1.

Expt-1

Linked List

Aim

write a menu driven program for performing the following operations on a linked list. 1. Display 2. Insert at beginning. 4. Insert at specified position. 5. Delete from beginning 6. Delete from End 7. Delete from specified position.

Algorithms

Step 1: Start

Step 2: Create a structure node with data and node. pointer link.

Step 3: Create a node. pointer head and initalize. to NULL.

Step 4: Declare alen() function.

i) intialize temp = head; length = 0

ii) repeat steps (iii)(iv) below till temp != null.

iii) length ++;

iv = temp = temp -> link.

v return length,

Step 5: Declare display

i) temp = head

ii) repeat iv till temp != NULL.

iii) print temp -> data.

iv temp = temp -> link.

step 6: Delare add_end fn with data as input.

i) Intialize temp->hed.

ii) make a newnode, newnod->data=dat

iii) newnode->link=NULL

iii) if (tem!=NULL). repeat step blat. till temp->link != NULL

iv tem = temp->link,

temp->link= newnode

else.

vii else head=newnode.

step 7: Declare add_beg (int data).

i) Create newnode, newnode->data=data.

ii) newnod->link= head.

iii) head =newnode

step 8: Declare add_pos (int data, int pos).

i) node * temp=head

ii create newnode newnod->data = data.

iii if (pos<=0 && pos > lon'hod

point invalid.

iv. else if (po->1)

repeat step below till --pos > 1 && temp->link != NULL).

vi) temp = temp->link.

vii newnode->link = temp->link.

temp->link = newnode.

ix else add_beg (data).

step 9: Declare del_beg().

i) if(head != NULL) head = head -> link;

ii) else. printf("LL is empty

step 10: Declare del_end() to delete at end.

i) temp = head.

ii) if(temp == NULL) print empty

iii) else.

iv) if(head -> link != NULL).

v) repeat #step below tom temp -> link != NULL.

vi) temp = temp -> link.

vii) temp -> = NULL

else head = NULL

step 11. Declare del pos (int pos).

i) node * temp = head;

ii) if(pos <= 0 || pos > len(head) print invalid.

iii) else if(pos > 1).

repeat step below till --pos > 1 ||
temp != NULL.

temp = temp -> link.

iv) temp -> link = temp -> link -> link

v) else del beg().

Step 12 Declare main function..

Step 13 ~~Declare~~ a menu driven program to call the necessary function

Step 14: Stop.

Code

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

struct node
{
    int data;
    struct node *link;
};

struct node *head = NULL;

int length()
{
    struct node *temp = head;
    int length = 0;
    while (temp != NULL)
    {
        temp = temp->link;
        length++;
    }
    return length;
}

void display()
{
    struct node *temp = head;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}

void add_end(int data)
{
    struct node *temp = head;
    struct node *newnode = (struct node *)malloc(sizeof(struct node));

    newnode->data = data;

    newnode->link = NULL;

    if (temp != NULL)
    {
        while (temp->link != NULL)
        {
```

```
            temp = temp->link;
        }
        temp->link = newnode;
    }
    else
    {

        head = newnode;
    }
}

void add_beg(int data)
{
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = data;
    newnode->link = head;
    head = newnode;
}

void add_pos(int data, int pos)
{
    struct node *temp = head;
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = data;

    if (pos > length() || pos < 1)
    {
        printf("Invalid Position\n");
    }
    if (pos > 1)
    {
        while (--pos > 1 && temp != NULL)
        {
            temp = temp->link;
        }

        newnode->link = temp->link;
        temp->link = newnode;
    }
    else
    {
        add_beg(data);
    }
}

void del_beg()
{
    if (head != NULL)
```

```
    {
        head = head->link;
    }
    else
    {
        printf("Linkd List is empty\n");
    }
}

void del_end()
{
    struct node *temp = head;
    if (temp == NULL)
    {
        printf("The Linked List is empty\n");
    }
    else
    {
        if (head->link != NULL)
        {
            while (temp->link->link != NULL)
            {
                temp = temp->link;
            }
            temp->link = NULL;
        }
        else
        {
            head = NULL;
        }
    }
}

void del_pos(int pos)
{
    struct node *temp = head;
    // if(pos>length() || pos<1)
    // {
    //  printf("Invalid Position\n");
    // }
    if (pos > 1)
    {

        while (--pos > 1 && temp != NULL)
        {
            temp = temp->link;
        }
```

