

ECE 220 Computer Systems & Programming

Lecture 13 – Problem Solving with Pointers and Arrays

February 28, 2019



Exercise: implement a function that interchanges two rows of a 5x5 matrix. The function takes three arguments: pointer to the matrix, row number x and row number y.

```
#define SIZE 5  
void row_interchange(int matrix[SIZE][SIZE], int x, int y){  
  
}
```

Note:

- 2D arrays
 - `<type> <name> [<dim1>][<dim2>];`
 - e.g., `int matrix[5][5];`

*multi-dimensional array is stored in **row-major** order

linear index = row x (width of row) + col
(i.e. `matrix[3][2] = 3*5+2 = 17`)

```

1  #include <stdio.h>
2  #define SIZE 5
3  //function to interchange 2 rows (x,y) in a 5x5 matrix
4
5  int matrix_ptr(int *matrix, int x, int y);
6  int main()
7  {
8      int i, j;
9      int matrix[SIZE][SIZE];
10
11     printf("Initial Matrix: \n");
12     for(i=0;i<SIZE;i++)
13     {
14         for(j=0;j<SIZE;j++)
15         {
16             matrix[i][j]= i*SIZE+j;
17             printf("%d ", matrix[i][j]);
18         }
19         printf("\n");
20     }
21     printf("New Matrix: \n");
22     //int rc = matrix_change(matrix, 2, 4);
23     int rc = matrix_ptr(&matrix[0][0], 2, 4);
24     if (rc != 0){
25         printf("exchange row index out of bound\n");
26         return rc;
27     }

```

main Function (continue..)

```
29     for (i=0; i<SIZE; i++)
30     {
31         for (j=0; j<SIZE; j++)
32         {
33             printf("%d ", matrix[i][j]);
34         }
35         printf("\n");
36     }
37     return 0;
38 }
```

```
40 int matrix_ptr(int *matrix, int x, int y){
41     //if x and y is greater than 5 or less than 0, just exit and return 1
42     if((x>SIZE-1) || (y>SIZE-1) || (x<0) || (y<0))
43         return 1;
44
45     int i, j, temp;
46     for (j=0; j<SIZE; j++){
47         temp=matrix[x*SIZE+j];
48         matrix[x*SIZE+j] = matrix[y*SIZE+j];
49         matrix[y*SIZE+j] = temp;
50     }
51     return 0;
52 }
```

Function without pointer argument:

```
int matrix_change(int matrix[SIZE][SIZE], int x, int y);
```

```
int matrix_change(int matrix[SIZE][SIZE], int x, int y)
{
    //if x and y is greater than 5 or less than 0, just exit and return 1
    if((x>SIZE-1) || (y>SIZE-1) || (x<0) || (y<0))
        return 1;

    int j, temp=0;
    for(j=0;j<SIZE;j++)
    {
        temp = matrix[x][j];
        matrix[x][j] = matrix[y][j];
        matrix[y][j] = temp;
    }
    return 0;
}
```

Pointers & Array Recap

- Reverse an integer array: `void array_reverse(int array[], int size);`

10	30	50	70
----	----	----	----

Index: 0 1 2 3

10	30	50	70	90
----	----	----	----	----

Index: 0 1 2 3 4

```
1  #include <stdio.h>
2  void array_reverse(int array[], int n);
3  void print_array(int array[], int n);
4
5  int main() {
6      int n;
7
8      printf("Enter the size of array: ");
9      scanf("%d", &n);
10
11     int array[n];
12     int i;
13
14     printf("Set each element for this array\n");
15     for(i=0; i<n; i++) {
16         printf("Input number %d: ", i);
17         scanf("%d", &array[i]);
18     }
19
20     printf("Array before reverse:\n");
21     print_array(array, n);
22
23     array_reverse(array, n);
24
25     printf("Array after reverse:\n");
26     print_array(array, n);
27
28     return 0;
29 }
```

```

31 void array_reverse(int array[], int n){
32     int i, temp;
33
34     for(i=0;i<(n/2);i++){
35         temp = array[i];
36         array[i] = array[n-i-1];
37         array[n-i-1] = temp;
38     }
39 }
40
41 void print_array(int array[], int n){
42     int i;
43     for(i=0;i<n;i++){
44         printf("%d ", array[i]);
45     }
46     printf("\n");
47 }

