



BHARATI VIDYAPEETH'S
COLLEGE OF
ENGINEERING

Data Structures Lab

Submitted To:

Mr.Varun Srivastava

Submitted by:

Shivam Sharma

04251202719

Experiment- 1

AIM: Experiment to implement Binary Search.

Software Used: Onlinedgb Compiler (<https://www.onlinegdb.com>)

Code

```
// Binary Search on 1D array
#include <stdio.h>

int main()
{ int a[9], i, item, f;
  //Enter an ascending ARRAY
  printf("Enter Array (max 9 elements)
\n"); for(i=0;i<9;i++)
  scanf("%d", &a[i]);
  //Enter the element to search
  printf("Enter Element To search \n");
  scanf("%d", &item);

  //Searching the Element using binary search. int beg, last, mid, pos;
  beg = 0;
  last = 8;
  while(beg<=last)
  { mid = (beg+last)/2;
    if(item == a[mid])
    {pos = mid;
     break;
    }
    else
    if(item > a[mid])
    beg = mid + 1; // The beginning shifts else
    last = mid - 1; // The end shifts }

  printf("Element Found at %d position", pos+1);
  return 0;
}
```

OUTPUT:

```
Enter Array (max 9 elements)
1
2
3
4
5
6
7
8
9
Enter Element To search
5
Element Found at 5 position
```

Conclusion: The binary search algorithm follows the method of *Divide* and *conquer*. It begins by comparing the middle element with the target value. If the target value is matches the position is returned. If the target element is less then the middle element the search continues to the left, or if it is greater the search follows towards the right.

Result: Successfully implemented binary search.

Note: The code should also contain a part which is invoked if the target element is not found.

Experiment-2

Aim: To implement Sparse Matrix using array.

Software Used: Turbo C

Code

```
// Sparse Matrix
#include <stdio.h>
#include <conio.h>
void main()
{ clrscr();
  // Defining Sparse matrix
  int sarray[4][5] = { {0,0,0,0,1},
                      {0,0,8,2,0},
                      {4,2,0,0,0},
                      {1,0,0,0,2}
                    };

  int size = 0;
  // Finding total no of Non zero elements
  for(int r=0;r<4;r++)
    for(int c=0; c<5; c++)
      if(sarray[r][c] !=0 )
        size++ ;
  printf("No. of Non-zero Representation:  %d \n ",size);

  //Making matrix for representation:(Row , Column, Value )
  int rarray[3][size] ; // is a 3x7 matrix

  //Generating Matrix
  int k=0; // Counter for Resulting matrix
  for(int row=0; row<4; row++)
    for(int col=0; col<5; col++)
      if(sarray[row][col] != 0)
        { rarray[0][k]=row;
          rarray[1][k]=col;
          rarray[2][k]=sarray[row][col];
          k++;
        }

  //Display Resulting Matrix
  printf("Triplet Representation: \n") ;
  for(r=0;r<3;r++)
```

```
{printf(" \n ");  
for(int c=0; c<size; c++)  
    printf(" %d ", rarray[r][c]);  
}  
getch();  
}
```

OUTPUT:

		No. of Non-zero Representation: 7						
		Triplet Representation:						
Row	→	0	1	1	2	2	3	3
Column	→	4	2	3	0	1	0	4
Value	→	1	8	2	4	2	1	2

Conclusion: Sparse matrix is a matrix which contains very few non-zero elements. When a sparse matrix is represented with a 2-dimensional array, we waste a lot of space to represent that matrix. Thus to optimize this, we use '*Triplet Representation*' to display or store the Sparse Matrix.

The Triplet Representation stores the value as:

Row Index
Column Index
Value

Result: Successfully implemented Sparse Matrix using array.

Experiment-3

Aim: To create a linked list having data and perform insertion and deletion at specified position.

