

# EECS 10: Computational Methods in Electrical and Computer Engineering

## Lecture 20

Rainer Dömer

doemer@uci.edu

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

## Lecture 20: Overview

- Course Administration
  - Final course evaluation
- Data Structures
  - Structures
    - Declaration and definition
    - Instantiation and initialization
    - Member access
  - Unions
    - Declaration and definition
    - Member access
  - Enumerators
    - Declaration and definition
  - Type definitions

## Course Administration

- Final Course Evaluation
  - Starting this week, until 10<sup>th</sup> week
  - Nov. 17, 2004, 8am through Dec. 10, 2004, 5pm
  - Online via EEE Evaluation application
- Feedback from students to instructors
  - Completely voluntary
  - Completely anonymous
  - Very valuable
    - Help to improve this class!
- Evaluation asks for current grade estimate
  - Suggest to wait after midterm 2 scores are in

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

3

## Data Structures

- Structures (aka. records)
  - 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;
```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

4

## Data Structures

- Structure Declaration
  - Declaration of a user-defined data type

- Example:

```
struct Student;          /* declaration */
```

## Data Structures

- Structure Declaration
  - Declaration of a user-defined data type
- Structure Definition
  - Definition of structure members and their type

- Example:

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

## Data Structures

- Structure Declaration
  - Declaration of a user-defined data type
- Structure Definition
  - Definition of structure members and their type
- Structure Instantiation
  - Definition of a variable of structure type
- Example:

```

struct Student;           /* declaration */

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

struct Student Jane;     /* instantiation */

```

## 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);
}

```

Jane	
ID	1001
Name	"Jane Doe"
Grade	'A'

```

ID:    1001
Name:  Jane Doe
Grade: A

```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

9

## Data Structures

- Unions
  - User-defined, composite data type
    - Type is a composition of (different) sub-types
  - Fixed set of mutually exclusive members
    - Names and types of members are fixed at union definition
  - Member access by name
    - Member-access operator: *union\_name.member\_name*
  - *Only one member may be used at a time!*
    - *All members share the same location in memory!*
- Example:

```

union U { int i; float f;} u1, u2;

u1.i = 42;      /* access to members */
u2.f = 3.1415;
u1.f = u2.f;   /* destroys u1.i! */

```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

10

## Data Structures

- Union Declaration
  - Declaration of a user-defined data type

- Example:

```
union HeightOfTriangle; /* declaration */
```

## Data Structures

- Union Declaration
  - Declaration of a user-defined data type
- Union Definition
  - Definition of union members and their type

- Example:

```
union HeightOfTriangle; /* declaration */  
union HeightOfTriangle /* definition */  
{ int Height; /* members */  
  int LengthOfSideA;  
  float AngleBeta;  
};
```

## Data Structures

- Union Declaration
  - Declaration of a user-defined data type
- Union Definition
  - Definition of union members and their type
- Union Instantiation
  - Definition of a variable of union type
- Example:

```
union HeightOfTriangle; /* declaration */
union HeightOfTriangle /* definition */
{ int Height; /* members */
  int LengthOfSideA;
  float AngleBeta;
};
union HeightOfTriangle H; /* instantiation */
```

## Data Structures

- Union Declaration
  - Declaration of a user-defined data type
- Union Definition
  - Definition of union members and their type
- Union Instantiation and Initialization
  - Definition of a variable of union type
  - *Single* initializer defines value of first member
- Example:

```
union HeightOfTriangle; /* declaration */
union HeightOfTriangle /* definition */
{ int Height; /* members */
  int LengthOfSideA;
  float AngleBeta;
};
union HeightOfTriangle H /* instantiation */
= { 42 }; /* initialization */
```

