Data Structures

Experiments

60004210159

Dhruvin Chawda

Comps

B3

Experiment 1 : Implementation of Linked Lists menu driven program

Theory :

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory. A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory. The last node of the list contains pointer to the null.

Code :

```c
#include <stdio.h>
// #include<conio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node *next;
    // since the pointer next is pointing to the node , its data
type is also node
};
struct Node *head; // declaring head global
struct Node *createnode()
{
    struct Node *temp;
    temp = (struct Node *)malloc(sizeof(struct Node));
    temp->next = NULL;
    return temp;
}
void createlist(int n)
{
    struct Node *temp, *new;
    head = temp = NULL;
    while (n > 0)
    {
        new = createnode();
        printf("Enter data: ");
        scanf("%d", &new->data);
        if (head == NULL)
        {
            head = new;
            temp = new;
        }
        else
        {
            temp->next = new;
            temp = new;
        }
        n--;
```

```
    }
}
void display()
{
    struct Node *temp;
    temp = head;
    if (temp == NULL)
    {
        printf("Linked list not present");
    }
    else
    {
        while (temp != NULL)
        {
            printf(" %d ", temp->data);
            temp = temp->next;
        }
    }
}
void insertbeg()
{
    struct Node *temp, *new;
    new = createnode();
    printf("Enter the value u wanna add at beginning: ");
    scanf("%d", &new->data);
    temp = head;
    if (head == NULL)
    {
        head = new;
    }
    else
    {
        new->next = temp;
        head = new;
    }
}
void insertend()
{
    struct Node *temp, *new;
    new = createnode();
    printf("Enter the value u wanna add at end: ");
    scanf("%d", &new->data);
    temp = head;
    if (head == NULL)
    {
        head = new;
```

```
    }
    else
    {
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = new;
        new->next = NULL;
    }
}
void insertafter()
{
    struct Node *temp, *new;
    int val;
    new = createnode();
    printf("Enter the data u wanna add in the node: ");
    scanf("%d", &new->data);
    printf("Enter the value of data after which u wanna insert: ");
    scanf("%d", &val);
    temp = head;
    if (head == NULL)
    {
        head = new;
        temp = new;
    }
    else
    {
        while (temp != NULL)
        {
            if (temp->data == val)
            {
                new->next = temp->next;
                temp->next = new;
                // break;
            }
            {
                temp = temp->next;
            }
        }
    }
}
void search()
{
    struct Node *temp, *new;
    int val, k = 0;
```

```
    printf("Enter the value u wanna search: ");
    scanf("%d", &val);
    temp = head;
    if (head == NULL)
    {
        printf("Linked List not present");
    }
    else
    {
        while (temp->next != NULL)
        {
            if (temp->data == val)
            {
                printf("%d found at %d", val, k);
            }
            k++;
            temp = temp->next;
        }
        if (!k)
        {
            printf("%d not found", val);
        }
    }
}
void delete_ll()
{
    struct Node *temp, *prev;
    int val;
    printf("Enter the value to delete: ");
    scanf("%d", &val);
    temp = head;
    prev = NULL;
    while (temp->next != NULL)
    {
        if (temp->data == val)
        {
            // printf("inside if");
            if (prev == NULL)
            {
                head = temp->next;
                temp->next = NULL;
                free(temp);
            }
            else
            {
                prev->next = temp->next;
```

```
                temp->next = NULL;
                free(temp);
            }
            break;
        }
        else
        {
            prev = temp;
            temp = temp->next;
        }
    }
    printf("%d deleted \n",val);
}
void main()
{
    int n, ins, op,c=0;
    printf("Enter no. of nodes - ");
    scanf("%d", &n);
    createlist(n);
    printf("Linked list:\n");
    display();
```

```
    while(c<4){
        printf("\nEnter 1 to add a node , 2 to search & 3 to delete:
");
            scanf("%d", &c);
    switch (c)
    {

    case 1:
        printf("\nEnter 1 to add a node at beginning , 2 for end ,
else 0: ");
        scanf("%d", &ins);
        if (ins == 1)
        {
            insertbeg();
            // display();
            // break;
        }
        else if (ins == 2)
        {
            insertend();
            // display();
            // break;
        }
```

```
        else
        {
            insertafter();
            // display();
            // break;
        }
        display();
        break;
    case 2:
        search();
        break;
    case 3:
        delete_ll();
        display();
        break;
    }}
}
```

