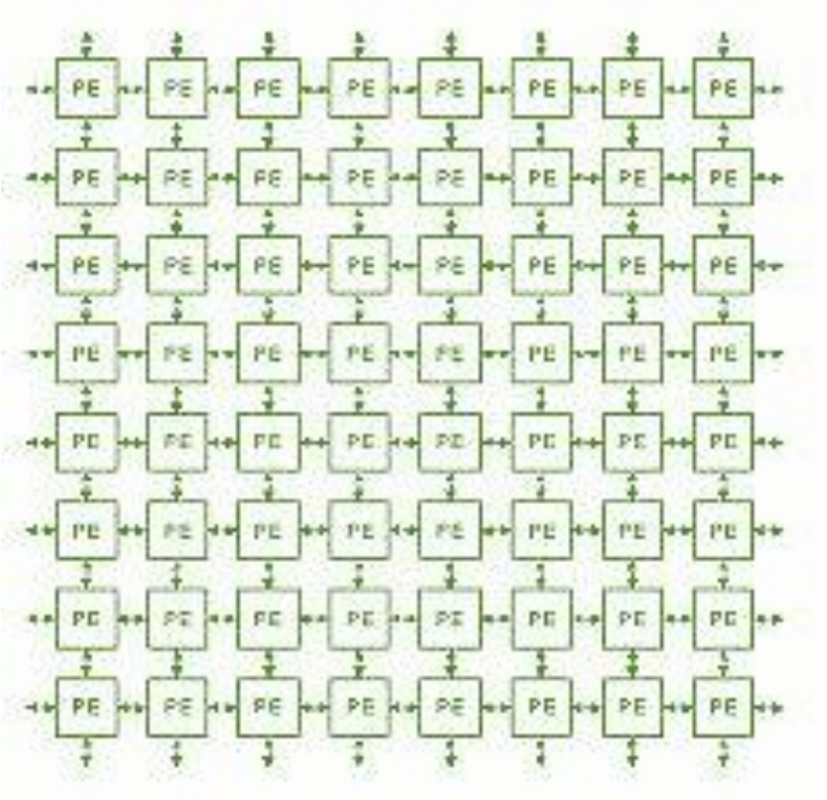


Arrays

1d arrays

LINEAR ARRAYS

- A linear array is a list of a finite number of **n** homogeneous data elements (that is data elements of the same type) such that
  - The elements are of the arrays are referenced respectively by an index set consisting of **n** consecutive numbers
  - The elements of the arrays are stored respectively in **successive memory locations**

LINEAR ARRAYS

- The number **n** of elements is called the length or size of the array.
- The index set consists of the integer **1, 2, … n**
- **Length** or the number of data elements of the array can be obtained from the index set by

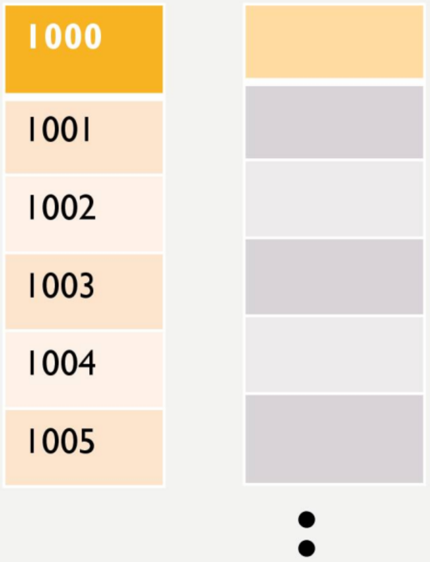
  Length = UB – LB + 1 where **UB** is the largest index called the **upper bound** and **LB** is the smallest index called the **lower bound** of the arrays

LINEAR ARRAYS

- Element of an array **A** may be denoted by
  - Subscript notation **A_1,A_2, , …. ,A_n**
  - Parenthesis notation **A(1),A(2), …. ,A(n)**
  - Bracket notation **A[1],A[2], ….. ,A[n]**

- The number **K** in A[K] is called subscript or an index and A[K] is called a **subscripted variable**

