Experiment No 3.1

Date:

STACK IMPLEMENTATION USING LINKED LIST

AIM: Stack implementation using linked list

ALGORITHM

PROGRAM

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

struct node
{
   int data;
   struct node *next;
}*top,*newnode,*ptr;

int topelement();
void push(int data);
void pop();
void empty();
void display();

void stack_count();
void create();

int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Push");
   printf("\n 2 - Pop");
   printf("\n 3 - Display");
   printf("\n 4 - Peek");
   printf("\n 5 - Stack Count");
   printf("\n 6 -Exit");

   create();

   while (1)
   {
      printf("\n Enter choice : ");
      scanf("%d", &ch);

      switch (ch)
      {
      case 1:
         printf("Enter data : ");
         scanf("%d", &no);
         push(no);
         break;
      case 2:
         pop();
```

```
 break;
        case 3:
          display();
            break;

        case 4:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 5:
          stack_count();
            break;
        case 6:
          exit(0);

        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* Create empty stack */
void create()
{
   top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{

        newnode=(struct node *)malloc(1*sizeof(struct node));
        newnode->data = data;
        newnode->next = top;
        top=newnode;
    count++;
```

```
}

/* Display stack elements */
void display()
{
    ptr = top;

    if (top == NULL)
    {
        printf("Stack is empty");
        return;
    }
    else{

    printf("top");
    while (ptr != NULL)
    {
        printf("->%d ", ptr->data);
        ptr = ptr->next;
    }
          }
 }

/* Pop Operation on stack */
void pop()
{
    ptr= top;

    if (ptr == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else {

          printf("\n Popped value : %d", top->data);
        top= top->next;
         free(ptr);

    count--;
                  }


}

/* Return top element */
int topelement()
{
```

```
    return(top->data);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}
```

OUTPUT

```
1 - Push
 2 - Pop
 3 - Display
 4 - Peek
 5 - Stack Count
 6 -Exit
 Enter choice : 1
Enter data : 3

 Enter choice : 1
Enter data : 6

 Enter choice : 3
top->6 ->3
 Enter choice : 4

 Top element : 6
 Enter choice : 5

 No. of elements in stack : 2
 Enter choice : 2

 Popped value : 6
 Enter choice : 2

 Popped value : 3
 Enter choice : 2

 Error : Trying to pop from empty stack
 Enter choice : 6
```

Experiment No 3.2

Date:

QUEUE IMPLEMENTATION USING LINKED LIST

AIM: Queue implementation using linked list

ALGORITHM

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *link;
}*front=NULL,*rear=NULL;

int isEmpty(){
    if(front==NULL){
        return 1;
    }
    else{
        return 0;
    }
}

void enqueue(int data){
    struct node * newNode;
    newNode = malloc(sizeof(newNode));

    newNode->data= data;
    newNode->link=NULL;
    if (rear==NULL)
      front=rear=newNode;

    else{
        rear->link  =newNode;
        rear=newNode;
    }

}
void dequeue(){
    struct node* temp;
    temp=front;
    int t=isEmpty();
    if(t==1)
    {
        printf("Stack Underflow");

    }
    else{
    int val = temp ->data;
    front=front->link;
    free(temp);
    temp=NULL;
    printf("the poped element is %d",val);
    }

}
```

```
void peek(){
    int t;
    t=isEmpty();
    if(t==1){
        printf("Queue is empty");
        exit(1);

    }printf("The element at the front is %d",front->data);

}

void display(){
    struct node* temp;
    temp = front;
    int t=isEmpty();
    if(t==1)
    {
        printf("Stack Underflow");

    }
    else{
    printf("Queue elemets are ");
    while (temp!=NULL){
        printf("%d\t\t",temp->data);
        temp =temp->link;
    }
    }
    printf("\n");

}

void main()
{
    int no, ch, e;

    printf("\n 1 - Enqueue");
    printf("\n 2 - Dequeue");
    printf("\n 3 - Peek");
    printf("\n 4 - Display");
    printf("\n 5 - Exit");

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
           enqueue(no);
```

```
            break;
        case 2:
           dequeue();
            break;

        case 3:
           peek();
            break;
        case 4:
            display();
            break;
        case 5:
           exit(0);

        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}
```