Output :

```
Enter no. of nodes - 5
Enter data: 1
Enter data: 2
Enter data: 3
Enter data: 4
Enter data: 5
Linked list:
 1  2  3  4  5
Enter 1 to add a node , 2 to search & 3 to delete: 3
Enter the value to delete: 3
3 deleted
 1  2  4  5
Enter 1 to add a node , 2 to search & 3 to delete: 1

Enter 1 to add a node at beginning , 2 for end , else 0: 0
Enter the data u wanna add in the node: 3
Enter the value of data after which u wanna insert: 2
 1  2  3  4  5
```

```
Enter 1 to add a node , 2 to search & 3 to delete: 1

Enter 1 to add a node at beginning , 2 for end , else 0: 1
Enter the value u wanna add at beginning: 0
 0  1  2  3  4  5
Enter 1 to add a node , 2 to search & 3 to delete: 2
Enter the value u wanna search: 3
3 found at 3
Enter 1 to add a node , 2 to search & 3 to delete: 4
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion :

We have written a menu driven program to implement all the different functions of linked lists.

Experiment 2 : Implementation of polynomials operations (addition, subtraction) using Linked List.

Theory :

For addition :
To add two polynomials that are represented as a linked list, we will add the coefficients of variables with the degree.
We will traverse both the list and at any step we will compare the degree of current nodes in both the list:
1)we will add their coefficients if their degree is same and append to the resultant list .
2)Otherwise we will append the node with greater node in the resultant list.
For subtraction same procedure will be done.

Code :

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int coeff,power;
    struct node *next;
};
struct node *start1,*start2,*start3;
struct node *createnode()
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    temp->next=NULL;
    return temp;
}
struct node* createlist(int num,struct node* head)
{
    struct node *temp,*n;
    head=NULL;
    temp=NULL;
    while(num>0)
    {
        n=createnode();
        printf("Enter coeff and power ");
        scanf("%d %d",&n->coeff,&n->power);
        if(head==NULL)
        {
            head=n;
            temp=n;
        }
```

```
        else
        {
            temp->next=n;
            temp=n;
        }
        num--;
    }
    return head;
}
void display(struct node*head)
{
    struct node *temp;
    temp=head;
    if(temp==NULL)
    printf("LL not present");
    else
    {
        while(temp!=NULL)
        {
            printf("%dx^%d ",temp->coeff,temp->power);
            temp=temp->next;
        }
    }
}
void addsubnode(int c,int p)
{
    struct node *sum,*temp;
    sum=createnode();
    sum->coeff=c;
    sum->power=p;
    if(start3==NULL)
    {
        start3=sum;
        temp=sum;
    }
    else
    {
        temp->next=sum;
        temp=sum;
    }
}
void main()
{
    int n1,n2,c,ch;
    struct node *t1,*t2;
    printf("enter no. of elements in l1 ");
```

```
    scanf("%d",&n1);
    start1=createlist(n1,start1);
    printf("enter no. of elements in l2 ");
    scanf("%d",&n2);
    start2=createlist(n2,start2);
    t1=start1;
    t2=start2;
    printf("Enter 1 for addition and 2 for subtraction ");
    scanf("%d",&ch);
    if(ch==1)
    {
    while(t1!=NULL && t2!=NULL)
    {
        if(t1->power==t2->power)
        {
            c=t1->coeff+t2->coeff;
            addsubnode(c,t1->power);
            t1=t1->next;
            t2=t2->next;
        }
        else if(t1->power>t2->power)
        {
            addsubnode(t1->coeff,t1->power);
            t1=t1->next;
        }
        else
        {
            addsubnode(t2->coeff,t2->power);
            t2=t2->next;
        }
    }
    while(t1!=NULL)
    {
        addsubnode(t1->coeff,t1->power);
        t1=t1->next;
    }
    while(t2!=NULL)
    {
        addsubnode(t2->coeff,t2->power);
        t2=t2->next;
    }
    }
    else if(ch==2)
    {
        while(t1!=NULL && t2!=NULL)
    {
```

```
        if(t1->power==t2->power)
        {
            c=t1->coeff-t2->coeff;
            addsubnode(c,t1->power);
            t1=t1->next;
            t2=t2->next;
        }
        else if(t1->power>t2->power)
        {
            addsubnode(t1->coeff,t1->power);
            t1=t1->next;
        }
        else
        {
            addsubnode(-t2->coeff,t2->power);
            t2=t2->next;
        }
    }
    while(t1!=NULL)
    {
        addsubnode(t1->coeff,t1->power);
        t1=t1->next;
    }
    while(t2!=NULL)
    {
        addsubnode(-t2->coeff,t2->power);
        t2=t2->next;
    }
    }
    display(start1);
    printf("\n");
    display(start2);
    printf("\n");
    display(start3);
}
```

