COP 3223: C Programming
Spring 2009

Pointers In C – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

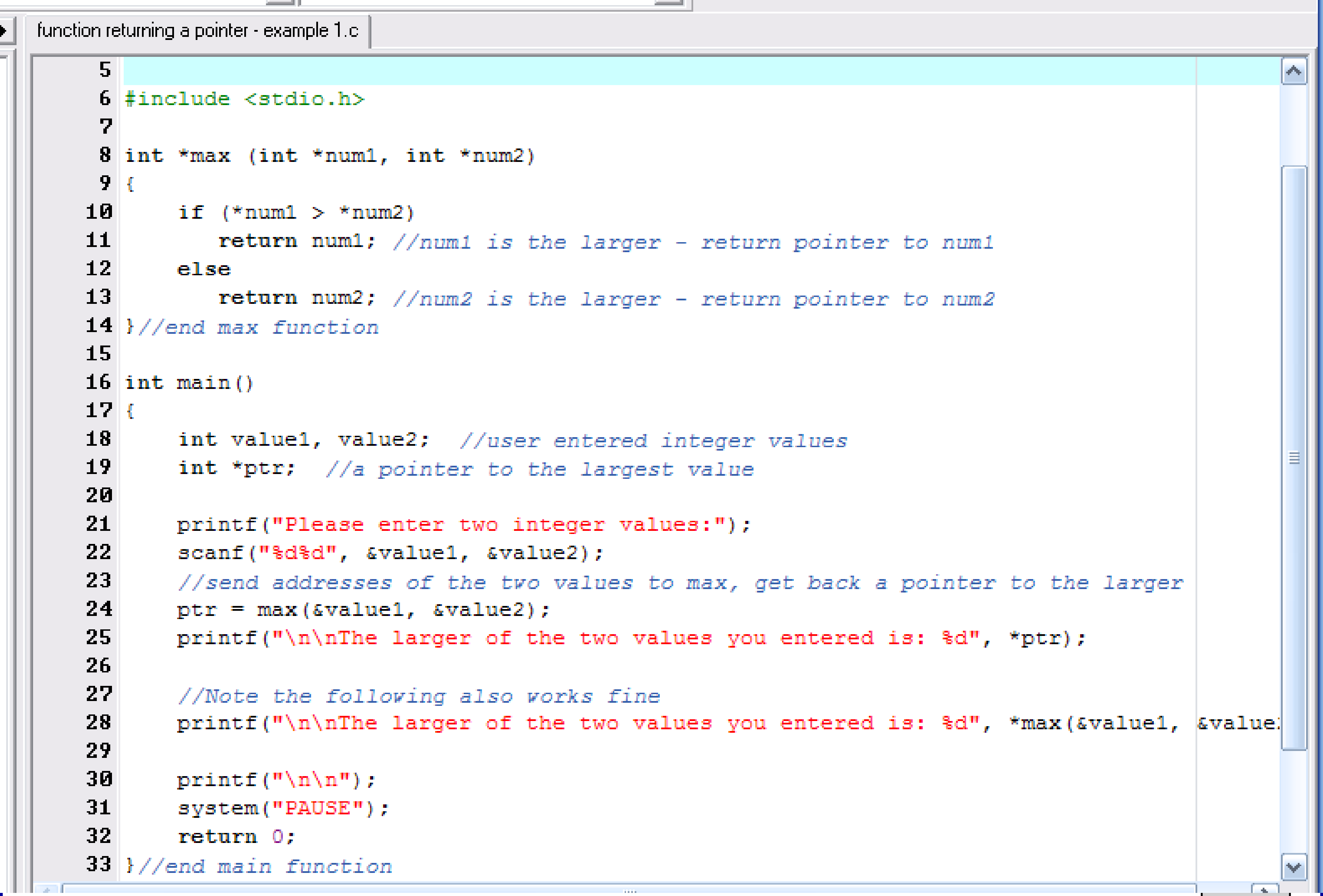
School of Electrical Engineering and Computer Science
University of Central Florida

Functions That Return Pointers

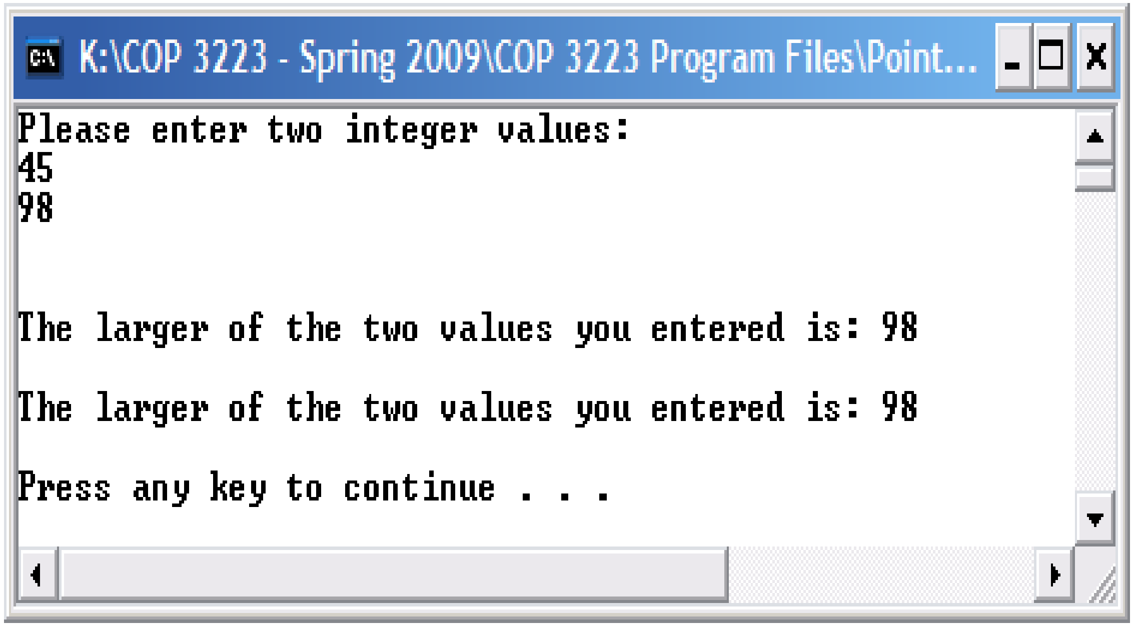
- Recall that when passing a pointer to a function, the passing mechanism is known as pass by reference. When passing a scalar value to a function the passing mechanism is known as pass by value.
- Since the function receives a copy of the caller's value of the parameter in pass by value, it is not possible for the function to modify the value of the caller's variable. This is because the function is working with its own local copy of the caller's variable's value and has no access to the caller's variable.
- In pass by reference, a pointer to the caller's variable is passed to the function which provides the function access to the variable through the same location as it is accessed by the caller. Thus, modification by the function modifies the same variable as is accessed by the caller.