OUTPUT

```
1 - Enqueue
 2 - Dequeue
 3 - Peek
 3 - Display
 4 - Exit
 Enter choice : 1
Enter data : 2

 Enter choice : 1
Enter data : 3

 Enter choice : 1
Enter data : 1

 Enter choice : 3
The element at the front is 2
 Enter choice : 4
Queue elemets are 2             3               1

 Enter choice : 2
the poped element is 2
 Enter choice : 2
the poped element is 3
 Enter choice : 2
the poped element is 1
 Enter choice : 2
Stack Underflow
 Enter choice : 5
```

Experiment No 3.3

Date:

DEQUE

AIM:DEQUE implementation using linked list(DLL)

ALGORITHM

PROGRAM

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

struct node

{
        int data;
        struct node *prev, *next;
};

struct node *head = NULL, *tail = NULL;

struct node *createNode(int data)

{

        struct node *newnode = (struct node *)malloc(sizeof(struct node));

        newnode->data = data;
        newnode->next = newnode->prev = NULL;
        return (newnode);
}

void createSentinels()
{

        head = createNode(0);
        tail = createNode(0);
        head->next = tail;
        tail->prev = head;
}

void enqueueAtFront(int data)
{
        struct node *newnode, *temp;
        newnode = createNode(data);
        temp = head->next;
head->next = newnode;
        newnode->prev = head;

        newnode->next = temp;
        temp->prev = newnode;
}

void enqueueAtRear(int data)

{
```

```
        struct node *newnode, *temp;
        newnode = createNode(data);

        temp = tail->prev;
        tail->prev = newnode;
        newnode->next = tail;
        newnode->prev = temp;
        temp->next = newnode;
}

void dequeueAtFront()
{

        struct node *temp;
        if (head->next == tail)

        {
        printf("Queue is empty\n");
        }
else
        {
        temp = head->next;
        head->next = temp->next;
        temp->next->prev = head;

        free(temp);
        }
        return;
}


void dequeueAtRear()

{
        struct node *temp;
        if (tail->prev == head)

        {
        printf("Queue is empty\n");
        }
        else
        {

        temp = tail->prev;
        tail->prev = temp->prev;
        temp->prev->next = tail;
        free(temp);
        }
```

```
        return;
}


void display()
{
        struct node *temp;

        if (head->next == tail)
        {
        printf("Queue is empty\n");
        return;
        }
temp = head->next;
        while (temp != tail)
        {

        printf("%-3d", temp->data);
        temp = temp->next;
        }
        printf("\n");
}

int main()
{
        int data, ch,c,d;

        createSentinels();
        while (1)

        {
        printf("1. Input Restricted Dequeue\n2. Output Restricted Dequeue\n");

        printf("3. Display\n4. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {

        case 1:
        c=0;
        while(c<5){
        printf("1. Enqueue at front\n");
        printf("2. Dequeue at front\n3. Dequeue at rear\n4. Display\n 5. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &c);
```

```
        switch (c){
        case 1:
printf("Enter the data to insert:");
        scanf("%d", &data);
        enqueueAtFront(data);
        break;
        case 2:
        dequeueAtFront();
        break;

        case 3:
        dequeueAtRear();
        break;
        case 4:
        display();
        break;
        case 5:
        break;



        }
        }
        break;

        case 2:
        d=0;
        while(d<5){

        printf("1. Enqueue at front\n2. Enqueue at rear\n");
        printf("3. Dequeue at Rear\n4. Display\n5. Exit\n");

        printf("Enter your choice:");
        scanf("%d", &d);

        switch (d){
        case 1:
        printf("Enter the data to insert:");
        scanf("%d", &data);
        enqueueAtFront(data);
        break;
        case 2:
        printf("Enter ur data to insert:");
        scanf("%d", &data);
enqueueAtRear(data);
        break;

        case 3:
```

```
        dequeueAtRear();
        break;

        case 4:
        display();
        break;
        case 5:
        break;
        }
        }
        break;

        case 3:
        display();
        break;

        case 4:
        exit(0);

        default:
        printf(" enter correct option\n");
        break;
        }
        }
        return 0;
}
```

OUTPUT