Output :

```
enter no. of elements in l1 3
Enter coeff and power
5
5
Enter coeff and power
6
3
Enter coeff and power
8
2
enter no. of elements in l2
2
Enter coeff and power
9
2
Enter coeff and power
7
0
Enter 1 for addition and 2 for subtraction 2
5x^5 6x^3 8x^2
9x^2 7x^0
5x^5 6x^3 -1x^2 -7x^0
PS C:\Coding And Programming\c Programming\DS>
```

Experiment 2 : Implementation of Infix to Prefix Transformation and its evaluation program

.

Experiment 4 : Implementing variant of Binary search and Fibonacci search

Theory :

Binary Search is a searching algorithm for finding an element's position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array.
Fibonacci search : Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
-Works for sorted arrays
-A Divide and Conquer Algorithm.
-Has Log n time complexity.
Differences with Binary Search:
Fibonacci Search divides given array into unequal parts
Binary Search uses a division operator to divide range. Fibonacci Search doesn't use /, but uses + and -. The division operator may be costly on some CPUs.
Fibonacci Search examines relatively closer elements in subsequent steps. So when the input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.

Code:

Binary search :

```
#include<stdio.h>

int bin(int a[], int l,int r,int val)
{
    while (l<r)
    {
        int m = l+r;
        if(a[m]==val)
        {return (m);}
        else if(a[m]>val)
        {r=m-1;}
        else if(a[m]<val)
        {l=m+1;}
    }
    return -2;
}
```

```c
void main()
{
    int n;
    int a[100],c;
    printf("Enter the number of elements : ");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        printf("\nEnter : ");
        scanf("%d",&a[i]);
    }
    printf("Enter what to search : ");
    scanf("%d",&c);
    int m = bin(a,0,n,c);
     if(m!=-2){printf("found at : %d",m+1);}
     else {printf("Not found  ! ");}

}
```

Output :

```
Enter the number of elements : 5

Enter : 10

Enter : 56

Enter : 485

Enter : 500

Enter : 899
Enter what to search : 56
found at : 2
PS C:\Coding And Programming\c Programming\DS>
```

Fibonacci search :

```c
#include<stdio.h>
int min(int x,int y)
{
    if(x<y)
        return x;
    else
        return y;
}
int fibosearch(int s,int n,int a[])
{
    int i,offset=-1,fibm1=1,fibm2=0;
    int fibm=fibm1+fibm2;
```

```
    while(fibm<n)
    {
        fibm2=fibm1;
        fibm1=fibm;
        fibm=fibm1+fibm2;
    }
    while(fibm>=1)
    {
        i=min(offset+fibm2,n-1);
        if(s>a[i])
        {
            fibm=fibm1;
            fibm1=fibm2;
            fibm2=fibm-fibm1;
            offset=i;
        }
        else if(s<a[i])
        {
            fibm=fibm2;
            fibm1=fibm1-fibm2;
            fibm2=fibm-fibm1;
        }
        else
        return i;
    }
    return -1;
}
void main()
{
    int n,a[100],i,s,flag=-1;
    printf("enter no. of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter no. ");
        scanf("%d",&a[i]);
    }
    printf("enter search ");
    scanf("%d",&s);
    flag=fibosearch(s,n,a);
    if(flag==-1)
    printf("Search not found");
    else
    printf("element found at pos %d",flag+1);
}
```

Output :

```
enter no. of elements6
enter no. 10
enter no. 56
enter no. 148
enter no. 485
enter no. 900
enter no. 999
enter search 148
element found at pos 3
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion :

Hence polynomial Subtraction is performed and excuted.

Experiment 5 : Implementation of Insertion sort, Selection sort menu driven program

Theory :

- **Insertion Sort:** Insertion sort is a very simple sorting algorithm in which the sorted array is built one element at a time. The main idea behind insertion sort is that it inserts each item into its proper place in the final list.
- **Selection Sort**: Selection sort is another sorting technique in which we find the minimum element in every iteration and place it in the array beginning from the first index. Thus, a selection sort also gets divided into a sorted and unsorted subarray.

Code :

