C Programming for Engineers

Arrays

ICEN 360– Spring 2017

Prof. Dola Saha

Passing Arrays to Functions

- To pass an array argument to a function, specify the array's name without any brackets.
- For example,

```
int hourlyTemperatures[HOURS_IN_A_DAY];
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY);
```

  the function call passes array `hourlyTemperatures` and its size to function `modifyArray`.
- The name of the array evaluates to the address of the first element of the array.
- The called function *can modify* the element values in the callers' original arrays.

Passing Array to Functions (1)

```c
// Fig. 6.13: fig06_13.c
// Passing arrays and individual array elements to functions.
#include <stdio.h>
#define SIZE 5

// function prototypes
void modifyArray(int b[], size_t size);
void modifyElement(int e);

// function main begins program execution
int main(void)
{
   int a[SIZE] = {0, 1, 2, 3, 4}; // initialize array a

   puts("Effects of passing entire array by reference:\n\nThe "
      "values of the original array are:");

   // output original array
   for (size_t i = 0; i < SIZE; ++i) {
      printf("%3d", a[i]);
   }

   puts(""); // outputs a newline

```

Passing Array to Functions (2)

```
25      modifyArray(a, SIZE); // pass array a to modifyArray by reference
26      puts("The values of the modified array are:");
27
28      // output modified array
29      for (size_t i = 0; i < SIZE; ++i) {
30         printf("%3d", a[i]);
31      }
32
33      // output value of a[3]
34      printf("\n\n\nEffects of passing array element "
35         "by value:\n\nThe value of a[3] is %d\n", a[3]);
36
37      modifyElement(a[3]); // pass array element a[3] by value
38
39      // output value of a[3]
40      printf("The value of a[3] is %d\n", a[3]);
41   }
42
```

Passing Array to Functions (3)

```
43  // in function modifyArray, "b" points to the original array "a"
44  // in memory
45  void modifyArray(int b[], size_t size)
46  {
47     // multiply each array element by 2
48     for (size_t j = 0; j < size; ++j) {
49        b[j] *= 2; // actually modifies original array
50     }
51  }
52
53  // in function modifyElement, "e" is a local copy of array element
54  // a[3] passed from main
55  void modifyElement(int e)
56  {
57     // multiply parameter by 2
58     printf("Value in modifyElement is %d\n", e *= 2);
59  }
```

Passing Array to Functions (4)

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

Memory location of Arrays

- `array`, `&array` and `&array[0]` have the same value, namely `0012FF78`

```
1   // Fig. 6.12: fig06_12.c
2   // Array name is the same as the address of the array's first element.
3   #include <stdio.h>
4
5   // function main begins program execution
6   int main(void)
7   {
8      char array[5]; // define an array of size 5
9
10     printf("    array = %p\n&array[0] = %p\n   &array = %p\n",
11        array, &array[0], &array);
12  }
```

```
    array = 0031F930
&array[0] = 0031F930
   &array = 0031F930
```

Protecting Array Elements

- Function `tryToModifyArray` is defined with parameter `const int b[]`, which specifies that array `b` is constant and cannot be modified.
- The output shows the error messages produced by the compiler—the errors may be different for your compiler.

```
1  // in function tryToModifyArray, array b is const, so it cannot be
2  // used to modify its array argument in the caller
3  void tryToModifyArray(const int b[])
4  {
5     b[0] /= 2; // error
6     b[1] /= 2; // error
7     b[2] /= 2; // error
8  }
```

Classwork Assignment

- Search an Array: Write a program to initialize an array of size `S` with an initializer list. Also get a value for `num1` from user. Pass the array as well as `num1` to a function. Within the function, check each element of array whether it matches num1. If it matches, return 1, else return 0 to the `main` function.

Binary Search – searching in a sorted array

- The linear searching method works well for *small* or *unsorted* arrays.
- However, for large arrays linear searching is *inefficient*.
- If the array is sorted, the high-speed binary search technique can be used.
- The binary search algorithm eliminates from consideration *one-half* of the elements in a sorted array after each comparison.

