

CAN WE SPECIFY THE LENGTH OF THE ARRAY AS 5.679?

OR

CAN IT BE OF TYPE OTHER THAN INTEGER?



The length of an array can be specified by any **positive integer** constant expression.

```
int arr[5];
```

```
int arr[5+5];
```

```
int arr[5*3];
```

```
int arr[-5];
```

```
int a;  
int arr[a = 21/3];
```

BEST PRACTICE

Specifying the length of an array using **macro** is considered to be an excellent practice

```
#define N 10  
int arr[N];
```

This is macro,
remember?



BUT WHY?

WE'LL SEE WHY IN THE NEXT LECTURE.



INITIALIZING 1D ARRAY

METHOD 1:

```
arr[5] = {1, 2, 5, 67, 32};
```

METHOD 2:

```
arr[] = {1, 2, 5, 67, 32};
```

METHOD 3:

```
int arr[5];
```

```
arr[0] = 1;
```

```
arr[1] = 2;
```

```
arr[2] = 5;
```

```
arr[3] = 67;
```

```
arr[4] = 32;
```

METHOD 4:

```
int arr[5];
```

```
for(i=0; i<5; i++){  
    scanf("%d", &arr[i]);  
}
```


Q

What if number of elements are lesser than the length specified?

```
int arr[10] = {45, 6, 2, 78, 5, 6};
```

A

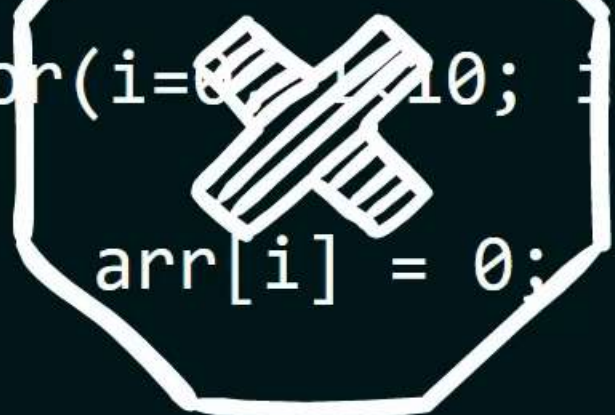
The remaining locations of the array are filled by value 0.

```
int arr[10] = {45, 6, 2, 78, 5, 6, 0, 0, 0, 0};
```



A SMALL TIP

```
int arr[10];  
for(i=0; i<10; i++)  
{  
    arr[i] = 0;  
}
```



```
int arr[10] = {0};
```



BUT...



Why not `int arr[10] = {};` ?

Because, this is illegal.

You must have to specify at least 1 element. It cannot be completely empty.
and it is also illegal to add more elements than the length of an array.

For example:

`int arr[5] = {`  `2, 3, 4, 5, 6};`

The diagram shows the opening curly brace of the array initialization being crossed out with a large 'X' inside an octagon, indicating that the provided list of elements is invalid because it contains more elements than the array's declared size of 5.

SUMMARY



If the number of elements are lesser than the length of the array than the rest of the locations are automatically filled by value 0.



Easy way to initialize an array with all zeros is given by:

```
int arr[10] = {0};
```



At the time of initialization, never leave these flower brackets {} empty and also never exceed the limit of number of elements specified by the length of an array.

```
int arr[5] = {};
```



```
int arr[5] = {1, 2, 3, 4, 5, 6};
```

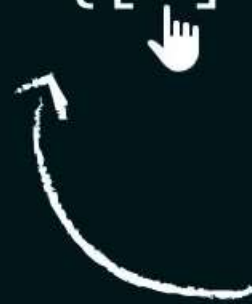



```
int arr[10] = {1, 0, 0, 0, 0, 2, 3, 0, 0, 0};
```

We want:

- 1 in position 0
- 2 in position 5
- 3 in position 6

```
int arr[10] = {[0] = 1, [5] = 2, [6] = 3};
```



This way of initialization is called **designated initialization**.

And each number in the square brackets is said to be a **designator**.

ADVANTAGES

1. No need to bother about the entries containing zeros.

```
int a[15] = {1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

```
int a[15] = {[0] = 1, [5] = 2};
```



ADVANTAGES

2. No need to bother about the order at all.

