Name: Swarnabh Paul

Section: Y

Roll no: 19CS8122

Assignment no: 3

Questions attempted: a,b,c,d

Question (a)

Code:

```
#include<iostream>
#include<stdbool.h>
using namespace std;
class stack
{
private:
    int top, size;
    int *a;
    bool isempty();
    bool isfull();
    void initialize(int);
    void deconstruct();
public:
    stack(int);
    stack(int,int);
    stack();
    ~stack();
    void push(int);
    int pop();
    void display();
};
void stack::push(int x)
{
    if(isfull())
    {
        cout<<"Stack overflow!!\n";
        return;
    }
    top++;
    a[top]=x;
}
int stack::pop()
{
    if(isempty())
    {
        cout<<"Stack underflow!!\n";
        return -1;
    }
    int x=a[top];
    top--;
    return x;
}
stack::stack(int n)
{
    initialize(n);
    cout<<"Constructed stack of size "<<n<<endl;
}
stack::stack(int n,int x)
{
    initialize(n);
    cout<<"Constructed stack of size "<<n<<endl;
    top++;
```

```
    a[top]=x;
}
stack::stack()
{
    int n=10;
    initialize(n);
    cout<<"Constructed stack of size "<<n<<endl;
}
stack::~stack()
{
    deconstruct();
    cout<<"Destroyed stack of size "<<size<<endl;
}
void stack::display()
{
    if(isempty())
    {
        cout<<"Stack is empty\n";
        return;
    }
    cout<<"Displaying stack from top to bottom:\n";
    for(int i=top;i>=0;i--)
        cout<<a[i]<<' ';
    cout<<endl;
}
bool stack::isempty()
{
    return (top==-1);
}
bool stack::isfull()
{
    return (top==(size-1));
}
void stack::initialize(int n)
{
    a=new int[n];
    top=-1;
    size=n;
}
void stack::deconstruct()
{
    delete []a;
}
int main(void)
{
     stack s1(2);
     stack s2[2]={4,7};
     stack s3[3]={{2,20},{5,89},{3,5}};
     s3[1].display();
     s1.push(10);
     s1.push(20);
     s1.display();
     {
        stack *p=new stack(3,7);
        stack *q=new stack[4];
        (*p).push(10);
        p->display();
```

```
        delete []q;
        delete p;

    }

    s1.push(30);
    cout<<s1.pop()<<" "<<s1.pop()<<endl;
    s1.pop();
    return 0;
}
```

Output:

Constructed stack of size 2
Constructed stack of size 4
Constructed stack of size 7
Constructed stack of size 2
Constructed stack of size 5
Constructed stack of size 3
Displaying stack from top to bottom:
89
Displaying stack from top to bottom:
20 10
Constructed stack of size 3
Constructed stack of size 10
Constructed stack of size 10
Constructed stack of size 10
Constructed stack of size 10
Displaying stack from top to bottom:
10 7
Destroyed stack of size 10
Destroyed stack of size 10
Destroyed stack of size 10
Destroyed stack of size 10
Destroyed stack of size 3
Stack overflow!!
20 10
Stack underflow!!
Destroyed stack of size 3
Destroyed stack of size 5
Destroyed stack of size 2
Destroyed stack of size 7
Destroyed stack of size 4
Destroyed stack of size 2

Question (b)

Code:

```
#include<iostream>
#include<stdbool.h>
using namespace std;
class queue
{
private:
    int size, front, rear;
    int *a;
    bool isempty();
    bool isfull();
    void initialize(int);
    void deconstruct();
public:
    void push(int);
    int pop();
    void display();
    queue(int,int);
    queue(int);
    queue();
    ~queue();
};
bool queue::isempty()
{
    return (rear==-1);
}
bool queue::isfull()
{
    return (((rear+1)%size==front)&&(rear!=-1));
}
void queue::push(int x)
{
    if(isfull())
    {
        cout<<"Queue overflow!!\n";
        return;
    }
    rear=(rear+1)%size;
    a[rear]=x;
}
int queue::pop()
{
    if(isempty())
    {
        cout<<"Queue underflow!!\n";
        return -1;
```

```
    }
    int x=a[front];
    if(front==rear)
    {
        front=0;
        rear=-1;
    }
    else
    {
        front=(front+1)%size;
    }
    return x;
}
void queue::display()
{
    if(isempty())
    {
        cout<<"Queue is empty\n";
        return;
    }
    int i;
    cout<<"Displaying queue: ";
    if(front<=rear)
    {
        for(i=front;i<=rear;i++)
            cout<<a[i]<<' ';
    }
    else
    {
        for(i=front;i<size;i++)
            cout<<a[i]<<' ';
        for(i=0;i<=rear;i++)
            cout<<a[i]<<' ';
    }
    cout<<endl;
}
queue::queue(int n)
{
    initialize(n);
    cout<<"Constructed queue of size "<<n<<endl;
}
queue::queue(int n,int x)
{
    initialize(n);
    cout<<"Constructed queue of size "<<n<<endl;
    rear++;
    a[rear]=x;
}
queue::queue()
{
    int n=10;
    initialize(n);
    cout<<"Constructed queue of size "<<n<<endl;
}
queue::~queue()
{
    deconstruct();
```

```
    cout<<"Destroyed queue of size "<<size<<endl;
}
void queue::initialize(int n)
{
    a=new int[n];
    size=n;
    front=0;
    rear=-1;
}
void queue::deconstruct()
{
    delete []a;
}
int main(void)
{
     queue s1(2);
     queue s2[2]={4,7};
     queue s3[3]={{2,20},{5,89},{3,5}};
     s3[1].display();
     s1.push(10);
     s1.push(20);
     s1.display();
     {
        queue *p=new queue(3,7);
        queue *q=new queue[4];
        (*p).push(10);
        p->display();
        delete []q;
        delete p;

     }

     s1.push(30);
     cout<<s1.pop()<<" "<<s1.pop()<<endl;
     s1.pop();
     return 0;
}
```

