ECE 220 Computer Systems & Programming

Lecture 13 – Problem Solving with Pointers and Arrays

February 28, 2019

ECE ILLINOIS

ILLINOIS

Exercise: implement a function that interchanges two rows of a 5x5 matrix. The functions 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){

}
```

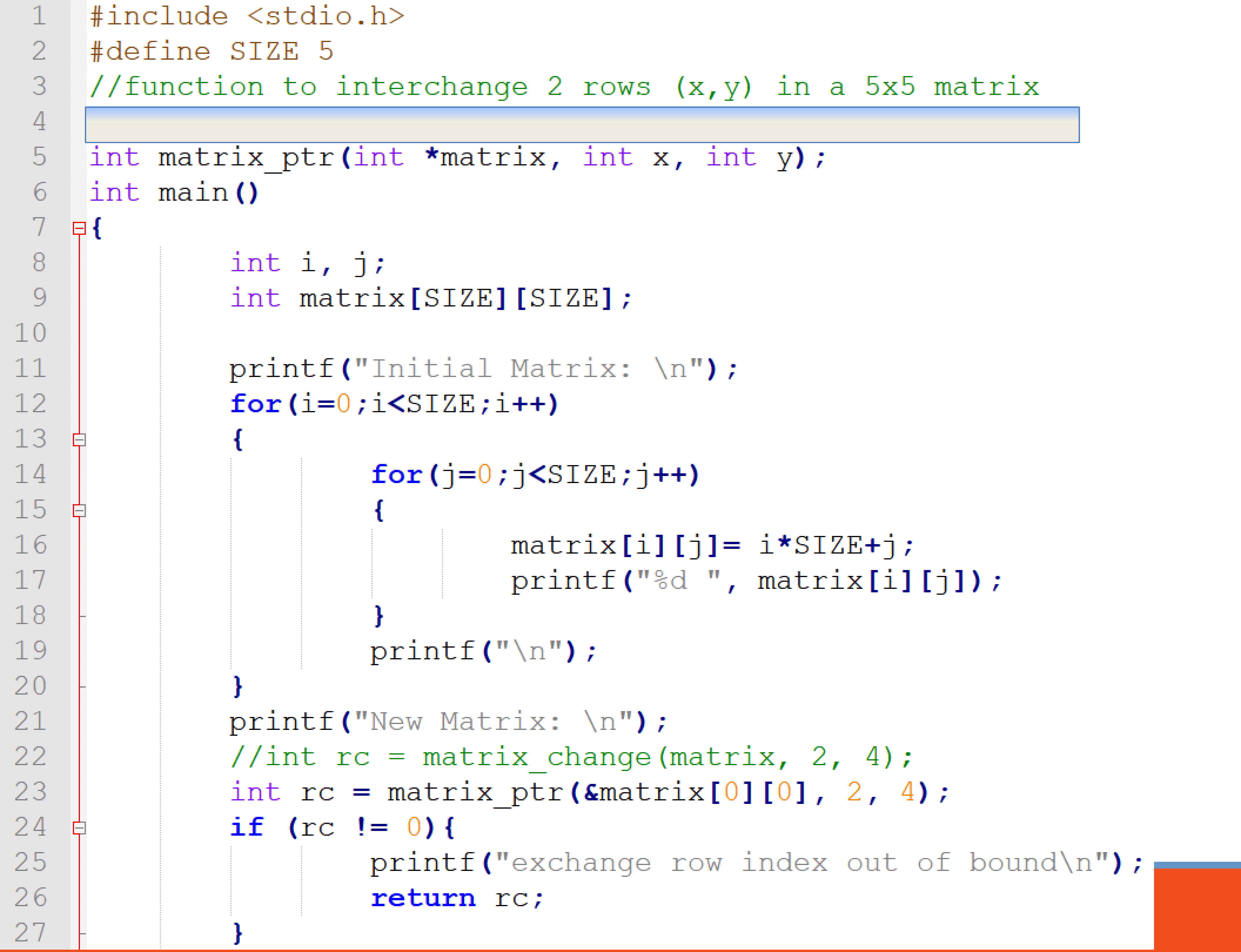
Note:

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

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

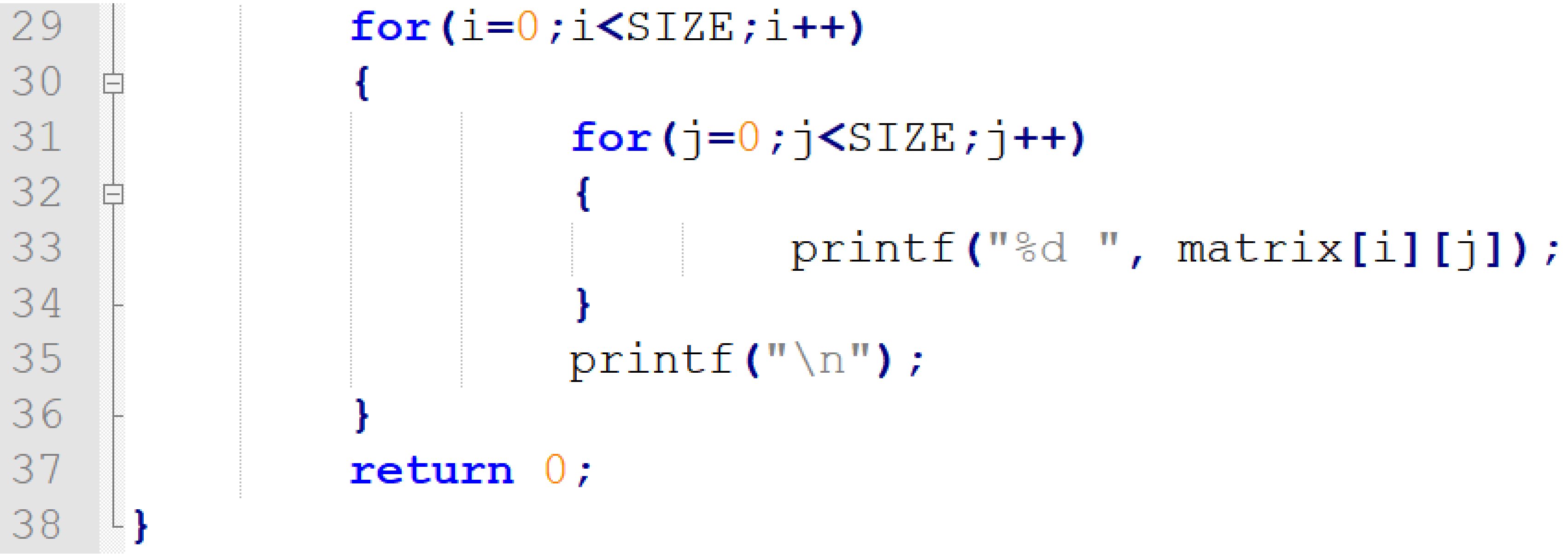
linear index = row x (width of row) + col

(i.e. matrix[3][2] = 3*5+2 = 17)

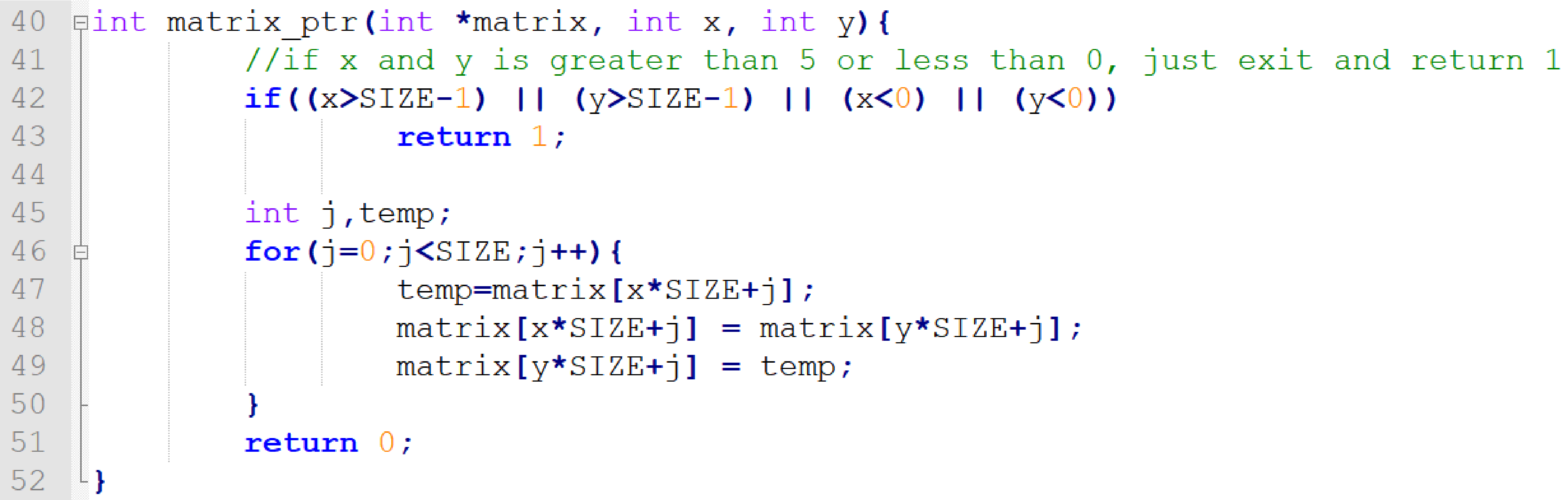