```
1. Input Restricted Dequeue
2.Output Restricted Dequeue
3. Display
4. Exit
Enter your choice:1
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:1
Enter the data to insert:2
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
```

```
Enter your choice:1
Enter the data to insert:4
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:1
Enter the data to insert:7
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:4
7  4  2
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:2
Dequeued element from front is 7
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:3
Dequeued element from rear is 2
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:4
4
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:2
Dequeued element from front is 4
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
```

```
 5. Exit
Enter your choice:2
Queue is empty
1. Enqueue at front
2. Dequeue at front
3. Dequeue at rear
4. Display
 5. Exit
Enter your choice:5
1. Input Restricted Dequeue
2.Output Restricted Dequeue
3. Display
4. Exit
Enter your choice:2
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:1
Enter the data to insert:4
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:1
Enter the data to insert:6
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:2
Enter ur data to insert:23
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:4
 6  4  23
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:2
```

```
Enter ur data to insert:25
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:4
 6  4  23 25
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:3
Dequeued element from rear is 25
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:3
Dequeued element from rear is 23
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:3
Dequeued element from rear is 4
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:3
Dequeued element from rear is 6
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
Enter your choice:3
Queue is empty
1. Enqueue at front
2. Enqueue at rear
3. Dequeue at Rear
4. Display
5. Exit
```

Enter your choice:5
1. Input Restricted Dequeue
2.Output Restricted Dequeue
3. Display
4. Exit
Enter your choice:4

Experiment No 3.4

Date:

POLYNOMIAL ADDITION AND MULTIPLICATION

AIM:Polynomial addition and multiplication using linkedlist(SLL/DLL)

ALGORITHM

PROGRAM

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

// Structure for a term in a polynomial
struct Term {
int coefficient;
int exponent;
struct Term *prev;
struct Term *next;
};

// Structure for a polynomial
struct Polynomial {
struct Term *head;
struct Term *tail;
};

// Function to create a new term
struct Term* createTerm(int coefficient, int exponent)
{
struct Term* newTerm = (struct Term*) malloc(sizeof(struct Term));
newTerm->coefficient = coefficient;
newTerm->exponent = exponent;
newTerm->prev = NULL;
newTerm->next = NULL;
return newTerm;
}

// Function to create a new polynomial
struct Polynomial* createPolynomial()
{
struct Polynomial* newPoly = (struct Polynomial*) malloc(sizeof(struct Polynomial));
newPoly->head = NULL;
newPoly->tail = NULL;
return newPoly;
}

// Function to add a term to a polynomial
void addTerm(struct Polynomial poly, int coefficient, int exponent)
{
struct Term newTerm = createTerm(coefficient, exponent);
if (poly->head == NULL)
{
poly->head = newTerm;
poly->tail = newTerm;
}
else
```

```
{
newTerm->prev = poly->tail;
poly->tail->next = newTerm;
poly->tail = newTerm;
}
}

// Function to add two polynomials
struct Polynomial* addPolynomials(struct Polynomial *poly1, struct Polynomial
*poly2)
{
struct Polynomial *result = createPolynomial();
struct Term *term1 = poly1->head;
struct Term *term2 = poly2->head;

// Loop through both polynomials and add terms with same exponent
while (term1 != NULL && term2 != NULL)
{
    if (term1->exponent == term2->exponent)
    {
        addTerm(result, term1->coefficient + term2->coefficient, term1->exponent);
        term1 = term1->next;
        term2 = term2->next;
    }
    else if (term1->exponent > term2->exponent)
    {
        addTerm(result, term1->coefficient, term1->exponent);
        term1 = term1->next;
    }
    else
    {
        addTerm(result, term2->coefficient, term2->exponent);
        term2 = term2->next;
    }
}

// Add remaining terms from first polynomial
while (term1 != NULL)
{
    addTerm(result, term1->coefficient, term1->exponent);
    term1 = term1->next;
}

// Add remaining terms from second polynomial
while (term2 != NULL)
{
    addTerm(result, term2->coefficient, term2->exponent);
    term2 = term2->next;
```

```
}
}
```