REPRESENTATION OF LINEAR ARRAY IN MEMORY



Computer Memory

REPRESENTATION OF LINEAR ARRAY IN MEMORY

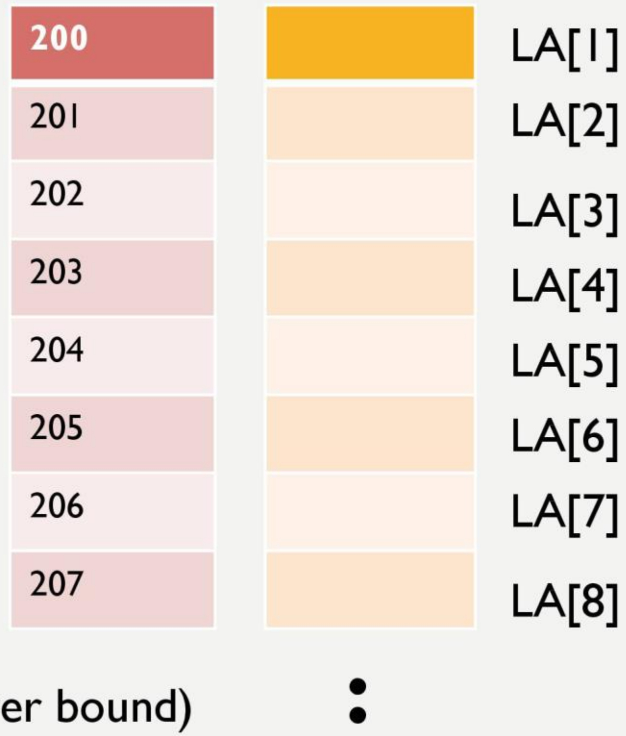
- Let **LA** be a linear array in the memory of the computer
- **LOC(LA[K]) = address of the element LA[K] of the array LA**
- The element of **LA** are stored in the successive memory cells
- Computer does not need to keep track of the address of every element of **LA,** but need to track only the address of the first element of the array denoted by **Base(LA)** called the **base address** of LA

REPRESENTATION OF LINEAR ARRAY IN MEMORY

- **LOC(LA[K]) = Base(LA) + w(K – lower bound)** where **w** is the number of words per memory cell of the array LA [**w** is aka size of the **data type**]

EXAMPLE 1

Find the address for LA[6]
Each element of the array occupy
1 byte



200	LA[1]
201	LA[2]
202	LA[3]
203	LA[4]
204	LA[5]
205	LA[6]
206	LA[7]
207	LA[8]

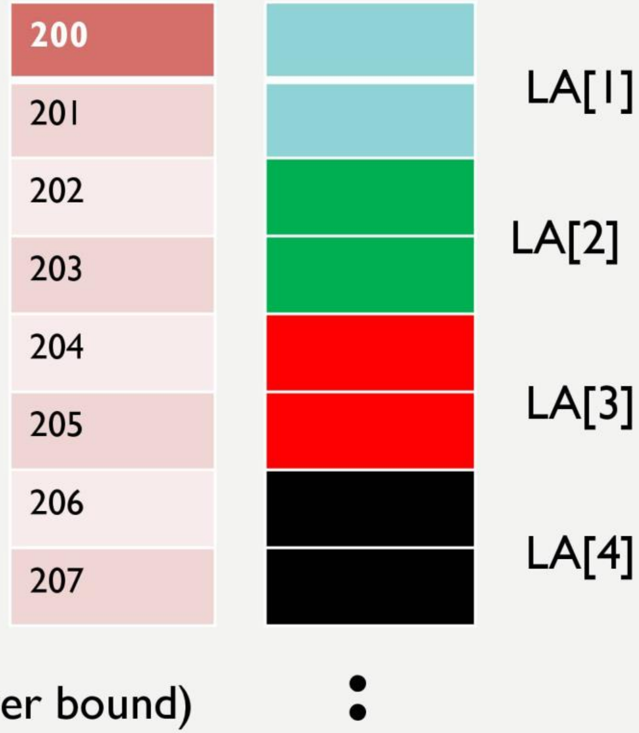
:

LOC(LA[K]) = Base(LA) + w(K – lower bound)

LOC(LA[6]) = 200 + 1(6 – 1) = 205

EXAMPLE 2

Find the address for LA[16]
Each element of the array occupy
2 byte



LOC(LA[K]) = Base(LA) + w(K – lower bound)

LOC(LA[16]) = 200 + 2(16 – 1) = 230

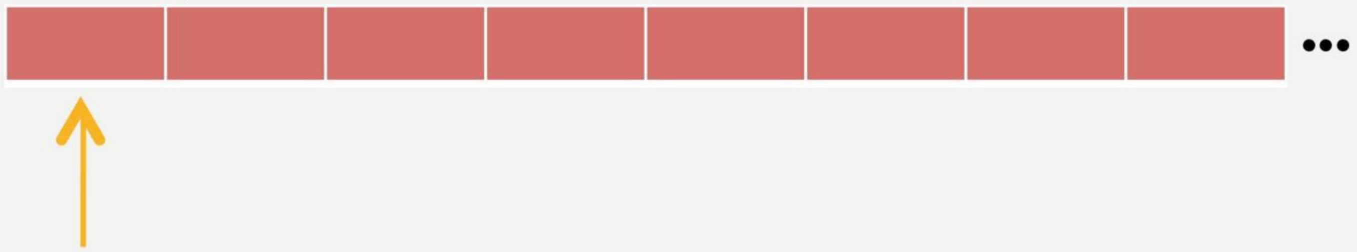
REPRESENTATION OF LINEAR ARRAY IN MEMORY