Software Used: Turbo C

Code

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
struct node
{ int data;
  struct node *p;
};
struct node *head = NULL; //Declared Globally

void addatbeg(int); //add a node in beginning
void display();    // Display the linked list
void insertpos(); // Insert at a specified position
void delepos();
void main()
{ clrscr() ;
  addatbeg(30);
  addatbeg(40);
  addatbeg(50);
  display();
  insertpos();
  display();
  delepos();
  display();
  getch();
}
void addatbeg(int ele)
{struct node *temp; //Pointer
  temp = (struct node *) (malloc(sizeof(struct node))) ;
  temp->data = ele;
  if(head==NULL)
  {temp->p = NULL;
   head = temp;
  }
}
```

```
else
{ temp->p = head; // The ptr of new node points to prev node that was head.
  head = temp; //Temp becomes head.
}
}

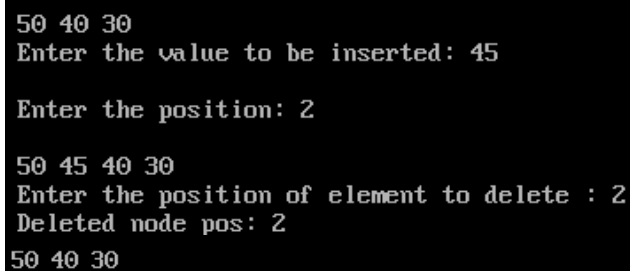
void insertpos()
{ struct node *insert = head; //Pointer to traverse till the given position.
  struct node *temp;
  temp = (struct node *) (malloc(sizeof(struct node))); //New pointer to store values.
  int val, pos=0;
  printf("\n Enter the value to be inserted: ");
  scanf("%d", &val);
  printf("\n Enter the position: ");
  scanf("%d", pos);
  temp->data = val;
  for(int i=0;i<pos;i++)
  {insert = insert->p ;
  if(insert == NULL)
  { printf("Cannot Insert here");
    return;
  }
  }
  temp->p = insert->p; // temp holds the next node
  insert->p = temp; // The prev node points to temp now.
}

void delepos()
{ struct node *del = head; //Pointer to traverse till the given position.
  struct node *temp;
  temp = (struct node *) (malloc(sizeof(struct node))); //New pointer to store new
values.
  int pos=0;
  printf("\n Enter the position of element: ");
  scanf("%d",&pos);

  for(int i=0;i<pos;i++)
  {del = del->p ;
  if(del == NULL)
  { printf("Overflow");
    return;
  }
  }
}
```

```
temp->p = del->p; // temp holds the next node
free(temp); // The prev node points to temp now.
printf(" Deleted node pos: %d ", pos);
}
void display()
{ printf("\n ") ;
  struct node *display = head; //A pointer pointing to the head node of linked list
  while(display != NULL)
  { printf("%d ", display->data);
    display = display->p;
  }
}
```

OUTPUT:



```
50 40 30
Enter the value to be inserted: 45

Enter the position: 2

50 45 40 30
Enter the position of element to delete : 2
Deleted node pos: 2
50 40 30
```

Conclusion: A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Insertion after specified node :**

It involves insertion after the specified node of the linked list. We need to skip the desired number of nodes in order to reach the node after which the new node will be inserted.

- **Deletion after specified node :**

In order to delete the node, which is present after the specified node, we need to skip the desired number of nodes to reach the node after which the node will be deleted.

We need to keep track of the two nodes. The one which is to be deleted the other one if the node which is present before that node.

Result: Successfully created a linked list and performed insertion and deletion at specified position.

Experiment-4

Aim: To create a circular linked list having data and perform insertion at end and deletion from beginning.

Software Used: Turbo C

Code

```
//Circular linked list
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct node {
    int num;
    struct node * nextp;
}*start;

struct node *tail,*p,*q; //All the pointers are pre-declared globally

void Ccreate(int ); // Create the list
void Caddlast(int ); // add element in end
void Cpopfirst(); // Delete from beginning
void displayC(int ); //display the list

void main()
{clrscr();
    int block,endl,a;
    start = NULL;
    printf(" Input the number of nodes : ");
    scanf("%d", &block);
    Ccreate(block);
    a=1; // displays original usual list
    displayC(a);
    printf(" Input the element to be inserted at end : ");
    scanf("%d", &endl);
    Caddlast(endl);
    a=2;
    displayC(a);
    Cpopfirst();
    a=2;
    displayC(a);
    getch();
}
```

```
void Ccreate(int n)
{
    int i, num;
    struct node *prevp, *newnode;
    if(n >= 1)
    {start = (struct node *)malloc(sizeof(struct node));
        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        start->num = num;
        start->nextp = NULL;
        prevp = start;
        for(i=2; i<=n; i++)
        {
            newnode = (struct node *)malloc(sizeof(struct node));
            printf(" Input data for node %d : ", i);
            scanf("%d", &num);
            newnode->num = num;
            newnode->nextp = NULL;
            prevp->nextp = newnode;
            prevp = newnode;
        }
        prevp->nextp = start;
    }
}

void Caddlast(int ele)
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    temp->num=ele;
    p=start; // pointing to the first node
    while(p->nextp!=start) // Till it doesnot reaches to the start pointer
    {
        p=p->nextp;          //P traverses
    }
    p->nextp=temp;
    temp->nextp=start;
}

void Cpopfirst()
{ struct node *store;
    p=start;
    while(p->nextp!=start)
    {
```

