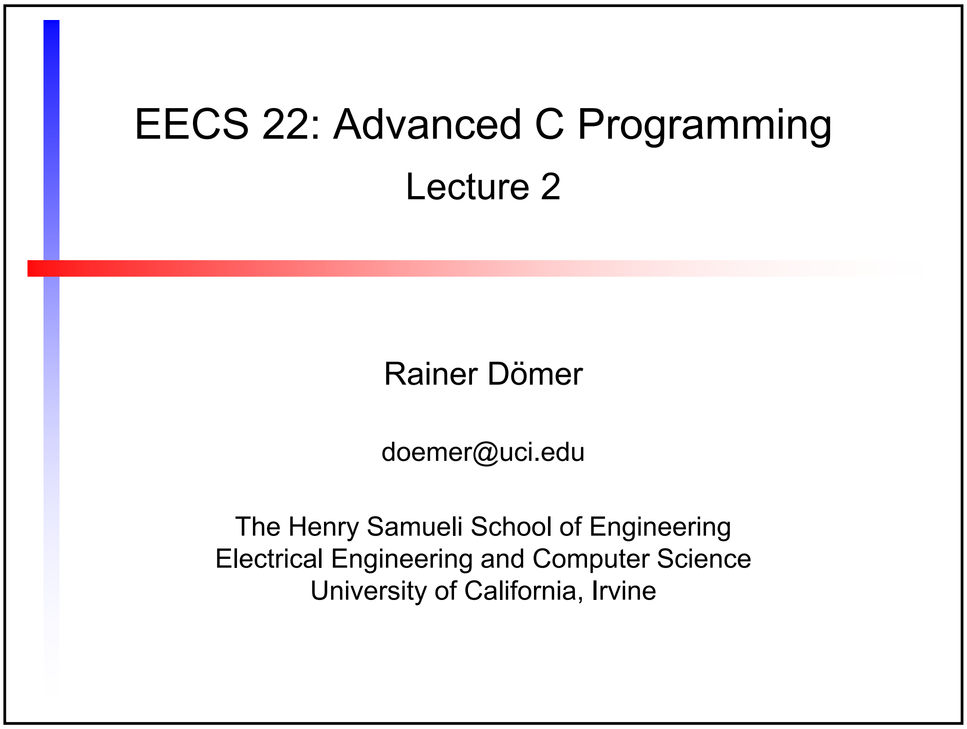


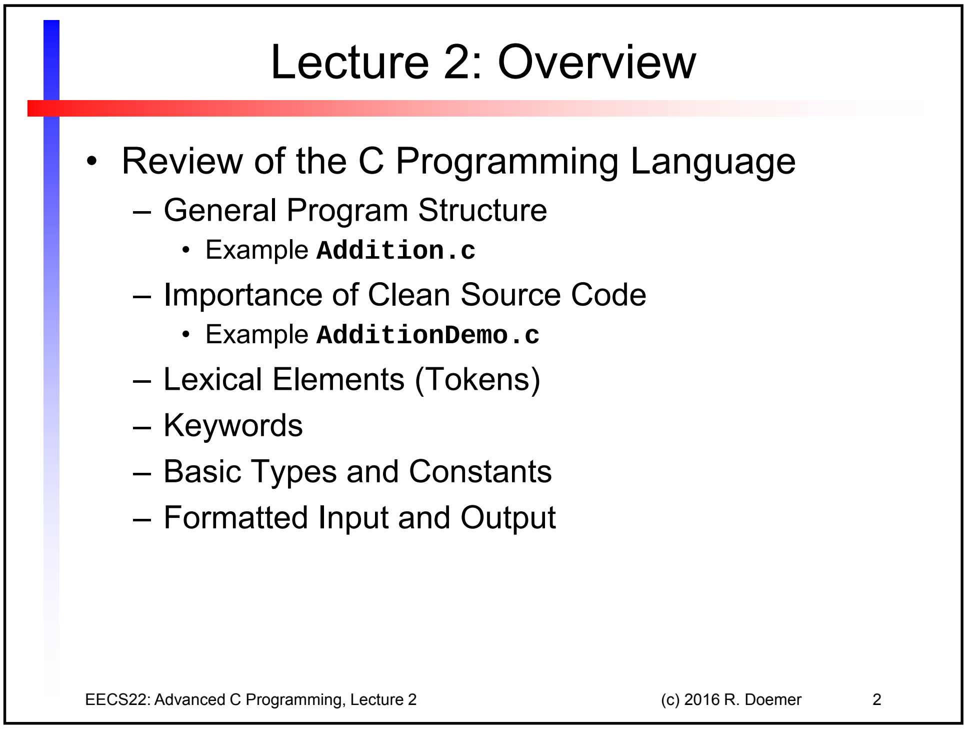
EECS 22: Advanced C Programming

Lecture 2

Rainer Dömer

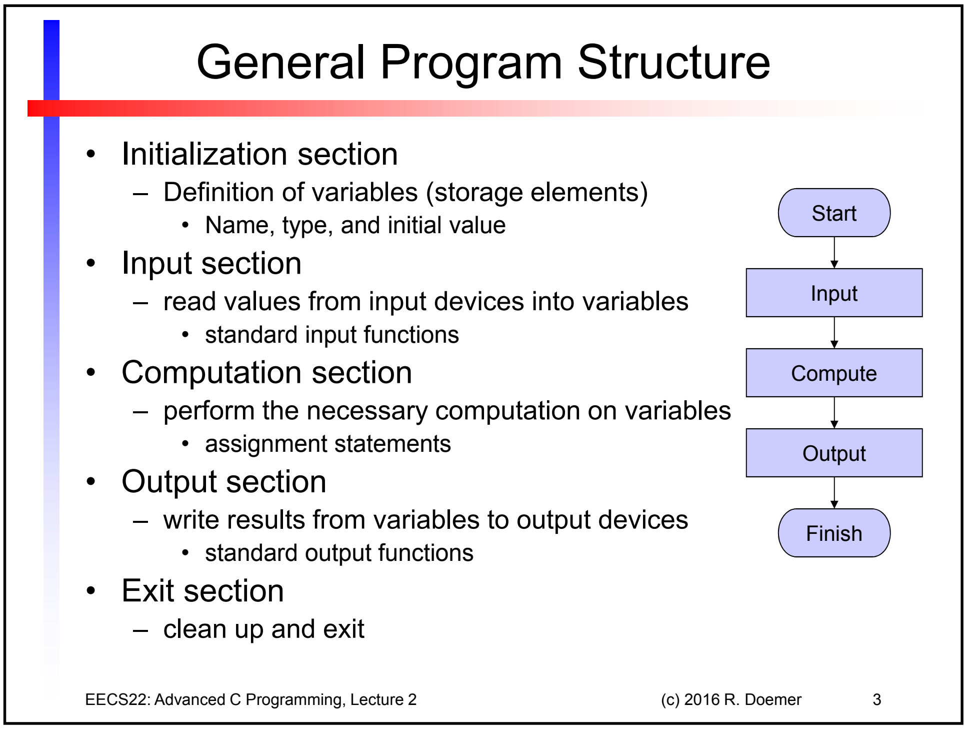
doemer@uci.edu

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



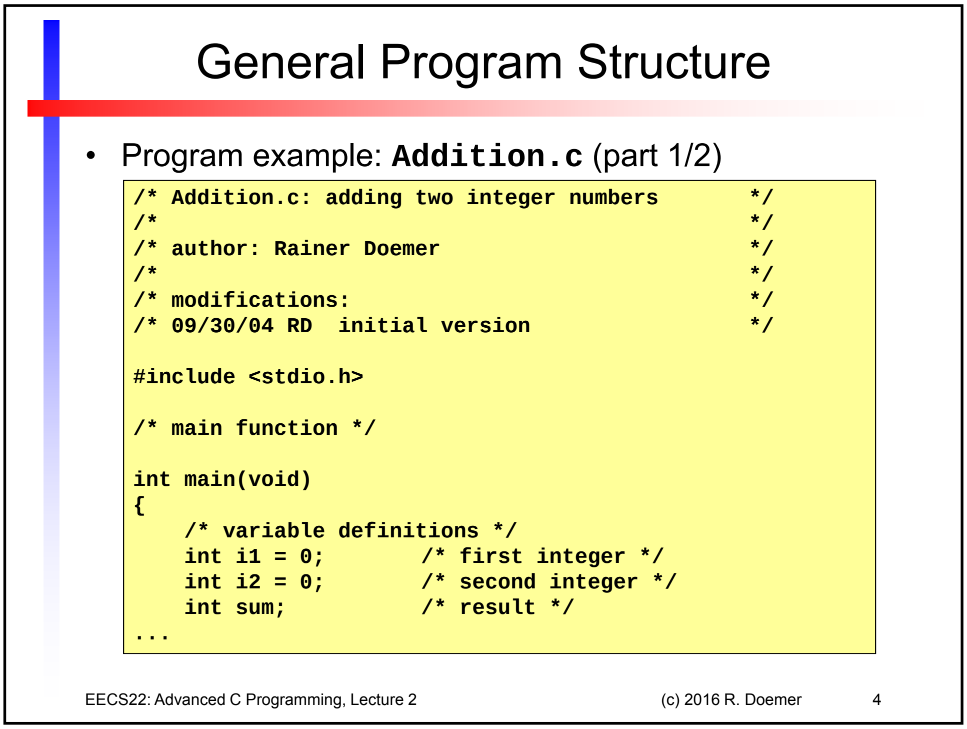
Lecture 2: Overview

- Review of the C Programming Language
  - General Program Structure
    - Example **`Addition.c`**
  - Importance of Clean Source Code
    - Example **`AdditionDemo.c`**
  - Lexical Elements (Tokens)
  - Keywords
  - Basic Types and Constants
  - Formatted Input and Output