- Given any value of **K**, time to calculate **LOC(LA[K])** is same
- Given any subscript **K** one can access and locate the content of **LA[K]** without scanning any other element of **LA**
- A collection **A** of data element is said to be index if any element of **A** called $\mathbf{A}_k$ can be located and processed in time that is independent of **K**

TRAVERSING LINEAR ARRAYS

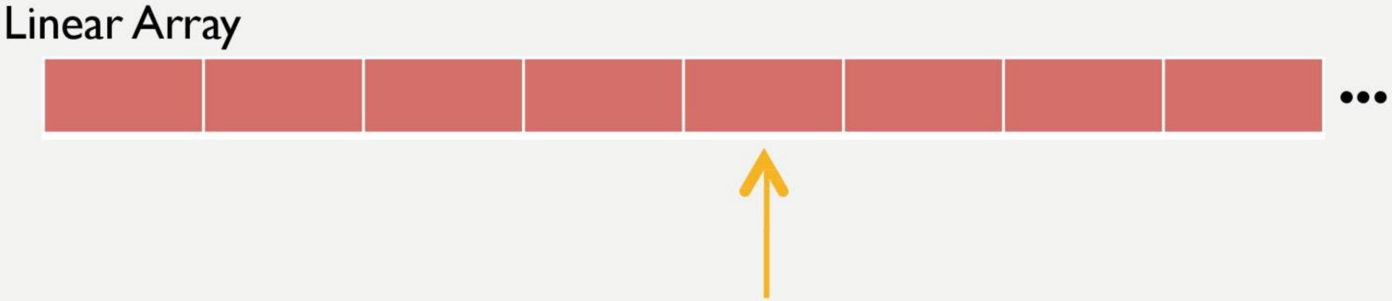
- Traversing is accessing and processing (aka visiting) each element of the data structure exactly ones

Linear Array



TRAVERSING LINEAR ARRAYS

- Traversing is accessing and processing (aka visiting) each element of the data structure exactly ones



```
1.  Repeat for K = LB to UB
    Apply PROCESS to LA[K]
    [End of Loop]
2.  Exit
```

INSERTING AND DELETING

- **Insertion**: Adding an element
  - Beginning
  - Middle
  - End

Deletion: Removing an element

- Beginning
- Middle
- End

INSERTION ALGORITHM

- **INSERT (LA, N , K , ITEM)** [LA is a linear array with N elements and K is a positive integers such that $K \le N$. This algorithm insert an element ITEM into the K^{th} position in LA]

```
1. [Initialize Counter] Set $J := N$
2. Repeat Steps 3 and 4 while $J \ge K$
3. [Move the $J^{th}$ element downward ] Set $LA[J + 1] := LA[J]$
4. [Decrease Counter] Set $J := J - 1$
5  [Insert Element] Set $LA[K] := ITEM$
6. [Reset N] Set $N := N + 1$;
7. Exit
```

DELETION ALGORITHM

- **DELETE (LA, N , K , ITEM)** [LA is a linear array with N elements and K is a positive integers such that K ≤ N.This algorithm deletes K^{th} element from LA]

```
1. Set ITEM := LA[K]
2. Repeat for J = K to N -1:
   [Move the J + 1st element upward] Set LA[J]        := LA[J + 1]
3. [Reset the number N of elements] Set N := N - 1;
4. Exit
```

Program to read n elements into an array and print it

```
int a[10], i, n;

printf("enter no of numbers“);

scanf(“%d”,&n);

printf(“enter n numbers  \n”);

for(i=0;i<n;i++)

scanf(“%d\n”,&x[i]);


printf(“\nNumbers entered are:\n”);

for(i=0;i<n;i++)

printf(“%d\n”,a[i]);
```

```
Output:
enter no of numbers
3
enter n numbers
9
11
13
Numbers entered are:
9
11
13
```

