Array in C

SDC OSW 3541

Department of Computer Science & Engineering
ITER, Siksha 'O' Anusandhan Deemed To Be University
Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

Book(s)

Text Book(s)

Jeri R Hanly, & Elliot B. Koffman

Problem Solving & Program Design in C

Seventh Edition, Pearson Education

Robert Love

LINUX Systems Programming

Second Edition, SPD, O' REILLY

Talk Flow

Array

- To solve many programming problems, it is more efficient to group data items together in main memory than to allocate an individual memory cell for each variable.
- An array is an arrangement of clubbing multiple entities of similar type into a larger group.
- Technically, an array is a collection of two or more adjacent memory cells.
- The memory cells, each of same size can store only same type of data.

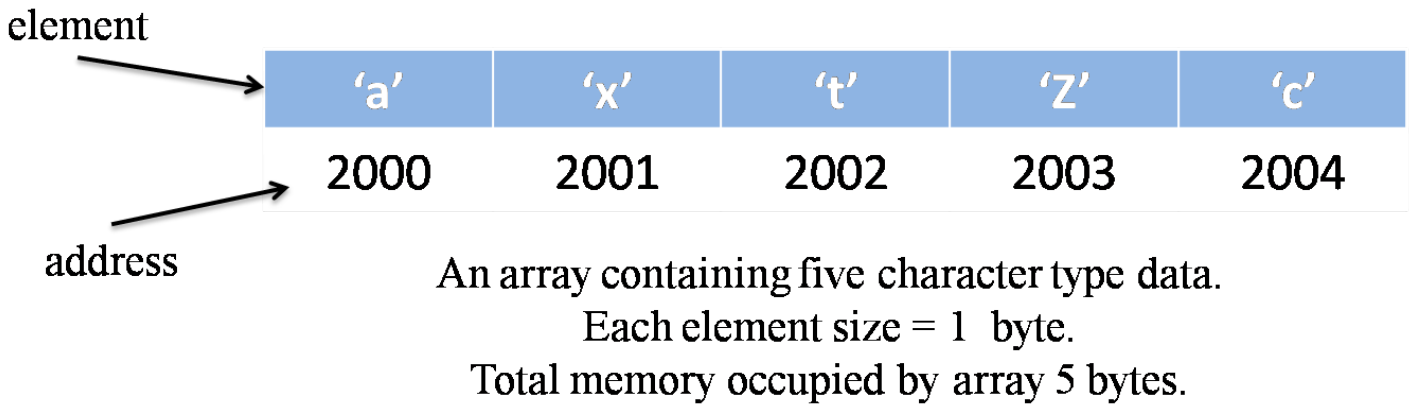

Figure 1: Array: each element containing **char** type data

Array

- Note that, every byte of memory in our main memory has an address.
- While using gcc compiler, an **int** type data takes 4 bytes space.
- Fig 2 shows an array which stores **int** type data.
- As every bytes has an address, and an **int** occupies 4 bytes memory, address of an next elements differs four positions from the previous element.