General Program Structure

- Initialization section
  - Definition of variables (storage elements)
    - Name, type, and initial value
- Input section
  - read values from input devices into variables
    - standard input functions
- Computation section
  - perform the necessary computation on variables
    - assignment statements
- Output section
  - write results from variables to output devices
    - standard output functions
- Exit section
  - clean up and exit



General Program Structure

- Program example: **`Addition.c`** (part 1/2)

```
/* Addition.c: adding two integer numbers       */
/*                                              */
/* author: Rainer Doemer                        */
/*                                              */
/* modifications:                               */
/* 09/30/04 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int i1 = 0;        /* first integer */
    int i2 = 0;        /* second integer */
    int sum;           /* result */
...
```

General Program Structure

- Program example: **`Addition.c`** (part 2/2)

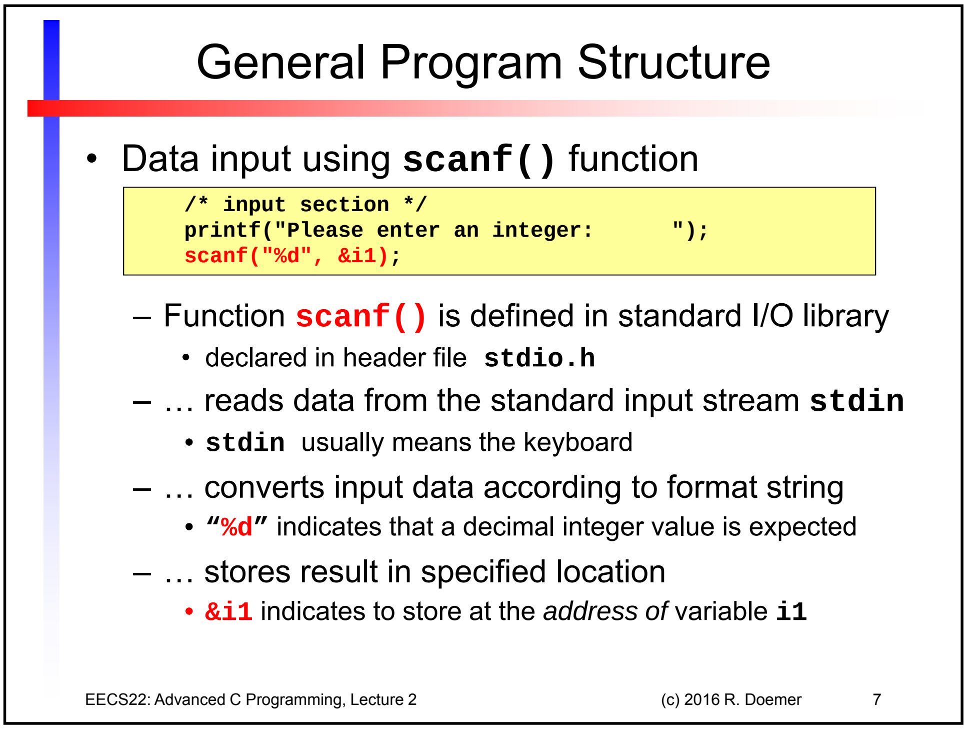
```
...
    /* input section */
    printf("Please enter an integer:      ");
    scanf("%d", &i1);
    printf("Please enter another integer: ");
    scanf("%d", &i2);

    /* computation section */
    sum = i1 + i2;

    /* output section */
    printf("The sum of %d and %d is %d.\n", i1, i2, sum);

    /* exit */
    return 0;
} /* end of main */

/* EOF */
```

General Program Structure

- Variable definition and initialization

```
    /* variable definitions */
    int i1 = 0;         /* first integer */
    int i2 = 0;         /* second integer */
    int sum;            /* result */
```

  - Variable type: `int`
    - integer type, stores whole numbers (e.g. -5, 0, 42)
    - many other types exist (**`float`**, **`double`**, **`char`**, ...)
  - Variable name: `i1`
    - valid identifier, i.e. name composed of letters, digits
    - variable name should be descriptive
  - Initializer: `= 0`
    - specifies the initial value of the variable
    - optional (if omitted, initial value is undefined)

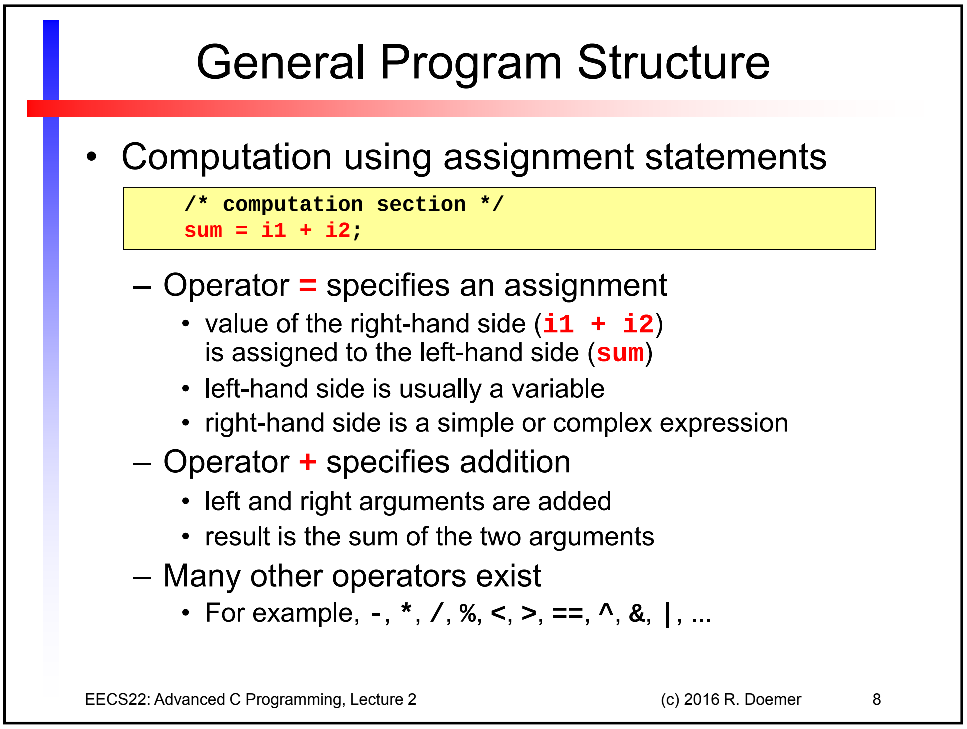


General Program Structure

- Data input using **scanf()** function

```
/* input section */
printf("Please enter an integer:      ");
scanf("%d", &i1);
```

- Function **scanf()** is defined in standard I/O library
  - declared in header file **stdio.h**
- … reads data from the standard input stream **stdin**
  - **stdin** usually means the keyboard
- … converts input data according to format string
  - **"%d"** indicates that a decimal integer value is expected
- … stores result in specified location
  - **&i1** indicates to store at the *address of* variable **i1**

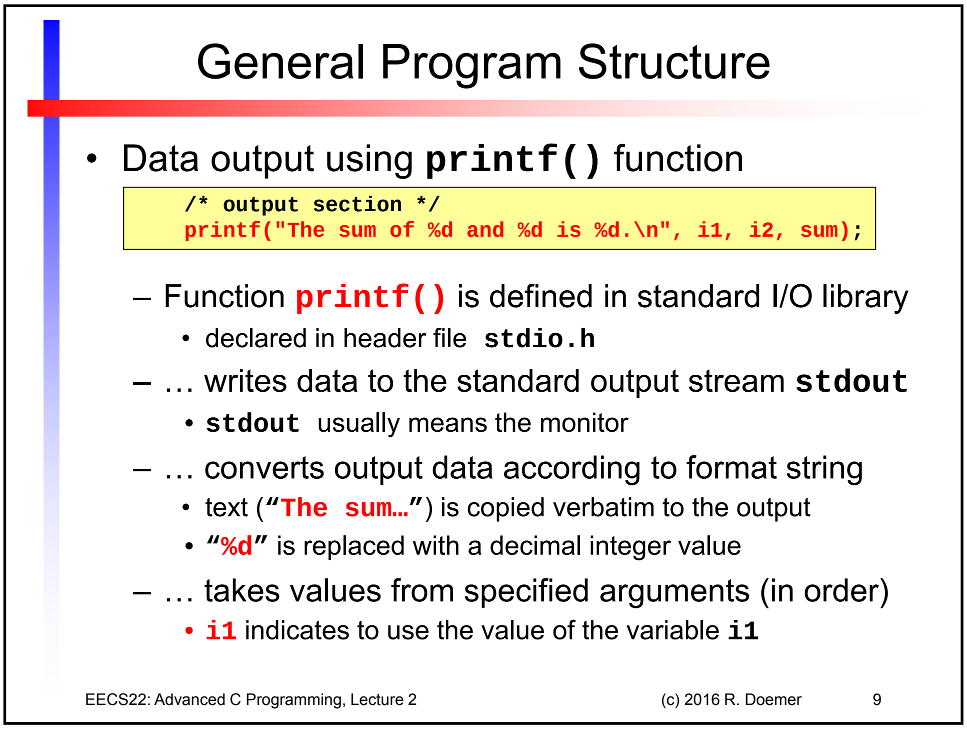


General Program Structure

- Computation using assignment statements

```
/* computation section */
sum = i1 + i2;
```

- Operator **=** specifies an assignment
  - value of the right-hand side (**i1 + i2**) is assigned to the left-hand side (**sum**)
  - left-hand side is usually a variable
  - right-hand side is a simple or complex expression
- Operator **+** specifies addition
  - left and right arguments are added
  - result is the sum of the two arguments
- Many other operators exist
  - For example, **-**, *****, **/**, **%**, **<**, **>**, **==**, **^**, **&**, **|**, ...