## Data Structures

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

```
union HeightOfTriangle
{ int   Height;
  int   SideA;
  float Beta;
};

union HeightOfTriangle t1, t2, t3
= { 42 };

void SetHeight(void)
{
  t1.Height = 10;
  t2.SideA = t1.Height / 2;
  t3.Beta = 90.0;
}
```

	t1
Height/	
SideA/	10
Beta	
	t2
Height/	
SideA/	5
Beta	
	t3
Height/	
SideA/	90.0
Beta	

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

15

## Data Structures

- Enumerators
  - User-defined data type
    - Members are an enumeration of integral constants
  - Fixed set of members
    - Names and values of members are fixed at enumerator definition
  - Members are constants
    - Member values cannot be changed after definition
- Example:

```
enum E { red, yellow, green };
enum E LightNS, LightEW;

LightEW = green;          /* assignment */
if (LightNS == green)    /* comparison */
  { LightEW = red; }
```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

16



## Data Structures

- Enumerator Declaration
  - Declaration of a user-defined data type

- Example:

```
enum Weekday;          /* declaration */
```

## Data Structures

- Enumerator Declaration
  - Declaration of a user-defined data type
- Enumerator Definition
  - Definition of enumerator members and their value

- Example:

```
enum Weekday;          /* declaration */
enum Weekday           /* definition */
{ Monday, Tuesday,    /* members */
  Wednesday, Thursday,
  Friday, Saturday, Sunday;
};
```

## Data Structures

- Enumerator Declaration
  - Declaration of a user-defined data type
- Enumerator Definition
  - Definition of enumerator members and their value
- Enumerator Instantiation
  - Definition of a variable of enumerator type
- Example:

```
enum Weekday;           /* declaration */
enum Weekday           /* definition */
{ Monday, Tuesday,    /* members */
  Wednesday, Thursday,
  Friday, Saturday, Sunday;
};
enum Weekday Today;    /* instantiation */
```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

19

## Data Structures

- Enumerator Declaration
  - Declaration of a user-defined data type
- Enumerator Definition
  - Definition of enumerator members and their value
- Enumerator Instantiation and Initialization
  - Definition of a variable of enumerator type
  - Initializer should be one member of the enumerator
- Example:

```
enum Weekday;           /* declaration */
enum Weekday           /* definition */
{ Monday, Tuesday,    /* members */
  Wednesday, Thursday,
  Friday, Saturday, Sunday;
};
enum Weekday Today     /* instantiation */
= Wednesday;          /* initialization */
```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

20

## Data Structures

- Enumerator Values
  - Enumerator values are integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
- Example:

Today

Wednesday

Day of week: 2

```
enum Weekday
{ Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday;
};

enum Weekday Today
= Wednesday;

void PrintWeekday(
    enum Weekday d)
{
    printf("Day: %d\n", d);
}
```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

21

## Data Structures

- Enumerator Values
  - Enumerator values are integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
  - Specific enumerator values may be defined by the user
- Example:

Today

Wednesday

Day of week: 3

```
enum Weekday
{ Monday = 1,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday;
};

enum Weekday Today
= Wednesday;

void PrintWeekday(
    enum Weekday d)
{
    printf("Day: %d\n", d);
}
```

EECS10: Computational Methods in ECE, Lecture 20

(c) 2004 R. Doemer

22

## Data Structures

- Enumerator Values
  - Enumerator values are integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
  - Specific enumerator values may be defined by the user
- Example:

Today

Wednesday

Day of week: 4

```
enum Weekday
{ Monday = 2,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday = 1;
};

enum Weekday Today
= Wednesday;

void PrintWeekday(
    enum Weekday d)
{
    printf("Day: %d\n", d);
}
```

## Data Structures

- Type definitions
  - A *typedef* can be defined as an alias type for another type
  - A *typedef* definition follows the same rules as a variable definition
  - Type definitions are usually used to abbreviate access to user-defined types
- Examples:

```
typedef long MyInteger;

typedef enum Weekday Day;
Day Today;

typedef struct Student Scholar;
Scholar Jane, John;
```