



Sri
SAI RAM
ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY

West Tambaram, Chennai - 44

Sairam
INSTITUTIONS



YEAR

II

SEM

III

CS8391

DATA STRUCTURES
(COMMON TO CSE &IT)

UNIT No. 1

LINEAR DATA STRUCTURES LIST
1.2 ARRAY BASED IMPLEMENTATION



ARRAY BASED IMPLEMENTATION

1. Array based Implementation
2. Linked List based implementation

Array Implementation of list:

Array is a collection of specific number of same type of data stored in consecutive memory locations. Array is a static data structure i.e., the memory should be allocated in advance and the size is fixed. This will waste the memory space when used space is less than the allocated space.

Insertion and Deletion operation are expensive as it requires more data movements Find and Print list operations takes constant time.

20	10	30	40	50	60
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

The basic operations performed on a list of elements are

- a. Creation of List.
- b. Insertion of data in the List
- c. Deletion of data from the List
- d. Display all data's in the List
- e. Searching for a data in the list

Declaration of Array:

```
#define maxsize 10
```

```
int list[maxsize], n ;
```

Create Operation:

Create operation is used to create the list with ,, n ,, number of elements .If ,, n ,, exceeds the array's maxsize, then elements cannot be inserted into the list. Otherwise the array elements are stored in the consecutive array locations (i.e.) list [0], list [1] and so on.

```
void Create ( )
{
    int i;
    printf("\nEnter the number of elements to be added in the list:\t");
    scanf("%d",&n);
    printf("\nEnter the array
elements:\t"); for(i=0;i<n;i++)
    scanf("%d",&list[i]);
}
```

If n=6, the output of creation is as follows:

list[6]

20	10	30	40	50	60
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

Insert Operation:

Insert operation is used to insert an element at particular position in the existing list. Inserting the element in the last position of an array is easy. But inserting the element at a particular position in an array is quite difficult since it involves all the subsequent elements to be shifted one position to the right.

Routine to insert an element in the array:

```
void Insert( )
{
    int i,data,pos;
    printf("\nEnter the data to be
inserted:\t"); scanf("%d",&data);
    printf("\nEnter the position at which element to be inserted:\t");
    scanf("%d",&pos);
    if (pos==n)
        printf ("Array overflow");
```

```
for(i = n-1 ; i >= pos-1 ; i--)
```

```
list[i+1] = list[i];
```

```
list[pos-1] = data;
```

```
n=n+1;
```

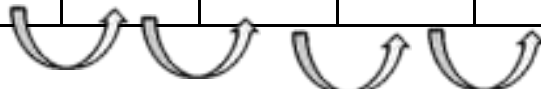
```
Display();}
```

Consider an array with 5 elements [max elements = 10]

10	20	30	40	50					
----	----	----	----	----	--	--	--	--	--

If data 15 is to be inserted in the 2nd position then 50 has to be moved to next index position, 40 has to be moved to 50 position, 30 has to be moved to 40 position and 20 has to be moved to 30 position.

10	20	30	40	50					
----	----	----	----	----	--	--	--	--	--



10		20	30	40	50				
----	--	----	----	----	----	--	--	--	--

After this four data movement, 15 is inserted in the 2nd position of the array.

10	15	20	30	40	50				
----	----	----	----	----	----	--	--	--	--

Operation:

Deletion is the process of removing an element from the array at any position.

Deleting an element from the end is easy. If an element is to be deleted from any particular position, it requires all subsequent element from that position is shifted one position towards left.

Routine to delete an element in the array:

```
void Delete( )
{
    int i, pos ;
    printf("\nEnter the position of the data to be deleted:\t");
    scanf("%d",&pos);
    printf("\nThe data deleted is:\t %d",
    list[pos-1]); for(i=pos-1;i<n-1;i++)
        list[i
    ]=list[i+
    1]; n=n-
    1;
    Display();
}
```

Consider an array with 5 elements [max elements = 10]

10	20	30	40	50					
----	----	----	----	----	--	--	--	--	--

If data 20 is to be deleted from the array, then 30 has to be moved to data 20 position, 40 has to be moved to data 30 position and 50 has to be moved to data 40 position.

10	20	30	40	50					
----	----	----	----	----	--	--	--	--	--

After this 3 data movements, data 20 is deleted from the 2nd position of the array.

10	30	40	50						
----	----	----	----	--	--	--	--	--	--

Display Operation/Traversing a list

Traversal is the process of visiting the elements in a array.

