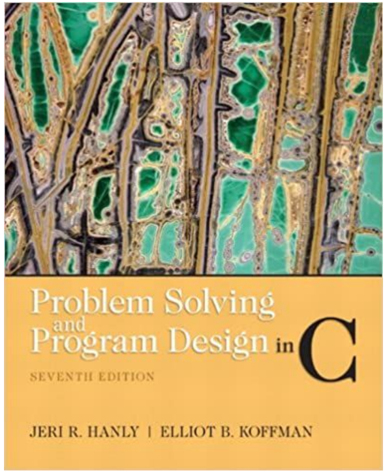
جامعة بيرزيت
BIRZEIT UNIVERSITY



Faculty of Engineering and Technology
Department of Computer Science

Introduction to Computers and Programming (Comp 133)

References :
Book : Problem Solving and Program Design in C (7th Edition) 7th Edition
Slides : Dr. Radi Jarrar , Dr. Abdallah Karakra , Dr. Majdi Mafarja.

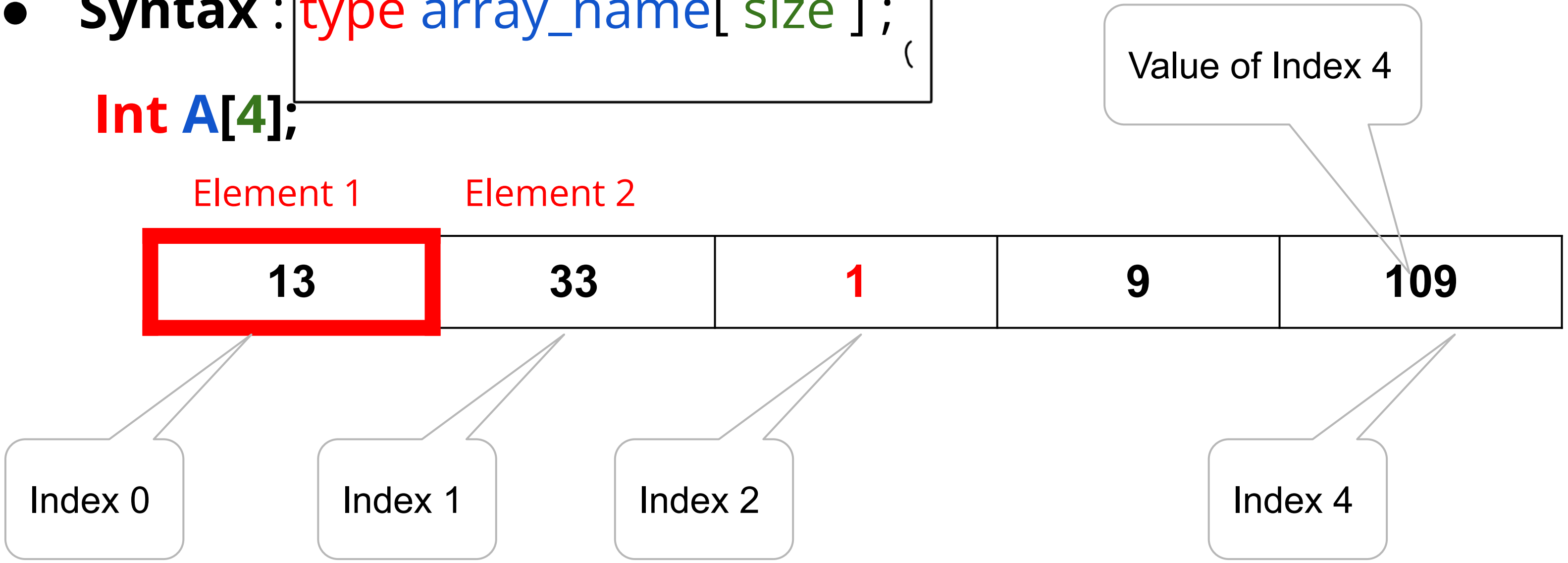
Arrays

Chapter 7

Arrays

- **Array** a collection of data **items \ variables** of the same type.
- **Array element** a data item that is part of an array.
- **Array index** to access specific element or variable in the array.
- **Syntax** : type array_name[size] ;

Int A[4];



A[0] : 13

A[1] : 33

A[2] : 1

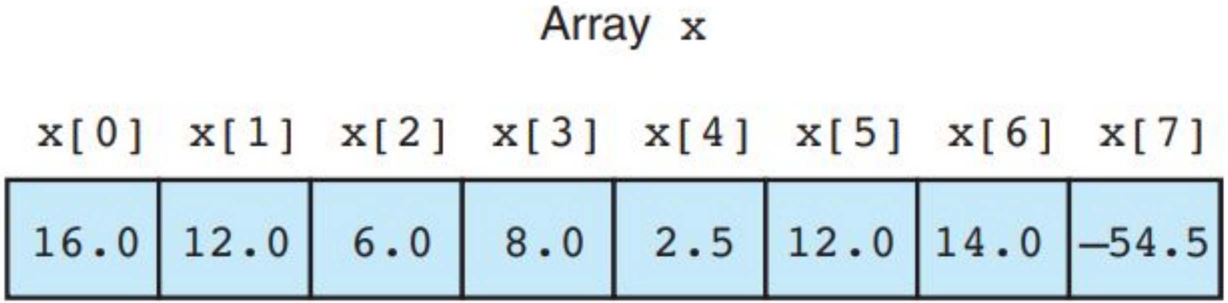
A[3] : 9

A[4] : 109

Declaring Arrays

- Type arrayName[numberOfElements];
  - int c[10];
  - float myArray[100];
- Declaring multiple arrays of same type
  - int b[100], x[8];

Hi How are u ?



Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Array Initialization

- int Arr1[3]={1,2,3};

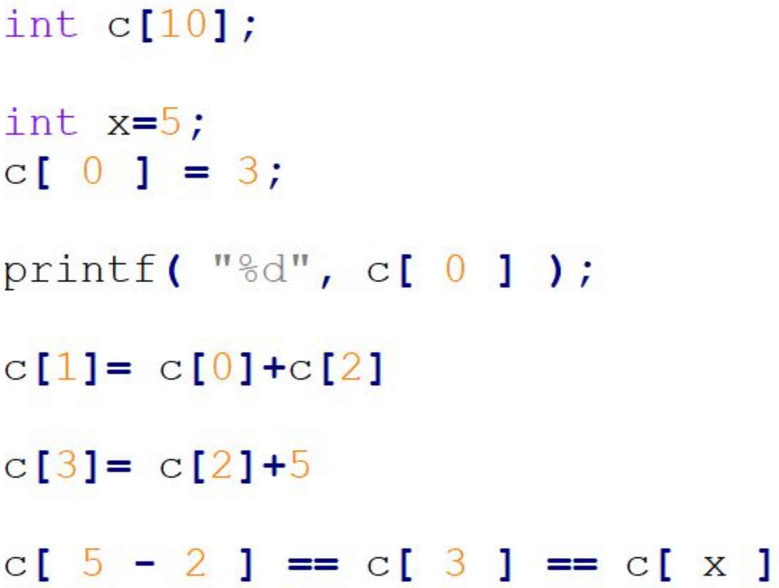
1	**2**	**3**
Arr1[0]	**Arr1[1]**	**Arr1[2]**

index

- double Arr2**[]**={1.4 ,2.55 , 3.0, 4.12};

1.4	**2.55**	**3.0**	**4.12**
Arr2[0]	**Arr2[1]**	**Arr2[2]**	**Arr2[3]**

Access Array elements

- char vowels[] = {'A', 'E', 'I', 'O', 'U'}; // the size depend on the values
  - printf("%c", vowels[0]); // print A
- 

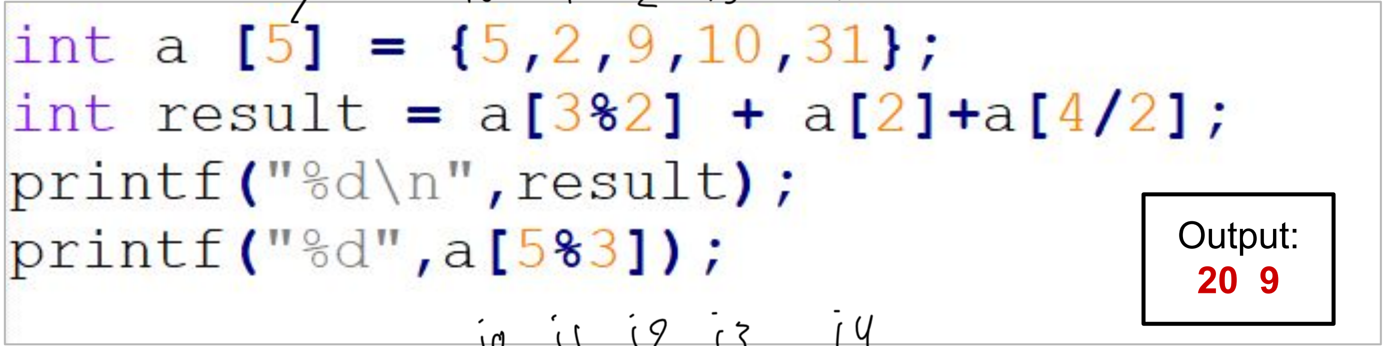
```
int c[10];

int x=5;
c[ 0 ] = 3;

printf( "%d", c[ 0 ] );

c[1]= c[0]+c[2]

c[3]= c[2]+5

c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

Array Example

num of index = (size of array - 1)

size

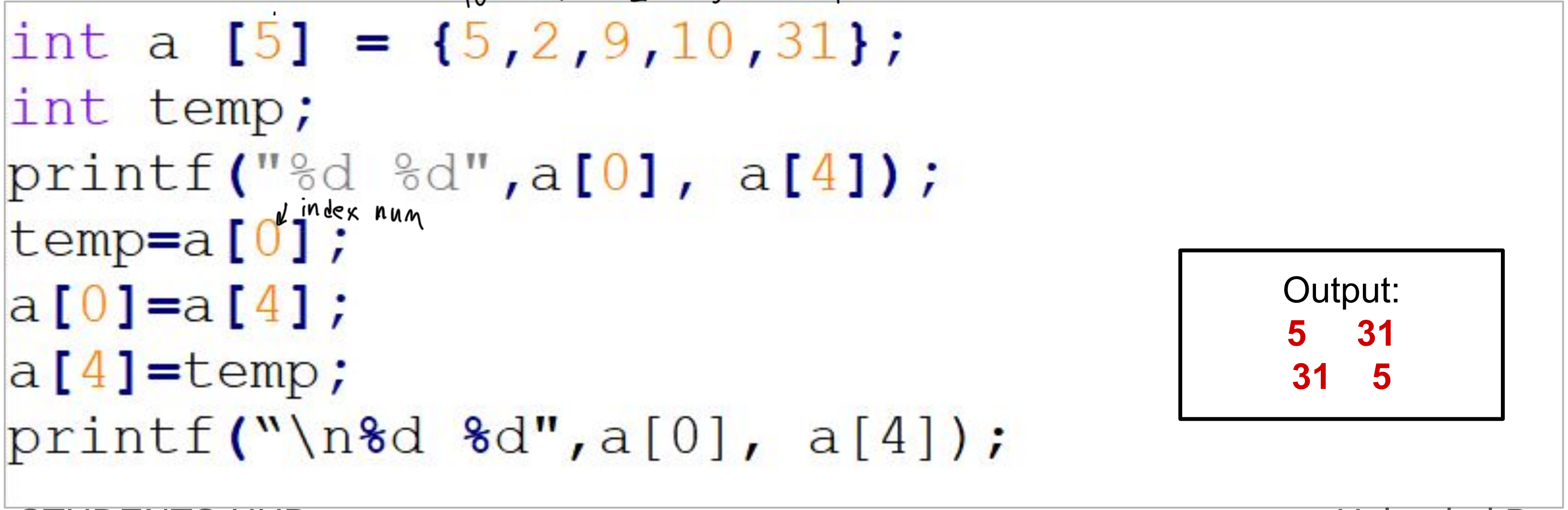
i0 i1 i2 i3 i4



