DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake
Expt. Title Implementation of Program based on array
Class F.Y. MCA Batch B1 Performed on
Roll No. 32 Expt. No. 1.1 Submitted on
Remarks Returned on

1) Algorithm for list all elements :-

Procedure LIST-All (A, S, n)
This Procedure list all elements of Array A(1:s). 's' is size of array and 'n' is the number of elements currently present in arry list

```
Global integer A(1:S), S, n,
Local integer i
        if n=0, then
            Print ("List is empty")
            return
        else
          for i← 1 to n by +1 do
              Print (A(i))
          repeat
        endif
end LIST-All
```

2) Algorithm for adding element at end :-

Procedure ADD-END (A, S, n, ele)
This procedure add elements 'ele' at the end of an Array
A(1:S). 'S' is the maximum size of array. 'n' is the number of elements currently present in array

```
Global integer A(1:S), S, n
Local integer i
            if  n=S, then
                Print ("List or Array is full.
                return
        end if
            n← n+1
            A(n) ← ele.
```

plete for :
orithm
w Chart
gramme Listing
sults
mments

end ADD - END

3) Adding element at begining :-

```
Procedure ADD_BEG (A, s, n, ele)
This procedure add elements 'ele' at begini
Array A(1:3) 's' is maximum size of arra
is the number of elements
currently present in array
Global integer A(1:s), s, n
Local integer i
integer ele
        if n=s, then
            print ("Array is full.")
            return
        end if
            for i←n to 1 by -1 do
                A(i+1) ← A(i)
        repeat
          A(1) ← ele
          n ← n+1
    end ADD_BEG
```

4) Algorithm for adding element at position :-

```
Procedure ADD_POSITION (A, s, n, ele, pos)
This procedure add a element in an Array A
is the maximum size of array. 'n' is numb
elements currently present in array 'Pos' is
position in array where new
element 'ele' get added
Global integer A(1:s), s, n
Local integer i
integer ele, Pos
        if n=s, then
          print ("Array if full.")
          return
```

Name _____
Expt. Title _____
Class _____ Batch _____ Performed on _____
Roll No. _____ Expt. No. _____ Submitted on _____
Remarks _____ Returned on _____

```
if Pos >= 1 and Pos <= n+1, then
    for i <- n to Pos by -1 do
        A(i+1) <- A(i)
    repeat
    A(Pos) <- ele
else
    Print ("Position is invalid.")
end if
end ADD_POSITION
```

5) Algorithm for delete element from end :-

```
Procedure DEL_END (A, n, ele)
This procedure delete an element 'ele' from
Array 's' is maximum size of array. 'n'
is the number
currently present in array
Global integer A(1:s), s, n
integer ele
    if n=0, then
        Print ("Array is empty!!")
        return (Null)
    end if
    ele <- A(n)
    n <- n-1
    return (ele)
    end if
end DEL_END
```

6) Algorithm for delete element from beginin

Procedure DEL – BEG (A, n, ele)

This procedure delete element 'ele' from begining of Array (1:s) 's' maximum si array. 'n' is number of elements currently present in array

Global integer A(1:S), S, n.

Local integer i

integer ele

```
if n=0, then
   print ("Array is empty;")
   return (NULL)
end if
ele ← A(1)
for i←2 to n by +1 do
    A (i-1) ← A(1)
repeat
n ← n+1
return (ele)
end DEL - BEG
```

7) Algorithm for delete element from positio

Procedure DEL - POSITION (A, n, ele)

This procedure delete element 'de' at a position 'pos' from array A(1:S). 'n' is number of elements currently present in array.

Global integer A (1:S), S, n

Local integer i

integer pos

```
if n=0, then
   Print (" Array is empty.")
   return (Null)
end if
   ele ← A (POS)
```

Assignment No:- 1.1
Title:- Implementation of program based on Arrays.
Name :- NIKHIL MANOHAR DHAKE
Roll no :- 32

```
#include"iostream.h"
#include"conio.h"
class LIST_32
{
private:
        int *A, s, n;
public:
        LIST_32(int);
        void ADD_END(int ele);
        void ADD_BEG(int  ele);
        void ADD_POS(int ele, int pos);
        int DEL_END();
        int DEL_BEG();
        int DEL_POS(int pos);
        void LIST_ALL();
};
void LIST_32::LIST_32(int par)
{
        n = 0; s = par;
        A = new int[s + 1];
}
void LIST_32::ADD_END(int ele)
{
        if (n == s)
        {
                cout << "List is full";
                return;
        }
        n = n + 1;
        A[n] = ele;
}
void LIST_32::ADD_BEG(int ele)
{
        if (n == s)
        {
                cout << "\n list is full" << endl;
                //return;
        }
        else
        {
                for (int i = n; i>=1; i--)
                {
                        A[i + 1] = A[i];
                }
                A[1] = ele;
        }
        n++;
}
void LIST_32::ADD_POS(int ele, int pos)
{
```

```
        if (n == s)
        {
                cout << "\n list is full" << endl;
                return;

        }
        if (pos >= 1 && pos <= n + 1)
        {
                for (int i = n; i <= pos; i--)
                {
                        A[i + 1] = A[i];
                }
                A[pos] = ele;
        }
        else
        {
                cout << "position is invalid";
        }


}
int LIST_32::DEL_END()
{
        int ele;
        if (n == 0)
        {
                cout << "List is empty";
                return NULL;
        }
        else
        {
                ele = A[n];
                n = n - 1;
                return ele;
        }

}
int LIST_32::DEL_BEG()
{
        int ele;
        if (n == 0)
        {
                cout << "list is empty";
                return NULL;
        }
        for (int i = 2; i <= n; i++)
        {
                A[i - 1] = A[i];
        }
        n++;
        return ele;

}
int LIST_32::DEL_POS(int pos)
{
```

```
        int ele;
        if (n == 0)
        {
                cout << "\n List is empty" << endl;
                return NULL;
        }
        ele = A[pos];
        for (int i = pos + 1; i <= n; i++)
        {
                A[i - 1] = A[i];
        }
        n--;
        return ele;
}
void LIST_32::LIST_ALL()
{
        if (n == 0)
        {
                cout << "\n list is empty" << endl;
                return;
        }
        for (int i = 1; i <= n; i++)
        {
                cout << A[i] << " ";
        }
}
void MENU()
{
        int opt, ele, pos, s;
        cout << "Enter the size of the list:";
        cin >> s;
        LIST_32 obj(s);
        do
        {
                cout << "\n\n";
                cout << "\n #_#_#_#_#_#_#_#_#_Choose To_#_#_#_#_#_#_#_#_#";
                cout << "\n\n 1. Add at end :-";
                cout << "\n 2. Add at begining :-";
                cout << "\n 3. Add at position :-";
                cout << "\n 4. Delete from end :-";
                cout << "\n 5. Delete from begining :-";
                cout << "\n 6. Delete from position :-";
                cout << "\n 7. List all elements :-";
                cout << "\n 8. Exit";

                cout << "\n\n Enter your choice :-";


                cin >> opt;
                switch (opt)
                {
                case 1:
                        cout << "\n Enter elements add to end:-";
                        cin >> ele;
                        obj.ADD_END(ele);
```

```
                break;
        case 2:
                cout << "\n Enter elements add at begining:";
                cin >> ele;
                obj.ADD_BEG(ele);
                break;
        case 3:
                cout << "\n Enter elements add at position:";
                cin >> pos;
                obj.ADD_POS(ele, pos);
                break;
        case 4:
                cout << "\n Enter elements delete from end:";
                cin >> ele;
                ele = obj.DEL_END();

                break;
        case 5:
                cout << "\n Enter ele delete from begining:";
                cin >> ele;
                ele = obj.DEL_BEG();
                break;
        case 6:
                cout << "\n Enter elements delete from position:";
                cin >> pos;
                ele = obj.DEL_POS(pos);
                break;
        case 7:

                obj.LIST_ALL();
                break;
        case 8:
                return;
        default:
                cout << "\n Invalid option";
        }

    } while (1);

}
void main()
{
    int ele, obj;
    clrscr();
    MENU();
    getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on STACK
Class F.Y MCA Batch B1 Performed on
Roll No. 32 Expt. No. Submitted on
Remarks Good Returned on

* Write algorithm for Push element into stack.

Procedure - PUSH (A, S, top, ele)
Description :- This Procedure add element 'ele' at the end of the stack 'A'. 'A(1:S) is an array. 'S' is maximum size of array 'A'. 'top' is the last element in stack.

Global integer A(1:S), S, top
integer ele.

if (top == S), then
Print ("stack is full.")
return
end if
top ← top + 1
A(top) ← ele
END PUSH

2) Algorithm for POP/remove element from end of stack

Procedure POP (A, top

This Procedure POP element of the stack A(1:S). 'A' is an linear array. 'top' is the last element Present in a stack 'A'.

lete for :
ithm
Chart
ramme Listing
ılts
ıments

```
Global integer A(1:S),
if (top = 0), then
    Print ("stack is empty.")
    return (NULL)
End if
ele ← A[top]
top ← top - 1
return (ele)
END POP
```

3) Algorithm for display all elements present

Procedure DISPLAY (A, top

This Procedure display all the ele currently present in stack 'A(1:S)'. 'A' is Linear with maximum size 's', 'top' is the last el currently present in stack 'A(1:S)'.

```
Global integer A(1:S), top
Local integer i
    if (top = 0), then
        Print ("stack is empty!")
        return;
    else.
        for i ← 1 to top by +1 do
            Print (A(i))
        repeat
    end if
END DISPLAY
```

Assignment No:- 1.3
Assignment Title:- Implementation of Program based on Stack
Name:-Dhake Nikhil Manohar
Roll.No- 32

```
#include<iostream.h>
#include<conio.h>

class STACK_32
{
     private:
                int *A,s,top;

      public:
                STACK_32(int);
                void PUSH(int ele);
                int POP();
                void LIST_ALL();
;
/---------------------------------
STACK_32::STACK_32(int par)
{
     s=par;
     top=0;
     A=new int[s+1];
}
void STACK_32::PUSH(int ele)
     {
      if(top==s)
      {
            cout<<"\n stack is full" ;
            return;
      }
      top=top+1;
      A[top]=ele;

nt STACK_32::POP()
     {
     if (top==0)
     {
          cout <<"\n stack is empty" ;
          return NULL;
          }
          int ele=A[top];
          top=top-1;
          return ele;


id STACK_32::LIST_ALL()
    {
        if (top==0)
        {
              cout<<"\n stack is empty" ;
        }
        else
        {
              cout<<endl<<"\n element in stack" ;
              for(int i=top;i>=1;i--)
```

Global integer A[1:5]

```
                {
                 cout<<A[i]<<" " ;
                }
            }
        }

void MENU()
{
      int opt,ele,size;
      cout<<"\n enter size of stack" ;
      cin>>size;
      STACK_32 obj (size);
      do
      {
            cout<<"\n 1.PUSH" ;
            cout<<"\n 2.POP"  ;
            cout<<"\n 3.LIST" ;
            cout<<"\n 4.EXIT" ;
            cout<<"\n enter your option" ;
            cin>>opt;
            switch (opt)
            {
                 case 1:
                       cout<<"\n enter ele to add" ;
                       cin>>ele;
                       obj.PUSH (ele);
                       break;

                 case 2:
                       int ele=obj.POP();
                       cout<<endl<<ele<<"is deleted" ;
                       break;

                 case 3:
                       obj.LIST_ALL();
                       break;

                 case 4:return;

            default:cout<<"\n Invalid Option" ;
            }
      }while(1);
}
void main()
{
            int ele;
            clrscr();
            MENU();
            getch();