Functions That Return Pointers

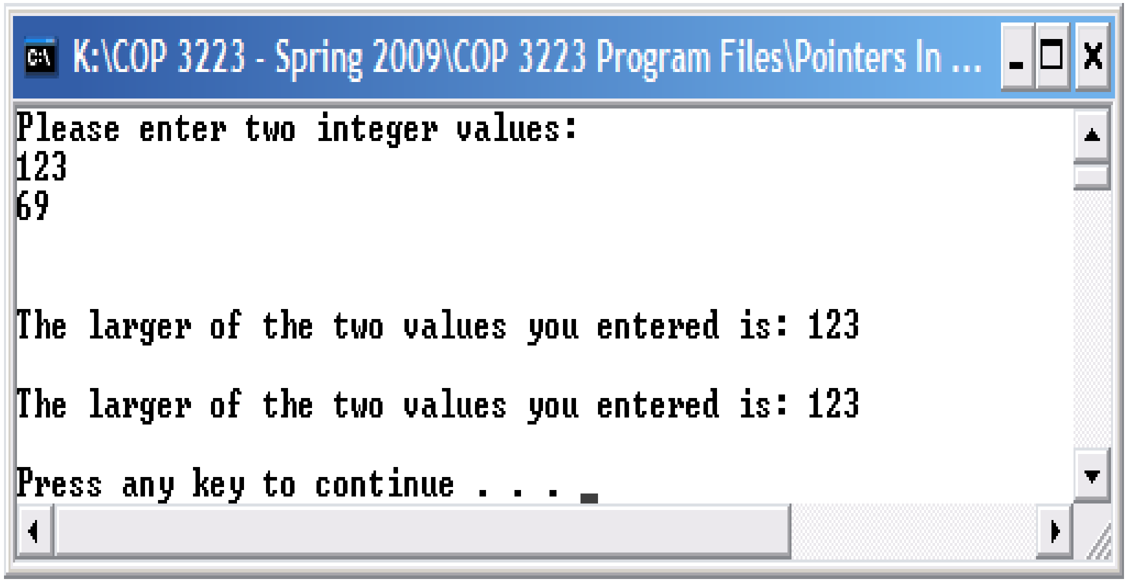
- It is also possible for a function in C to return a pointer value as the return type of the function.
- This is a very common technique utilized with functions that deal with arrays, where the function will return a pointer to a specific location (index) in the array.
- Let's look at a fairly simple function first (before adding the complexity of arrays to the mix), that simply returns a pointer to the larger of two integer values sent to it.



```c

#include <stdio.h>

int *max (int *num1, int *num2)
{
    if (*num1 > *num2)
       return num1; //num1 is the larger - return pointer to num1
    else
       return num2; //num2 is the larger - return pointer to num2
}//end max function

int main()
{
    int value1, value2;  //user entered integer values
    int *ptr;  //a pointer to the largest value

    printf("Please enter two integer values:");
    scanf("%d%d", &value1, &value2);
    //send addresses of the two values to max, get back a pointer to the larger
    ptr = max(&value1, &value2);
    printf("\n\nThe larger of the two values you entered is: %d", *ptr);

    //Note the following also works fine
    printf("\n\nThe larger of the two values you entered is: %d", *max(&value1, &value

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```



```
Please enter two integer values:
45
98


The larger of the two values you entered is: 98

The larger of the two values you entered is: 98

Press any key to continue . . .
```

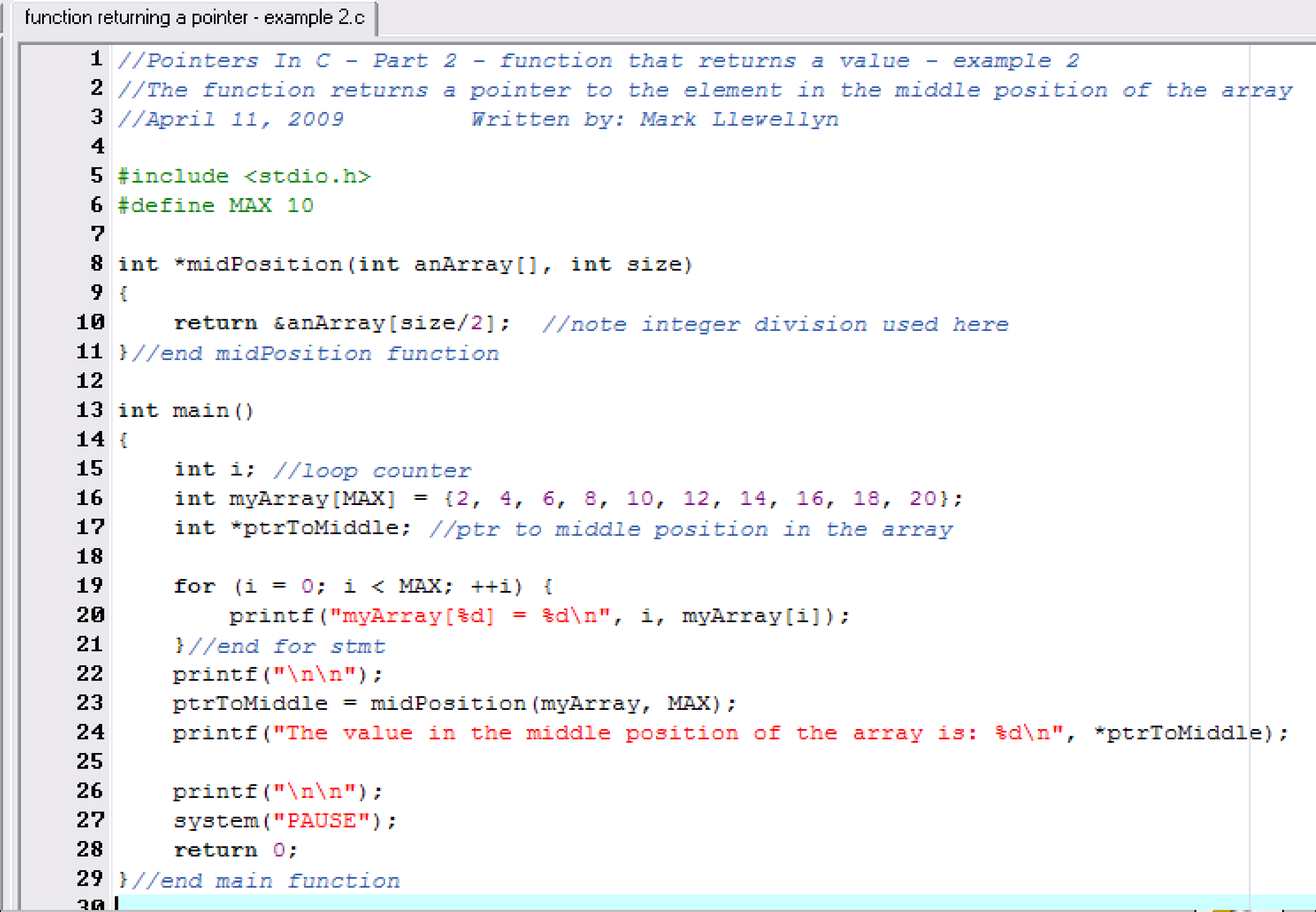


K:\COP 3223 - Spring 2009\COP 3223 Program Files\Pointers In ...

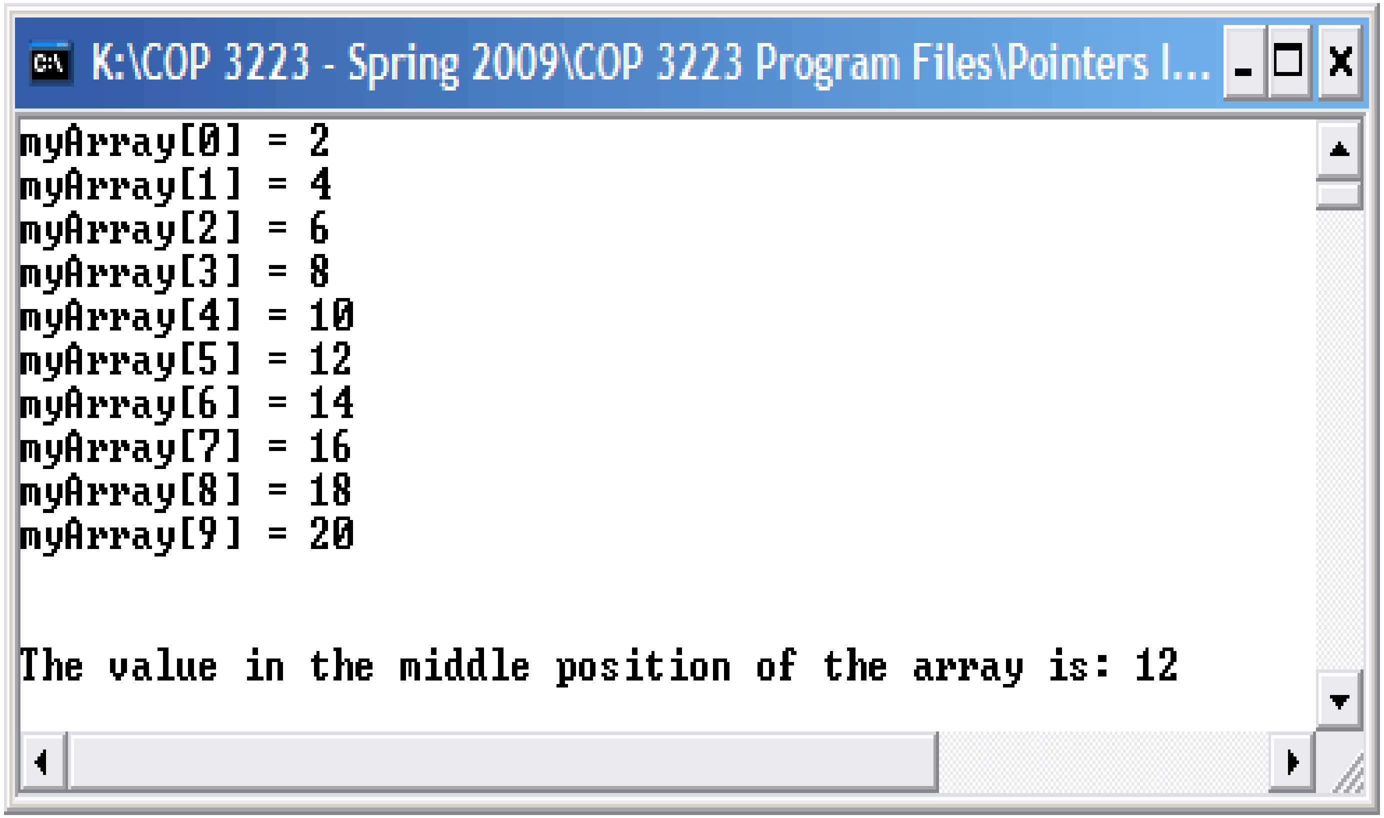
```
Please enter two integer values:
123
69