```
int a [5] = {5,2,9,10,31};
int result = a[3%2] + a[2]+a[4/2];
printf("%d\n",result);
printf("%d",a[5%3]);
```

Output:
20 9

i0 i1 i2 i3 i4

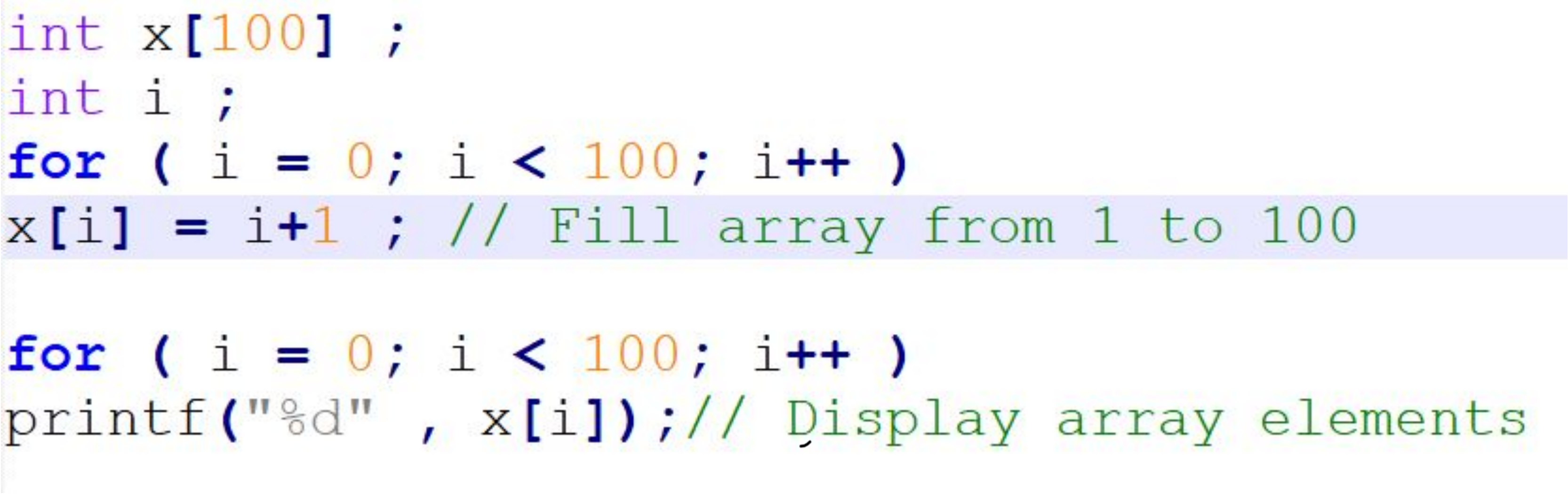


```
int a [5] = {5,2,9,10,31};
int temp;
printf("%d %d",a[0], a[4]);
temp=a[0];
a[0]=a[4];
a[4]=temp;
printf(“\n%d %d",a[0], a[4]);
```

index num

Output:
5 31
31 5

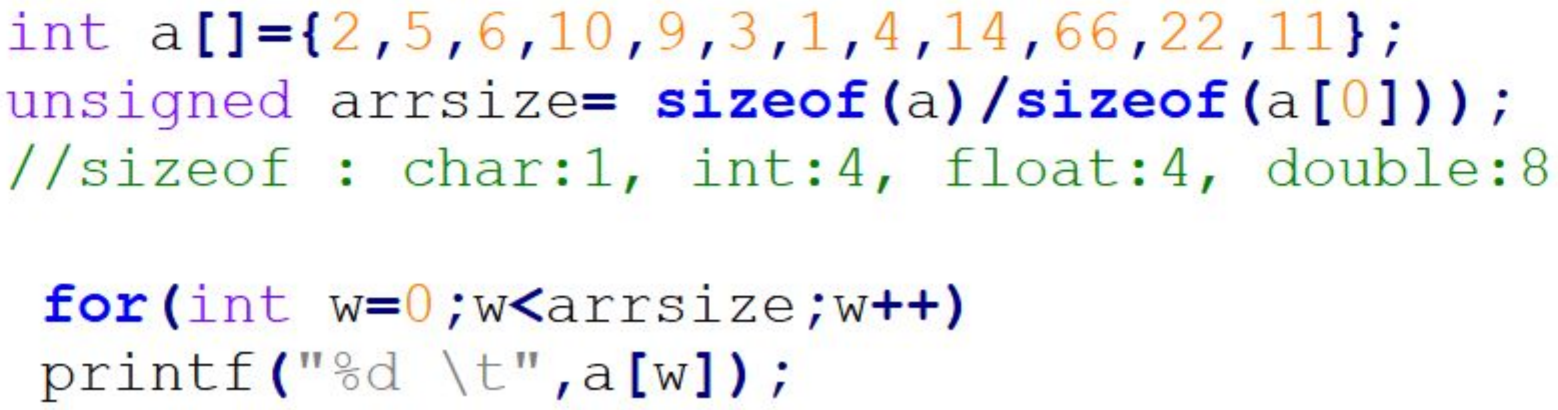
Arrays Example



```
int x[100] ;
int i ;
for ( i = 0; i < 100; i++ )
x[i] = i+1 ; // Fill array from 1 to 100

for ( i = 0; i < 100; i++ )
printf("%d" , x[i]);// Display array elements
```

Arrays Sizeof

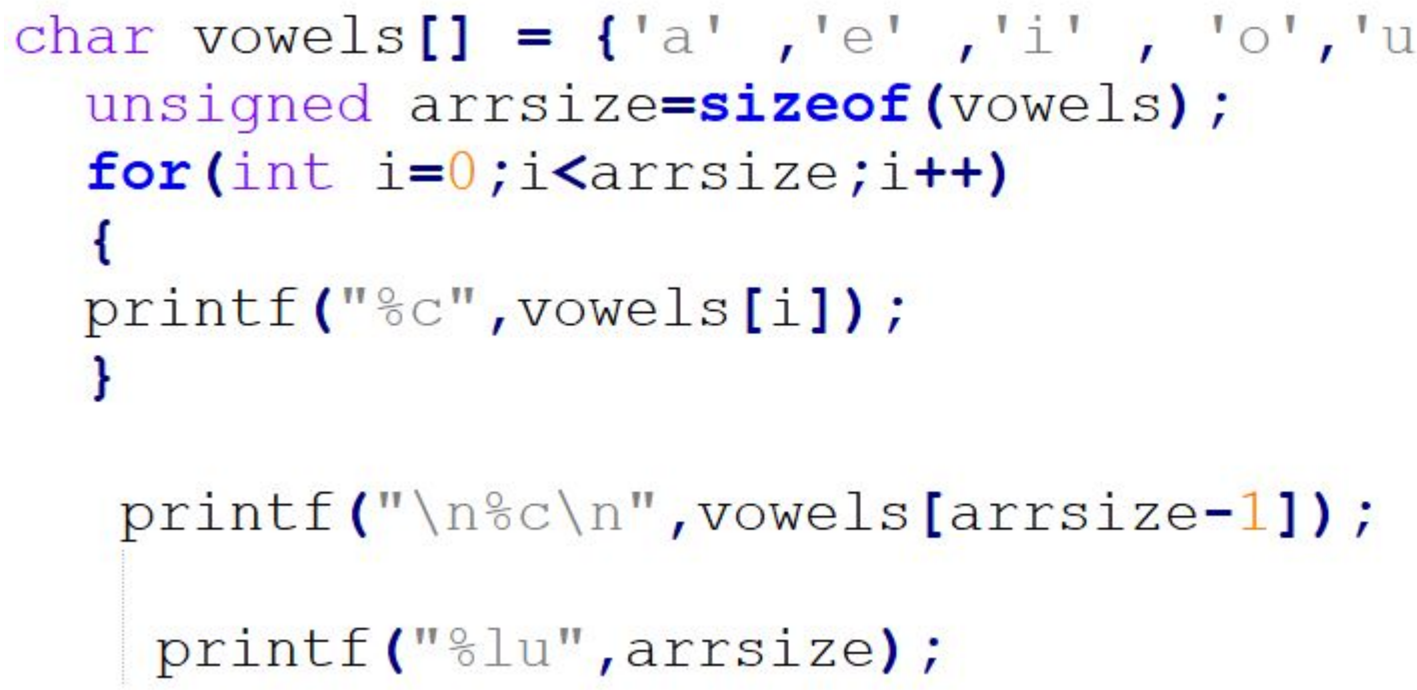
- **Sizeof** is a **compile time** unary operator can be used to compute the size of its operand. The result of sizeof is of **unsigned integral** type.
- Compile time refers to the time at which the source code is converted to a binary code. It doesn't execute (run) the code inside ().
    - `y = sizeof(x++); //value of x doesn't change`



```
int a[]={2,5,6,10,9,3,1,4,14,66,22,11};
unsigned arrsize= sizeof(a)/sizeof(a[0]));
//sizeof : char:1, int:4, float:4, double:8

 for(int w=0;w<arrsize;w++)
 printf("%d \t",a[w]);
```

https://www.geeksforgeeks.org/sizeof-operator-c/

Arrays Example



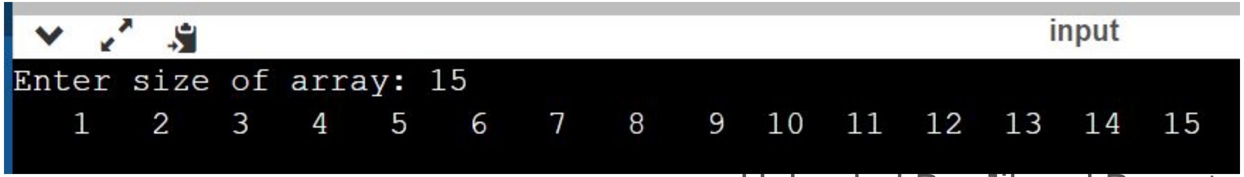
```
char vowels[] = {'a' ,'e' ,'i' , 'o','u' };
   unsigned arrsize=sizeof(vowels);
   for(int i=0;i<arrsize;i++)
   {
   printf("%c",vowels[i]);
   }

    printf("\n%c\n",vowels[arrsize-1]);

      printf("%lu",arrsize);
```

Output :