}
```

4/6/12

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake
Expt. Title write a Program to based on stack using linked list
Class M'CA - 1 Batch B1 Performed on
Roll No. 32 Expt. No. 1.6 Submitted on
Remarks Returned on

1 write a program to list all the elements in stack

Procedure list_All (top, data, next)
Desceiption :- This procedure display all nodes from stack. organized using linked list. where 'top' is a pointer pointing to first node in the list. which is initially set to null. 'Data' is variable part of node which hold the information. 'next' is a pointer pointing to next node in list by storing the address of next node which is set to NULL of the last node in the list. 'Avail List of free nodes.

Declaration :- Global integer data, Pointer top, Pointer next.

Algorithm :-

```
if top = NULL, then
    Print ("Stack is empty")
else
    ptr ← top
    while ptr ≠ NULL. do
        Print (Ptr → data)
        Ptr ← Ptr → next
    repeat.
endif
END list_ALL
```

plete for :
orithm
w Chart
gramme Listing
sults
mments

Assignment NO:-1.6
Title : Implementation of program based on stack using linked list.
Name : Dhake Nikhil Manohar
Roll No:32

```
#include<iostream.h>
#include<conio.h>
class NODE
{
      public:
             int data;
             NODE*next;
};
class STACK
{
     private:
             NODE *top;
     public:
            STACK();
            void PUSH(int ele);
            int POP();
            void LIST_ALL();
};
STACK::STACK()
{
     top=NULL;
}
void STACK::PUSH(int ele)
{
          NODE*NEW=new NODE();
          NEW->data=ele;
          NEW->next=NULL;
          NEW->next=top;
          top=NEW;

int STACK::POP()

        if(top==NULL)
        {
              cout<<"List is empty";
              return NULL;
        }
        else
        {
           int ele=top->data;
           NODE*TEMP=top;
           top=top->next;
           delete TEMP;
           return ele;
        }

void STACK::LIST_ALL()

            cout<<"\n List elements are:";
            if(top==NULL)
            {
                  cout<<"List is empty";
            }
            else
            {
                NODE*ptr;
                ptr=top
                while(ptr!=NULL)
                {
                 cout<<ptr->data<<" ";
                 ptr=ptr->next;
                }
            }
```

11) If there is a left [illegible] push it to the Stack

```
        }
void menu()
{
        int ch,ele;
        STACK obj;
        do
        {
                cout<<"\n1.PUSH";
                cout<<"\n2.POP";
                cout<<"\n3.LIST ALL";
                cout<<"\n4.Exit";
                cout<<"\n Enter your choice:";
                cin>>ch;
                switch(ch)
                {
                     case 1:
                                cout<<"Enter Element to ADD in Stack:";
                                cin>>ele;
                                obj.PUSH(ele);
                                break;
                     case 2:
                                ele=obj.POP();
                                if(ele!=NULL)
                                cout<<ele<<"is deleted";
                                break;
                     case 3:
                                obj.LIST_ALL();
                                break;
                     case 4:
                                return;
                     default:
                                cout<<"Invalid choice";
                }
        }while(1);
}
void main()
{
        clrscr();
        MENU();
        getch();
}
```

ete for :
hm
Chart
amme Li
ts
ients

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation Program based on Infix to Postfix
Class MCA-I Batch B1 Performed on
Roll No. 32 Expt. No. Submitted on
Remarks Returned on

write a algorithm for Converting the infix Expression to the Postfix expression.

Procedure Post_fix (INPOST)
Description:- This Procedure converts an infix expression IN (1:1) into a Postfix expression Post (1:1) using a stack STK (1:1) where 1 is the length of expression

Declaration:-

① Append ')' to given infix expression
② Push '(' on to the stack
③ Pass the infix expression from left to right and repeat following
   a) If the current character is an operand (a,b,c) then append it to the Postfix expression ['Post']
   b) If it is an operator [o] then
      i) check the top of stacks if it is an operator with higher or equal Priority of o then repeatedly, top it and append the same to Postfix expression then Push 'o' the operator on to the stack.
      ii) If there is a left parenthesis '(' on top of stack then Push o it to the stack

iii) If it is left Paranthesis '(' Push it
the stack.
iv) If it is right Paranthesis ')'
a. Repeatedly POP the operator s
append it 'POST' left Paranthesi
b. POP the left Paranthesis

[end of loop step 3)]

4] Exit

END POSTFIX

Assignment No:- 1.3.2

Title :- Implementation of program based on Infix to postfix

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class CONVERT
{
  char infix[89],postfix[89],s[89];
  int i,p,top;
  public:
      CONVERT()
      {
        top=-1;
        i=p=0;
        cout<<"\n Enter Infix Expression:";
        cin>>infix;
        strcat(infix,")");
        s[++top]='(';
      }
      int precedance(char);
      void post();
      void display();
};
int CONVERT::precedance(char ch)
{
  switch(ch)
  {
```

```
        case '^':return 3;
        case '*':return 2;
        case '/':return 2;
        case '+':return 1;
        case '-':return 1;
        default: return 0;
    }
}
void CONVERT::post()
{
   char ch;
   while(top!=-1)
   {
        ch=infix[i++];
        if((ch>='A'&& ch<='Z')|| (ch>='a'&& ch<='z')||(ch>='1'&& ch<='9'))
          postfix[p++]=ch;
        else if(ch=='(')
          s[++top]=ch;
        else if(ch=='+'|| ch=='-'|| ch=='*'|| ch=='/'|| ch=='^')
        {
          while(precedance(ch)<=precedance(s[top]))
          postfix[p++]=s[top--];
          s[++top]=ch;
        }
        else if(ch==')')
        {
          while(s[top]!='(')
          postfix[p++]=s[top--];
          top--;
```

```
        }
        else
        cout<<"\n Wrong string";
    }
    postfix[p]='\0';
}
void CONVERT::display()
{
    cout<<"\n Postfix Expression is
void main()

    clrscr();
    CONVERT c;
    c.post();
    c.display();
    getch();
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Postfix Evaluation
Class MCA-1 Batch B1 Performed on
Roll No. 32 Expt. No. Submitted on
Remarks Returned on

* write algorithm for Postfix evaluation

Procedure POST-EVAL (POST)
Description :- This Procedure evaluates the postfix expression 'POST' is steing form. This Procedure uses 'STK' (1:t) stack to hold
where,
t is length of Postfix expression
The stack is initialize to empty
Declaration :-

step 1 :- Pass the postfix expression 'Pos T' form left to right and repeat following
a) IF it is operand (a, b, c, ... etc),
then
i) Read the value of operand
ii) Push the value on to the stack.

b) If it is operator
i) POP from stack and assign the value to a variable
right ← POP (STK) (TOP)
ii) POP from stack & assign the value to another variable.
left ← POP (STK) (TOP-1)

lete for :
ithm
Chart
ramme Listing
lts
ments

iii) Evaluate the expression left ⊙ right
the result to a variable 'result'
iv) Push the 'result' on the stack.
[endif]
[end of loop]

step 2 :- result ← POP (STK) (TOP)
step 3 :- END

Assignment No:- 1.3.2

Title :- Implementation of program based on Postfix evaluation

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>

const int MAX=89;
class POSTFIX
{
  private:
        int stack[MAX];
        int top,num,ele,data;
        char s[20];
  public:
        POSTFIX();
        void SET_EXPR();
        void PUSH(int ele);
        int POP();
        void CALCULATE();
        void LIST_ALL();
};
POSTFIX::POSTFIX()
{
  top=-1;
}
```

```
void POSTFIX::SET_EXPR()
{
  gets(s);
}
void POSTFIX::PUSH(int ele)
{
   if(top==MAX-1)
   cout<<endl <<"Stack is Full";
   else
   {
         top++;
         stack[top]=ele;
    }
  }
  int POSTFIX::POP()
  {
    if(top==-1)
    {
         cout<<endl <<"Stack is Empty";
         return NULL;
    }
    int data=stack[top];
    top--;
    return data;

  }
  void POSTFIX::CALCULATE()
  {
    int n1,n2,n3;
    int i=0;
    char ch;
```

```
while(s[i]!='\0')
{
    int n=0;
    ch =s[i];
    if(ch==' '|| ch =='\t')
    {
      i++;
      continue;
    }
    if(isdigit (ch))
    {
      while(isdigit(s[i]))
      {
        num=s[i]-'0';
        n=n*10+num;
        i++;
      }
      PUSH(n);
}
else
{
      n1=POP();
      n2=POP();
      switch(ch)
      {
            case '+':n3=n2+n1;
            break;
            case '-':n3=n2-n1;
            break;
            case '/':n3=n2/n1;
            break;
```

nments

```
                case '*':n3=n2*n1;
                break;
                case '%':n3=n2%n1;
                break;
                case '^':n3=pow(n2,n1);
                break;
                default:
                  cout<<" Invalid Choice ";
                exit(1);
            }
            PUSH(n3);
        }
        i++;
      }
    }
    void POSTFIX::LIST_ALL()
    {
      num=POP();
      cout<<"\n Result is: "<< num;
    }
    void main()
    {
      clrscr();
      cout<<"\n Enter Postfix expression to be evaluated :";
      POSTFIX p;
      p.SET_EXPR();
      p.CALCULATE();
      p.LIST_ALL();
      getch();
    }
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation algorithm based on Simple Queue
Class F.Y.M.C.A Batch B.1 Performed on
Roll No. 32 Expt. No. 1.4.1 Submitted on
Remarks Good Returned on

* write a algorithm based on Simple Queue

1) Algorithm for add element at the end of Queue

Procedure Queue_ADD (A, front, rear, size, ele)
This Procedure adds the new element in a simple Queue 'A'. 'A' is a linear array 'A(1:s)'. 'size' is the maximum capacity of Queue. 'rear' and 'front' is the last element and first element are currently Present in Queue respectively. 'ele' is an element to be added in Queue

```
Global integer :A (1:size) ,front,rear, size.
integer ele.

if (rear = size), then
    Print ("Queue is full.")
    return
endif
if (front = 0), then
    f ← 1
end if
rear ← rear + 1
A[rear] ← ele
END Queue_ADD
```

plete for :
orithm
w Chart
gramme Listing
sults
mments

2) Algorithm for delete element of a Queue

Procedure QUEUE-DEL (A, front, rear)

This Procedure delete an element from A Queue is maintain using a linear array (1:S 'Size' is a maximum capacity of Queue. 'front 'rear' is are pointer pointing to front an element in queue respectively.

