WRITE AN ALGORITHM AND PROGRAM FOR THE FOLLOWING:

1. IMPLEMENT STACK AS LINKED LIST

ALGORITHM:

1. Create a linked list with first node as null pointer.
2. Push Operation:
3. Get data to be entered from user.
4. Check if the stack is full or not. If full, print OVERFLOW and terminate.
5. Else create a newNode and set its item to given data.
6. Set newNode->next to start and start=newNode
7. Pop Operation:
8. Check if the list is empty, if so, throw UNDERFLOW and terminate process
9. If list is not empty then return start->item as dequeued data
10. Then, remove the item at the beginning of the linked list (i.e. the top of the stack) by setting start=start->next.
11. Display the item stored in the first node of the list when peek ( ) is called.

PROGRAM CODE:

#include <iostream>

using namespace std;

template <class t>

struct NODE

{

    t item;

    NODE<t>\* next;

};

template <class t>

class StackL

{

    typedef NODE<t> Node;

    int sizeL;

    const int maxSize;

    Node\* start;

public:

    class UNDERFLOW{};

    class OVERFLOW{};

    StackL(int s=5):maxSize(s)

    {

        sizeL=0;

        start=nullptr;

    }

    ~StackL()

    {

        if(start!=nullptr)

        {

            Node\* p =start;

            Node\* q = p->next;

            if(q==nullptr)

                delete p;

            else

            for(;q!=nullptr;p=q,q=q->next)

                delete p;

        }

    }

    int getSize()

    {

        return sizeL;

    }

    void peek()

    {

        if(start==nullptr)

        {

            throw UNDERFLOW();

        }

        cout<<start->item;

    }

    void push(t);

    t pop();

    //void display();

};

template <class t>

void StackL<t>::push(t data)

{

    if(sizeL<maxSize)

    {

        sizeL++;

        Node\* newNode = new Node();

        newNode->item=data;

        if(start==nullptr)

        {

            newNode->next=nullptr;

            start=newNode;

        }

        else

        {

            newNode->next=start;

            start=newNode;

        }

    }

    else

    {

        throw OVERFLOW{};

    }

}

template <class t>

t StackL<t>::pop()

{

    if(start==nullptr)

        throw UNDERFLOW();

    sizeL--;

    t temp=start->item;

    start=start->next;

    return temp;

}

/\*template <class t>

void StackL<t>::display()

{

    if(start == nullptr)

        throw UNDERFLOW();

    Node<t>\* p = start;

    for(; p != nullptr ; p = p->next)

        std::cout<<" "<<p->item;

}\*/

int main()

{

    int op,sizeSt,num;

    cout<<"Size of Stack:";

    cin>>num;

    sizeSt=num;

    StackL<int> s(num);

    while(1)

    {

    try

    {

        cout<<"\nEnter Operation:\t1)Push\t2)Pop\t3)Exit:";

        cin>>op;

        if(op==1)

        {

            if(s.getSize()<sizeSt)

            {

                cout<<"\nEnter number to be pushed:";

                cin>>num;

                s.push(num);

                cout<<"\nTop of Stack:\t";

                s.peek();

            }

            else

            {

                throw(StackL<int>::OVERFLOW());

            }

        }

        else if(op==2)

        {

            s.pop();

            cout<<"\nTop of Stack:\t";

            s.peek();

        }

        else

        {

           exit(0);

        }

    }

    catch(StackL<int>::UNDERFLOW)

    {

        cout<<"\nStack is Empty!";

    }

    catch(StackL<int>::OVERFLOW)

    {

        cout<<"\nStack is Full!";

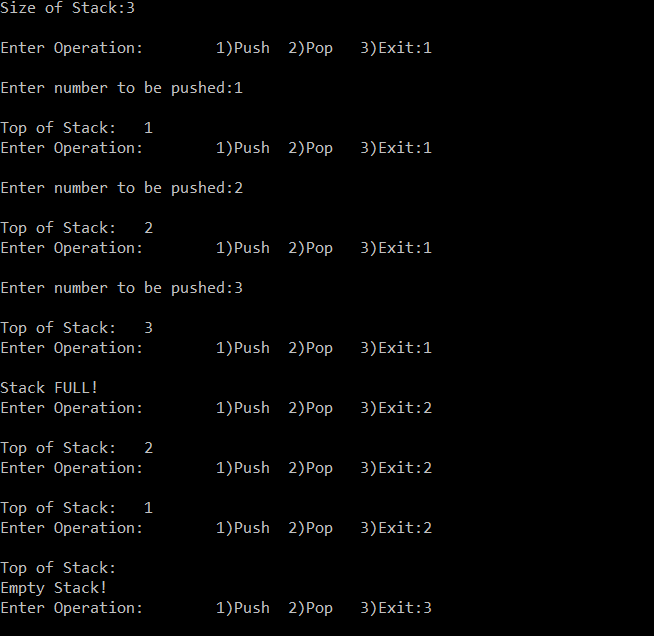
    }

    }

    return 0;

}

Output



2. IMPLEMENT QUEUE AS LINKED LIST

ALGORITHM:

1. Enqueue:
2. Get data to be enqueued from user.
3. Check if the queue is full or not. If full, print OVERFLOW and terminate.
4. Else create a newNode and set its item to given data.
5. Set newNode->next to start and start=newNode
6. Dequeue:
7. Check if the queue is empty or not. If empty, print UNDERFLOW and terminate
8. Else create a newNode and set its item to given data
9. Traverse list until presnetNode->next!=null pointer and then store item of last node as dequeued item
10. Set next of node before presentNode to null pointer (i.e. delete last node from list.)
11. Free the space of removed node.

