

EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 7

Rainer Dömer

doemer@uci.edu

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

Lecture 7.1: Overview

- Data Structures
 - Introduction
 - Arrays
 - Introduction
 - Indexing
 - Initialization
 - Multi-dimensional arrays
 - Operator associativity and precedence
 - Example 1
 - `Histogram.c`
 - Example 2
 - `Dice2.c`

Data Structures

- Introduction
 - Until now, we have used (mostly) single data elements of basic (non-composite) type
 - integral types
 - floating point types
 - Most programs, however, require complex *data structures* using composite types
 - arrays, lists, queues, stacks
 - trees, graphs
 - dictionaries
 - ANSI C provides built-in support for
 - arrays
 - structures, unions, enumerators
 - pointers

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

3

Arrays

- Array data type in C
 - Composite data type
 - Type is an array of a sub-type (e.g. array of `int`)
 - Fixed number of elements
 - Array size is fixed at time of definition (e.g. 100 elements)
 - Element access by index (aka. subscript)
 - Element-access operator: `array[index]` (e.g. `A[42]`)
- Example:

```
int A[10]; /* array of ten integers */  
  
A[0] = 42; /* access to elements */  
A[1] = 100;  
A[2] = A[0] + 5 * A[1];
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

4

Arrays

- Array Indexing
 - Start counting from 0
 - First element has index 0
 - Last element has index $Size-1$
- Example:

```
int A[10];

A[0] = 42;
A[1] = 100;
A[2] = A[0] + 5 * A[1];
A[3] = -1;
A[4] = 44;
A[5] = 55;
/* ... */
A[9] = 99;
```

	A
0	42
1	100
2	542
3	-1
4	44
5	55
6	0
7	0
8	0
9	99

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

5

Arrays

- Array Indexing
 - for loops are often very helpful
 - `for(i=0; i<N; i++)`
`{...A[i]...}`
- Example:

```
int A[10];
int i;

for(i=0; i<10; i++)
{ A[i] = i*10 + i;
}
for(i=0; i<10; i++)
{ printf("%d, ", A[i]);
}
```

	A
0	0
1	11
2	22
3	33
4	44
5	55
6	66
7	77
8	88
9	99

```
0, 11, 22, 33, 44, 55, 66, 77, 88, 99,
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

6

Arrays

- Array Indexing
 - Array indices are *not* checked by the compiler, nor at runtime!
 - Accessing an array with an *index out of range* results in undefined behavior!
- Example:


```
int A[10];
int i;

A[-1] = 42; /* INVALID ACCESS! */

for(i=0; i<=10; i++)
  /* INVALID LOOP RANGE! */
  { printf("%d, ", A[i]);
  }
```

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

EECS10: Computational Methods in ECE, Lecture 7 (c) 2016 R. Doemer 7

Arrays

- Array Initialization
 - Static initialization at time of array definition
 - Initial elements listed in { }
- Example:


```
int A[10] = { 42, 100,
              310, 44,
              55, 0,
              3, 4,
              0, 99};
```

	A
0	42
1	100
2	310
3	44
4	55
5	0
6	3
7	4
8	0
9	99

EECS10: Computational Methods in ECE, Lecture 7 (c) 2016 R. Doemer 8

Arrays

- Array Initialization
 - Static initialization at time of array definition
 - Initial elements listed in { }

- Example:

```
int A[ ] = { 42, 100,
            310, 44,
            55, 0,
            3, 4,
            0, 99};
```

- With given initializer list, array size may be omitted
 - automatically determined

	A
0	42
1	100
2	310
3	44
4	55
5	0
6	3
7	4
8	0
9	99

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

9

Arrays

- Array Initialization
 - Static initialization at time of array definition
 - Initial elements listed in { }

- Example:

```
int A[10] = { 1, 2, 3};
```

- With given initializer list *and* array size, unlisted elements are zero-initialized
 - array is filled up with zeros

	A
0	1
1	2
2	3
3	0
4	0
5	0
6	0
7	0
8	0
9	0

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

10

Arrays

- Multi-dimensional Arrays

- Array of an array...

- Example:

```
int M[3][2] = {{1, 2},
               {3, 4},
               {5, 6}};

int i, j;

for(i=0; i<3; i++)
  { for(j=0; j<2; j++)
    { printf("%d ",
             M[i][j]);
      }
    printf("\n");
  }
```

M	0	1
0	1	2
1	3	4
2	5	6

1	2
3	4
5	6

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

11

Arrays

- Operator associativity and precedence

– parentheses, array access	(), []	left to right
– unary operators	+, -, !, ++, --	right to left
– type casting	(<i>typename</i>)	right to left
– multiplication, division, modulo	*, /, %	left to right
– addition, subtraction	+, -	left to right
– shift left, shift right	<<, >>	left to right
– relational operators	<, <=, >=, >	left to right
– equality	==, !=	left to right
– logical and	&&	left to right
– logical or		left to right
– conditional operator	?:	left to right
– assignment operators	=, +=, *=, etc.	right to left
– comma operator	,	left to right