```
int a[15] = {[0] = 1, [5] = 2};
```

```
int a[15] = {[5] = 2, [0] = 1};
```

Both are same



BE CAREFUL!

If the length of an array is 'n', then each designator must be between 0 and n-1.



```
int a[5] = {[0] = 4, [4] = 78};
```



```
int a[5] = {[0] = 4, [5] = 78};
```



WHAT IF I WON'T MENTION THE LENGTH?

- ★ Designators could be any non-negative integer.
- ★ Compiler will deduce the length of the array from the largest designator in the list.

```
int a[] = {[5] = 90, [20] = 4, [1] = 45, [49] = 78};
```

Because of this designator,
maximum length of this
array would be 50.

FINALLY, NO ONE CAN STOP YOU FROM DOING THIS:

```
int a[] = {1, 7, 5, [5] = 90, 6, [8] = 4};
```

≡

```
int a[] = {1, 7, 5, 0, 0, 90, 6, 0, 4};
```



But, if there is a clash, then designated initializer will win.



```
int a[] = {1, 2, 3, [2] = 4, [6] = 45};
```

≡

```
int a[] = {1, 2, 4, 0, 0, 0, 45};
```

Q. Write a program to print the following numbers in reverse order:

34 56 54 32 67 89 90 32 21

Assume that all these numbers are stored in an array.




```
int main() {  
    int a[9] = {34, 56, 54, 32, 67, 89, 90, 32, 21};  
    int i;  
  
    //Original order  
    for(i=0; i<9; i++) {  
        printf("%d ", a[i]);  
    }  
  
    printf("\n");  
  
    //Reverse order  
    for(i=8; i>=0; i--) {  
        printf("%d ", a[i]);  
    }  
    return 0;  
}
```

Q. Write a program to check whether any of the digits in a number appears more than once:

Example:

Input: 67827

Output: Yes

Part 1

```
int seen[10] = {0};
```

0	0	1	1	0
0	1	2	3	4

Input: 23

Part 2

```
while(N>0) {  
    rem = N%10;  
    if(seen[rem] == 1)  
        break;  
    seen[rem] = 1;  
    N = N/10;  
}
```

Input: 23



$23 \% 10 = 3$

$2 \% 10 = 2$

```
while(N>0) {  
    rem = N%10;  
    if(seen[rem] == 1)  
        break;  
    seen[rem] = 1;  
    N = N/10;  
}
```



0	0	1	1	0
0	1	2	3	4

Part 3

Two cases:

- ★ When $N > 0$ and break
- ★ When $N == 0$

Example:


Input: 1232

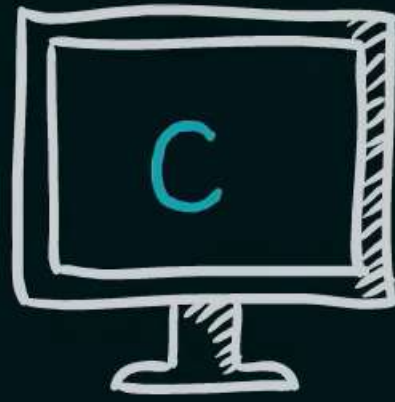
$1232 \% 10 = 2$ $123 \% 10 = 3$

$12 \% 10 = 2$

Part 3

```
if(N > 0)
    printf("Yes");
else
    printf("No");
```





PROGRAMMING AND **DATA** STRUCTURES



Using sizeof operator with
arrays

```
sizeof(name_of_arr)/sizeof(name_of_arr[0])
```

`sizeof(name_of_arr)`

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
sizeof(a);
```

There are total 10 integers

Assume that each integer requires 4 bytes

`sizeof(a)` = 4 x 10 = 40 bytes



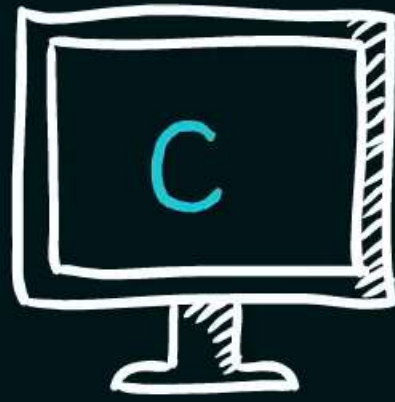
`sizeof(name_of_arr[0])`

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
sizeof(a[0]);
```

```
sizeof(a[0]) = 4 bytes
```





PROGRAMMING AND **DATA** STRUCTURES



Introduction to multi dimensional
Arrays

DEFINITION AND SYNTAX

Multidimensional arrays can be defined as an **array of arrays**

General form of declaring N-dimensional array is as follows:

```
data_type name_of_array[size1][size2]...[sizeN];
```

For example:

```
int a[3][4];    //Two Dimensional Array
```

```
int a[3][4][6]; //Three Dimensional Array
```



We will discuss them in details in the upcoming lectures.

SIZE CALCULATION

Size of multidimensional array can be calculated by multiplying the size of all the dimensions.

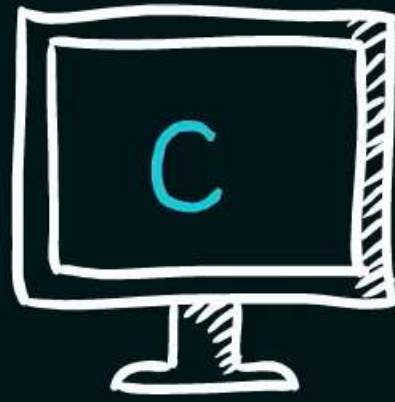
For example:

$$\begin{aligned}\text{size of } a[10][20] &= 10 \times 20 = 200 \\ &= 200 \times 4 = 800 \text{ bytes}\end{aligned}$$

We can store upto 200 elements in this array

$$\begin{aligned}\text{size of } a[4][10][20] &= 4 \times 10 \times 20 = 800 \\ &= 800 \times 4 = 3200 \text{ bytes}\end{aligned}$$

We can store upto 800 elements in this array



PROGRAMMING AND **DATA** STRUCTURES



Introduction to two dimensional
Arrays

VISUALIZING TWO DIMENSIONAL ARRAY

Recall that a multidimensional array is an array of arrays

How to declare a 1D array?

```
int arr[5];
```



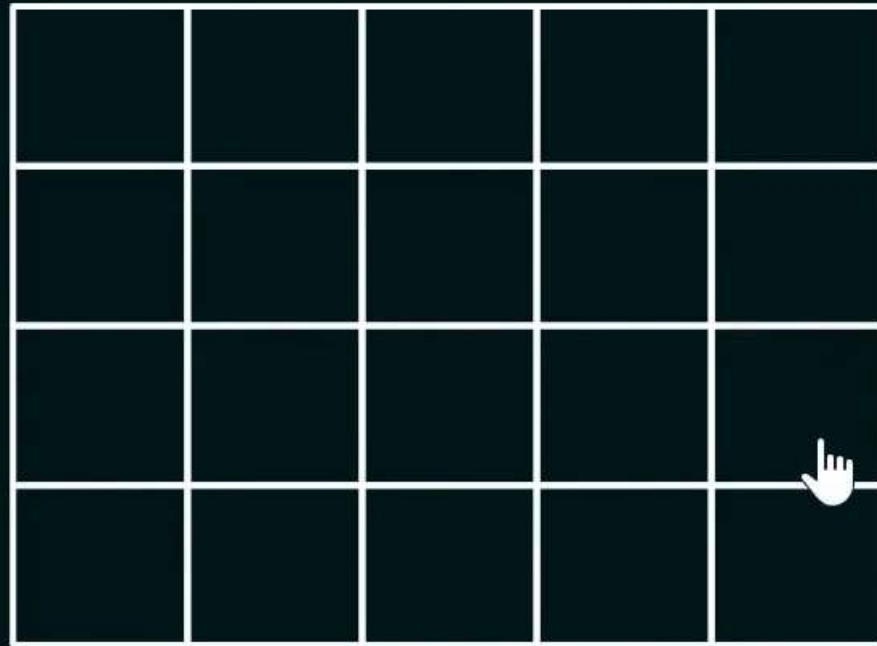
VISUALIZING TWO DIMENSIONAL ARRAY

```
int arr[4][5];
```



VISUALIZING TWO DIMENSIONAL ARRAY

```
int arr[4][5];
```



4 x 5

VISUALIZING TWO DIMENSIONAL ARRAY

```
int arr[4][5];
```

rows

columns


size of arr[4][5] = $4 \times 5 = 20$ elements

= $20 \times 4 = 80$ bytes

HOW TO INITIALIZE TWO DIMENSIONAL ARRAY?

Method 1:

```
int a[2][3] = {1, 2, 3, 4, 5, 6}; //sometimes confusing
```



	0	1	2
0	1	2	3
1	4	5	6


HOW TO INITIALIZE TWO DIMENSIONAL ARRAY?

Method 2:

Row 1

Row 2

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
```



The diagram illustrates the memory layout of the 2D array. It shows a grid where the first row contains the values 1, 2, and 3, and the second row contains 4, 5, and 6. Above the grid, the column indices 0, 1, and 2 are labeled. To the left of the grid, the row indices 0 and 1 are labeled. Two hand icons point to the first and second rows of the grid, corresponding to the row indices 0 and 1.

	0	1	2
0	1	2	3
1	4	5	6

HOW TO ACCESS 2D ARRAY ELEMENTS?

Using row index and column index.

For example:

We can access element stored in 1st row and 2nd column of below array

	0	1	2
0	1	2	3
1	4	5	6

Using:

`a[0][1]`

HOW TO PRINT 2D ARRAY ELEMENTS?

1D array elements can be printed using single for loop

```
int a[5] = {1, 2, 3, 4, 5};
```

```
for(i=0; i<5; i++)  
{  
    printf("%d ", a[i]);  
}
```

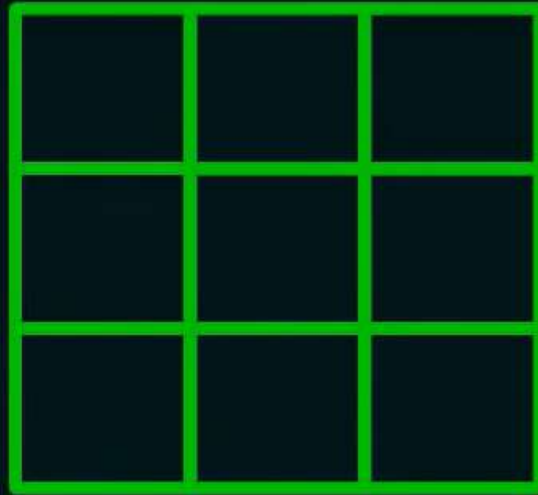
2D array elements can be printed using two nested for loops.

```
int a[2][3] = {{1, 2, 3},  
               {4, 5, 6}};
```

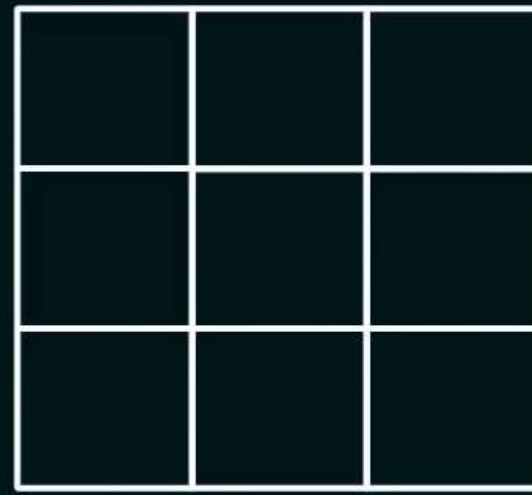
```
for(i=0; i<2; i++)  
{  
    for(j=0; j<3; j++)  
    {  
        printf("%d ", a[i][j]);  
    }  
}
```