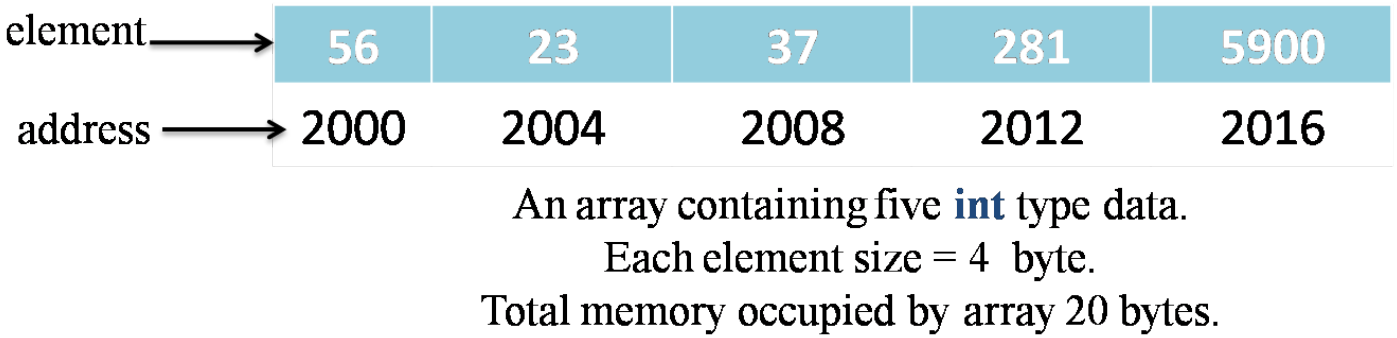

Figure 2: Array: each element containing **int** type data

Array Declaration

- To set up an array in memory, we must declare the type of data, name of the array and the number of cells/elements associated with it.
- Declare an **int** type array to store 8 **int** type elements as follows:
  int myarr[8];

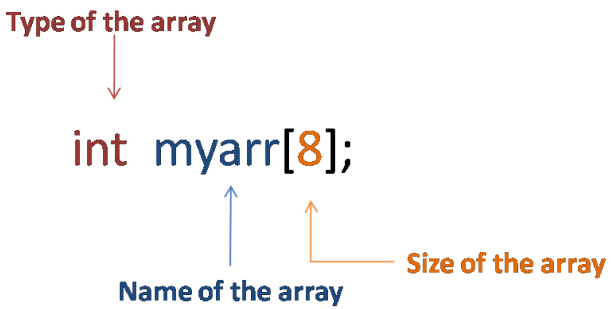

Figure 3: Declaring an array to hold 8 **int** type element

- The above statement instructs the compiler to associate eight memory cells with the name **myarr**; these memory cells will be adjacent to each other in memory. Each element of array **myarr** may contain a single type **int** value, so a total of eight such numbers may be stored and referenced using the array name **myarr**.

Accessing Array Elements

```
int x[8];
```

45	67	2	56	678	90	1432	48
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]

Figure 4: Referring array elements

- **x[0]** may be used to reference the initial or 0^{th} element of the array **x**,
  x[1] the next element,
  x[7] the last element.

- The above statement instructs the compiler to associate eight memory cells with the name **myarr**; these memory cells will be adjacent to each other in memory. Each element of array **myarr** may contain a single type **int** value, so a total of eight such numbers may be stored and referenced using the array name we reference each individual element by specifying the array name and identifying the element desired (for example, element

Manipulating Array

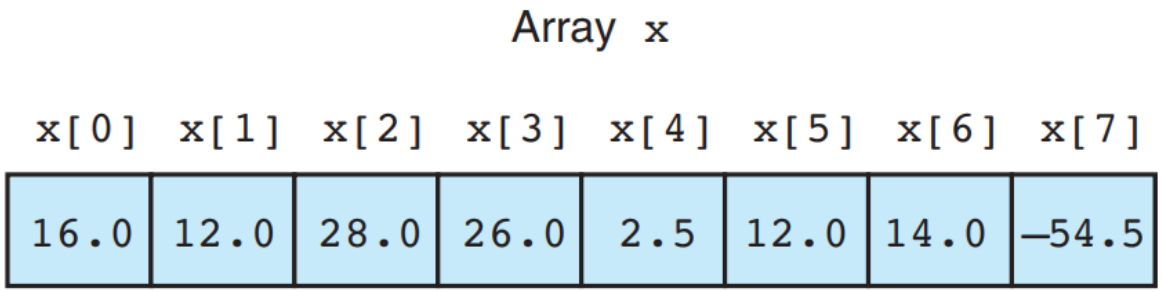

Figure 5: double x[8];

Statement	Explanation
printf("%.1f", x[0]);	Displays the value of x[0], which is 16.0
x[3] = 25.0;	Stores the value 25.0 in x[3]
sum = x[0] + x[1];	Stores the sum of x[0] and x[1], which is 28.0 in the variable sum.
sum += x[2];	Adds x[2] to sum. The new sum is 34.0.
x[3] += 1.0;	Adds 1.0 to x[3]. The new x[3] is 26.0
x[2] = x[0] + x[1];	Stores the sum of x[0] and x[1] in x[2]. The new x[2] is 28.0

