Lecture 4

Abstract Data Types (ADT): Stacks, and Queue

(Chapter 3 of textbook 1)

Single Linked List vs Array

Array	Linked List
Pros Directly supported by C Provides random access (index)	Pros No need to determine size Inserting and deleting elements is quick
Cons Size determined at compile time Inserting and deleting elements is time consuming	Cons No random access 1-way walking Wasted space for pointer

Depending on the applications, we will use Linked List or Array

Linked List application: Polynomial ADT

- $F(x) = \sum_i a_i x^i$.
- Example: $F(x) = 10x^{1000} + 5x^{14}+1$
  (a_{1000}=10, a_{14} = 5, a_0 =1, and other a_i = 0)
  placeholder of the power
- Operations:
  - Print_Poly
  - Addition
  - Multiplication

Polynomial (2)

```
struct Node {
 int coef;
int exp;
struct Node* next;
};

typedef struct Node* PtrToNode;
```

$4x^3$ → exponent
↓
coefficient

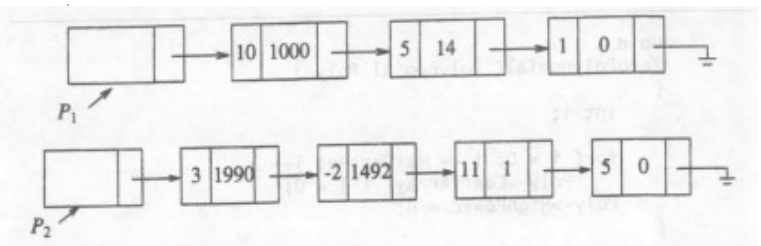


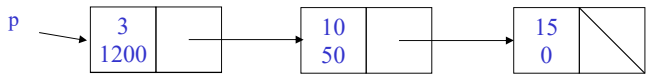
Figure 3.22 Linked list representations of two polynomials

Two linked List are already sorted by exponent

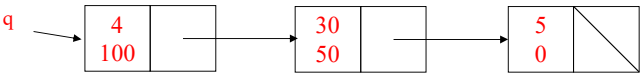
Addition of Two Polynomials

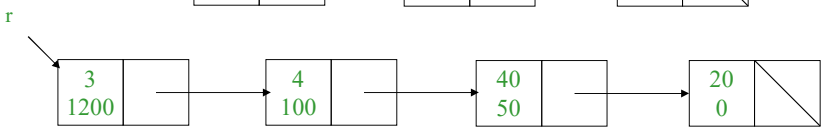
- Similar to merging two sorted lists

$3x^{1200}+10x^{50}+15$



$4x^{100}+30x^{50}+5$





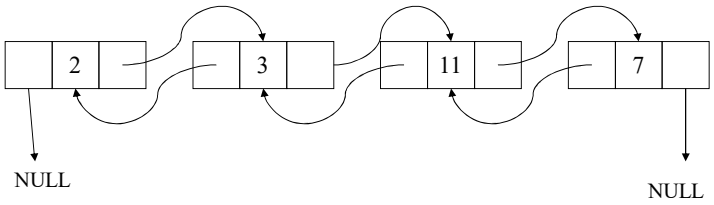
Quiz

Write C program for addition of two polynomials

Doubly Linked List

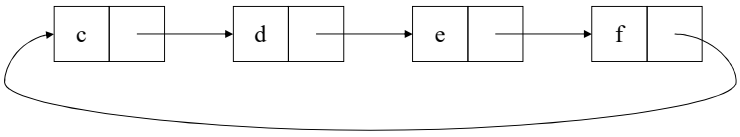
A node contains 2 pointers: to previous node and next node;

One can move in both directions;



Circular Linked List

The last node points to the first one. It can be done with doubly lined list.



Examples of circular linked list

Registration problem:

An university with 2500 courses and 40000 students to generate two types of reports: the list of students of each course and list of courses of each student.