```c
#include <stdio.h>
#define SIZE 5
//function to interchange 2 rows (x,y) in a 5x5 matrix

int matrix_ptr(int *matrix, int x, int y);
int main()
{
        int i, j;
        int matrix[SIZE][SIZE];

        printf("Initial Matrix: \n");
        for(i=0;i<SIZE;i++)
        {
                for(j=0;j<SIZE;j++)
                {
                        matrix[i][j]= i*SIZE+j;
                        printf("%d ", matrix[i][j]);
                }
                printf("\n");
        }
        printf("New Matrix: \n");
        //int rc = matrix_change(matrix, 2, 4);
        int rc = matrix_ptr(&matrix[0][0], 2, 4);
        if (rc != 0){
                printf("exchange row index out of bound\n");
                return rc;
        }
```

main Function (continue..)

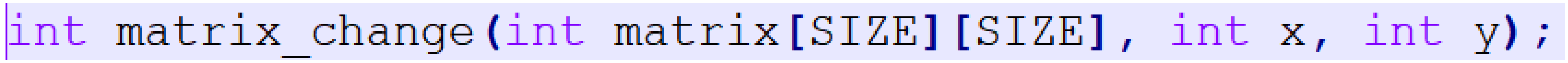


```
        for(i=0;i<SIZE;i++)
        {
            for(j=0;j<SIZE;j++)
            {
                printf("%d ", matrix[i][j]);
            }
            printf("\n");
        }
        return 0;
}
```

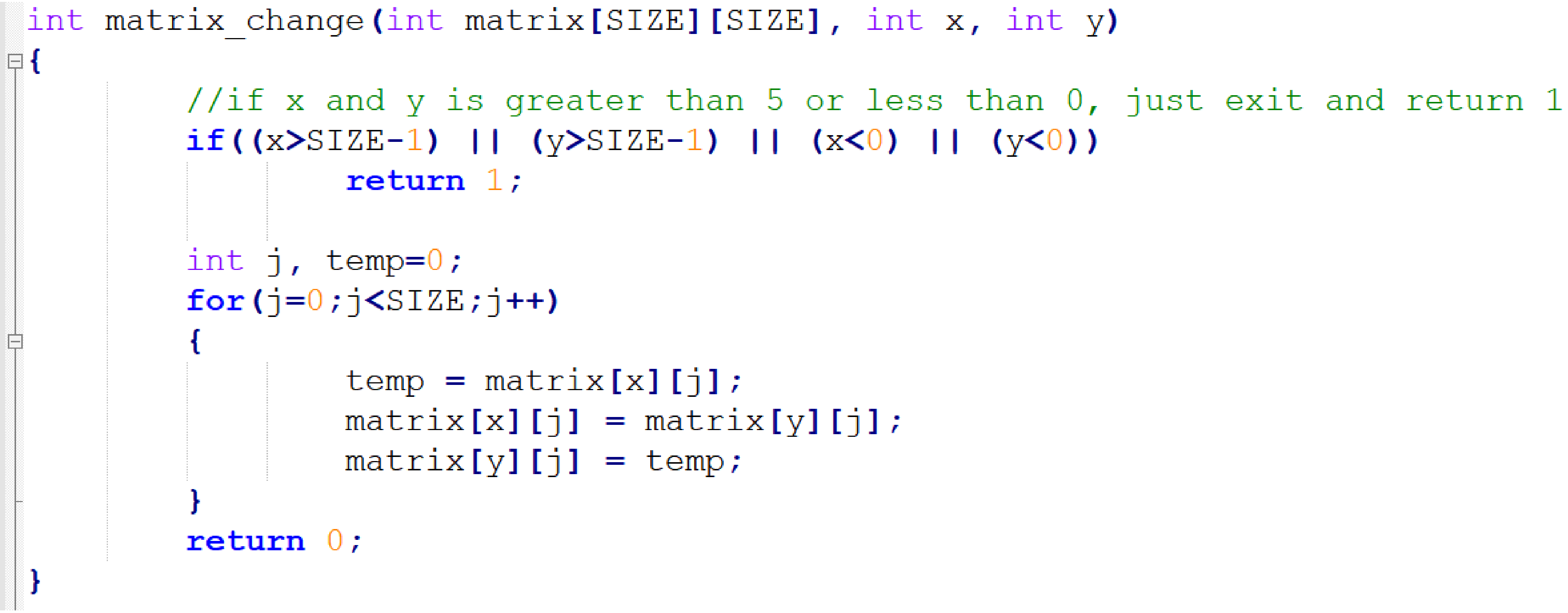


```
int matrix_ptr(int *matrix, int x, int y){
    //if x and y is greater than 5 or less than 0, just exit and return 1
    if((x>SIZE-1) || (y>SIZE-1) || (x<0) || (y<0))
        return 1;

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

Function without pointer argument:



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



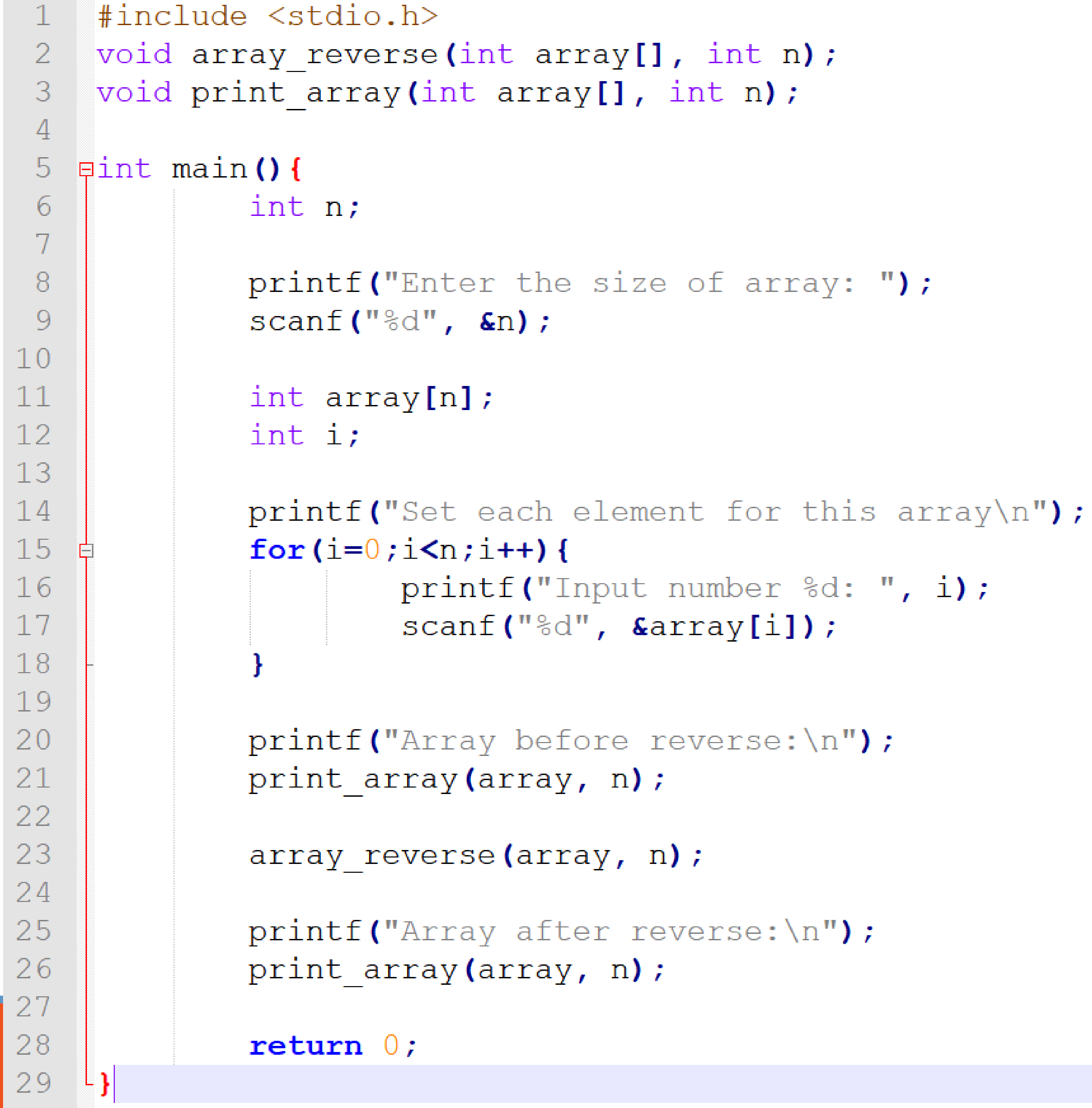
```c
int matrix_change(int matrix[SIZE][SIZE], int x, int y)
{
        //if x and y is greater than 5 or less than 0, just exit and return 1
        if((x>SIZE-1) || (y>SIZE-1) || (x<0) || (y<0))
                return 1;

        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

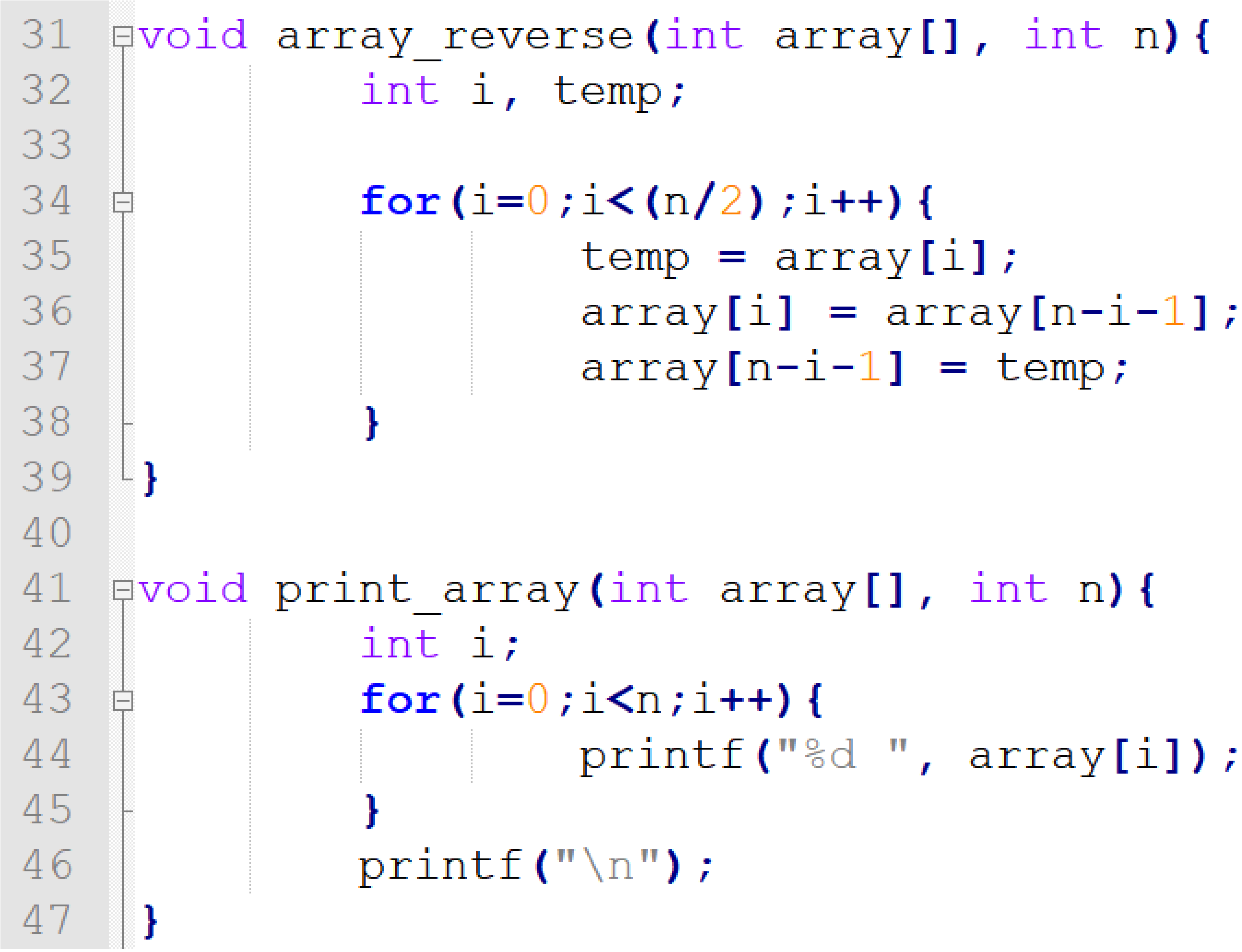
- Reverse an integer array: void array_reverse(int array[], int size);

	10	30	50	70
Index:	0	1	2	3

	10	30	50	70	90
Index:	0	1	2	3	4



```c
#include <stdio.h>
void array_reverse(int array[], int n);
void print_array(int array[], int n);

int main(){
    int n;

    printf("Enter the size of array: ");
    scanf("%d", &n);

    int array[n];
    int i;

    printf("Set each element for this array\n");
    for(i=0;i<n;i++){
        printf("Input number %d: ", i);
        scanf("%d", &array[i]);
    }

    printf("Array before reverse:\n");
    print_array(array, n);

    array_reverse(array, n);

    printf("Array after reverse:\n");
    print_array(array, n);

    return 0;
}
```



```c
void array_reverse(int array[], int n){
    int i, temp;

    for(i=0;i<(n/2);i++){
        temp = array[i];
        array[i] = array[n-i-1];
        array[n-i-1] = temp;
    }
}