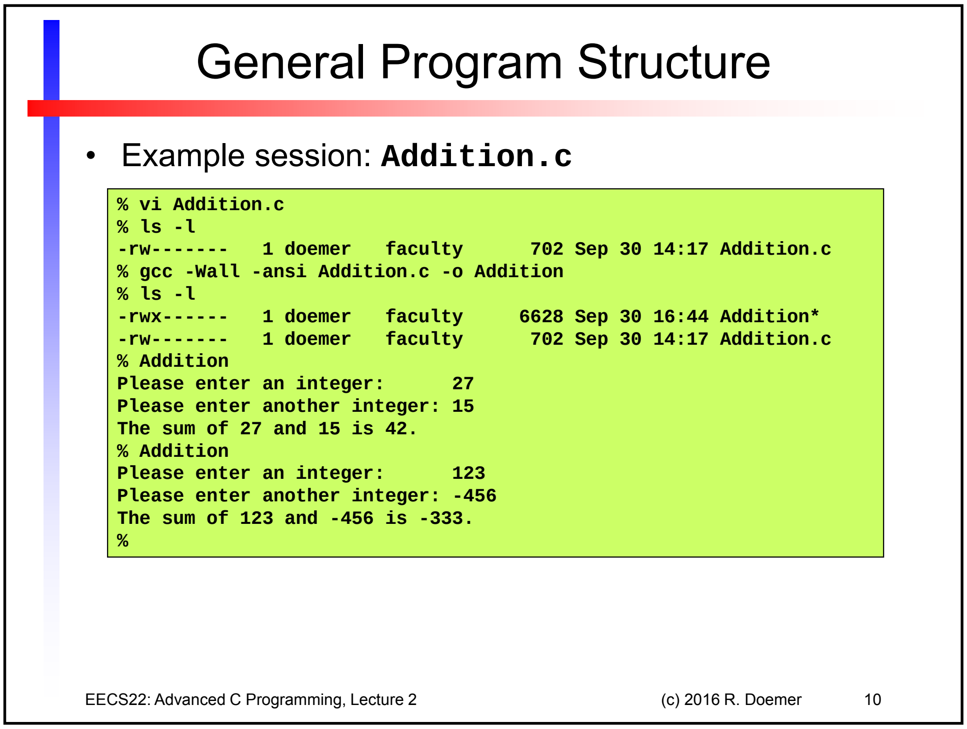


General Program Structure

- Data output using **`printf()`** function

```
/* output section */
printf("The sum of %d and %d is %d.\n", i1, i2, sum);
```

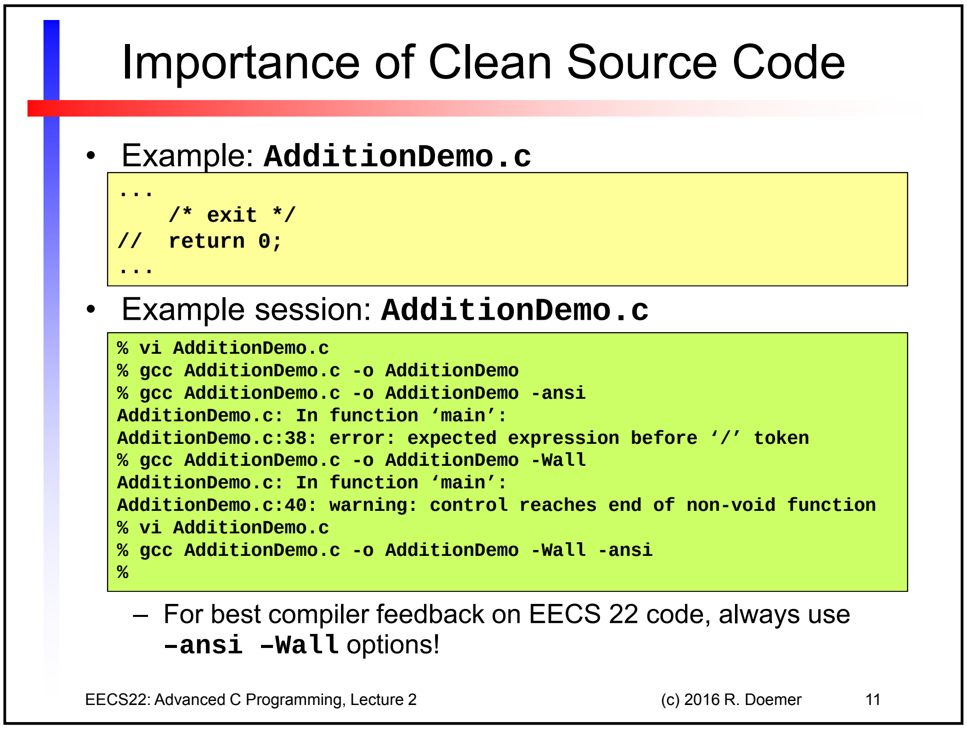
- Function **`printf()`** is defined in standard I/O library
  - declared in header file **`stdio.h`**
- … writes data to the standard output stream **`stdout`**
  - **`stdout`** usually means the monitor
- … converts output data according to format string
  - text (**“The sum…”**) is copied verbatim to the output
  - **“%d”** is replaced with a decimal integer value
- … takes values from specified arguments (in order)
  - **`i1`** indicates to use the value of the variable **`i1`**



General Program Structure

- Example session: **`Addition.c`**

```
% vi Addition.c
% ls -l
-rw-------   1 doemer   faculty      702 Sep 30 14:17 Addition.c
% gcc -Wall -ansi Addition.c -o Addition
% ls -l
-rwx------   1 doemer   faculty     6628 Sep 30 16:44 Addition*
-rw-------   1 doemer   faculty      702 Sep 30 14:17 Addition.c
% Addition
Please enter an integer:        27
Please enter another integer: 15
The sum of 27 and 15 is 42.
% Addition
Please enter an integer:        123
Please enter another integer: -456
The sum of 123 and -456 is -333.
%
```



Importance of Clean Source Code

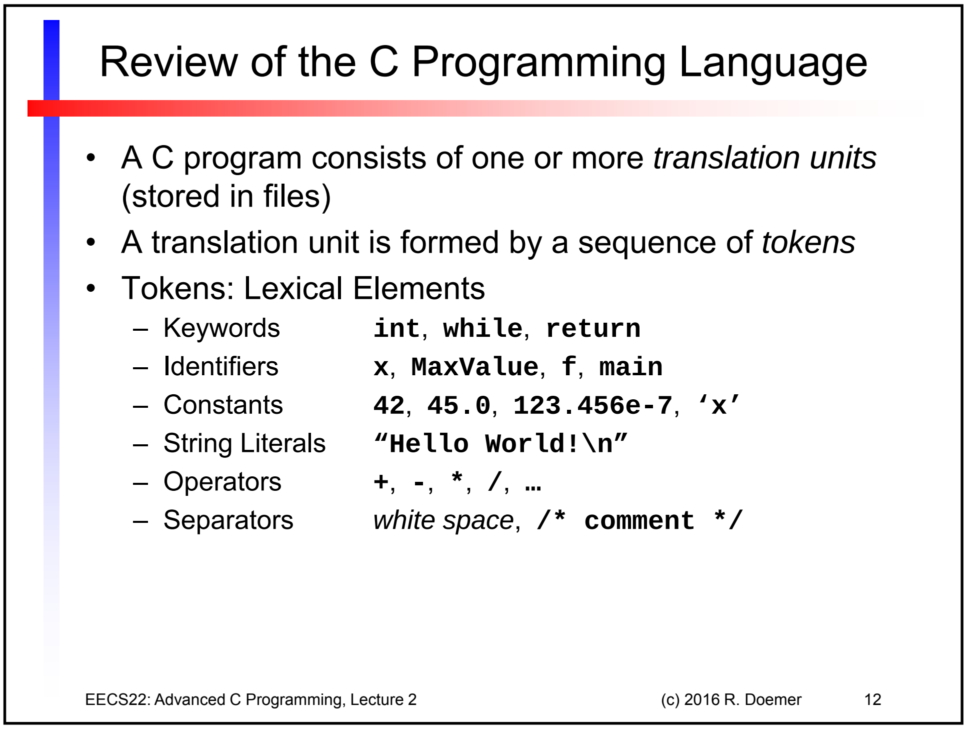
- Example: **`AdditionDemo.c`**

```
...
    /* exit */
//  return 0;
...
```