Solution 1: 2-D array=> 100 million entries=> bad idea

Solution 2: Multi-Lists

Multi Lists

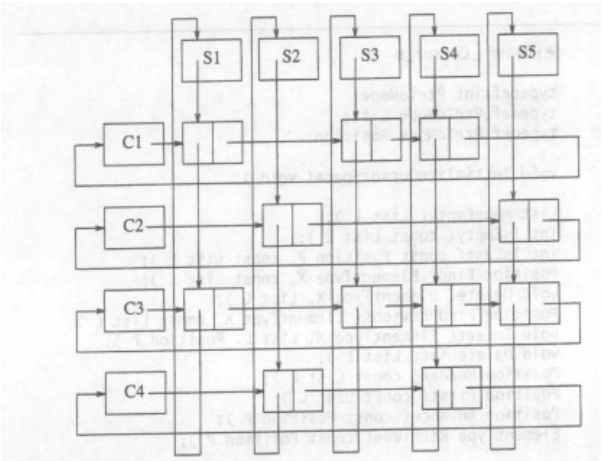


Figure 3.27 Multilist implementation for registration problem

Advantage:
Saving space
Disadvantage:
Expense of time

Stack ADT

Stack is a list where you can access add or delete elements at one end.

Stacks are called ``last in first out'' (LIFO), the last added element among the current ones can be accessed or deleted.

Operations of Stack:

Stack CreateStack ()

int isEmpty (Stack S)

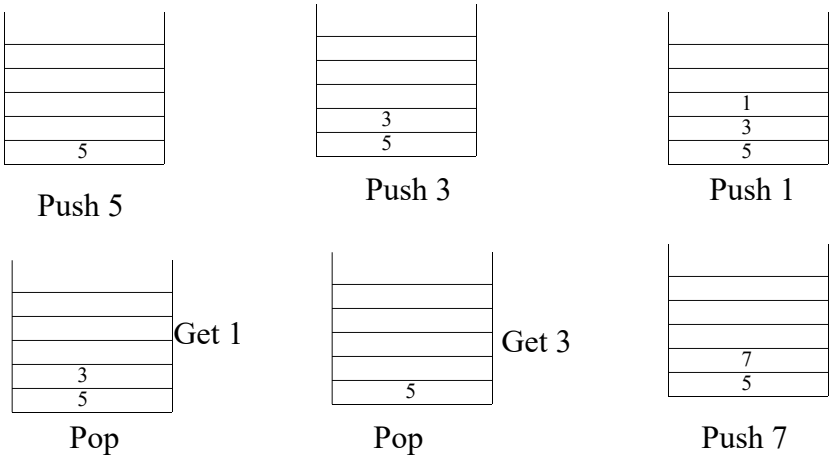
ElementType Pop (Stack S) -> Delete the element at top of a stack

void Push(Stack S, ElementType X) -> Add an element X at top of a stack

ElementType Top(Stack S) -> Access the element at top of a stack

EXAMPLE

Push 5, 3, 1, Pop, Pop, Push 7



Stack Implementation

Array

Need to know the maximum number of elements

Delete/Access/Add at the end

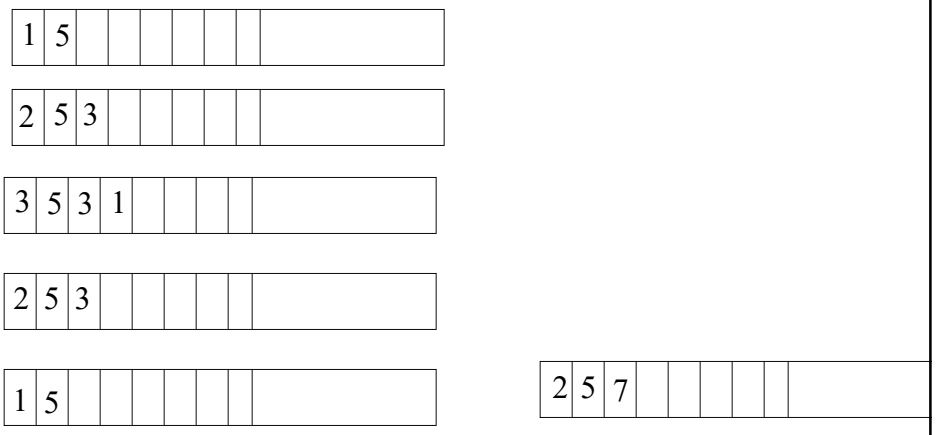
O(1) operation.

Store array length in the 0^{th} position.