Polynomial addition and Multipliction using linked list

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

struct node {
   float coeff;
   int expo;
   struct node* link;
};

struct node* insert(struct node* head, float co, int ex)
{
   struct node* temp;
   struct node* newP = malloc(sizeof(struct node));
   newP->coeff = co;
   newP->expo = ex;
   newP->link = NULL;

   if(head == NULL || ex > head->expo)
   {
      newP->link = head;
      head = newP;
   }
   else
   {
      temp = head;
      while(temp->link != NULL && temp->link->expo >= ex)
         temp = temp->link;
      newP->link = temp->link;
      temp->link = newP;
   }
   return head;
}

struct node* create(struct node* head)
{
   int n, i;
   float coeff;
   int expo;

   printf("Enter the number of terms: ");
   scanf("%d", &n);

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

```
    {
        printf("Enter the coefficient for term %d: ", i+1);
        scanf("%f", &coeff);

        printf("Enter the exponent for term %d: ", i+1);
        scanf("%d", &expo);

        head = insert(head, coeff, expo);
    }
    return head;
}

void print(struct node* head)
{
    if(head == NULL)
        printf("No Polynomial.");
    else {
        struct node* temp = head;
        while(temp != NULL)
        {
            printf("(%.1fx^%d)", temp->coeff, temp->expo);
            temp = temp->link;
            if(temp!=NULL)
                printf(" + ");
            else printf("\n");
        }
    }
}

void polyMult(struct node* head1, struct node* head2)
{
    struct node* ptr1 = head1;
    struct node* ptr2 = head2;
    struct node* head3 = NULL;

   if(head1 == NULL || head2 == NULL)
    {
        printf("Zero polynomial\n");
        return;
    }

    //Multiplication of two polynomials
    while(ptr1 != NULL)
    {
        while(ptr2 != NULL)
        {
            head3 = insert(head3, ptr1->coeff * ptr2->coeff, ptr1->expo + ptr2->expo);
            ptr2 = ptr2->link;
```

```
        }
        ptr1 = ptr1->link;
        ptr2 = head2;
    }

    //Adding the like terms (terms with the same exponent) for simplification
    struct node* ptr3 = head3;
    struct node* temp = NULL;

    while(ptr3->link != NULL) {
        if(ptr3->expo == ptr3->link->expo)
        {
            ptr3->coeff = ptr3->coeff + ptr3->link->coeff;
            temp = ptr3->link;
            ptr3->link = ptr3->link->link;
            free(temp);
        }
        else {
            ptr3 = ptr3->link;
        }
    }
    printf("\n Multiplication result :");
    print(head3);
}
void polyAdd(struct node* head1, struct node* head2)
{
    struct node* ptr1 = head1;
    struct node* ptr2 = head2;
    struct node* head3 = NULL;
    while(ptr1!=NULL && ptr2!=NULL)
    {
        if(ptr1->expo == ptr2->expo)
    {
            head3 = insert(head3, ptr1->coeff+ptr2->coeff, ptr1->expo);
            ptr1 = ptr1->link;
            ptr2 = ptr2->link;
        }
        else if(ptr1->expo > ptr2->expo)
        {
            head3 = insert(head3, ptr1->coeff, ptr1->expo);
            ptr1 = ptr1->link;
        }
        else if(ptr1->expo < ptr2->expo)
        {
         head3 = insert(head3, ptr2->coeff, ptr2->expo);
         ptr2 = ptr2->link;
        }
    }
```

```
    while(ptr1!=NULL)
    {
        head3 = insert(head3, ptr1->coeff, ptr1->expo);
        ptr1 = ptr1->link;
    }
    while(ptr2!=NULL)
    {
        head3 = insert(head3, ptr2->coeff, ptr2->expo);
        ptr2 = ptr2->link;
    }
    printf("Added polynomial is: ");
    print(head3);
}

int main()
{
    struct node* head1 = NULL;
    struct node* head2 = NULL;
    printf("Enter the first polynomial\n ");
    head1 = create(head1);
    printf("Enter the second polynomial\n ");
    head2 = create(head2);
    int ch;
    do{
        printf("\n1.Addition ");
        printf("\n2.Multiplication");
        printf("\n3.Exit");
        printf("\nChoose the choice");
        scanf("%d",&ch);
        switch(ch){
            case 1:
              polyAdd(head1,head2);
              break;
            case 2:
               polyMult(head1, head2);
               break;
            default:
              exit(1);


        }
    }while(ch<3);


    return 0;
}
```