The larger of the two values you entered is: 123

The larger of the two values you entered is: 123

Press any key to continue . . . _
```

Functions That Return Pointers

- Now let’s look at a function that returns a pointer which is addressing a location in an array of integer values.
- We’ll keep this first function fairly simple in that all it will do is return a pointer to the middle element of an array. For example, if the array has 10 elements, our function will return a pointer to the element with an index of 5. (If the array had 11 elements we’d also expect the middle index to be 5 since we’ll use integer division to find the middle location.)



```c
//Pointers In C - Part 2 - function that returns a value - example 2
//The function returns a pointer to the element in the middle position of the array
//April 11, 2009          Written by: Mark Llewellyn

#include <stdio.h>
#define MAX 10

int *midPosition(int anArray[], int size)
{
    return &anArray[size/2];  //note integer division used here
}//end midPosition function

int main()
{
    int i; //loop counter
    int myArray[MAX] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
    int *ptrToMiddle; //ptr to middle position in the array

    for (i = 0; i < MAX; ++i) {
        printf("myArray[%d] = %d\n", i, myArray[i]);
    }//end for stmt
    printf("\n\n");
    ptrToMiddle = midPosition(myArray, MAX);
    printf("The value in the middle position of the array is: %d\n", *ptrToMiddle);

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```



K:\COP 3223 - Spring 2009\COP 3223 Program Files\Pointers I...

```
myArray[0] = 2
myArray[1] = 4
myArray[2] = 6
myArray[3] = 8
myArray[4] = 10
myArray[5] = 12
myArray[6] = 14
myArray[7] = 16
myArray[8] = 18
myArray[9] = 20


The value in the middle position of the array is: 12
```

Pointer Arithmetic In C

- As the previous example illustrates, pointers can be used to point to any element of an array. While C uses an implicit pointer to reference the first element of every array, so to can any element be referenced with a pointer.

- In fact, it is very common to perform many array operations using only pointers and pointer arithmetic.

- Pointer arithmetic is used to modify the value of a pointer and allowing us to “walk around” in the array using the pointer value.

Pointer Arithmetic In C

- C supports three forms of pointer arithmetic (also referred to as address arithmetic).
- The three forms of pointer arithmetic are:
  - Adding an integer value to a pointer.
  - Subtracting an integer value from a pointer.
  - Subtracting one pointer value from another pointer value.
- We'll look at each form of pointer arithmetic and illustrate their usage as well as potential issues with each form.

Pointer Arithmetic – Adding An Integer To A Pointer

- Let’s assume the following declarations have been made:

  ```
  int anArray[10], *ptr1, *ptr2, i, j;
  ```

- In general, adding an integer `x` to a pointer `p` causes the pointer to reference the array element `x` places after the one that `p` is initially referencing.

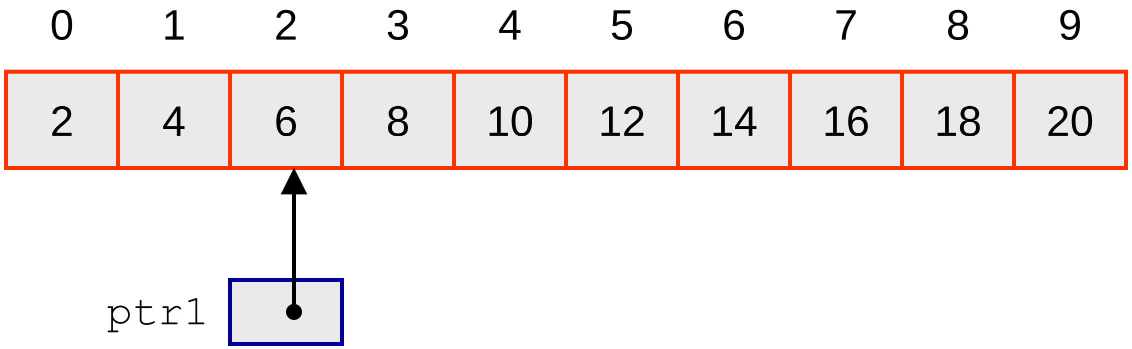
- More precisely, using the example declarations shown above, if `ptr1` points to `anArray[i]`, then `ptr1+j` references `anArray[i+j]`.

- NOTE: It is a run-time error in the above expression if the array element `[i+j]` does not exist (an out of bounds error)!

