Arrays in C

Rab Nawaz Jadoon

Assistant Professor

COMSATS IIT, Abbottabad

Pakistan

Department of Computer Science

DCS

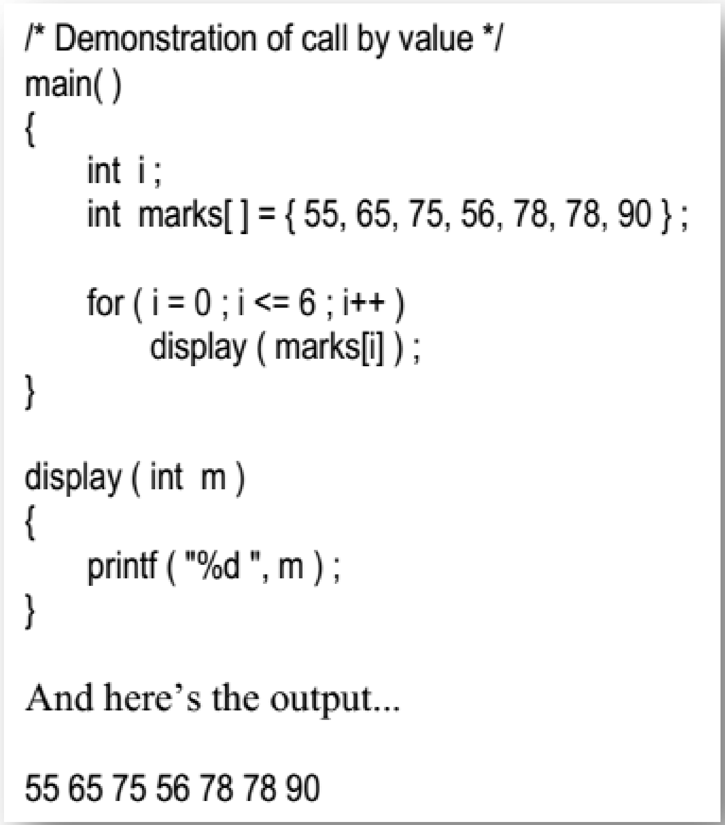
COMSATS Institute of Information Technology

Introduction to Computer Programming (ICP)

Passing Array elements to function

- Array elements can be passed to a function by calling the function by value, or by reference.
  - In the call by value we pass values of array elements to the function, whereas
  - In the call by reference we pass addresses of array elements to the function.

Demonstration



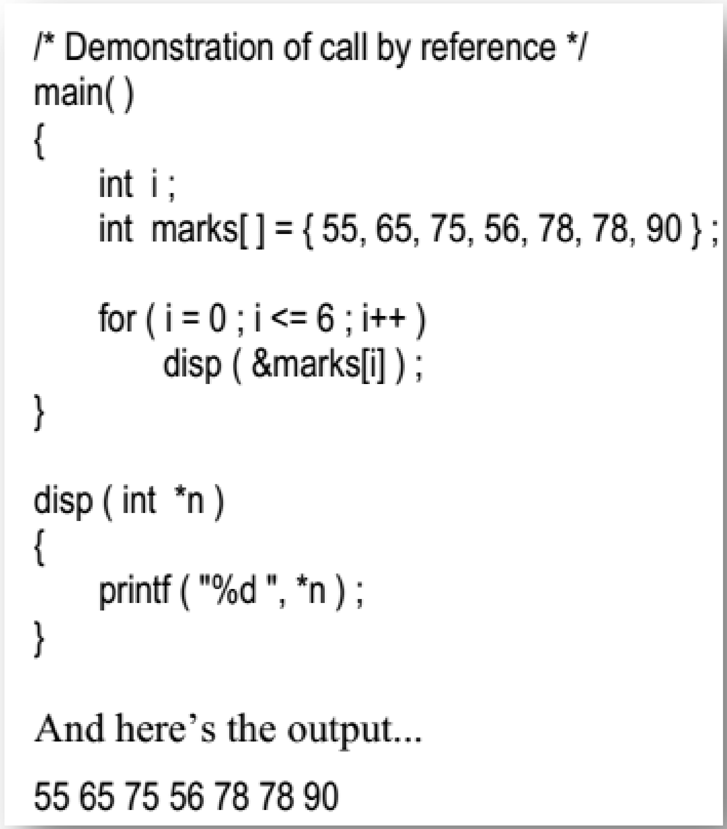
```
/* Demonstration of call by value */
main( )
{
      int  i ;
      int  marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;

      for ( i = 0 ; i <= 6 ; i++ )
            display ( marks[i] ) ;
}

display ( int  m )
{
      printf ( "%d ", m ) ;
}
```

And here's the output...

```
55 65 75 56 78 78 90
```



```
/* Demonstration of call by reference */
main( )
{
      int  i ;
      int  marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;

      for ( i = 0 ; i <= 6 ; i++ )
            disp ( &marks[i] ) ;
}

disp ( int  *n )
{
      printf ( "%d ", *n ) ;
}
```

