**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**DSA LAB ASSIGNMENT #14**

1. **A PROGRAM TO FIND THE SUM AND SMALLEST VALUES IN THE ARRAY USING POINTER.**
2. **PREPARE A REPORT ON “LINKED LIST”.**

**Submitted by**

**Ashim Bajracharya**

**B.Sc.IT (2nd Semester)**

**014BSCIT004**

**Submitted to**

**Mr. Bal Krishna Subedi**

**(Lecturer)**

**Department of Computer Science**

## (a)

## A PROGRAM TO FIND THE SUM AND SMALLEST VALUES IN THE ARRAY USING POINTER.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,n,\*ptr;

int temp,sum=0;

clrscr();

printf("\nHow Many Nos.");

scanf("%d",&n);

printf("\nEnter %d numbers\n",n);

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

{

scanf("%d",(ptr+i));

sum=sum+ \*(ptr+i);

}

for (i=0;i<n-1;i++)

{

for(j=1;j<n;j++)

{

if(\*(ptr+i)>\*(ptr+j))

{

temp=\*(ptr+i);

\*(ptr+i)=\*(ptr+j);

\*(ptr+j)=temp;

}

}

}

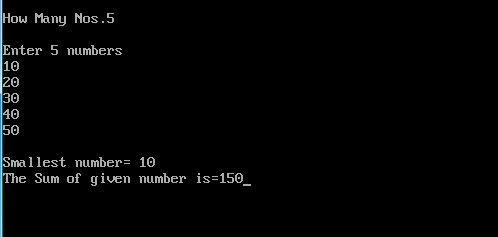
printf("\nSmallest number= %d",\*(ptr));

printf("\nThe Sum of given number is=%d",sum);

getch();

}

**OUTPUT:**

****

**CONCLUSION:**

Hence, the sum and smallest values were found in the array using pointer.

## (b)

## LINKED LIST

**INTRODUCTION:**

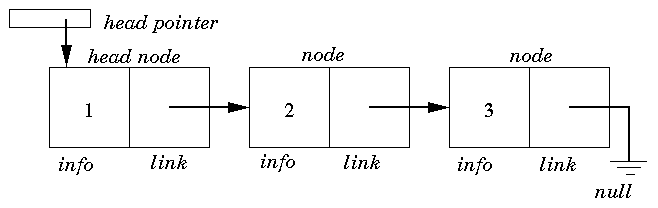
A **linked list** is a linear data structure where each element is a separate object. Each element (we will call it a node) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null.

It is also a collection of node where each node consists of two parts:

1st part -> Info & 2nd part -> Link.

1. **Info:** the actual element to be stored in the list. It is also called Data Field.
2. **Link:** one or two links that point to next and previous node in the list. It is also called next or pointer field.

**Illustration:**

****

**Fig: A Singly Linked List**

* The nodes in a linked list are not stored contiguously in the memory
* You don't have to shift any element in the list.
* Memory for each node can be allocated dynamically whenever the need arises.
* The size of a linked list can grow or shrink dynamically

**IMPORTANCES OF LINKED LIST:**

1. Linked lists are a dynamic data structure, allocating the needed memory while the program is running.
2. Insertion and deletion node operations are easily implemented in a linked list.
3. Linear data structures such as stacks and queues are easily executed with a linked list.
4. They can reduce access time and may expand in real time without memory overhead.

**DISADVANTAGES OF LINKED LIST:**

1. They have a tendency to use more memory due to pointers requiring extra storage space.
2. Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
3. Nodes are stored in-contiguously, greatly increasing the time required to access individual elements within the list.
4. Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back pointer.

**OPERATIONS ON LINKED LIST:**

The basic operations to be performed on the linked list are as follows:

1. **Creation:** This operation is used to create a linked list.
2. **Insertion:** This operation is used to insert a new nose in a kinked list in a specified position. A new node may be inserted;
   1. At the beginning of the linked list
   2. At the end of the linked list
   3. At the specified position in a linked list
3. **Deletion:** The deletion operation is used to delete a node from the linked list. A node may be deleted from;
   1. The beginning of the linked list
   2. The end of the linked list
   3. The specified position in the linked list.
4. **Traversing:** The list traversing is a process of going through all the nodes of the linked list from one end to the other end. The traversing may be either forward or backward.
5. **Searching or find:** This operation is used to find an element in a linked list. In the desired element is found then we say operation is successful otherwise unsuccessful.
6. **Concatenation:** It is the process of appending second list to the end of the first list.