```
Global int A, front, rear

    if (front = 0), then
        Print ("Queue is Empty.")
        return NULL
    end if
    ele ← A[front]
    if (front = rear), then
        front ← rear ← 0
    else
        front ← front + 1
    end if
END QUEUE-DEL
```

3) Algorithm for display all elements currend in Queue

Procedure QUEUE-DISPLAY (A, front, rear)

This Procedure display all elements queue. 'A' is linear array (1:S) containing elem Segments front to rear. 'front' and 'rear' are Pointing to first and last element in queue re

```
Global int A(1:size), front, rear
Local int i
    if (front = 0), then
        Print ("Queue is Empty.")
        return
    endif
    for i ← front to rear by +1 do
        Print (A(i))
    repeat
END QUEUE_DISPLAY
```

Assignment No:- 1.4
Assignment Title:- Implementation of Program based on queue
Name:- Dhake Nikhil Manohar
Roll.No-32

```
#include<iostream.h>
#include<conio.h>
class QUEUE_32
{
      private:
           int *A,size,front,rear;
      public:
          QUEUE_32(int);
          void insert(int);
          int del();
          void listall();
};
QUEUE_32::QUEUE_32(int par)
{
      size=par;
      A=new int[size+1];
      front=rear=0;
}
void QUEUE_32::insert(int ele)
{
      if(rear==size)
      {
         cout<<"Queue is full"<<endl;
         return;
      }
      if(front==0)
         front++;
      rear++;
      A[rear]=ele;
}
int QUEUE_32::del()
{
     if(front==0)
      cout<<"Queue is empty"<<endl;
     int ele=A[front];
     if(front==rear)
      front=rear=0;
     else
      front++;
     return ele;
}
void QUEUE_32::listall()
{
     if(front==0)
      cout<<"Queue is empty"<<endl;
     else
      for(int i=front;i<=rear;i++)
            cout<<A[i]<<"\t";
}
void MENU()
{
    int ch,size,ele;
    cout<<"Enter size of queue"<<endl;
```

2) Algorithm for

```
cin>>size;
QUEUE_32 obj(size);
do
{
  cout<<"\n1.Insert\n2.Delete\n3.List All\n4.Exit"<<endl;
  cin>>ch;
  switch(ch)
  {
     case 1: cout<<"Enter element in queue"<<endl;
           cin>>ele;
           obj.insert(ele);
           break;

      case 2:
           cout<<obj.del()<<" is deleted"<<endl;
           break;

      case 3: obj.listall();
           break;

      case 4: return;

      default:cout<<"Invalid case"<<endl;
  }
 }while(1);
}
void main()
{
      clrscr();
  MENU();
  getch();
}
```

4/6/22

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title write algorithm based on circular Queue
Class FY MCA Batch B1 Performed on
Roll No. 32 Expt. No. 1.4.1 Submitted on
Remarks Returned on

* write a algorithm based on circular queue.

1) Algorithm for add element in a QUEUE

Procedure QUEUE_ADD (A, front, rear, size, ele)

This Procedure add new element 'ele' in a queue using array (1:s) in circular queue. 'A' is linear array A (1:size). 'front' and 'rear' are the pointer to pointing first and last element in queue respectively. 'size' is maximum capacity of Array.

```
Global int A(1:size), front, rear, size
int ele.
    if (front = 1 && rear = size) || (rear+1 =
                                    front), then
        print ("Queue is full.")
        return
    else
        if (front = 0), then
            front ← front + 1
        end if
        if (rear = size), then
                rear ← 0
        end if
        rear ← rear + 1
        A[rear] ← ele
    end if
END QUEUE_ADD
```

ete for :
hm
Chart
mme Listing
s
ents

2) Algorithm for delete element from a Cir

Procedure QUEUE - DEL (A, front, rear,
This Procedure delete element from queue. a circular queue in maintaining using array A(1:size) where 'size' is a maximu of a queue. 'front' and 'rear' are the Poin Pointing the first and last elements present respectively.

```
Global int A(1:size), front, rear, si
    If (front = 0), then
    Print: ("Queue is Empty.")
    return NULL
  else
      ele ← A[front]
      if (front = rear), then
          front ← rear ← 0
    else
      if (front = size), then
          front ← 0
      endif
      front ← front + 1
    end if
    return ele
  end if
END QUEUE - DEL
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation Program based on circular queue.
Class F.Y MCA Batch B1 Performed on
Roll No. 32 Expt. No. 1.4.1 Submitted on
Remarks Returned on

3) Algorithm for display all elements present in a circular queue.

Procedure QUEUE_DISPLAY (A, front, rear, size)
This Procedure display all elements present in a circular queue using maintaing linear array A(1:size) in a circular from containing elements in segments front to rear. 'front' and 'rear' are the pointer pointing the first and last elements in a queue respectively. 'size' is maximum size of queue.

```
Global int A (1: size), front, rear, size
Local int i;
    If (front = 0), then
        Print ("Queue is Empty.")
    else
        if (front ≤ rear), then
            for i ← front to rear by+1 do
                Print (A(i))
            repeat
        else
            for i ← front to size by+1 do
                Print (A[i])
            repeat
        endif
    endif
END QUEUE_DISPLAY
```

ete for :
thm
Chart
amme Listing
ts
nents

Assignment No:-1.4
Assignment Title:- Implementation of Program based on Circular queue
Name:- Dhake Nikhil Manohar
Roll.No-32

```
#include<iostream.h>
#include<conio.h>
class QUEUE_32
{
      private:
                  int *A,size,front,rear;
      public:
                  QUEUE_32(int);
                  void insert(int);
                  int del();
                  void listall();
};
QUEUE_32::QUEUE_32(int par)
{
      size=par;
      front=rear=0;
      A=new int[size+1];
}
void QUEUE_32::insert(int ele)
{
      if((front==1 && rear==size)||(rear+1==front))
      {
            cout<<"Queue is Full"<<endl;
            return;
      }
      if(front==0)
             front=1;
      if(rear==size)
             rear=0;
      rear++;
      A[rear]=ele;
}
int QUEUE_32::del()
{
            if(front==0)
            {
                  cout<<"Queue is Empty"<<endl;
                  return NULL;
            }
            int ele=A[front];
            if(front==rear)
                   front=rear=0;
            else
            {
                  if(front==size)
                              front=0;
                  front++;
            }
            return ele;
}
void QUEUE_32::listall()
{
      int i;
```

rear ← NEW

```
        if(front==0)
        {
                cout<<"Queue is Empty"<<endl;
                return;

        }
        if(front<=rear)
        {
                for(i=front;i<=rear;i++)
                        cout<<A[i]<<"\t";

        }
        else
        {
                for(i=front;i<=size;i++)
                        cout<<A[i]<<"\t";
                for(i=1;i<=rear;i++)
                        cout<<A[i]<<"\t";

        }
}
void MENU()
{
        int ch,size,ele;
        cout<<"Enter size of queue"<<endl;
        cin>>size;
        QUEUE_32 obj(size);
        do
        {
      cout<<"\n1.Insert\n2.Delete\n3.List All\n4.Exit"<<endl;
      cin>>ch;
      switch(ch)
      {
              case 1: cout<<"Enter element in queue"<<endl;
                              cin>>ele;
                              obj.insert(ele);
                              obj.listall();
                              break;

              case 2:
                      cout<<obj.del()<<" is deleted"<<endl;
                      obj.listall();
                      break;

              case 3: obj.listall();
                      break

              case 4: return;

              default:cout<<"Invalid case"<<endl;
      }
            }while(1);
}
void main()
{
      clrscr();
      MENU();
      getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake
Expt. Title Implementation of Program of Queue Using linked list
Class MCA-1 Batch B1 Performed on
Roll No. 32 Expt. No. 1.4.2 Submitted on
Remarks Returned on

write a program to add element in Queue.
Procedure ADD_QUEUE (front, rear, data, next, ele)

Description :- This procedure add a node in using a linked list. front and rear are pointers pointing two nodes at begining & end of Queue which are initially set to null, when Queue is empty. 'data' is a part of node that holds information and 'next' is a part of node that holds address of next node to form a link 'element' 'ele' is an element to be newly added in the queue. 'Avail' is a list of free nodes

Declaration - Global Pointer front, Pointer rear, Pointer next, integer data, Parameter integer ele.

Algorithm :-

```
if AVAIL = NULL, then
    print ("Queue is full")
else
    NEW ← DELETE (Avail)
    NEW → Data ← ele
    NEW → next ← NULL
    if front = NULL then
        front ← NEW
        rear ← NEW
    else
        rear → next ← NEW
        rear ← NEW
    endif
end if
```

END Procedure ADD_QUEUE

ete for :
thm
Chart
amme Listing
ts
nents

write a program for delete the Queue.

Procedure Delete_Queue (front, rear, data, next)

Description :- This Procedure deletes a node Queue using a linked list. front and rear a Pointers to two nodes at begining and end of which are initially set to null, when queue is 'data' is a part of node that holds informa 'next' is a part of node that holds address node to form a link

Declaration :- Global pointer front, Pointer re Pointer next, integer data.
Local Pointer temp

Algorithm :-

```
if front = NULL, then
    Print ("Queue is empty")
    return (NULL)
else.
    ele ← front → data
    Temp ← front
    if front = rear, then
        rear ← NULL
    endif
    front ← front → next
    APP (AVAIL) ← Temp
end if.
```

END Procedure_Delete_Queue.

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake

Expt. Title Implementation of Program based on ~~Queue~~ Queue using linked list

Class MCA - I Batch B1 Performed on

Roll No. 32 Expt. No. 1.4.2 Submitted on

Remarks Returned on

write a program to list all elements from Queue

Procedure List All (front, data, next)

Description :- this procedure Display all nodes from Queue linked list. where, "front" is Pointer Pointing to node at begining. "data" is Part of node that holds information and "next" is a Part of node. that holds address of next node to from the list. "AVAIL" is a list of free Nodes.

Declaration :- Global Pointer front, Pointer next, integer data.
Local Pointer ptr.

Algorithm :-

```
if front = NULL, then
    Print ("Queue is Empty")
else
    Ptr ← front
    while Ptr ≠ NULL, then
        Ptr (Ptr → data1)
    repeat
endif.
```

END List - All

or :

e Listing

Assignment NO:-1.6
Title : Implementation of program based on queue using linked list.
Name : Dhake Nikhil Manohar
Roll No:32

```
#include<iostream.h>
#include<conio.h>
class NODE
{
      public:
           int data;
           NODE *next;
};
class QUEUE
{
      private:
            NODE *front,*rear;
      public:
            QUEUE();
            void ADD_QUEUE(int ele);
            int DELETE_QUEUE();
            void LIST_ALL();
;
oid QUEUE::QUEUE()


     front=rear=NULL;

oid QUEUE::ADD_QUEUE(int ele)

     NODE*NEW=new NODE();
     NEW ->data=ele;
     NEW ->next=NULL;
     if(NEW==NULL)
     {
         cout<<"Queue is full";
         return;
     }
     if(front==NULL)
     {
         front=NEW;
         rear=NEW;
     }
     else
         rear->next=NEW;
         rear=NEW;
     }
t QUEUE::DELETE_QUEUE()

    if(front==NULL)
    {
        cout<<"QUEUE is Empt.";
        return NULL;
    }
    int ele=front->data;
    NODE*TEMP=front;
    if(front==rear)
        front=rear=NULL;
```