Delete/Access/Add at A[array length+1]

Example

Push 5, 3, 1, Pop, Pop, Push 7

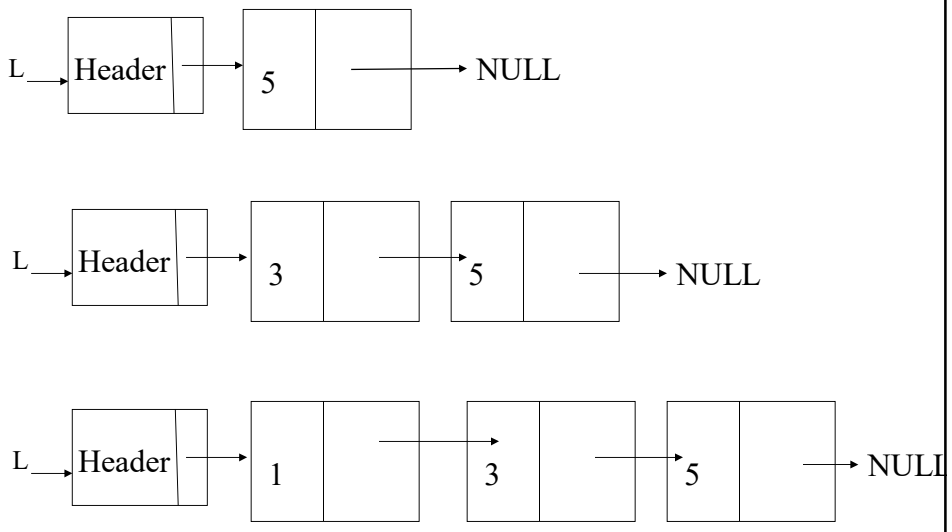


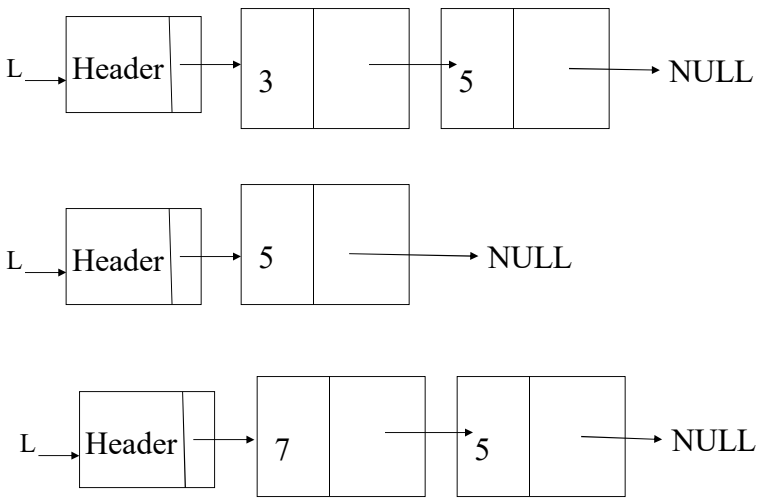
Linked List Implementation.

Need not know the maximum size

Add/Access/Delete in the beginning, O(1)

Push 5, 3, 1, Pop, Pop, Push 7

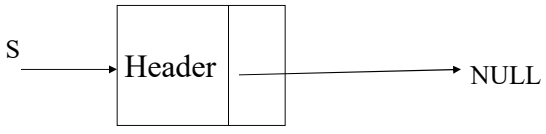




Create a empty Stack with a dummy header

```
void main()
{
    Stack S = NULL;
    S = malloc (size of (struct Node));
    S->Next = NULL;
}
```

```
typedef struct Node {
  ElementType element;
  struct Node *Next;
};
typedef struct Node* Stack
```



Basic operations of Stack

```
void Push(Stack S, ElementType X ) {
        PtrToNode TmpCell;
        TmpCell = malloc( sizeof( struct Node ) );
        if( TmpCell == NULL ) FatalError( "Out of space!!!" );
        else {
                TmpCell->Element = X;
                TmpCell->Next = S->Next;
                S->Next = TmpCell;
        }
}
```

```
int IsEmpty( Stack S ) {
        return S->Next == NULL;
}
```

```
ElementType Pop( Stack S ) {
        PtrToNode FirstCell;
        ElementType X;
        if( IsEmpty( S ) ) Error( "Empty stack" );
        else {
                FirstCell = S->Next;
                X = FirstCell->Element;
                S->Next = S->Next->Next;
                free( FirstCell );
        }
        return X;
}
```

```
ElementType Top( Stack S ) {
        if( !IsEmpty( S ) )
                return S->Next->Element
        Error( "Empty stack" );
        return 0;
}
```

Stack Application

Function Calls

Decimal to Binary Conversion

Mathematical Expression
- Symbol Matching
- Evaluation of Postfix Expressions
- Infix to Postfix Conversion

Decimal to Binary Conversion

Quiz: Please write a C program for Decimal to Binary Conversion using Stack

Example 2: Symbol Balancing

Braces, paranthenses, brackets, begin, ends must match each other

[{ [( )] }] => correct expression

[{]}() => wrong expression

Easy check using stacks

Start from beginning of the file.