OUTPUT

```
Enter the first polynomial
 Enter the number of terms: 2
Enter the coefficient for term 1: 3
Enter the exponent for term 1: 2
Enter the coefficient for term 2: 2
Enter the exponent for term 2: 1
Enter the second polynomial
 Enter the number of terms: 3
Enter the coefficient for term 1: 3
Enter the exponent for term 1: 3
Enter the coefficient for term 2: 4
Enter the exponent for term 2: 5
Enter the coefficient for term 3: 2
Enter the exponent for term 3: 0

1.Addition
2.Multiplication
3.Exit
Choose the choice1
Added polynomial is: (4.0x^5) + (3.0x^3) + (3.0x^2) + (2.0x^1) + (2.0x^0)

1.Addition
2.Multiplication
3.Exit
Choose the choice2

 Multiplication result :(12.0x^7) + (8.0x^6) + (9.0x^5) + (6.0x^4) + (6.0x^2) + (4.0x^1)

1.Addition
2.Multiplication
3.Exit
Choose the choice3
```

Experiment No 3.5

Date:

CIRCULAR LINKEDLIST

AIM: Implement a circular linkedlist(SLL/DLL)

ALGORITHM

PROGRAM

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

struct Node{
        int data;
        struct Node *next;
}*newnode;
struct Node *tail=NULL;
int length=0;
struct Node *createnode(){
        int data;
        newnode=(struct Node *)malloc(sizeof(struct Node));
        printf("enter data\n");
        scanf("%d",&data);
        newnode->data=data;
        newnode->next=NULL;
        return newnode;
}

void insertbeg(){
        newnode=createnode();
        if(tail==NULL){

        tail=newnode;
        tail->next=newnode;
        }
        else{
        newnode->next=tail->next;
        tail->next=newnode;
        }
        length++;

}

void insertend(){
        newnode=createnode();
        if(tail==NULL){
        tail=newnode;
        tail->next=newnode;
        }
        else{
        newnode->next=tail->next;
        tail->next=newnode;
        tail=newnode;
        }
        length++;
```

```
}

void insertpos(){

        struct Node *ptr;
        int pos,i=1;
        printf("enter pos\n");
        scanf("%d",&pos);
        if(pos<0 || pos>length+1){
        printf("invalid \n");
        }
        else if (pos==1)
        {
        insertbeg();
        }
        else if (pos==length+1)
        {
        insertend();
        }
        else{
        newnode=createnode();
        ptr=tail->next;
        while (i<pos-1)
        {
        ptr=ptr->next;
        i++;
        }
        newnode->next=ptr->next;
        ptr->next=newnode;
        length++;
        }




}


void deletefront(){
        if (tail==NULL)
        {
        printf("Empty\n");
        }
        else if(length==1){
        printf("%d deleted \n",tail->data);
        tail=NULL;
        length--;
        }
```

```
        else{
        printf("%d deleted \n",tail->next->data);
        tail->next=tail->next->next;
        length--;

        }


}

void deleteend(){

        if (tail==NULL)
        {
        printf("Empty\n");
        }
        else if(length==1){
        printf("%d deleted \n",tail->data);
        tail=NULL;
        }
        else{
        struct Node *ptr=tail->next;
        while (ptr->next!=tail)
        {
        ptr=ptr->next;
        }
        printf("%d deleted \n",tail->data);
        ptr->next=tail->next;
        tail->next=NULL;
        tail=ptr;
        length--;
        }


}

void deletepos(){


        int pos,i=1;
        printf("enter pos\n");
        scanf("%d",&pos);
        if(pos<0 || pos>length+1){
        printf("invalid \n");
        }
        else if (pos==1)
        {
        deletefront();
```

```
        }
        else if (pos==length+1)
        {
        deleteend();
        }
        else{
        struct Node *prev,*other;
        prev=tail->next;
        while (i<pos-1)
        {
        prev=prev->next;
        i++;
        }
        printf("%d deleted \n",prev->next->data);
        other=prev->next->next;
        prev->next->next=NULL;
        prev->next=other;
        length--;
        }