```
            else
               front=front->next;
               delete TEMP;
               return ele;
    }
    void QUEUE::LIST_ALL()
    {
          if(front==NULL)
          {
              cout<<"Queue is Empty.";
          }
          NODE*ptr;
          ptr=front;
          while(ptr!=NULL)
          {
              cout<<ptr->data<<" ";
              ptr=ptr->next;
          }
    }
    void MENU()
    {
          int ch,ele;
          QUEUE obj;
          do
          {
              cout<<"\n1.ADD QUEUE";
              cout<<"\n2.DELETE QUEUE";
              cout<<"\n3.LIST ALL";
              cout<<"\n4.EXIT";
              cout<<"\n Enter your choice:";
              cin>>ch;
              switch(ch)
              {
                 case 1:
                      cout<<"Enter Element to Add:";
                      cin>>ele;
                      obj.ADD_QUEUE(ele);
                      break;
                 case 2:
                      ele=obj.DELETE_QUEUE();
                      if(ele!=NULL)
                      {
                          cout<<ele<<"is deleted";
                      }
                      break;
                 case 3:
                      obj.LIST_ALL();
                      break;
                 case 4:
                      return;
                 default:
                      cout<<"\n Invalid choice";
              }
          }
          while(1);
    }
    void main()
    {
```

```
        clrscr();
        MENU();
        getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on dequeue
Class FY MCA Batch B1 Performed on
Roll No. 32 Expt. No. 1.5 Submitted on
Remarks Returned on

write algorithm for ADD NEW element from the end of the queue (Dqueue)

Procedure ADD-END (A, right, left, size, ele)
Desciption :- this Procedure added new element in double ended queue. where 'A' is a linear array A(1:size): size is a the maximum capacity of dequeue 'ele' is the element which to be newly added in the dqueue 'Right & 'left' are the pointers. Pointing to last & first element currently Present in the dequeue respectively

Declaration :- Global integer A(1:size) int size
integer Pointer Right
integer pointer left
integer ele.

```
if (Right = size), then
    Print ("Dequeue is full")
else
    if (left =0) then
        left ← 1
    endif
    Right ← Right+1
    A[Right] ← ele
endif
END ADD-END
```

mplete for :
gorithm
ow Chart
ogramme Listing
esults
omments

2) write algorithm for delete element from the end of dequeue

Procedure DEL-END (A, right, left, size)

Description:- This procedure an element from last position in a double ended queue. A Dequeue is maintain using a linear array. A(1:size) size is the maximum capacity of dequeue. 'left' and 'right' are pointers pointing to first & left element present in a double ended queue (Dequeue). respectively.

Declaration:-

```
Global integer A(1:size), size
integer Pointer right.
integer Pointer left
if (left = 0) then
   Print ("Dequeue is Empty")
   return ele
endif
   ele ← A (right)
   if (left = right ≠) then
      left ← right ← NULL
   else
      right ← right - 1
   endif
      return ele
END DEL-END
```

3] write algorithm for add element at the begining of dequeue

Procedure ADD-BEG (A, left, Right, size, ele)

Description:- this Procedure added new element at beginning of dequeue. where A is the linear array A(1:size). 'size' is maximum capacity of dequeue. 'ele' is the element which to be added in dequeue 'left' and right are pointers. Pointing to first and left element of dequeue resp.

Declaration :- Global int A, size
integer Pointer left, Pointer right.
Parameter integer ele.

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on Dequeue.
Class FY MCA Batch B1 Performed on
Roll No. 32 Expt. No. 1.5 Submitted on
Remarks Returned on

```
if (right = size & (left = 1) then
    Print ("queue is full")
else
    if (left = 0), then
        left ← left + 1
    endif
    if (left > 1) then
        left ← left - 1
    else
        for (i ← right to 1 by -1 do)
            A [i+1] ← A [i]
        repeat.
            right ← right - 1
        endif.
        A [left] ← ele.
    endif
END ADD_BEG
```

4] Write algorithm for delete Element from begining of double ended dequeue.

Procedure DEL_BEG (A, left, right, size)

Description :- This Procedure delet an element from begining of dequeue A dequeue is maintain using linear array A(1:size) where 'size' is maximum capacity of dequeue 'left & 'right' are Pointers pointing to first & left element of the double ended queue resp

Declaration :- Global integer, A, size
integer Pointer left
integer Pointer right

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Assignment No :- 1.5
Title :- Implementation of program base on Dequeue using array
Name : Dhake Nikhil Manohar
Roll No: 32

```
#include<iostream.h>
#include<conio.h>
class DQUEUE
{
     private:
          int *A;
          int left,right,size;
     public:
          DQUEUE(int);
          void ADD_FIRST(int);
          void ADD_END(int);
          int DEL_FIRST();
          int DEL_END();
          void LISTALL();
};
DQUEUE::DQUEUE(int par)
{
     size=par;
     left=right=0;
     A=new int[size+1];
}
void DQUEUE::ADD_FIRST(int ele)
{
     if(left==1 && right==size)
          cout<<"Queue is full"<<endl;
     else
     {
        if(left==right)
            left=1;
        else if(left>1)
          left=left-1;
        else
        {
          for(int i=right;i>=1;i--)
               A[i+1]=A[i];
          right=right+1;
        }
        A[left]=ele;
     }
}
void DQUEUE::ADD_END(int ele)
{
     if(left==1 && right==size)
          cout<<"Queue is full"<<endl;
     else
     {
```

```
            if(left==0)
                left=1;
            right=right+1;
            A[right]=ele;
        }
}
int DQUEUE::DEL_FIRST()
{
        if(left==0)
        {
            cout<<"list is empty";
            return NULL;
        }
        int ele=A[left];
        if(left==right)
            left=right=0;
        else
            left=left+1;
        return ele;

}
int DQUEUE::DEL_END()
{
        if(left==0)
        {
            cout<<"Queue is empty"<<endl;
            return NULL;
        }
        int ele=A[right];
        if(left==right)
            left=right=0;
        else
            right=right-1;
        return ele;
}
void DQUEUE::LISTALL()
{
        if(left==0)
            cout<<"queue is full"<<endl;
        else
        {
            for(int i=left;i<=right;i++)
                cout<<A[i]<<" ";
        }
}
void MENU()
{
        int ch,ele,size;
        cout<<"Enter size";
        cin>>size;
        DQUEUE obj(size);
```

```
        do
        {
              cout<<"\n1.Add_FIRST";
              cout<<"\n2.ADD_END";
              cout<<"\n3.DEL_END";
              cout<<"\n4.DEL_FRONT";
              cout<<"\n5.LISTALL";
              cout<<"\n6.Exit";
              cout<<"\nEnter your choice";
              cin>>ch;
              switch(ch)
              {
                    case 1: cout<<"Enter element"<<endl;
                          cin>>ele;
                          obj.ADD_FIRST(ele);
                          break;
                    case 2: cout<<"Enter element"<<endl;
                          cin>>ele;
                          obj.ADD_END(ele);
                          break;
                    case 3: ele=obj.DEL_END();
                          if(ele!=NULL)
                                cout<<ele<<" is deleted"<<endl;
                          break;
                    case 4: ele=obj.DEL_FIRST();
                          if(ele!=NULL)
                                cout<<ele<<" is deleted"<<endl;
                          break;
                    case 5: obj.LISTALL();
                          break;
                    case 6:
                          return;
                    default:cout<<"Invalid Case"<<endl;
              }
        }while(1);
}
void main()
{
        clrscr();
        MENU();
        getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake
Expt. Title Implementation of Program based on Singly linked list
Class MCA - 1 Batch B1 Performed on
Roll No. 32 Expt. No. 1.6 Submitted on
Remarks Returned on

1) write a Program to add element at begin in linked list Procedure ADD-first (start, data, next, de)

Discription :- This procedure inserts a new node at the first Position in the linked list 'start' is a pointer pointing to the first node in the list which is initialy set to NULL 'data' is a variable to hold / store the information at each node. 'next' is a Pointer Pointing to next node in the list. It stores address of next node to Point it. 'AVAIL' is a list of free nodes which is said to NULL if no pronode is available. The next Pointer will hold null value in the last node of list.

Declaration :- Global Pointer start
int data, Pointer next
Parameter integer ele.

Algorithm :-

```
if Avail - NULL, then
    Print ("list is full")
else
    step 1. (reate New Node.
    New ← DELETE (Avail)
    NEW → data ← ele
    NEW → next ← NULL
    NEW → next ← start
    Start ← NEW
endif
END ADD_first
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

2) write program to delete an first element from list

Procedure delete_first (start, data, next,)

Description :- This procedure deletes a node from a linked list from the first location. 'start' is a Pointer Pointing to the first node in the list which is initially said to null. 'Data' is a part of node which holds the information. 'next' is a pointer pointing to the next node in the list by storing the address of next node which is said to be null in case of last node. 'Avail' is a list of free nodes.

Declaration :- Global Pointer start, Pointer next,
integer data,
local - integer ele, Pointer temp

Algorithm :-

```
if start = NULL, then
    Print ("list is empty")
    return (NULL)
else
    ele ← start → data
    Temp ← start
    start ← start → next
    ADD (AVAIL) ← Temp
    return (ele)
end if
```

END Delete - first

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake

Expt. Title Implementation of Program based on Singly linked list

Class MCA -1 Batch B1 Performed on

Roll No. 32 Expt. No. 16 Submitted on

Remarks Returned on

write a program to add element end in linked list

Procedure ADD_End (start, data, next, ele)

Description :- This procedure inserts a new node at the end i.e last position in the linked list. 'start' is a pointer Pointing to the first node in the list which is initialy set to NULL. 'data' is a variable to hold the information at each node. 'next' is a pointer pointing to next node in the list. It stores address of next node to point it. The next Pointer will hold null value in the last node of list. 'Avail' is a list of free nodes which is set to ~~NULL~~ null if no prenode is available.

Declaration :- Global - Pointer start
int data, Pointer next
Parameter integer ele.

Algorithm :-

```
if Avail = NULL
    | Print ("list is full")
else
    | // Create new node
      NEW ← Delete (Avail)
    // set the links
if start = NULL, then
    | start ← New
else
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

```
        ptr ← start
        while ptr → next ≠ null do
              ptr ← ptr → next
        repeat
              ptr → next ← new
        endif.
End Procedure ADD-END.
```

2) Procedure DELETE-END (start, data, next)

Description :- this procedure deletes a node from a linked list from last location. 'start' is a pointer pointing to the first node in the list which is initially set to null. 'Data' is a part of node which holds the information. 'Next' is a pointer pointing to the next node in the first by storing the address of next node, which is set to null in case of last node. Avail is a list of free nodes.

Declaration :- Global - integer data, Pointer next, Pointer start
local - integer ele, pointer temp

```
if start = null, then
| Print ("list is empty")
else
    ptr 1 ← start;
    ptr ← NULL
    while ptr 1 → next ≠ NULL, do
    |     ptr 2 ← ptr 1
    |     ptr 1 ← ptr 1 → next
    repeat
    ele ← ptr 1 → data.
    Temp. ← ptr 1
    if ptr 2 ≠ NULL, then
    | ptr 2 → next ← null
    else.
          start ← x.
    endif.
    ADD (Temp, AVAIL)
    return (ele)
endif
DELETE
END Procedure DELETE-END.
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake
Expt. Title Write a program to based on Stack using linked list
Class MCA -1 Batch B1 Performed on
Roll No. 32 Expt. No. 1.6 Submitted on
Remarks Returned on

write algorithm to perform PUSH operation in linked list.

Procedure PUSH (top, data, next, ele)

Description - This procedure add new node in stack using linked list. 'top' is the Pointer to pointing to the First node in the list which is initially set NULL, 'data' is a variable to hold the information to each node. 'next' is Pointer Pointing to the next node in the list its store the address of next node to Point it. The 'next' Pointer hold the null value in last node of the list. 'AVAIL' is a list of free nodes which is set to null, if no free node is available 'ele' is element which is ddd to be.