Output:

Constructed queue of size 2
Constructed queue of size 4
Constructed queue of size 7
Constructed queue of size 2
Constructed queue of size 5
Constructed queue of size 3
Displaying queue: 89
Displaying queue: 10 20
Constructed queue of size 3
Constructed queue of size 10
Constructed queue of size 10
Constructed queue of size 10

```
Constructed queue of size 10
Displaying queue: 7 10
Destroyed queue of size 10
Destroyed queue of size 10
Destroyed queue of size 10
Destroyed queue of size 10
Destroyed queue of size 3
Queue overflow!!
10 20
Queue underflow!!
Destroyed queue of size 3
Destroyed queue of size 5
Destroyed queue of size 2
Destroyed queue of size 7
Destroyed queue of size 4
Destroyed queue of size 2
```

Question (c)

Code:

```
#include<iostream>
using namespace std;
class matrix
{
private:
    int **a;
    int r, c, maxr, maxc;
    int** allocatespace(int,int);
    void deallocatespace();
    matrix* cofactor(int);
    matrix* createCramerMatrix(matrix*,int);
public:
    matrix();
    matrix(int);
    matrix(int,int);
    ~matrix();
    void readmatrix();
    void displayMatrix();
    matrix* addMatrix(matrix*);
    matrix* multMatrix(matrix*);
    int detMatrix();
    void CramerRule(matrix*);
};
int** matrix::allocatespace(int r,int c)
{
    this->r=0;
    this->c=0;
    maxr=r;
    maxc=c;
    int **t=new int*[r];
    for(int i=0;i<r;i++)
        t[i]=new int[c];
    return t;
}
void matrix::deallocatespace()
{
    for(int i=0;i<r;i++)
        delete []a[i];
    delete []a;
}
matrix::matrix()
{
    maxr=maxc=10;
    a=allocatespace(maxr,maxc);
    cout<<"Matrix of size "<<maxr<<" * "<<maxc<<" allocated\n";
}
matrix::matrix(int n)
{
    maxr=maxc=n;
    a=allocatespace(n,n);
    cout<<"Matrix of size "<<n<<" * "<<n<<" allocated\n";
}
```

```
matrix::matrix(int r,int c)
{
    maxr=r;
    maxc=c;
    a=allocatespace(r,c);
    cout<<"Matrix of size "<<r<<" * "<<c<<" allocated\n";
}
matrix::~matrix()
{
    deallocatespace();
    cout<<"Matrix of size "<<maxr<<" * "<<maxc<<" destroyed\n";
}
void matrix::readmatrix()
{
    do
    {
        cout<<"Enter order of matrix: ";
        cin>>r>>c;
        if(r>maxr||c>maxc||r<0||c<0)
            cout<<"Wrong input. Enter again.\n";
    }while(r>maxr||c>maxc||r<0||c<0);
    int i, j;
    cout<<"Input matrix:\n";
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            cin>>a[i][j];
}
void matrix::displayMatrix()
{
    if(r==0&&c==0)
    {
        cout<<"Matrix is empty\n";
    }
    int i, j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
            cout<<a[i][j]<<' ';
        cout<<endl;
    }
}
matrix* matrix::addMatrix(matrix *m)
{
    matrix *result=new matrix(r,c);
    if(r!=m->r||c!=m->c)
    {
        cout<<"Matrix addition not possible\n";
        return result;
    }
    int i, j;
    result->r=r;
    result->c=c;
    for(i=0;i<r;i++)
        for(int j=0;j<c;j++)
            result->a[i][j]=a[i][j]+m->a[i][j];
    return result;
}
```

```
matrix* matrix::multMatrix(matrix *m)
{
    matrix *result=new matrix(r,m->c);
    if(c!=m->r)
    {
        cout<<"Matrix multiplication not possible\n";
        return result;
    }
    int i, j, k;
    result->r=r;
    result->c=m->c;
    for(i=0;i<r;i++)
    {
        for(j=0;j<m->c;j++)
        {
            result->a[i][j]=0;
            for(k=0;k<c;k++)
                result->a[i][j]+=(a[i][k]*m->a[k][j]);
        }
    }
    return result;
}
int matrix::detMatrix()
{
    if(r!=c)
    {