- Example session: **`AdditionDemo.c`**

```
% vi AdditionDemo.c
% gcc AdditionDemo.c -o AdditionDemo
% gcc AdditionDemo.c -o AdditionDemo -ansi
AdditionDemo.c: In function ‘main’:
AdditionDemo.c:38: error: expected expression before ‘/’ token
% gcc AdditionDemo.c -o AdditionDemo -Wall
AdditionDemo.c: In function ‘main’:
AdditionDemo.c:40: warning: control reaches end of non-void function
% vi AdditionDemo.c
% gcc AdditionDemo.c -o AdditionDemo -Wall -ansi
%
```

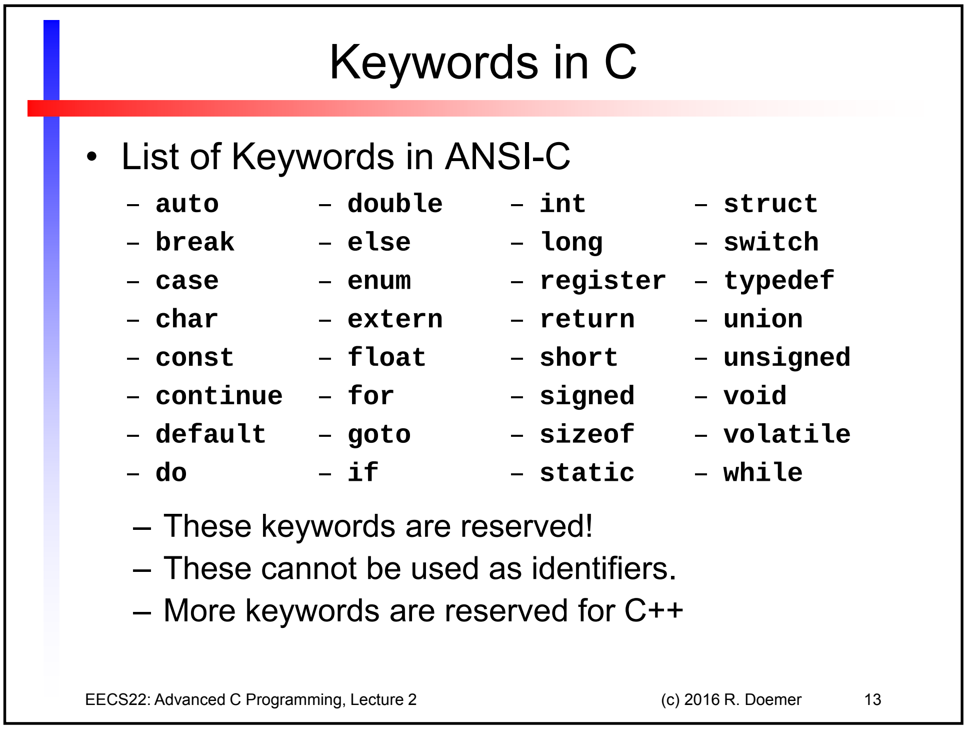
  - For best compiler feedback on EECS 22 code, always use **`-ansi -Wall`** options!



Review of the C Programming Language

- A C program consists of one or more *translation units* (stored in files)
- A translation unit is formed by a sequence of *tokens*
- Tokens: Lexical Elements

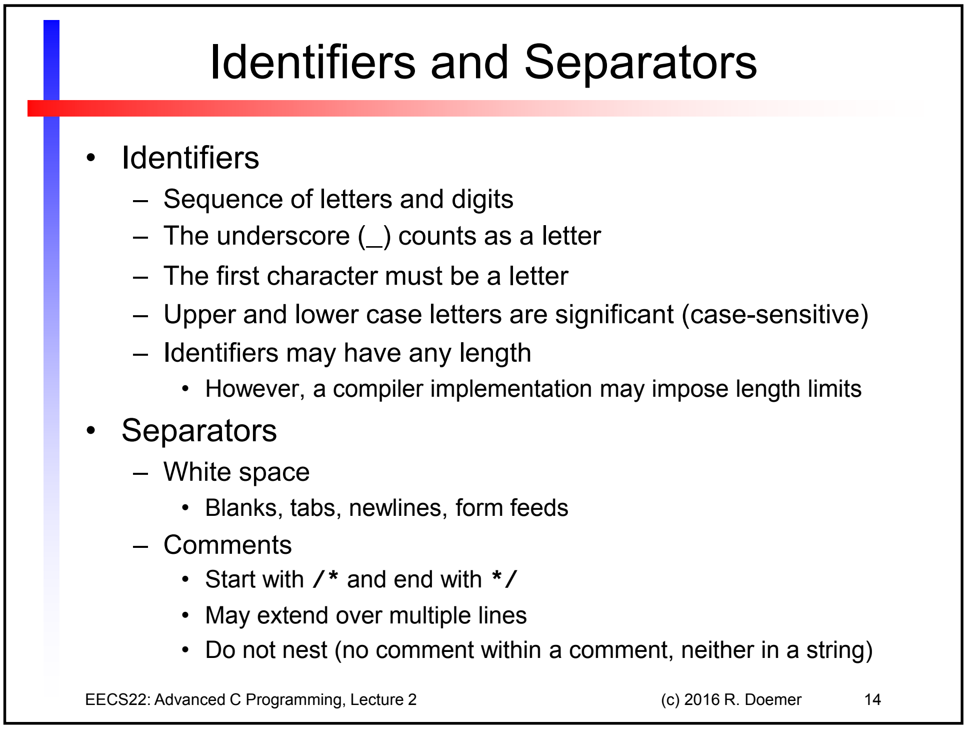
– Keywords	**`int`**, **`while`**, **`return`**
– Identifiers	**`x`**, **`MaxValue`**, **`f`**, **`main`**
– Constants	**`42`**, **`45.0`**, **`123.456e-7`**, **`‘x’`**
– String Literals	**`“Hello World!\n”`**
– Operators	**`+`**, **`-`**, **`*`**, **`/`**, **…**
– Separators	*white space*, **`/* comment */`**



Keywords in C

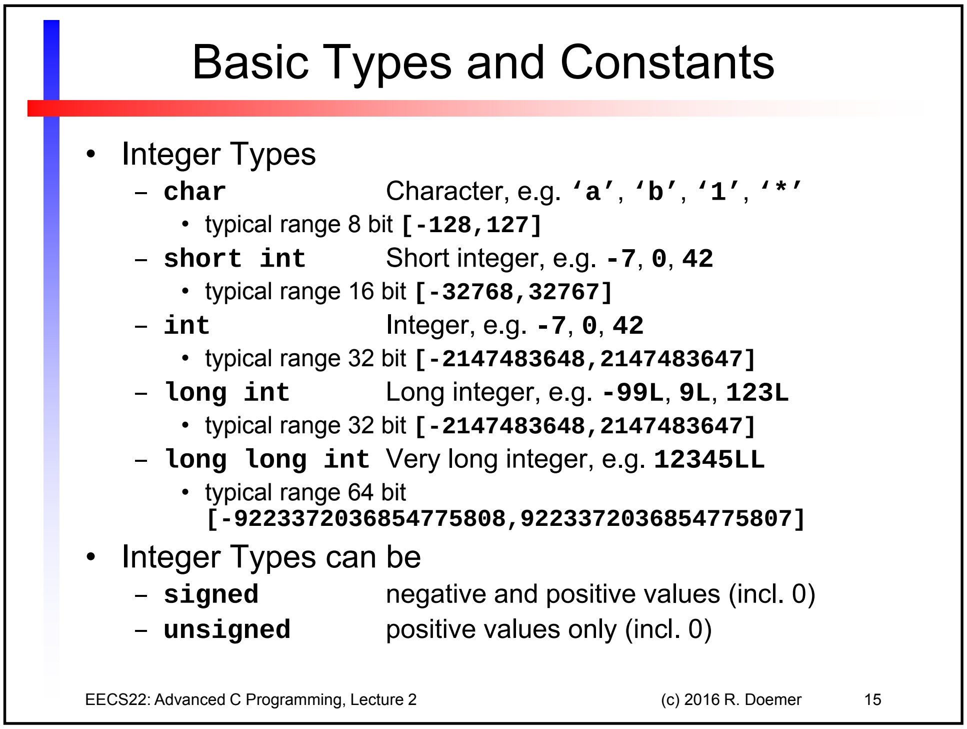
- List of Keywords in ANSI-C
  - `auto`
  - `break`
  - `case`
  - `char`
  - `const`
  - `continue`
  - `default`
  - `do`
  - `double`
  - `else`
  - `enum`
  - `extern`
  - `float`
  - `for`
  - `goto`
  - `if`
  - `int`
  - `long`
  - `register`
  - `return`
  - `short`
  - `signed`
  - `sizeof`
  - `static`
  - `struct`
  - `switch`
  - `typedef`
  - `union`
  - `unsigned`
  - `void`
  - `volatile`
  - `while`