```
        p=p->nextp;
    }
    store=start;
    start=start->nextp;
    printf("\n The deleted node is: %d",store->num);
    p->nextp=start;
    free (store);
}

void displayC(int m)
{
    struct node *temp;
    int n = 1;
    if(start == NULL) //checks if list is empty
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        temp = start;
        if (m==1)
        {
            printf("\n Data entered in the list are :\n");
        }
        else
        {
            printf("\n New list :\n");
        }
        // Produces the new list if value other then 1 encountered.
        do {
            printf(" Data %d = %d\n", n, temp->num);
            temp = temp->nextp;
            n++;
        }while(temp != start);
    }
}
```

OUTPUT:

```
Input the number of nodes : 3
Input data for node 1 : 10
Input data for node 2 : 20
Input data for node 3 : 30

Data entered in the list are :
Data 1 = 10
Data 2 = 20
Data 3 = 30
Input the element to be inserted at end : 40

New list :
Data 1 = 10
Data 2 = 20
Data 3 = 30
Data 4 = 40

The deleted node is: 10
New list :
Data 1 = 20
Data 2 = 30
Data 3 = 40
```

Conclusion: In single linked list, every node points to its next node in the sequence and the last node points NULL. But in circular linked list, every node points to its next node in the sequence but the last node points to the first node in the list.

Result: Successfully created a circular linked list and performed insertion at beginning and deletion at end.

Experiment-5

Aim: To create doubly linked list perform Insertion at front and perform deletion at end of that doubly linked list.

Software Used: Onlinedgb Compiler (<https://www.onlinegdb.com>)

Code

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node * prev;
    struct node * next;
} *head, *last; //Globally declared pointers
void createDL(int n);
void displayDL();
void addatbeg(int);
void delete(int);
int main()
{ int n,m,data;

    head = NULL;
    last = NULL;
    printf("Enter the total number of nodes : ");
    scanf("%d", &n);
    createDL(n);
    displayDL();
    printf("\n Enter the data to be inserted at the beginning of the list: ");
    scanf("%d", &data);
    addatbeg(data);
    displayDL();
    printf("\nEnter the position of the node which you want to delete: ");
    scanf("%d", &m);
    delete(m);
    displayDL();
    return 0;
}

void createDL(int n)
{
    int i, data;
    struct node *newNode;
    if(n >= 1)
```

```
{ head = (struct node *)malloc(sizeof(struct node));
printf("Enter data of 1 node: ");
scanf("%d", &data);
head->data = data;
head->prev = NULL;
head->next = NULL;
last = head;
for(i=2; i<=n; i++)
{
newNode = (struct node *)malloc(sizeof(struct node));
printf("Enter data of %d node: ", i);
scanf("%d", &data);
newNode->data = data;
newNode->prev = last;
newNode->next = NULL;
last->next = newNode;
last = newNode;
}
printf("\nDoubly linked list has been created\n");
}
}

void displayDL()
{ struct node * temp;
int n = 1;
if(head == NULL)
{
printf("List is empty.\n");
}
else
{ temp = head;
printf("Elements of the list are:\n");
while(temp != NULL)
{printf("Elements of %d node = %d\n", n, temp->data);
n++;
temp = temp->next;
}
}
}

void addatbeg(int ele)
{ struct node * newNode;
if(head == NULL)
{ printf("Error, List is Empty!\n");
```

```
}
else
{
newNode = (struct node *)malloc(sizeof(struct node));
newNode->data = ele;
newNode->next = head;
newNode->prev = NULL;
head->prev = newNode;
head = newNode;
}
}
void delete(int p)
{ struct node *del;
struct node * Del;
int i;
del = head; // pointer pointing to head of list
for(i=1; i<p && del!=NULL; i++)
{
del = del->next;
}
if(p == 1)
{
Del = head;
head = head->next;
if (head != NULL)
head->prev = NULL;
free(Del);
printf("Deleted the node from the %d position.\n", p);
}
else if(del == last)
{
Del = last;
last = last->prev;
if (last != NULL)
last->next = NULL;
free(Del);
printf("Deleted the node from the %d position.\n", p);
}
else if(del != NULL)
{
del->prev->next = del->next; // using both the interconnected pointer
del->next->prev = del->prev;
free(del);
}
```

```
printf("Deleted the node from the %d position.\n", p);  
}  
else  
{printf("Invalid position!\n");  
}  
}
```