```
#include<stdio.h>

void insertion(int a[], int n)
{
    int j;
    for(int i =1;i<n;i++)
    {
        int temp = a[i];
        j=i-1;
        while(a[j]> temp && j>=0)
        {
            a[j+1]=a[j];
            j--;
        }
        a[j+1]= temp;
    }
}
int small(int a[],int p, int n)
{
    int small = a[p],pos=p;
    for(int i = p+1 ;i<n;i++)
    {
        if(a[i]<small)
        {
            small=a[i];
            pos=i;
```

```
        }
    }
        return pos;
}
void select(int a[],int n)
{
    int pos,temp;

    for(int i=0;i<n;i++)
    {
        pos = small(a,i,n);
        temp = a[i];
        a[i] = a[pos];
        a[pos] = temp;
    }
}
void main()
{
    int arr[100],n,i,c=0;
    printf("Enter size : ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter : ");
        scanf("%d",&arr[i]);
    }
    while (c<3)
    {
        printf("Enter choice : ");
        scanf("%d",&c);
        switch (c)
            {
            case 1 : {
                        select(arr,n);
                        printf("selectioon sort : ");
                        for(i=0;i<n;i++)
                            {printf("%d\t",arr[i]);}
                        break;
                    }
            case 2 : {
                        insertion(arr,n);
                        printf("insertion sort : ");
                        for(i=0;i<n;i++)
                            {printf("%d\t",arr[i]);}
                            break;
                    }
```

```
            default:
                break;
            }
    }
}
```

Output :

```
Enter size : 5

Enter : 98

Enter : 5

Enter : -98

Enter : 0

Enter : 16
Enter choice : 1
selectioon sort : -98   0       5       16      98
 Enter choice : 3
PS C:\Coding And Programming\c Programming\DS> cd "c:
\Coding And Programming\c Programming\DS\" ; if ($?)
{ gcc insert_Selection.c -o insert_Selection } ; if (
$?) { .\insert_Selection }
Enter size : 5

Enter : 98

Enter : 5

Enter : -98

Enter : 0

Enter : 16
Enter choice : 2
insertion sort : -98    0       5       16      98
Enter choice : 3
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion :

Hence both sorting were performed and excuted .

Experiment 6 : Implementation of double ended queue menu driven program

Theory: A deque, also known as a double-ended queue, is an ordered collection of items similar to the queue. It has two ends, a front and a rear, and the items remain positioned in the collection. What makes a deque different is the unrestrictive nature of adding and removing items. New items can be added at either the front or the rear. Likewise, existing items can be removed from either end. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.

Code :

```
# include<stdio.h>
# define Size 5
int deque_arr[Size];
int front = -1;
int rear = -1;
void insert_rear()
{
    int added_item;
    if((front == 0 && rear == Size-1) || (front == rear+1))
    {   printf("Queue Overflow\n");
        return;}
    if (front == -1)
    {   front = 0;
        rear = 0;}
    else
    if(rear == Size-1)
        rear = 0;
    else
        rear = rear+1;

    printf("Input the element for adding in queue : ");
    scanf("%d", &added_item);
    deque_arr[rear] = added_item ;
}
void insert_front()
{   int added_item;
    if((front == 0 && rear == Size-1) || (front == rear+1))
    {   printf("Queue Overflow \n");
    }
    if (front == -1)
```

```
    {   front = 0;
        rear = 0;    }
    else
    if(front== 0)
        front=Size-1;
    else
        front=front-1;
    printf("Input the element for adding in queue : ");
    scanf("%d", &added_item);
    deque_arr[front] = added_item ;  }
void delete_front()
{   if (front == -1)
    {   printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",deque_arr[front]);
    if(front == rear)
    {   front = -1;
        rear=-1;
    }
    else
        if(front == Size-1)
            front = 0;
        else
            front = front+1;
}
void delete_rear()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",deque_arr[rear]);
    if(front == rear)
    {
        front = -1;
        rear=-1;
    }
    else
        if(rear == 0)
            rear=Size-1;
        else
            rear=rear-1;    }

void display_queue()
```

```
{
    int front_pos = front,rear_pos = rear;

    if(front == -1)
    {   printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
    if( front_pos <= rear_pos )
    {
        while(front_pos <= rear_pos)
        {
            printf("%d ",deque_arr[front_pos]);
            front_pos++;
        }
    }
    else
    {
        while(front_pos <= Size-1)
        {   printf("%d ",deque_arr[front_pos]);
            front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
        {
            printf("%d ",deque_arr[front_pos]);
            front_pos++;
        }
    }
    printf("\n");
}
void input_que()
{   int choice;
    do
    {   printf("1.Insert at rear\n");
        printf("2.Delete from front\n");
        printf("3.Delete from rear\n4)Exit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {   case 1:
            insert_rear();
            break;
         case 2:
            delete_front();
```

```
            break;
         case 3:
            delete_rear();
            break;
         case 4:break;
         default:
            printf("Wrong choice\n");
        }
    }while(choice!=4);
}
void output_que()
{   int choice;
    do
    {   printf("1.Insert at rear\n");
        printf("2.Insert at front\n");
        printf("3.Delete from front\n4)Exit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
         case 1:
            insert_rear();
            break;
         case 2:
            insert_front();
            break;
         case 3:
            delete_front();
            break;
            case 4:break;
         default:
            printf("Wrong choice\n");
        }
    }while(choice!=4);
}
void main()
{   int choice;
    do
    {
    printf("1.Input restricted dequeue\n");
    printf("2.Output restricted dequeue\n");
    printf("3.Display\n");
    printf("4.Exit\n");
    printf("Enter your choice : ");
    scanf("%d",&choice);
    switch(choice)
```

```
    {
     case 1 :
        input_que();
        break;
     case 2:
        output_que();
        break;
    case 3:display_queue();
            break;
        case 4:printf("Bye");
                break;
     default:
        printf("Wrong choice\n");
    }
    }while(choice!=4);
}
```