Program Code

#include <iostream>

using namespace std;

template <class t>

struct NODE

{

    t item;

    NODE<t>\* next;

};

template <class t>

class QueueL

{

    typedef NODE<t> Node;

    int sizeL;

    const int maxSize;

    Node\* start;

public:

    class UNDERFLOW{};

    class OVERFLOW{};

    QueueL(int s=5):maxSize(s)

    {

        sizeL=0;

        start=nullptr;

    }

    ~QueueL()

    {

        if(start!=nullptr)

        {

            Node\* p =start;

            Node\* q = p->next;

            if(q==nullptr)

                delete p;

            else

            for(;q!=nullptr;p=q,q=q->next)

                delete p;

        }

    }

    int getSize()

    {

        return sizeL;

    }

    void enqueue(t);

    t dequeue();

    void display();

};

template <class t>

void QueueL<t>::enqueue(t data)

{

    if(sizeL<maxSize)

    {

        sizeL++;

        Node\* newNode = new Node();

        newNode->item=data;

        if(start==nullptr)

        {

            newNode->next=nullptr;

            start=newNode;

        }

        else

        {

            newNode->next=start;

            start=newNode;

        }

    }

    else

    {

        throw OVERFLOW{};

    }

}

template <class t>

t QueueL<t>::dequeue()

{

    if(start == nullptr)

            throw UNDERFLOW();

    sizeL--;

        t temp;

        Node\* p = start;

        Node\* q = nullptr;

            for(;p->next != nullptr; q = p, p = p->next) ;

        if(q == nullptr)

        {

            start = nullptr;

        }

        else

        {

            q->next = nullptr;

        }

        temp = p->item;

        delete p;

        return  temp;

}

template <class t>

void QueueL<t>::display()

{

    if(start == nullptr)

        throw UNDERFLOW();

    Node\* p = start;

    for(; p != nullptr ; p = p->next)

        std::cout<<" "<<p->item;

}

int main()

{

    int op,sizeSt,num;

    cout<<"Size of Queue:";

    cin>>num;

    sizeSt=num;

    QueueL<int> s(num);

    while(1)

    {

    try

    {

        cout<<"\nEnter Operation:\t1)Enqueue\t2)Dequeue\t3)Exit:";

        cin>>op;

        if(op==1)

        {

            if(s.getSize()<sizeSt)

            {

                cout<<"\nEnter number to be enqueued:";

                cin>>num;

                s.enqueue(num);

                cout<<"\nData in Queue:\t";

                s.display();

            }

            else

            {

                throw(QueueL<int>::OVERFLOW());

            }

        }

        else if(op==2)

        {

            s.dequeue();

            cout<<"\nData in Queue:\t";

            s.display();

        }

        else

        {

           exit(0);

        }

    }

    catch(QueueL<int>::UNDERFLOW)

    {

        cout<<"\nQueue is Empty!";

    }

    catch(QueueL<int>::OVERFLOW)

    {

        cout<<"\nQueue is Full!";

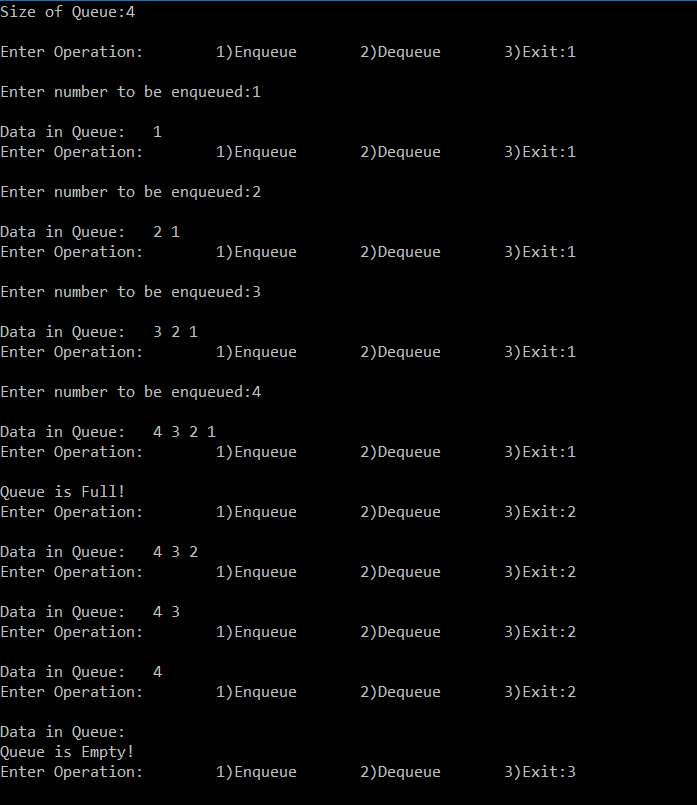
    }

    }

    return 0;

}

Output



3.REPRESENT THE POLYNOMIAL EQUATION USING LINKED LIST TO PERFORM ADDITION OF TWO POLYNOMIALS.