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

12

Arrays

- Program example: `Histogram.c`
 - Display a simple bar chart for 10 integer values
- Desired output:

```
% Histogram
Please enter data value 1: 111
Please enter data value 2: 222
Please enter data value 3: 33
Please enter data value 4: 333
[...]
Please enter data value 10: 111
1: 111 *****
2: 222 *****
3: 33 ****
4: 333 *****
[...]
10: 111 *****
%
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

13

Arrays

- Program example: `Histogram.c` (part 1/3)

```
/* Histogram.c: print a histogram of data values */
/* author: Rainer Doemer */
/* modifications: */
/* 11/02/04 RD initial version */

#include <stdio.h>

/* constants */
#define NUM_ROWS 10

/* main function */
int main(void)
{
    /* variable definitions */
    int Data[NUM_ROWS];
    int i, j, max;
    double scale;

    ...
}
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

14

Arrays

- Program example: `Histogram.c` (part 2/3)

```

...
/* input section */
for(i = 0; i < NUM_ROWS; i++)
{ printf("Please enter data value %2d: ", i+1);
  scanf("%d", &Data[i]);
} /* rof */

/* computation section */
max = 0;
for(i = 0; i < NUM_ROWS; i++)
{ if (Data[i] > max)
  { max = Data[i];
  } /* fi */
} /* rof */
scale = 70.0 / max;
...

```

Arrays

- Program example: `Histogram.c` (part 3/3)

```

...
/* output section */
for(i = 0; i < NUM_ROWS; i++)
{ printf("%2d: %5d ", i+1, Data[i]);
  for(j = 0; j < Data[i]*scale; j++)
  { printf("");
  } /* rof */
  printf("\n");
} /* rof */

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

/* EOF */

```


Arrays

- Example session: `Histogram.c`

```
% vi Histogram.c
% gcc Histogram.c -o Histogram -Wall -ansi
% Histogram
Please enter data value 1: 11
Please enter data value 2: 22
Please enter data value 3: 3
Please enter data value 4: 33
Please enter data value 5: 44
Please enter data value 6: 55
Please enter data value 7: 66
Please enter data value 8: 33
Please enter data value 9: 22
Please enter data value 10: 22
1: 11 *****
2: 22 *****
3: 3 ****
4: 33 *****
5: 44 *****
6: 55 *****
7: 66 *****
8: 33 *****
9: 22 *****
10: 22 *****
%
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

17

Arrays

- Earlier program example: `Dice.c` (part 1/4)

```
/* Dice.c: roll the dice */
/* author: Rainer Doemer */
/* modifications: */
/* 10/28/04 RD initial version */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* function definition */

int roll(void)
{
    int r;

    r = rand() % 6 + 1;
    /* printf("Rolled a %d.\n", r); */
    return r;
} /* end of roll */
...
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

18

Arrays

- Earlier program example: `Dice.c` (part 2/4)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    int i, n;
    int count1 = 0, count2 = 0, count3 = 0,
        count4 = 0, count5 = 0, count6 = 0;

    /* random number generator initialization */
    srand(time(0));

    /* input section */
    printf("Roll the dice: How many times? ");
    scanf("%d", &n);

    ...

```

Arrays

- Earlier program example: `Dice.c` (part 3/4)

```

... /* computation section */
for(i = 0; i < n; i++)
{ switch(roll())
  { case 1:
    { count1++; break; }
    case 2:
    { count2++; break; }
    case 3:
    { count3++; break; }
    case 4:
    { count4++; break; }
    case 5:
    { count5++; break; }
    case 6:
    { count6++; break; }
    default:
    { printf("INVALID ROLL!");
      exit(10); }
    } /* hctiws */
  } /* rof */
...

```

Arrays

- Earlier program example: `Dice.c` (part 4/4)

```

...

/* output section */
printf("Rolled a 1 %5d times.\n", count1);
printf("Rolled a 2 %5d times.\n", count2);
printf("Rolled a 3 %5d times.\n", count3);
printf("Rolled a 4 %5d times.\n", count4);
printf("Rolled a 5 %5d times.\n", count5);
printf("Rolled a 6 %5d times.\n", count6);

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

/* EOF */

```

Arrays

- Improved program example: `Dice2.c` (part 1/3)

```

/* Dice2.c: roll the dice */
/* author: Rainer Doemer */
/* modifications: */
/* 11/04/04 RD version using arrays */
/* 10/28/04 RD initial version */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* function definition */
int roll(void)
{
    int r;

    r = rand() % 6 + 1;
    /* printf("Rolled a %d.\n", r); */
    return r;
} /* end of roll */

...

```

Arrays

- Improved program example: `Dice2.c` (part 2/3)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    int i, n;
    int count[6] = { 0, 0, 0, 0, 0, 0 };

    /* random number generator initialization */
    srand(time(0));

    /* input section */
    printf("Roll the dice: How many times? ");
    scanf("%d", &n);

    ...