  - These keywords are reserved!
  - These cannot be used as identifiers.
  - More keywords are reserved for C++



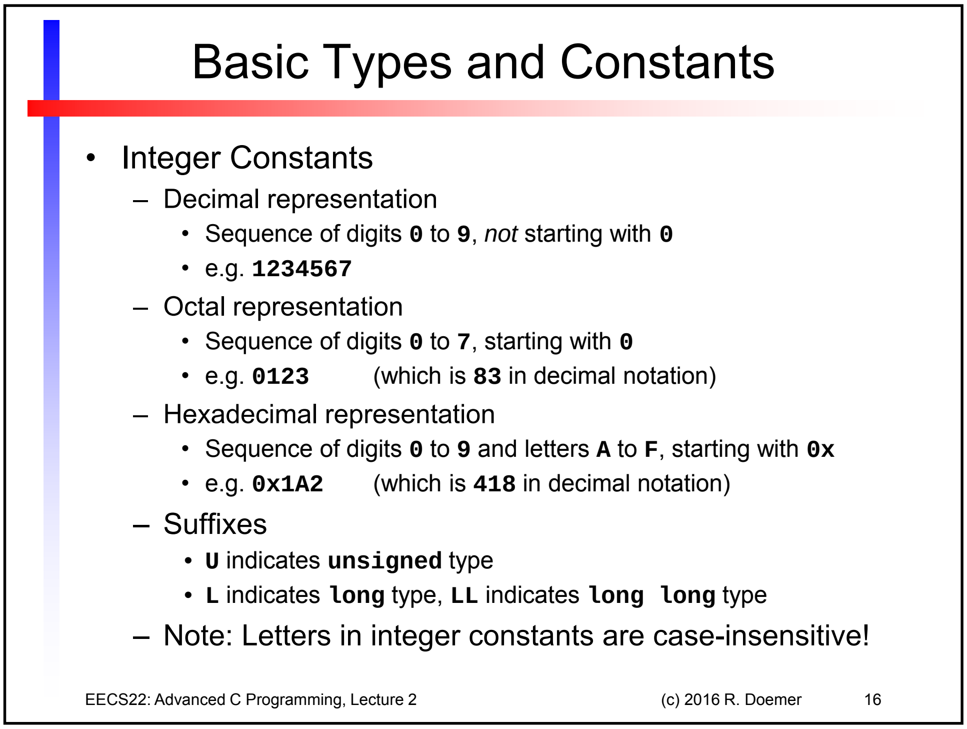
Identifiers and Separators

- Identifiers
  - Sequence of letters and digits
  - The underscore (_) counts as a letter
  - The first character must be a letter
  - Upper and lower case letters are significant (case-sensitive)
  - Identifiers may have any length
    - However, a compiler implementation may impose length limits
- Separators
  - White space
    - Blanks, tabs, newlines, form feeds
  - Comments
    - Start with `/*` and end with `*/`
    - May extend over multiple lines
    - Do not nest (no comment within a comment, neither in a string)



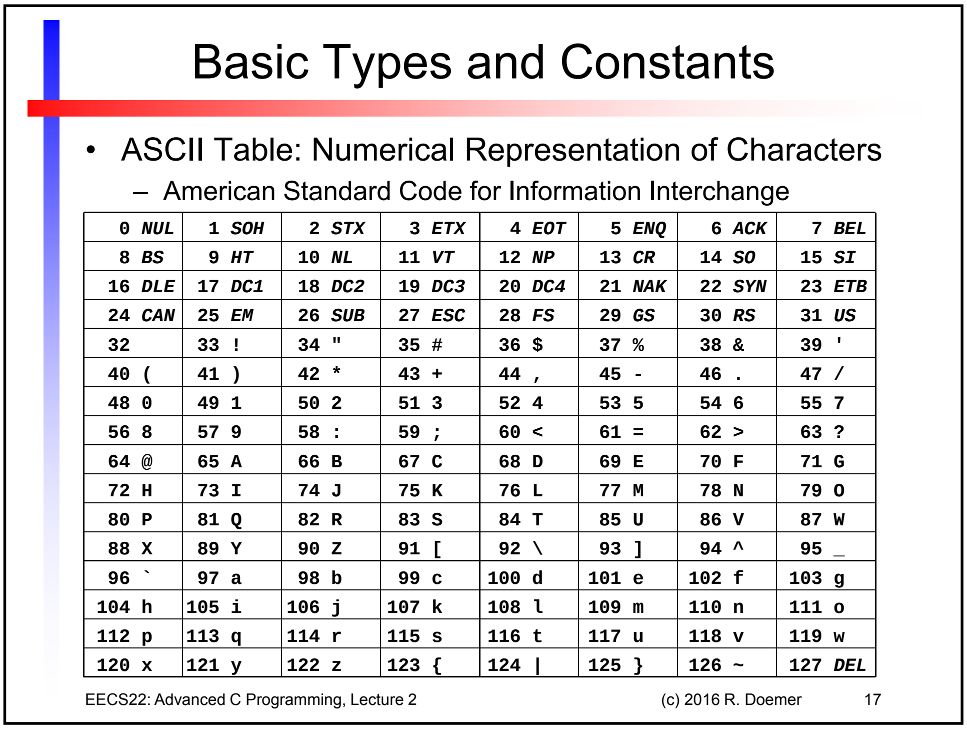
Basic Types and Constants

- Integer Types
  - **`char`** Character, e.g. **`'a'`**, **`'b'`**, **`'1'`**, **`'*'`**
    - typical range 8 bit **`[-128,127]`**
  - **`short int`** Short integer, e.g. **`-7`**, **`0`**, **`42`**
    - typical range 16 bit **`[-32768,32767]`**
  - **`int`** Integer, e.g. **`-7`**, **`0`**, **`42`**
    - typical range 32 bit **`[-2147483648,2147483647]`**
  - **`long int`** Long integer, e.g. **`-99L`**, **`9L`**, **`123L`**
    - typical range 32 bit **`[-2147483648,2147483647]`**
  - **`long long int`** Very long integer, e.g. **`12345LL`**
    - typical range 64 bit **`[-9223372036854775808,9223372036854775807]`**
- Integer Types can be
  - **`signed`** negative and positive values (incl. 0)
  - **`unsigned`** positive values only (incl. 0)



Basic Types and Constants

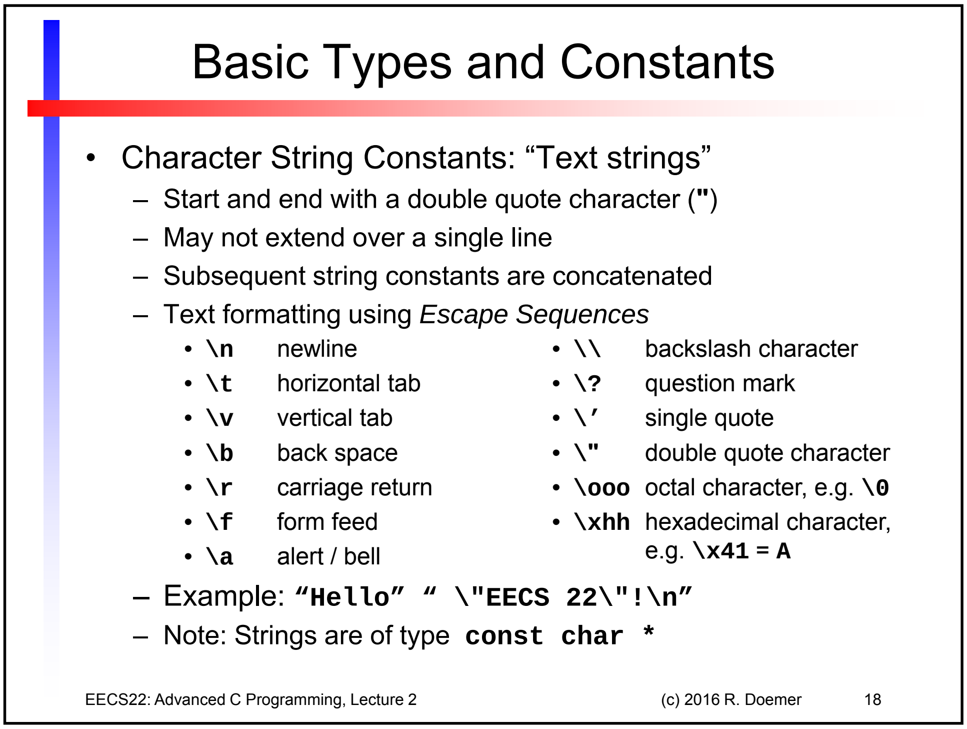
- Integer Constants
  - Decimal representation
    - Sequence of digits **`0`** to **`9`**, *not* starting with **`0`**
    - e.g. **`1234567`**
  - Octal representation
    - Sequence of digits **`0`** to **`7`**, starting with **`0`**
    - e.g. **`0123`** (which is **`83`** in decimal notation)
  - Hexadecimal representation
    - Sequence of digits **`0`** to **`9`** and letters **`A`** to **`F`**, starting with **`0x`**
    - e.g. **`0x1A2`** (which is **`418`** in decimal notation)
  - Suffixes
    - **`U`** indicates **`unsigned`** type
    - **`L`** indicates **`long`** type, **`LL`** indicates **`long long`** type
  - Note: Letters in integer constants are case-insensitive!