```
aeiou
u
5
```

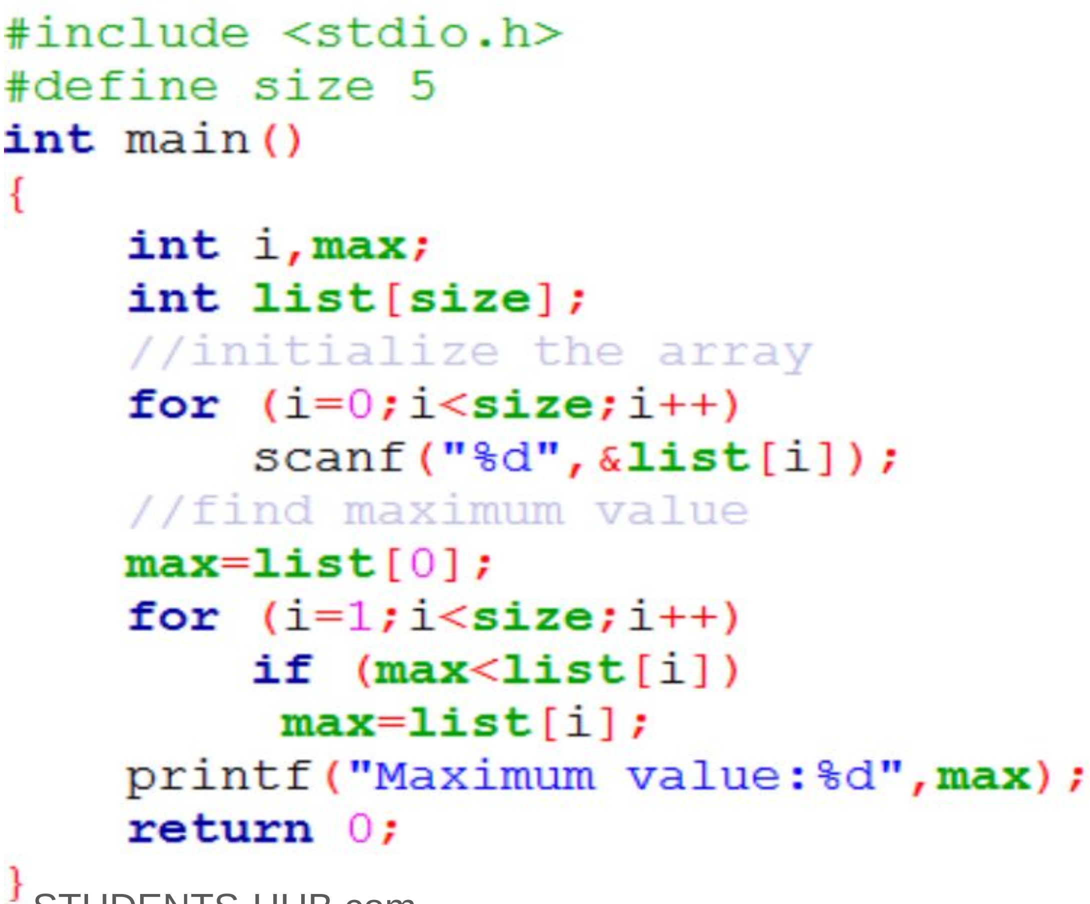
Array Example

```c
#include <stdio.h>
int main()
{
printf("Enter size of array: ");
int size=0;
scanf("%d",&size);
int x[size];
int i ;
for ( i = 0; i < size; i++ )
x[i] = i+1 ;

for ( i = 0; i < size; i++ )
printf("%4d" , x[i]);

    return 0;
}
```



```
Enter size of array: 15
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

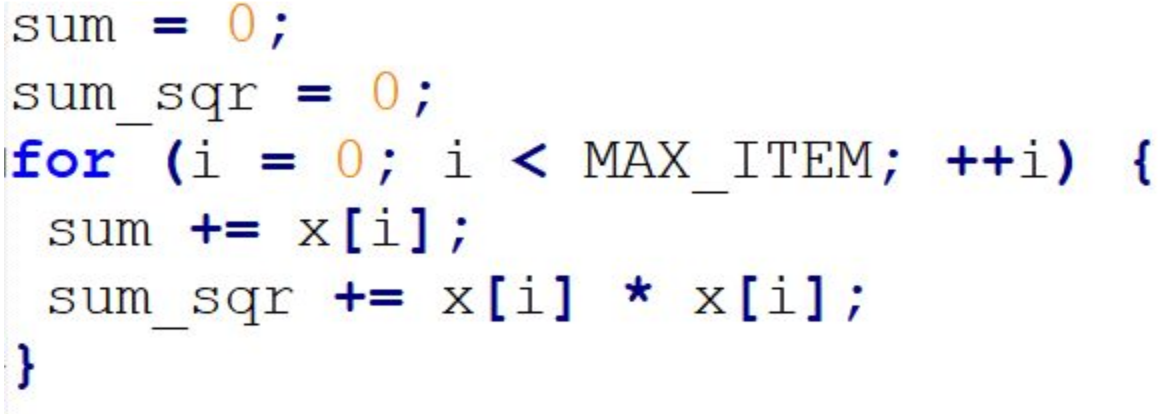
Arrays Example



```
#include <stdio.h>
#define size 5
int main()
{
    int i,max;
    int list[size];
    //initialize the array
    for (i=0;i<size;i++)
        scanf("%d",&list[i]);
    //find maximum value
    max=list[0];
    for (i=1;i<size;i++)
        if (max<list[i])
         max=list[i];
    printf("Maximum value:%d",max);
    return 0;
}
```

Arrays Example

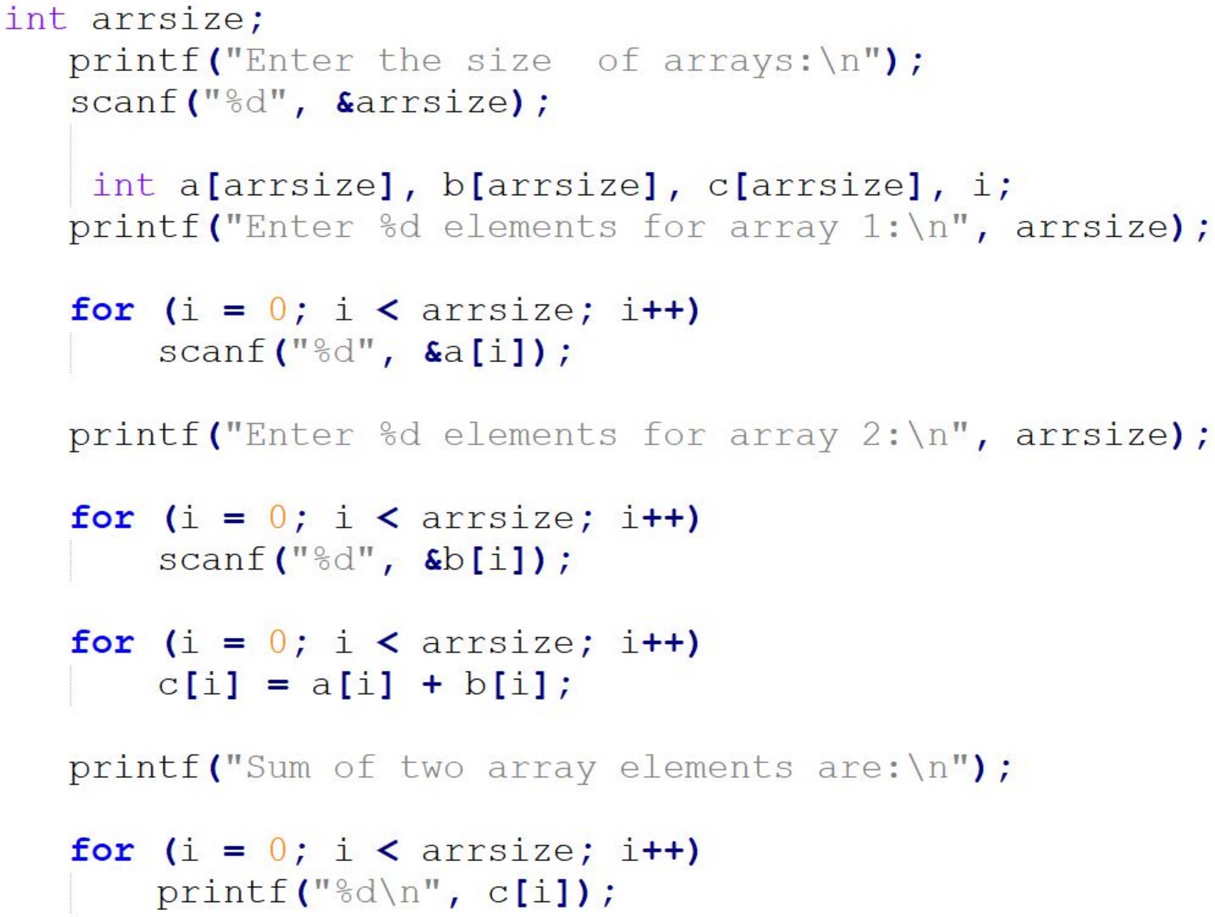
Computes the sum and the sum of the squares of all data.



```
sum = 0;
sum_sqr = 0;
for (i = 0; i < MAX_ITEM; ++i) {
 sum += x[i];
 sum_sqr += x[i] * x[i];
}
```

Array - Example

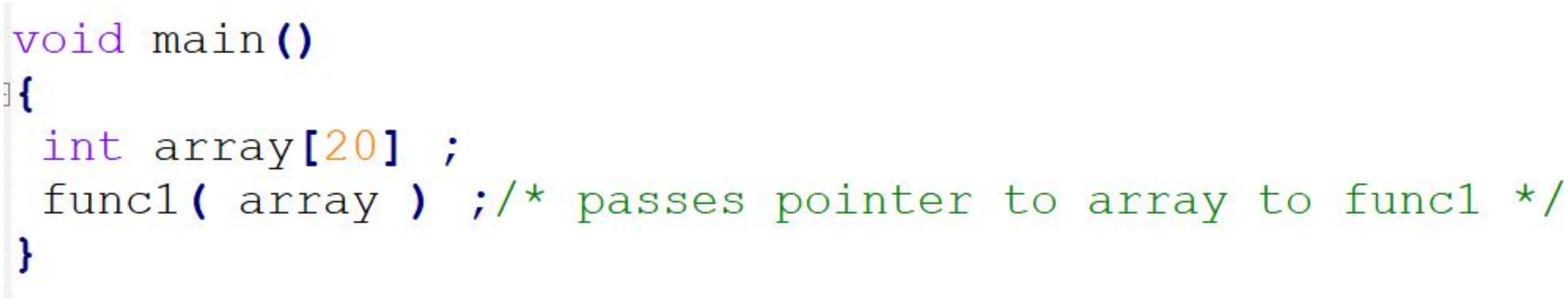
Sum two arrays.



```
int arrsize;
    printf("Enter the size  of arrays:\n");
    scanf("%d", &arrsize);

     int a[arrsize], b[arrsize], c[arrsize], i;
    printf("Enter %d elements for array 1:\n", arrsize);

    for (i = 0; i < arrsize; i++)
        scanf("%d", &a[i]);

    printf("Enter %d elements for array 2:\n", arrsize);

    for (i = 0; i < arrsize; i++)
        scanf("%d", &b[i]);

    for (i = 0; i < arrsize; i++)
        c[i] = a[i] + b[i];

    printf("Sum of two array elements are:\n");

    for (i = 0; i < arrsize; i++)
        printf("%d\n", c[i]);
```

Arrays as arguments & parameters

- In C it is impossible to pass an entire array as an argument to a function
- The **address of the array** is passed as a parameter to the function.

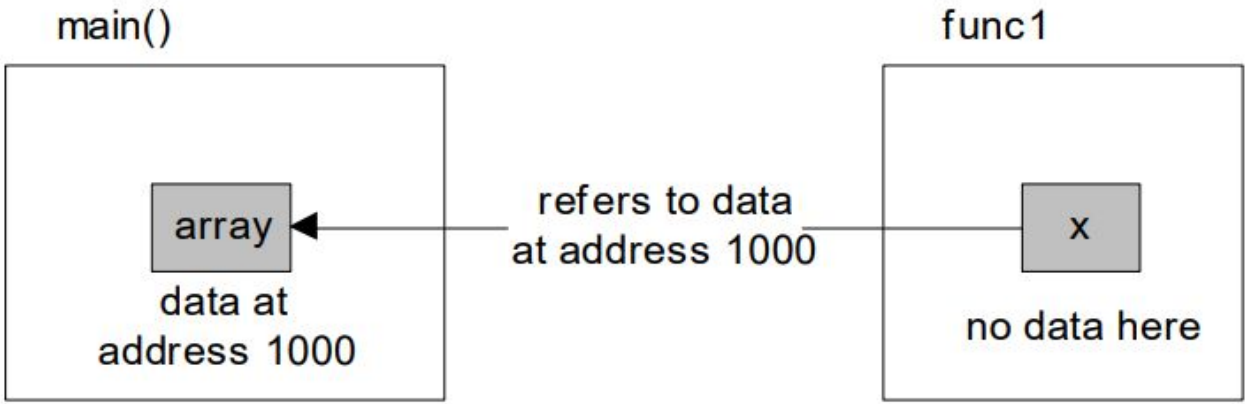


```
void main()
{
 int array[20] ;
 func1( array ) ;/* passes pointer to array to func1 */
}
```

- The address of the array passed to the function that will be able to manipulate the actual data of the array in main(). This is call by reference.