Pointer Arithmetic – Adding An Integer To A Pointer

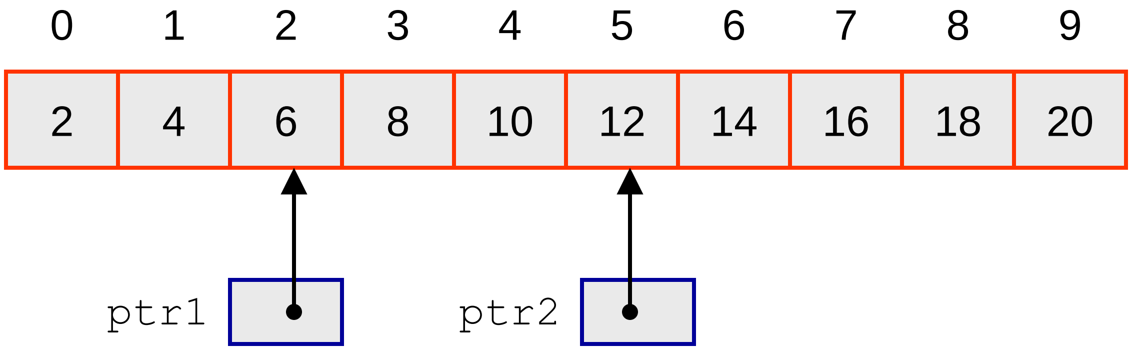
EXAMPLE

```
ptr1 = &anArray[2];
```



0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20

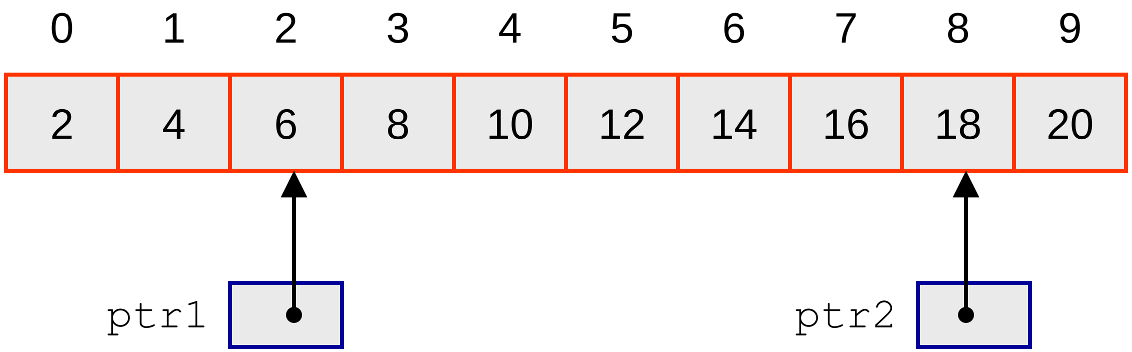
```
ptr2 = ptr1 + 3;
```



0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20

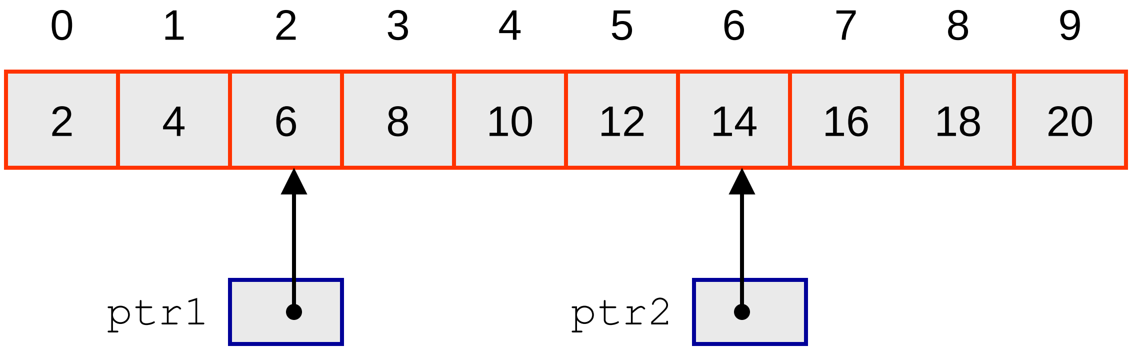
Pointer Arithmetic – Adding An Integer To A Pointer EXAMPLE – continues from previous page

```
ptr2 += 3;
```



0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20

```
int i = -2;
ptr2 += i;
```



0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20

Pointer Arithmetic – Subtracting An Integer From A Pointer

- Let’s assume the following declarations have been made:

```
int anArray[10], *ptr1, *ptr2, i, j;
```

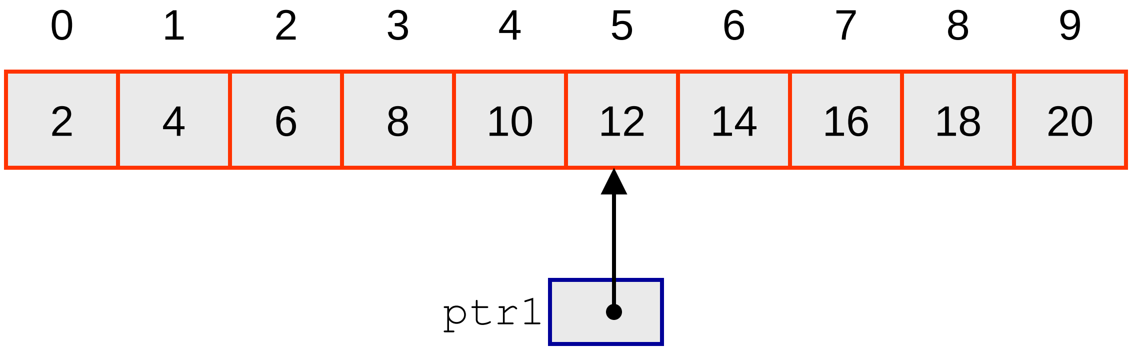
- In general, subtracting an integer `x` from a pointer `p` causes the pointer to reference the array element `x` places before the one that `p` is initially referencing.

- More precisely, using the example declarations shown above, if `ptr1` points to `anArray[i]`, then `ptr1-j` references `anArray[i-j]`.

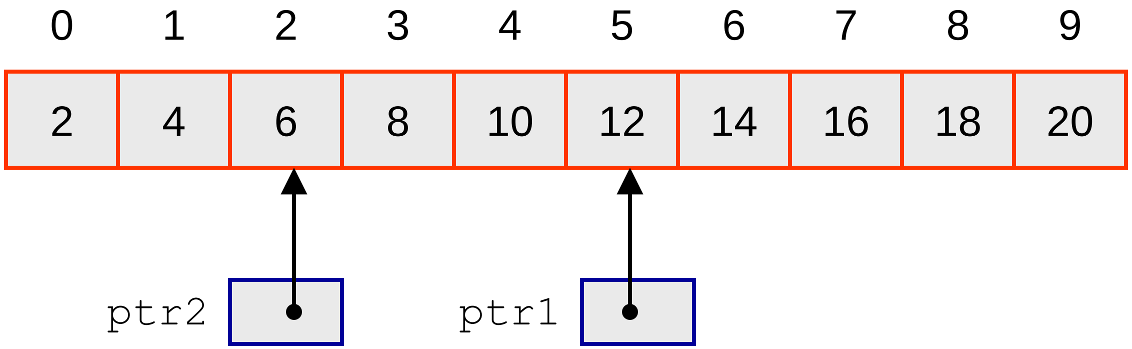
- NOTE: It is a run-time error in the above expression if the array element `[i-j]` does not exist (an out of bounds error)!

Pointer Arithmetic – Subtracting An Integer From A Pointer – EXAMPLE

```
ptr1 = &anArray[5];
```

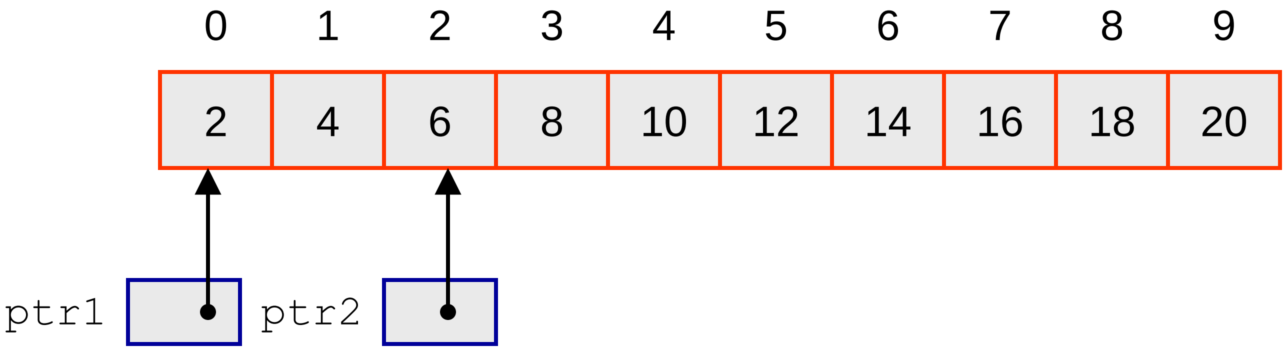


```
ptr2 = ptr1 - 3;
```



Pointer Arithmetic – Subtracting An Integer From A Pointer – EXAMPLE – continues from previous page