Binary Search – searching in a sorted array

- The algorithm locates the *middle* element of the array and compares it to the search key.
- If they're equal, the search key is found and the index of that element is returned.
- If they're not equal, the problem is reduced to searching *one-half* of the array.
- If the search key is less than the middle element of the array, the *first half* of the array is searched, otherwise the *second half* of the array is searched.

Demo

- Demo from Princeton

https://www.cs.princeton.edu/courses/archive/fall06/cos226/demo/demo-bsearch.ppt

Binary Search – C code (1)

```c
// Fig. 6.19: fig06_19.c
// Binary search of a sorted array.
#include <stdio.h>
#define SIZE 15

// function prototypes
size_t binarySearch(const int b[], int searchKey, size_t low, size_t high);
void printHeader(void);
void printRow(const int b[], size_t low, size_t mid, size_t high);

// function main begins program execution
int main(void)
{
   int a[SIZE]; // create array a

   // create data
   for (size_t i = 0; i < SIZE; ++i) {
      a[i] = 2 * i;
   }

   printf("%s", "Enter a number between 0 and 28: ");
   int key; // value to locate in array a
   scanf("%d", &key);

```

Binary Search – C code (2)

```
25      printHeader();
26
27      // search for key in array a
28      size_t result = binarySearch(a, key, 0, SIZE - 1);
29
30      // display results
31      if (result != -1) {
32         printf("\n%d found at index %d\n", key, result);
33      }
34      else {
35         printf("\n%d not found\n", key);
36      }
37   }
38
39   // function to perform binary search of an array
40   size_t binarySearch(const int b[], int searchKey, size_t low, size_t high)
41   {
42      // loop until low index is greater than high index
43      while (low <= high) {
44
45         // determine middle element of subarray being searched
46         size_t middle = (low + high) / 2;
47
```

Binary Search – C code (3)

```
48          // display subarray used in this loop iteration
49          printRow(b, low, middle, high);
50
51          // if searchKey matched middle element, return middle
52          if (searchKey == b[middle]) {
53             return middle;
54          }
55
56          // if searchKey is less than middle element, set new high
57          else if (searchKey < b[middle]) {
58             high = middle - 1; // search low end of array
59          }  if
60
61          // if searchKey is greater than middle element, set new low
62          else {
63             low = middle + 1; // search high end of array
64          }
65       } // end while
66
67       return -1; // searchKey not found
68    }
69
```

Binary Search – C code (4)

```
70  // Print a header for the output
71  void printHeader(void)
72  {
73     puts("\nIndices:");
74
75     // output column head
76     for (unsigned int i = 0; i < SIZE; ++i) {
77        printf("%3u ", i);
78     }
79
80     puts(""); // start new line of output
81
82     // output line of - characters
83     for (unsigned int i = 1; i <= 4 * SIZE; ++i) {
84        printf("%s", "-");
85     }
86
87     puts(""); // start new line of output
88  }
89
```

Binary Search – C code (5)

```
90  // Print one row of output showing the current
91  // part of the array being processed.
92  void printRow(const int b[], size_t low, size_t mid, size_t high)
93  {
94     // loop through entire array
95     for (size_t i = 0; i < SIZE; ++i) {
96
97        // display spaces if outside current subarray range
98        if (i < low || i > high) {
99           printf("%s", "    ");
100       }
101       else if (i == mid) { // display middle element
102          printf("%3d*", b[i]); // mark middle value
103       }
104       else { // display other elements in subarray
105          printf("%3d ", b[i]);
106       }
107    }
108
109    puts(""); // start new line of output
110 }
```

Binary Search – C code (6)

```
Enter a number between 0 and 28: 25

Indices:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-----------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                                 24  26* 28
                                                 24*

25 not found
```

Binary Search – C code (7)

```
Enter a number between 0 and 28: 8

Indices:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-----------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

8 found at index 4
```