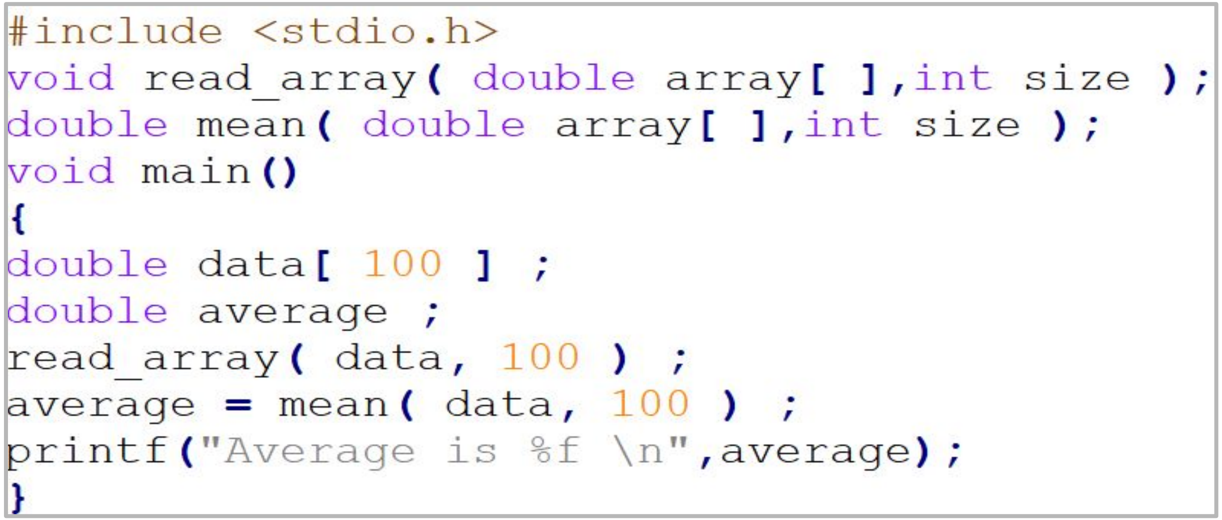
Arrays as arguments & parameters

- In the function receiving the array the formal parameters can be declared in one of three almost equivalent ways as follows.
    - func1 (int x[10]) { ... } // As a sized array
    - func1 (int x[ ]) { ... } // As an unsized array
    - func1 (int *x) { ... } // As an actual pointer

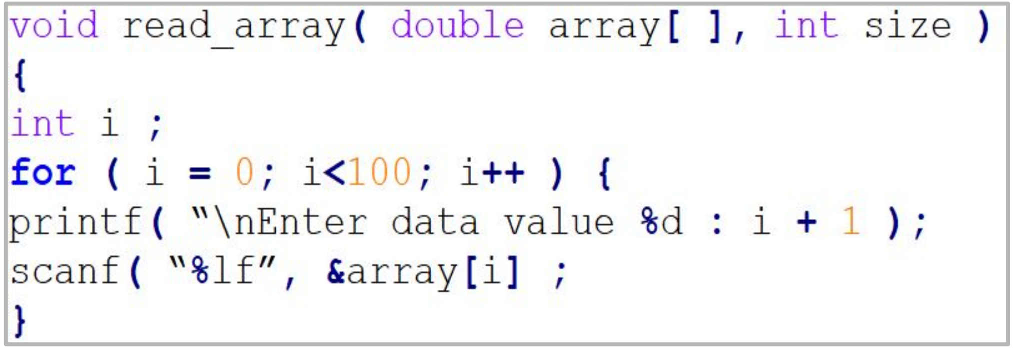


Arrays as arguments & parameters

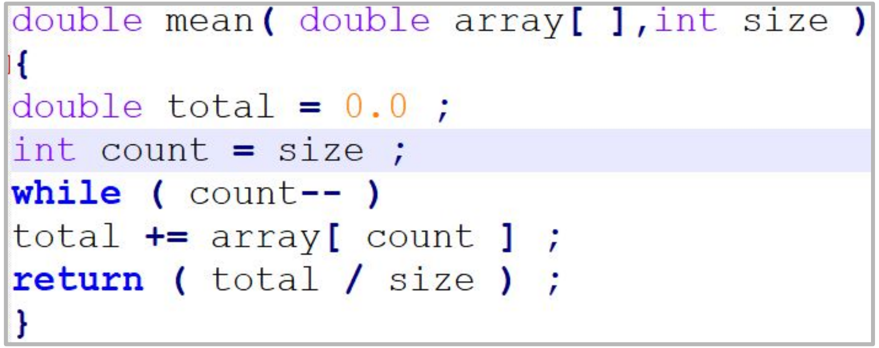
جرب



```
#include <stdio.h>
void read_array( double array[ ],int size );
double mean( double array[ ],int size );
void main()
{
double data[ 100 ] ;
double average ;
read_array( data, 100 ) ;
average = mean( data, 100 ) ;
printf("Average is %f \n",average);
}
```



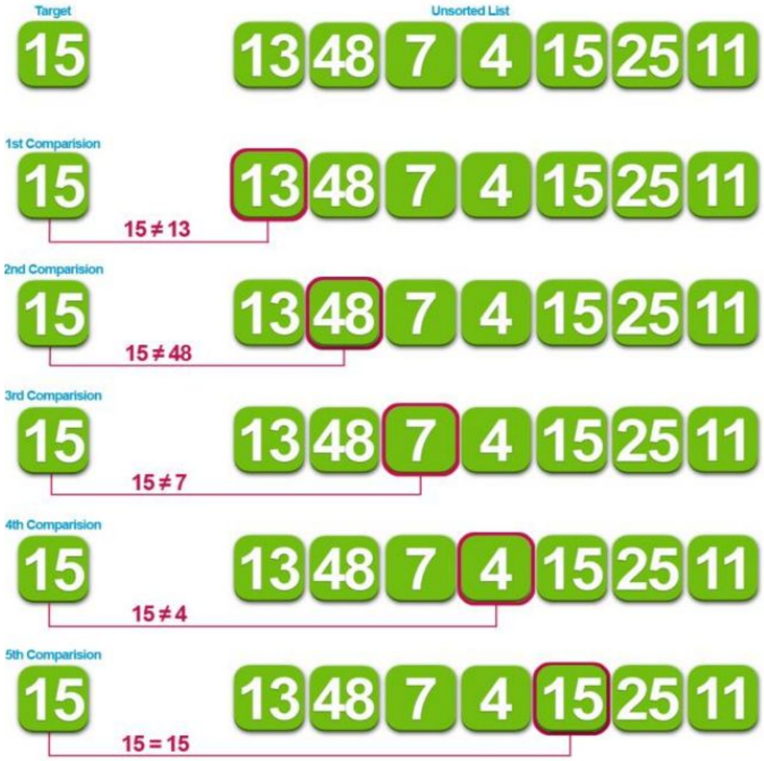
```
void read_array( double array[ ], int size )
{
int i ;
for ( i = 0; i<100; i++ ) {
printf( “\nEnter data value %d : i + 1 );
scanf( “%lf”, &array[i] ;
}
```



```
double mean( double array[ ],int size )
{
double total = 0.0 ;
int count = size ;
while ( count-- )
total += array[ count ] ;
return ( total / size ) ;
}
```

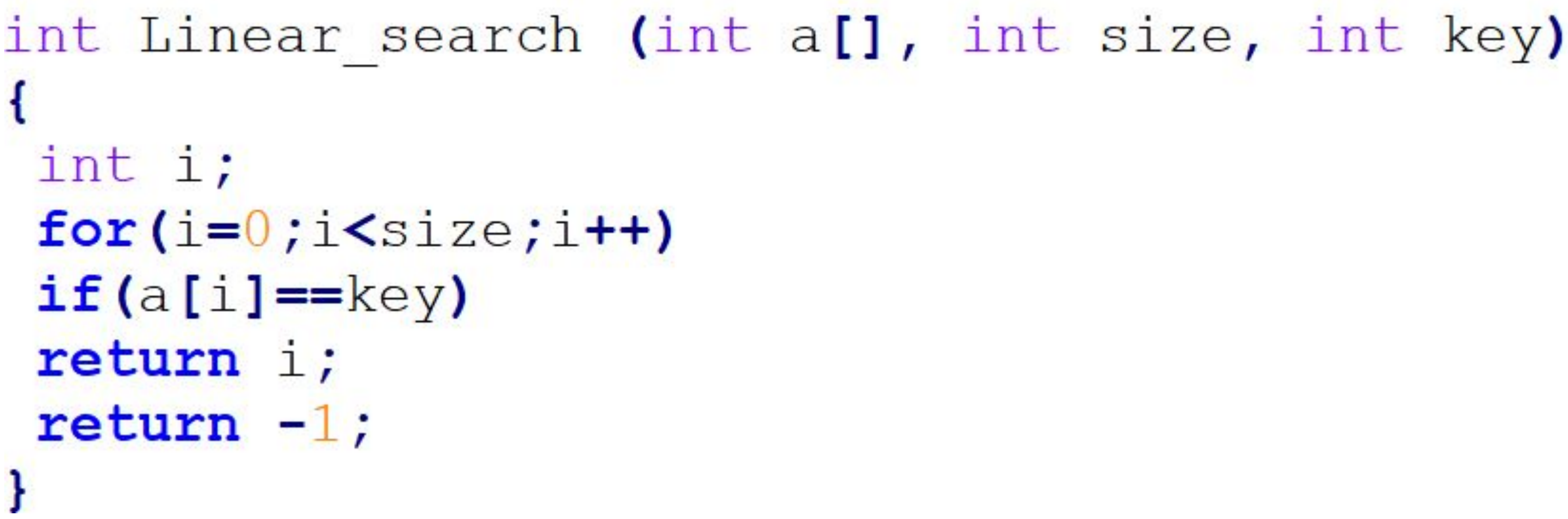
Arrays Linear Search

- **linear search** : compare target element with each element in the array list.
- E.g : Find the value=7
  - Target value =7



- Note : The array is unsorted.