void print_array(int array[], int n){
    int i;
    for(i=0;i<n;i++){
        printf("%d ", array[i]);
    }
    printf("\n");
}
```

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

Multi-dimensional Arrays Recap

int a [2][3];

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

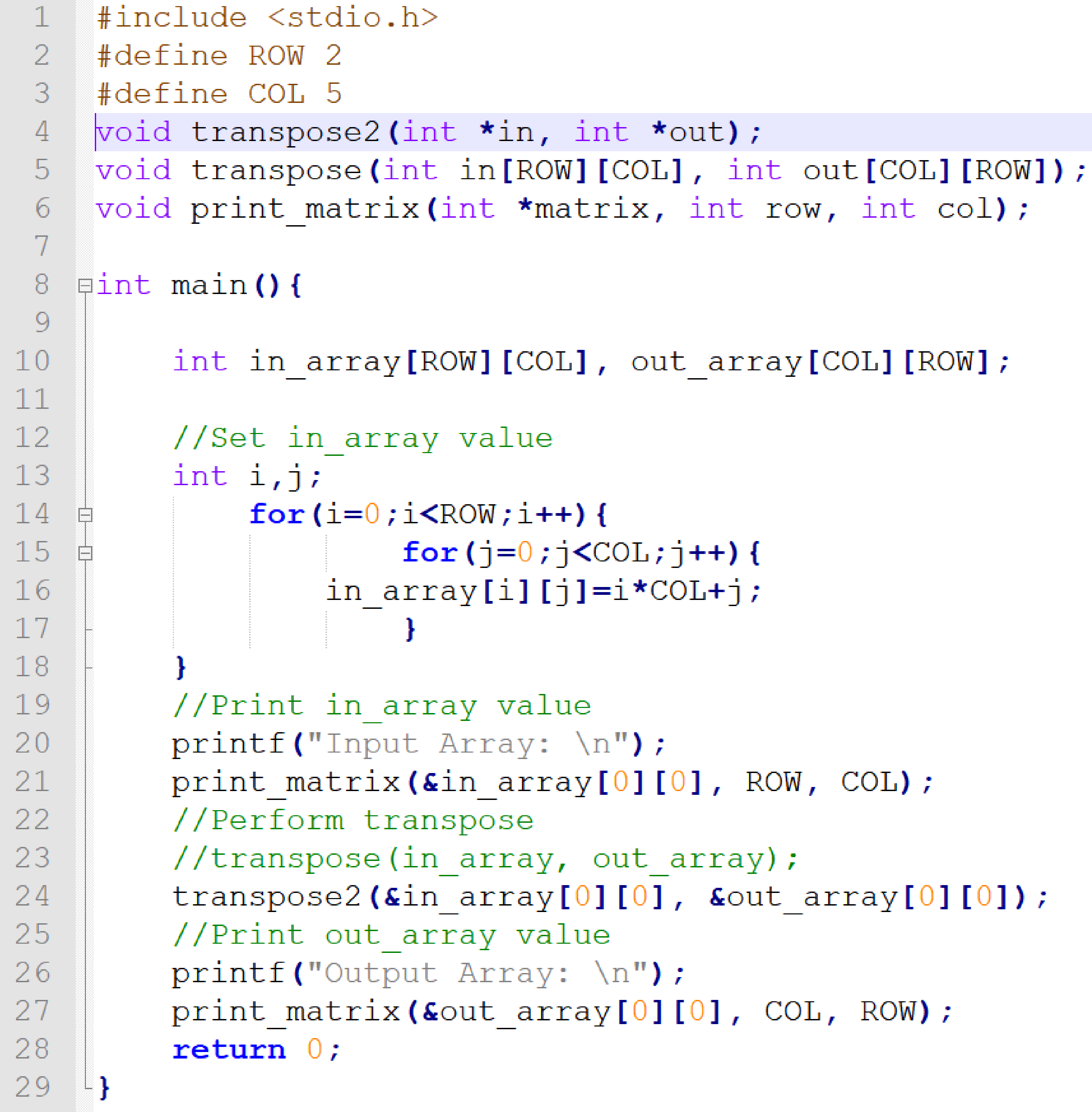
In memory

	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:

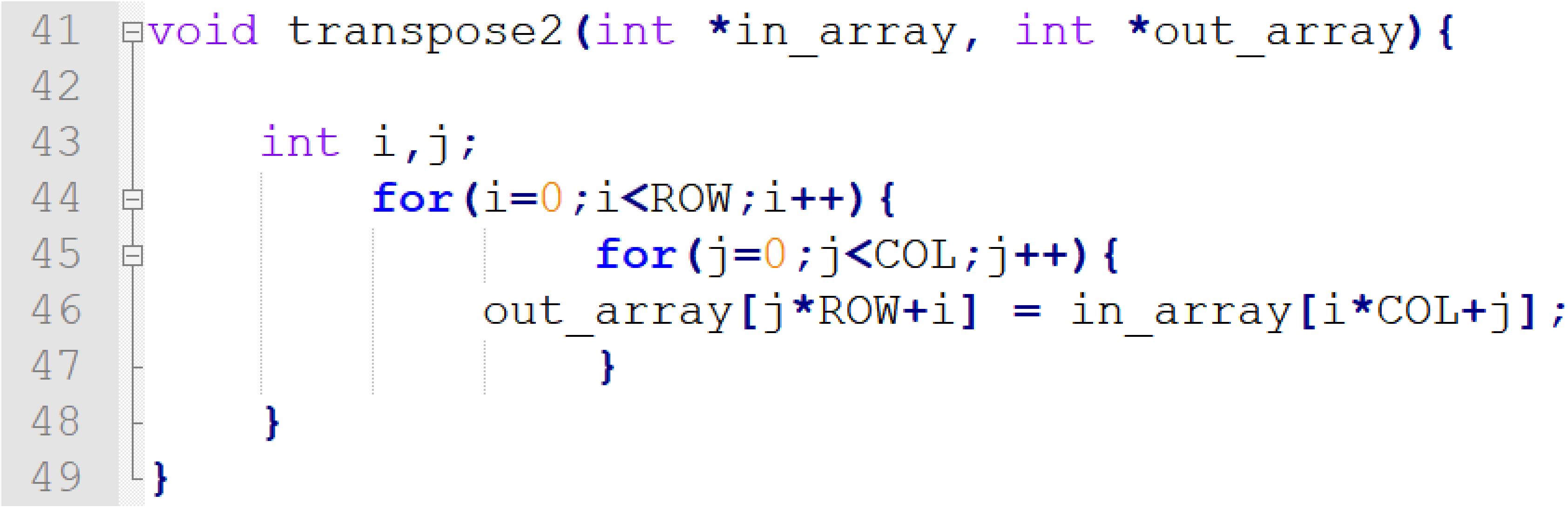
Implement a function that transpose an n x m matrix



```c
#include <stdio.h>
#define ROW 2
#define COL 5
void transpose2(int *in, int *out);
void transpose(int in[ROW][COL], int out[COL][ROW]);
void print_matrix(int *matrix, int row, int col);

int main(){

    int in_array[ROW][COL], out_array[COL][ROW];

    //Set in_array value
    int i,j;
        for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++){
            in_array[i][j]=i*COL+j;
                }
    }
    //Print in_array value
    printf("Input Array: \n");
    print_matrix(&in_array[0][0], ROW, COL);
    //Perform transpose
    //transpose(in_array, out_array);
    transpose2(&in_array[0][0], &out_array[0][0]);
    //Print out_array value
    printf("Output Array: \n");
    print_matrix(&out_array[0][0], COL, ROW);
    return 0;
}
```

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

Solution:

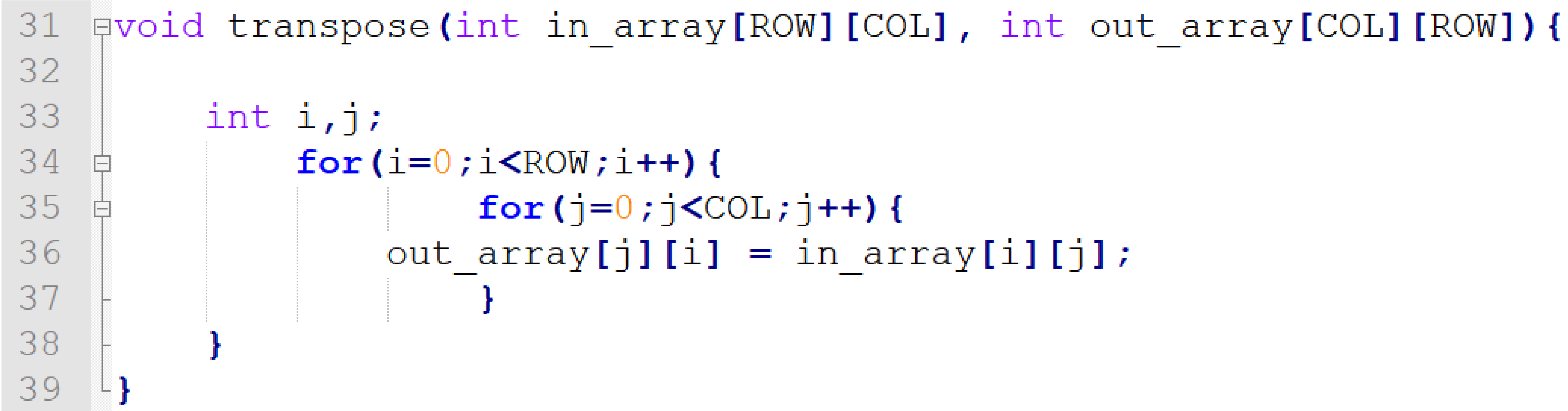


```
void transpose2(int *in_array, int *out_array){

    int i,j;
        for(i=0;i<ROW;i++){
                for(j=0;j<COL;j++){
            out_array[j*ROW+i] = in_array[i*COL+j];
                }
    }
}
```