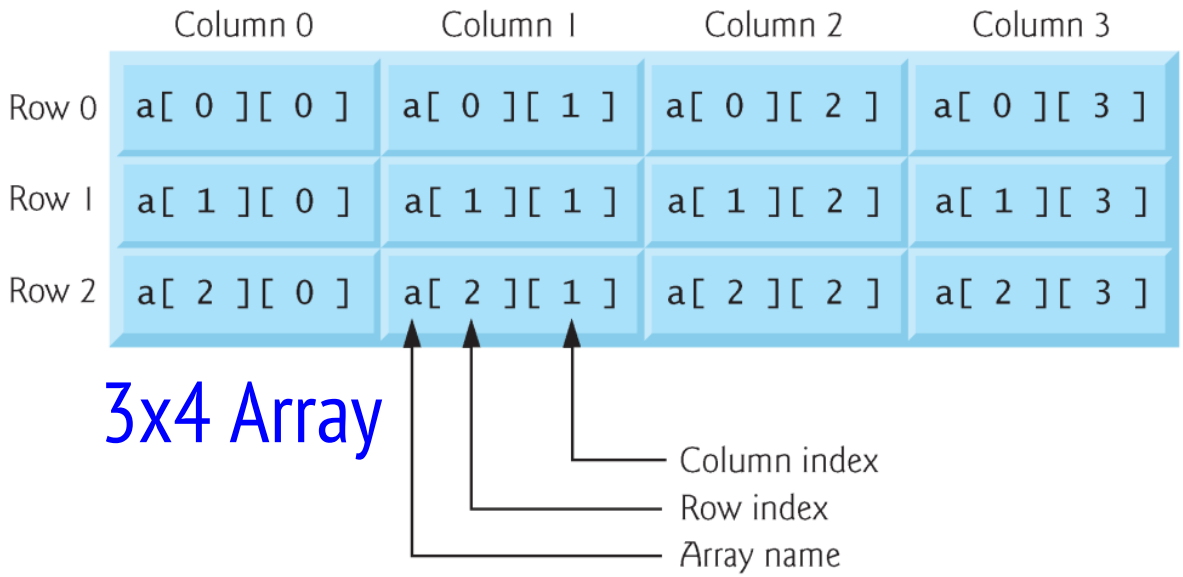
```
Enter a number between 0 and 28: 6

Indices:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-----------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found at index 3
```

Multidimensional Arrays

- Arrays in C can have multiple indices.
- A common use of multidimensional arrays is to represent tables of values consisting of information arranged in *rows* and *columns*.
- Multidimensional arrays can have more than two indices.



Initialization

- Where it is defined
  - Braces for each dimension
    - `int b[2][2] = {{1, 2}, {3, 4}};`
  - If there are not enough initializers for a given row, the remaining elements of that row are initialized to `0`.
    - `int b[2][2] = {{1}, {3, 4}};`
  - If the braces around each sublist are removed from the `array1` initializer list, the compiler initializes the elements of the first row followed by the elements of the second row.
    - `int b[2][2] = {1, 2, 3, 4};`

Multidimensional Array Example Code (1)

```
1  // Fig. 6.21: fig06_21.c
2  // Initializing multidimensional arrays.
3  #include <stdio.h>
4
5  void printArray(int a[][3]); // function prototype
6
7  // function main begins program execution
8  int main(void)
9  {
10    int array1[2][3] = {{1, 2, 3}, {4, 5, 6}};
11    puts("Values in array1 by row are:");
12    printArray(array1);
13
14    int array2[2][3] = {1, 2, 3, 4, 5};
15    puts("Values in array2 by row are:");
16    printArray(array2);
17
18    int array3[2][3] = {{1, 2}, {4}};
19    puts("Values in array3 by row are:");
20    printArray(array3);
21 }
22
```

Multidimensional Array Example Code (2)

```
23  // function to output array with two rows and three columns
24  void printArray(int a[][3])
25  {
26     // loop through rows
27     for (size_t i = 0; i <= 1; ++i) {
28
29        // output column values
30        for (size_t j = 0; j <= 2; ++j) {
31           printf("%d ", a[i][j]);
32        }
33
34        printf("\n"); // start new line of output
35     }
36  }
```

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

Two Dimensional Array Manipulation

- Example
  - `studentGrades[3][4]`
  - Row of the array represents a student.
  - Column represents a grade on one of the four exams the students took during the semester.
- The array manipulations are performed by four functions.
  - Function `minimum` determines the lowest grade of any student for the semester.
  - Function `maximum` determines the highest grade of any student for the semester.
  - Function `average` determines a particular student's semester average.
  - Function `printArray` outputs the two-dimensional array in a neat, tabular format.