Declaration :- Global Pointer top, Pointer next
integer data.
Parameter integer ele.

Algorithm -

```
AVAIL = NULL, then
Print ("stack is full")
else
    NEW ← DELETE (AVAIL)
    NEW → data ← ele
    NEW → next ← NULL
    NEW → next ← top
    top ← NEW
endif
END_PUSH
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

2) write algorithm to perform PoP operation in linked list

Procedure PoP (top, data, next)

Description :- This procedure delete node from stack organized using linked list from the first position. 'top' is a pointer pointing to the first node in the list which is initially set to NULL. 'Data' is a variable part of node which hold the information. 'next' is a pointer pointing to the next node in the list by storing the address of next which is set to null of the last node in list. 'Avail' is a list of free nodes.

Declaration :- Global Pointer top, Pointer next
Parameter integer ele.
Local Pointer temp.

Algorithm :-

```
if top = NULL, then
    Print ("stack is empty")
    return (NULL)
else
    ele ← top → data
    temp ← top
    start ← top → next
    ADD (AVAIL) ← temp
    return (ele)
end if
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Nikhil Manohar Dhake.
Expt. Title Implementation of Program based on singly linked list
Class MCA-1 Batch B1 Performed on
Roll No. 32 Expt. No. Submitted on
Remarks Returned on

Procedure ADD-POS (start, data, next, ele, Pos)

Description :- This procedure adds an new node at the given particular position of list. 'start' is a pointer pointing to the first node. in the list. which is initially set to null. 'data' is a variable to store the information at each node 'next' is a pointer pointing to next node in the list. It stores address of next node. to point it the next pointer will hold Snull value in the last node. list. 'Pos' is a position where new node to be add.

Declaration :- Global - Pointer start, Pointer next.
int data, Pointer - integer Pos.
integer ele.

```
if Avail := NULL, then
|   Print ("list is full")
else
|   NEW ← DELETE (Avail)
|   NEW → data ← ele.
|   NEW → next ← NULL
|   if Pos = 1, then
|   |   NEW → Next ← start
|   |   start ← NEW
|   else.
|       ptr1 ← start; ptr2 ← NULL
|       Count = 1
|       while Count < Pos, do
|           ptr2 ← ptr1
|           ptr1 ← ptr1 ← next
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

```
        Count ← count + 1
    repeat
        New → next ← ptr 1
        Ptr2 → next ← NEW
    endif.
endif
END Procedure ADD-POS
```

2) procedure DELETE-POS (start, data, next, Pos)

Description :- This procedure deletes node from particular Position from an linked list. 'start' is a pointer, Pointing to the first node in the list, which is initially said to null. 'Data' is a part of node. which holds the information. 'next' is a pointer to Pointing to the next node in the list by storing the address of next node. which is said to null in case of last node. 'Pos' is a position of on node. where new node will added

Declaration:- Global - int data, next, Pointer start.
local - integer ele. Pointer temp.

```
if start = Null, then
    Print ("list is empty")
    return NULL.
else.
    // search node to delete
    if Pos = 1, then
        ele ← start → data
        Temp ← start
        start ← start → data
    else
        Ptr1 ← start ; Ptr2 ← Null; count = 1
        while count < Pos, do
            Ptr2 ← Ptr 1
            Ptr1 ← Ptr → next.
            Count ← count + 1.
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name _____

Expt. Title _____

Class _____ Batch _____ Performed on _____

Roll No. _____ Expt. No. _____ Submitted on _____

Remarks _____ Returned on _____

```
        repeat
          ele ← ptr → data
          temp ← ptr 1
          ptr 2 → next ← ptr 1 → next
        endif
          ADD (AVIL) ← temp
          return (ele)
    endif
END Procedure Delete_Pos
```

Incomplete for :

1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on singly linked list
Class MCA-I Batch B1 Performed on
Roll No. 32 Expt. No. 1.6 Submitted on
Remarks Returned on

write a Program to Display all the elements from list

Procedure list-ALL (start, data, next)

Description :- This procedure display all nodes from linked list where, 'start' is a pointer Pointing to first node in the list which is initially set to be NULL. 'data' is variable Part of node which hold the information 'next' is a Pointer Pointing to the next node in list by storing the address of next node. which is set to NULL of the last node in the list. 'Avail' is a list of free nodes.

Declaration :- Global integer data, Pointer start
Pointer next

```
if start = NULL, then
|  Print ("list is empty")
else
Ptr ← start
    while Ptr ≠ NULL, do
    |  Print (Ptr → data)
    |  Ptr ← Ptr → next
    repeat
end if
```

END List_ALL

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Assignment No:-1.6
Assignment Name:- Implementation of Program based on Linear Linked List
Name :- Dhake Nikhil Manohar
Roll No :- 32

```
#include<iostream.h>
#include<conio.h>
class NODE
{
        public:
                NODE *next;
                int data;
};
class LIST
{
        private:
                NODE *start;
        public:
                LIST();
                void ADD_FIRST(int);
                void ADD_END(int);
                int DELETE_FIRST();
                int DELETE_END();
                void ADD_POS(int,int);
                int DELETE_POS(int);
                void LISTALL();
};
LIST::LIST()
{
        start=NULL;
}
void LIST::ADD_FIRST(int ele)
{
   //create New Node
   NODE *NEW =new NODE();
   //Populate New Node
   NEW->data=ele;
   NEW->next=NULL;
   //set the Links
   NEW->next=start;
   start=NEW;
}
void LIST_::ADD_END(int ele)
{
        NODE *NEW = new NODE();
        NEW->data=ele;
        NEW->next=NULL;
```

```
        //Set the Links
        if (start==NULL)
                start=NEW;
        else
        {
                NODE *ptr;
                ptr=start;
                while(ptr->next!=NULL)
                        ptr=ptr->next;
                ptr->next=NEW;
        }
}
int LIST::DELETE_FIRST()
{
        if(start==NULL)
        {
                cout<<"List is empty"<<endl;
                return NULL;
        }
        int ele=start->data;
        NODE *TEMP;
        TEMP=start;
        start=start->next;
        delete TEMP;
        return ele;
}
int LIST::DELETE_END()
{
        if(start==NULL)
        {
                cout<<"List is empty"<<endl;
                return NULL;
        }
        NODE *ptr1=start;
        NODE *ptr2=NULL;
        while(ptr1->next!=NULL)
        {
                ptr2=ptr1;
                ptr1=ptr1->next;
        }
        int ele=ptr1->data;
        NODE *TEMP=ptr1;
        delete TEMP;
        if(ptr2!=NULL)
                ptr2->next=NULL;
        else
                start=NULL;
```

```
        return ele;
}
void LIST::ADD_POS(int pos,intele)
{
        //create Node
        NODE *NEW = new NODE();
        //Populate Node
        NEW->data=ele;
        NEW->next=NULL;
        //Set Links
        if(pos==1)
        {
                NEW->next=start;
                start=NEW;
        }
        else
        {
         NODE *ptr1=start;
         NODE *ptr2=NULL;
         int count=1;
         while(count<pos)
         {
                ptr2=ptr1;
                ptr1=ptr1->next;
                count=count+1;
         }
         NEW->next=ptr1;
         ptr2->next=NEW;
        }

}
int LIST::DELETE_POS(int pos)
{
        NODE *TEMP;
        int ele;
        if(start==NULL)
        {
                cout<<"List is Empty"<<endl;
                return NULL;
        }
        if(pos==1)
        {
        ele=start->data;
          TEMP=start;
          start=start->next;
        }
        else
```

end → next = null

```
        {
          NODE *ptr1=start;
          NODE *ptr2=NULL;
          int count=1;
          while(count<pos)
          {
                ptr2=ptr1;
                ptr1=ptr1->next;
                count=count+1;
          }
         ele= ptr1->data;
          TEMP=ptr1;
          ptr2->next=ptr1->next;
        }
        delete TEMP;
        return ele;
  }
  void LIST::LISTALL()
  {
        if(start==NULL)
              cout<<"List is Empty"<<endl;
        else
        {
              NODE *ptr;
              ptr=start;
              while(ptr!=NULL)
              {
              cout<<ptr->data<<" ";
              ptr=ptr->next;
              }
       }
}
void MENU()
{
 int ch,ele,pos;
 LIST obj;
 do
 {
      cout<<"\n1.ADD_FIRST";
      cout<<"\n2.ADD_END";
      cout<<"\n3.DELETE_FIRST";
      cout<<"\n4.DELETE_END";
      cout<<"\n5.ADD_POS";
      cout<<"\n6.DELETE_POS";
      cout<<"\n7.LISTALL";
      cout<<"\n8.Exit";
      cout<<"\nEnter your choice"<<endl;
```

```
        cin>>ch;
        switch(ch)
        {
                case 1: cout<<"Enter element"<<endl;
                        cin>>ele;
                        obj.ADD_FIRST(ele);
                        break;
                case 2: cout<<"Enter element"<<endl;
                        cin>>ele;
                        obj.ADD_END(ele);
                        break;
                case 3: ele=obj.DELETE_FIRST();
                        if(ele!=NULL)
                                cout<<ele<<" is deleted"<<endl;
                        break;
                case 4: ele=obj.DELETE_END();
                        if(ele!=NULL)
                                cout<<ele<<" is deleted"<<endl;
                        break;
                case 5: cout<<"Enter Position"<<endl;
                        cin>>pos;
                        cout<<"Enter Element"<<endl;
                        cin>>ele;
                        obj.ADD_POS(pos,ele);
                        break;
                case 6: cout<<"Enter Position"<<endl;
                        cin>>pos;
                        ele=obj.DELETE_POS(pos);
                        if(ele!=NULL)
                                cout<<ele<<" is deleted"<<endl;
                        break;
                case 7: obj.LISTALL();
                        break;
                case 8:
                        return;
                default:cout<<"Invalid Case"<<endl;
        }
}while(1);
}
void main()
{
        clrscr();
        MENU();
        getch();}
```

5) Comments

endif
endif

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on Circular linked list
Class MCA-1 Batch B1 Performed on
Roll No. 32 Expt. No. 1.6.2 Submitted on
Remarks Returned on

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Circular linked list :-

1) Write an algorithm to add node at first position in circular linked list.

Procedure ADD_FIRST (start, ele, end, next, data)

Description :- This Procedure append a new node at the first position in circular linked list. 'start' is a pointer Pointing to the first node in the list. which is initialy set to NULL. 'Data' is a variable to hold the information to each node. 'next' is a pointer pointing to the next node in the list it store address of next node to point it. the 'next' node is not NULL in last node of the list circular link list there is no end.' 'AVAIL' is a list of free nodes. which is set to null. if no free node is available 'ele' is element is add to be

Declaration :-

Global Pointer start, Pointer next, Pointer end, integer data.
parameter integer ele

Algorithm :-

```
if AVAIL = NULL
   Print ("List is Full")
else
   NEW ← DEL (AVAIL)
   NEW → data ← ele
   NEW → next ← NULL
if start = NULL, then
   start ← NEW
   NEW → next ← start
else
   NEW → next ← start
   Start ← NEW
   end → next ← start
endif
endif
END ADD-FIRST
```

e) Write algorithm to add node at last position in circular linked list

Procedure ADD_LAST (start, end, data, next, ele)

Description :- this procedure delete node from last position in circular linked list. 'start' is a pointer Pointing to first node in the list which is initially said to NULL. 'data' is a variable to hold the information of each node. 'next' is a pointer Pointing to next node in the list. its store the address of next node. 'AVAIL' is a list of free node which is set to null if no free node is available 'ele' is an element which is add to be in queue.