```
}
```

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

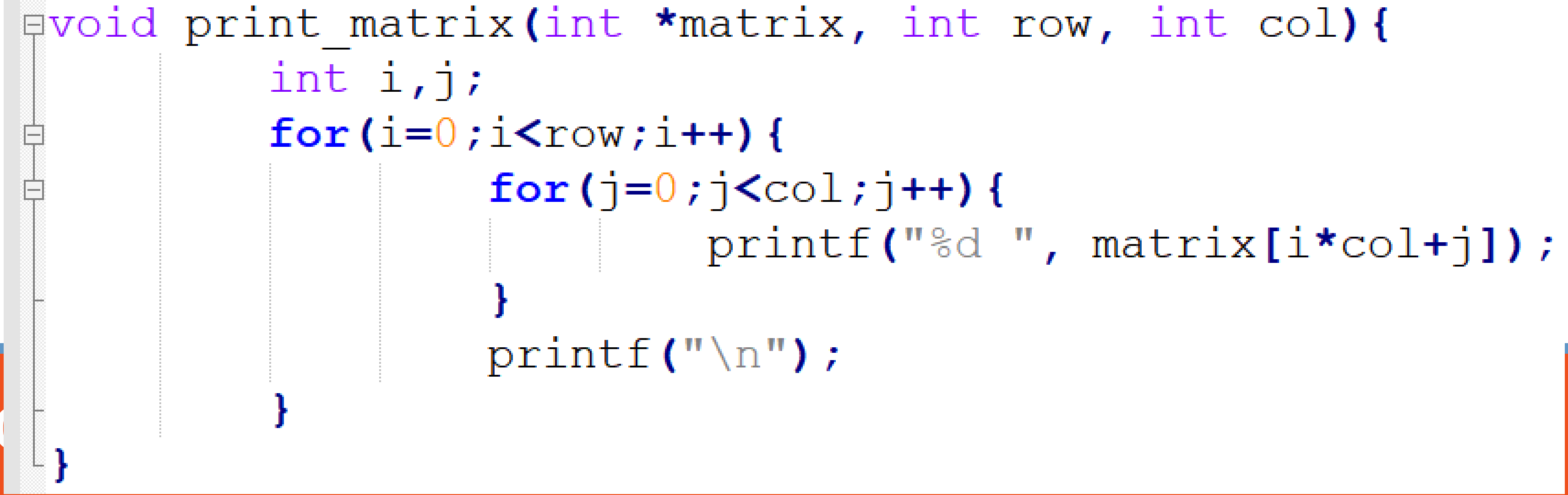
```
#define ROW 2
#define COL 5
void transpose(int in_matrix[ROW][COL], int out_matrix[COL][ROW]){

Simpler Solution:
```



```
void transpose(int in_array[ROW][COL], int out_array[COL][ROW]){

    int i,j;
        for(i=0;i<ROW;i++){
            for(j=0;j<COL;j++){
            out_array[j][i] = in_array[i][j];
            }
    }
}
```

```
}
```

Print Function:



```
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");
        }
}
```

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};
```

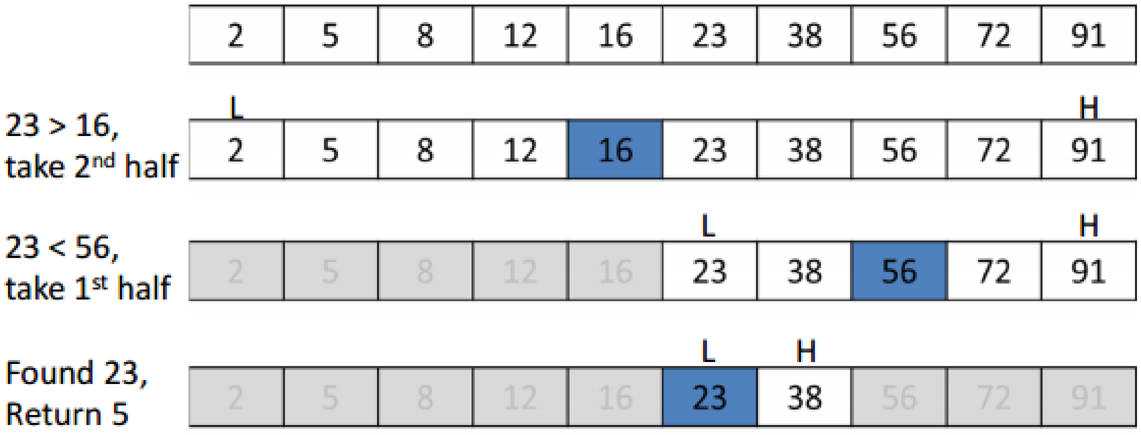
2. Search Algorithms

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:



23 > 16, take 2nd half

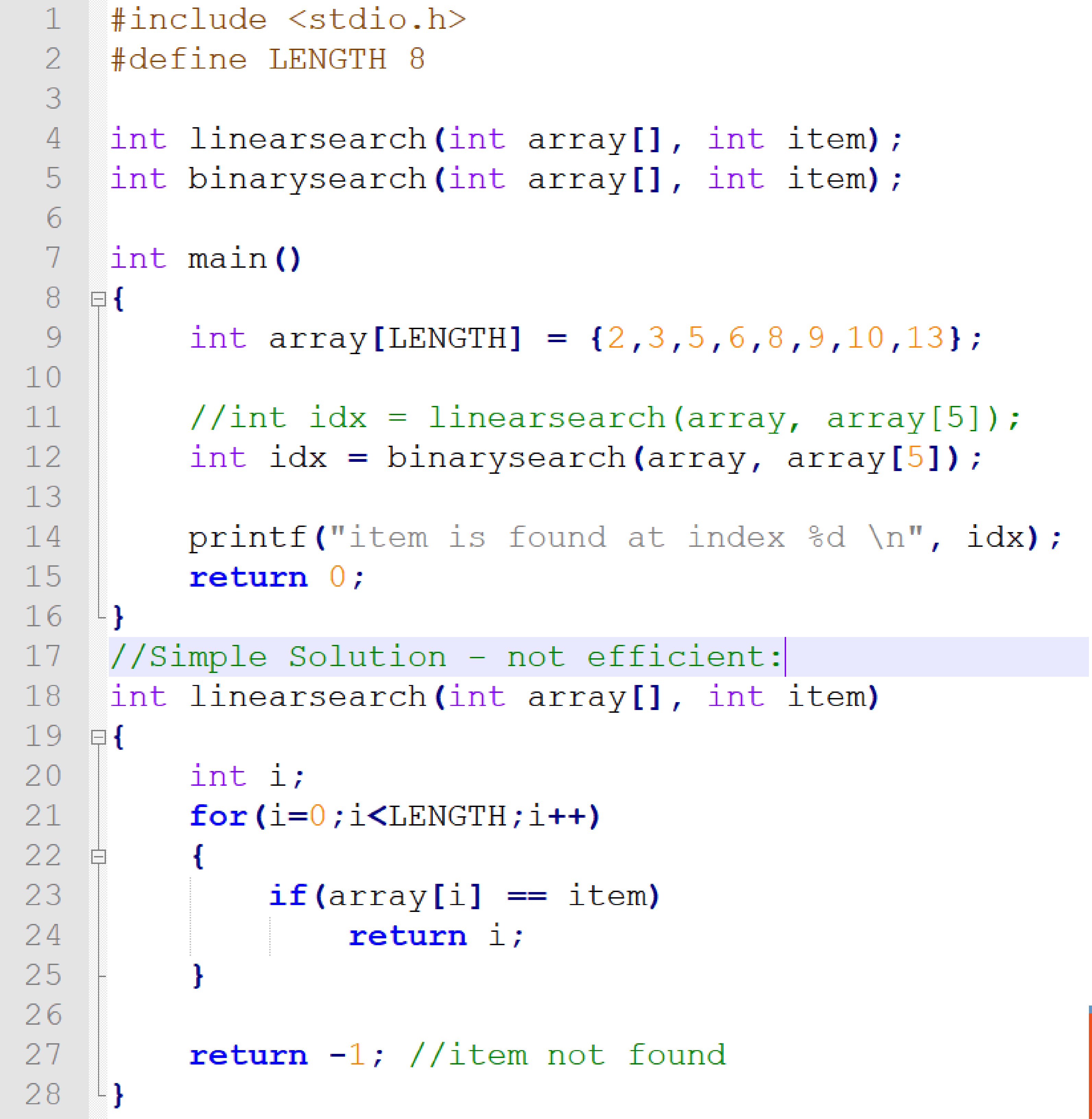
23 < 56, take 1st half

Found 23, Return 5

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.