Arrays Linear Search

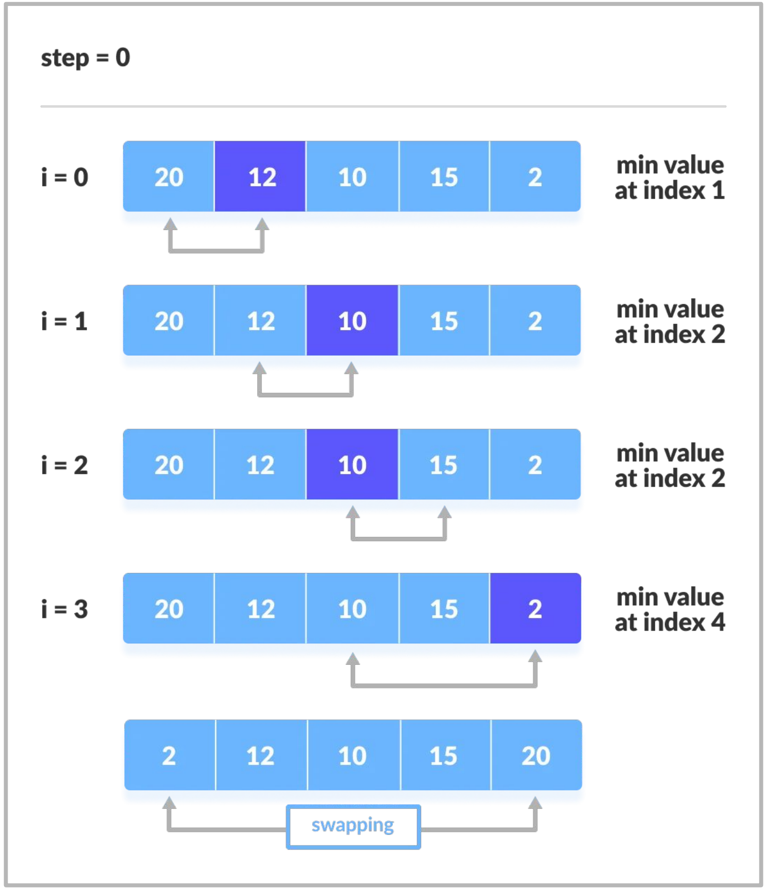


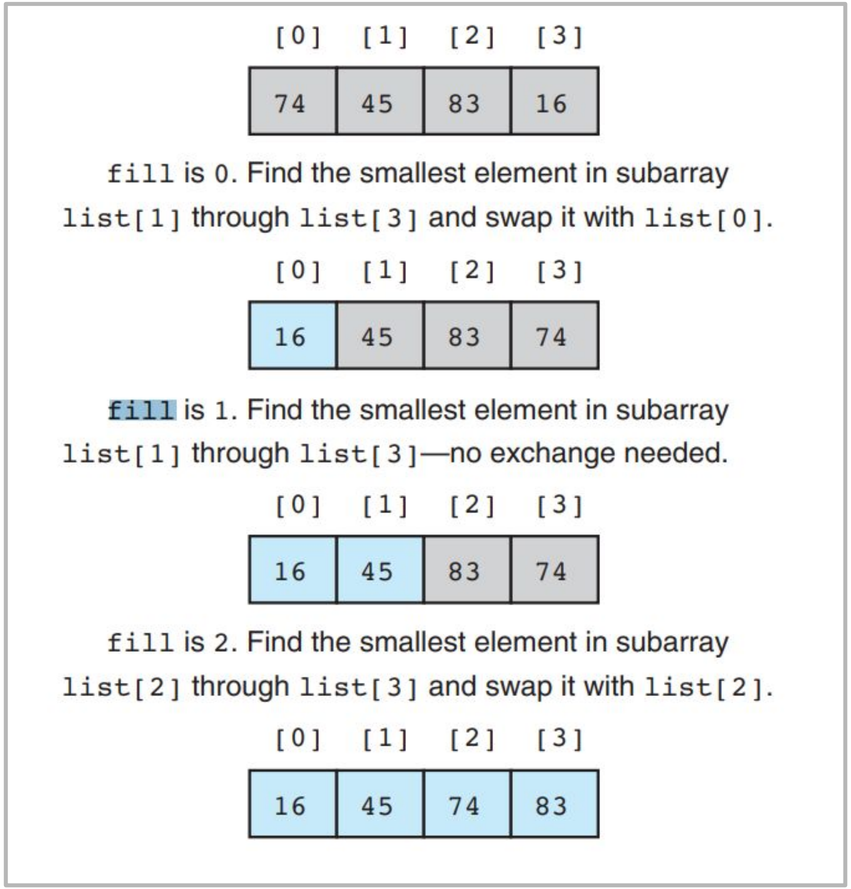
```
int Linear_search (int a[], int size, int key)
{
 int i;
 for(i=0;i<size;i++)
 if(a[i]==key)
 return i;
 return -1;
}
```

Sorting an Array

- ***Selection sort*** is a fairly intuitive sorting algorithm (but not very efficient).
- Finds the position of the smallest element in the subarray
  - list[first] through list[last].
- Pre: first < last and elements 0 through last of array list are defined.
  - first n elements of list are defined and n >= 0
- Post: Returns the subscript k of the smallest element in the subarray
  - i.e., list[k] <= list[i] for all i in the subarray

Sorting an Array - Selection sort



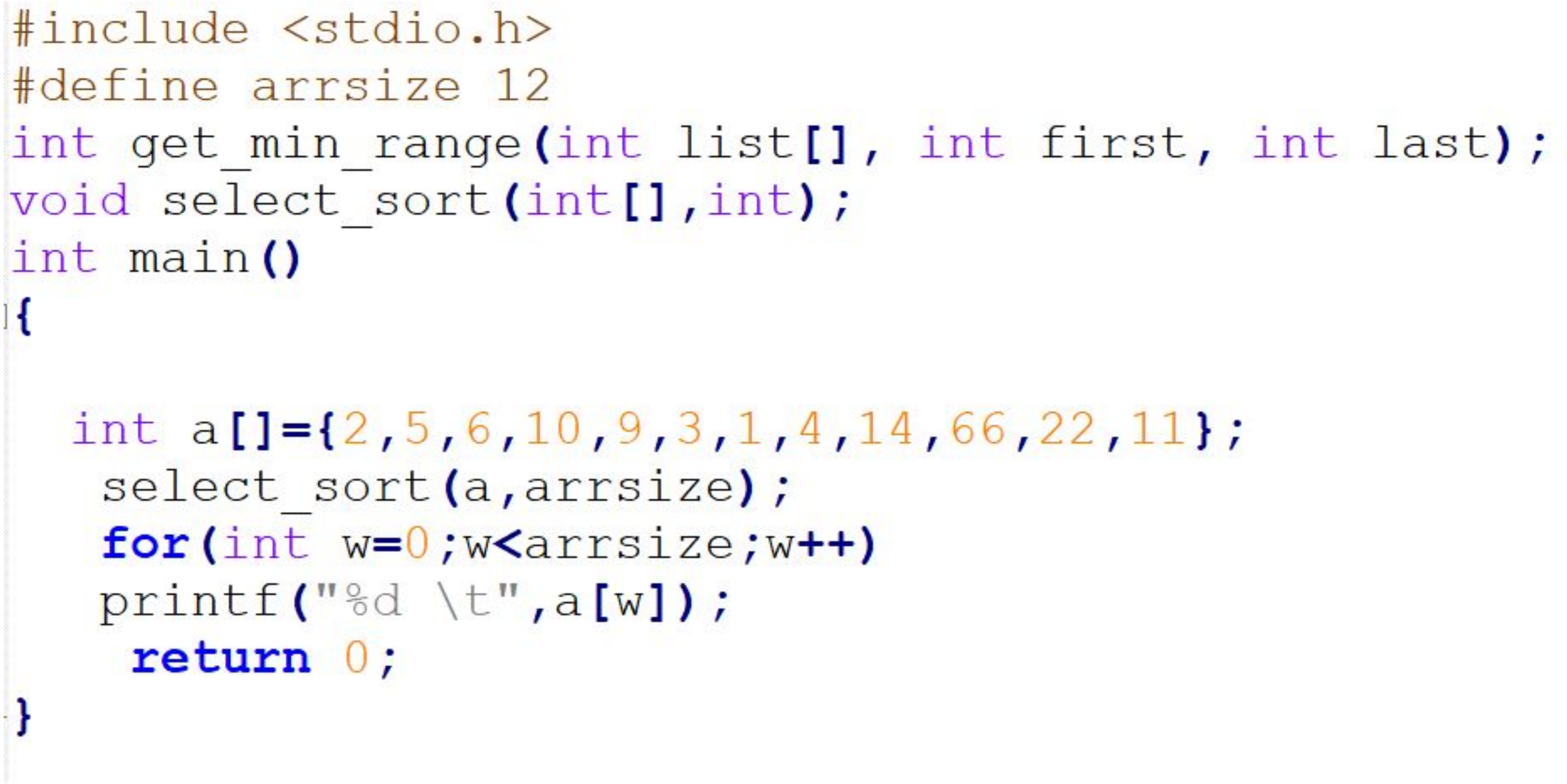


fill is 0. Find the smallest element in subarray list[1] through list[3] and swap it with list[0].

fill is 1. Find the smallest element in subarray list[1] through list[3]—no exchange needed.

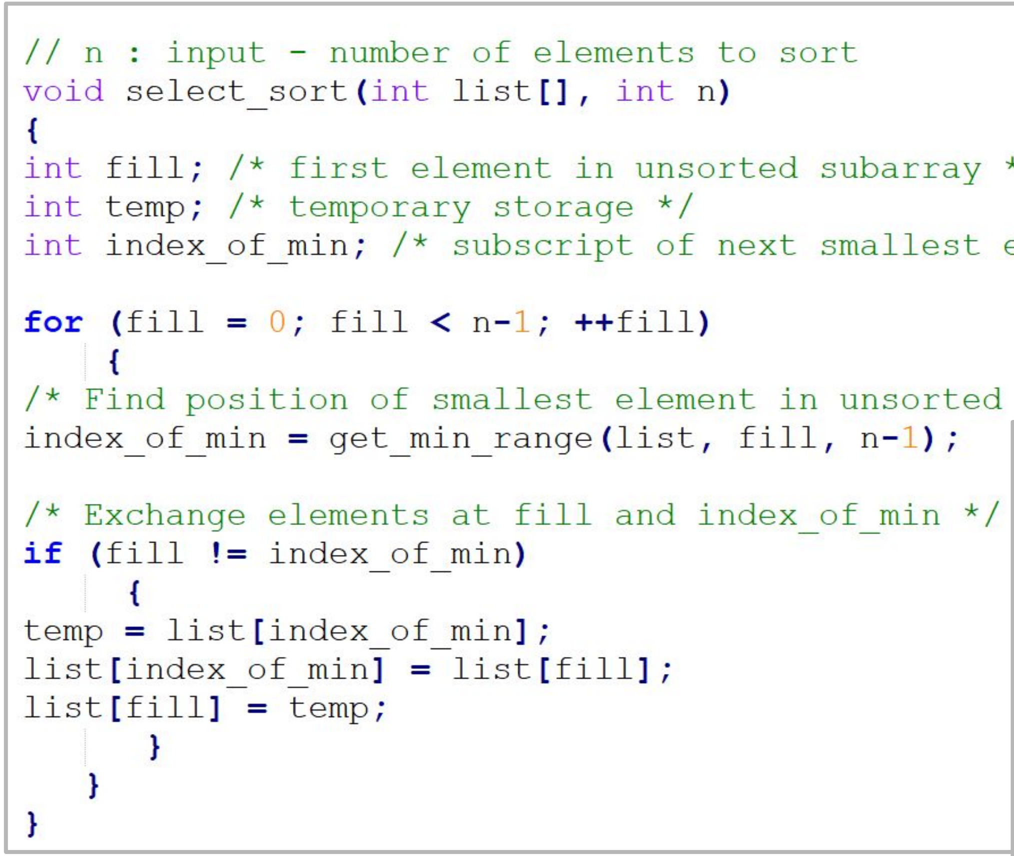
fill is 2. Find the smallest element in subarray list[2] through list[3] and swap it with list[2].