```

Arrays

- Improved program example: `Dice2.c` (part 3/3)

```

...
/* computation section */
for(i = 0; i < n; i++)
    { count[roll()-1]++;
    } /* rof */

/* output section */
for(i = 0; i < 6; i++)
    { printf("Rollea a %d %5d times.\n",
            i+1, count[i]);
    } /* rof */

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

/* EOF */

```

Arrays

- Example session: `Dice2.c`

```
% vi Dice2.c
% gcc Dice2.c -o Dice2 -Wall -ansi
% Dice2
Roll the dice: How many times? 6000
Rolled a 1 1009 times.
Rolled a 2 1005 times.
Rolled a 3 962 times.
Rolled a 4 998 times.
Rolled a 5 996 times.
Rolled a 6 1030 times.
% Dice2
Roll the dice: How many times? 6000
Rolled a 1 1042 times.
Rolled a 2 983 times.
Rolled a 3 972 times.
Rolled a 4 979 times.
Rolled a 5 1022 times.
Rolled a 6 1002 times.
%
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

25

Lecture 7.2: Overview

- Passing arguments to functions
 - Pass by value
 - Pass by reference
- Character Arrays: Strings
 - Input and output
 - ASCII table
 - Example: Sort strings alphabetically
 - Task
 - Approach
 - Algorithm *Bubble Sort*
 - Program `BubbleSort.c`

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

26

Passing Arguments to Functions

- Pass by Value
 - only the *current value* is passed as argument
 - the parameter is a *copy* of the argument
 - changes to the parameter *do not* affect the argument
- Pass by Reference
 - a *reference* to the object is passed as argument
 - the parameter is a *reference* to the argument
 - changes to the parameter *do* affect the argument
- In ANSI C, ...
 - ... basic types are passed by value
 - ... arrays are passed by reference

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

27

Passing Arguments to Functions

- Example: Pass by Value

```
void f(int p)
{
    printf("p before modification is %d\n", p);
    p = 42;
    printf("p after modification is %d\n", p);
}

int main(void)
{
    int a = 0;
    printf("a before function call is %d\n", a);
    f(a);
    printf("a after function call is %d\n", a);
}
```

```
a before function call is 0
p before modification is 0
p after modification is 42
a after function call is 0
```

Changes to the parameter *do not* affect the argument!

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

28

Passing Arguments to Functions

- Example: Pass by Reference

```
void f(int p[2])
{
    printf("p[1] before modification is %d\n", p[1]);
    p[1] = 42;
    printf("p[1] after modification is %d\n", p[1]);
}

int main(void)
{
    int a[2] = {0, 0};
    printf("a[1] before function call is %d\n", a[1]);
    f(a);
    printf("a[1] after function call is %d\n", a[1]);
}
```

```
a[1] before function call is 0
p[1] before modification is 0
p[1] after modification is 42
a[1] after function call is 42
```

Changes to the parameter **do** affect the argument!

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

29

Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
 - Strings are null-terminated arrays of characters
 - String output
 - `printf()` format specifier: `"%s"`
- Example:

```
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};

printf("s1 is %s.\n", s1);
```

```
s1 is Hello.
```

	s1
0	'H'
1	'e'
2	'l'
3	'l'
4	'o'
5	0

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

30

Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
 - Strings are null-terminated arrays of characters
 - String output
 - `printf()` format specifier: `"%s"`
- Example:

```
char s1[] = {'H','e','l','l','o',0};
char s2[] = "Hello";

printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
```

```
s1 is Hello.
s2 is Hello.
```

	s2
0	'H'
1	'e'
2	'l'
3	'l'
4	'o'
5	0

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

31

Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
 - Strings are null-terminated arrays of characters
 - String output
 - `printf()` format specifier: `"%s"`
- Example:

```
char s1[] = {'H','e','l','l','o',0};
char s2[] = "Hello";

printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
s1[1] = 'i';
s1[2] = 0;
printf("Modified s1 is %s.\n", s1);
```

```
s1 is Hello.
s2 is Hello.
Modified s1 is Hi.
```

	s1
0	'H'
1	'i'
2	0
3	'l'
4	'o'
5	0

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

32

Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
 - Strings are null-terminated arrays of characters
 - String input
 - `scanf()` format specifier: `"%Ns"`, where `N` specifies maximum field width = array size - 1
 - address argument can be `&string[0]`

- Example:

```
char s1[6];
printf("Enter a string: ");
scanf("%5s", &s1[0]);
printf("s1 is %s.\n", s1);
```

```
Enter a string: Test
s1 is Test.
```

	s1
0	'T'
1	'e'
2	's'
3	'\t'
4	0
5	0

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

33

Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
 - Strings are null-terminated arrays of characters
 - String input
 - `scanf()` format specifier: `"%Ns"`, where `N` specifies maximum field width = array size - 1
 - address argument can be `&string[0]` or simply `string`

- Example:

```
char s1[6];
printf("Enter a string: ");
scanf("%5s", s1);
printf("s1 is %s.\n", s1);
```

```
Enter a string: Test
s1 is Test.
```

	s1
0	'T'
1	'e'
2	's'
3	'\t'
4	0
5	0

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

34

Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
 - Strings are null-terminated arrays of characters
 - Characters are represented by numeric values
 - ASCII table defines character values 0-127
- Example:

```
char s1[] = "ABC12";
int i = 0;

while(s1[i])
{ printf("%c = %d\n",s1[i],s1[i]);
  i++; }
```

```
A = 65
B = 66
C = 67
1 = 49
2 = 50
```

	s1
0	'A'
1	'B'
2	'C'
3	'1'
4	'2'
5	0

Character Arrays: Strings

- ASCII Table
 - 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

Character Arrays: Strings

- Case Study: *Bubble Sort*
 - Task: Sort an array of strings alphabetically
 - Input: Array of 10 strings entered by the user
 - Output: Array of 10 strings in alphabetical order
- Approach: Divide and Conquer
 - Step 1: Let user enter 10 strings
 - Step 2: Sort the array of strings
 - Step 3: Output the strings in order

Character Arrays: Strings

- Case Study: *Bubble Sort*
 - Task: Sort an array of strings alphabetically
 - Input: Array of 10 strings entered by the user
 - Output: Array of 10 strings in alphabetical order
- Approach: Divide and Conquer
 - Step 1: Let user enter 10 strings
 - Step 2: Sort the array of strings
 - Algorithm
 - in 9 rounds, compare all adjacent pairs of strings and swap the pair if they are not in alphabetical order
 - String comparison
 - compare character pairs alphabetically: use ASCII values!
 - String swap (exchange two strings in place)
 - swap each character pair in the two strings
 - Step 3: Output the strings in order

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 1/7)

```

/* BubbleSort.c: sort strings alphabetically */
/* author: Rainer Doemer */
/* modifications: */
/* 11/01/06 RD swap only adjacent elements */
/* 11/06/04 RD initial version */

#include <stdio.h>

/* constant definitions */

#define NUM 10 /* ten strings */
#define LEN 20 /* of length 20 */

/* function declarations */

void EnterText(char Text[NUM][LEN]);
void PrintText(char Text[NUM][LEN]);
int CompareStrings(char s1[LEN], char s2[LEN]);
void SwapStrings(char s1[LEN], char s2[LEN]);
void BubbleSort(char Text[NUM][LEN]);
...

```

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 2/7)

```

...

/* function definitions */

/* let the user enter the text array */

void EnterText(char Text[NUM][LEN])
{
    int i;

    for(i = 0; i < NUM; i++)
    { printf("Enter text string %2d: ", i+1);
      scanf("%19s", Text[i]);
    } /* rof */
} /* end of EnterText */

...

```

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 3/7)

```

...
/* print the text array on the screen      */
void PrintText(char Text[NUM][LEN])
{
    int i;

    for(i = 0; i < NUM; i++)
        { printf("String %2d: %s\n", i+1, Text[i]);
          } /* rof */
} /* end of PrintText */
...

```

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 4/7)

```

...
/* alphabetically compare strings s1 and s2: */
/* return -1, if string s1 < string s2      */
/* return 0, if string s1 = string s2      */
/* return 1, if string s1 > string s2      */
int CompareStrings(char s1[LEN], char s2[LEN])
{
    int i;

    for(i = 0; i < LEN; i++)
        { if (s1[i] > s2[i])
          { return(1); }
          if (s1[i] < s2[i])
          { return(-1); }
          if (s1[i] == 0 || s2[i] == 0)
          { break; }
        } /* rof */
    return 0;
} /* end of CompareStrings */
...

```

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 5/7)

```

...
/* swap/exchange the strings s1 and s2 in place */
void SwapStrings(char s1[LEN], char s2[LEN])
{
    int i;
    char c;

    for(i = 0; i < LEN; i++)
    { c = s1[i];
      s1[i] = s2[i];
      s2[i] = c;
    } /* rof */
} /* end of SwapStrings */
...

```

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 6/7)

```

...
/* sort the text array by comparing every pair */
/* of strings; if the pair of strings is not in */
/* alphabetical order, swap it */
void BubbleSort(char Text[NUM][LEN])
{
    int p, i;

    for(p = 1; p < NUM; p++)
    { for(i = 0; i < NUM-1; i++)
      { if (CompareStrings(Text[i], Text[i+1]) > 0)
        { SwapStrings(Text[i], Text[i+1]);
          } /* fi */
        } /* rof */
      } /* rof */
    } /* end of BubbleSort */
...

```

Character Arrays: Strings

- Program example: `BubbleSort.c` (part 7/7)

```

...
/* main function: enter, sort, print the text */
int main(void)
{
    /* local variables */
    char Text[NUM][LEN]; /* NUM strings, length LEN */

    /* input section */
    EnterText(Text);

    /* computation section */
    BubbleSort(Text);

    /* output section */
    PrintText(Text);

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

/* EOF */

```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

45

Character Arrays: Strings

- Example session: `BubbleSort.c`

