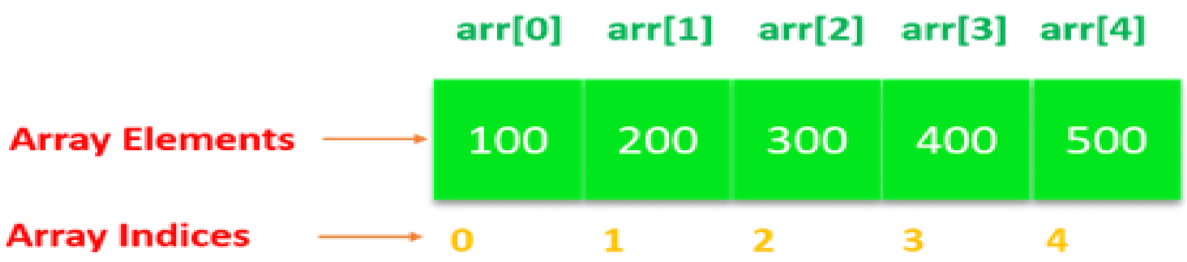
Vivekananda Institute
of Professional Studies

Arrays

Data Structures
Dr. Nivedita Palia

Introduction of Arrays

- An array is a fixed-size sequential collection of elements of same data type in continuous memory location.
- The elements of an array are referenced by an index (also known as subscript).



- Types of Arrays: one dimensional and multidimensional

One- Dimensional Arrays

A list of items can be given one variable name using only one subscript and such a variable is called a one- dimensional array.

Array Declaration: data-type name[size];

- Data type—the kind of values it can store, for example, int, char, float, double.
- Name—to identify the array.
- Size—the maximum number of values that the array can hold.

Declaration Examples:

float height[50]; int group[10]; char name[10];

Storing values in an array

Storing values in an array
- Initialize the elements during declaration
- Input values for the elements from the keyboard
- Assign values to individual elements

Compile Time Initialization

Example :

int number[3] = {4,5,9};

- Here array number of size 3 and will assign 4 to first element(number[0]), 5 is assign with second element(number[1]) and 9 is assign with third element(number[2]).

- If the number of values in the list is less than the number of elements, then only that many elements will be intialized. The remaining elements will be set to zero automatically.

Run Time Initialization

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

```
int marks [5] = {90, 45, 67, 85, 78};
```

[0]	[1]	[2]	[3]	[4]
90	45	67	85	78

```
int marks [5] = {90, 45};
```

[0]	[1]	[2]	[3]	[4]
90	45	0	0	0

Rest of the elements are filled with 0's

```
int marks [] = {90, 45, 72, 81, 63, 54};
```

[0]	[1]	[2]	[3]	[4]	[5]
90	45	72	81	63	54

```
int marks [5] = {0};
```

[0]	[1]	[2]	[3]	[4]
0	0	0	0	0

Figure 3.8 Initialization of array elements

Calculating Length of an Array

Length = upper_bound - lower_bound + 1

Example 3.2 Let Age[5] be an array of integers such that

Age[0] = 2, Age[1] = 5, Age[2] = 3, Age[3] = 1, Age[4] = 7

Show the memory representation of the array and calculate its length.

Solution

The memory representation of the array Age[5] is given as below.

Age[0]	Age[1]	Age[2]	Age[3]	Age[4]
2	5	3	1	7

Length = upper_bound - lower_bound + 1

Here, lower_bound = 0, upper_bound = 4

Therefore, length = 4 - 0 + 1 = 5

Consider the linear arrays AAA(5:50) and B(-5:10). Find the number of elements in each array.

Memory Representation of Linear Array

```
Address of data element, A[k] = BA(A) + w(k - lower_bound)
```

Here, `A` is the array, `k` is the index of the element of which we have to calculate the address, `BA` is the base address of the array `A`, and `w` is the size of one element in memory, for example, size of `int` is 2.

Example 3.1 Given an array `int marks[]={99,67,78,56,88,90,34,85}`, calculate the address of `marks[4]` if the `base address = 1000`.

Solution

99	67	78	56	88	90	34	85
marks[0]	marks[1]	marks[2]	marks[3]	**marks[4]**	marks[5]	marks[6]	marks[7]
1000	1002	1004	1006	**1008**	1010	1012	1014

We know that storing an integer value requires 2 bytes, therefore, its size is 2 bytes.

```
marks[4] = 1000 + 2(4 - 0)
         = 1000 + 2(4) = 1008
```

Q1 Consider a linear array A[5:50]. Suppose BA(A)= 300 and w=4 words per memory cell for array A. Find the address of A[15] and A[55].