Sorting an Array - Selection sort

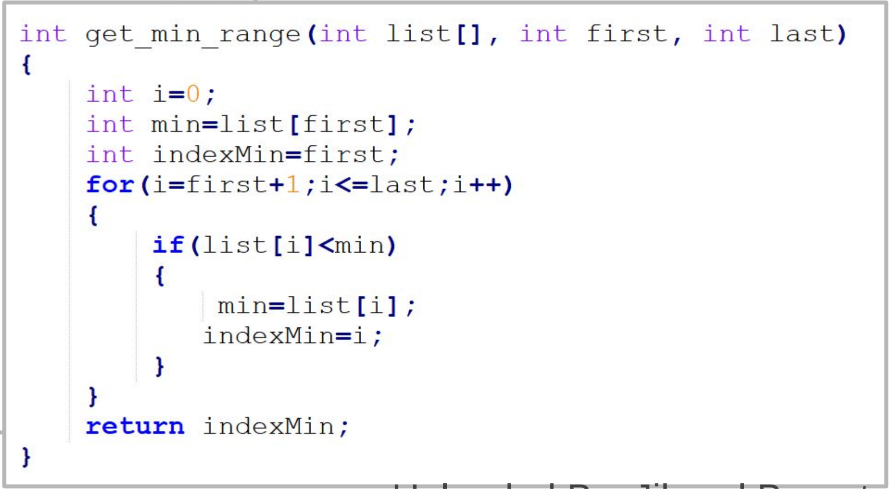


```
#include <stdio.h>
#define arrsize 12
int get_min_range(int list[], int first, int last);
void select_sort(int[],int);
int main()
{

  int a[]={2,5,6,10,9,3,1,4,14,66,22,11};
   select_sort(a,arrsize);
   for(int w=0;w<arrsize;w++)
   printf("%d \t",a[w]);
    return 0;
}
```

Sorting an Array - Selection sort



```
// n : input - number of elements to sort
void select_sort(int list[], int n)
{
int fill; /* first element in unsorted subarray */
int temp; /* temporary storage */
int index_of_min; /* subscript of next smallest element */

for (fill = 0; fill < n-1; ++fill)
    {
/* Find position of smallest element in unsorted subarray */
index_of_min = get_min_range(list, fill, n-1);

/* Exchange elements at fill and index_of_min */
if (fill != index_of_min)
     {
temp = list[index_of_min];
list[index_of_min] = list[fill];
list[fill] = temp;
      }
    }
}
```



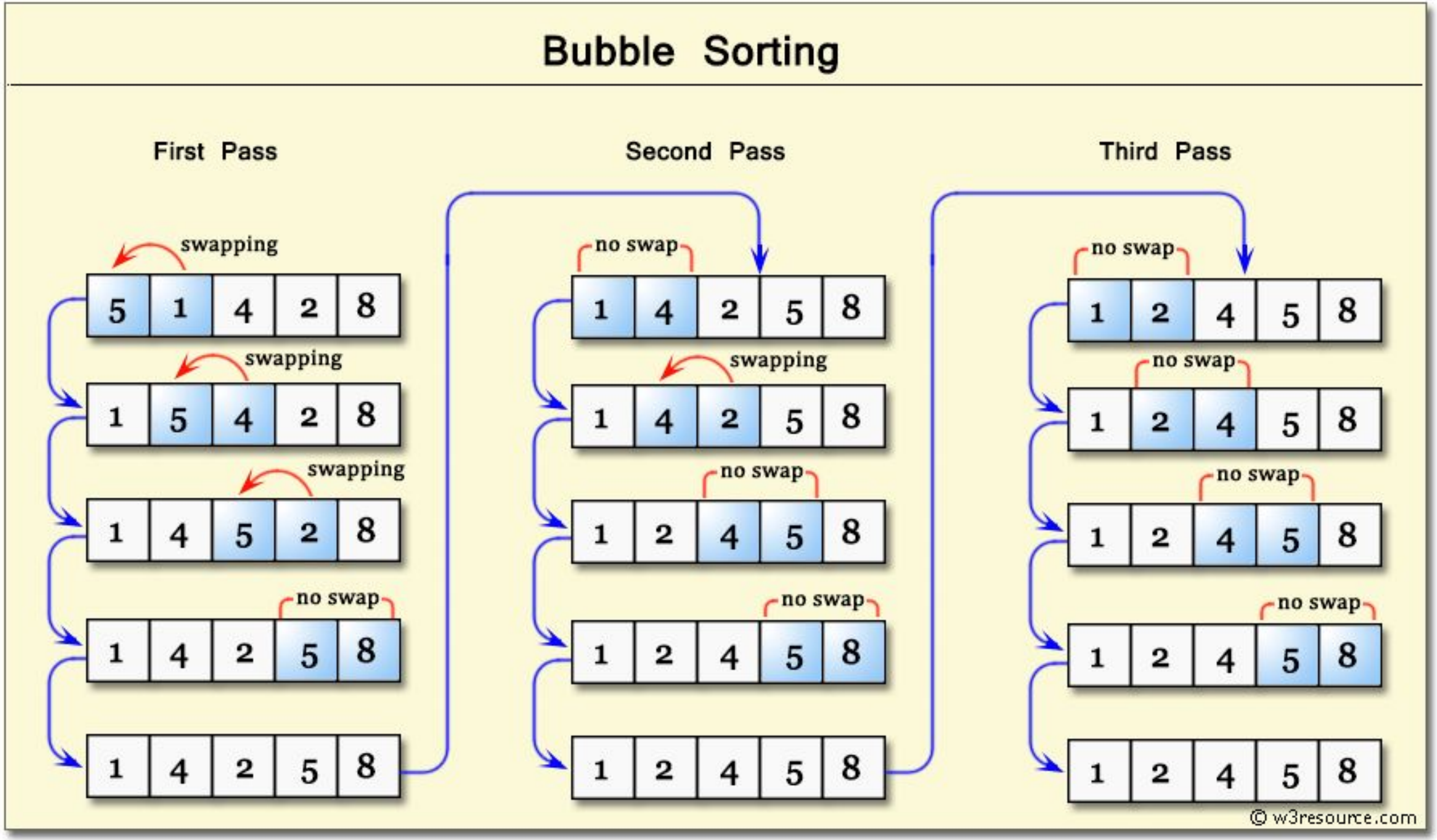
```
int get_min_range(int list[], int first, int last)
{
    int i=0;
    int min=list[first];
    int indexMin=first;
    for(i=first+1;i<=last;i++)
    {
        if(list[i]<min)
        {
            min=list[i];
            indexMin=i;
        }
    }
    return indexMin;
}
```

Sorting an Array - Bubble Sort

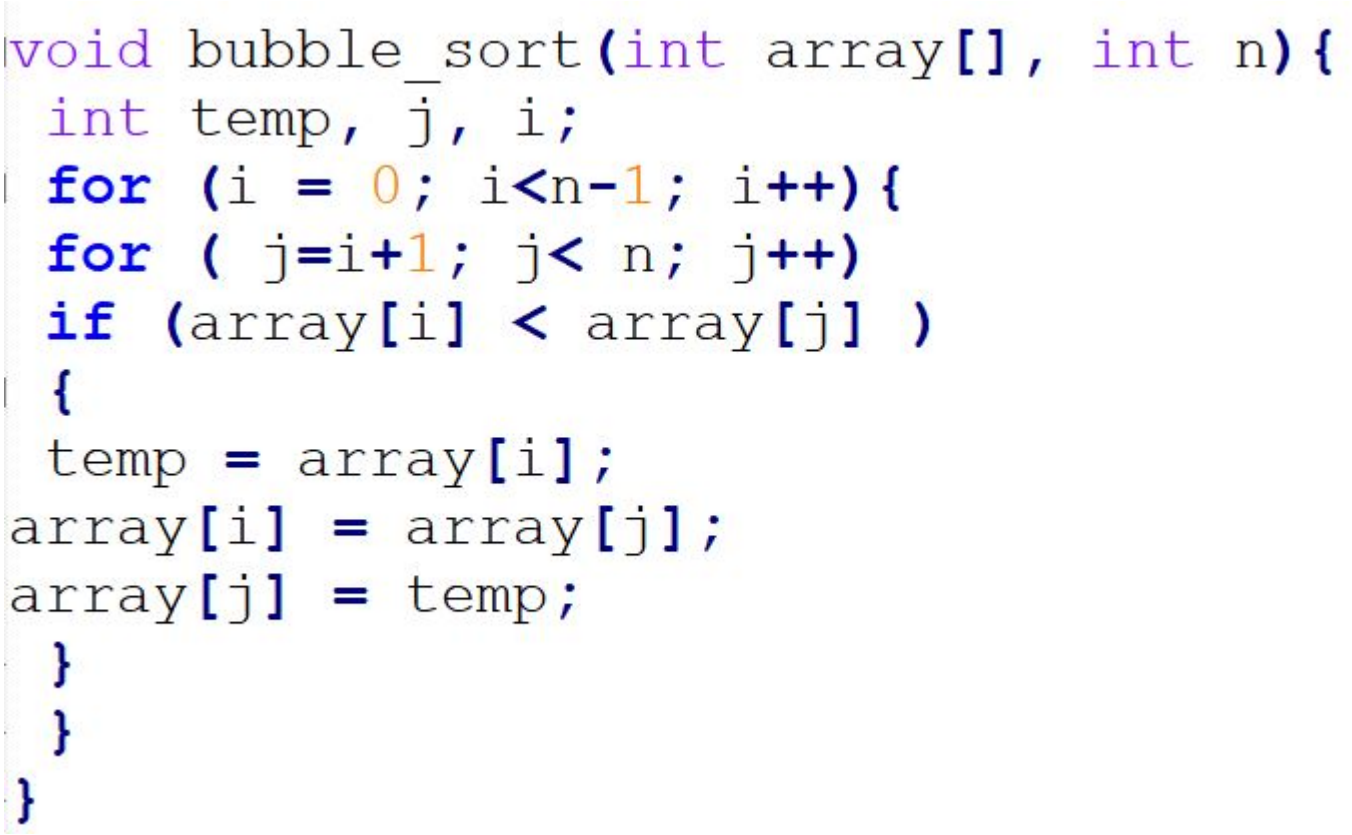
Bubble Sort is a sorting algorithm which compares two adjacent elements and swap them if they are not in the right order.