```

% vi BubbleSort.c
% gcc BubbleSort.c -o BubbleSort -Wall -ansi
% BubbleSort
Enter text string 1: Charlie
Enter text string 2: William
Enter text string 3: Donald
Enter text string 4: John
Enter text string 5: Jane
Enter text string 6: Jessie
Enter text string 7: Donald
Enter text string 8: Henry
Enter text string 9: George
Enter text string 10: Emily
String 1: Charlie
String 2: Donald
String 3: Donald
String 4: Emily
String 5: George
String 6: Henry
String 7: Jane
String 8: Jessie
String 9: John
String 10: William
%
EE

```

Lecture 7.3: Overview

- Review
 - Lecture 4.1: Formatted output
 - Lecture 4.2: Structured programming, conditions
 - Lecture 5.1: Structured programming, loops
 - Lecture 5.2: Jump statements, debugging
 - Lecture 5.3: Functions, terms and concepts
 - Lecture 6.1: Functions, hierarchy, stack frames
 - Lecture 6.2: Functions, scope rules
 - Lecture 6.3: Standard library functions
 - Lecture 7.1: Data structures, arrays
 - Lecture 7.2: Passing arrays to functions, strings
- Review Quiz

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

47

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:

- Prime number test:
Iterate over $2 \leq i < x$
to find a divisor of x .
What should go into
the box in line 4?

- a) $i = 0;$
- b) $i = 1;$
- c) $i = 2;$
- d) $i = x;$
- e) $x = 0;$

```

int x, i;
printf("Please input a number: ");
scanf("%d", &x);
initialize variable i
while(i < x)
{ if(x % i == 0)
  { printf("%d is not prime\n", x);
    break;
  }
  i++;
}
if( none of the i is a divisor of x )
{ printf("%d is prime\n", x);
}

```

EECS10: Computational Methods in ECE, Lecture 7


(c) 2016 R. Doemer

48

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:

- Prime number test:
Iterate over $2 \leq i < x$
to find a divisor of x .
What should go into
the box in line 4?

- a) $i = 0;$
- b) $i = 1;$
-  c) $i = 2;$
- d) $i = x;$
- e) $x = 0;$

```
int x, i;
printf("Please input a number: ");
scanf("%d", &x);
initialize variable i
while(i < x)
{ if(x % i == 0)
  { printf("%d is not prime\n", x);
    break;
  }
  i++;
}
if( none of the i is a divisor of x )
{ printf("%d is prime\n", x);
}
```

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:

- Prime number test:
Iterate over $2 \leq i < x$
to find a divisor of x .
What should go into
the box in line 12?


- a) $x / i == 0$
- b) $x < i$
- c) $i / x == 0$
- d) $i + 1 == x$
- e) $i == x$

```
int x, i;
printf("Please input a number: ");
scanf("%d", &x);
initialize variable i
while(i < x)
{ if(x % i == 0)
  { printf("%d is not prime\n", x);
    break;
  }
  i++;
}
if( none of the i is a divisor of x )
{ printf("%d is prime\n", x);
}
```

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:

- Prime number test:
Iterate over $2 \leq i < x$
to find a divisor of x .
What should go into
the box in line 12?

- a) `x / i == 0`
- b) `x < i`
- c) `i / x == 0`
- d) `i + 1 == x`
-  e) `i == x`

```
int x, i;
printf("Please input a number: ");
scanf("%d", &x);
initialize variable i
while(i < x)
{ if(x % i == 0)
  { printf("%d is not prime\n", x);
    break;
  }
  i++;
}
if(  )
{ printf("%d is prime\n", x);
}
```

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:

- Which of the following program fragments will *not* terminate? (Check all that apply!)

a)

```
int a = 1;
while(a < 1000000)
{ a++; }
```

d)

```
int a = 10;
while(a > 0)
{ a = a / 3; }
```

b)

```
int a = 0;
while(a < 1000)
{ a = a * 3; }
```

e)

```
int a = 1;
while(a < 1000)
{ a = a << 1; }
```

c)

```
int a = 1;
while(a == 1)
{ a = a % 10; }
```

Midterm 1 Review Quiz


- Top 5 “most difficult” questions:
- Which of the following program fragments will *not* terminate? (Check all that apply!)

a)

```
int a = 1;
while(a < 1000000)
{ a++; }
```

d)


```
int a = 10;
while(a > 0)
{ a = a / 3; }
```

 b)

```
int a = 0;
while(a < 1000)
{ a = a * 3; }
```

e)

```
int a = 1;
while(a < 1000)
{ a = a << 1; }
```

 c)

```
int a = 1;
while(a == 1)
{ a = a % 10; }
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

53

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:
- Which of the following C expressions yield the same result? (Check all that apply!)

a) $4 \ll 8 \% 5 / 2$

b) $(4 \ll 8) \% 5 / 2$

c) $4 \ll 8 \% (5 / 2)$

d) $(4 \ll 8 \% 5) / 2$

e) $4 \ll (8 \% 5) / 2$

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

54

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:
- Which of the following C expressions yield the same result?
(Check all that apply!)