ACCESSING THE 3D ARRAY ELEMENTS

```
int arr[2][3][3];
```



3 x 3



3 x 3

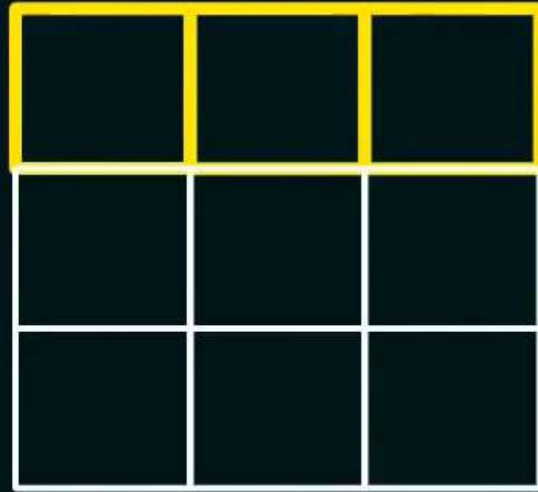
`arr[0][0][2]`



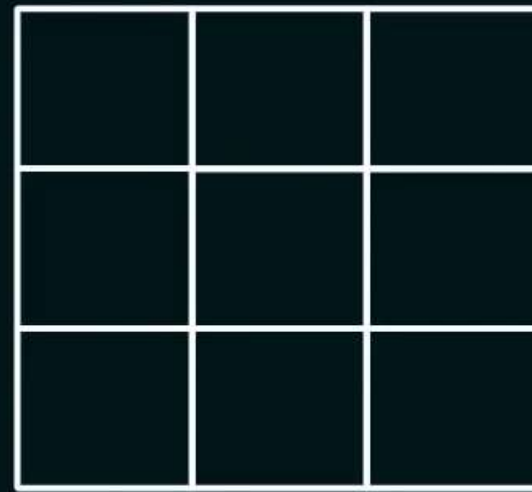
First 2D array

ACCESSING THE 3D ARRAY ELEMENTS

```
int arr[2][3][3];
```



3 x 3



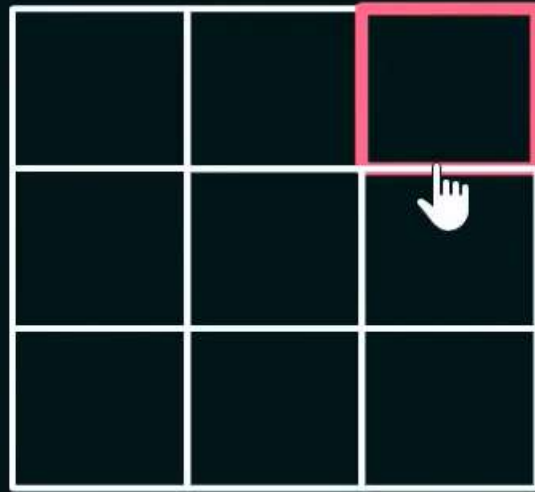
3 x 3

```
arr[0][0][2]
```

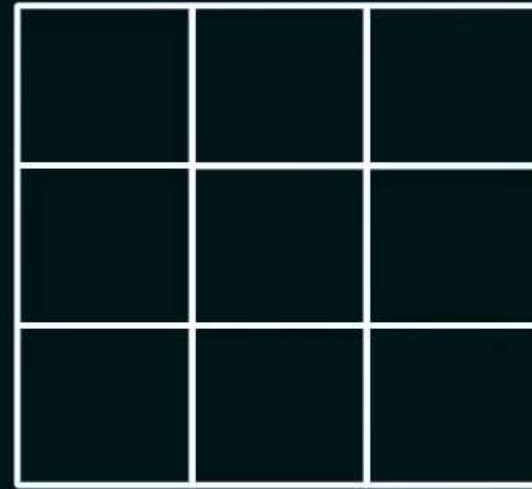
First row

ACCESSING THE 3D ARRAY ELEMENTS

```
int arr[2][3][3];
```



3 x 3



3 x 3

```
arr[0][0][2]
```

Third column

INITIALIZING 3D ARRAY

Method 1:

```
int a[2][2][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

1	2	3
4	5	6

2 x 3

7	8	9
10	11	12

2 x 3

INITIALIZING 3D ARRAY

Method 2: Better method

```
int a[2][2][3] = {  
    {{1, 2, 3}, {4, 5, 6}},  
    {{7, 8, 9}, {10, 11, 12}}  
};
```

1	2	3
4	5	6

2 x 3

7	8	9
10	11	12

2 x 3



INITIALIZING 3D ARRAY

Method 2: Better method

```
int a[2][2][3] = {  
    {{1, 2, 3}, {4, 5, 6}},  
    {{7, 8, 9}, {10, 11, 12}}  
};
```

1	2	3
4	5	6

2 x 3

7	8	9
10	11	12


2 x 3



PRINTING THE ELEMENTS OF 3D ARRAY

3D array elements can be printed using **three nested for loops**.

