EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 17

Rainer Dömer

doemer@uci.edu

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

Lecture 17: Overview

- Midterm Course Evaluation
  - Results
- Arrays
  - Indexing
  - Initialization
  - Multi-dimensional arrays
  - Operator associativity and precedence
  - Improved example
    `Dice2.c`
- Passing arguments to functions
  - Pass by value
  - Pass by reference

Midterm Course Evaluation: Results

- Only few respondents
  - Less than 20% for lectures, even less for labs/discussions
- Overall feedback
  - Very positive comments
- Suggested improvements
  - More TAs
  - Office hours, also for TAs
  - Better grades
  - Consistent lectures to text book
  - Better homework feedback
  - Midterm exam
    - More preparation
    - Different from lectures, homeworks
    - Difficulty in multi-choice questions

- Please spend 5 minutes!
- Thanks a lot!
- Budget is limited.
- Again, resource limitation.
- Well, ...
- Two tracks are intended.
- Solutions now online!
- Warm-up quizzes!
- Understood, but needed.
- Understood, but needed.

Arrays

- Array:
  - Composite data type
    - Type is an array of a sub-type
  - Fixed number of elements
    - Number of elements is fixed at array definition
  - Element access by index (aka. subscript)
    - Element-access operator: ***array_name*[*index*]**
- Example:

```
int A[10];   /* array of ten integers */

A[0] = 42;   /* access to elements */
A[1] = 100;
A[2] = A[0] + 5 * A[1];
```

Arrays

- Array Indexing
  - Start counting from 0
    - First element has index 0
    - Last element has index N-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

Arrays

- Array Indexing
  - **for** loops are very helpful
    - **for(i=0; i<N; 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]);
    }
```

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

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

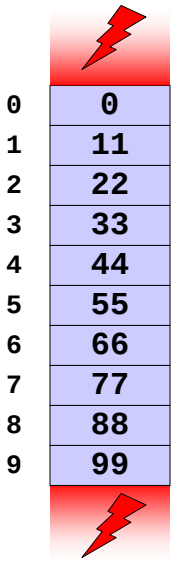
Arrays

- Array Indexing
  - Array indices are *not* checked by the compiler!
  - Accessing an array with an *index out of range* results in unpredictable 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]);
    }
```

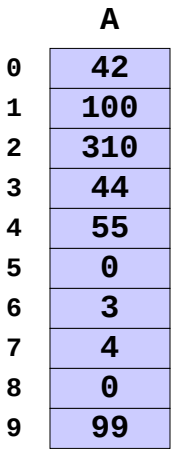


EECS10: Computational Methods in ECE, Lecture 17 (c) 2004 R. Doemer 7

Arrays

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

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



EECS10: Computational Methods in ECE, Lecture 17 (c) 2004 R. Doemer 8

Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements 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

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
```

Arrays

- Operator associativity and precedence

– parentheses, array access	**( )**, **[]**	left to right
– unary operators	**+**, **-**, **!**, **++**, **--**	right to left
– type casting	**(*typename*)**	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

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 */
...
```

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("Rolled 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.
%
```


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

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!

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!