Output :

```
1.Input restricted dequeue
2.Output restricted dequeue
3.Display
4.Exit
Enter your choice : 2
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 1
Input the element for adding in queue : 1
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 1
Input the element for adding in queue : 2
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 1
Input the element for adding in queue : 3
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice :
1
Input the element for adding in queue : 4
```

```
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 1
Input the element for adding in queue : 5
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 1
Queue Overflow
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 4
1.Input restricted dequeue
2.Output restricted dequeue
3.Display
4.Exit
Enter your choice : 3
Queue elements :
1 2 3 4 5
1.Input restricted dequeue
2.Output restricted dequeue
3.Display
4.Exit
Enter your choice : 2
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 3
Element deleted from queue is : 1
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 3
Element deleted from queue is : 2
```

```
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 2
Input the element for adding in queue : 6
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 3
Element deleted from queue is : 6
1.Insert at rear
2.Insert at front
3.Delete from front
4)Exit
Enter your choice : 4
1.Input restricted dequeue
2.Output restricted dequeue
3.Display
4.Exit
Enter your choice : 3
Queue elements :
3 4 5
1.Input restricted dequeue
2.Output restricted dequeue
3.Display
4.Exit
Enter your choice : 4
Bye
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion:

We have successfully written a menu driven program to implement double ended queue or deque

Experiment 7 :Implementations of Linked Lists menu driven program (stack and queue).

Theory :

- **Linked Stack:** Each node contains two fields: data(info) and next(link) The data field of each node contains an item in the stack and the corresponding next field points to the node containing the next item in the stack The top refers to the topmost node (The last item inserted) in the stack.
- **Linked Queue:** In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory. In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Code :

```
#include<stdio.h>
#include<malloc.h>
struct node
{
    int data;
    struct node *next;
};

struct node *head;
struct node*createnode()
{
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp ->next = NULL;
    return temp;
}
void display()
{
    struct node *temp;
    temp = head;
    if(temp == NULL)
```

```
    {printf("Empty ");}
    else
    {
        while (temp !=NULL)
        {
            printf("%d\t",temp->data);
            temp=temp->next;
        }
    }
}
void insert_end(int val)
{
    struct node *temp, *new1;
    temp =head;
    new1 = createnode();
    new1->data=val;
    if(temp == NULL)
    {
        head=new1;
        temp=new1;
    }
    else
    {
    while (temp->next != NULL)
    {temp = temp->next;}
    temp->next = new1;
    temp=new1;
    }
}
void delete_front()
{
    struct node *temp, *back;
    temp =head;
    back = NULL;
    if(head == NULL)
    {printf("\nEmpty");}
    else if (head->next==NULL)
    {
        printf("\npoped head : %d",temp->data);
        head=NULL;
        free(temp);
    }
    else
    {
        head=temp->next;
        temp->next=NULL;
```