```
        temp->link = temp->link->link;
    }
    else
    {
        del_beg();
    }
}
int main()
{

    while (1)
    {

        int choice;
        printf("1.Display\n2.Insert at End\n3.Insert at Beginning\n4.Delete
from beginning\n5.Delete from end\n6.Insert into position\n7.Delete from
position\n8.Exit");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
        {
            display();
            break;
        }
        case 2:
        {
            int temp;
            printf("Enter the data");
            scanf("%d", &temp);
            add_end(temp);

            display();

            break;
        }
        case 3:
        {
            int temp;
            printf("Enter the data");
            scanf("%d", &temp);
            add_beg(temp);
            display();
            break;
        }
        case 6:
        {
            int temp, pos;
```

```
            printf("Enter the data and position");
            scanf("%d %d", &temp, &pos);
            add_pos(temp, pos);
            display();
            break;
        }
        case 4:
        {
            del_beg();
            display();
            break;
        }
        case 5:
        {
            del_end();
            display();
            break;
        }
        case 7:
        {
            int pos;
            printf("Enter the position");
            scanf("%d", &pos);
            del_pos(pos);
            display();
            break;
        }

        case 8:
        {
            return 0;
        }
        }
    }
    return 0;
}
```

OUTPUT

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit2

Enter the data12

12

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit2

Enter the data13

12 13

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit3

Enter the data11

11 12 13

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit6

Enter the data and position10 2

11 10 12 13

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit4

10 12 13

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit5

10 12

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit7

Enter the position2

10

1.Display

2.Insert at End

3.Insert at Beginning

4.Delete from beginning

5.Delete from end

6.Insert into position

7.Delete from position

8.Exit8

m/1/22

Stack Using Linked List

Aim

Implement a stack using linked list with the operations. 1. Push elements to the queue. 2. pop elements from the queue 3. Display the queue after each operation.

Algorithm

Step 1: Start

Step 2: Create a structure node with data & link.

Step 3: Create a top pointer an intialize it to null.

Step 4: Declare push(data)

i) temp = top.

ii) Create a newnode newnode->data = data.
newnode->link = NULL.

iii) if (top == NULL) top = newnode.

iv) else new->link = top. top = newnode;

Step 5: Declare pop(). with int return type.

(i) if (top == NULL) print 'Stack empty return -1

else int val = top->data
top = top->link
return val.

Step 6: Declare a display() function.

(i) Create temp=top. until temp != NULL repeat below steps.

(ii) print(temp->data)

(iii) temp = temp->link

Step 7: Declare the main function.

// ~~Repeat steps to be~~ write a menu driven program to call these functions

Step 8: Stop

Result

The program is executed.

Code

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


struct node
{
    int data;
    struct node* link;
};


struct node* top=NULL;


void push(int data)
{
    struct node* temp=top;
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=data;
    newnode->link=NULL;
    if(top==NULL)
    {
        top=newnode;
    }else
    {
        newnode->link=top;
        top=newnode;
    }


}

int pop()
{
    if(top==NULL)
    {
        printf("Stack is empty");
        return -1;
    }else
    {
        int val=top->data;
        top=top->link;
        return val;
    }
}
```

```c
void display()
{
    printf("STACK : ");
    struct node* temp=top;
    while(temp!=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->link;
    }
    printf("\n");
}









int main()
{



    while(1)
    {



        int choice;
        printf("1.Push\n2.Pop\n3.Exit\n");
        scanf("%d",&choice);
        switch(choice)
        {


            case 1:
            {
                int temp;
                printf("Enter the value to push");
                scanf("%d",&temp);
                // add_end(temp);
                push(temp);
                display();
                break;
```

```
            }
            case 2:
            {
                int temp=pop();
                if(temp!=-1)
                {
                    printf("%d Popped\n",temp);
                }
                display();
                break;
            }
            case 3:
            {
                return 0;
            }
        }
    }
    return 0;
}
```