ALGORITHM

1. **Insert a term to the polynomial list:**
2. Get degree and coefficient of term to be added from the user
3. Create new node representing the term to be added
4. If the list is empty set start = new node
5. Else traverse list to find first node with degree less than given degree
6. If no such node exist i.e. all nodes have higher degree than given node then add the new node to end
7. Else if the node just before the traversed node has the same degree as given then replace it
8. Else if the node before is greater in degree then add the new node before the traversed node
9. **Addition of two polynomials:**
10. Set p and q as start of first and second polynomial respectively.
11. If (p.degree == q.degree), then, add the coefficients and store in result polynomial list
12. Else if p.degree > q.degree then add that node of p in result polynomial and set p = p.next
13. Else add that node of q in result polynomial and set q = q.next
14. Repeat the steps 2,3 and 4 until p != null or q!= null
15. Add the remaining nodes if any of list1 or list2 to result

Program Code

#include <iostream>

using namespace std;

template <class t>

struct NODE

{

    t coef;

    int deg;

    NODE<t>\* next;

};

template <class t>

class Poly

{

    typedef NODE<t> Node;

    Node\* start;

    public:

    Poly()

    {

        start=nullptr;

    }

    ~Poly()

    {

        if(start!=nullptr)

        {

            Node\* p=start;

            for(;p!=nullptr;p=p->next)

            delete p;

        }

    }

    void getInput();

    void insertInPoly(t,int);

    void display();

    Poly operator+(Poly);

};

template <class t>

void Poly<t>::getInput()

{

    t coef;

    int deg;

    char op;

    do

    {

        cout<<"\nEnter coefficient: ";

        cin>>coef;

        cout<<"Enter degree: ";

        cin>>deg;

        insertInPoly(coef,deg);

        cout<<"Continue?(Y/N):";

        cin>>op;

    }while (toupper(op)=='Y');

}

template <class t>

Poly<t> Poly<t>::operator+(Poly<t> addend)

{

    Poly<t> res;

    Node\* p= start;

    Node\* q= addend.start;

    while(p!=nullptr && q!=nullptr)

    {

        if(p->deg==q->deg)

        {

            res.insertInPoly(p->coef+q->coef,p->deg);

            p=p->next;

            q=q->next;

        }

        else if(p->deg > q->deg)

        {

            res.insertInPoly(p->coef,p->deg);

            p=p->next;

        }

        else

        {

            res.insertInPoly(q->coef,q->deg);

            q=q->next;

        }

    }

    if(p!=nullptr)

    {

        for(;p!=nullptr;p=p->next)

        res.insertInPoly(p->coef,p->deg);

    }

    else if(q!=nullptr)

    {

        for(;q!=nullptr;q=p->next)

        res.insertInPoly(q->coef,q->deg);

    }

    return res;

}

template <class t>

void Poly<t>::insertInPoly(t coef,int deg)

{

    Node\* newNode = new Node();

    newNode->coef=coef;

    newNode->deg=deg;

    newNode->next=nullptr;

    if(start==nullptr)

    {

        start=newNode;

        return;

    }

    Node\* p;

    Node \*q;

    for(p=start,q=nullptr;p!=nullptr && deg<=p->deg;q=p,p=p->next);

//traverses list to node just before node having degree less than given degree

    if(p!=nullptr && q==nullptr)

    {

        newNode->next=p;

        start=newNode;

    }

    else if(p!=nullptr && q!=nullptr)

    {

        if(q->deg==deg)

        q->coef=coef;

        else

        {

            newNode->next=q->next;

            q->next=newNode;

        }

    }

    else if(p==nullptr)

    {

        q->next=newNode;

        return;

    }

}

template <class t>

void Poly<t>::display()

{

    if(start==nullptr)

    {

        cout<<"";

        return;

    }

    Node\* p=start;

    for(;p!=nullptr;p=p->next)

    cout<<p->coef<<"x^"<<p->deg<<" + ";

    cout<<"\b\b\b   \b\b\b";

}

int main()

{

    Poly<int> P1,P2,res;

    cout<<"\nEnter Polynomial 1:";

    P1.getInput();

    cout<<"\nEnter Polynomial 2:";

    P2.getInput();

    cout<<"\n-------------------------Adding-------------------------\n\t";

    P1.display();

    cout<<"\n\n+\t";

    P2.display();

    cout<<"\n---------------------------------------------------------\n\t";

    res=P1+P2;

    res.display();

    return 0;

}

Output