```
        printf("\npoped : %d",temp->data);
        free(temp);
    }
}
void delete_end()
{
    struct node *temp, *back;
    temp =head;
    back = NULL;
    if(head == NULL)
    {printf("\nEmpty");}
    else if (head->next==NULL)
    {
        printf("\npoped head : %d",temp->data);
        head=NULL;
        free(temp);
    }
    else
    {
        while(temp->next!=NULL)
        {
            back=temp;
            temp=temp->next;
        }
        back->next=NULL;
        printf("\npoped : %d",temp->data);
        free(temp);
    }
}
void main()
{
    int n=0,c;
    while(n<3)
    {
        printf("\nEnter the choice : ");
        scanf("%d",&n);
    switch(n)
    {
        case 1 :
            {
                while(n<4){
                printf("\nEnter the choice \n1.)push 2.)Pop
3.)Display: ");
                scanf("%d",&n);
                switch(n)
                {
```

```
                    case 1 :
                    {
                        printf("\nEnter : ");
                        scanf("%d",&c);
                        insert_end(c);break;}
                    case 2 : delete_end();break;
                    case 3 : display();break;
                    default: break;
                }}
            }
        case 2 :
            {
                while(n<4){
                printf("\nEnter the choice \n1.)insert 2.)delete
3.)Display: ");
                scanf("%d",&n);
                switch(n)
                {
                    case 1 :
                    {
                        printf("\nEnter : ");
                        scanf("%d",&c);
                        insert_end(c);break;}
                    case 2 : delete_front();break;
                    case 3 : display();break;
                    default: break;
                }}
            }
        default:break;
    }
    }
}
```

Output :
Stack :

```
Enter the choice : 1

Enter the choice
1.)push 2.)Pop 3.)Display: 1

Enter : 5

Enter the choice
1.)push 2.)Pop 3.)Display: 1

Enter : 9

Enter the choice
1.)push 2.)Pop 3.)Display: 1

Enter : 7

Enter the choice
1.)push 2.)Pop 3.)Display: 1

Enter : 8

Enter the choice
1.)push 2.)Pop 3.)Display: 3
5       9       7       8
Enter the choice
1.)push 2.)Pop 3.)Display: 2

poped : 8
Enter the choice
1.)push 2.)Pop 3.)Display: 2

poped : 7
Enter the choice
1.)push 2.)Pop 3.)Display: 2

poped : 9
Enter the choice
1.)push 2.)Pop 3.)Display: 2

poped head : 5
Enter the choice
1.)push 2.)Pop 3.)Display: 2

Empty
```

Queue :

```
Enter the choice : 2

Enter the choice
1.)insert 2.)delete 3.)Display: 1

Enter : 6

Enter the choice
1.)insert 2.)delete 3.)Display: 1

Enter : 4

Enter the choice
1.)insert 2.)delete 3.)Display: 1

Enter : 8

Enter the choice
1.)insert 2.)delete 3.)Display: 1

Enter :
1

Enter the choice
1.)insert 2.)delete 3.)Display: 3
6       4       8       1
Enter the choice
1.)insert 2.)delete 3.)Display: 2

poped : 6
Enter the choice
1.)insert 2.)delete 3.)Display: 2

poped : 4
Enter the choice
1.)insert 2.)delete 3.)Display: 2

poped : 8
Enter the choice
1.)insert 2.)delete 3.)Display: 2

poped head : 1
Enter the choice
1.)insert 2.)delete 3.)Display: 2
```

```
Empty
Enter the choice
1.)insert 2.)delete 3.)Display: 4
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion :

Hence implemenation of Stack and Queue were performed and excuted using Linked List .

Experiment 8 : Implementation of BST program.

Theory:

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers. It is called a binary tree because each tree node has a maximum of two children. It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time. The properties that separate a binary search tree from a regular binary tree is: All nodes of left subtree are less than the root node All nodes of right subtree are more than the root node Both subtrees of each node are also BSTs i.e., they have the above two properties

Code :

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node*root;
struct node*createnode(int val)
{
    struct node*temp;
    temp=(struct node*)malloc(sizeof(struct node));
    temp->left=NULL;
    temp->right=NULL;
    temp->data=val;
    return temp;
}
struct node *insert(struct node*naya,int val)
{
    struct node*temp;
    if(naya==NULL)
    {
        return createnode(val);
    }
    else if(val<naya->data)
    {
        naya->left=insert(naya->left,val);
```