OUTPUT:

```
Enter data of 3 node: 30  
Enter data of 4 node: 40  
  
Doubly linked list has been created  
Elements of the list are:  
Elements of 1 node = 10  
Elements of 2 node = 20  
Elements of 3 node = 30  
Elements of 4 node = 40  
  
Enter the data to be inserted at the beginning of the list: 5  
Elements of the list are:  
Elements of 1 node = 5  
Elements of 2 node = 10  
Elements of 3 node = 20  
Elements of 4 node = 30  
Elements of 5 node = 40  
  
Enter the position of the node which you want to delete: 2  
Deleted the node from the 2 position.  
Elements of the list are:  
Elements of 1 node = 5  
Elements of 2 node = 20  
Elements of 3 node = 30  
Elements of 4 node = 40
```

Conclusion: A Doubly Linked List (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

Result: Successfully created doubly linked list and performed operations of insertion at beginning and deletion at end.

Experiment-6

Aim: Create a stack and perform Pop and Push operations on the stack using array.

Software Used: Turbo C3

Code

```
#include <stdio.h>
#include <conio.h>
int choice=0 ,n,top = -1; // declaring globally, top=-1 means that the top is null.
int stack[50]; // Stack as array -> upper bound 50.
//Function Prototypes
void push();
void pop();
void display();
void main()
{ clrscr();
printf("Enter size of stack : ");
scanf("%d ", &n);

do
{
printf("\n \t\t Choose the operation to perform \n ");
printf("\t\t ----- \n ");
printf("\t\t 1. Push \n\t\t 2. Pop\n\t\t 3. Display\n\t\t 4. Exit");
printf("\n \t\t Enter your choice: ");
scanf("%d ", &choice);
switch(choice)
{
case 1:
{
push();
break;
}
case 2:
{
pop(); // break is excluded to display the stack directly after pop operation.
}
case 3:
{display();
break;}
case 4:
```

```
        { clrscr();
          printf("\n \t\t Thank you");
          break;
        }
    default:
        {printf("Enter Valid Choice: ");
        }
    } // switch end
}while(choice!=4); // do while ends
getch();
}

void push()
{
int val;
printf("Enter Value ");
cin>>val;
top = top + 1; // top is incremented.
stack[top]=val;

if(top == n)
{printf("Overflow");
top = top - 1;
}

}

void pop()
{if(top == -1)
printf("\n stack Empty");// stack is empty
else
top = top-1; // now the top most element is excluded from the stack.
}

void display()
{ if(top == -1)
printf("\n stack Empty");
else
{ for(int i=top;i>=0;i--)
{printf("\t\t | %d | \n", stack[i]);
}
}
}
```

OUTPUT:

1) Size of stack

```
Enter size of stack : 4
```

2) Operations

a. PUSH

```
Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter Value 12

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter Value 13

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
| 13 |
| 12 |
```

b. POP

```
Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
| 12 |

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
```

c. DISPLAY

3) Stack Over flow

```
Choose the operation to perform
```

- ```

1. Push
2. Pop
3. Display
4. Exit
```

```
Enter your choice: 3
```

```
| 15 |
| 14 |
| 13 |
| 12 |
```

```
Enter your choice: 3
```

```
| 15 |
| 14 |
| 13 |
| 12 |
```

```
Choose the operation to perform
```

- ```
-----  
1. Push  
2. Pop  
3. Display  
4. Exit
```

```
Enter your choice: 1
```

```
Enter Value 16
```

```
Overflow
```

4) Stack Under flow

```
Enter your choice: 2
```

```
| 13 |  
| 12 |
```

```
Choose the operation to perform
```

- ```

1. Push
2. Pop
3. Display
4. Exit
```

```
Enter your choice: 2
```

```
| 12 |
```

```
Choose the operation to perform
```

- ```
-----  
1. Push  
2. Pop  
3. Display  
4. Exit
```

```
Enter your choice: 2
```

```
stack Empty
```

Conclusion: Stack is an ordered list in which, insertion and deletion can be performed only at one end that is called **top**. Stack is a recursive data structure having pointer to its top element. Stacks are sometimes called as Last-In-First-Out (LIFO) lists because the element which is inserted first in the stack, will be deleted last from the stack.