- a) `4 << 8 % 5 / 2` (8)
- b) `(4 << 8) % 5 / 2` (2)
- c) `4 << 8 % (5 / 2)` (4)
- d) `(4 << 8 % 5) / 2` (16)
- e) `4 << (8 % 5) / 2` (8)

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

55

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:
- What is the output of the following C program fragment?

```
int i1 = 5, i2 = 2, i;
float f1 = 5, f2 = 2, f;
i = i1 / i2;
f = (int)(f1 / f2);
printf("i = %d, f = %f", i, f);
```

- a) `i = 2, f = 2`
- b) `i = 1, f = 2`
- c) `i = 2, f = 2.00000`
- d) `i = 2.00000, f = 2.50000`
- e) `i = 2, f = 2.50000`

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

56

Midterm 1 Review Quiz

- Top 5 “most difficult” questions:
- What is the output of the following C program fragment?

```
int i1 = 5, i2 = 2, i;
float f1 = 5, f2 = 2, f;
i = i1 / i2;
f = (int)(f1 / f2);
printf("i = %d, f = %f", i, f);
```

- a) i = 2, f = 2
- b) i = 1, f = 2
- c) i = 2, f = 2.00000
- d) i = 2.00000, f = 2.50000
- e) i = 2, f = 2.50000

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

57

Quiz: Question 1

- Which of the following expressions would be treated as a true condition when used with an `if` statement?
(Check all that apply!)

- a) `(int)5.99 > 5`
- b) `1 || 0 && 1`
- c) `5 >= 5`
- d) `(1 + 2 + 3) == (3 << 2 >> 1)`
- e) `5 - 5`

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

58

Quiz: Question 1

- Which of the following expressions would be treated as a true condition when used with an `if` statement?

(Check all that apply!)

- a) `(int)5.99 > 5`
- b) `1 || 0 && 1`
- c) `5 >= 5`
- d) `(1 + 2 + 3) == (3 << 2 >> 1)`
- e) `5 - 5`

Quiz: Question 2

- If `count` is an integer counter that counts upwards in steps of 1, how could one update the value of `count`?

(Check all that apply!)

- a) `count += 1;`
- b) `count = count + 1;`
- c) `++count;`
- d) `count++;`
- e) `count += count;`

Quiz: Question 2

- If `count` is an integer counter that counts upwards in steps of 1, how could one update the value of `count`?
(Check all that apply!)

- a) `count += 1;`
- b) `count = count + 1;`
- c) `++count;`
- d) `count++;`
- e) `count += count;`

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

61

Quiz: Question 3

- What is the value of `x` after the following code fragment is executed?

```
int x = 0;
for(x = 1; x <= 10; x++)
{ }
```

- a) 0
- b) 1
- c) 9
- d) 10
- e) 11

EECS10: Computational Methods in ECE, Lecture 7


(c) 2016 R. Doemer

62

Quiz: Question 3

- What is the value of x after the following code fragment is executed?

```
int x = 0;
for(x = 1; x <= 10; x++)
{ }
```

- a) 0
- b) 1
- c) 9
- d) 10
-  e) 11

Quiz: Question 4

- What is the value of x after the following code fragment is executed?


```
int x = 0;
do { x++;
} while(x < 9);
```

- a) 0
- b) 1
- c) 9
- d) 10
- e) 11

Quiz: Question 4

- What is the value of **x** after the following code fragment is executed?

```
int x = 0;
do { x++;
    } while(x < 9);
```

- a) 0
- b) 1
-  c) 9
- d) 10
- e) 11

Quiz: Question 5

- What is the value of **x** after the following code fragment is executed?


```
int x = 10;
while(x > 0)
{ x -= 2;
}
```

- a) -2
- b) -1
- c) 0
- d) 1
- e) 2

Quiz: Question 5

- What is the value of x after the following code fragment is executed?

```
int x = 10;
while(x > 0)
{ x -= 2;
}
```

- a) -2
- b) -1
-  c) 0
- d) 1
- e) 2

Quiz: Question 6


- Given the following function g , what is the result of $g(85)$?

```
char g(int n)
{
    switch(n/10)
    { case 10:
      case 9: return('A');
      case 8: return('B');
      case 7: return('C');
      case 6: return('D');
      default: return('F');
    }
}
```

- a) 'A'
- b) 'B'
- c) 'C'
- d) 'D'
- e) 'F'

Quiz: Question 6

- Given the following function g , what is the result of $g(85)$?

- a) 'A'
-  b) 'B'
- c) 'C'
- d) 'D'
- e) 'F'

```
char g(int n)
{
    switch(n/10)
    { case 10:
      case 9: return('A');
      case 8: return('B');
      case 7: return('C');
      case 6: return('D');
      default: return('F');
    }
}
```

Quiz: Question 7

- What is output by the following C statement?


```
printf("x = %03d", 3 + 4);
```

- a) $x = 034$
- b) $x = 037$
- c) $x = 007$
- d) $x = 7$
- e) $x = 347$

Quiz: Question 7

- What is output by the following C statement?

```
printf("x = %03d", 3 + 4);
```


- a) x = 034
- b) x = 037
-  c) x = 007
- d) x = 7
- e) x = 347

Quiz: Question 8

- In the `gdb` debugger, what does `next` do?

- a) It moves to the next argument of the function.
- b) It calls the next function in the program.
- c) It executes the next statement in the program.
- d) It prints the value of the next variable.
- e) It loads the next program into the debugger.