}


void display(){

        if(tail==NULL){
        printf("empty\n");
        }
        else{
        struct Node *temp=tail ->next;
        while (temp->next!=tail)
        {
        printf("%d ->",temp->data);
        temp=temp->next;
        }
        if(tail==tail->next){
        printf("%d \n",temp->data);
        }
        else{printf("%d->",temp->data);
        temp=temp->next;
printf("%d \n",temp->data);}

        }
```

```
}

void main(){
        int ch=1;

        while (ch!=9)
        {
        printf("1.Insert at front \n2.Insert at end\n3.Insert at any pos\n4.Delete
front\n5.Delete end\n6.Delete any pos \n7.display\n8.Exit \n");
        printf("enter choice\n");
        scanf("%d",&ch);
        switch(ch){
        case 1:
        insertbeg();
        break;
        case 2:
        insertend();
        break;
        case 3:
        insertpos();
        break;
        case 4:
        deletefront();
        break;
        case 5:
        deleteend();
        break;
        case 6:
        deletepos();
        break;
        case 7:
        display();
        break;
        case 8:
        exit(0);
        default:
        printf("wrong choice \n");
        }
        }

}
```

OUTPUT

1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end

6.Delete any pos
7.display
8.Exit
enter choice
1
enter data
4
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
1
enter data
5
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
2
enter data
6
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
3
enter pos
2
enter data
18
1.Insert at front
2.Insert at end
3.Insert at any pos

4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
7
5 ->18 ->4->6
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
6
enter pos
3
4 deleted
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
7
5 ->18->6
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
4
5 deleted
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end

6.Delete any pos
7.display
8.Exit
enter choice
5
6 deleted
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
5
18 deleted
1.Insert at front
2.Insert at end
3.Insert at any pos
4.Delete front
5.Delete end
6.Delete any pos
7.display
8.Exit
enter choice
7
empty

Experiment No 3.6

Date:

BINARY TREE(BST)

AIM: Build a Binary tree(BST) and implement all operations and traversals

ALGORITHM

PROGRAM

```
 #include <stdio.h>
#include <stdlib.h>
struct node
{
int data;
struct node *leftchild;
struct node *rightchild;
}node;
struct node *root = NULL;
typedef struct node Node;
//insert node
void insert()
{
 struct node *newnode;
 newnode = (struct node *)malloc(sizeof(struct node));
 printf("Enter data : ");
 scanf("%d",&newnode->data);
 newnode->leftchild = NULL;
 newnode->rightchild = NULL;
 if(root == NULL)
 root = newnode;
 else
 {
 struct node *previous, *current;
 current = root;
 while(1)
 {
 if(newnode->data < current->data)
 {
 previous = current;
current = current->leftchild;
if(current == NULL)
{
 previous->leftchild = newnode;
 break;
 }
 }
 else if(newnode->data >= current->data)
 {
 previous = current;
current = current->rightchild;
 if(current == NULL)
 {
 previous->rightchild = newnode;
break;
 }
 }
```

```
 else
 {
 printf("Invalid Input\n");
 exit(0);
 }
 }
 }
}
//search for a element
void search()
{
 int key, f = 0;
 printf("Enter the key element to be searched : ");
 scanf("%d",&key);
 struct node *current = root;
 while(current != NULL)
 {
 if(current->data == key)
 {
 f = 1;
 break;
 }

 if(key < current->data)
 current = current->leftchild;

 else if(key > current->data)
 current = current->rightchild;
 }
 if(f)
 printf("Key Element Found !!!\n");
 else
 printf("Key Element NOT found !!!\n");
}
//inorder traversal
void inorder(struct node *ptr)
{
 if(ptr != NULL)
 {
 inorder(ptr->leftchild);
 printf(" %d\t",ptr->data);
 inorder(ptr->rightchild);
 }
}
//preorder traversal