The smaller values "*bubble*" to the top of the array (toward element 0), while the larger values sink to the bottom of the array. After the first pass of a bubble sort, the last array element is in the correct position; after the second pass the last two elements are correct, and so on

Sorting an Array - Bubble Sort



Sorting an Array - Bubble Sort

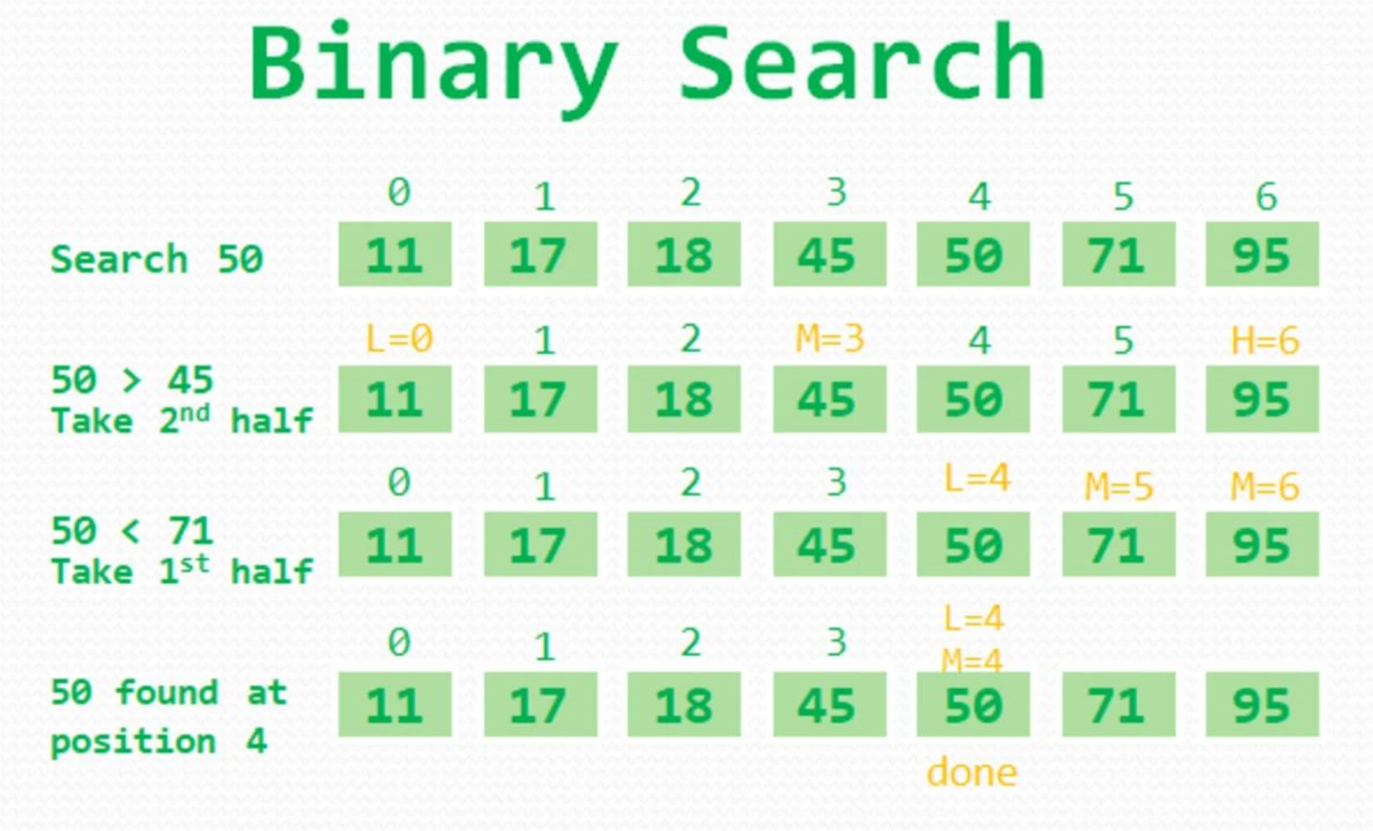


```
void bubble_sort(int array[], int n){
 int temp, j, i;
 for (i = 0; i<n-1; i++){
 for ( j=i+1; j< n; j++)
 if (array[i] < array[j] )
 {
 temp = array[i];
array[i] = array[j];
array[j] = temp;
 }
 }
}
```

Array - exercise

- Write a complete C function to create a ***Binary Search Algorithm*** using Iterative Method.
- A binary search technique works only on a sorted array, so an array must be sorted to apply binary search on the array.
- The logic behind the binary search is that there is a key. This key holds the value to be searched. The highest and the lowest value are added and divided by 2. Highest and lowest and the first and last element in the array. The mid value is then compared with the key. If mid is equal to the key, then we get the output directly. Else if the key is greater then mid then the mid+1 becomes the lowest value and the process is repeated on the shortened array. Else if the key value is less then mid, mid-1 becomes the highest value and the process is repeated on the shortened array. If it is not found anywhere, an error message is displayed

Array - exercise



Binary Search

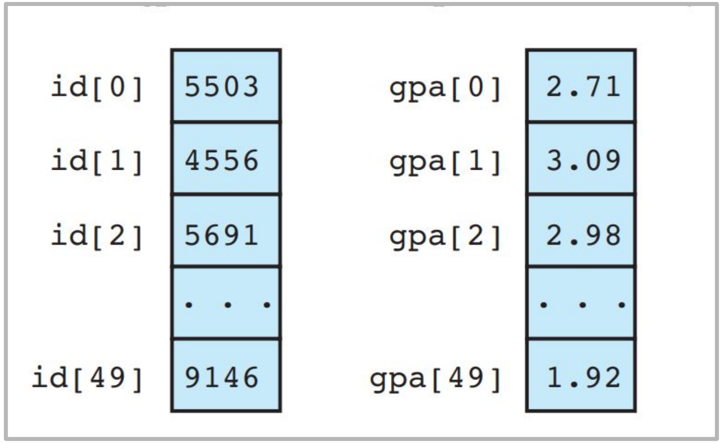
	0	1	2	3	4	5	6
Search 50	11	17	18	45	50	71	95
	L=0	1	2	M=3	4	5	H=6
50 > 45 Take 2nd half	11	17	18	45	50	71	95
	0	1	2	3	L=4	M=5	M=6
50 < 71 Take 1st half	11	17	18	45	50	71	95
	0	1	2	3	L=4 M=4		
50 found at position 4	11	17	18	45	50	71	95
					done		

Chapter 7

- Parallel Arrays

Parallel Arrays

- **Parallel Arrays** two or more arrays with the same number of elements used for storing related information about a collection of data objects.
  - int id[NUM_STUDENTS];
  - double gpa[NUM_STUDENTS];



id[0]	5503	gpa[0]	2.71
id[1]	4556	gpa[1]	3.09
id[2]	5691	gpa[2]	2.98
	. . .		. . .
id[49]	9146	gpa[49]	1.92

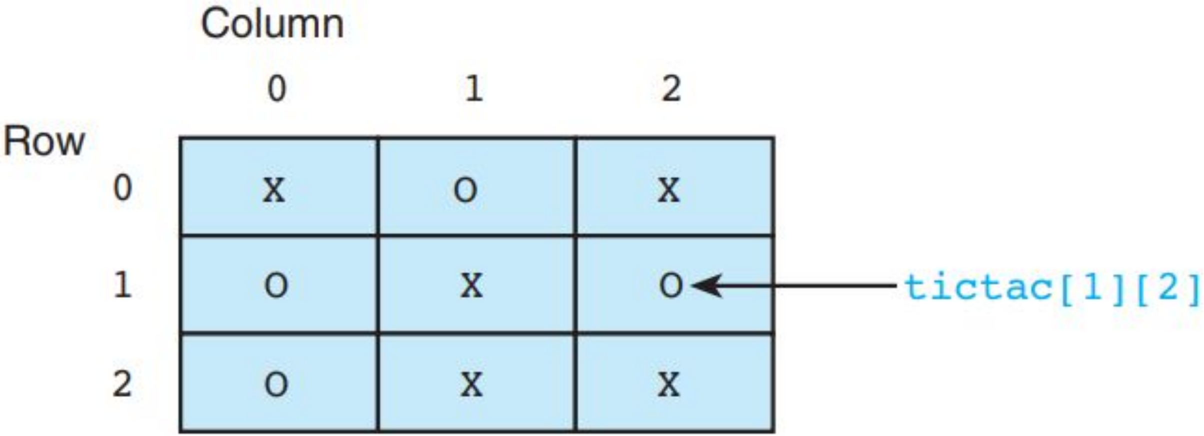
```c
for (i = 0; i < NUM_STUDENTS; ++i) {
    printf("Enter the id and gpa for student %d: ", i);
    scanf("%d%lf", &id[i], &gpa[i]);
    printf("%d   %4.2f\n", id[i], gpa[i]);
}
```

Chapter 7

- Multidimensional Arrays

2D Array

- We can use any dimension of array in C (Multidimensional).
- Only two or three dimensional arrays are practical.
- Syntax : **type** **name** [rows] [columns] ;
  - char tictac[3][3];



	0	1	2
0	X	O	X
1	O	X	O
2	O	X	X

2D Array example

- **int a[2][2]=8;**
- a[1][0]=9;
- a[0][3]=5;
- **a[0][1]**=a[0][3]+ a[1][0];

	0	1	2	0
0		14		5
1	9		8	

2D Array Initialize

- To initialise a multidimensional array all but the leftmost index must be specified.
  - int d[ ] [3] = { 1, 2, 3, 4, 5, 6 } ;
- It is more useful to enclose the individual row values in curly braces for clarity.
  - int d[ ] [3] = { {1, 2, 3}, {4, 5, 6} } ;

1	2	3
4	5	6

2D Array Initialize

- int grades[5][3] = **{**

  { 78, 83, 82 }, { 90, 88, 94 }, { 71, 73, 78 }, { 97, 96, 95 }, { 89, 93, 90 }

  };

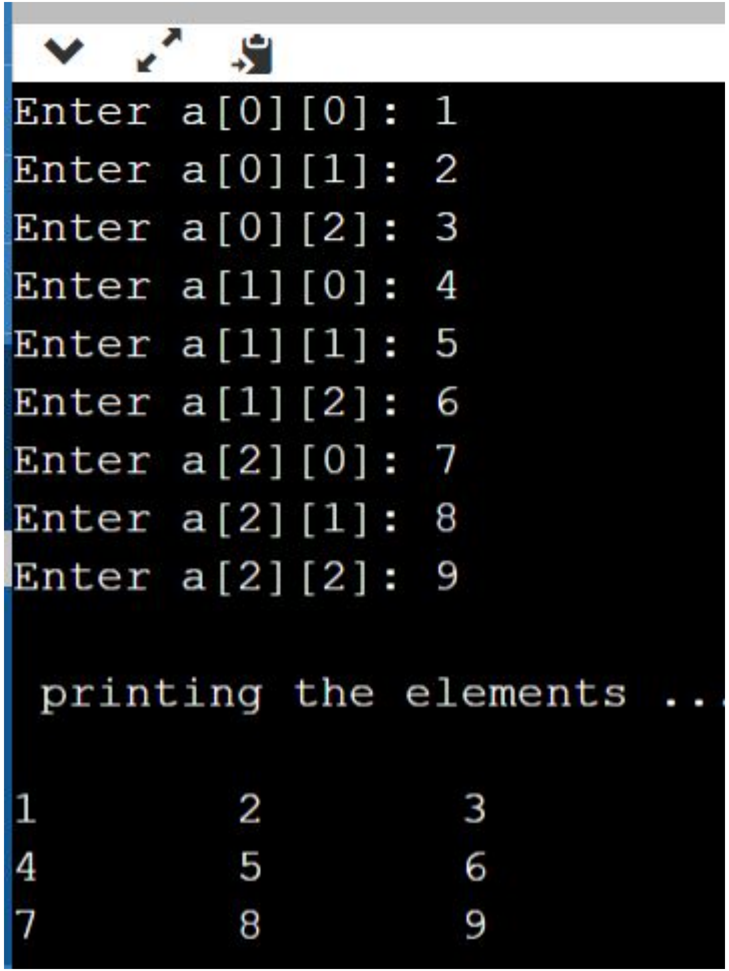
One-D array

A Two-D Array is an array of arrays.

Each row is itself a One-D array.