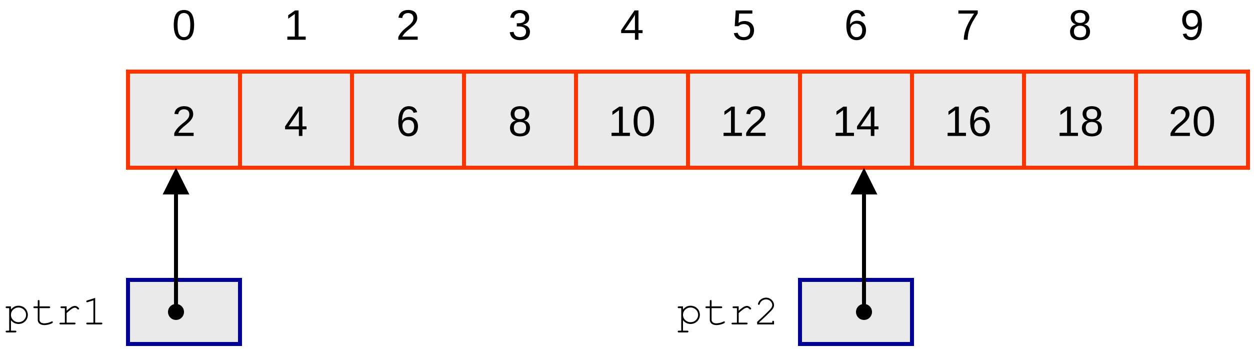
```
ptr1 -= 5;
```



0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20

ptr1 ptr2

```
int i = -4;
ptr2 -= i;
```



0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20

ptr1 ptr2

Pointer Arithmetic – Subtracting One Pointer From Another

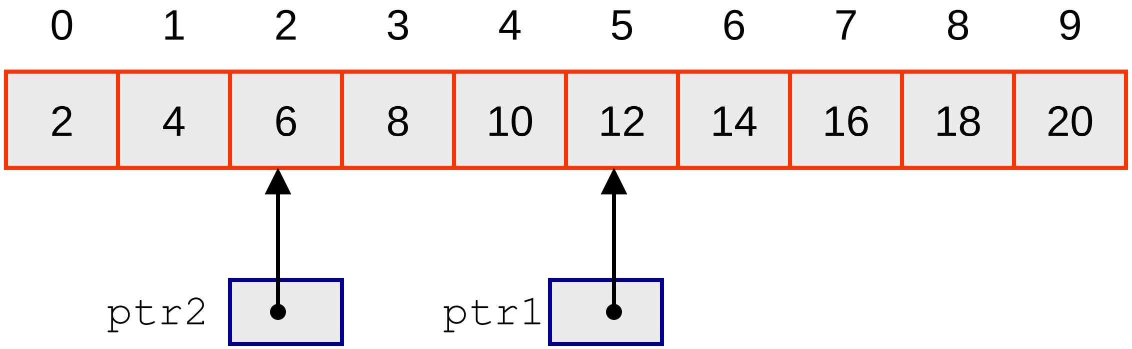
- Let’s assume the following declarations have been made:

  ```
  int anArray[10], *ptr1, *ptr2, i, j;
  ```

- In general, subtracting one pointer from another, the result is the distance (measured in array elements) between the pointers.
- Thus, if `ptr1` points to element `anArray[i]` and `ptr2` points to `anArray[j]`, then `ptr1 – ptr2` is equal to `i-j`.
- NOTE: It is a run-time error in the above expression if the array element `[i-j]` does not exist (an out of bounds error)!
- NOTE: Subtracting one pointer from another is undefined unless both pointers reference the same array!

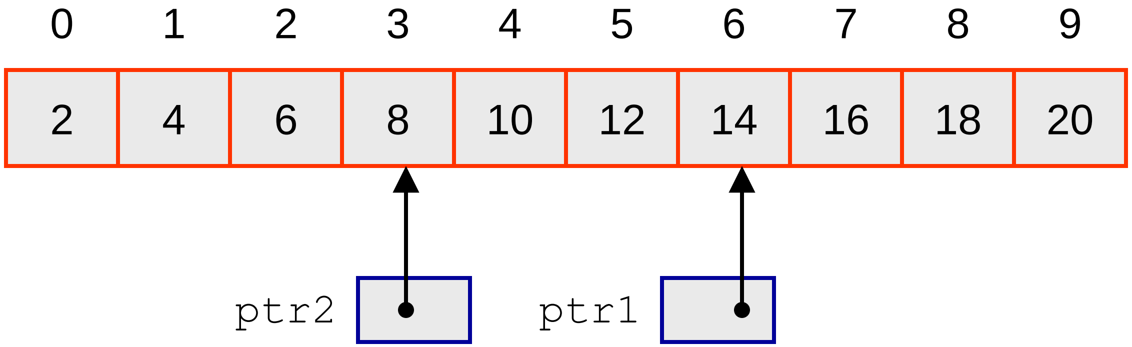
Pointer Arithmetic – Subtracting An Integer From A Pointer – EXAMPLE

```
ptr1 = &anArray[5];
ptr2 = &anArray[2];
i = ptr2 - ptr1; //i = 3
i = ptr1 - ptr2; //i = -3
```



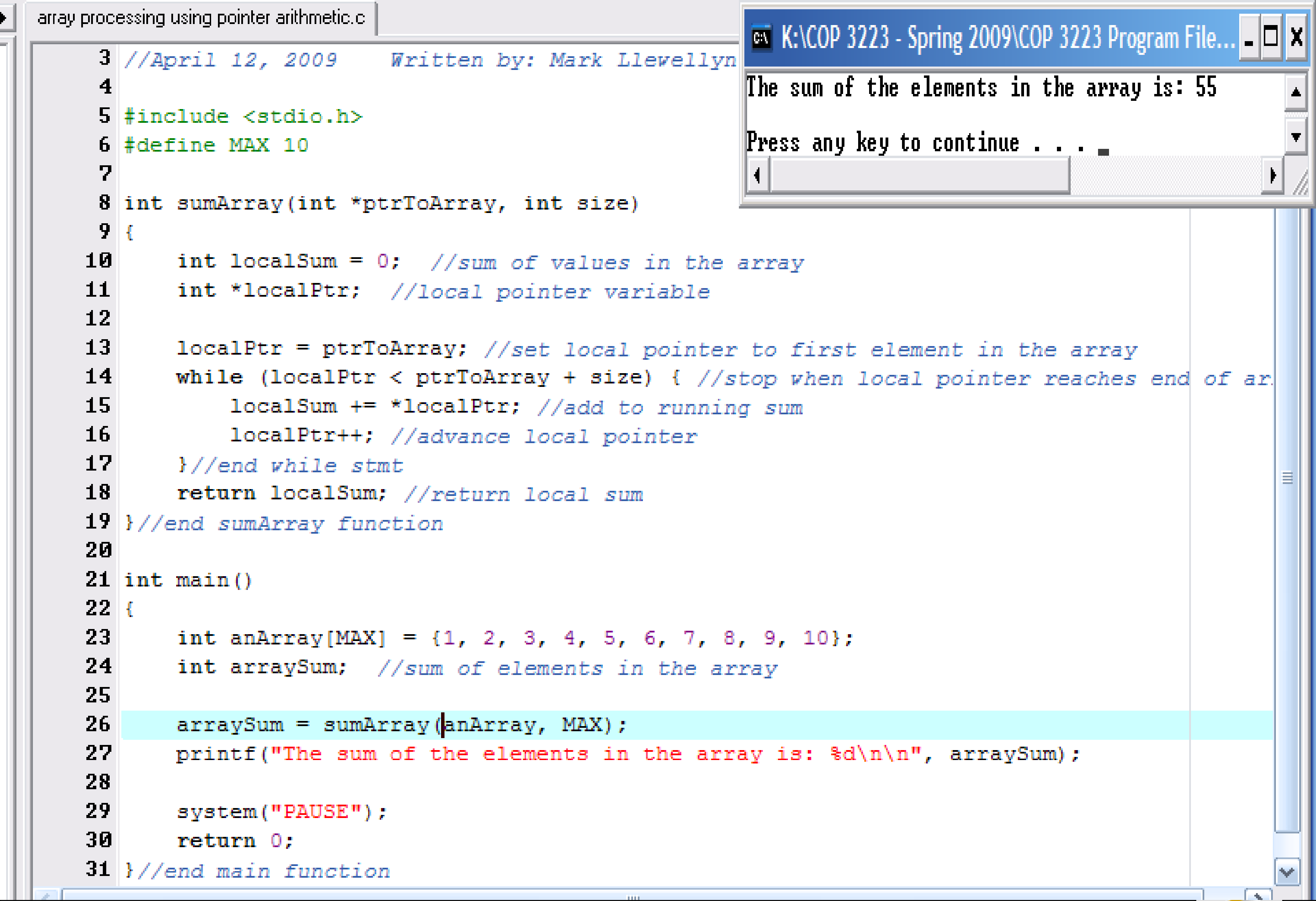
out of bounds error

```
ptr1 = &anArray[6];
ptr2 = &anArray[3];
i = ptr2 - ptr1; //i = -3
i = ptr1 - ptr2; //i = 3
```