void preorder(struct node *ptr)
{
```

```
 if(ptr)
 {
 printf(" %d\t",ptr->data);
 preorder(ptr->leftchild);
 preorder(ptr->rightchild);
 }
}
//postroder traversal
void postorder(struct node *ptr)
{
 if(ptr)
 {
 postorder(ptr->leftchild);
 postorder(ptr->rightchild);
 printf(" %d\t",ptr->data);
 }
}
struct node *ins(struct node *ptr) /*to find successor*/
{
 struct node *q = NULL;
 while(ptr->leftchild != NULL)
 {
 q = ptr;
 ptr = ptr->leftchild;
 }
 if(ptr->rightchild != NULL)
 q->leftchild = ptr->rightchild;
 else
 q->leftchild = NULL;
 return(ptr);
}
//deleet element
void delete()
{
 printf("Enter node value to delete : ");
 int d, f=0;
 scanf("%d",&d);
 struct node *current = root;
 struct node *ptr = NULL;
 while(current != NULL)
 {
 if(current->data == d)
 {
 f=1;
 break;
 }

 if(d < current->data)
```

```
{
ptr=current;
current = current->leftchild;
}
else if(d>current->data)
{
ptr = current;
current = current->rightchild;
}
}

if(f == 0)
printf("Element to delete NOT found !!!");
else
{struct node *t = current;
if(t->leftchild == NULL && t->rightchild == NULL)
{
if(ptr->leftchild == t)
ptr->leftchild = NULL;
if(ptr->rightchild == t)
ptr->rightchild = NULL;
}
else if(t->leftchild == NULL)
{
if(ptr->leftchild == t)
ptr->leftchild = t->rightchild;
if(ptr->rightchild == t)
ptr->rightchild = t->rightchild;
}

else if(t->rightchild == NULL)
{
if(ptr->leftchild == t)
ptr->leftchild = t->leftchild;
if(ptr->rightchild == t)
ptr->rightchild = t->leftchild;
}
else
{
struct node *in = t->rightchild;
if(in->leftchild == NULL)
{
t->data = in->data;
t->rightchild = in->rightchild;
}
else
{
in=ins(t->rightchild);
```

```
 t->data=in->data;
 }
 }
 }
}
//main fxn
void main()
{
 int ch;
 printf("\n---- Menu ----\n1.Insert a new node\n2.Inorder Traversal\n3.Preorder
Traversal\n4.Postorder Traversal\n5.Delete a node\n6.Search for an
Element\n7.Exit\n");
 do{
 printf("\nEnter your choice : ");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:insert();
 break;
 case 2:printf("Inorder Traversal\n");
 inorder(root);
break;
 case 3:printf("Preorder Traversal\n");
 preorder(root);
 break;
 case 4:printf("Postorder Traversal\n");
 postorder(root);
 break;
 case 5:delete();
 break;
 case 6:search();
 break;
 }
 }while(ch != 7);
}
```

OUTPUT

```
---- Menu ----
1.Insert a new node
2.Inorder Traversal
3.Preorder Traversal
4.Postorder Traversal
5.Delete a node
6.Search for an Element
7.Exit

Enter your choice : 1
Enter data : 25
```

```
Enter your choice : 1
Enter data : 15

Enter your choice : 1
Enter data : 50

Enter your choice : 1
Enter data : 10

Enter your choice : 1
Enter data : 22

Enter your choice : 1
Enter data : 35

Enter your choice : 1
Enter data : 70

Enter your choice : 1
Enter data : 4

Enter your choice : 1
Enter data : 12

Enter your choice : 1
Enter data : 18

Enter your choice : 1
Enter data : 24

Enter your choice : 1
Enter data : 31

Enter your choice : 1
Enter data : 44

Enter your choice : 1
Enter data : 66

Enter your choice : 1
Enter data : 90

Enter your choice : 2
Inorder Traversal
 4       10      12      15      18      22      24      25      31      35      44      50      66
70      90
Enter your choice : 3
```

```
Preorder Traversal
 25      15      10      4       12      22      18      24      50      35      31      44      70
66      90
Enter your choice : 4
Postorder Traversal
 4       12      10      18      24      22      15      31      44      35      66      90      70
50      25
Enter your choice : 6
Enter the key element to be searched : 12
Key Element Found !!!

Enter your choice : 5
Enter node value to delete : 24

Enter your choice : 2
Inorder Traversal
 4       10      12      15      18      22      25      31      35      44      50      66      70
90
Enter your choice : 7
```