

# EECS 22: Advanced C Programming

## Lecture 15

Rainer Dömer

doemer@uci.edu

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

## Lecture 15: Overview

- Data Structures
  - Structures
  - Unions
  - Bit fields
  - Enumerators
  - Type definitions

## 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, bit fields, unions, enumerators
- Dynamic Data Structures
  - Composite user-defined types with user-defined operations
    - Lists, queues, stacks
    - Trees, graphs
    - Dictionaries, ...
    - *Pointers!*

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

3

## 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;

```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

4

## 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 */

```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

5

## 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

```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

6

## Data Structures

- Unions: **union**
  - 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! */
```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

7

## 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 */
```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

8

## 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 };
```

Height/	t1
SideA/	0
Beta	
Height/	t2
SideA/	0
Beta	
Height/	t3
SideA/	42
Beta	

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

9

## 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;
}
```

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

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

10

## Data Structures

- Bit Fields: Packing a few bits into a machine word
  - User-defined, composite data type
    - Type is a structure of sub-word-length bit fields (small integers)
  - Fixed set of members
    - Names and size of bit fields are fixed at bit field definition
  - Member access by name
    - Member-access operator: *structure\_name.bitfield\_name*
- Example:

```
struct FontAttribute {
    unsigned int IsItalic : 1;
    unsigned int IsBold   : 1;
    int /* padding */     : 0;
    unsigned int Size     : 12;
} Style;
Style.IsItalic = 0;
Style.IsBold   = 1;
Style.Size     = 600;
```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

11

## Data Structures

- Bit Fields: Packing a few bits into a machine word
  - Examples for usage
    - Flags: Set of single bits indicating a condition, property, or attribute
    - Device registers (e.g. CPU status, or UART I/O register)
    - Packing of small integers (e.g. floating-point representation)
  - Advantages
    - Space-efficiency with convenient access
    - Better readability
      - As compared to using bit-wise operators, shifting, and bit constants
  - Portability
    - The layout of bit fields in memory is implementation defined!
    - Position of bits in memory depends on
      - Compiler (bit packing strategy, loose or tight)
      - Byte-order of target machine (big vs. little endian)
      - Machine word width

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

12

## Data Structures

- Example: `Bitfield.c`

```

/* Bitfield.c: 11/06/12, RD */
#include <stdio.h>

struct FloatFormat {
    unsigned int Mantissa : 23;
    unsigned int Exponent : 8;
    unsigned int Sign : 1;
};

union FloatUnion {
    float Value;
    struct FloatFormat Format;
} Float = { -1.0 };

int main(void)
{ printf("sizeof(float) = %lu\n", sizeof(float));
  printf("sizeof(Float) = %lu\n", sizeof(Float));
  printf("Float.Value = %f\n", Float.Value);
  printf("Float.Format.Sign = %u\n", Float.Format.Sign);
  printf("Float.Format.Exponent = %u\n", Float.Format.Exponent);
  printf("Float.Format.Mantissa = %u\n", Float.Format.Mantissa);
  return 0;
}

```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

13

## Data Structures

- Example: `Bitfield.c`

```

% gcc Bitfield.c -o Bitfield -Wall -ansi
% ./Bitfield
sizeof(float) = 4
sizeof(Float) = 4
Float.Value = -1.000000
Float.Format.Sign = 1
Float.Format.Exponent = 127
Float.Format.Mantissa = 0
%

```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

14

## Data Structures

- Enumerators: `enum`
  - 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; }
```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

15

## 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 */
= Monday;              /* initialization */
```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

16



## 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

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

enum Weekday Today
= Monday;

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

- Example:

Today

Monday

Day: 0

## 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

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

enum Weekday Today
= Monday;

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

- Example:

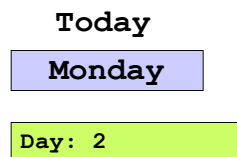
Today

Monday

Day: 1

## 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:



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

enum Weekday Today
= Monday;

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

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

19

## 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
    - Syntactically, **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 /* digital image type */
{ unsigned int Width, Height;
  unsigned char R[640*480], G[640*480], B[640*480];
} IMAGE;
```

EECS22: Advanced C Programming, Lecture 15

(c) 2017 R. Doemer

20