Push the beginning symbols you wish to match, ignore the rest.

Push brace, parantheses, brackets, ignore the alphabets

Whenever you encounter a right symbol, pop an element from the stack. If stack is empty, then error.

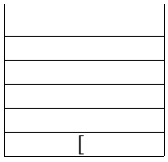
Otherwise, if the popped element is the corresponding left symbol, then fine, else there is an error.

What is the complexity of the operation? O(n)

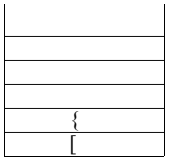
EXAMPLE

Check brace, bracket parentheses matching [a+b{1*2]9*1}+(2-1)

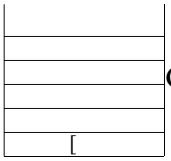
Push [, Push {, Pop, Pop, Push (, Pop



Push [



Push {



Get {

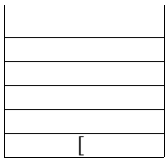
Pop Expect?

Oops! Something wrong, was expecting [

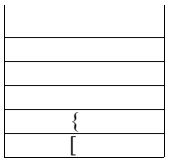
EXAMPLE

Check brace, bracket parentheses matching [a+b{1*2}9*1]+(2-1)

Push [, Push {, Pop , Pop, Push (, Pop



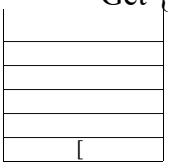
Push [



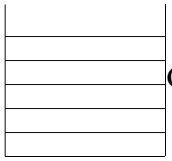
Push {

Perfect!

Get {

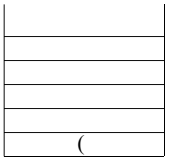


Pop
Expect {



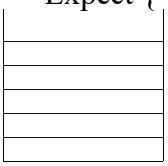
Get [

Pop Expect [



Push (

Get (



Pop Expect (

Symbol Priority

A*B + C

A*(B + C)

A+B+C*D

A*D+B+C*E

To do the correct operation, calculator needs to know the priority of operands

We don't need any priority nor parantheses if the operation is expressed in **postfix** or reverse polish notion.

Postfix

A*B + C = AB*C+

A*(B + C) = (B + C)*A = BC+A*

A+B+C*D = AB+CD*+

A*D+B+C*E = AD*B+CE*+

Suppose the expression is in **postfix**, how do we compute the value?

Computation of a Postfix Expression

Whenever you see a number push it in the stack.

Whenever you see an operator,

pop 2 numbers,

apply the operator,

Push the result

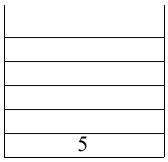
At end Pop to get answer

Complexity? O(n)