Quiz: Question 8

- In the `gdb` debugger, what does `next` do?
 - a) It moves to the next argument of the function.
 - b) It calls the next function in the program.
 -  c) It executes the next statement in the program.
 - d) It prints the value of the next variable.
 - e) It loads the next program into the debugger.

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

73

Quiz: Question 9

- Given the following code fragment, which of the following statements are true?
(Check all that apply!)

- a) Function `f` is declared.
- b) Function `g` calls function `f`
- c) Variable `z` is a local variable of function `g`
- d) Function `g` is declared and defined.
- e) `y` is a parameter of function `g`.

```
double f(int x);
void g(int x, int y)
{
    int z;

    z = f(x) + 2*y;
    return z;
}
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

74

Quiz: Question 9

- Given the following code fragment, which of the following statements are true?
(Check all that apply!)

```
double f(int x);  
void g(int x, int y)  
{  
    int z;  
  
    z = f(x) + 2*y;  
    return z;  
}
```

- a) Function `f` is declared.
- b) Function `g` calls function `f`
- c) Variable `z` is a local variable of function `g`
- d) Function `g` is declared and defined.
- e) `y` is a parameter of function `g`.

Quiz: Question 10

- Given that the C standard math library is included, which of the following expressions results in the value `4.0`?
(Check all that apply!)

- a) `pow(16.0, .5)`
- b) `4.0 * cos(0.0)`
- c) `3 + sin(0.0)`
- d) `log10(10000.00)`
- e) `sqrt(15.0) + 1`

Quiz: Question 10

- Given that the C standard math library is included, which of the following expressions results in the value 4.0?
(Check all that apply!)

- a) `pow(16.0, .5)`
- b) `4.0 * cos(0.0)`
- c) `3 + sin(0.0)`
- d) `log10(10000.00)`
- e) `sqrt(15.0) + 1`

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

77

Quiz: Question 11

- Given the following program fragment, what is the value of `g(2, f(3, 4))`?

- a) 8
- b) 9
- c) 10
- d) 11
- e) 12

```
int x = 7;

int f(int x, int y)
{
    return x + y;
}

int g(int x, int y)
{
    return f(y, x);
}
```

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

78

Quiz: Question 11

- Given the following program fragment, what is the value of $g(2, f(3, 4))$?

- a) 8
- b) 9
- c) 10
- d) 11
- e) 12

```
int x = 7;

int f(int x, int y)
{
    return x + y;
}

int g(int x, int y)
{
    return f(y, x);
}
```

Quiz: Question 12

- What is output by the following program fragment?

- a) EECS00 1
- b) EEC 10 0
- c) E E
- d) EECS C
- e) EEC C


```
char s[] = "EECS10";

s[4] = 0;
printf("%s %c", s, s[2]);
```


Quiz: Question 12

- What is output by the following program fragment?

```
char s[] = "EECS10";
s[4] = 0;
printf("%s %c", s, s[2]);
```

- a) EECS00 1
- b) EEC 10 0
- c) E E
-  d) **EECS C**
- e) EEC C

Quiz: Question 13

- Given the definition `double p=0.0125;` which of the following C statements will print out `p = 1.25%` ?
(Check all that apply!)

- a) `printf("p = %d.25%%", (int)(p*100.0));`
- b) `printf("p = %p", 100.0*p);`
- c) `printf("p = %.2f%%", p*100.0);`
- d) `printf("p = %.2f%c", p*100.0, '%');`
- e) `printf("p = ", 100.0 * p, "%");`

Quiz: Question 13

- Given the definition `double p=0.0125;` which of the following C statements will print out `p = 1.25%` ?
(Check all that apply!)

- a) `printf("p = %d.25%%", (int)(p*100.0));`
- b) `printf("p = %p", 100.0*p);`
- c) `printf("p = %.2f%%", p*100.0);`
- d) `printf("p = %.2f%c", p*100.0, '%');`
- e) `printf("p = ", 100.0 * p, "%");`

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

83

Quiz: Question 14

- Which of the following statements is true for an *algorithm*?
(Check all that apply!)


- a) An algorithm must be indeterministic.
- b) An algorithm solves a problem quickly.
- c) An algorithm is historically based on Al Gore's rythm.
- d) An algorithm executes a program using pseudo code.
- e) An algorithm must terminate after a finite number of steps.

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

84

Quiz: Question 14

- Which of the following statements is true for an *algorithm*?
(Check all that apply!)
 - a) An algorithm must be indeterministic.
 - b) An algorithm solves a problem quickly.
 - c) An algorithm is historically based on Al Gore's rythm.
 - d) An algorithm executes a program using pseudo code.
 -  e) An algorithm must terminate after a finite number of steps.

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

85

Quiz: Question 15

- Which of the following declarations can be added to the program in line 8 without creating a compilation error?
(Check all that apply!)

- | | |
|---|---|
| <ul style="list-style-type: none"> a) <code>int f(int v, double w);</code> b) <code>int g = 0;</code> c) <code>int g(int x, int y);</code> d) <code>int x = 2;</code> e) <code>int f(double v, double w);</code> | <pre> 1 int x = 2; 2 int f(int v, double w); 3 int g(int x, int y) 4 { int z; 5 z = 2*x + 5*y - 42; 6 return z; 7 } 8</pre> |
|---|---|

EECS10: Computational Methods in ECE, Lecture 7

(c) 2016 R. Doemer

86

Quiz: Question 15

- Which of the following declarations can be added to the program in line 8 without creating a compilation error?

(Check all that apply!)

- a) `int f(int v, double w);`
- b) `int g = 0;`
- c) `int g(int x, int y);`
- d) `int x = 2;`
- e) `int f(double v, double w);`