2D Array example

```c
int arr[3][3],i,j;
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)
        {
            printf("Enter a[%d][%d]: ",i,j);
            scanf("%d",&arr[i][j]);
        }
    }
    printf("\n printing the elements ....\n");
    for(i=0;i<3;i++)
    {
        printf("\n");
        for (j=0;j<3;j++)
        {
            printf("%d\t",arr[i][j]);
        }
    }
```



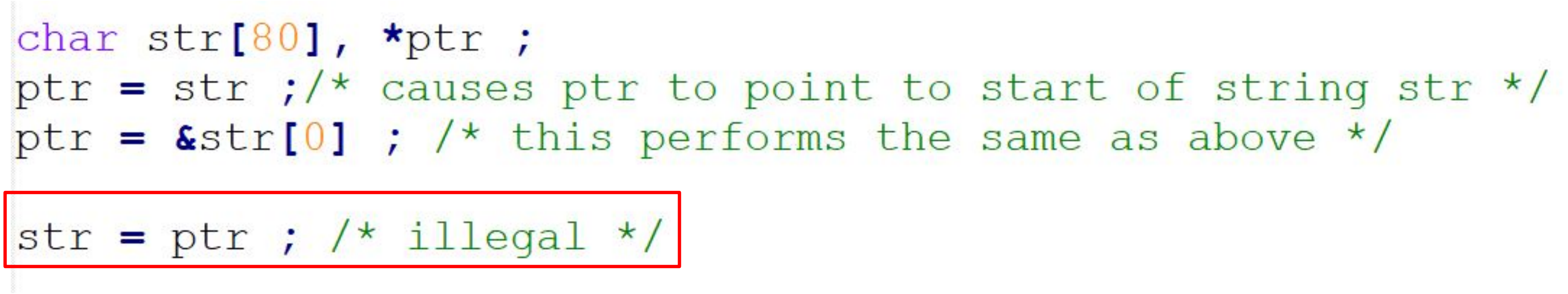
```
Enter a[0][0]: 1
Enter a[0][1]: 2
Enter a[0][2]: 3
Enter a[1][0]: 4
Enter a[1][1]: 5
Enter a[1][2]: 6
Enter a[2][0]: 7
Enter a[2][1]: 8
Enter a[2][2]: 9

 printing the elements ...

1       2       3
4       5       6
7       8       9
```

Chapter 7

- Pointers and Arrays

Pointers and Arrays

The name of an array is actually the address in memory of the array and so it is essentially a ***constant pointer***.

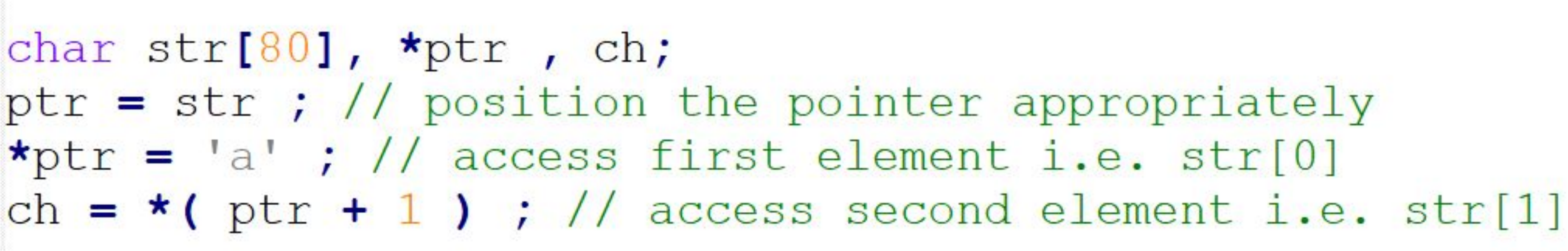


```
char str[80], *ptr ;
ptr = str ;/* causes ptr to point to start of string str */
ptr = &str[0] ; /* this performs the same as above */

str = ptr ; /* illegal */
```

str is a constant pointer and so its value i.e. the address it holds cannot be changed.

Pointers and Arrays

- We can use pointers in much the same way to access an array elements.
  - ***(array + index)** is equivalent to **array[index]**.
  - The **parentheses** are necessary above as the precedence of ***** is higher than that of **+.**



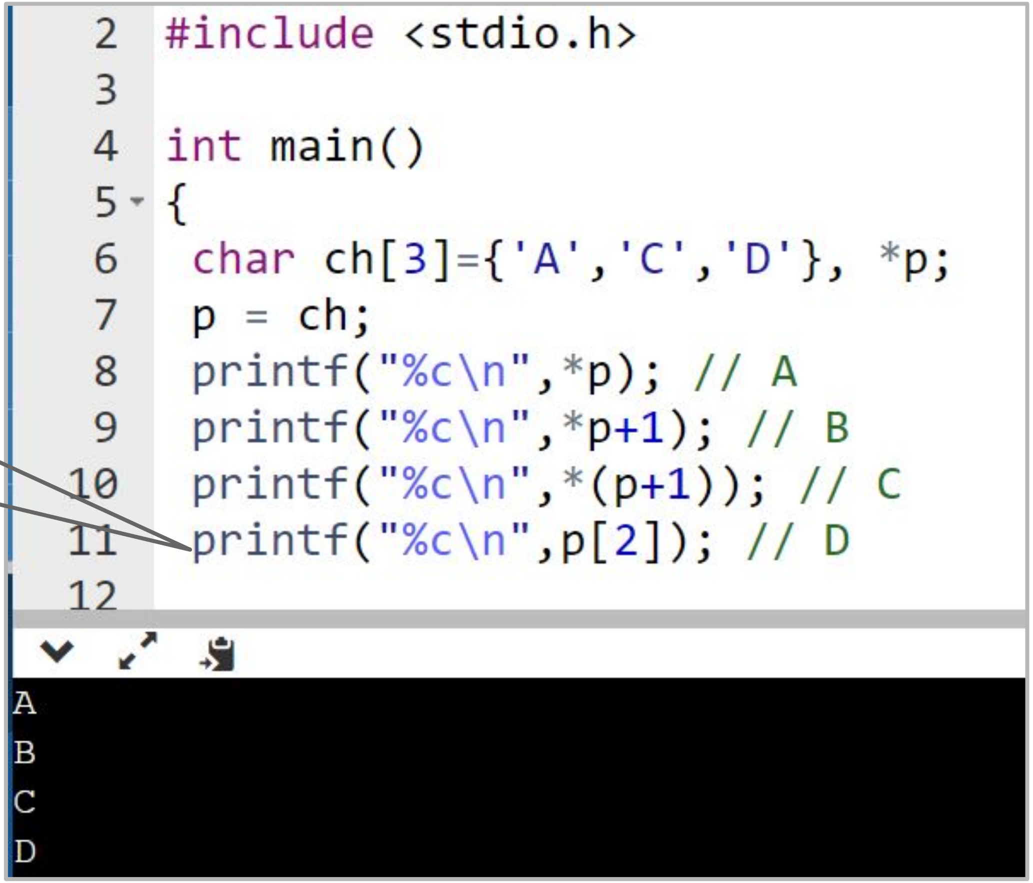
```
char str[80], *ptr , ch;
ptr = str ; // position the pointer appropriately
*ptr = 'a' ; // access first element i.e. str[0]
ch = *( ptr + 1 ) ; // access second element i.e. str[1]
```

- **ch = *ptr + 1 ;** // change the value by add 1 to ASCII code to become **b**

Pointers and Arrays

- Example

Accesses element 3 of array by indexing a pointer.



```c
#include <stdio.h>

int main()
{
 char ch[3]={'A','C','D'}, *p;
 p = ch;
 printf("%c\n",*p); // A
 printf("%c\n",*p+1); // B
 printf("%c\n",*(p+1)); // C
 printf("%c\n",p[2]); // D

```

```
A
B
C
D
```

Chapter 7

- Pointer Arithmetic

Pointer Arithmetic

- Assignment
  - int count, *p1, *p2 ;

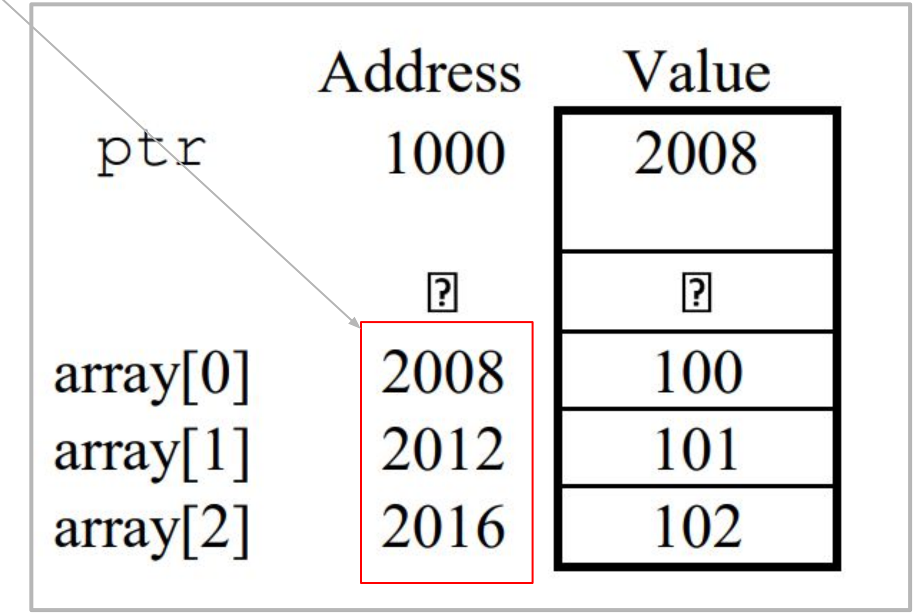
    p1 = &count ; // assign the address of a variable directly

    p2 = p1 ; // assign the value of another pointer variable, an address
- Addition / Subtraction
  - The value a pointer holds four byte address of a variable in memory. It is possible to modify this address by integer addition and subtraction if necessary.

Pointer Arithmetic

- Addition / Subtraction
  - Consider the following we assume a 32-bit system and hence 32-bit integers. (Each move step 4 bit added)

```
int *ptr ;

int array[3] = { 100, 101, 102 } ;

ptr = array ; // 100

ptr += 1 ; // 101

ptr = ptr - 1 ; //100

ptr++; //101
```



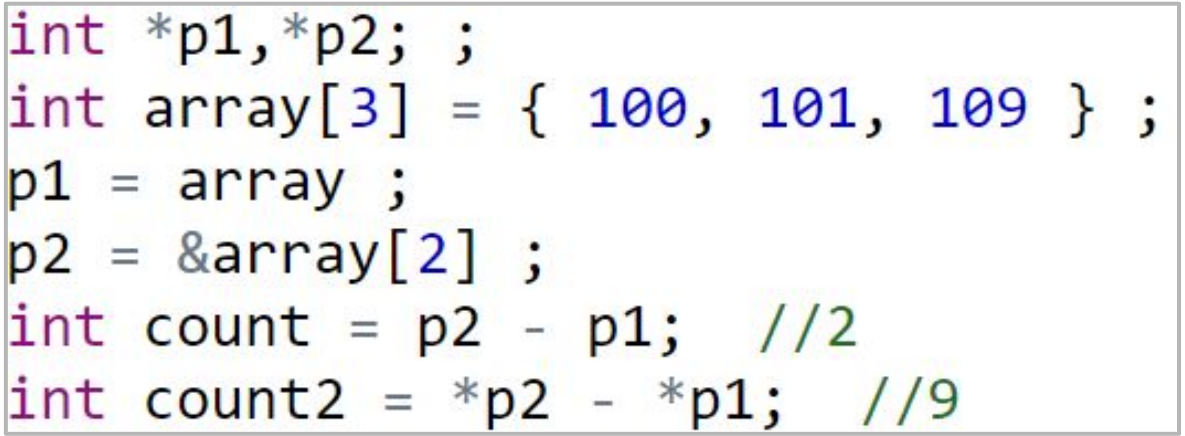
	Address	Value
ptr	1000	2008
	⍰	⍰
array[0]	2008	100
array[1]	2012	101
array[2]	2016	102

Pointer Arithmetic

- Addition / Subtraction
  - Two pointer variables may not be added together (it does not make any logical sense).

    char *p1, *p2 ;

    p1 = p1 + p2 ; **/* illegal operation */**



```
int *p1,*p2; ;
int array[3] = { 100, 101, 109 } ;
p1 = array ;
p2 = &array[2] ;
int count = p2 - p1;  //2
int count2 = *p2 - *p1;  //9
```

Thank You.