Program to add two array elements and store the corresponding elements sum in another array

```
int a[10], b[10], c[10],n, m, i;
printf("enter no. of numbers in
 first array\n“);
scanf(“%d”,&n);
//first  array
for(i=0;i<n;i++)
    scanf(“%d”,&a[i]);
printf("enter no of numbers in
second array\n“);
scanf(“%d”,&m);
for(i=0;i<m;i++) //second array
    scanf(“%d”,&b[i]);
```

```
if(m==n)
{
    for(i=0;i<m;i++)
            c[i]=a[i]+b[i];

   printf(“Sum of given array
            elements\n”);

    for(i=0;i<n;i++)
            printf(“%d\n”,c[i]);
}
else
printf("cannot add“);
}
```

Displaying elements of an array in reverse order.

```
int a[10], n, i;

printf(“Enter values\n“);

for(i=0;i<n;i++)

scanf(“%d”,&a[i]);

printf(“\nReverse order printing

 of array\n”);

for(i=n-1;i>=0;i--) // reverse loop

printf(“%d\n”,a[i]);
```

Example : a[ ]={1, 2, 3, 4, 5}

```
Enter values
   n=5
   1 2 3 4 5
   Reverse printing of array
   5   4    3    2    1
```

Array before	**Array after**
a[0]=1	**a[0]=1**
a[1]=2	**a[1]=2**
a[2]=3	**a[2]=3**
a[3]=4	**a[3]=4**
a[4]=5	**a[4]=5**

Write a program to reverse an array using only one array

```
int a[20], i, j, n, temp;

printf("enter  n \n“);

 scanf(“%d”,&n);

printf("\n Enter values for an array“);

for(i=0;i<n;i++)

 scanf(“%d”,&a[i]);
```

Contd…

Example : a[ ]={1, 2, 3, 4, 5}

```
Enter values
   n=5
   1 2 3 4 5
   Reversed array
   5   4    3    2    1
```

Array	**Reversed**
array	
a[0]=1	**a[0]=5**
a[1]=2	**a[1]=4**
a[2]=3	**a[2]=3**
a[3]=4	**a[3]=2**
a[4]=5	**a[4]=1**

Reversing an array

```
for(i=0, j=n-1; i<n/2; i++, j--)
{
    temp=a[i];
    a[i]=a[j];
    a[j]=temp;
}
printf("\n Reversed array: \n“);
for(i=0;i<n;i++)
printf(“%d\t”,a[i]);
}
```

Example :

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

```
Output:
   Enter values for an array
   n=5
  1 2 3 4 5
  Reversed array
  5   4    3    2    1
```

WAP to insert an element to an array at a given position

```
int a[100], n,i, pos,ele;
 scanf(“%d”,&n); // number of elements
printf("\nEnter the elements of array:“);
for(i=0;i<n;i++)
 scanf(“%d”,&a[i]);
printf("\nEnter the element and position  of insertion:“);
 scanf(“%d %d”,&ele,&pos);
for(i=n; i>=pos; i--) //shift the elements to right
    a[i]=a[i-1];
a[pos-1] = ele;//ele is inserted at  the specified  pos.
n = n + 1;   // increment the count of no of elements
printf("\nThe array after insertion is:“);
for(i=0;i<n; i++)     printf(“%d\n”,a[i]);
```

Example : insert 9 at 2nd position
a[ ]={1, 2, 3, 4, 5}

New array after inserting 9 :
a[ ]={1, 9, 2, 3, 4, 5}

WAP to delete an element from an array

Example : delete ele at 2nd position
a[ ]={1, 2, 3, 4, 5}

New array after deleting 2:
a[ ]={1, 3, 4, 5}

```
printf("enter no of numbers“);
scanf(“%d”,&n);
printf("enter  n numbers \n“);
for(i=0;i<n;i++)
        scanf(“%d”,&a[i]);
printf("enter the position at which the element to be deleted“);
scanf(“%d”,&pos);
for(i=pos-1; i<n-1; i++)
        a[i] =a[i+1];      //shift the elements to left
n = n-1;//decrement the count of no of elements
for(i=0;i<n;i++)
        printf(“%d”,a[i]);
```

Insert an element into a sorted array

Read array elements (in sorted order) & element '**ele**' to be inserted

Example: insert 3 into the array
a[ ] = {1, 2, 4, 5,6}

New array after inserting 3 :
a[ ] = {1, 2, 3, 4, 5,6}

```
//finding position
for(i=0;i<n;i++)
            if (ele<a[i]) break;
 pos = i+1; //position of insertion
for(i=n; i>=pos; i--) //shift the elements to right
    a[i]=a[i-1];
a[pos-1] = ele;//ele is inserted at  the specified  pos.
n = n + 1;   // increment the count of no of elements
```