Manipulating Array

* **Declare more than one array in a single type declaration.**
  double cactus[5], needle, pins[6];
  int factor[12], n, index;
* **Initializing an array in its declaration.**
  int a[]={5,67,23,45,89,12,34,27,24};
* Note: We can omit the size of an array that is being fully initialized since the size can be deduced from the initialization list.
* Alternative and better way to specify the array size

```
#define a_size 5
int main()
{
 int a[a_size];
 char vowels[] = {'A', 'E', 'I', 'O', 'U'};
}
```

Accessing Array Element using loop

```
#include <stdio.h>
#define a_size 5
int main()
{
int square[a_size], i,sum;
for (i = 0; i < a_size; i++)
    square[i] = i * i;
for (i = 0; i < a_size; i++)
    printf("square[%d] has address %d and value  %d \n",i,
        &square[i];square[i]);
sum=0;
for (i = 0; i < a_size; i++)
    sum=sum+square[i];

printf("Sum of all the data in square[] is %d \n",sum);
}
```

Note the address of each element. As an int variable occupies 4 bytes, each element address differ by a value 4. You may change array type and check for other data types.

Accessing Array Element using loop

```
/*Accept array data from standard input*/
#include <stdio.h>
#define a_size 5
int main()
{
int arr[a_size], i,sum;
printf("Enter %d numbers separated by blanks or <return>s\n
    >", a_size);
for (i = 0; i < a_size; i++)
    scanf("%d", &arr[i]);

sum=0;
for (i = 0; i < a_size; i++)
    sum=sum+arr[i];

printf("Sum of all the data in arr[] is %d \n",sum);
}
```

Accessing Array Element using loop

```
 //randarray.c
/*Set array data using as random numbers*/
#include <stdio.h>
#include <math.h>
#define a_size 5
#define randmax 100
int main()
{
int arr[a_size], i,sum;
for (i = 0; i < a_size; i++)
   arr[i]=rand()%randmax; //get a number berween 0 to
       randmax
for (i = 0; i < a_size; i++)
    printf("arr[%d] has value %d \n",i,arr[i]);
}
```

Compile above program as:

gcc randarray.c -o ra.out -lm

Passing Array Element to function

```
#define a_size 5
void printmyarr_ele(int arr_ele, int index);
int main()
{
int arr[a_size], i,sum;
printf("Enter %d numbers separated by blanks or <return>s\n
    >", a_size);
for (i = 0; i < a_size; i++)
    scanf("%d", &arr[i]);
    printmyarr_ele(arr[3],3);
    /*An array element is just passed as normal variable or
         value*/
}
void printmyarr_ele(int val, int index){
printf("At position %d you entered %d \n",index,val);
}
```

Passing address of array element to function

```
#define a_size 5
void printmyarr(int *ar_ele_ad1, int ar_ele); // Prototype
    of a function that takes an array input
int main()
{
int arr[a_size], i;
for (i = 0; i < a_size; i++)
     arr[i] = i * i;  //Stores square of index as value
    printmyarr(&arr[1], arr[2]);
    /* We use & with arr[1], to pass address of arr[1]*/
    /* When we pass arr[2],  it's value i.e. 4 is passed*/
}
void printmyarr(int *recv_addr, int recv_value){
    printf("First parameter is an address %p which has
        value %d \n",recv_addr,*recv_addr);
    printf("The second parameter is %d \n",recv_value);
}
```