Display() operation is used to display all the elements stored in the list. The elements are stored from the index 0 to n - 1. Using a for loop, the elements in the list are

viewed

Routine to traverse/display elements of the array:

```
void display( )  
{  
    int i;  
    printf("\n*****Elements in the array*****\n");  
    for(i=0;i<n;i++)  
        printf("%d\t",list[i]);  
}
```

Search Operation:

Search() operation is used to determine whether a particular element is present in the list or not. Input the search element to be checked in the list.

Routine to search an element in the array:

```
void Search( )  
{  
    int search,i,count = 0;  
    printf("\nEnter the element to be  
searched:\t"); scanf("%d",&search);  
    for(i=0;i<n;i++)  
    {  
        if(search == list[i]) count++;  
    }  
  
    if(count==0)  
        printf("\nElement not present in  
the list"); else  
        printf("\nElement present in the list");  
}
```

Program for array implementation of List

```
#include<stdio.h>
#include<conio.h>

#define maxsize 10
int list[maxsize],n;

void Create();
void Insert();
void Delete();
void Display();
void Search();
void main()
{
    int choice;
    clrscr();
    do
    {
        printf("\n Array Implementation of List\n");
        printf("\t1.create\n");
        printf("\t2.Insert\n");
        printf("\t3.Delete\n");
        printf("\t4.Display\n");
        printf("\t5.Search\n");
        printf("\t6.Exit\n");
        printf("\nEnter your choice:\t");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                Create();
                break;
            case 2:
                Insert();
                break;
            case 3:
                Delete();
                break;
            case 4:
                Display();
                break;
            case 5:
                Search();
                break;
            case 6: exit(1);
            default: printf("\nEnter option between 1 - 6\n"); break;
        }
    }while(choice<7);
}
```

```
void Create()
{
    int i;
    printf("\nEnter the number of elements to be added in the
list:\t"); scanf("%d",&n);
    printf("\nEnter the array
elements:\t");
    for(i=0;i<n;i++)
        scanf("%d",&list[i]);
    Display();
}
void Insert()
{
    int i,data,pos;
    printf("\nEnter the data to be inserted:\t");
    scanf("%d",&data);
    printf("\nEnter the position at which element to be
inserted:\t"); scanf("%d",&pos);
    for(i = n-1 ; i >= pos-1 ; i--)
        list[i+1] = list[i];
    list[pos-1] = data; n+=1;
    Display();
}
void Delete( )
{
    int i,pos;
    printf("\nEnter the position of the data to be deleted:\t");
    scanf("%d",&pos);
    printf("\nThe data deleted is:\t %d",
list[pos-1]); for(i=pos-1;i<n-1;i++)
        list[i]=list[i+1]; n=n-1;
    Display();
}
void Display()
{
    int i;
    printf("\n*****Elements in the array*****\n");
    for(i=0;i<n;i++)
        printf("%d\t",list[i]);
}

void Search()
{
    int search,i,count = 0;
    printf("\nEnter the element to be searched:\t");
    scanf("%d",&search);
    for(i=0;i<n;i++)
    {
        if(search == list[i])
```



```
    {  
        count++;  
    }  
}  
if(count==0)  
    printf("\nElement not present in  
the list"); else  
    printf("\nElement present in the list");  
}
```

Advantages of array implementation:

1. The elements are faster to access using random access
2. Searching an element is easier

Limitation of array implementation

- An array store its nodes in consecutive memory locations.
- The number of elements in the array is fixed and it is not possible to change the number of elements .
- Insertion and deletion operation in array are expensive. Since insertion is performed by pushing the entire array one position down and deletion is performed by shifting the entire array one position up.

Applications of arrays:

Arrays are particularly used in programs that require storing large collection of similar type data elements.

Differences between Array based and Linked based implementation

	Array	Linked List
Definition	Array is a collection of elements having same data type with common name	Linked list is an ordered collection of elements which are connected by links/pointers
Access	Elements can be accessed using index/subscript, random access	Sequential access
Memory structure	Elements are stored in contiguous memory locations	Elements are stored at available memory space
Insertion & Deletion	Insertion and deletion takes more time in array	Insertion and deletion are fast and easy
Memory Allocation	Memory is allocated at compile time i.e static memory allocation	Memory is allocated at run time i.e dynamic memory allocation
Types	1D,2D,multi-dimensional	SLL, DLL circular linked list
Dependency	Each elements is independent	Each node is dependent on each other as address part contains address of next node in the list