2D Array Manipulation Code (1)

```
1  // Fig. 6.22: fig06_22.c
2  // Two-dimensional array manipulations.
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  // function prototypes
8  int minimum(const int grades[][EXAMS], size_t pupils, size_t tests);
9  int maximum(const int grades[][EXAMS], size_t pupils, size_t tests);
10 double average(const int setOfGrades[], size_t tests);
11 void printArray(const int grades[][EXAMS], size_t pupils, size_t tests);
12
13 // function main begins program execution
14 int main(void)
15 {
16    // initialize student grades for three students (rows)
17    int studentGrades[STUDENTS][EXAMS] =
18       { { 77, 68, 86, 73 },
19         { 96, 87, 89, 78 },
20         { 70, 90, 86, 81 } };
21
22    // output array studentGrades
23    puts("The array is:");
24    printArray(studentGrades, STUDENTS, EXAMS);
```

2D Array Manipulation Code (2)

```
25
26      // determine smallest and largest grade values
27      printf("\n\nLowest grade: %d\nHighest grade: %d\n",
28         minimum(studentGrades, STUDENTS, EXAMS),
29         maximum(studentGrades, STUDENTS, EXAMS));
30
31      // calculate average grade for each student
32      for (size_t student = 0; student < STUDENTS; ++student) {
33         printf("The average grade for student %u is %.2f\n",
34            student, average(studentGrades[student], EXAMS));
35      }
36   }
37
```

2D Array Manipulation Code (3)

```
38  // Find the minimum grade
39  int minimum(const int grades[][EXAMS], size_t pupils, size_t tests)
40  {
41     int lowGrade = 100; // initialize to highest possible grade
42
43     // loop through rows of grades
44     for (size_t i = 0; i < pupils; ++i) {
45
46        // loop through columns of grades
47        for (size_t j = 0; j < tests; ++j) {
48
49           if (grades[i][j] < lowGrade) {
50              lowGrade = grades[i][j];
51           }
52        }
53     }
54
55     return lowGrade; // return minimum grade
56  }
57
```

2D Array Manipulation Code (4)

```
58  // Find the maximum grade
59  int maximum(const int grades[][EXAMS], size_t pupils, size_t tests)
60  {
61     int highGrade = 0; // initialize to lowest possible grade
62
63     // loop through rows of grades
64     for (size_t i = 0; i < pupils; ++i) {
65
66        // loop through columns of grades
67        for (size_t j = 0; j < tests; ++j) {
68
69           if (grades[i][j] > highGrade) {
70              highGrade = grades[i][j];
71           }
72        }
73     }
74
75     return highGrade; // return maximum grade
76  }
77
```

2D Array Manipulation Code (5)

```
78  // Determine the average grade for a particular student
79  double average(const int setOfGrades[], size_t tests)
80  {
81     int total = 0; // sum of test grades
82
83     // total all grades for one student
84     for (size_t i = 0; i < tests; ++i) {
85        total += setOfGrades[i];
86     }
87
88     return (double) total / tests; // average
89  }
90
```

2D Array Manipulation Code (6)

```
91  // Print the array
92  void printArray(const int grades[][EXAMS], size_t pupils, size_t tests)
93  {
94     // output column heads
95     printf("%s", "                  [0]  [1]  [2]  [3]");
96
97     // output grades in tabular format
98     for (size_t i = 0; i < pupils; ++i) {
99
100       // output label for row
101       printf("\nstudentGrades[%u] ", i);
102
103       // output grades for one student
104       for (size_t j = 0; j < tests; ++j) {
105          printf("%-5d", grades[i][j]);
106       }
107    }
108 }
```

