

What is the types of Data Structure for implementing Stack?

Data Structure:
Stack

Convert infix-expr to postfix-expr.

- 1. Understanding Data-Structure ADT and Basic concepts
- 2. Understand Operations of Data-Structure(or Algorithms)
- 3. Application of Data-Structure(or Algorithms)
- 4. Performance of Stack Operation
- 5. Solve Example problems

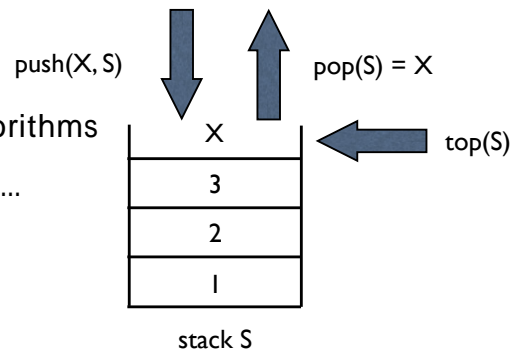
Analysis using Asymptotic Notation

Stack ADT

- 1. Last in First out
- 2. Stack

Stack (Top_of_Stack)
a list that insertions and deletions can be performed at the end of the list
operations
push(X, S): insert X in the list S
pop(S): deletes the most recently inserted element from S

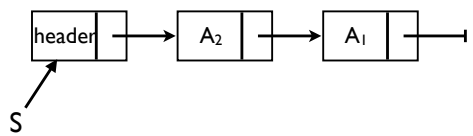
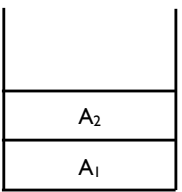
Data Structure & Algorithms
Class



push(1, S) → push(2, S) → push(3, S) → push(X, S)

Stack ADT: linked list implementation

Stack ADT: linked list implementation



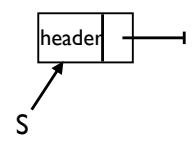
```
struct Node;
typedef struct Node *PtrToNode;
typedef PtrToNode Stack;

struct Node{
    ElementType Element;
    PtrToNode Next;
};
```

```
Stack CreateStack (){
    Stack S;
    S = malloc(sizeof (struct Node));

    if (S==NULL)
        FatalError("Out of space !!!");

    S -> Next = NULL;
    return S;
}
```



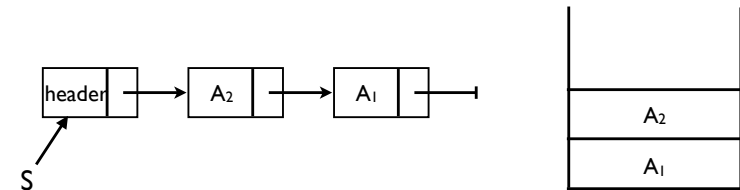
Stack ADT: linked list implementation

```
void MakeEmpty(Stack S) {
    if (S == NULL)
        Error ("No stack exists");
    else
        while( !IsEmpty(S))
            Pop(S);
}
```

Stack ADT: linked list implementation

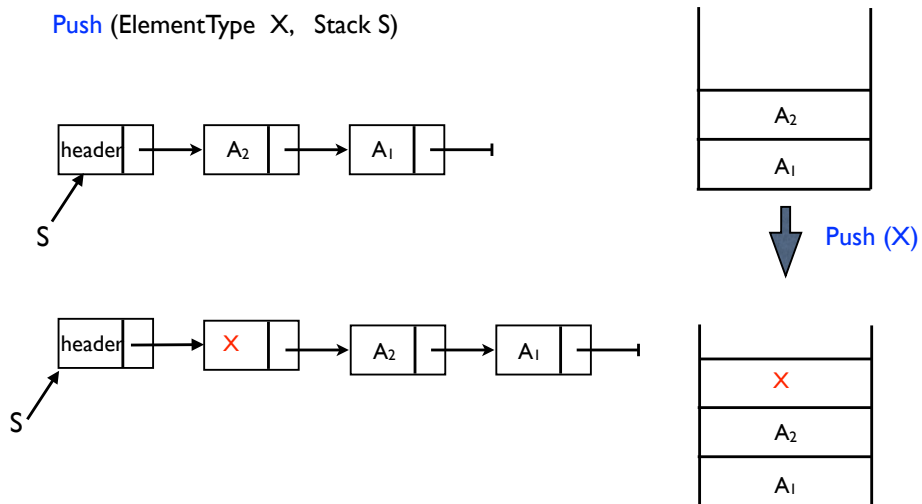
```
ElementType Top (Stack S) {
    if (!IsEmpty(S))
        return S->Next->Element;

    Error ("Empty stack");
    return 0;
}
```

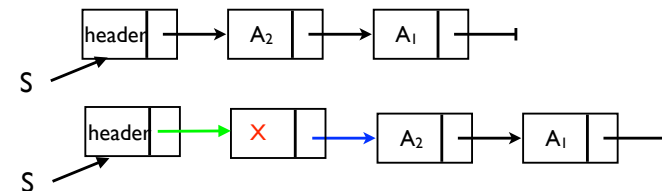


Stack ADT: linked list implementation

Push (ElementType X, Stack S)