```

How about reversing a string of unknown size? See the Text, page 445 (code on github)

Multi-dimensional Arrays Recap

		Column 1	Column 2	Column 3
int a [2][3];	Row 1	a[0][0]	a[0][1]	a[0][2]
	Row 2	a[1][0]	a[1][1]	a[1][2]

In memory		index
	a[0][0]	0
	a[0][1]	1
	a[0][2]	2
	a[1][0]	3
	a[1][1]	4
	a[1][2]	5

* multi-dimensional array is stored in **row-major** order

Exercise:

Implement a function that transpose an $n \times m$ matrix

```
1  #include <stdio.h>
2  #define ROW 2
3  #define COL 5
4  void transpose2(int *in, int *out);
5  void transpose(int in[ROW][COL], int out[COL][ROW]);
6  void print_matrix(int *matrix, int row, int col);
7
8  int main() {
9
10     int in_array[ROW][COL], out_array[COL][ROW];
11
12     //Set in_array value
13     int i,j;
14     for(i=0;i<ROW;i++){
15         for(j=0;j<COL;j++){
16             in_array[i][j]=i*COL+j;
17         }
18     }
19     //Print in_array value
20     printf("Input Array: \n");
21     print_matrix(&in_array[0][0], ROW, COL);
22     //Perform transpose
23     //transpose(in_array, out_array);
24     transpose2(&in_array[0][0], &out_array[0][0]);
25     //Print out_array value
26     printf("Output Array: \n");
27     print_matrix(&out_array[0][0], COL, ROW);
28     return 0;
29 }
```

```
#define ROW 2
#define COL 5
void transpose2(int *in_matrix, int *out_matrix){
```

Solution:

```
41 void transpose2(int *in_array, int *out_array){
42
43     int i,j;
44     for(i=0;i<ROW;i++){
45         for(j=0;j<COL;j++){
46             out_array[j*ROW+i] = in_array[i*COL+j];
47         }
48     }
49 }
```

```
}
```

Exercise: implement a function that transpose an n x n matrix

```
#define ROW 2
#define COL 5
void transpose(int in_matrix[ROW][COL], int out_matrix[COL][ROW]) {
```

Simpler Solution:

```
31 void transpose(int in_array[ROW][COL], int out_array[COL][ROW]) {
32
33     int i,j;
34     for(i=0;i<ROW;i++) {
35         for(j=0;j<COL;j++) {
36             out_array[j][i] = in_array[i][j];
37         }
38     }
39 }
```

```
void print_matrix(int *matrix, int row, int col) {
    int i,j;
    for(i=0;i<row;i++) {
        for(j=0;j<col;j++) {
            printf("%d ", matrix[i*col+j]);
        }
        printf("\n");
    }
}
```

Print Function:

1. Pointer Array & Pointer to an Array

```
int a[4];  
int b[5];  
int *ptr_array[2];  
ptr_array[0] = &a[0]; /* ptr_array[0] = a; */  
ptr_array[1] = &b[0]; /* ptr_array[1] = b; */
```

or

```
int a[4];  
int b[5];  
int *ptr_array[2] = {a,b};
```

2. Search Algorithms

Linear Search: search from the beginning of the array until item is found

Binary Search: (for sorted array)

- 1) find the middle of the array and check if it's the search item;
- 2) search the first half if the search item is smaller than the center item, else search the second half;
- 3) repeat step 1 & 2 until search item is found.