```
    }
    else
    {
       naya->right=insert(naya->right,val);
    }
    return naya;
}
void preorder(struct node* root)
{
    if (root != NULL)
    {
       printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(struct node* root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void postorder(struct node* root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

struct node *inOrderSucc(struct node *temp)
{
    while(temp->left!=NULL)
    temp=temp->left;
    return temp;
}
struct node *deleteNode(struct node *temp,int val)
{
    struct node* t;
    if(val<temp->data)
    temp->left=deleteNode(temp->left,val);
```

```
    else if(val>temp->data)
    temp->right=deleteNode(temp->right,val);
    else {
        //only for nodes with 0 or 1 child
        if (temp->left == NULL) {
            t = temp->right;
            free(temp);
            return t;
        }
        else if (temp->right == NULL) {
            t = temp->left;
            free(temp);
            return t;
        }

        //only for nodes with 2 child
        t=inOrderSucc(temp->right);
        temp->data=t->data;
        temp->right=deleteNode(temp->right, t->data);
    }
    // return temp;
}
void main()
{
    int ch,val;
    struct node*root=NULL;
    do
    {
    printf("\n1)Insert\n2)All Traversal\n3)Delete\n4)Exit\n");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:printf("Enter value ");
                scanf("%d",&val);
                if(root==NULL)
                root=insert(root,val);
                else
                insert(root,val);
                break;
        case 2:if(root==NULL)
                printf("empty\n");
                else
                printf("All Traversal : \n");
                printf("preorder : ");preorder(root);
                printf("\ninorder : ");inorder(root);
                printf("\npostorder : ");postorder(root);
```

```
                break;
        case 3:if(root==NULL)
                printf("empty\n");
                else
                {
            printf("\nEnter val to be deleted ");
            scanf("%d",&val);
            deleteNode(root,val);
                }
            break;
        case 4:printf("Thank you");
            break;
        default:printf("Invalid\n");
    }
    }while(ch<4);
}
```

Output :

```
1)Insert
2)All Traversal
3)Delete
4)Exit
1
Enter value 10

1)Insert
2)All Traversal
3)Delete
4)Exit
1
Enter value 20

1)Insert
2)All Traversal
3)Delete
4)Exit
1
Enter value 1

1)Insert
2)All Traversal
3)Delete
4)Exit
1
Enter value 5
```

```
1)Insert
2)All Traversal
3)Delete
4)Exit
1
Enter value 15

1)Insert
2)All Traversal
3)Delete
4)Exit
2
All Traversal :
preorder : 10 1 5 20 15
inorder : 1 5 10 15 20
postorder : 5 1 15 20 10
1)Insert
2)All Traversal
3)Delete
4)Exit
3

Enter val to be deleted 20

1)Insert
2)All Traversal
3)Delete
4)Exit
2
All Traversal :
preorder : 10 1 5 15
inorder : 1 5 10 15
postorder : 5 1 15 10
1)Insert
2)All Traversal
3)Delete
4)Exit
4
Thank you
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion : hence all Bst operation of trees were performed and excuted.

Experiment 9 : Implementation of Graph menu driven program (DFS & BFS).

Theory:

To visit each node or vertex which is a connected component, tree-based algorithms are used. You can do this easily by iterating through all the vertices of the graph, performing the algorithm on each vertex that is still unvisited when examined. Two algorithms are generally used for the traversal of a graph: Depth first search (DFS) and Breadth first search (BFS). Depth-first Search (DFS) is an algorithm for searching a graph or tree data structure. The algorithm starts at the root (top) node of a tree and goes as far as it can down a given branch (path), and then backtracks until it finds an unexplored path, and then explores it. The algorithm does this until the entire graph has been explored.