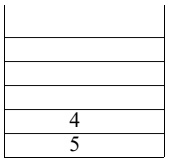
EXAMPLE

Compute 5*4+6+7*2 Result: 40

Postfix: 5 4*6+7 2*+

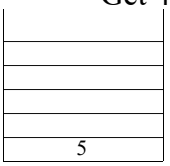


Push 5

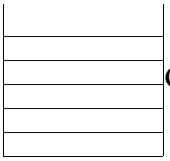


Push 4

Get 4

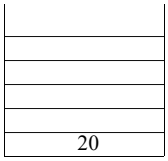


Pop

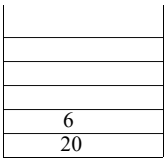


Get 5

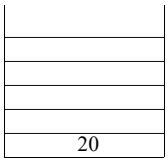
Pop



Push 20

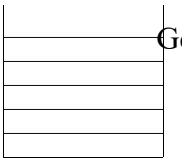


Push 6



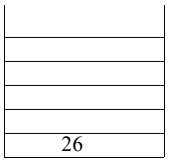
Get 6

Pop

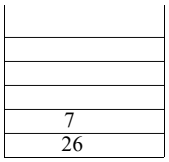


Get 20

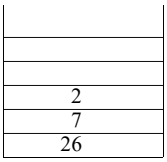
Pop



Push 26

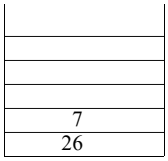


Push 7



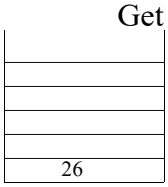
Push 2

Get 2

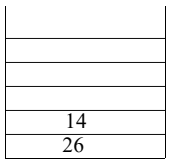


Pop

Get 7

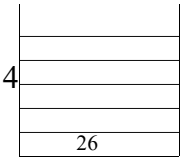


Pop

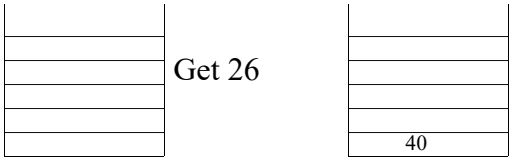


Push 14

Get 14



Pop



Pop Push 40

Pop to get answer

Infix to Postfix Conversion

We can use a stack to convert an expression in standard form (infix) into postfix.

We will concentrate on a small version of general problem by allowing only the operators +, *, (,).

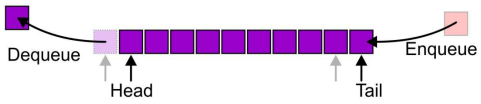
Example: a + b*c + (d *e + f) *g
⇒ a b c * + d e * f +g * +

Implement Infix to Postfix conversion is your homework

Queues ADT

List where an element is inserted at the end, and deleted from the beginning. First in First out

Insertion and deletion at different ends. Deletion and Insertion should be O(1)



Some terminologies:

Dequeue = Pop = Delete
Enqueue = Push = Insert
Head = Front
Tail = Rear

Basic operations of Queue

Queue CreateQueue ()

int isEmpty (Queue Q)

ElementType Dequeue (Queue Head) -> Delete the element **at head of a Queue (Deleting)**

void Enqueue(Queue Tail, ElementType X) -> Add an element X **at tail of a Queue (Inserting)**