Basic Types and Constants

- ASCII Table: Numerical Representation of Characters
  - American Standard Code for Information Interchange

0 *NUL*	1 *SOH*	2 *STX*	3 *ETX*	4 *EOT*	5 *ENQ*	6 *ACK*	7 *BEL*
8 *BS*	9 *HT*	10 *NL*	11 *VT*	12 *NP*	13 *CR*	14 *SO*	15 *SI*
16 *DLE*	17 *DC1*	18 *DC2*	19 *DC3*	20 *DC4*	21 *NAK*	22 *SYN*	23 *ETB*
24 *CAN*	25 *EM*	26 *SUB*	27 *ESC*	28 *FS*	29 *GS*	30 *RS*	31 *US*
32	33 !	34 "	35 #	36 $	37 %	38 &	39 '
40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 *DEL*

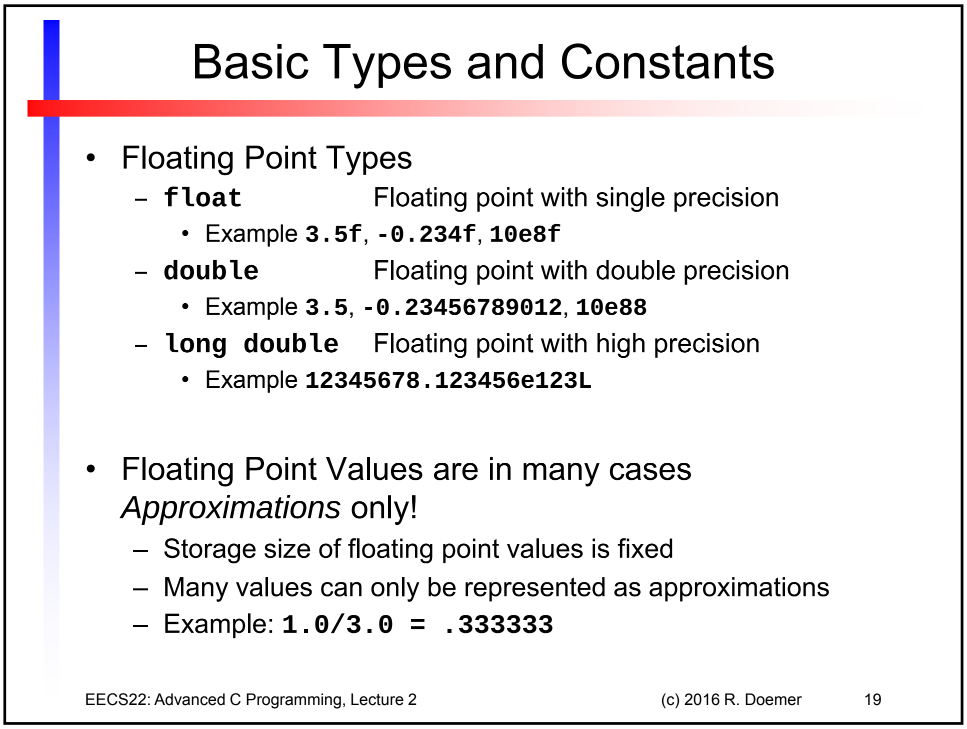


Basic Types and Constants

- Character String Constants: “Text strings”
  - Start and end with a double quote character (**"**)
  - May not extend over a single line
  - Subsequent string constants are concatenated
  - Text formatting using *Escape Sequences*

\n	newline	\\	backslash character
\t	horizontal tab	**\?**	question mark
\v	vertical tab	**\’**	single quote
\b	back space	**\"**	double quote character
\r	carriage return	**\ooo**	octal character, e.g. **\0**
\f	form feed	**\xhh**	hexadecimal character, e.g. **\x41** = **A**
\a	alert / bell		

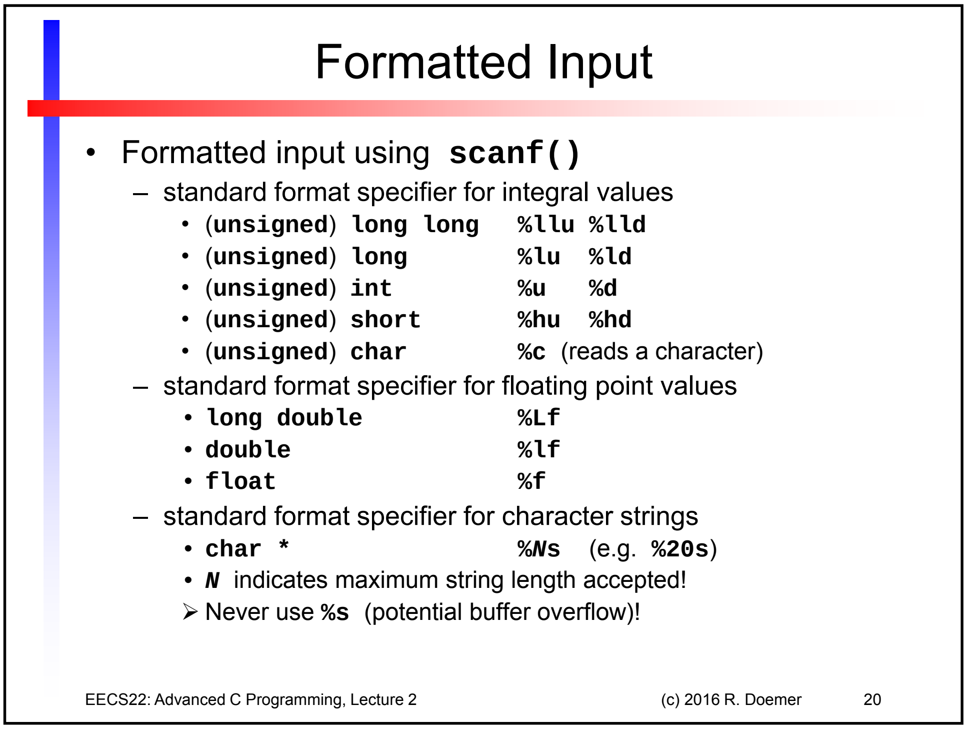
  - Example: **“Hello” “ \"EECS 22\"!\n”**
  - Note: Strings are of type **const char ***



Basic Types and Constants

- Floating Point Types
  - **`float`** Floating point with single precision
    - Example **`3.5f`**, **`-0.234f`**, **`10e8f`**
  - **`double`** Floating point with double precision
    - Example **`3.5`**, **`-0.23456789012`**, **`10e88`**
  - **`long double`** Floating point with high precision
    - Example **`12345678.123456e123L`**

- Floating Point Values are in many cases *Approximations* only!
  - Storage size of floating point values is fixed
  - Many values can only be represented as approximations
  - Example: **`1.0/3.0 = .333333`**



Formatted Input

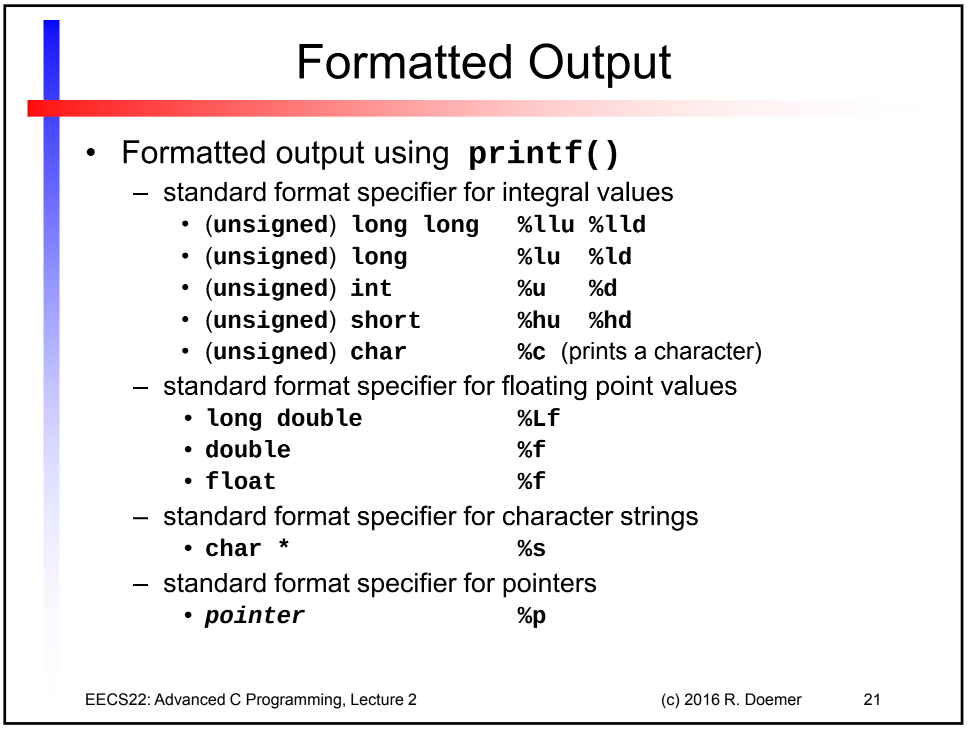
- Formatted input using **`scanf()`**
  - standard format specifier for integral values