And here's the output...

```
55 65 75 56 78 78 90
```

Pointers and arrays

```
main( )
{
      int  i = 3, *x ;
      float  j = 1.5, *y ;
      char  k = 'c', *z ;

      printf ( "\nValue of i = %d", i ) ;
      printf ( "\nValue of j = %f", j ) ;
      printf ( "\nValue of k = %c", k ) ;
      x = &i ;
      y = &j ;
      z = &k ;
      printf ( "\nOriginal address in x = %u", x ) ;
      printf ( "\nOriginal address in y = %u", y ) ;
      printf ( "\nOriginal address in z = %u", z ) ;
      x++ ;
      y++ ;
      z++ ;
      printf ( "\nNew address in x = %u", x ) ;
      printf ( "\nNew address in y = %u", y ) ;
      printf ( "\nNew address in z = %u", z ) ;
}
```

```
Value of i = 3
Value of j = 1.500000
Value of k = c
Original address in x = 65524
Original address in y = 65520
Original address in z = 65519
New address in x = 65526
New address in y = 65524
New address in z = 65520
```

Observe the last three lines of the output.
65526 is original value in x plus 2, 65524 is original value in y plus 4, and 65520 is original value in z plus 1.
This so happens because every time a pointer is incremented it points to the immediately next location of its type.
That is why, when the integer pointer x is incremented, it points to an address two locations after the current location, since an int is always 2 bytes long.
Similarly, y points to an address 4 locations after the current location and z points 1 location after the current location.

This is a very important result and can be effectively used while **passing the entire array to a function**.

Pointers (increment & Decrement)

- The way a pointer can be incremented, it can be decremented as well, to point to earlier locations.
  - Thus, the following operations can be performed on a pointer:

Addition of a number to a pointer. For example,

```
int  i = 4, *j, *k ;
j = &i ;
j = j + 1 ;
j = j + 9 ;
k = j + 3 ;
```

Subtraction of a number from a pointer. For example,

```
int i = 4, *j, *k ;
j = &i ;
j = j - 2 ;
j = j - 5 ;
k = j - 6 ;
```

Subtraction of one pointer from another.

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
 int *i, *j ;
 i = &arr[1] ;
 j = &arr[5] ;
 for(int k=0;k<7;k++)
 {
 printf ("\n Element[%d]--> %d Address %u", k,arr[k], &arr[k]);
 }
 printf ( "\n\n%u - %u is %d  and %u - %u is %d", j, i, j - i, *j, *i,
*j - *i ) ;
 getche();
}
```

What would be the output???

Output

```
Element[0]--> 10 Address 1638200
Element[1]--> 20 Address 1638204
Element[2]--> 30 Address 1638208
Element[3]--> 45 Address 1638212
Element[4]--> 67 Address 1638216
Element[5]--> 56 Address 1638220
Element[6]--> 74 Address 1638224

1638220 - 1638204 is 4  and 56 - 20 is 36
```

Comparison of two pointers veriable

- Pointer variables can be compared provided both variables point to objects of the same data type.
  - Such comparisons can be useful when both pointer variables point to elements of the same array.
  - The comparison can test for either equality or inequality.

Program (Pointers Comparison)

```
main( )
{
 int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
 int *j, *k ;
 j = &arr [ 4 ] ;
 k = ( arr + 4 ) ; //assign the address of arr[4] to k
 if ( j == k )
 printf("The two pointers point to the same location");
 else
 printf("The two pointers do not point to the same
location”);
}
```

What would be the output???

A word of Caution

- Do not attempt the following operations on pointers… they would never work out.
  - Addition of two pointers
  - Multiplication of a pointer with a constant
  - Division of a pointer with a constant

Important points

- Now we will try to correlate the following two facts, which we have learnt before:
  - Array elements are always stored in contiguous memory locations.
  - A pointer when incremented always points to an immediately next location of its type.

Simple program

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
 int i, *j ;
 j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
  printf ( "\naddress = %u ", j ) ;
  printf ( "element = %d", *j ) ;
  j++ ; /* increment pointer to point to nextlocation */
 }
  getche();
}
```

OUTPUT

```
address = 1638204 element = 24
address = 1638208 element = 34
address = 1638212 element = 12
address = 1638216 element = 44
address = 1638220 element = 56
address = 1638224 element = 17
```

Passing an Entire Array to a Function

- Let us now see how to pass an entire array to a function rather than its individual elements. Consider the following example:

```
/* Demonstration of passing an entire array to a
function */
#include<stdio.h>
#include<conio.h>
void display(int *, int);
main( )
{
 int num[ ] = { 24, 34, 12, 44, 56, 17 };
 display ( &num[0], 6 );
}
void display ( int *j, int n )
{
int i;
for ( i = 0 ; i <= n - 1 ; i++ )
{
printf ( "\nelement = %d", *j );
j++; /* increment pointer to point to next element */
}
 getche();
}
```

```
element = 24
element = 34
element = 12
element = 44
element = 56
element = 17
```

Accessing Array Elements in different ways

```
/* Accessing Array Elements in different ways */
#include<stdio.h>
#include<conio.h>
main( )
{
 int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
 int i ;
 for ( i = 0 ; i <= 5 ; i++ )
{
  printf ( "\nAddress = %u ", &num[i] ) ;
  printf ( "Element = %d %d ", num[i], *( num + i ) )
;
  printf ( "%d %d", *( i + num ), i[num] ) ;
}
  getche();
}
```

What would be the output???

```
Address = 1638204  Element = 24 24 24 24
Address = 1638208  Element = 34 34 34 34
Address = 1638212  Element = 12 12 12 12
Address = 1638216  Element = 44 44 44 44
Address = 1638220  Element = 56 56 56 56
Address = 1638224  Element = 17 17 17 17
```

Thank You!