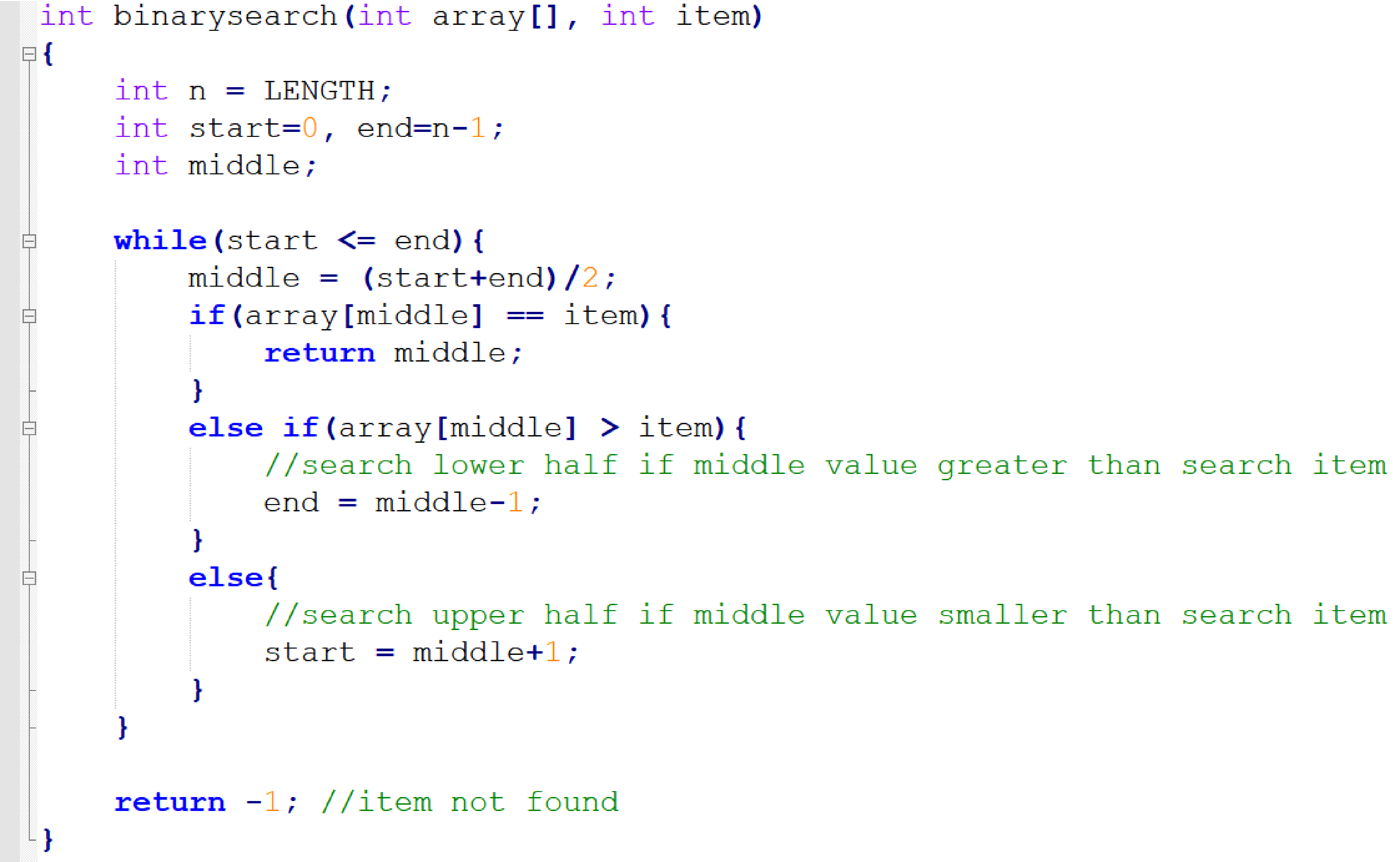
```
#define SIZE 8
int binary_search(int array[], int item)
{
















}
```



```c
#include <stdio.h>
#define LENGTH 8

int linearsearch(int array[], int item);
int binarysearch(int array[], int item);

int main()
{
    int array[LENGTH] = {2,3,5,6,8,9,10,13};

    //int idx = linearsearch(array, array[5]);
    int idx = binarysearch(array, array[5]);

    printf("item is found at index %d \n", idx);
    return 0;
}
//Simple Solution - not efficient:
int linearsearch(int array[], int item)
{
    int i;
    for(i=0;i<LENGTH;i++)
    {
        if(array[i] == item)
            return i;
    }

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

Exercise: implement a function that performs binary search

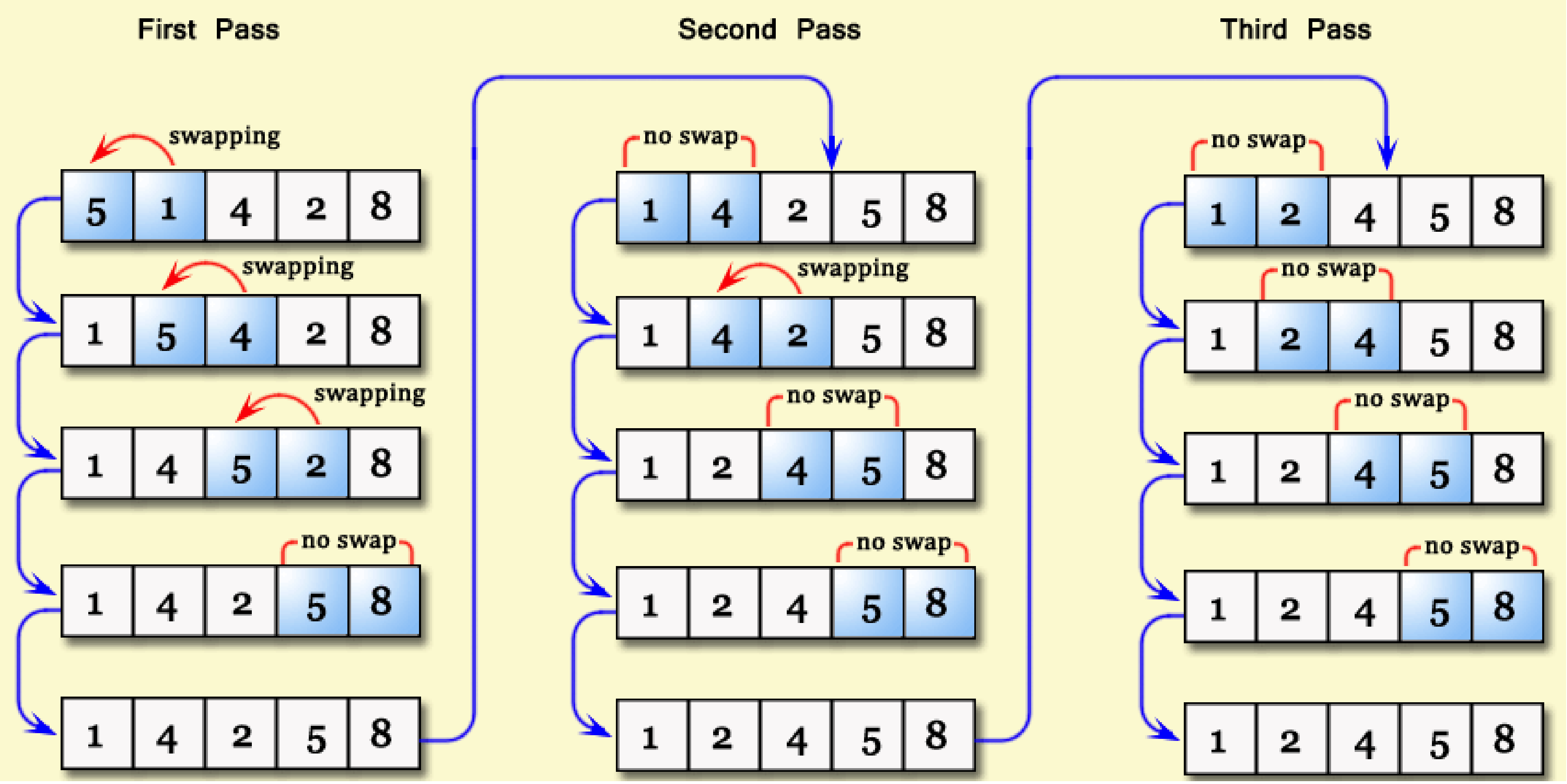
This function takes two arguments: a pointer to the sorted array and the search item. If the search item is found, the function returns its index in the array. Otherwise, it returns -1.



```
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 (bubble_sort)

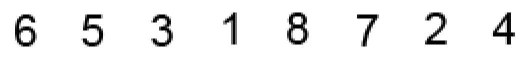
Bubble Sort: 1) compare items next to each other and swap them if needed;
2) repeat this process until the entire array is sorted.