Passing address of array element to function

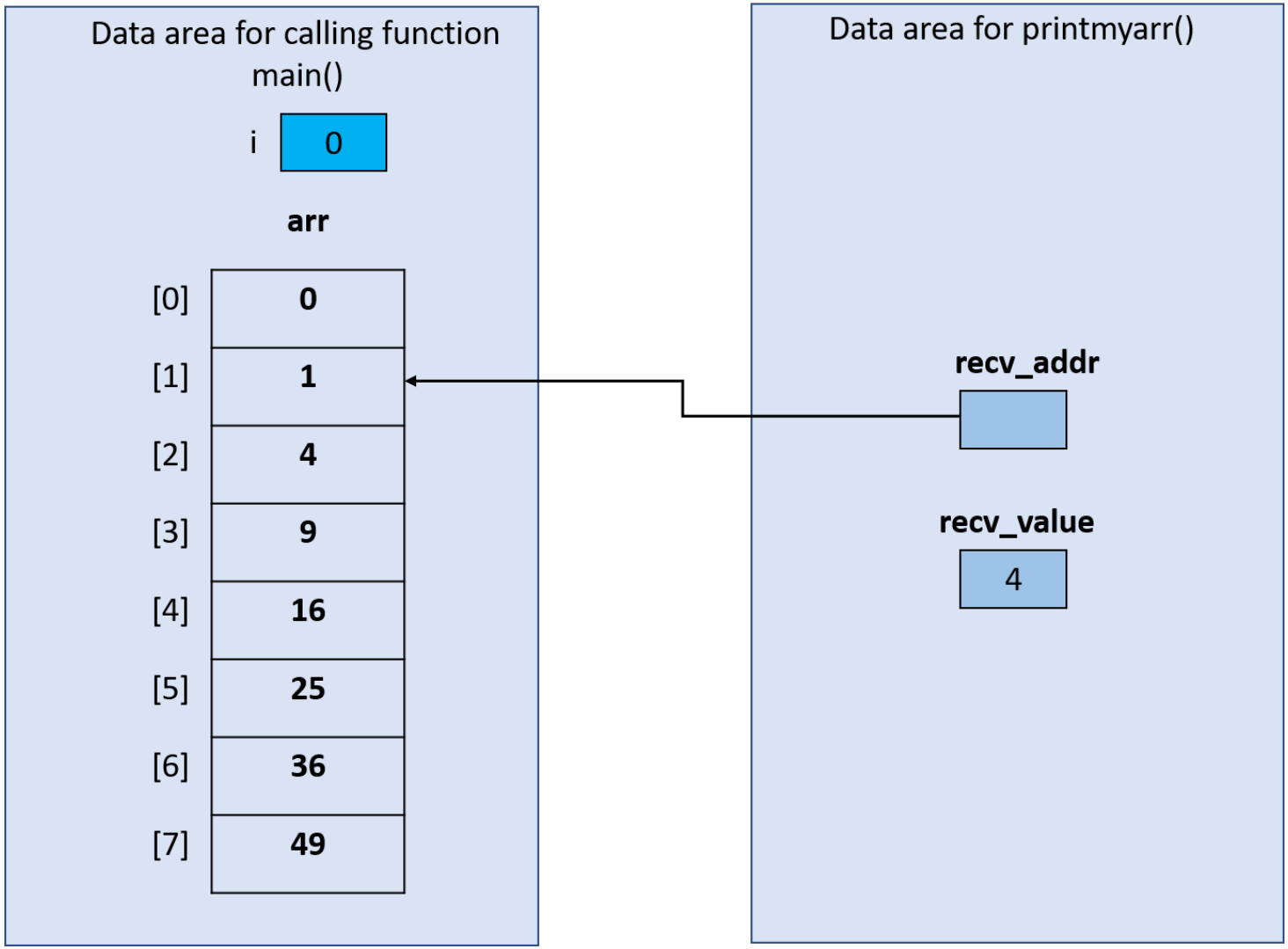

Figure 6: Call by value and call by reference

Passing array to function

```
#define a_size 5
void printmyarr(int *a); // Prototype of a function that
    takes an array input
int main()
{
int arr[a_size], i;
for (i = 0; i < a_size; i++)
    arr[i] = i * i; //Stores square of index as value
    printmyarr(arr);
    /* We do not use & with arr, as arr is address/pointer
        type*/
}
void printmyarr(int *recv_arr){
int j;
for (j = 0; j < a_size; j++)
    printf("At position %d value is %d \n",j,recv_arr[j]);
}
```