Stack ADT: linked list implementation

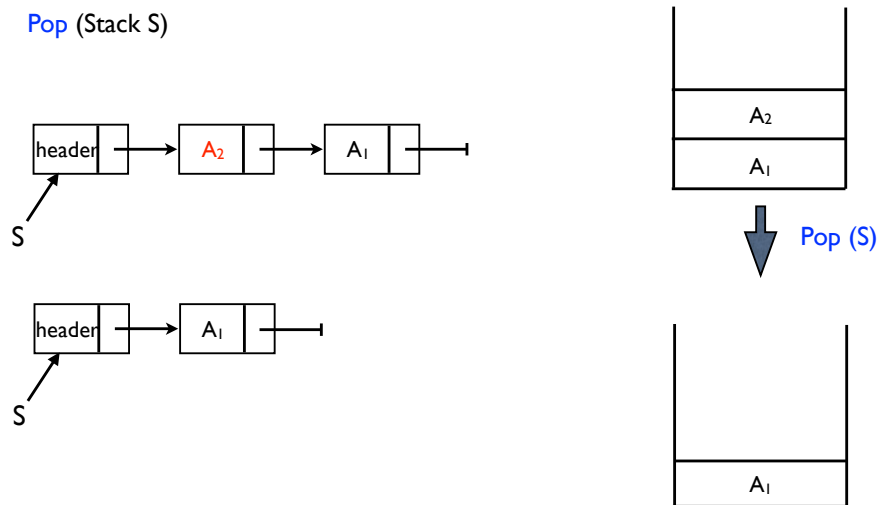


```
void Push (ElementType X, Stack S) {
    PtrToNode TmpCell;
    TmpCell = malloc (sizeof (struct Node));

    if (TmpCell == NULL) {
        FatalError ("Out of space !!!");
    } else {
        TmpCell -> Element = X;
        TmpCell -> Next = S -> Next;
        S -> Next = TmpCell;
    }
}
```

Stack ADT: linked list implementation

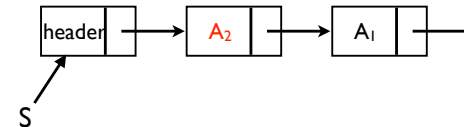
Pop (Stack S)



Stack ADT: linked list implementation

```
void Pop (Stack S) {
    PtrToNode FirstCell;

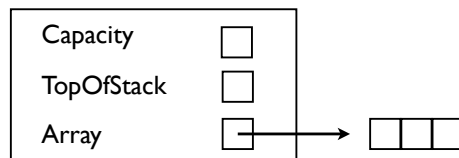
    if (IsEmpty(S))
        Error("Empty stack");
    else{
        FirstCell = S->Next;
        S->Next = S->Next->Next;
        free(FirstCell);
    }
}
```



Stack ADT: array implementation

```
typedef struct StackRecord *Stack;
```

```
struct StackRecord
{
    int Capacity;
    int TopOfStack;
    ElementType *Array;
};
```



Stack ADT: array implementation

```
#define EmptyTOS ( -1 )
```

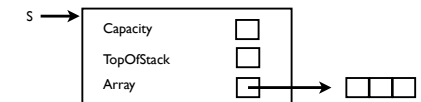
```
Stack CreateStack( int MaxElements )
{
    Stack S;
```

```
    S = malloc( sizeof( struct StackRecord ) );
    if( S == NULL )
        FatalError( "Out of space!!!" );
```

```
    S->Array = malloc( sizeof( ElementType ) * MaxElements );
    if( S->Array == NULL )
        FatalError( "Out of space!!!" );
```

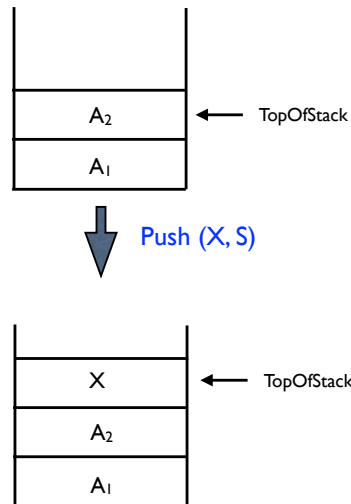
```
    S->Capacity = MaxElements;
    S->TopOfStack = EmptyTOS;
```

```
    return S;
}
```



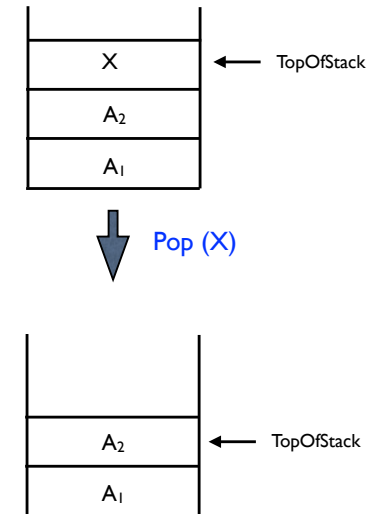
Stack ADT: array implementation

```
void Push( ElementType X, Stack S )
{
    if( IsFull( S ) )
        Error( "Full stack" );
    else
        S->Array[ ++S->TopOfStack ] = X;
}
```