Using Pointers For Array Processing

- Pointer arithmetic is often used for moving about in an array.
- The following example illustrates how pointer arithmetic is used to "walk through" the elements of an array producing the sum of the elements in the array.
- To keep things fairly simple, we'll assume a small array of 10 elements and use an initializer to place the values in the array. The practice problem at the end of these notes will have you modify this example to use an array of user-determined length to accomplish the same task.

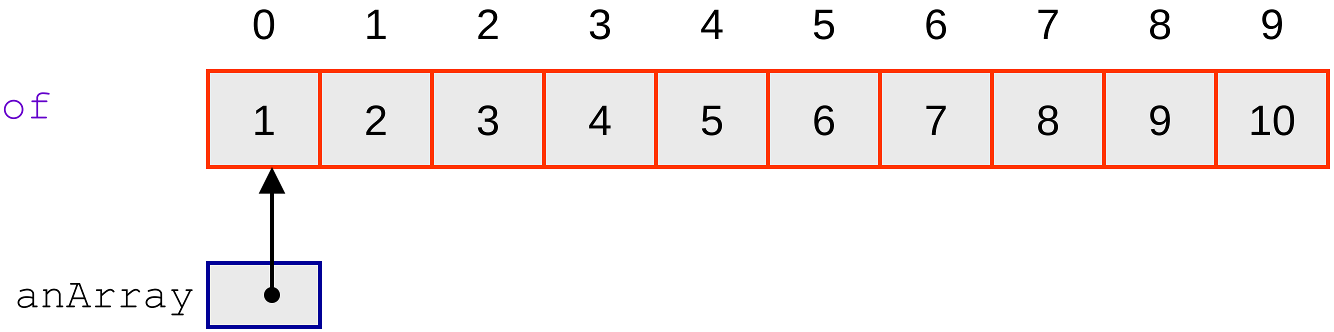

array processing using pointer arithmetic.c
K:\COP 3223 - Spring 2009\COP 3223 Program File...

```c
//April 12, 2009    Written by: Mark Llewellyn

#include <stdio.h>
#define MAX 10

int sumArray(int *ptrToArray, int size)
{
    int localSum = 0;  //sum of values in the array
    int *localPtr;  //local pointer variable

    localPtr = ptrToArray; //set local pointer to first element in the array
    while (localPtr < ptrToArray + size) { //stop when local pointer reaches end of ar
        localSum += *localPtr; //add to running sum
        localPtr++; //advance local pointer
    }//end while stmt
    return localSum; //return local sum
}//end sumArray function

int main()
{
    int anArray[MAX] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int arraySum;  //sum of elements in the array

    arraySum = sumArray(anArray, MAX);
    printf("The sum of the elements in the array is: %d\n\n", arraySum);

    system("PAUSE");
    return 0;
}//end main function
```

```
The sum of the elements in the array is: 55

Press any key to continue . . . _
```

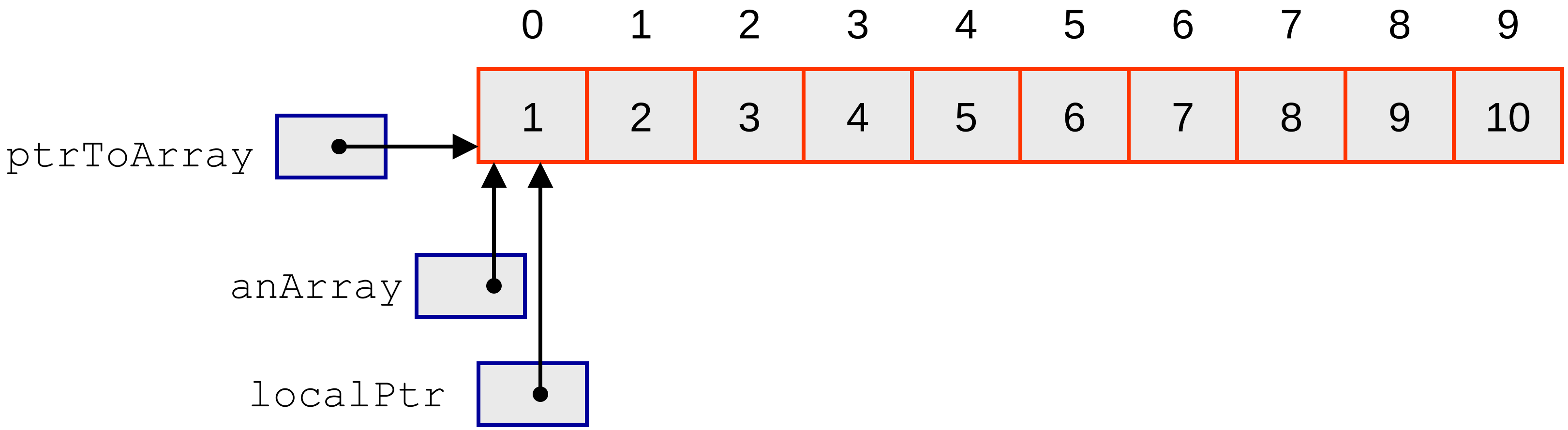
Trace of Array Example Using Pointer Arithmetic