OUTPUT

1.Push

2.Pop

3.Exit

1

Enter the value to push12

STACK : 12

1.Push

2.Pop

3.Exit

1

Enter the value to push13

STACK : 13 12

1.Push

2.Pop

3.Exit

1

Enter the value to push14

STACK : 14 13 12

1.Push

2.Pop

3.Exit

1

Enter the value to push15

STACK : 15 14 13 12

1.Push

2.Pop

3.Exit

2

15 Popped

STACK : 14 13 12

1.Push

2.Pop

3.Exit

2

14 Popped

STACK : 13 12

1.Push

2.Pop

3.Exit

2

13 Popped

STACK : 12

1.Push

2.Pop

3.Exit

2

12 Popped

STACK :

1.Push

2.Pop

3.Exit

2

Stack is emptySTACK :

1.Push

2.Pop

3.Exit

2

Stack is emptySTACK :

1.Push

2.Pop

3.Exit

3

Queue Using Linked List

Aim

Implement a queue using linked list with the operations. 1. Insert an elements to the queue 2. Delete element from the queue 3. Display the queue after each operation.

Algorithm

Step 1: Start

Step 2: Declare a structure node that contains.
1) Int data
2) pointer to struct node, link.

Step 3: Declare variables of node, *front, *rear.

Step 4: declare a function calle enqueue.
1) Allocate memory to temp.
2) Add the data to temp->data.
3) make temp->link to NULL
4) if front == NULL
front = rear = temp.
5) else
rear->link = temp.
rear = temp.

step 5: Inside the function deque.

1) if front == NULL then
display that the queue is empty.

2). else,
temp = front.
front = front1 -> link.
and free the memory of temp.

step 6: Inside the function display.

1) if front == NULL.
print que is empty
2) else temp = front
while (tem != NULL)
display tem -> data
temp = tem1 -> data

step 7: Stop.

Result
The program is Executed an output is verified.

Code

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


struct node
{
    int data;
    struct node* link;
};

struct node* head=NULL;


struct node* front=NULL;
struct node* rear=NULL;


// void display()
// {
//  struct node* temp=head;
//  while(temp!=NULL)
//  {
//      printf("%d ",temp->data);
//      temp=temp->link;
//  }
//  printf("\n");
// }


void enqueue(int data)
{
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=data;
    newnode->link=NULL;
    if(front==NULL || rear==NULL)
    {
        front=rear=newnode;
    }else
    {
        rear->link=newnode;
        rear=newnode;
    }
}

int dequeue()
{
    if(front==NULL || rear==NULL)
```

```
    {
        printf("Queue is empty");
        return -1;
    }else
    {
        int val=front->data;
        front=front->link;
        return val;
    }
}

void display()
{
    printf("QUEUE:\n");
    struct node* temp=front;
    while(temp!=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->link;
    }
    printf("\n");
}

// void enqueue(int data)
// {
//  struct node* newnode=(struct node*)malloc(sizeof(struct node));
//  newnode->data=data;
//  newnode->link=head;
//  head=newnode;
// }




// int dequeue()
// {
//  struct node* temp=head;
//  int val=-1;
//  if(temp==NULL)
//  {
//      printf("The Queue is empty\n");
//  }else
//  {
//      if(head->link!=NULL)
//      {
//          while(temp->link->link!=NULL)
//          {
//              temp=temp->link;
```

```
//          }
//          val=temp->link->data;
//          temp->link=NULL;

//      }else
//      {
//          val=head->data;
//          head=NULL;
//      }
//  }
//  return val;
// }



int main()
{



    while(1)
    {



        int choice;
        printf("1.Enqueue\n2.Dequeue\n3.Exit\n");
        scanf("%d",&choice);
        switch(choice)
        {

            case 1:
            {
                int temp;
                printf("Enter the number to enqueue");
                scanf("%d",&temp);
                enqueue(temp);
                display();
                break;
            }
            case 2:
            {
                int temp=dequeue();
                if(temp!=-1)
                {
                    printf("%d Dequeued\n",temp);
                }
                display();
                break;
```