If searching for 23 in the 10-element array:

	2	5	8	12	16	23	38	56	72	91
23 > 16, take 2 nd half	L				16	23			H	
23 < 56, take 1 st half						23		56		
Found 23, Return 5						23				


```

1  #include <stdio.h>
2  #define LENGTH 8
3
4  int linearsearch(int array[], int item);
5  int binarysearch(int array[], int item);
6
7  int main()
8  {
9      int array[LENGTH] = {2,3,5,6,8,9,10,13};
10
11     //int idx = linearsearch(array, array[5]);
12     int idx = binarysearch(array, array[5]);
13
14     printf("item is found at index %d \n", idx);
15     return 0;
16 }
17 //Simple Solution - not efficient:
18 int linearsearch(int array[], int item)
19 {
20     int i;
21     for(i=0;i<LENGTH;i++)
22     {
23         if(array[i] == item)
24             return i;
25     }
26
27     return -1; //item not found
28 }

```


Exercise: implement a function that performs binary search

This function takes two arguments: a pointer to the sorted array and the search item. If the search item is found, the function returns its index in the array. Otherwise, it returns -1.

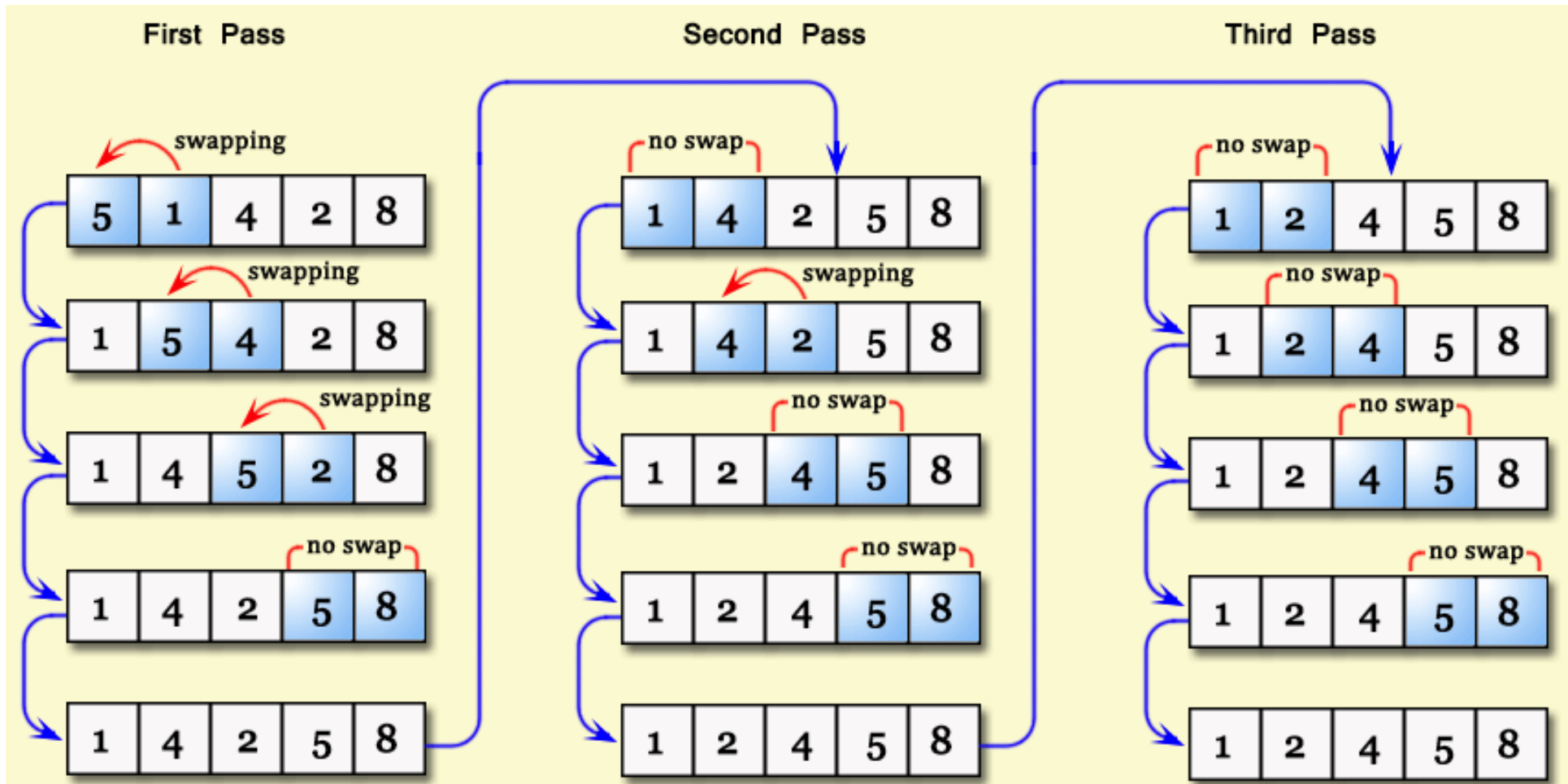
```
int binarysearch(int array[], int item)
{
    int n = LENGTH;
    int start=0, end=n-1;
    int middle;

    while(start <= end){
        middle = (start+end)/2;
        if(array[middle] == item){
            return middle;
        }
        else if(array[middle] > item){
            //search lower half if middle value greater than search item
            end = middle-1;
        }
        else{
            //search upper half if middle value smaller than search item
            start = middle+1;
        }
    }

    return -1; //item not found
}
```