Operations on Arrays

1. Traversing an array
2. Inserting an element in an array
3. Searching an element in an array
4. Deleting an element from an array
5. Merging two arrays
6. Sorting an array in ascending or descending order

Traversing Linear Arrays

Accessing and processing each elements of an array exactly once.
Example: Print an array, count the number of elements in an array

```
Step 1: [INITIALIZATION] SET I = lower_bound
Step 2: Repeat Steps 3 to 4 while I <= upper_bound
Step 3:       Apply Process to A[I]
Step 4:       SET  I = I + 1
        [END OF LOOP]
Step 5: EXIT
```

Figure 3.12 Algorithm for array traversal

Program for Array Traversal

```c
// Array Traversal
#include<stdio.h>
int main()
{
    int a[5]={10,20,30,40,50},i;
    for(i=0;i<5;i++)
    printf("%d\t",a[i]);
    return 0;
}
```

```
10      20      30      40      50
--------------------------------
Process exited after 0.09632 seconds with return value 0
Press any key to continue . . .
```

Q1 Write a program to find the sum and average of array elements .
Q2 Write a program to print an array in reverse order

Inserting an element in an array

Insertion at end

```
Step 1: Set upper_bound = upper_bound + 1
Step 2: Set A[upper_bound] = VAL
Step 3: EXIT
```

Insert an element at particular position

The algorithm INSERT will be declared as INSERT (A, N, POS, VAL). The arguments are

(a) A, the array in which the element has to be inserted
(b) N, the number of elements in the array
(c) POS, the position at which the element has to be inserted
(d) VAL, the value that has to be inserted

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
45	23	34	12	56	20

Calling INSERT (Data, 6, 3, 100) will lead to the following processing in the array:

```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:           SET A[I + 1] = A[I]
Step 4:           SET I = I - 1
          [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]
45	23	34	12	56	20	20

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]
45	23	34	12	56	56	20

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]
45	23	34	12	12	56	20

Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]
45	23	34	100	12	56	20

Q1. WAP to insert a number in an array that is already sorted in ascending order.

Deleting an element from an array

Deletion from the end

```
Step 1: SET upper_bound = upper_bound - 1
Step 2: EXIT
```

Delete an element from a particular location

The algorithm DELETE will be declared as DELETE(A, N, POS). The arguments are:

(a) A, the array from which the element has to be deleted

(b) N, the number of elements in the array

(c) POS, the position from which the element has to be deleted

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:         SET A[I] = A[I + 1]
Step 4:         SET I = I + 1
        [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]

Merging two unsorted arrays

Array 1-	90	56	89	77	69							
Array 2-	45	88	76	99	12	58	81					
Array 3-	90	56	89	77	69	45	88	76	99	12	58	81

Figure 3.18 Merging of two unsorted arrays

```
//Merge two unsorted Arrays
#include<stdio.h>
int main()
{int a[]={100,20,30}, b[]={40,33,25,60,5},n1,n2,m,i,c[20],j;
n1=sizeof(a)/sizeof(a[0]);
n2=sizeof(b)/sizeof(b[0]);
m=n1+n2;
for(i=0;i<n1;i++)
c[i]=a[i];
for(j=0;j<n2;j++)
c[i+j]=b[j];
for(i=0;i<m;i++)
printf("%d\t",c[i]);
return 0;
}
```

```
100     20      30      40      33      25      60      5
--------------------------------
Process exited after 0.08572 seconds with return value 0
Press any key to continue . . .
```

Merging two sorted Arrays

Array 1-	20	30	40	50	60							
Array 2-	15	22	31	45	56	62	78					
Array 3-	15	20	22	30	31	40	45	50	56	60	62	78

Figure 3.19 Merging of two sorted arrays

```
//Merge two unsorted Arrays
#include<stdio.h>
int main()
{int a[]={10,20,30},
b[]={4,33,35,60,65},n1,n2,m,i,c[20],j,k;
n1=sizeof(a)/sizeof(a[0]);
n2=sizeof(b)/sizeof(b[0]);
m=n1+n2;
i=0;
j=0;
k=0;
while(i<n1&&j<n2)
{
        if(a[i]<=b[j])
        c[k++]=a[i++];
        else
        c[k++]=b[j++];
}
if(j<n2)
{
while(j<n2)
c[k++]=b[j++];
}
if(i<n1)
{
while(i<n1)
c[k++]=a[i++];
}
for(i=0;i<m;i++)
printf("%d\t",c[i]);
return 0;
}
```

```
4       10      20      30      33      35      60      65
--------------------------------
Process exited after 0.02651 seconds with return value 0
Press any key to continue . . .
```

Vivekananda Institute of Professional Studies

Arrays and Pointers

The name of an array is actually a pointer that points to the first element of the array.