In main() at time of call to sumArray



0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

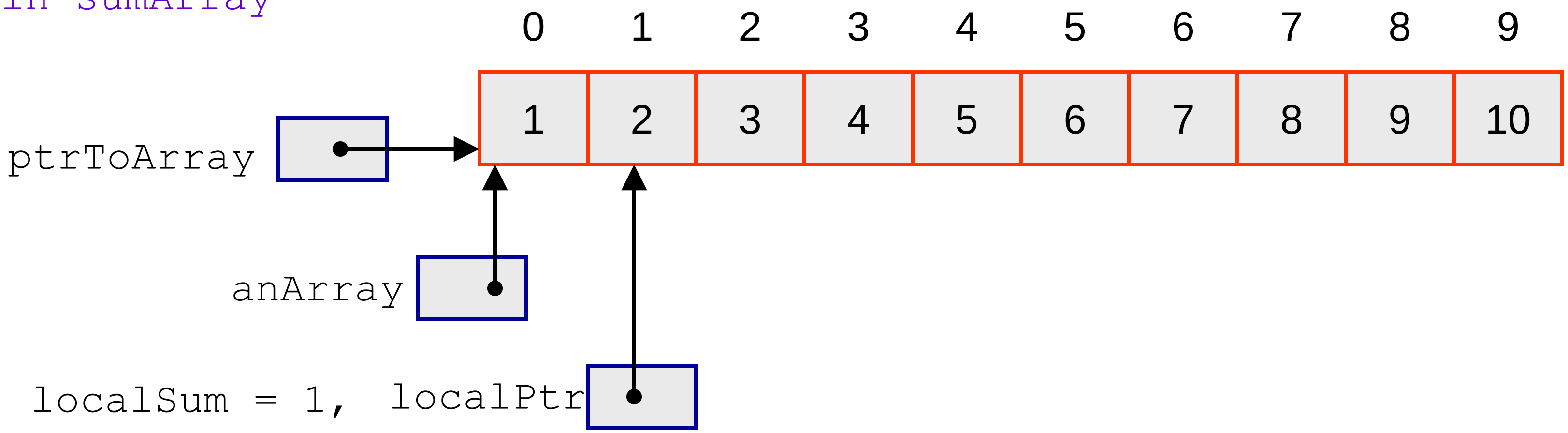
Line 13 in sumArray



0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

Trace of Array Example Using Pointer Arithmetic

Line 16 in sumArray



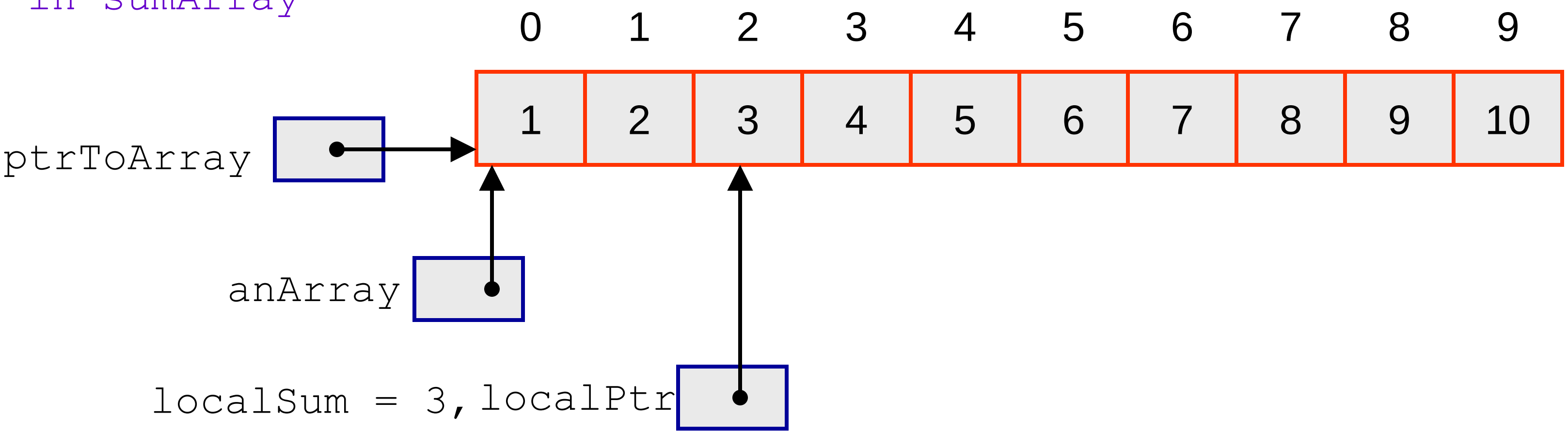
0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