(**`unsigned`**) **`long long`**	**`%llu %lld`**
(**`unsigned`**) **`long`**	**`%lu %ld`**
(**`unsigned`**) **`int`**	**`%u %d`**
(**`unsigned`**) **`short`**	**`%hu %hd`**
(**`unsigned`**) **`char`**	**`%c`** (reads a character)

  - standard format specifier for floating point values

`long double`	**`%Lf`**
`double`	**`%lf`**
`float`	**`%f`**

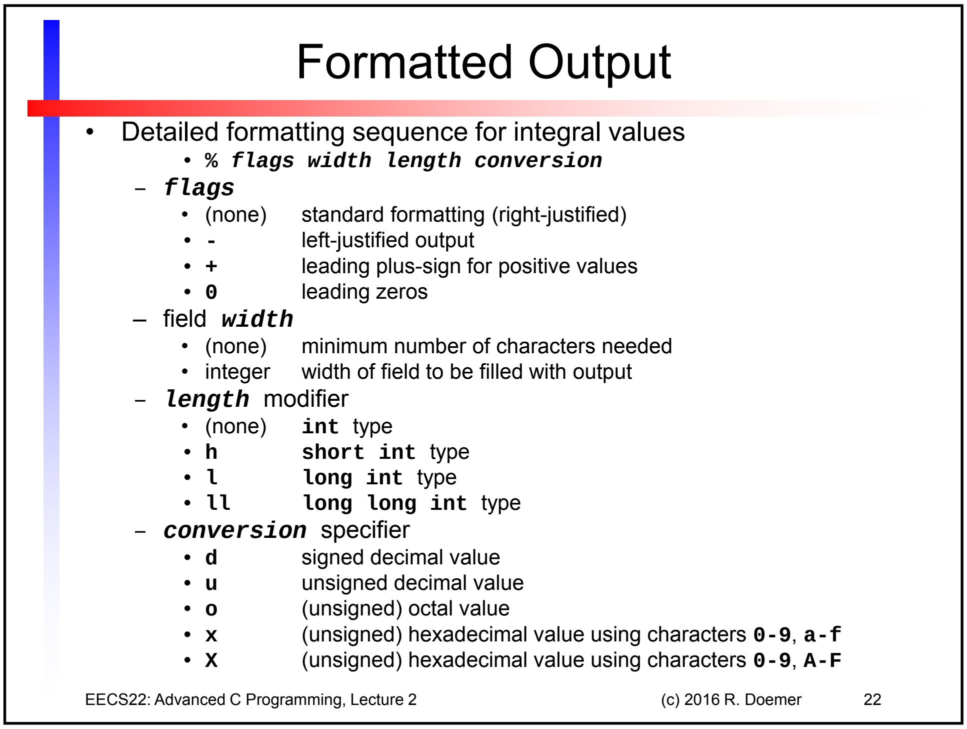
  - standard format specifier for character strings
    - **`char *`** **`%`*****`N`*****`s`** (e.g. **`%20s`**)
    - ***`N`*** indicates maximum string length accepted!
    - ➢ Never use **`%s`** (potential buffer overflow)!



Formatted Output

- Formatted output using **`printf()`**
  - standard format specifier for integral values
    - (**`unsigned`**) **`long long`** **`%llu %lld`**
    - (**`unsigned`**) **`long`** **`%lu %ld`**
    - (**`unsigned`**) **`int`** **`%u %d`**
    - (**`unsigned`**) **`short`** **`%hu %hd`**
    - (**`unsigned`**) **`char`** **`%c`** (prints a character)
  - standard format specifier for floating point values
    - **`long double`** **`%Lf`**
    - **`double`** **`%f`**
    - **`float`** **`%f`**
  - standard format specifier for character strings
    - **`char *`** **`%s`**
  - standard format specifier for pointers
    - ***`pointer`*** **`%p`**

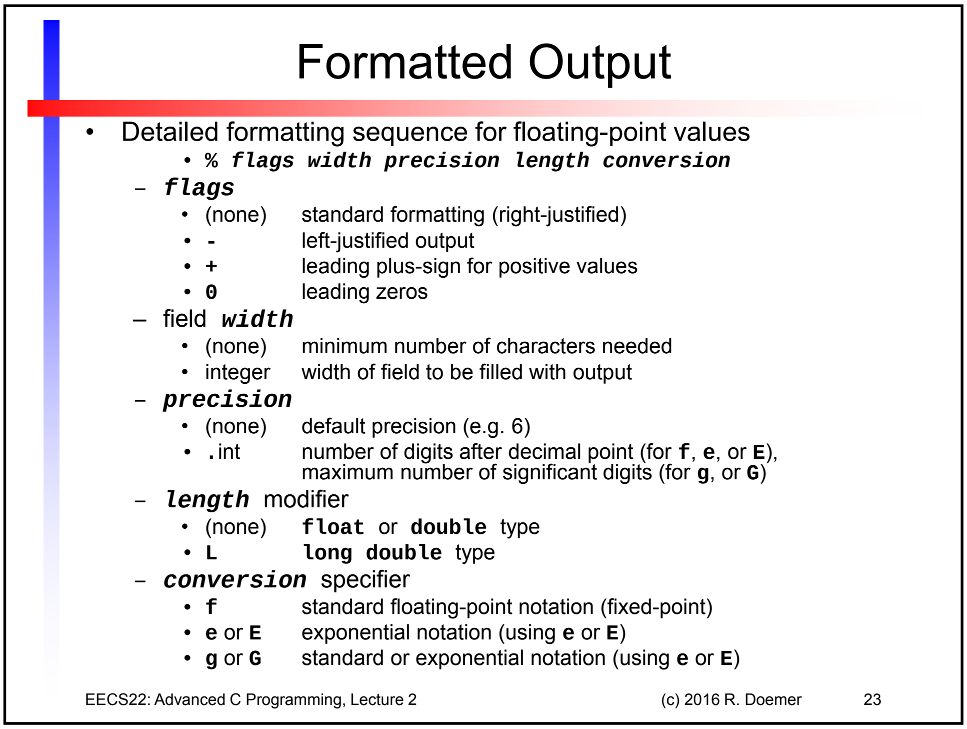
EECS22: Advanced C Programming, Lecture 2 (c) 2016 R. Doemer 21



Formatted Output

- Detailed formatting sequence for integral values
    - **`%`** ***`flags width length conversion`***
  - ***`flags`***
    - (none) standard formatting (right-justified)
    - **`-`** left-justified output
    - **`+`** leading plus-sign for positive values
    - **`0`** leading zeros
  - field ***`width`***
    - (none) minimum number of characters needed
    - integer width of field to be filled with output
  - ***`length`*** modifier
    - (none) **`int`** type
    - **`h`** **`short int`** type
    - **`l`** **`long int`** type
    - **`ll`** **`long long int`** type
  - ***`conversion`*** specifier
    - **`d`** signed decimal value
    - **`u`** unsigned decimal value
    - **`o`** (unsigned) octal value
    - **`x`** (unsigned) hexadecimal value using characters **`0-9`**, **`a-f`**
    - **`X`** (unsigned) hexadecimal value using characters **`0-9`**, **`A-F`**

EECS22: Advanced C Programming, Lecture 2 (c) 2016 R. Doemer 22



Formatted Output

- Detailed formatting sequence for floating-point values
    - `% flags width precision length conversion`
  - ***flags***

(none)	standard formatting (right-justified)
-	left-justified output
+	leading plus-sign for positive values
0	leading zeros

  - field ***width***

(none)	minimum number of characters needed
integer	width of field to be filled with output

  - ***precision***

(none)	default precision (e.g. 6)
.int	number of digits after decimal point (for **f**, **e**, or **E**), maximum number of significant digits (for **g**, or **G**)