```
//Arrays & Pointer
#include<stdio.h>
int main()
{int a[]={10,20,30};
int *ptr;
ptr=&a[0];
printf("%u\n%u\n%u\n%u\n",a,&a[0],&a,ptr);
}
```

```
6684172
6684172
6684172
6684172
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50},*p,i;
clrscr();
p=a;
p=&a[0];
printf("%u",p);
for(i=0;i<5;i++)
printf("\n%d",*(p+i));
getch();
}
```

```
65516    65516
10
20
30
40
50_
```

arr[i], i[arr], *(arr+i), *(i+arr) gives the same value.

```
//Arrays & Pointer2
#include<stdio.h>
int main()
{int a[]={10,20,30};
int i;
for(i=0;i<3;i++)
{printf("%d\t%d\t%d\t%d\t",a[i],i[a],*(a+i),*(i+a));
printf("\n");
}
return 0;
}
```

```
10      10      10      10
20      20      20      20
30      30      30      30
```

2-D Array

Figure 3.26 Two-dimensional array

Declaration :

datatype name_of_array[row_size][column_size2];

int marks[3][5];

The pictorial form of a two-dimensional array is shown in Fig. 3.27.

Rows / Columns	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Figure 3.27 Two-dimensional array

Vivekananda Institute
of Professional Studies

Memory Representation of 2-D Array

Row-major order

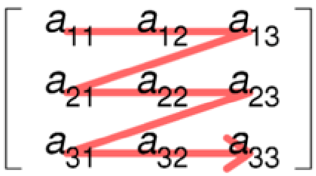
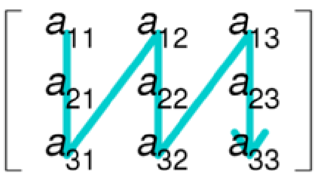


Figure 3.29 Elements of a 3×4 2D array in row major order

Column-major order



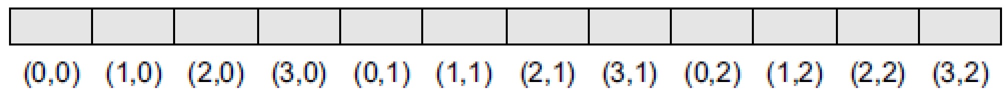


Figure 3.30 Elements of a 4×3 2D array in column major order

If the array elements are stored in column major order,

```
Address(A[I][J]) = Base_Address + w{M ( J – 1) + (I – 1)}
```

And if the array elements are stored in row major order,

```
Address(A[I][J]) = Base_Address + w{N ( I – 1) + (J – 1)}
```

Example 3.5 Consider a 20×5 two-dimensional array `marks` which has its base address = 1000 and the size of an element = 2. Now compute the address of the element, `marks[18][4]` assuming that the elements are stored in row major order.

Solution

```
Address(A[I][J]) = Base_Address + w{N (I – 1) + (J – 1)}
Address(marks[18][4]) = 1000 + 2 {5(18 – 1) + (4 – 1)}
                      = 1000 + 2 {5(17) + 3}
                      = 1000 + 2 (88)
                      = 1000 + 176 = 1176
```

Example: Given an array, arr[1.........10][1.........15] with base value 100 and the size of each element is 1 Byte in memory. Find the address of arr[8][6] with the help of row-major order.

Initialize 2-D Array

- int m[2][2] ={34,66,454,88};
- int m[2][2]= { {34,66},{454,88} };
- int m[][2]={{34,66},{454,88} };

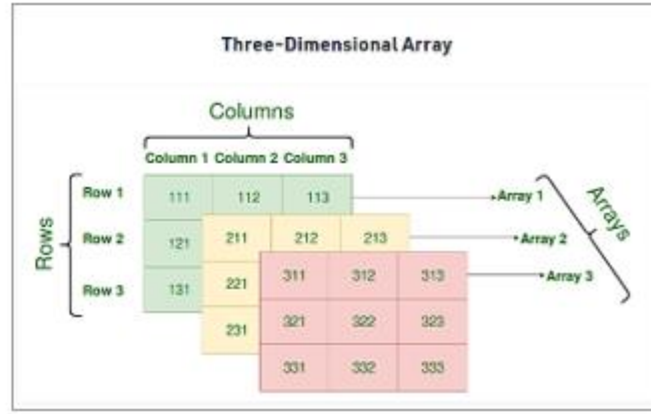
```
#include<stdio.h>
int main()
{int a[5][5],b[5][5],i,j,m,n;
printf("enter the number of rows and column of the first array A");
scanf("%d%d",&m,&n);
printf("Enter the element of first array");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf(" Print an array A\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%d\t",a[i][j]);
printf("\n");
}}
```

