */
s1.i = s1.i + 2*s2.i;
```

Data Structures

- Structure Declaration
  - Declaration of a user-defined data type
- Structure Definition
  - Definition of structure members and their type
- Structure Instantiation and Initialization
  - Definition of a variable of structure type
  - Initializer list defines initial values of members
- Example:

```
struct Student;          /* declaration */

struct Student           /* definition */
{ int   ID;              /* members */
  char  Name[40];
  char  Grade;
};

struct Student Jane =    /* instantiation */
{1001, "Jane Doe", 'A'}; /* initialization */
```

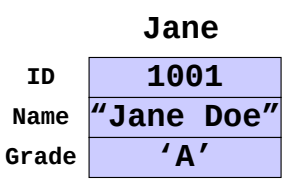

Data Structures

- Structure Access
  - Members are accessed by their name
  - Member-access operator `.`
- Example:

```
struct Student
{  int  ID;
   char Name[40];
   char Grade;
};

struct Student Jane =
{1001, "Jane Doe", 'A'};

void PrintStudent(struct Student s)
{
   printf("ID:    %d\n", s.ID);
   printf("Name:  %s\n", s.Name);
   printf("Grade: %c\n", s.Grade);
}
```




```
ID:    1001
Name:  Jane Doe
Grade: A
```

Data Structures

- Type definitions: **`typedef`**
  - A type definition creates an *alias* type name for another type
  - A type definition uses the same syntax as a variable definition
    - Technically, **`typedef`** is a storage class!
  - Type definitions are often used…
    - as common type name used in several places in the code
    - as shortcut for composite user-defined types (objects)
- Examples:

```
typedef unsigned long UInt64;   /* 64-bit type */

typedef struct Student Scholar; /* shortcut */
Scholar Jane, John;

typedef struct Image            /* PhotoLab image type */
{ unsigned int  Width, Height;
  unsigned char *R, *G, *B;
} IMAGE;
```