Declaration :-

Global Pointer start, Pointer next, Pointer end
integer data.
Parameter int ele

Algorithm :-

```
if AVAIL = NULL
    Print ("list is full");
else
    NEW ← DEL (AVAIL)
    NEW → data ← ele
    NEW → next ← NULL
    if start = x, then
        start ← NEW
        NEW → next ← start
    else
        end → next ← NEW
        NEW → next ← start
    endif
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on circular (linked list)
Class MCA-I Batch B1 Performed on
Roll No. 32 Expt. No. Submitted on
Remarks Returned on

3) Write algorithm for Delete at first in circular linked list

Procedure DEL_BEG (start, end, data, next)
Description :- this procedure delete an element from begining in circular linked list. where 'start' is pointer pointing to first node and it is initially said to be null. 'data' is variable which stores the information. 'next' is a pointer pointing to next node it 'stores' the base address of next node. 'end' is pointer pointing to last node. in circular linked list

Declaration :-
Global :- Pointer start, Pointer end Pointer next
integer data.
Local : Pointer Ptr1, Ptr2

```
if start = NULL, then
    Print ("list is empty")
    return null
endif
if start = end, then
    ele ← start → data
    TEMP ← start.
    start ← end ← NULL
else.
    ele ← start → data.
    TEMP ← start
    start ← start → next.
    end → next ← start
endif
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

ADD (AVAIL) ← TEMP
return ele
END DEL - BEG

4) write algorithm for delete from last of circular linked list

Procedure DEL-LAST (start, next, data, end)
Description: This procedure delete an element or node from last in circular linked list. where 'start' is pointer pointing to first node. initially set to be null 'next' is pointer to ~~first~~ next node and it stores base address of next node. 'data' is variable which stores information 'end' is pointer pointing to last element.

Declaration:
Global: Pointer start, Pointer, next, Pointer end
integer data.

if start = NULL, then
Print ("list is empty")
return NULL.
endif
if
Start = end, then
ele ← start → data
TEMP ← start.
Start ← end ← NULL
else
Ptr1 = start,
Ptr2 = NULL
while Ptr1 → next ! = start, do
Ptr2 ← Ptr1
Ptr1 ← Ptr1 + 1
repeat
ele ← Ptr1 → data.
TEMP ← Ptr1
Ptr2 → next ← start.
Ptr 2 ← end
endif
ADD (AVAIL) ← TEMP

return ele.
END DEL - LAST

5) write algorithm for VIEW-ALL in circular linked list

Procedure VIEW - ALL (start, end, next, data)
Description :- This Procedure List all the elements from circular linked list. where 'start' is Pointer Pointing to first node. initially said to be on null. 'end' is a Pointer Pointing to last node. 'next' is pointer Pointing to next node it stores base address of next node. 'data' is variable which stores information.

Declaration :-
Global Pointer, start Pointer next
Pointer end. integer data.

if start = NULL, then
Print ("list is empty")
else
Ptr = start
while Ptr → next != start, do
Print (Ptr → data)
Ptr ← Ptr → next
repeat
endif Print (Ptr → data)
END VIEW - ALL

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation Program based on Doubly linked list
Class MCA-Ist Batch B1 Performed on
Roll No. 32 Expt. No. 1.6.3 Submitted on
Remarks Returned on

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Algorithm for Doubly linked list

1) Algorithm for add element at begining of doubly linked list

Procedure ADD_first (start, data, Prev, next, data ele
This procedure inserts a new node at the first position in the Doubly linked list. Node is divided into three parts i.e Prev, next and data. 'Prev' is used to store base address of Previous node. 'next' it is used to store base address of next node. 'data' is used to store information of node in form of data. 'prev' and 'next' are both Pointers to Pointing Previous and next node respectively 'next' of last node in Doubly linked list is set to be 'NULL'. 'AVAIL' is the ~~first~~ list of free nodes. 'ele' is the Global node NODE *start, *Next, *Prev, data.

Parameter. int ele
If (AVAIL = NULL), then
Print ("list is full")
else.
NEW ← DEL (AVAIL) // Create node
NEW → Prev ← NULL // Fill the node
NEW → data ← ele
NEW → next ← NULL
NEW → next ← start // Sets the links
if (start ≠ NULL), then
start → Prev ← NEW
endif

start ← NEW
end if
END ADD_FIRST

2) Algorithm for delete the node from the begining of Doubly linked list

Procedure DEL_FIRST (start, data, next, prev)

This Procedure delete node from begining of doubly linked list. Node is dividing into three parts that parts are 'data' and 'Prev' and 'next'. 'Prev' and 'next' are Pointers Pointing to Previous and next node respectively 'Prev' Contains base address of Previous node. 'next' in form of data. 'Prev' of first node of list should be always set to be NULL. 'next' of last node in list should be set to be NULL. ~~'next' of las~~ 'AVAIL' is list of Free nodes. 'start' is pointer Pointing to the first NODE of list.

```
Global  NODE * start, * Prev, * next
Local   NODE * Temp

If (start = NULL), then
    Print (" list is empty ")
    return (NULL),
else.
    ele ← start → data
    Temp ← start
    start ← start → next
    if (start ≠ NULL), then
        Start → Prev ← NULL
    end if
    ADD (AVAIL) ← TEMP
    return ele
end if

END DEL_FIRST
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name _____

Expt. Title _____

Class _____ Batch _____ Performed on _____

Roll No. _____ Expt. No. _____ Submitted on _____

Remarks _____ Returned on _____

3) Algorithm for add element at ending of Doubly Linked list

Procedure ADD_END (start, data, Prev, next, ele)
This procedure adds a new node at the Ending of Doubly linked list. 'NODE' is combination of 'Prev', 'next', 'data'. 'Prev' and 'next' are the Pointers Pointing to the Previous and next node respectively 'Prev' contains base address of previous NODE. 'next' Contains base address of next node. 'Prev' of first NODE in list always of next node. should be NULL. and 'next' of last node in list is set to be NULL. 'data' Contains informations in form of data. 'start' is Pointer Pointing to the first node in list. 'ele' is an element tobe newly added in new node. 'AVAIL' is list of free nodes.

```
Global NODE * start, * Prev, * next, data
Parameter int ele
Local NODE * ptr

If (AVAIL = NULL), then
        Print ("list is full.")
else
        NEW ← DEL (AVAIL)    // Create NODE.

        NEW ↔ data ← ele
        NEW ↔ Prev ← NULL
        NEW → Next ← NULL

        if (start = NULL), then

                start ← NEW
        else.
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

```
        ptr ← start
        while (ptr → next ≠ NULL), do
            ptr ← ptr → next
        repeat

        ptr → next ← NEW
        NEW → Prev ← ptr
    end if

endif

END ADD_END
```

4] Algorithm for delete the node from the ending of Doubly linked list.

Procedure DEL-END (start, data, next, Prev)
This procedure delete NODE from ending of ~~Doubly~~ Doubly linked list. 'NODE' is a combination of 'Prev', 'next' and 'data' 'prev' and 'next' are the pointers pointing to the previous or next respectively. 'prev' and 'next' contains the base address of previous node and next node in form of link respectively. 'start' is a Pointer pointing to first node in list. 'data' is contains the information in form of data the 'next' of last node always set to be NULL. The 'Prev' of first node also set to be NULL. 'AVAIL' is list of free nodes.

```
Global NODE *start, *Prev, *next.
Local NODE *TEMP, *ptr1, *ptr2
If (start = NULL), then
        Print ("list is empty.")
        return (NULL)
else
        ptr ← start ; Ptr ← NULL
        while (ptr1 → next ≠ NULL), do
            ptr2 ← ptr 1
            ptr 1 ← ptr 1 → next
        repeat.
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name _____
Expt. Title _____
Class _____ Batch _____ Performed on _____
Roll No. _____ Expt. No. _____ Submitted on _____
Remarks _____ Returned on _____

```
ele ← ptr1 → data1
TEMP ← ptr1
if (ptr2 = NULL), then
        start ← NULL
else
        ptr2 → next ← NULL
end if
ADD (AVAIL) ← TEMP
return (ele)
end if

END DEL-END
```

5] Algorithm for add element at Particular Position of Doubly linked list. ~~'ele' is an~~

Procedure ADD_POS (start, Prev, next, data, ele, Pos)
This procedure add elements means NODE at given position 'Pos' in the Doubly linked list 'ele' is an element a newly added in node. 'start' is Pointer Pointing to the first node of list. 'Prev' & 'next' are Pointers Pointing to the Previous and next node. respectively. 'Prev' & 'next' Contains base address of Previous and next node in form of links respectively the 'Prev' at First node in list is always set to be NULL. the 'next' of Last Node in list is set to be NULL. 'data' is Part of node Containing information in form of data.
'AVAIL' is List of free nodes.

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

```
Global  NODE * start, * Next, * Prev, data
Parameter  int ele, POS
Local  NODE *Ptr1, *Ptr2. int Count
// Assuming that the given position is valid.
   If (AVAIL = NULL), then
         print ("list is full.")
   else
         NEW ← DEL (AVAIL)   // create Node.
         NEW → data ← ele    // Fill data.
         NEW → next ← NULL
         NEW → Prev ← NULL
         If (start = NULL), then
               start ← NEW.
      else.
         Ptr1 ← start; ptr2 ← NULL; Count ← 1
         while (count < POS), do
               Ptr2 ← Ptr1
               Ptr1 ← Ptr1 → next
               Count ← count + 1
         repeat
         NEW → next ← Ptr1
         if (Ptr1 ≠ NULL), then
               Ptr1 → Prev ← NEW
         endif
            NEW → Prev ← Ptr2
            Ptr2 → next ← new
         endif
      endif
endif

END  ADD_POS
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name _____
Expt. Title _____
Class _____ Batch _____ Performed on _____
Roll No. _____ Expt. No. _____ Submitted on _____
Remarks _____ Returned on _____

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

6) Algorithm for delete node from Particular Position of Doubly linked list.

Procedure DEL_Pos (start, Prev, next, data, Pos)
This Procedure delete node at given position as 'Pos' in the doubly linked list. 'start' is Pointer Pointing to the first node of list. 'Prev' & 'next' are the Pointers Pointing to the Previous and next node respectively. 'Prev' & 'next' contains the base address of Previous and next in form of links respectively. The 'Prev' of first node is set to be NULL The 'next' of last node in is set to be NULL 'AVAIL' is list of free node. 'data' is Part of node contains the information in form data

```
// assuming position is valid.
Global NODE * start, * Prev, * next, data.
Parameter int Pos
Local NODE * TEMP, * Ptr1, * Ptr2, int count

if (start = NULL), then
    Print ("List is empty.")
    return NULL
else.
  if (start → next = NULL), then
        ele ← start → data
        TEMP ← start.
        start ← NULL
else
      if (Pos = 1), then
          ele ← start → data
          TEMP ← start
          start ← start → next
          Start → Prev ← NULL
```

```
        Ptr1 ← start ; Ptr2 ← NULL; count ← 1
        while (count ≤ Pos), do
            Ptr2 ← ptr1
            Ptr1 ← Ptr1 → next
            Count ← Count+1
        repeat
        ele ← Ptr1 → data
        TEMP ← Ptr1
        if (Ptr1 → next ≠ NULL), then
        Ptr → next → Prev ← Ptr2
      endif
    endif
  endif
   ADD (AVAIL) ← TEMP
   Return (ele)
  endif
END DEL_POS
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name _____

Expt. Title _____

Class _____ Batch _____ Performed on _____

Roll No. _____ Expt. No. _____ Submitted on _____

Remarks _____ Returned on _____

7) Algorithm for display all elements in Doubly linked list in forward direction.

Procedure VIEW_FWD (start, data, next, Prev)
This Procedure display all elements currently press in a list in forward direction. 'start' is a pointer pointing to the starting or first node of the list 'NODE' is a combination of 'Prev', 'data' and 'next'. 'Prev' and 'next' are the pointer pointing to the previous and next node respectively. 'Prev' and 'next' stores base address of pointing node in form of links. 'data' contains the informations in form of data.