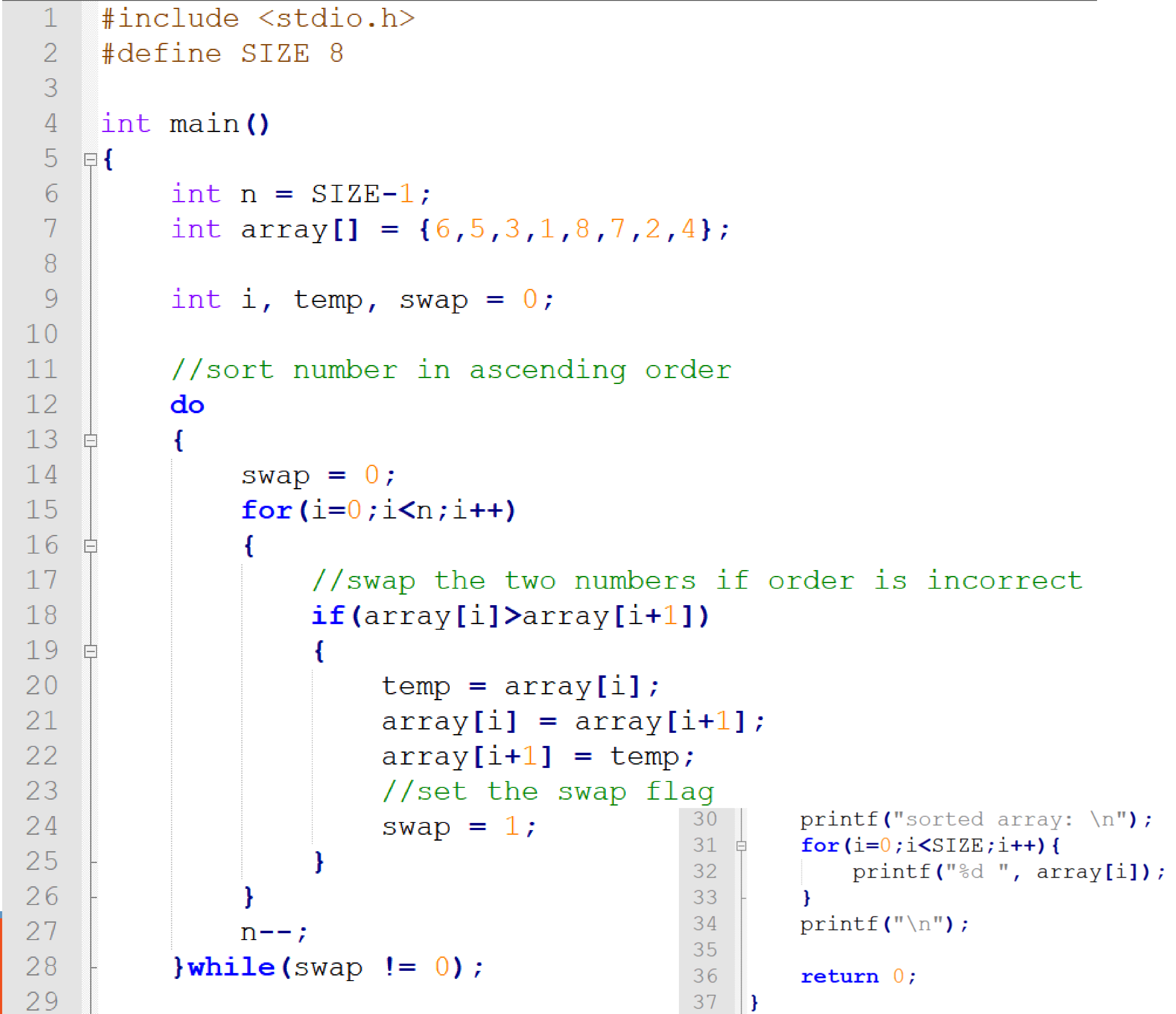
3. Sorting Algorithms (bubble_sort animation)

Bubble Sort: 1) compare items next to each other and swap them if needed;

2) repeat this process until the entire array is sorted.



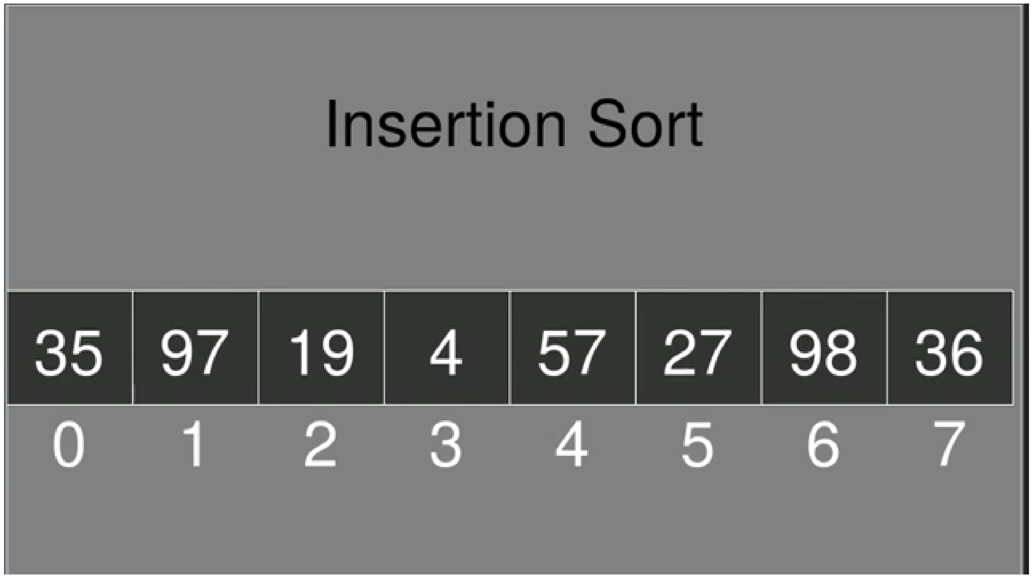
Exercise: implement a function that performs bubble sort



```c
#include <stdio.h>
#define SIZE 8

int main()
{
    int n = SIZE-1;
    int array[] = {6,5,3,1,8,7,2,4};

    int i, temp, swap = 0;

    //sort number in ascending order
    do
    {
        swap = 0;
        for(i=0;i<n;i++)
        {
            //swap the two numbers if order is incorrect
            if(array[i]>array[i+1])
            {
                temp = array[i];
                array[i] = array[i+1];
                array[i+1] = temp;
                //set the swap flag
                swap = 1;
            }
        }
        n--;
    }while(swap != 0);

    printf("sorted array: \n");
    for(i=0;i<SIZE;i++){
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

4. Sorting Algorithms (Insertion Sort animation)

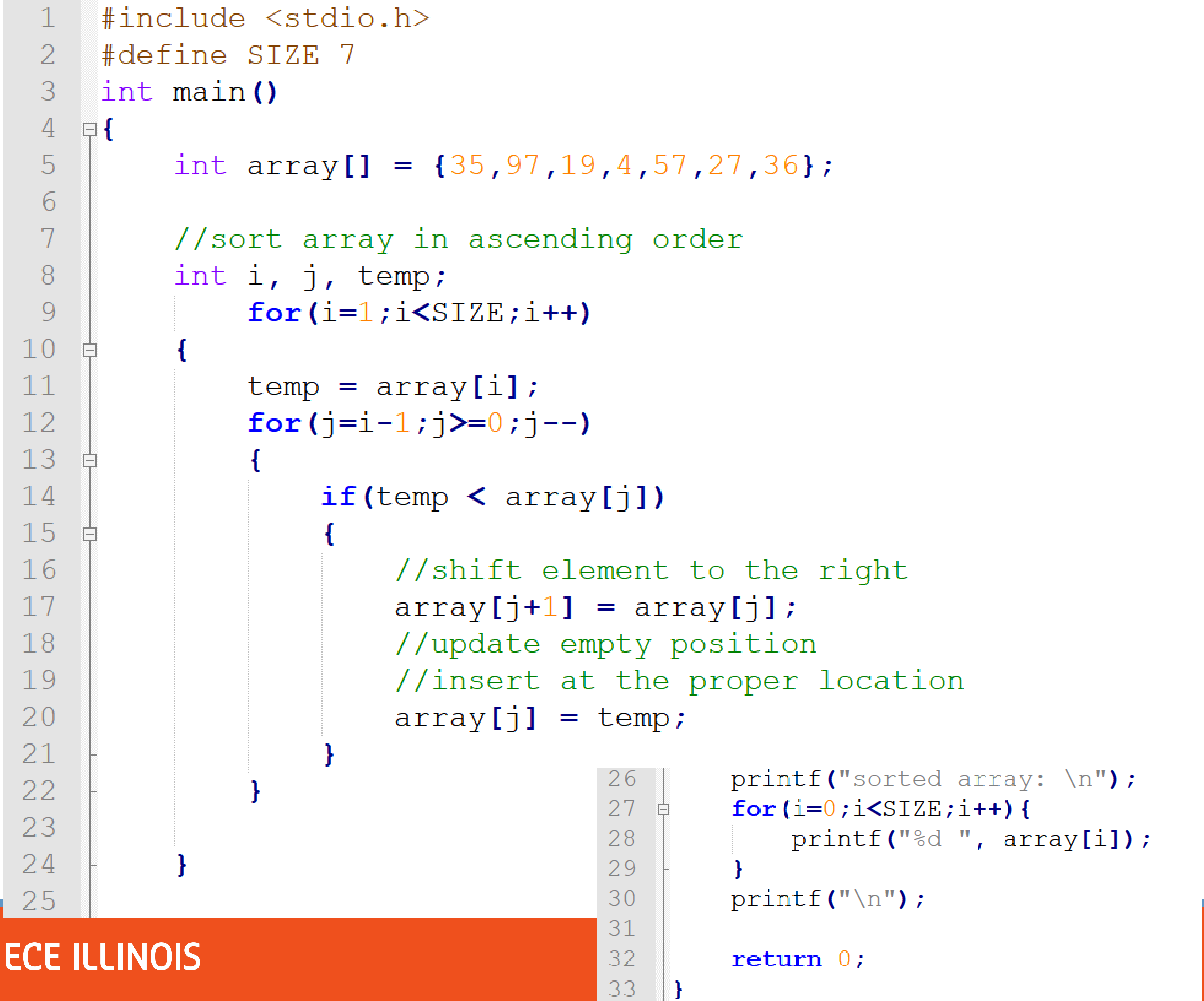
Bubble Sort: 1) compare pairs to the left and swap them if needed;
2) repeat this process until the entire array is sorted.



Insertion Sort

35	97	19	4	57	27	98	36
0	1	2	3	4	5	6	7

Exercise: implement a function that performs insertion sort