3. Sorting Algorithms (<http://visualgo.net/sorting>)

Bubble Sort: 1) compare items next to each other and swap them if needed;
2) repeat this process until the entire array is sorted.



Exercise: implement a function that performs bubble sort

```
1  #include <stdio.h>
2  #define SIZE 5
3
4  int main()
5  {
6      int n = SIZE-1;
7      int array[] = {7,4,6,3,1};
8
9      int i, temp, swap = 0;
10
11     //sort number in ascending order
12     do
13     {
14         swap = 0;
15         for(i=0;i<n;i++)
16         {
17             //swap the two numbers if order is incorrect
18             if(array[i]>array[i+1])
19             {
20                 temp = array[i];
21                 array[i] = array[i+1];
22                 array[i+1] = temp;
23                 //set the swap flag
24                 swap = 1;
25             }
26         }
27         n--;
28     }while(swap != 0);
29
30     printf("sorted array: \n");
31     for(i=0;i<SIZE;i++){
32         printf("%d ", array[i]);
33     }
34     printf("\n");
35
36     return 0;
37 }
```

Insertion Sort:

- 1) remove item from array, insert it at the proper location in the sorted part by shifting other items;
- 2) repeat this process until the end of array is reach.

Step 1

Assume first item is "sorted"

5	2	6	1	3	9
---	---	---	---	---	---

Step 2

Identify the value to compare

5		6	1	3	9
---	--	---	---	---	---

2

Step 3

Since $5 > 2$, shift 5 over to create space for 2 in the sorted section

	5	6	1	3	9
--	---	---	---	---	---

2

Step 4

Insert 2 into the empty space in the sorted section

2	5	6	1	3	9
---	---	---	---	---	---

Exercise: implement a function that performs insertion sort

```
1  #include <stdio.h>
2  #define SIZE 5
3  int main()
4  {
5      int array[] = {7,4,6,3,1};
6
7      //sort array in ascending order
8      int i, j, temp, empty = 0;
9      for(i=1;i<SIZE;i++)
10     {
11         temp = array[i];
12         for(j=i-1;j>=0;j--)
13         {
14             if(temp < array[j])
15             {
16                 //shift element to the right
17                 array[j+1] = array[j];
18                 //update empty position
19                 empty = j;
20             }
21         }
22         //insert at the proper location
23         array[empty] = temp;
24     }
25
26     printf("sorted array: \n");
27     for(i=0;i<SIZE;i++){
28         printf("%d ", array[i]);
29     }
30     printf("\n");
31
32     return 0;
33 }
```