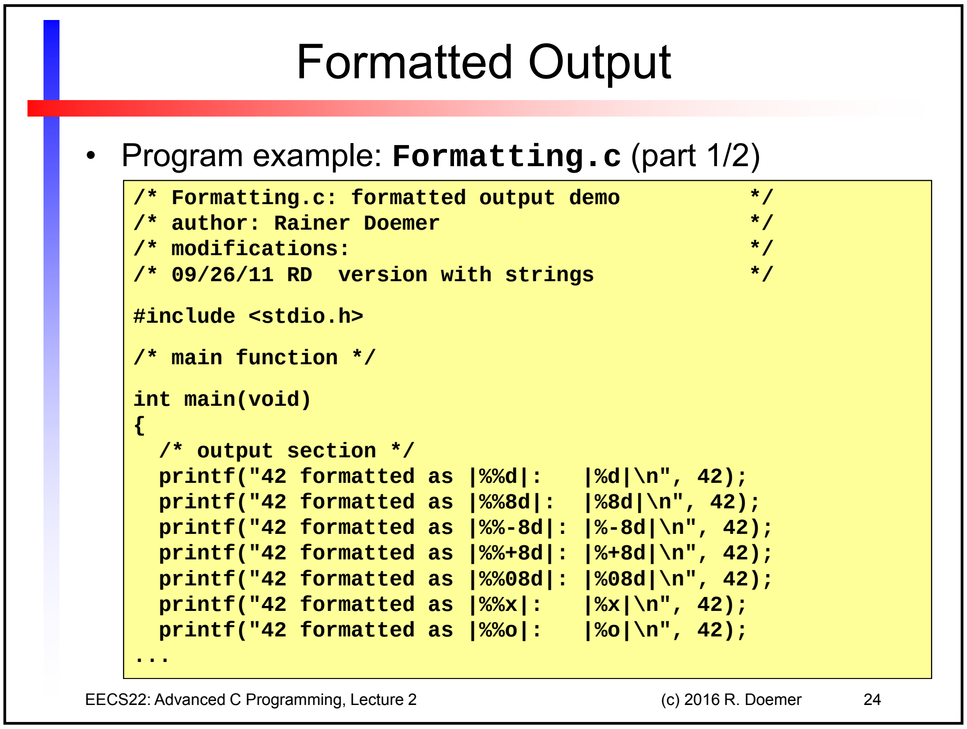
  - ***length*** modifier

(none)	**float** or **double** type
L	**long double** type

  - ***conversion*** specifier

f	standard floating-point notation (fixed-point)
e or **E**	exponential notation (using **e** or **E**)
g or **G**	standard or exponential notation (using **e** or **E**)

EECS22: Advanced C Programming, Lecture 2 (c) 2016 R. Doemer 23

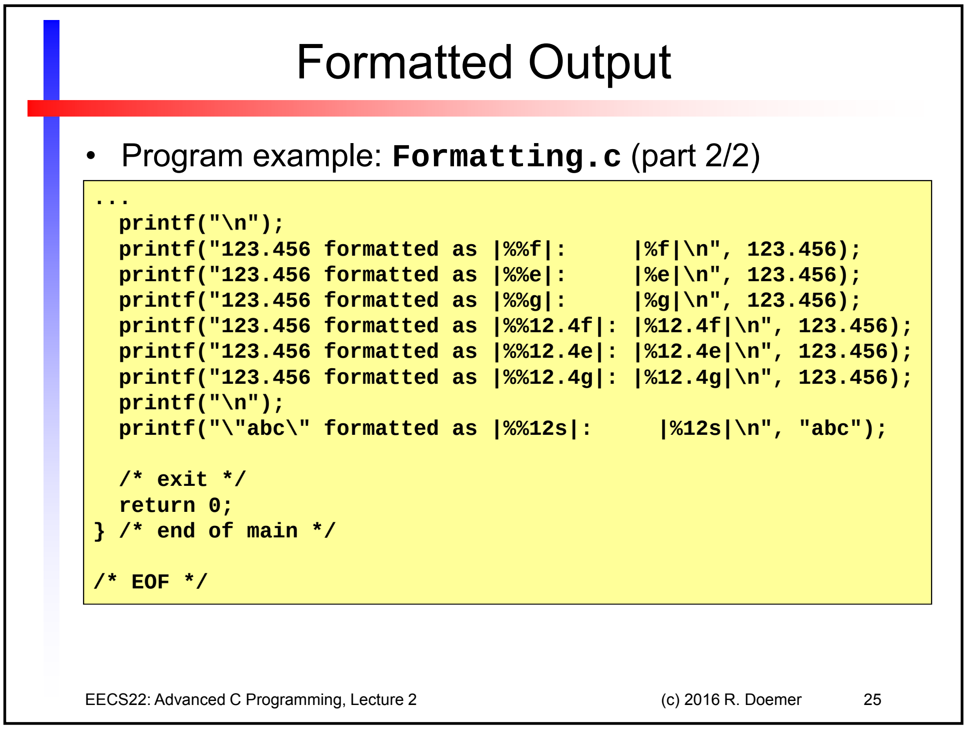


Formatted Output

- Program example: **Formatting.c** (part 1/2)

```
/* Formatting.c: formatted output demo          */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 09/26/11 RD  version with strings            */

#include <stdio.h>

/* main function */

int main(void)
{
  /* output section */
  printf("42 formatted as |%%d|:   |%d|\n", 42);
  printf("42 formatted as |%%8d|:  |%8d|\n", 42);
  printf("42 formatted as |%%-8d|: |%-8d|\n", 42);
  printf("42 formatted as |%%+8d|: |%+8d|\n", 42);
  printf("42 formatted as |%%08d|: |%08d|\n", 42);
  printf("42 formatted as |%%x|:   |%x|\n", 42);
  printf("42 formatted as |%%o|:   |%o|\n", 42);
...
```

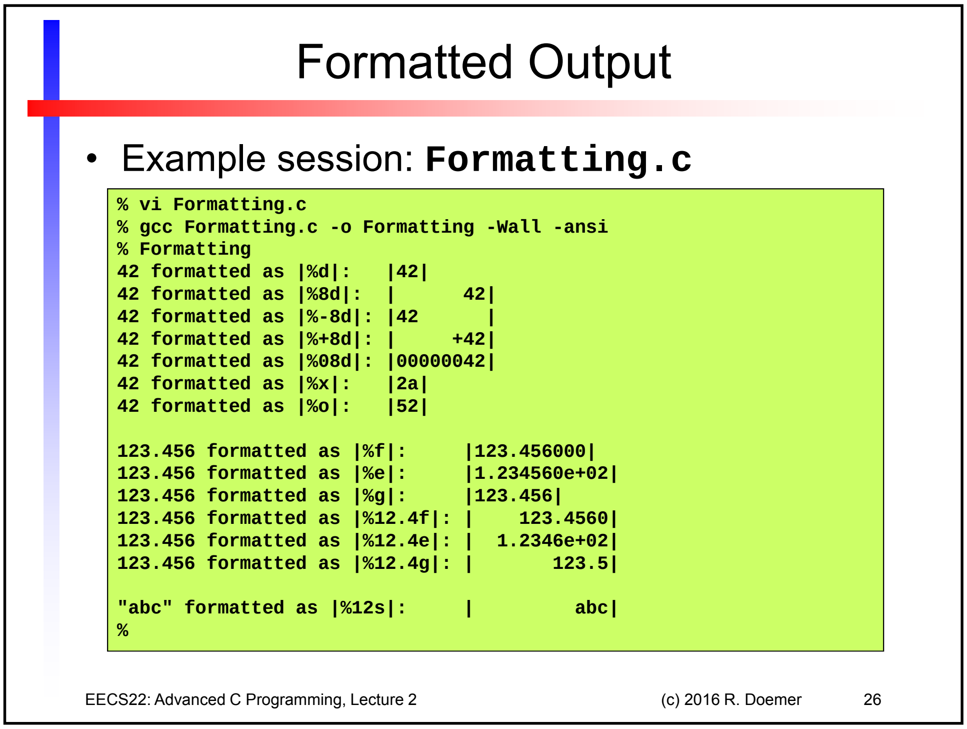
EECS22: Advanced C Programming, Lecture 2 (c) 2016 R. Doemer 24



Formatted Output

- Program example: **`Formatting.c`** (part 2/2)

```
...
  printf("\n");
  printf("123.456 formatted as |%%f|:     |%f|\n", 123.456);
  printf("123.456 formatted as |%%e|:     |%e|\n", 123.456);
  printf("123.456 formatted as |%%g|:     |%g|\n", 123.456);
  printf("123.456 formatted as |%%12.4f|: |%12.4f|\n", 123.456);
  printf("123.456 formatted as |%%12.4e|: |%12.4e|\n", 123.456);
  printf("123.456 formatted as |%%12.4g|: |%12.4g|\n", 123.456);
  printf("\n");
  printf("\"abc\" formatted as |%%12s|:     |%12s|\n", "abc");

  /* exit */
  return 0;
} /* end of main */

/* EOF */
```



Formatted Output

- Example session: **`Formatting.c`**

```
% vi Formatting.c
% gcc Formatting.c -o Formatting -Wall -ansi
% Formatting
42 formatted as |%d|:   |42|
42 formatted as |%8d|:  |      42|
42 formatted as |%-8d|: |42      |
42 formatted as |%+8d|: |     +42|
42 formatted as |%08d|: |00000042|
42 formatted as |%x|:   |2a|
42 formatted as |%o|:   |52|

123.456 formatted as |%f|:     |123.456000|
123.456 formatted as |%e|:     |1.234560e+02|
123.456 formatted as |%g|:     |123.456|
123.456 formatted as |%12.4f|: |    123.4560|
123.456 formatted as |%12.4e|: |  1.2346e+02|
123.456 formatted as |%12.4g|: |       123.5|

"abc" formatted as |%12s|:     |         abc|
%
```