Passing array to function

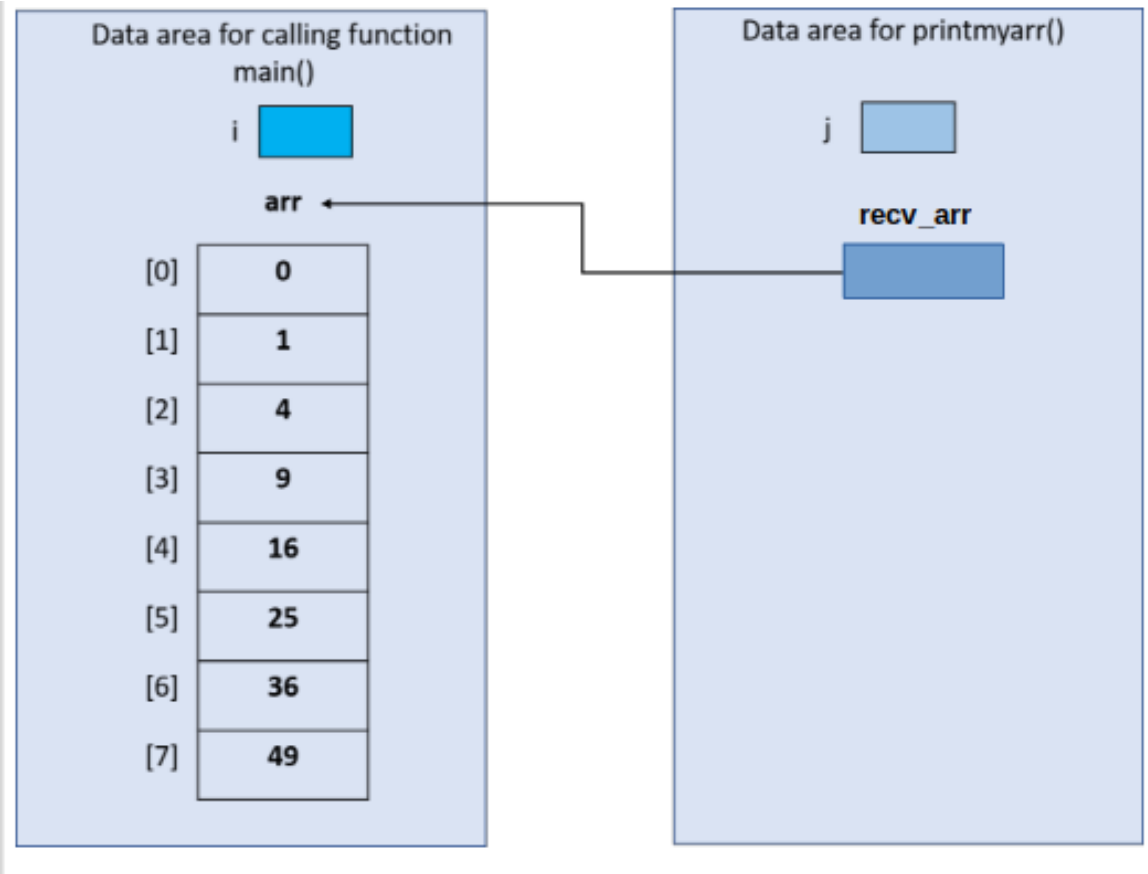

Figure 7: Passing an array address

Passing array to function

```
#define a_size 5
void printmyarr(int *a); void plus_two(int *a);
int main(){
int arr[a_size], i;
for (i = 0; i < a_size; i++)
            arr[i] = rand()%100;
printmyarr(arr); plus_two(arr);
//Since, both arr and recv_arr in plus_two() refers to same
     array (memory), any changes to recv_arr is permanent
    and affects arr.
    printmyarr(arr);
}
void printmyarr(int *printar){
printf("\n\n");
for (int j = 0; j < a_size; j++)
    printf("> At position %d value is %d \n",j,printar[j]);
}
void plus_two(int *recv_arr){
for (int j = 0; j< a_size; j++)
            recv_arr[j]+=2;     }//end of plus_two
```

Multi-dimensional Array

char tictac[3][3];