```
int a[2][2][3] = { {{1, 2, 3}, {4, 5, 6}},  
                  {{7, 8, 9}, {10, 11, 12}} };
```

```
      
    for(i=0; i<2; i++)  
    {  
        for(j=0; j<2; j++)  
        {  
            for(k=0; k<3; k++)  
            {  
                printf("%d ", a[i][j][k]);  
            }  
        }  
    }  
}
```



PROGRAMMING AND **DATA** STRUCTURES



Row sum and column sum

```
int main() {
    int a[5][5] = {
        {8, 3, 9, 0, 10},
        {3, 5, 17, 1, 1},
        {2, 8, 6, 23, 1},
        {15, 7, 3, 2, 9},
        {6, 14, 2, 6, 0}
    };

    int i,j;
    int sum=0;
```

```
//Row sum
printf("Row total: ");

for(i=0; i<5; i++)
{
    for(j=0; j<5; j++)
    {
        sum += a[i][j];
    }
    printf(" %d", sum);
    sum=0;
}
```

```
int main() {  
    int a[5][5] = {  
        {8, 3, 9, 0, 10},  
        {3, 5, 17, 1, 1},  
        {2, 8, 6, 23, 1},  
        {15, 7, 3, 2, 9},  
        {6, 14, 2, 6, 0}  
    };  
  
    int i,j;  
    int sum=0;
```

```
    //Column sum  
    printf("\nColumn total: ");  
  
    for(j=0; j<5; j++)  
    {  
        for(i=0; i<5; i++)  
        {  
            sum += a[i][j];  
        }  
        printf(" %d", sum);  
        sum=0;  
    }
```




PROGRAMMING AND **DATA** STRUCTURES



Matrix Multiplication program
Part 1

IMPORTANT POINT

- ★ Also, size of the resultant matrix depends on #rows of 1st matrix and #columns of 2nd matrix

1	2	3
1	2	1
3	1	2

3 x 3

1	2	3
1	2	1
3	1	2

3 x 3

12	9	11
6	7	7
10	10	14

3 x 3

WE WANT SOMETHING LIKE THIS:

Enter the rows and columns of matrix a: 3 3

Enter the elements of matrix a:

1 2 3

1 2 1

3 1 2

Enter the rows and columns of matrix b: 3 3

Enter the elements of matrix b:

1 2 3

1 2 1

3 1 2

Resultant matrix:

12 9 11

6 7 7

10 10 14

PART 1:

Enter the rows and columns of matrix a: 3 3

```
int a[MAX][MAX];  
int arows, acolumns;  
  
printf("Enter the rows and columns of the matrix a: ");  
scanf("%d %d", &arows, &acolumns);
```



PART 2:

Enter the elements of matrix a:

1	2	3
1	2	1
3	1	2

```
printf("Enter the elements of matrix a:\n");
```

```
for(i=0; i<arows; i++)  
{  
    for(j=0; j<acolumns; j++)  
    {  
        scanf("%d", &a[i][j]);  
    }  
}
```

PART 3: RESULTANT MATRIX

Resultant matrix:

12 9 11

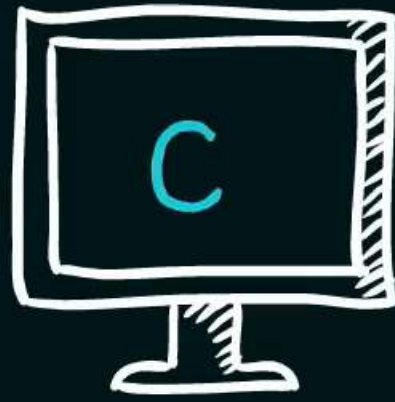
6 7 7

10 10 14

```
int product[MAX][MAX];
int sum=0;
for(i=0; i<arows; i++)
{
    for(j=0; j<bcolums; j++)
    {
        for(k=0; k<brows; k++)
        {
            sum += a[i][k] * b[k][j];
        }
        product[i][j] = sum;
        sum=0;
    }
}
```

1	2	3
1	2	1
3	1	2

1	2	3
1	2	1
3	1	2



PROGRAMMING AND **DATA** STRUCTURES



Constant arrays in C



Either one dimensional or multi-dimensional arrays can be made constant by starting the declaration with the keyword **const**.

For example:

```
const int a[10] = {1,2,3,4,5,6,7,8,9,10};
```

```
a[1] = 45;
```

ERROR!



ADVANTAGE

It assures us that the program will not modify the array which may contain some valuable information.

It also helps the compiler to catch errors by informing that there is no intention to modify this array.