        cout<<"Determinant not defined\n";
        return -1;
    }
    int i, det=0, f=1;
    matrix *cofmat=NULL;
    if(r==1)
        return a[0][0];
    else if(r==2)
        return (a[0][0]*a[1][1]-a[0][1]*a[1][0]);
    else
    {
        for(i=0;i<r;i++)
        {
            cofmat=cofactor(i);
            det+=(f*a[0][i]*cofmat->detMatrix());
            f=-f;
            delete cofmat;
        }
        return det;
    }
}
matrix* matrix::cofactor(int x)
{
    int i, j, cj=0;
    matrix *m=new matrix(r-1,r-1);
    m->r=r-1;
    m->c=r-1;
    for(i=1;i<r;i++)
    {
        cj=0;
        for(j=0;j<c;j++)
```

```
        {
            if(j!=x)
            {
                m->a[i-1][cj]=a[i][j];
                cj++;
            }
        }
    }
    return m;
}
void matrix::CramerRule(matrix *m)
{
    if(m->c!=1||r!=c||r!=m->r)
    {
        cout<<"Solution not possible by Cramer's rule\n";
        return;
    }
    matrix *cram=NULL;
    double x;
    int d=detMatrix(), d_i, i;
    for(i=0;i<r;i++)
    {
        cram=createCramerMatrix(m,i);
        d_i=cram->detMatrix();
        delete cram;
        x=double(d_i)/d;
        cout<<"x_"<<i+1<<" = "<<x<<endl;
    }
}
matrix* matrix::createCramerMatrix(matrix *C,int index)
{
    matrix *t=new matrix(r,c);
    t->r=r;
    t->c=c;
    int i, j;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        {
            if(j==index)
                t->a[i][j]=C->a[i][0];
            else
                t->a[i][j]=a[i][j];
        }
    return t;
}
int main()
{
    cout<<"Input matrices A and C respectively to solve set of linear
equations AX=C by Cramer's rule\n";
    matrix A, C;
    A.readmatrix();
    C.readmatrix();
    A.CramerRule(&C);
    matrix D1[5]={5,4,6,8,10};
    matrix D2[3]={{2,3},{3,4},{4,5}};
    {
        matrix *E=new matrix(4,5);
```

```
        matrix *F=new matrix[4];
        cout<<"For matrix E:\n";
        E->readmatrix();
        cout<<"Displaying matrix E:\n";
        E->displayMatrix();
        delete []F;
        delete E;
    }
    return 0;
}
```

Output:

Input matrices A and C respectively to solve set of linear equations AX=C by Cramer's rule
Matrix of size 10 * 10 allocated
Matrix of size 10 * 10 allocated
Enter order of matrix: 3 3
Input matrix:
3 2 -1
1 -1 5
2 1 0
Enter order of matrix: 3 1
Input matrix:
1
-2
3
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 3 * 3 allocated
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 3 * 3 destroyed
x_1 = 12
Matrix of size 3 * 3 allocated
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed

```
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 3 * 3 destroyed
x_2 = -21
Matrix of size 3 * 3 allocated
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 2 * 2 allocated
Matrix of size 2 * 2 destroyed
Matrix of size 3 * 3 destroyed
x_3 = -7
Matrix of size 5 * 5 allocated
Matrix of size 4 * 4 allocated
Matrix of size 6 * 6 allocated
Matrix of size 8 * 8 allocated
Matrix of size 10 * 10 allocated
Matrix of size 2 * 3 allocated
Matrix of size 3 * 4 allocated
Matrix of size 4 * 5 allocated
Matrix of size 4 * 5 allocated
Matrix of size 10 * 10 allocated
Matrix of size 10 * 10 allocated
Matrix of size 10 * 10 allocated
Matrix of size 10 * 10 allocated
For matrix E:
Enter order of matrix: 2 3
Input matrix:
1 2 3
4 5 6
Displaying matrix E:
1 2 3
4 5 6
Matrix of size 10 * 10 destroyed
Matrix of size 10 * 10 destroyed
Matrix of size 10 * 10 destroyed
Matrix of size 10 * 10 destroyed
Matrix of size 4 * 5 destroyed
```