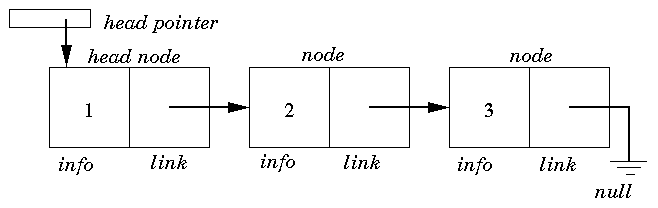
**TYPES OF LINKED LIST:**

There are four types of Linked List:

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Circular Doubly Linked List
5. **Singly Linked List:**

A singly linked list is a dynamic data structure which may grow or shrink, and growing and shrinking depends on the operation made. In this type of linked list each node contains two fields one is info field which is used to store the data items and another is link field that is used to point the next node in the list. The last node has a NULL pointer.

The following example is a singly linked list that contains three elements 1, 2, 3.

****

**Fig: A Singly Linked List**

1. **Doubly Linked List:**

In a 'doubly linked list', each node contains, besides the next-node link, a second link field pointing to the 'previous' node in the sequence. The two links may be called 'forward('s') and 'backwards', or 'next' and 'prev'('previous').

[Doubly-linked-list.svg](https://en.wikipedia.org/wiki/File:Doubly-linked-list.svg)

**Fig: Doubly Linked List**

A technique known as [XOR-linking](https://en.wikipedia.org/wiki/XOR_linked_list) allows a doubly linked list to be implemented using a single link field in each node. However, this technique requires the ability to do bit operations on addresses, and therefore may not be available in some high-level languages.

Many modern operating systems use doubly linked lists to maintain references to active processes, threads, and other dynamic objects. A common strategy for [root kits](https://en.wikipedia.org/wiki/Rootkits) to evade detection is to unlink themselves from these lists.

1. **Circular Linked List:**

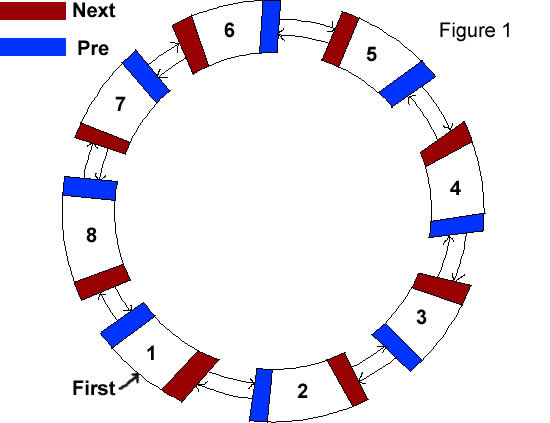
A circular linked list is a list where the link field of last node points to the very first node of the list. Circular linked lists can be used to help the traverse the same list again and again if needed. A circular list is very similar to the linear list where in the circular list the pointer of the last node points not NULL but the first node.

**[Circularly-linked-list.svg](https://en.wikipedia.org/wiki/File:Circularly-linked-list.svg)**

**Fig: A Circular Linked List**

1. **Circular Doubly Linked List:**

Doubly Circular linked list has both the properties of doubly linked list and circular linked list. Two consecutive elements are linked by previous and next pointer and the last node points to first node by next pointer and also the previous pointer of the head node points to the tail node. This list has eliminated all the short comings of all previous lists discussed in the previous sections. Node traversal from any direction is possible and also jumping from head to tail or from tail to head is only one operation: head pointer previous is tail and also tail pointer next is head. Find the visual representation of the doubly circular linked list in the below figure.



**Fig: A Circular Doubly Linked List**

**CONCLUSION:**

This report gave us a brief introduction to Linked List. It taught us about different types of Linked List. This project also enabled us to know about the various operations of the Linked List and it also lightened up some of the benefits and drawbacks of the Linked List.

**REFERENCES:**

# [1] Kevin C. Klement. (29 Nov 2010). Prepositional Logic(firstEd.)[Online].Available: <http://www.iep.utm.edu/prop-log/#SH3a>

[2]C. L. Hamblin, "An Addressless Coding Scheme based on Mathematical Notation", N.S.W University of Technology, May 1957 (typescript). Available: <http://scanftree.com/Data_Structure/infix-to-prefix>