```c
#include <stdio.h>
#define SIZE 7
int main()
{
    int array[] = {35,97,19,4,57,27,36};

    //sort array in ascending order
    int i, j, temp;
        for(i=1;i<SIZE;i++)
    {
        temp = array[i];
        for(j=i-1;j>=0;j--)
        {
            if(temp < array[j])
            {
                //shift element to the right
                array[j+1] = array[j];
                //update empty position
                //insert at the proper location
                array[j] = temp;
            }
        }

    }

    printf("sorted array: \n");
    for(i=0;i<SIZE;i++){
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

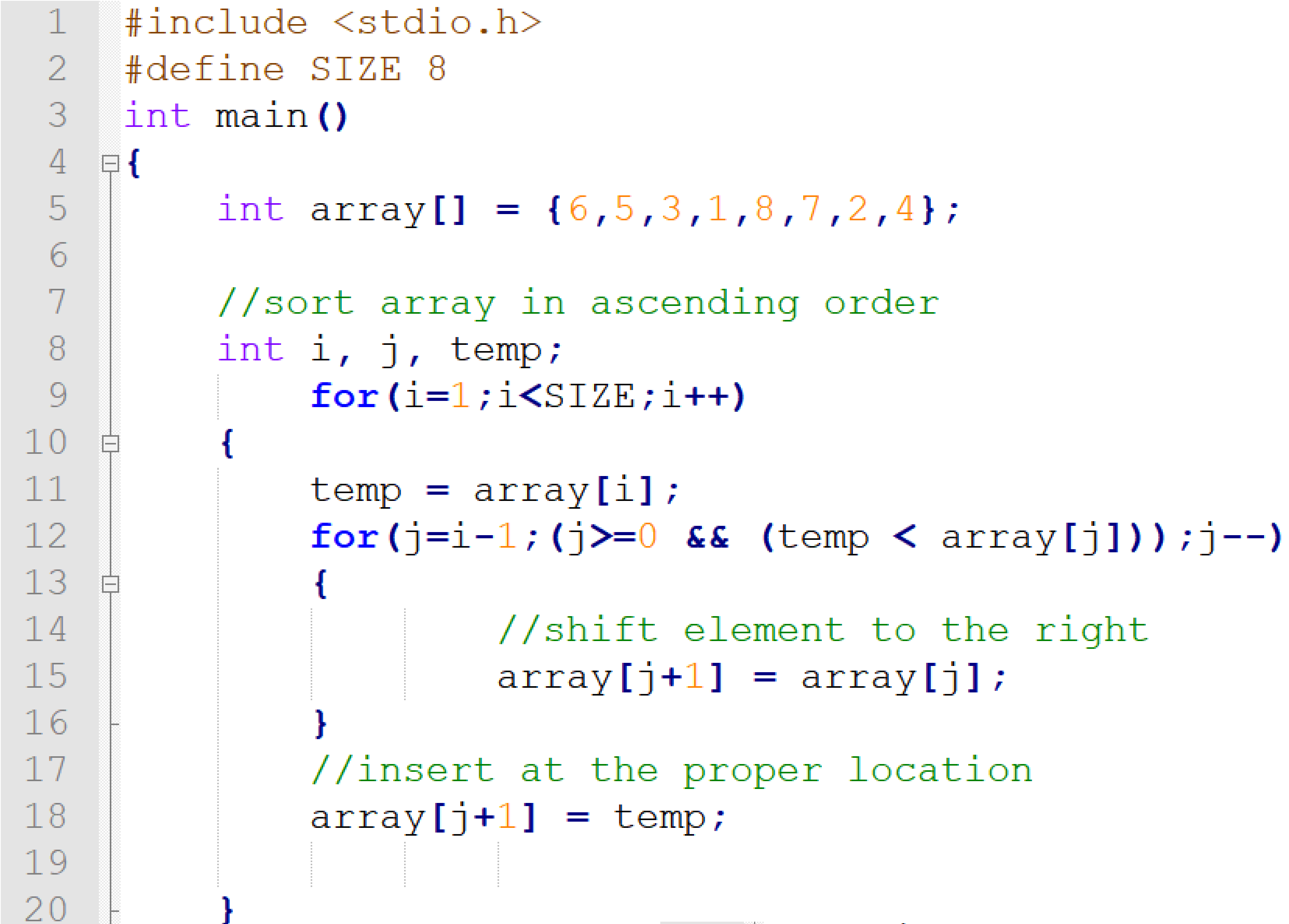
Merge_Insertion Sort:

1) remove item from array, insert it at the proper location in the sorted part by shifting other items;

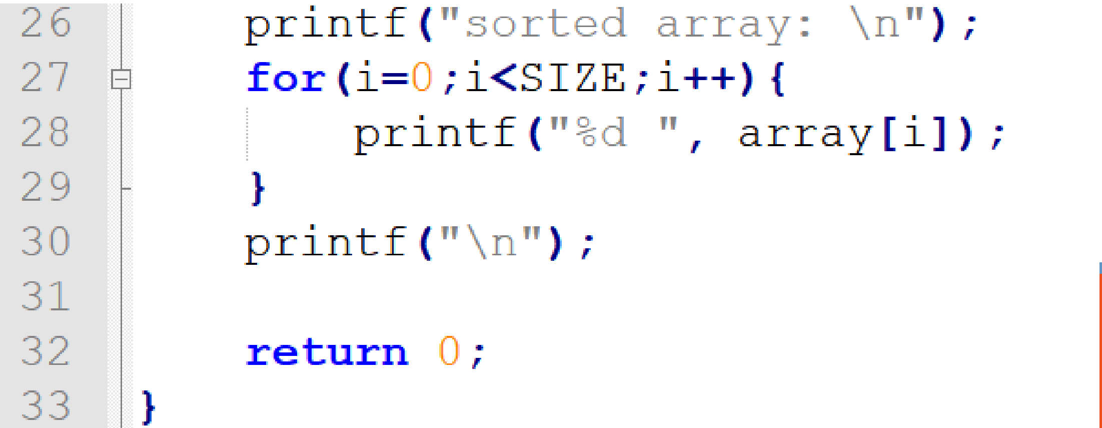
2) repeat this process until the end of array is reach.

6 5 3 1 8 7 2 4

Exercise: implement a function that performs insertion sort



```c
#include <stdio.h>
#define SIZE 8
int main()
{
    int array[] = {6,5,3,1,8,7,2,4};

    //sort array in ascending order
    int i, j, temp;
        for(i=1;i<SIZE;i++)
    {
        temp = array[i];
        for(j=i-1;(j>=0 && (temp < array[j]));j--)
        {
                //shift element to the right
                array[j+1] = array[j];
        }
        //insert at the proper location
        array[j+1] = temp;

    }
```



```c
    printf("sorted array: \n");
    for(i=0;i<SIZE;i++){
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```