```

1 int x = 2;
2 int f(int v, double w);
3 int g(int x, int y)
4 { int z;
5   z = 2*x + 5*y - 42;
6   return z;
7 }
8

```

Quiz: Question 16

- The following function `issorted` is supposed to return true if and only if the given array `L` is sorted in increasing order.
- What should go into `Box1` in line 3?

- a) `i=1; i<10; i++`
- b) `i=0; i<10; i++`
- c) `i=0; i<9; i++`
- d) `i=10; i>0; i--`
- e) `i=9; i>=0; i--`


```

1 int issorted(int L[10])
2 { int i;
3   for(Box1)
4     { if(L[i] >= L[i+1])
5       { Box2; }
6     }
7   Box3 ;
8 }

```

Quiz: Question 16

- The following function `issorted` is supposed to return true if and only if the given array `L` is sorted in increasing order.
- What should go into `Box1` in line 3?

- a) `i=1; i<10; i++`
 b) `i=0; i<10; i++`
 c) `i=0; i<9; i++`
 d) `i=10; i>0; i--`
 e) `i=9; i>=0; i--`

```

1 int issorted(int L[10])
2 { int i;
3   for(Box1)
4     { if(L[i] >= L[i+1])
5       { Box2; }
6     }
7   Box3 ;
8 }
```

Quiz: Question 17

- The following function `issorted` is supposed to return true if and only if the given array `L` is sorted in increasing order.
- What should go into `Box2` in line 5?


- a) `return 0`
 b) `return 1`
 c) `continue`
 d) `break`
 e) `return`

```

1 int issorted(int L[10])
2 { int i;
3   for(Box1)
4     { if(L[i] >= L[i+1])
5       { Box2; }
6     }
7   Box3 ;
8 }
```

Quiz: Question 17

- The following function `issorted` is supposed to return true if and only if the given array `L` is sorted in increasing order.
- What should go into `Box2` in line 5?

-  a) `return 0`
 b) `return 1`
 c) `continue`
 d) `break`
 e) `return`

```

1 int issorted(int L[10])
2 { int i;
3   for(Box1)
4     { if(L[i] >= L[i+1])
5       { Box2; }
6     }
7   Box3 ;
8 }
```

Quiz: Question 18

- The following function `issorted` is supposed to return true if and only if the given array `L` is sorted in increasing order.
- What should go into `Box3` in line 7?


- a) `return 0`
 b) `return 1`
 c) `continue`
 d) `break`
 e) `return`

```

1 int issorted(int L[10])
2 { int i;
3   for(Box1)
4     { if(L[i] >= L[i+1])
5       { Box2; }
6     }
7   Box3 ;
8 }
```

Quiz: Question 18

- The following function `issorted` is supposed to return true if and only if the given array `L` is sorted in increasing order.
- What should go into `Box3` in line 7?

- a) `return 0`
-  b) `return 1`
- c) `continue`
- d) `break`
- e) `return`

```
1 int issorted(int L[10])
2 { int i;
3   for(Box1)
4     { if(L[i] >= L[i+1])
5       { Box2; }
6     }
7   Box3 ;
8 }
```

Quiz: Question 19

- What is output by the following C statement?


```
int x = 0, y = 5;
x = y++;
printf("x = %d, y = %d", x, y);
```

- a) `x = 0, y = 5`
- b) `x = 5, y = 5`
- c) `x = 5, y = 6`
- d) `x = 6, y = 5`
- e) `x = 6, y = 6`

Quiz: Question 19

- What is output by the following C statement?

```
int x = 0, y = 5;  
x = y++;  
printf("x = %d, y = %d", x, y);
```

- a) `x = 0, y = 5`
- b) `x = 5, y = 5`
-  c) `x = 5, y = 6`
- d) `x = 6, y = 5`
- e) `x = 6, y = 6`

Quiz: Question 20

- What is output by the following C statement?

```
int x = 0, y = 5;  
x = ++y;  
printf("x = %d, y = %d", x, y);
```

- a) `x = 0, y = 5`
- b) `x = 5, y = 5`
- c) `x = 5, y = 6`
- d) `x = 6, y = 5`
- e) `x = 6, y = 6`

Quiz: Question 20

- What is output by the following C statement?


```
int x = 0, y = 5;  
x = ++y;  
printf("x = %d, y = %d", x, y);
```

a) `x = 0, y = 5`

b) `x = 5, y = 5`

c) `x = 5, y = 6`

d) `x = 6, y = 5`

 e) `x = 6, y = 6`