Code :

```
#include<stdio.h>
#define Max 10
int f=-1;
int r=-1;
int a[Max];

void insert(int val)
{
    if(r==Max-1)
        printf("Overflow");
    else if(f==-1 && r==-1)
    {f=r=0;}
    else
        r++;
    a[r]=val;
}
int delete()
{
    if(f==-1 && r==-1)
       return -10;
    else
    {
        int t = a[f];
        f++;
        if(f>r)
        {f=r=-1;}
        return t;
    }
```

```
}
int isempty()
{
    if(f==-1 && r==-1)
       return -10;
    else
        {return a[f];}
}
void display()
{
    if(f==-1 && r==-1)
        printf("Under flow");
    else
    {
        for(int i=f;i<=r;i++)
        {
            printf("%d\t",a[i]);
        }
    }
}
void DFS(int a[100][100] ,int v1[], int s,int n)
{
    int stack[n];
    int top = -1,j;
    printf("%c ",s+65);
    v1[s]=1;
    stack[++top]=s;
    while(top != -1)
    {
        s = stack[top--];
        for(j=0; j<n;j++)
        {
            if(a[s][j]==1 && v1[j]==0)
            {
                stack[++top]= j;
                printf("%c ",j+65);
                v1[j]=1;
                break;
            }
        }

    }
}
void main()
{
    int n=5,v[100],v1[100],c=0;
```

```
    int i=0;
    int a[100][100];
    printf("Enter the size of matrix : ");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        printf("\nEnter the %d row : ",i+1);
        for(int j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    for(int i=0;i<n;i++)
    {v[i]=0;v1[i]=0;}
// v[i] = 0,0,0,0,0,0...
    for(int i=0;i<n;i++)
    {
        printf("\n");
        for(int j=0;j<n;j++)
        {
           printf(" %d ",a[i][j]);
        }
    }
    while(c<3)
    {
        printf("\nEnter choice : ");
        scanf("%d",&c);
        switch (c)
        {
        case 1 : {
            printf("\nvertex : ");
            scanf("%d",&i);
            printf("%c ",i+65);//soucrre
            v[i] = 1;
            insert(i);//inserting in queue
            while(isempty() != -10)
            {
                int node = delete();
                // printf("%d",isempty());
                // printf("\n%d\n",node);
                for(int j =0 ;j<n;j++)
                {
                    if(a[node][j] == 1 && v[j] == 0)
                    {
                        printf("%c ", j+65);//remaing  bfs
                        v[j] = 1;
```

```
                        insert(j);
                    }
                }
            }
            break;}
        case 2 :
        {
            printf("\nvertex : ");
            scanf("%d",&i);
            DFS(a,v1,i,n);
        }
        default:
            break;
        }
    }
}
```

Output :

```
Enter the size of matrix :
4

Enter the 1 row :
0
1
1
0

Enter the 2 row :
1
0
1
1

Enter the 3 row :
1
1
0
1

Enter the 4 row :
0
1
1
0

 0  1  1  0
 1  0  1  1
 1  1  0  1
 0  1  1  0
Enter choice : 1

vertex : 0
A B C D
Enter choice : 2

vertex : 0
A B C D
Enter choice : 3
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion:

Thus, in this experiment we have implemented Graph menu driven program (breadth-first search and depth-first search)

Experiment 10 : Implementation of hashing functions with different collision resolution techniques

Theory:

A hash function is any function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. The values are usually used to index a fixed-size table called a hash table. Use of a hash function to index a hash table is called hashing or scatter storage addressing. Since a hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique

Code :

```
#include<stdio.h>

void hash(int h[], int k, int i, int n)
{
    int j;
    j = (k%n+i)%n;
    if(i>n)
    {printf("Sorry i cant do any thing :(");}
    else if(h[j]==-1){h[j] = k;}
    else if(h[j]!=-1)
    {hash(h,k,i+1,n);}
    else
    {printf("Sorry :(");}
}
void main()
{
    int n, h[100],c=1,k,j;
    printf("Enter the size of hash table : ");
    scanf("%d",&n);
    for(int i=0;i<n;i++){h[i]=-1;}
    while(c == 1)
    {
        printf("\nEnter the key : ");
        scanf("%d",&k);
        hash(h,k,0,n);
        printf("\nWant to enter key(press 1) : ");
```

```
        scanf("%d",&c);
    }
    for(int i=0;i<n;i++)
    {
    if(h[i]!=-1)
    {printf("|%d",h[i]);}
    else{printf("|NULL");}
    }printf("|");
}
```

Output :

```
Enter the size of hash table : 11

Enter the key : 54

Want to enter key(press 1) : 1

Enter the key : 15

Want to enter key(press 1) : 1

Enter the key : 95

Want to enter key(press 1) : 1

Enter the key : 10

Want to enter key(press 1) : 1

Enter the key : 56

Want to enter key(press 1) : 1

Enter the key : 87

Want to enter key(press 1) : 1

Enter the key : 23

Want to enter key(press 1) : 3
|10|56|87|23|15|NULL|NULL|95|NULL|NULL|54|
PS C:\Coding And Programming\c Programming\DS>
```

Conclusion :

Thus, in this experiment we have implemented hashing functions with different collision resolution techniques