```
enter the number of rows and column of the first array A3
3
Enter the element of first array1
2
3
4
5
6
7
8
9
 Print an array A
1       2       3
4       5       6
7       8       9

--------------------------------
```

3-D Arrays

Syntax:
Datatype name_of_array[size1][size2][size3];
int A[2][3][3];
Total elements = 2*3*3=18




```
int arr[3][3][3]=
        { {{10,20,30},{40,50,60},{70,80,90}},    //elements of block 1
          {{11,22,33},{44,55,66},{77,88,99}},    //elements of block 2
          {{12,23,34},{45,56,67},{78,89,90}}     //elements of block 3
        };
```

Memory Representation of 3-D Array

Example 3.6 Consider a three-dimensional array defined as `int A[2][2][3]`. Calculate the number of elements in the array. Also, show the memory representation of the array in the row major order and the column major order.

Solution

A three-dimensional array consists of pages. Each page, in turn, contains `m` rows and `n` columns.

(0,0,0)	(0,0,1)	(0,0,2)	(0,1,0)	(0,1,1)	(0,1,2)	(1,0,0)	(1,0,1)	(1,0,2)	(1,1,0)	(1,1,1)	(1,1,2)

(a) Row major order

(0,0,0)	(0,1,0)	(0,0,1)	(0,1,1)	(0,0,2)	(0,1,2)	(1,0,0)	(1,1,0)	(1,0,1)	(1,1,1)	(1,0,2)	(1,1,2)

(b) Column major order

The three-dimensional array will contain $2 \times 2 \times 3 = 12$ elements.

Vivekananda Institute
of Professional Studies

Address Calculation for 3-D array

- Column major : $BA+w[(E_3*L_2+E_2)*L_1+E_1)]$
- Row major: $BA+w[(E_1*L_2+E_2)*L_3+ E_3]$
- $E_i= K_i-Lb_i$
- E_i: Effective index
- L_i: Length of I index

Suppose a three-dimensional array MAZE is declared using

MAZE(2:8, −4:1, 6:10)

Then the lengths of the three dimensions of MAZE are, respectively,

$$L_1 = 8 - 2 + 1 = 7, \qquad L_2 = 1 - (-4) + 1 = 6, \qquad L_3 = 10 - 6 + 1 = 5$$

Accordingly, MAZE contains $L_1 \cdot L_2 \cdot L_3 = 7 \cdot 6 \cdot 5 = 210$ elements.

Suppose the programming language stores MAZE in memory in row-major order, and suppose *Base*(MAZE) = 200 and there are $w = 4$ words per memory cell. The address of an element of MAZE—for example, MAZE[5, −1, 8]—is obtained as follows. The effective indices of the subscripts are, respectively,

$$E_1 = 5 - 2 = 3, \qquad E_2 = -1 - (-4) = 3, \qquad E_3 = 8 - 6 = 2$$

Using Eq. (4.9) for row-major order, we have:

$$E_1L_2 = 3 \cdot 6 = 18$$
$$E_1L_2 + E_2 = 18 + 3 = 21$$
$$(E_1L_2 + E_2)L_3 = 21 \cdot 5 = 105$$
$$(E_1L_2 + E_3)L_3 + E_3 = 105 + 2 = 107$$

Therefore,

LOC(MAZE[5, −1, 8]) = 200 + 4(107) = 200 + 428 = 628

Q Consider an array B[1:8,-5:5,-10:5]

a) Find the length of each dimension
b) Find the address of B[3,3,3], assuming BA= 400 and w=4 using row major

Address Calculation for multidimensional array

- Column major :

$BA+w[(((\ldots(E_nL_{n-1}+E_{n-1})L_{n-2})+\ldots E_3)*L_2+E_2)*L_1+E_1)]$

- Row major:

$BA+w[(\ldots((E_1*L_2+E_2)*L_3+ E_3)L_4+\ldots+E_{n-1})L_n+E_n]$

END