Result: Successfully created a stack and performed pop and Push operations on the stack using array.

Experiment-7

Aim: Create a Linear Queue using Linked List and perform operations: insert and delete on the queue elements.

Software Used: Turbo C3

Code

```
// Exp 6 Queue using linked list
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>

struct queue
{int data;
 struct queue *ptr;

};

struct queue *f= NULL, *r = NULL;  //f front , r rear-> which points to null.

void insert(int);
void remove();
void display();

void main()
{clrscr();
 int choice,n,num;

do
{
 printf("\n \t\t Choose the operation to perform \n ");
 printf("\t\t ----- \n ");
 printf("\t\t 1. Push \n\t\t 2. Pop\n\t\t 3. Display\n\t\t 4. Exit");
 printf("\n \t\t Enter your choice: ");
 scanf(" %d ", &choice);
 switch(choice)
 {
 case 1:
 {printf("\t\t Enter the element");
 scanf("%d ", &num);
 insert(num);
 break;
 }
 }
```

```
    case 2:
    {
        remove();

    }
    case 3:
    {
        display();
        break;
    }
    case 4:
    { clrscr();
        printf("\n \n \t\t Thank you");
        break;
    }
    default:
    {printf("\t\t Enter Valid Choice: ");
    }

}
}while(choice != 4);
getch();
}

void insert(int num)
{ struct queue *temp; // temp pointer to traverse.
temp = (struct queue *) (malloc(sizeof(struct queue))) ;
if(f == NULL)
{temp->ptr = NULL;
f = r = temp; // both the front and rear point to the first and only node.
temp->data = num;
}
else
{r->ptr = temp;
temp->ptr = NULL;
temp->data = num;
r = r->ptr;
}
}

void remove()
{ struct queue *remove = f; //A pointer that points to the front element.
```

```

f = f->ptr; //front pointer points to the next element in the queue.
free(remove); //free the space for the element stored in the front most part of the queue.
}
void display()
{struct queue *dis = f;
if((f == r) && (f == NULL))
{printf("\t\t Queue Empty");
}
else
{ printf( "\t\t Front->");
while(dis != NULL)
{printf(" | %d | ->", dis->data);
dis= dis->ptr;
}
printf(" Rear\n ");
}
}
}

```

OUTPUT:

1) Operations

a. INSERT

Insertion happens in the rear end of the list

```

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element 23

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element 45

```

```

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Front-> | 23 | -> | 45 | -> Rear

```

b. REMOVE

The removal takes place from the front of queue.

```
Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Front-> | 23 | -> | 45 | -> | 55 | -> | 456 | -> Rear

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Front-> | 45 | -> | 55 | -> | 456 | -> Rear
```

c. Under flow

```
Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Front-> | 55 | -> | 456 | -> Rear

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Front-> | 456 | -> Rear

Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Queue Empty
Choose the operation to perform
-----
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: _
```

Conclusion: A queue is a FIFO (First-In, First-Out) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the rear and removed from the other one end called the front.

All insertions will be done at the rear end and all the deletions will be done at the front end.

Result: Successfully created a Linear Queue using Linked List and performed operations: insert and delete on the queue elements.

Experiment-8

Aim: To implement Bubble sort using array as a data structure.

Software Used: *Turbo C3*

Code

```
#include <stdio.h>
#include <conio.h>
void main()
{ int i,j,n,temp,a[15];
  printf("What is the number of elements? - ");
  scanf("%d", &n);
  clrscr();
  printf("Number of elements: %d \n \n", n);

  printf("Please Enter the array elements : \n");
  for(i = 0; i<n; i++)
  { scanf("%d",&a[i]);
    }

  clrscr();
  printf("Before Sorting\n");
  for(i = 0; i<n; i++)
  { printf("\t %d ",a[i]);
    }
  for(i=0;i<n;i++)
  { for(j=0;j<n-i;j++) //n-i to decrement the size of list after every pass.
    { if(a[j] > a[j+1])
      { temp=a[j];
        a[j] = a[j+1];
        a[j+1] = temp;

        // swaping using third variable

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

OUTPUT:

```
Number of elements: 6
Please Enter the array elements :
10
7
5
89
12
4_
```

Before Sorting

10

7

5

89

12

4

After Sorting

4

5

7

10

12

89

Conclusion: Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are unordered. The process of sequentially traversing through all or part of list is called a pass. The bubble sort requires $n-1$ passes where n is the number of inputs.

- **Worst and Average Case Time Complexity:** $O(n*n)$, Worst case occurs when array is reverse sorted.
- **Best Case Time Complexity:** $O(n)$. Best case occurs when array is already sorted.

Result: Successfully implemented Bubble sort using array as a data structure.

Content Beyond Syllabus

Data Structure Lab

EXP-1:

Write a C program to find the largest element in a doubly linked list.

EXP-2:

Write a C program to sort a given list of strings.

Experiment-1

Aim: Write a C program to find the largest element in a doubly linked list.

Software Used: Turbo C3

Code

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

struct node
{
    int num;
    struct node *next;
    struct node *prev;
};

void create(struct node **, int ); // Creates a new node and enters the number.
int max(struct node *); // Finds the maximum number from the entered values.
void release(struct node **); // Deletes the stored values once the code ends.

int main()
{
    struct node *p = NULL; // is the initial list in which entry takes place.
    int n,size;
    printf("Enter the length of list: \n");
    scanf("%d", &size);
    printf("Enter data into the list\n");
    create(&p,size); // function call for calling the list.
    n = max(p); // function call for max.
    printf("The maximum number entered in the list is %d.\n", n);
    release (&p); // releases the captured space in memory.

    return 0;
}

int max(struct node *head)
{
    struct node *max, *q; // one is for storing max element, other to traverse.

    q = max = head;
    while (q != NULL)
    {
```

```
    if (q->num > max->num)
    {
        max = q;

    }
    q = q->next;
}

return (max->num);
}

void create(struct node **head, int size)
{
    int ele, ch;

    struct node *temp, *rear;
    for(int i=0;i<size;i++)
    {
        printf("Enter element %d :",i+1);
        scanf("%d", &ele);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->num = ele;
        temp->next = NULL;
        temp->prev = NULL;
        if (*head == NULL)
        { *head = temp;
        }
        else
        { rear->next = temp;
          temp->prev = rear;
        }
        rear = temp;
    }
    printf("\n");
}

void release(struct node **head)
{ struct node *temp = *head;

  *head = (*head)->next;
  while ((*head) != NULL)
  {
      free(temp);
      temp = *head;
      (*head) = (*head)->next;
  }
}
```

OUTPUT:

```
Enter the length of list:
5
Enter data into the list
Enter element 1 :34
Enter element 2 :56
Enter element 3 :78
Enter element 4 :43
Enter element 5 :5

The maximum number entered in the list is 78.
```

Conclusion: A Doubly Linked List (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list. Thus while dealing with doubly linked list we have to change two pointers every time a new element is added.

Result: Successfully found the largest element in a doubly linked list.

Experiment-2

Aim: Write a C program to sort a given list of strings.

Software Used: *Turbo C3*

Code

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

int main(){
    int i,j,count;
    char str[25][25],temp[25];
    printf("Enter the number of strings: ");
    scanf("%d",&count);

    printf("\n Enter Your Strings Here:\n ");
    for(i=0;i<=count;i++)
    {
        gets(str[i]);
    }

    for(i=0;i<=count;i++)
        for(j=i+1;j<=count;j++)
    {
        if(strcmp(str[i],str[j])>0) // the strings are compared to each other
        {
            // in each pass and swapped accordingly.
            strcpy(temp,str[i]);
            strcpy(str[i],str[j]);
            strcpy(str[j],temp);
        }
    }

    printf("\n Order of Sorted Strings:\n");
    for(i=0;i<=count;i++)
        puts(str[i]);
    return 0;
}
```

OUTPUT:

- 1) The comparison between the Uppercase and Lowercase :

```
Enter the number of strings:  2

Enter Your Strings Here:
h
H

Order of Sorted Strings:

H
h
```

- 2) Comparison between given strings :

```
Enter the number of strings:  6

Enter Your Strings Here:
Hi
my
name
is
Shivam
Tronix

Order of Sorted Strings:

Hi
Shivam
Tronix
is
my
name
```

Conclusion: To determine which string comes first, compare corresponding characters of the two strings from left to right. The first character where the two strings differ determines which string comes first. Characters are compared using the Unicode character set. All uppercase letters come before lower case letters. If two letters are the same case, then alphabetic order is used to compare them. If two strings contain the same characters in the same positions, then the shortest string comes first.

Result: Successfully sorted a given list of strings.