```
Global NODE * start, * next, * Prev, data
Local  NODE * ptr
    If (start = NULL), then
            Print ("List is empty.")
    else
        ptr ← start
        while (ptr ≠ NULL)
            Print (ptr → data)
            ptr ← ptr → next
        repeat
    endif
END VIEW_FWD
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

8) Algorithm for display all elements in doubly linked list in reverse direction

Procedure VIEW_REV (start, Prev, next, data)

This Procedure display all elements currently present in the list in Reverse direction. 'start' is a pointer Pointing to the first node of List. 'NODE' is a combination of 'Prev', 'next' and 'data' where, 'Prev' and 'next' are the pointers Pointing to the previous and next node respectively. 'Prev' and 'next' stores base address of pointing node in form of links. 'data' Contains the information in form of data.

```
Global NODE *start, *next, *prev, data
Local NODE *ptr.
    If (start = NULL), then
        Print ("List is empty.")
    else
        ptr ← start
        while (ptr → next ≠ NULL), do
            ptr ← ptr → next.
        repeat
        //
        while (ptr ≠ NULL)
            Print (ptr → data)
            ptr ← ptr → Prev
        repeat
    endif
END VIEW_REV
```

Assignment No:- 1.6.3

Title :- Implementation of program based on Doubly Linked List

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
class NODE
{
    public:
        NODE *next,*prev;
        int data;
};
class LIST
{
    private:
        NODE *start;
    public:
        LIST();
        void ADD_FIRST(int);
        int DEL_FIRST();
        void ADD_END(int);
        int DEL_END();
        void ADD_POS(int,int);
        int DEL_POS(int);
        void LISTALL();
};
LIST::LIST()
{
    start=NULL;
}
void LIST::ADD_FIRST(int ele)
{
    //create node
    NODE *NEW = new NODE();
    //Populate node
    NEW->data=ele;
    NEW->next=NULL;
    NEW->prev=NULL;
    //set links
    NEW->next=start;
    if(start!=NULL)
        start->prev=NEW;
    start=NEW;
}
int LIST::DEL_FIRST()
```

```
{
    if(start==NULL)
    {
        cout<<"List is Empty"<<endl;
        return NULL;
    }
    int ele=start->data;
    NODE *TEMP=start;
    start=start->next;
    if(start!=NULL)
        start->prev=NULL;
    delete TEMP;
    return ele;
}
void LIST::ADD_END(int ele)
{
    //create node
    NODE *NEW = new NODE();
    //populate node
    NEW->data=ele;
    NEW->next=NULL;
    NEW->prev=NULL;
    //set links
    if(start==NULL)
        start=NEW;
    else
    {
        NODE *ptr=start;
        while(ptr->next!=NULL)
            ptr=ptr->next;
        ptr->next=NEW;
        NEW->prev=ptr;
    }
}
int LIST::DEL_END()
{
    if(start==NULL)
    {
        cout<<"List is empty"<<endl;
        return NULL;
    }
    NODE *ptr1=start;
    NODE *ptr2=NULL;
    while(ptr1->next!=NULL)
    {
        ptr2=ptr1;
        ptr1=ptr1->next;
    }
    int ele=ptr1->data;
```

```
    NODE *TEMP=ptr1;
    if(ptr2==NULL)
        start=NULL;
    else
        ptr2->next=NULL;
    delete TEMP;
    return ele;

}
void LIST::ADD_POS(int ele,int pos)
{
    //create node
    NODE *NEW = new NODE();
    //populate node
    NEW->data=ele;
    NEW->next=NULL;
    NEW->prev=NULL;
    //set links
    if(start==NULL)
        start=NEW;
    else
    {
        if(pos==1)
        {
          NEW->next=start;
          start->prev=NEW;
          start=NEW;
        }
        else
        {
            NODE *ptr1=start;
            NODE *ptr2=NULL;
            int count=1;
            while(count<pos)
            {
                ptr2=ptr1;
                ptr1=ptr1->next;
                count=count+1;

            }
            NEW->next=ptr1;
            if(ptr1!=NULL)
                ptr1->prev=NEW;
            ptr2->next=NEW;
            NEW->prev=ptr2;

        }
    }
}
int LIST::DEL_POS(int pos)
{
```

```
        NODE *TEMP;
        int ele;
        if(start==NULL)
        {
            cout<<"List is empty"<<endl;
            return NULL;
        }
        else
        {
            if(start->next==NULL)
            {
                ele=start->data;
                TEMP=start;
                start=NULL;
            }
            else
            {
                if(pos==1)
                {
                    ele=start->data;
                    TEMP=start;
                    start=start->next;
                    start->prev=NULL;
                }
                else
                {
                   NODE *ptr1=start;
                   NODE *ptr2=NULL;
                   int count=1;
                   while(count<pos)
                   {
                    ptr2=ptr1;
                    ptr1=ptr1->next;
                    count=count+1;
                   }
                   ele=ptr1->data;
                   TEMP=ptr1;
                   ptr2->next=ptr1->next;
                   if(ptr2->next!=NULL)
                     ptr1->next->prev=ptr2;
                }
            }
            delete TEMP;
            return ele;
        }
    }
    void LIST::LISTALL()
    {
        if(start==NULL)
```

```
        cout<<"List is empty"<<endl;
    else
    {
        NODE *ptr=start;
        while(ptr!=NULL)
        {
            cout<<ptr->data<<" ";
            ptr=ptr->next;
        }
    }
}
void MENU()
{
    int ch,ele,pos;
    LIST obj;
    do
    {
        cout<<"\n1.ADD_FIRST";
        cout<<"\n2.DEL_FIRST";
        cout<<"\n3.ADD_END";
        cout<<"\n4.DEL_END";
        cout<<"\n5.ADD_POS";
        cout<<"\n6.DEL_POS";
        cout<<"\n7.LISTALL";
        cout<<"\n8.EXIT";
        cout<<"\nEnter your choice";
        cin>>ch;
        switch(ch)
        {
            case 1: cout<<"Enter element"<<endl;
                cin>>ele;
                obj.ADD_FIRST(ele);
                break;
            case 2: ele=obj.DEL_FIRST();
                if(ele!=NULL)
                    cout<<ele<<" is deleted"<<endl;
                break;
            case 3: cout<<"Enter element"<<endl;
                cin>>ele;
                obj.ADD_END(ele);
                break;
            case 4: ele=obj.DEL_END();
                if(ele!=NULL)
                    cout<<ele<<" is deleted"<<endl;
                break;
            case 5: cout<<"Enter element"<<endl;
                cin>>ele;
                cout<<"Enter Position"<<endl;
                cin>>pos;
```

```
                obj.ADD_POS(ele,pos);
                break;
            case 6: cout<<"Enter Position to delete"<<endl;
                cin>>pos;
                ele=obj.DEL_POS(pos);
                if(ele!=NULL)
                    cout<<ele<<" is deleted"<<endl;
                break;
            case 7: obj.LISTALL();
                break;
            case 8:
                return;
            default:cout<<"Invalid case"<<endl;
        }
    }while(1);
}
void main()
{
    clrscr();
    MENU();
    getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program of Searching techniques using array (Binary & linear search)
Class MCA - 1st Batch B1 Performed on
Roll No. 32 Expt. No. 4.2 Submitted on
Remarks Returned on

1) Linear Search

Procedure linear_SEARCH (A, n, x)

Description :- this Procedure searches element 'x' in an array A[] ; 'n' is the number of elements Present in an array

Declaration - Global integer A(1:n), n, x
local integer i

Algorithm :-

```
for i=1 to n by 1 do
    if A(i)=x, then
        exit loop
    endif
repeat
if i≤n, then
    return (1)
else
    return (0)
endif
```

END - linear_SEARCH

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

2) Binary search

Procedure - SEARCH (A, n, x, j, low, mid, high)

Description :- Given of array A(1:n) of elements in increasing order. $n \geq 0$, determine if x is present if so, set j such that $x = A(j)$ else $j = 0$

Declaration - Global int A (1:n), n

integer x, j

local integer low, high, mid

Algorithm :-

low ← 1, high ← n

while low ≤ high do

mid ← (low + high)/2

case :

: x < A (mid) : high ← mid - 1

: x > A (mid) : low ← mid + 1

else j ← mid

return

endcase.

repeat

j ← 0

return

Assignment No:-4.2
Assignment Title:-Implementation of Program based on searching techniques using array (Linear Search).
Name:- Dhake Nikhil Manohar
Roll No:- 32

```
#include<iostream.h>
#include<conio.h>
void LINEAR_SEARCH();
void LINEAR_SEARCH(int*arr,int ele,int n)
{
 int count=0;
 for(int i=1;i<=n;i++)
 {
  if(arr[i]==ele)
  {
        count=1;
        cout<<ele<<"found at"<<i+1<<"position"<<endl;
  }
 }
 if(count==0)
        cout<<"Element is not present"<<endl;
}
void main()
{
        clrscr();
        int ele,n;
        int arr[10];
        cout<<"Enter size of an array"<<endl;
        cin>>n;
        cout<<"Enter elements in an array"<<endl;
        for(int i=0;i<n;i++)
        {
```

2) Binary search

```
            cin>>arr[i];
        }
        cout<<"Enter element to search"<<endl;
        cin>>ele;
        LINEAR_SEARCH(arr,ele,n);
        getch();
}
```