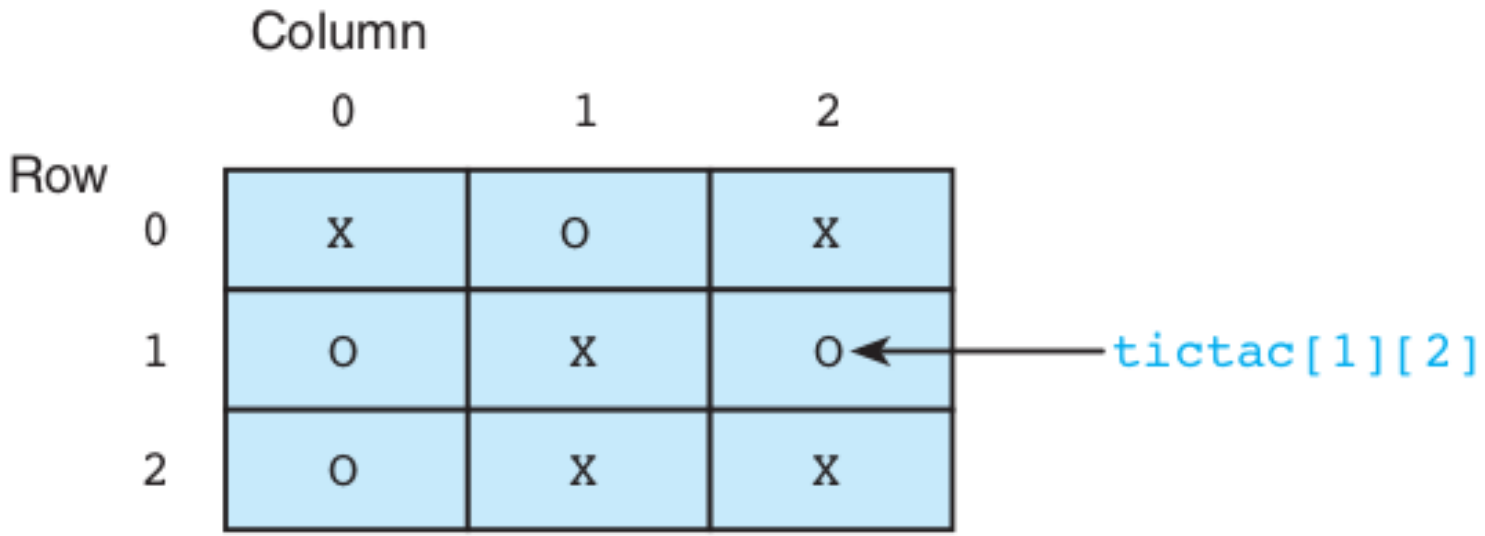

Figure 8: A 2D Array

Multi-dimensional Array

Initializing an 2D Array

```
char tictac[3][3] = { {' ', ' ', ' '}, {' ', ' ', ' '},{' '
    , ' ', ' '} };
```

Initializing an 2D Array

```
void initialize(char ttt_brd[3][3]) /* input - tic-tac-toe
    board*/
{
int r, c,
/* Resets ans to zero if a blank is found */
for (r = 0; r < 3; ++r)
for (c = 0; c < 3; ++c)
        ttt_brd[r][c] == ' ';

}
```

Multi-dimensional Array

A function to check if the 2D array tictac is filled.

```
int isfilled(char ttt_brd[3][3]) /* input - tic-tac-toe
    board*/
{
int r, c,
int ans = 1;
/* Resets ans to zero if a blank is found */
for (r = 0; r < 3; ++r)
for (c = 0; c < 3; ++c)
        if (ttt_brd[r][c] == ' ')
            ans = 0;
return (ans);
}
```

Common Programming Errors

```
int arr[100];
```

- A subscript-range error occurs when arr is used with a subscript that has a value less than 0 or greater than 99. You get a "Segmentation Fault" error with gcc.
- While using array in a loop, verify that the subscript is in range for both the initial and the final values of the loop control variable.
- When using arrays as arguments to functions, be careful not to apply the address of operator (&) to the array name even if the array is an output argument.

```
int *z;
```

- could represent a single integer output parameter or an integer array parameter. Comment your own prototypes carefully and use the alternate declaration form

```
int z[];
```

Sample Questions

? Identify error.

```
int x[8], i;
for (i = 0; i <= 8; ++i)
x[i] = i;
```

? Write a program segment to display the sum of the values in each row of a 5×3 type double array named table. How many row sums will be displayed? How many elements are included in each sum?

? Answer Above Question for the column sums.

THANK YOU