Matrix of size 4 * 5 destroyed
Matrix of size 3 * 4 destroyed
Matrix of size 2 * 3 destroyed
Matrix of size 10 * 10 destroyed
Matrix of size 8 * 8 destroyed
Matrix of size 6 * 6 destroyed
Matrix of size 4 * 4 destroyed
Matrix of size 5 * 5 destroyed
Matrix of size 10 * 10 destroyed
Matrix of size 10 * 10 destroyed

Question (d)

Code:

```
#include<iostream>
#include<stdbool.h>
#include<time.h>
using namespace std;
struct node
{
    int data;
    node *next;
};
class list
{
private:
    node *head;
    void initialize();
    void deconstruct();
    bool isempty();
    node* createNewNode(int);
public:
    list();
    list(int);
    ~list();
    void insertBeg(int);
    void display();
    void Delete(int);
    bool search(int);
    void concatenate(list*);
};
void list::initialize()
{
    head=NULL;
}
void list::deconstruct()
{
    node *t=head;
    while(head!=NULL)
    {
        head=head->next;
        delete t;
        t=head;
    }
}
bool list::isempty()
{
    return head==NULL;
}
node* list::createNewNode(int x)
{
    node *t=new node;
    t->data=x;
    t->next=NULL;
    return t;
}
```

```
list::list()
{
    initialize();
    cout<<"List created\n";
}
list::list(int x)
{
    initialize();
    cout<<"List created\n";
    head=createNewNode(x);
}
void list::insertBeg(int x)
{
    node *t=createNewNode(x);
    if(t==NULL)
    {
        cout<<"Out of memory\n";
        return;
    }
    if(head==NULL)
        head=t;
    else
    {
        t->next=head;
        head=t;
    }
}
void list::display()
{
    if(isempty())
    {
        cout<<"List is empty\n";
        return;
    }
    cout<<"Displaying list from beginning:\n";
    for(node *t=head;t!=NULL;t=t->next)
        cout<<t->data<<' ';
    cout<<endl;
}
void list::Delete(int x)
{
    if(isempty())
    {
        cout<<"List is empty\n";
        return;
    }
    node *p=head;
    node *q;
    int val;
    if(head->data==x)
    {
        val=head->data;
        head=head->next;
        delete p;
        cout<<"Successfully deleted "<<val<<endl;
    }
    else
```

```
    {
        while(p!=NULL&&p->data!=x)
        {
            q=p;
            p=p->next;
        }
        if(p==NULL)
            cout<<"No match :: deletion failed\n";
        else
        {
            val=p->data;
            q->next=p->next;
            delete p;
            cout<<"Successfully deleted "<<val<<endl;
        }
    }
}
bool list::search(int x)
{
    for(node *t=head;t!=NULL;t=t->next)
        if(t->data==x)
            return true;
    return false;
}
void list::concatenate(list *l)
{

    if(head==NULL)
    {
        head=l->head;
        return;
    }
    else if(l->head==NULL)
    {
        l->head=head;
        return;
    }
    else
    {
        node *t;
        for(t=head;t->next!=NULL;t=t->next);
        t->next=l->head;
    }
}
list::~list()
{
    deconstruct();
    cout<<"List destroyed\n";
}
unsigned long int myrand(unsigned long int x)
{
    unsigned long long int m=2147483647, a=65539;
    unsigned long int r=(x*a)%m;
    return r;
}
int main()
{
```

```
    list l1(50), l2, l3[4]={1,2,3,4};
    time_t t=time(NULL);
    unsigned long int seed=t;
    int x;
    for(int i=0;i<10;i++)
    {
        seed=myrand(seed);
        l2.insertBeg(seed%100);
    }
    l2.display();
    cout<<"Enter element to be deleted: ";
    cin>>x;
    l2.Delete(x);
    l2.display();
    l1.display();
    {
        list *l4=new list(10);
        list *l5=new list[5];
        l4->insertBeg(5);
        l4->display();
        delete []l5;
        delete l4;
    }
    l3[2].display();
    return 0;
}
```

Output:

List created
List created
List created
List created
List created
List created
Displaying list from beginning:
66 14 32 54 77 49 68 42 33 24
Enter element to be deleted: 66
Successfully deleted 66
Displaying list from beginning:
14 32 54 77 49 68 42 33 24
Displaying list from beginning:
50
List created
List created
List created
List created
List created
List created

Displaying list from beginning:
5 10
List destroyed
List destroyed
List destroyed
List destroyed
List destroyed
List destroyed
Displaying list from beginning:
3
List destroyed
List destroyed
List destroyed
List destroyed
List destroyed
List destroyed