2D Array Manipulation Code (7)

```
The array is:
                  [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

Classroom Assignment

- Matrix Addition/Subtraction – two matrices should have same number of rows and columns.

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}+\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

$$=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

Addition

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix}+\begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix}=\begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix}=\begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

Subtraction

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix}-\begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix}=\begin{bmatrix} 1-0 & 3-0 \\ 1-7 & 0-5 \\ 1-2 & 2-1 \end{bmatrix}=\begin{bmatrix} 1 & 3 \\ -6 & -5 \\ -1 & 1 \end{bmatrix}$$

Variable Length Array

- In early versions of C, all arrays had constant size.
- If size is unknown at compilation time
  - Use dynamic memory allocation with `malloc`

- The C standard allows a variable-length array
  - An array whose length, or size, is defined in terms of an expression evaluated at execution time.

Variable Length Array Code (1)

```
1  // Fig. 6.23: fig06_23.c
2  // Using variable-length arrays in C99
3  #include <stdio.h>
4
5  // function prototypes
6  void print1DArray(size_t size, int array[size]);
7  void print2DArray(int row, int col, int array[row][col]);
8
9  int main(void)
10 {
11    printf("%s", "Enter size of a one-dimensional array: ");
12    int arraySize; // size of 1-D array
13    scanf("%d", &arraySize);
14
15    int array[arraySize]; // declare 1-D variable-length array
16
17    printf("%s", "Enter number of rows and columns in a 2-D array: ");
18    int row1, col1; // number of rows and columns in a 2-D array
19    scanf("%d %d", &row1, &col1);
20
21    int array2D1[row1][col1]; // declare 2-D variable-length array
22
```

Variable Length Array Code (2)

```
23      printf("%s",
24         "Enter number of rows and columns in another 2-D array: ");
25      int row2, col2; // number of rows and columns in another 2-D array
26      scanf("%d %d", &row2, &col2);
27
28      int array2D2[row2][col2]; // declare 2-D variable-length array
29
30      // test sizeof operator on VLA
31      printf("\nsizeof(array) yields array size of %d bytes\n",
32         sizeof(array));
33
34      // assign elements of 1-D VLA
35      for (size_t i = 0; i < arraySize; ++i) {
36         array[i] = i * i;
37      }
38
39      // assign elements of first 2-D VLA
40      for (size_t i = 0; i < row1; ++i) {
41         for (size_t j = 0; j < col1; ++j) {
42            array2D1[i][j] = i + j;
43         }
44      }
45
```

Variable Length Array Code (3)

```
46      // assign elements of second 2-D VLA
47      for (size_t i = 0; i < row2; ++i) {
48         for (size_t j = 0; j < col2; ++j) {
49            array2D2[i][j] = i + j;
50         }
51      }
52
53      puts("\nOne-dimensional array:");
54      print1DArray(arraySize, array); // pass 1-D VLA to function
55
56      puts("\nFirst two-dimensional array:");
57      print2DArray(row1, col1, array2D1); // pass 2-D VLA to function
58
59      puts("\nSecond two-dimensional array:");
60      print2DArray(row2, col2, array2D2); // pass other 2-D VLA to function
61   }
62
63   void print1DArray(size_t size, int array[size])
64   {
65      // output contents of array
66      for (size_t i = 0; i < size; i++) {
67         printf("array[%d] = %d\n", i, array[i]);
68      }
69   }
```

Variable Length Array Code (4)

```
70
71  void print2DArray(size_t row, size_t col, int array[row][col])
72  {
73     // output contents of array
74     for (size_t i = 0; i < row; ++i) {
75        for (size_t j = 0; j < col; ++j) {
76           printf("%5d", array[i][j]);
77        }
78
79        puts("");
80     }
81  }
```

Variable Length Array Code (5)

```
Enter size of a one-dimensional array: 6
Enter number of rows and columns in a 2-D array: 2 5
Enter number of rows and columns in another 2-D array: 4 3

sizeof(array) yields array size of 24 bytes

One-dimensional array:
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25

First two-dimensional array:
    0    1    2    3    4
    1    2    3    4    5

Second two-dimensional array:
    0    1    2
    1    2    3
    2    3    4
    3    4    5
```