```
            }
            case 3:
            {
                return 0;
            }
        }
    }
    return 0;
}
```

OUTPUT

1.Enqueue

2.Dequeue

3.Exit

1

Enter the number to enqueue12

QUEUE:

12

1.Enqueue

2.Dequeue

3.Exit

1

Enter the number to enqueue13

QUEUE:

12 13

1.Enqueue

2.Dequeue

3.Exit

1

Enter the number to enqueue14

QUEUE:

12 13 14

1.Enqueue

2.Dequeue

3.Exit

1

Enter the number to enqueue15

QUEUE:

12 13 14 15

1.Enqueue

2.Dequeue

3.Exit

2

12 Dequeued

QUEUE:

13 14 15

1.Enqueue

2.Dequeue

3.Exit

2

13 Dequeued

QUEUE:

14 15

1.Enqueue

2.Dequeue

3.Exit

2

14 Dequeued

QUEUE:

15

1.Enqueue

2.Dequeue

3.Exit

2

15 Dequeued

QUEUE:

1.Enqueue

2.Dequeue

3.Exit

2

Queue is emptyQUEUE:

1.Enqueue

2.Dequeue

3.Exit

3

14/1/22

Reverse a Queue using Stack.

Aim

Write a program to reverse content of que using stack.

Algorithm

step 1: Start

step 2: Define a structure node that contains.
① Int data.
② pointer to structure node, link.

step 3: Declare 3 variables of node *front, *rear, *top.

step 4: ~~Using infinite loop~~ Inside function.
enque that accepts an integer value.
1) Allocate memory for struct node *temp.
2) temp -> data = value
3) temp -> link = NULL
4) if front == NULL
front = rear = temp.

5 else
rear -> link = temp
rear = temp

Step 5) Inside the function deque.

1) if front=NULL display the queue is empty

2) else temp=front.
allocate memory for newnode
newnode->data=temp->data
newnode->link=top.
top=newnode
front=front->link
free the memory of temp.

3)

Step 6) Inside function display

① if front==NULL.
display the queue is empty

② else display the nodes from front until the node is NULL.

Step 7: Inside the function reverse.

① temp=top.

② ~~while temp!=NULL~~
~~pop the elems dequeue.~~

① deque all the elements from the queue. and push it to stack.

② pop all elements from stack and ~~push~~ enque it

③ display the stack

Step 8: Stop.

Code

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


struct node
{
    int data;
    struct node* link;
};

struct node* top=NULL;
struct node* front=NULL;
struct node* rear=NULL;




void display()
{
    struct node* temp=front;
    while(temp!=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->link;
    }
    printf("\n");
}




void enqueue(int data)
{
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=data;
    newnode->link=NULL;
    if(front==NULL || rear==NULL)
    {
        front=rear=newnode;
    }else
    {
        rear->link=newnode;
        rear=newnode;
    }
}
```

```c
int dequeue()
{
    if(front==NULL || rear==NULL)
    {
        printf("Queue is empty");
        return -1;
    }else
    {
        int val=front->data;
        front=front->link;
        return val;
    }
}



void push(int data)
{
    struct node* temp=top;
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=data;
    newnode->link=NULL;
    if(top==NULL)
    {
        top=newnode;
    }else
    {
        newnode->link=top;
        top=newnode;
    }


}

int pop()
{
    if(top==NULL)
    {
        printf("Stack is empty");
        return -1;
    }else
    {
        int val=top->data;
        top=top->link;
```

```
        return val;
    }
}


void reverse()
{
    printf("Reversed Queue  ");
    while(front!=NULL)
    {
        push(dequeue());

    }

    while(top!=NULL)
    {
        enqueue(pop());
    }
    display();
    printf("\n");
}
int main()
{



    while(1)
    {



        int choice;
        printf("1.Enqueue\n2.Dequeue\n3.Reverse the Queue\n4.Exit\n");
        scanf("%d",&choice);
        switch(choice)
        {

            case 1:
            {
                int temp;
                printf("Enter the number to enqueue");
                scanf("%d",&temp);
                enqueue(temp);
                display();
                break;
            }
            case 2:
            {
                int temp=dequeue();
```

```
                if(temp!=-1)
                {
                    printf("%d Dequeued\n",temp);
                }
                display();
                break;
            }
            case 3:
            {
                reverse();
                break;
            }
            case 4:
            {
                return 0;
            }
        }
    }
    return 0;
}
```

OUTPUT

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

1

Enter the number to enqueue12

12

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

1

Enter the number to enqueue13

12 13

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

1

Enter the number to enqueue14

12 13 14

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

1

Enter the number to enqueue15

12 13 14 15

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

1

Enter the number to enqueue16

12 13 14 15 16

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

2

12 Dequeued

13 14 15 16

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

3

Reversed Queue 16 15 14 13

1.Enqueue

2.Dequeue

3.Reverse the Queue

4.Exit

4