Assignment No:-4.2
Assignment Title:-Implementation of Program based on searching techniques using array (Binary Search).
Name:-Dhake Nikhil Manohar
Roll No:- 32

```
#include<iostream.h>
#include<conio.h>
int BINARY_SEARCH(int*arr,int ele,int n);
int BINARY_SEARCH(int*arr,int ele,int n)
{
 int low=1,high=n;
 while(low<=high)
 {
  int mid=(low+high)/2;
  if(ele==mid)
        return 1;
  else if(ele<arr[mid])
        high=mid-1;
  else if(ele<arr[mid])
        low=mid+1;
 }
 return -1;
}
void main()
{
        clrscr();
        int n,ele,arr[10];
        cout<<"Enter size of an array"<<endl;
        cin>>n;
        cout<<"Enter element in an array"<<endl;
```

```
    for(int i=1;i<=n;i++)
    {
        cin>>arr[i];
    }
    cout<<"Enter element to search"<<endl;
    cin>>ele;
    int res=BINARY_SEARCH(arr,ele,n);
    if(res==-1)
        cout<<"Element is not present"<<endl;
    else
        cont<<"Element is present"<<endl;

    getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program for hash Table technique Bubble Sort
Class MCA 1 Batch B1 Performed on
Roll No. 32 Expt. No. 4.3 Submitted on
Remarks Returned on

A. Bubble Sort Algorithm

Procedure Bubble_SORT (A,n)
This Procedure sorts elements 'x' in the array A. 'n' is the number of elements currently present in array A (1:s)

Global integer A (1:s), n
Local integer i, j

for i ← 1 to n-1 do
  for j ← 1 to n-i do
    if A(j) > A (j+1), then
      exchange (A (j). A(j+1))
    endif
  repeat
repeat
END Bubble-SORT

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Assignment No: 4.3

Title: Implementation of program based on Bubble Sort.

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
class BUBBLE_89
{
   private:
        int A[10],i,EXCH,j,n,TEMP;
   public:
        void GET();
        void PROCESS();
        void DISPLAY();
};
void BUBBLE_89::GET()
{
   cout<<"\n Enter the array size :";
   cin>>n;
   cout<<"\n Enter the array element =>";
   for(i=1;i<=n;i++)
   cin>>A[i];
}
void BUBBLE_89::PROCESS()
{
   for(i=1;i<=n-1;i++)
   {
        EXCH=0;
```

5) Comments

```
        for(j=1;j<=n-1;j++)
        {
          if(A[j]>A[j+1])
          {
                TEMP=A[j];
                A[j]=A[j+1];
                A[j+1]=TEMP;
                EXCH=EXCH+1;
          }
        }
    }
}
void BUBBLE_89:DISPLAY()
{
    cout<<"\n The array element are =>";
    for(i=1;i<=n;i++)
     cout<<A[i]<<"\t";
}
void main()
{
    clrscr();
    BUBBLE_89 b;
    b.GET();
    b.DISPLAY();
    b.PROCESS();
    cout<<"\n After sorting: ";
    b.DISPLAY();
    getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program for hash Table, Searching & sorting technique Selection Sort
Class MCA-1 Batch B1 Performed on
Roll No. 32 Expt. No. 4.4 Submitted on
Remarks Returned on

Algorithm for Selection sort

Procedure of SELECTION SORT (A, i, j)

Local integer i, j
integer min, Pos - min

for i ← 1 to n by + 1 do
min ← A(i)
Pos min ← i
for j ← (i + 1) to n by + 1 do
if A(j) < min then
min ← A(j)
Pos min ← j
endif
repeat
Exchange A(i), A(Pos. min)
repeat
end

SELECTION_SORT

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Assignment No: 4.4

Title: Implementation of program based on selection Sort.

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
class SELECTION
{
    private:
          int A[],i,Pos_min,j,n,TEMP;
    public:
     void GET();
     void PROCESS();
     void DISPLAY();
};
void SELECTION::GET()
{
cout<<"\n Enter the array size :";
cin>>n;
cout<<"\n Enter the array element =>";
    for(i=1;i<=n;i++)
cin>>A[i];
)
void SELECTION::PROCESS()
{
    for(i=1;i<=n-1;i++)
    {

          Pos_min=i;
          for(j=i+1;j<=n;j++)
```

```
            {
              if(A[j]<A[Pos_min])
            Pos_min=j;
            }
            if(Pos_min!=i)
            {
              TEMP=A[Pos_min];
              A[Pos_min]=A[i];
              A[i]=TEMP;
            }
        }
    }
    void SELECTION::DISPLAY()
    {
    cout<<"\n array elements are =>";
       for(i=1;i<=n;i++)
    cout<<A[i]<<" ";

    }
    void main()
    {
    clrscr();
       SELECTION s;
   s.GET();
   s.DISPLAY();
   s.PROCESS();
   cout<<"\n After Sorting: ";
  s.DISPLAY();
  getch();
  }
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation Program based on Insertion sort
Class MCA-1st Batch B1 Performed on
Roll No. 32 Expt. No. 4.5 Submitted on
Remarks Returned on

Algorithm for Insertion sort:-
Procedure INSERTION_SORT (A, ele, i, j)
This Procedure inserts the element 'ele' from into the sorted array A(1:s). 'n' is the num elements currently present in array

```
Global integer A(1:s)
Local integer i, j
  integer ele
      for i←2 to n by+1 do
       ele ← A(i)
        for j ← (i-1) to 1 by=1 do
          if ele < A(j)
            A (j+1) ←A(j)
      else
          exit loop
      end if
    repeat
END INSERTION SORT
```

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

Assignment No:- 4.5

Title :-Implementation of program based on Insertion sort

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
class INSERTION
{
   private:
        int A[89],n,i,j,ele,temp,ptr;
   public:
    void GET();
    void PROCESS();
    void DISPLAY();
};
void INSERTION::GET()
{
   cout<<"\n Enter the Range :";
   cin>>n;
   cout<<"\n Element are =>";
   for(i=1;i<=n;i++)
   cin>>A[i];
}
void INSERTION::PROCESS()
{
   for(i=2;i<=n;i++)
   {
        ele=A[i];
```

```
        j=i-1;
        while(ele<A[j])
        {
          A[j+1]=A[j];
          j--;
        }
        A[j+1]=ele;
    }
}
void INSERTION::DISPLAY()
{
    cout<<"\n Sorted element using Insertion sort:\n\t =>";
    for(i=1;i<=n;i++)
    cout<<A[i]<<" ";
}
void main()
{
    clrscr();
    INSERTION s;
    s.GET();
    s.PROCESS();
    s.DISPLAY();
    getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation Program based on Quick sort
Class MCA - 1 Batch B1 Performed on
Roll No. 32 Expt. No. Submitted on
Remarks Returned on

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

★ Write algorithm for quick sort

Procedure QUICK-SORT (P, q)
Description :- This procedure sorts the elements A(P), ... A(q) which resides in the global array A(1:n) into ascending order.
A(n+1) is considered to be defined and must be ≥ all elements in A(P:q):
A(n+1) = +∞

Declaration :-

```
Integer P, q
Global n, A (1:n)
if P<q , then
    j = q+1
    Call Partition (P, j)
    Call QUICK-SORT (P, j-1)
    Call QUICK-SORT (j+1, q)
end if
END QUICK-SORT
```

Procedure PARTITION (m, p)
Description :- within A(m), A(m+1), ... A(p-1) the elements are rearranged in such a way that if initially temp = A(m), then after Completion A(q) = temp for some q between 'm' and (p-1). A(k) <= temp for (m≤k≤q-1) and A(k) > temp for (q<k≤p-1). The final value of temp is q.
Declaration :- Global A (m:p)
integer m, p, ;

```
temp = A(n) ; i = m
loop
    do
        i = i+1
    while A(i) < temp repeat.
    do
        p = p-1
    while A(p) > temp repeat.
    if i<p, then.
        EXCHANGE (A(i), A(p))
    else
        exit, loop
    endif
repeat
A (m) = A(p)
A (p) = temp
END PARTITION
```

Assignment No:- 4.9

Title :- Implementation of program based on Quick Sort

Name : Dhake Nikhil Manohar

Roll No: 32

```
#include<iostream.h>
#include<conio.h>
void QUICK_SORT(int *arr,int low,int high)
{
  int pivot,i,j,temp;
  if(low<high)
  {
    pivot=arr[low];
    i=low+1;
    j=high;
    while(i<j)
    {
      while(arr[i]<pivot && i<=high)
        i++;
      while(arr[j]>pivot && j>=low)
        j--;
      if(i<j)
      {
        temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
      }
}
arr[low]= arr[j];
arr[j]=pivot;
```

```
        QUICK_SORT(arr,low,j-1);
        QUICK_SORT(arr,j+1,high);
    }
}
void main()
{
    clrscr();
    int n,arr[10];
    cout<<"Enter size of an array"<<endl;
    cin>>n;
    cout<<"****Enter elements in an array****"<<endl;
    for(int i=1;i<=n;i++)
    {
        cin>>arr[i];
    }
    QUICK_SORT(arr,1,n);
    cout<<"Array after sort"<<endl;
    for(i=1;i<=n;i++)
    {
        cout<<arr[i]<<" ";
    }
    getch();
}
```

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MANAGEMENT AND RESEARCH, JALGAON

Name Dhake Nikhil Manohar
Expt. Title Implementation of Program based on Sorting technique. (merge Sort)
Class MCA.1 Batch B1 Performed on
Roll No. 32 Expt. No. 4.8 Submitted on
Remarks Returned on

Procedure MERGE-SORT (A, low, high)

Description :- Merge sort algorithm is a sorting algorithm that is considered as an example of the divide & conquer strategy ; Array is initially divided into two equal halves and they are combined in a sorted manner. where A is an Array A(low : high)

Declaration :- integer A, low, high

```
if low < high, then
    mid ← (low + high)/2
    Call merge_sort (A, low, mid)
    Call merge_sort (A, mid+1, high)
    Call merge (A, low, mid, high)
endif
```

END MERGE-SORT

Procedure MERGE (A, low, mid, high)

Description :-
This Procedure merges two sublist A(low : mid) and B(mid+1 : high) if uses Auxitary B (low : high) & sorted

Declaration :- Global A(low : mid) and A(mid+1 : high)
integer A, low, mid high
local integer i, j, K.

Incomplete for :
1) Algorithm
2) Flow Chart
3) Programme Listing
4) Results
5) Comments

```
i ← low
j ← mid+1
k ← low
while i≤mid and j≤high do
    if A(i) < A(j), then
        B(k) ← A(i)
        i ← i+1
    else
        B(k) ← A(i)
        j ← k+1
    end if
repeat
    if i ≤ mid
    while i ≤ mid, do
        B(k) ← A(i)
        i ← i+1
        k ← k+1
    repeat
else
    while j ≤ high, do
        B(k) ← A(j)
        j ← j+1
        k ← k+1
    repeat
endif
for k ← low to high, do
    A(k) ← B(k)
repeat
END MERGE
```

Assignment No :- 4.8
Title: Implementation of program for merge sort.
Name: Dhake Nikhil Manohar.
Roll No : 32

```
#include<iostream.h>
#include<conio.h>
int arr[10];
void MERGE(int low,int mid,int high)
{
        int i=low,j=mid+1,k=low;
        int brr[10];
        while(i<=mid && j<=high)
        {
                if(arr[i]<arr[j])
                {
                   brr[k]=arr[i];
                   i++;
                }
                else
                {
                   brr[k]=arr[j];
                   j++;
                }
                k++;
        }
        while(i<=mid)
        {
           brr[k]=arr[i];
           i++;
           k++;
        }
        while(j<=high)
        {
           brr[k]=arr[j];
           j++;
           k++;
        }
        for(k=low;k<=high;k++)
        {
            arr[k]=brr[k];
        }
}
void MERGE_SORT(int low,int high)
{
    if(low!=high)
    {
```

```
        int mid=(low+high)/2;
        MERGE_SORT(low,mid);
        MERGE_SORT(mid+1,high);
        MERGE(low,mid,high);
    }
}
void main()
{
        clrscr();
        int n;
        cout<<"Enter size of an array"<<endl;
        cin>>n;
        cout<<"Enter elements into array"<<endl;
        for(int i=0;i<n;i++)
        {
          cin>>arr[i];
        }
        MERGE_SORT(0,n-1);
        cout<<"After sort"<<endl;
        for(i=0;i<n;i++)
        {
          cout<<arr[i]<<"\t";
        }
        getch();
}
```