Linked List Implementation

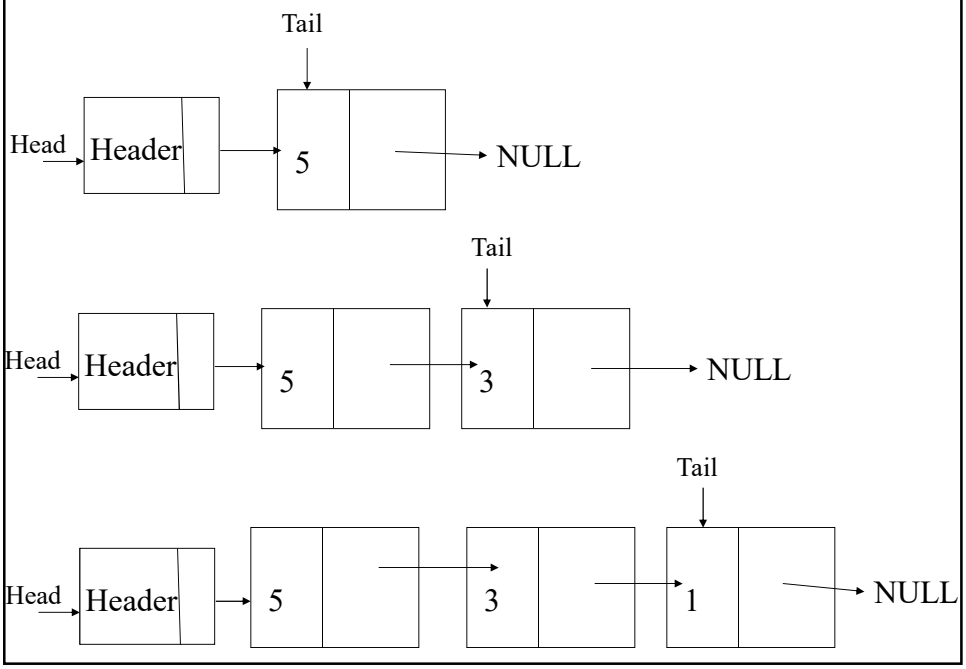
Insert at the end of the linked list

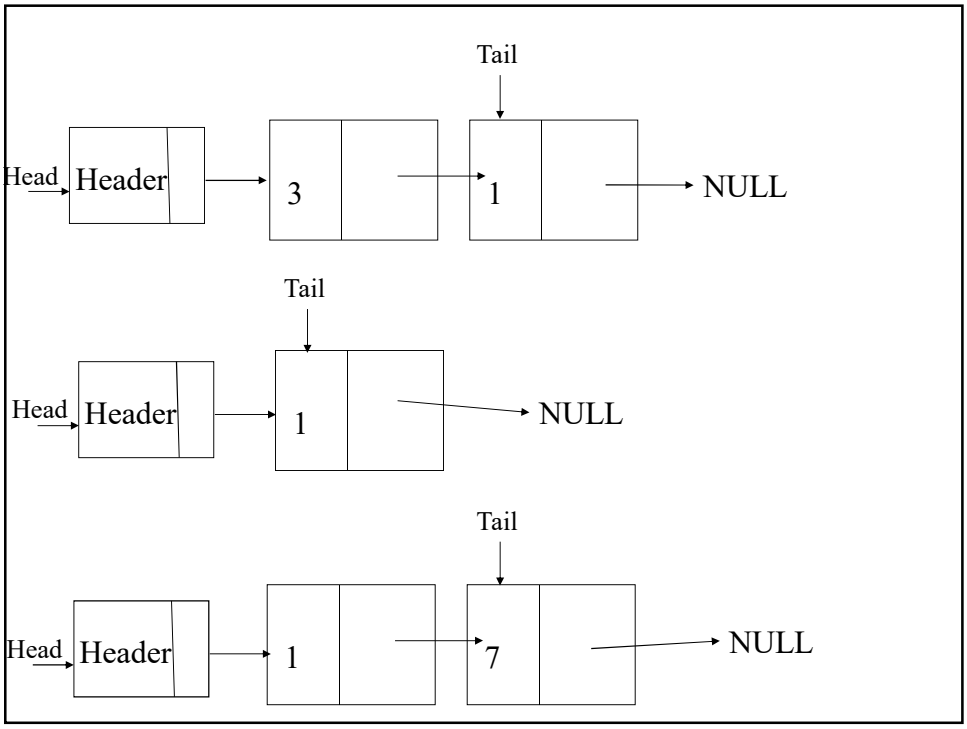
Maintain a pointer to the last element. Whenever there is insertion, update this ``last'' pointer to point to the newly inserted element.

Delete from the beginning of the list

Both insertion and deletion are O(1)

Insert 5, 3, 1, Delete, Delete, Insert 7

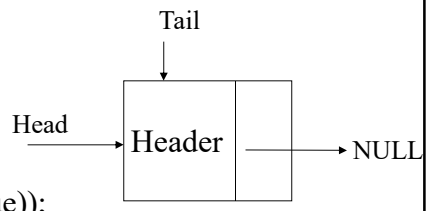




Create a empty Queue with a dummy header

```
struct queue {
  PtrToNode Head;
  PtrToNode Tail;
};
typedef struct queue* Queue

Queue CreateQueue(){
    Queue Q = malloc(sizeof (struct queue));
    Q->Head = malloc (sizeof (struct Node));
    Q->Head->Next = NULL;
    Q->Tail = Q->Head;
    return Q;
}
```

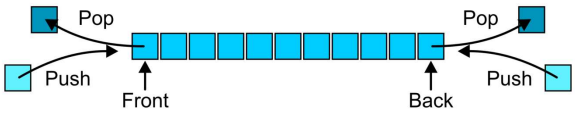


Basic operations of Queue

```
void Enqueue(Queue Q, ElementType X ) {
        PtrToNode TmpCell;
        TmpCell = malloc( sizeof( struct Node ) );
        if( TmpCell == NULL ) printf( "Out of space!!!" );
        else {
                TmpCell->Element = X;
                TmpCell->Next = Q->Tail->Next;
                Q->Tail->Next = TmpCell;
                Q->Tail = TmpCell;
        }
}
ElementType Dequeue( Queue Q ) {
        PtrToNode FirstCell;
        ElementType X;
        if( isEmpty( Q ) ) printf( "Empty stack" );
        else {
                FirstCell = Q->Head->Next;
                X = FirstCell->Element;
                Q->Head->Next = Q->Head->Next->Next;
                free( FirstCell );
        }
        return X;
}
```

Deque ADT

A Deque is an abstract data structure which emphasizes specific operations: Allows insertions and deletion at both the front and back of the deque



Useful as a general-purpose tool: Can be used as either a queue or a stack

Implement Deque as your homework

Summary

- Doubly Linked List
- Circular Linked List
- Stack
- Queue, Deque
- Next week: Tree (chapter 4 of textbook 1)
- Please finish your homework