ptrToArray

anArray

localSum = 1, localPtr

Line 16 in sumArray



0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

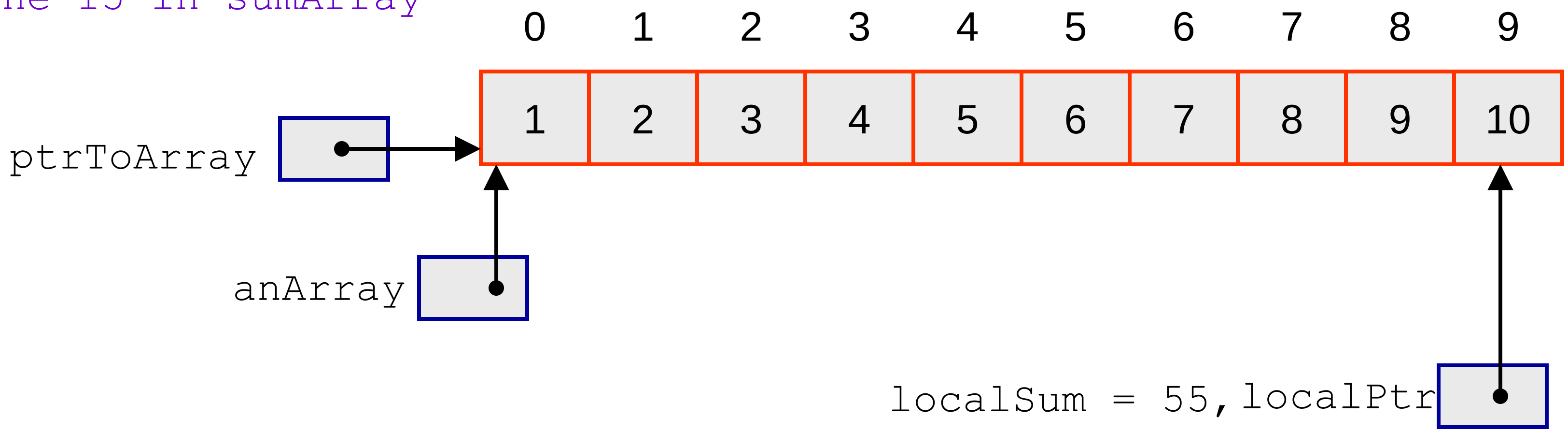
ptrToArray

anArray

localSum = 3, localPtr

Trace of Array Example Using Pointer Arithmetic

. . . Line 15 in sumArray



0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

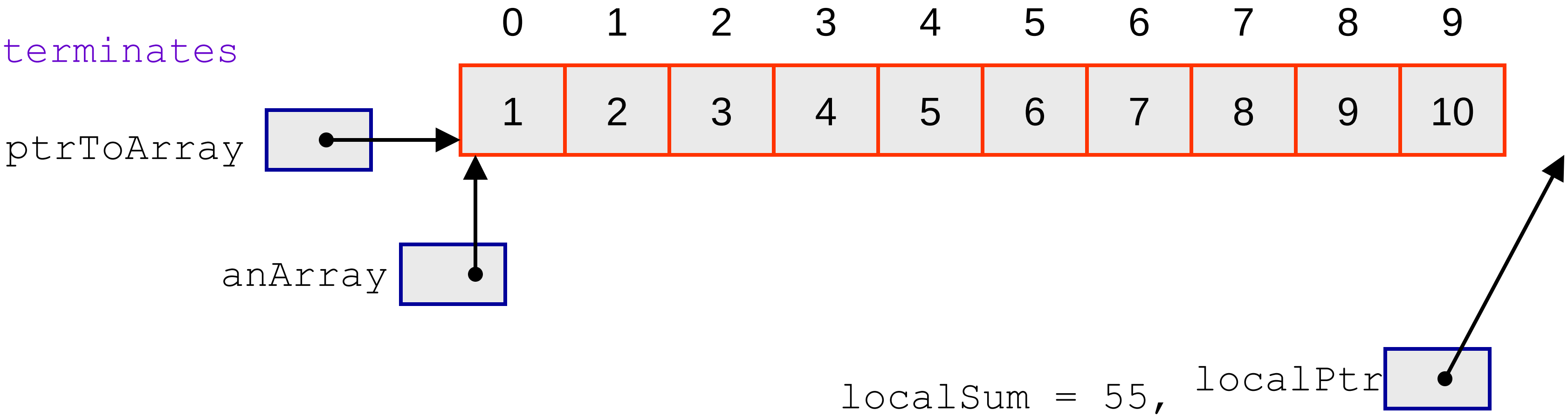
ptrToArray

anArray

localSum = 55, localPtr

Line 16 in sumArray

Loop terminates



0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

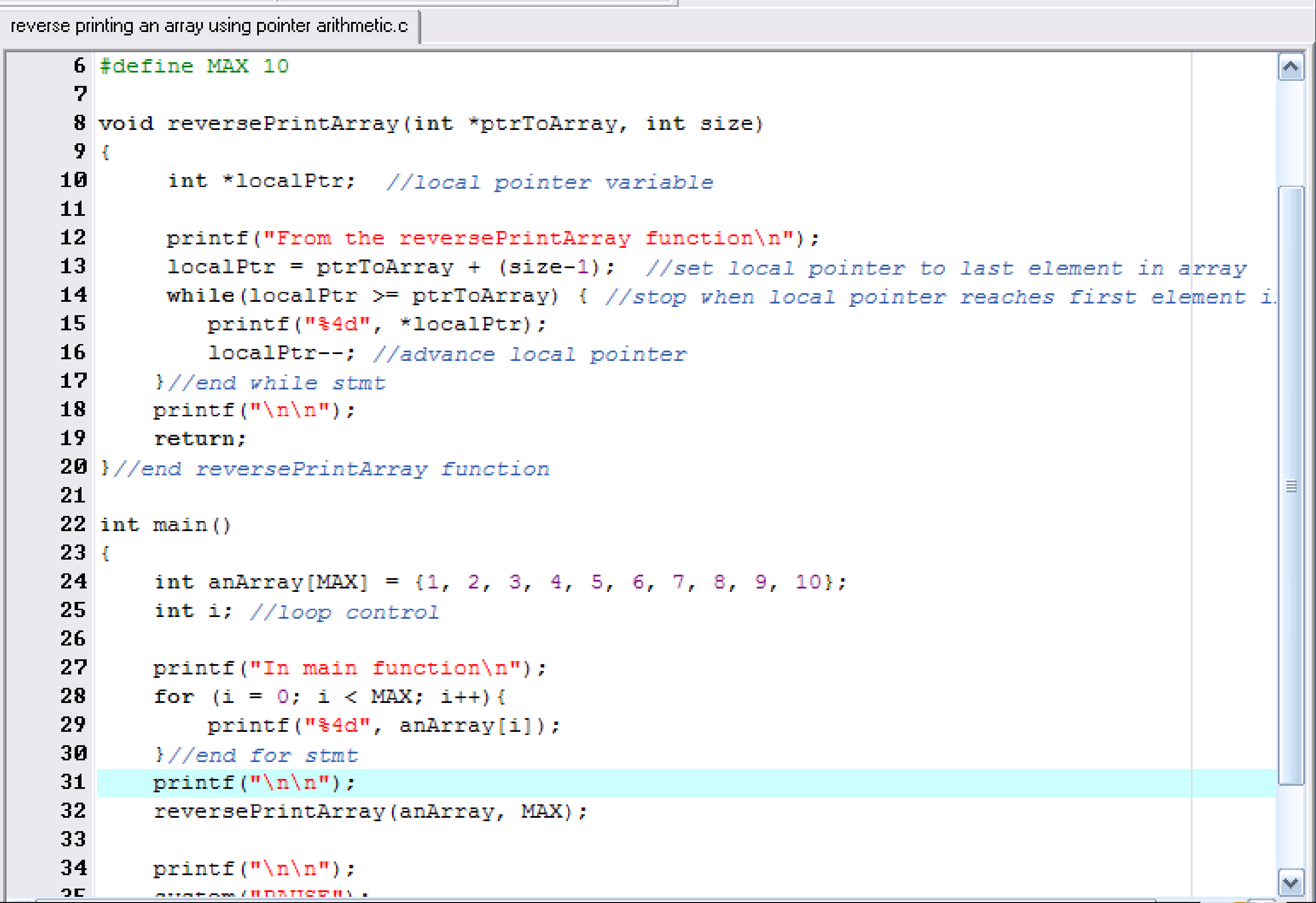
ptrToArray

anArray

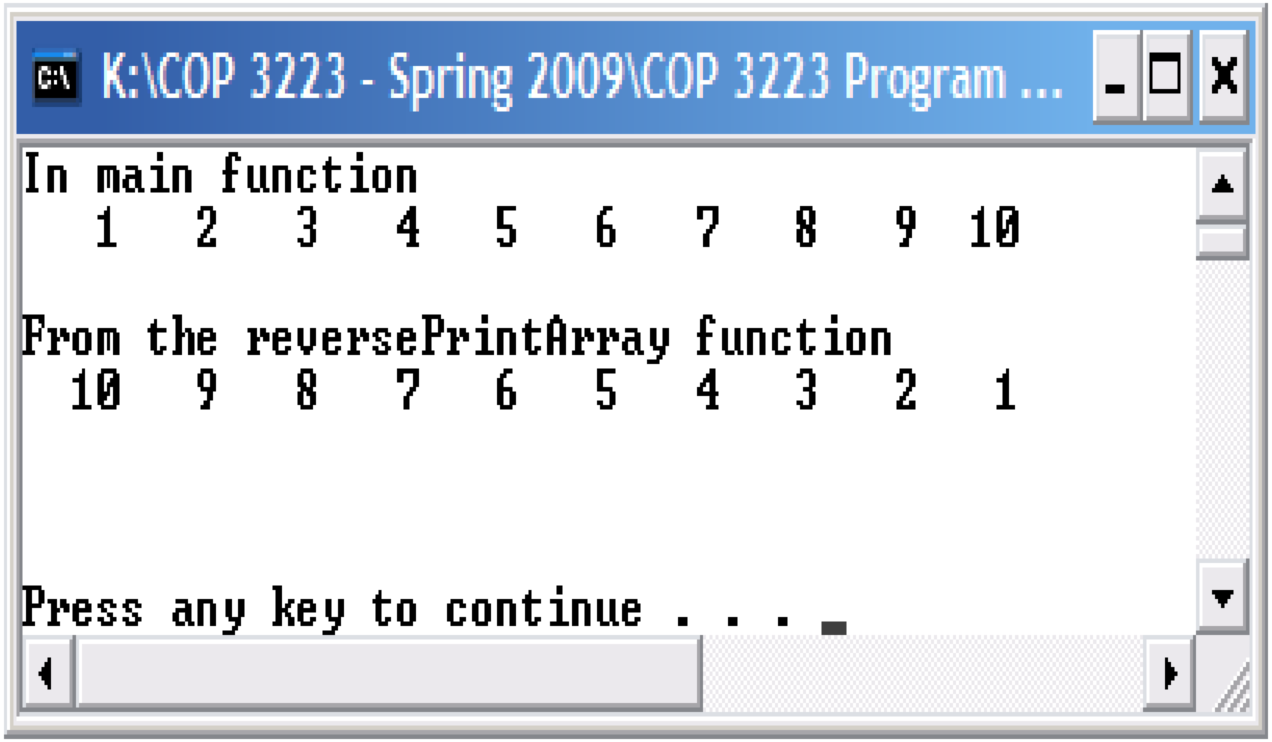
localSum = 55, localPtr

Using Pointers For Array Processing

- As a final example of using pointer arithmetic for array processing, the example on the next page prints the elements in an array in the reverse order in which they are stored.
- Notice that in this case we are subtracting an integer (1) from the value of a pointer, as well as using various pointer assignments.
- Once again, I just used an initializer for a fairly simple array of values, but you might want to modify the code so that the array values could be either read in by the user from the keyboard or even better from a file.



```
#define MAX 10

void reversePrintArray(int *ptrToArray, int size)
{
     int *localPtr;  //local pointer variable

     printf("From the reversePrintArray function\n");
     localPtr = ptrToArray + (size-1);  //set local pointer to last element in array
     while(localPtr >= ptrToArray) { //stop when local pointer reaches first element i
        printf("%4d", *localPtr);
        localPtr--; //advance local pointer
    }//end while stmt
    printf("\n\n");
    return;
}//end reversePrintArray function

int main()
{
    int anArray[MAX] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int i; //loop control

    printf("In main function\n");
    for (i = 0; i < MAX; i++){
        printf("%4d", anArray[i]);
    }//end for stmt
    printf("\n\n");
    reversePrintArray(anArray, MAX);

    printf("\n\n");
    system("PAUSE");
```



```
In main function
   1    2    3    4    5    6    7    8    9   10

From the reversePrintArray function
  10    9    8    7    6    5    4    3    2    1



Press any key to continue . . . _
```

Practice Problems

1. Modify the example on page 20 so that the program asks the user how many elements will be in the array and then reads that many values from the keyboard entered by the user.

2. Write a C program that uses a function that returns a pointer to the element of an array that contains the largest value in the array. Have the function accept an explicit pointer to the first element of the array that is passed to it.