Stack ADT: array implementation

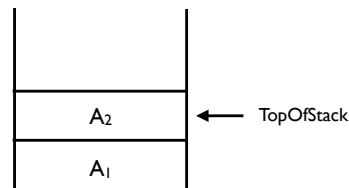
```
void Pop( Stack S )
{
    if( IsEmpty( S ) )
        Error( "Empty stack" );
    else
        S->TopOfStack--;
}
```



Stack ADT: array implementation

```
ElementType Top( Stack S )
{
    if( !IsEmpty( S ) )
        return S->Array[ S->TopOfStack ];

    Error( "Empty stack" );
    return 0;
}
```



infix, prefix, and postfix notation

infix

$$3 + 4 * 6 \begin{cases} \xrightarrow{?} (3 + 4) * 6 \\ \xrightarrow{?} 3 + (4 * 6) \end{cases}$$

prefix

$$(3 + 4) * 6 \longrightarrow * + 3 4 6$$

$$3 + (4 * 6) \longrightarrow + 3 * 4 6$$

postfix

$$(3 + 4) * 6 \longrightarrow 3 4 + 6 *$$

$$3 + (4 * 6) \longrightarrow 3 4 6 * +$$

postfix evaluation

$7 \ 2 \ 3 \ * \ - \ 4 \ \uparrow \ 9 \ 3 \ / \ +$
 $\underline{2 \ * \ 3 = 6}$
 $7 \ 6 \ - \ 4 \ \uparrow \ 9 \ 3 \ / \ +$
 $\underline{7 - 6 = 1}$
 $1 \ 4 \ \uparrow \ 9 \ 3 \ / \ +$
 $\underline{1^4 = 1}$
 $1 \ 9 \ 3 \ / \ +$
 $\underline{9 / 3 = 3}$
 $1 \ 3 \ +$
 $\underline{1 + 3 = 4}$

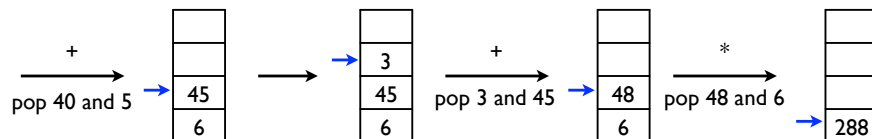
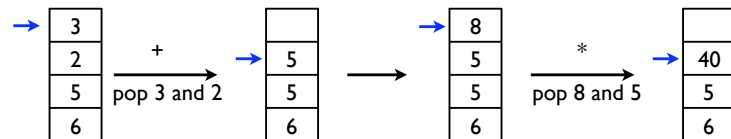
Stack ADT: postfix evaluation

- ▶ scan left-to-right
- ▶ place the operands on a stack until an operator is found
- ▶ perform operations by popping two elements in the stack when an operator is found

Stack Post-fix
infix-to-postfix

Stack ADT: postfix evaluation

6 5 2 3 + 8 * + 3 + * → TopOfStack



Stack ADT: postfix evaluation

10 2 8 * + 3 -

Stack ADT: translation of infix to postfix

$3 + 4 * 6 \longrightarrow 3\ 4\ 6\ * +$

$(3 + 4) * 6 \longrightarrow 3\ 4 + 6 *$

$3 + (4 * 6) \longrightarrow 3\ 4\ 6\ * +$

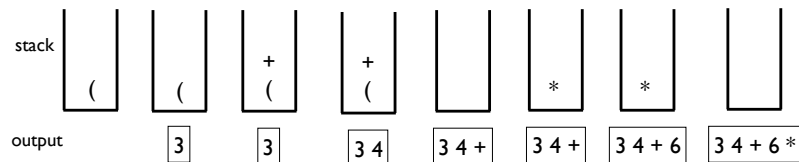
Stack ADT: translation of infix to postfix

- When you meet an **operand**, print it.
- When you meet an **operator**, push it as long as the precedence of the **operator at the top of the stack is less** than the precedence of the incoming operator.
- When you meet an operator whose precedence is **equal to or less than the precedence of the top of the stack**, pop the top element and print it.
- When you meet the **right parenthesis**, pop all the operators until we reach the corresponding left parenthesis.
- When you meet the left parenthesis, push it in the stack.
- When you reach the **end of expression**, pop all the operators from the stack.

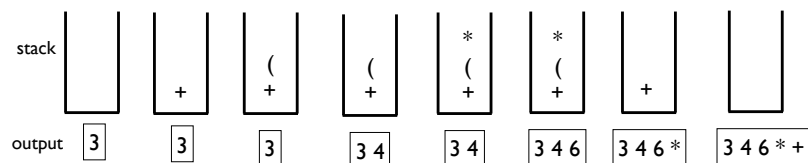
가

Stack ADT: translation of infix to postfix

$(3 + 4) * 6 \longrightarrow 3\ 4 + 6 *$



$3 + (4 * 6) \longrightarrow 3\ 4\ 6\ * +$

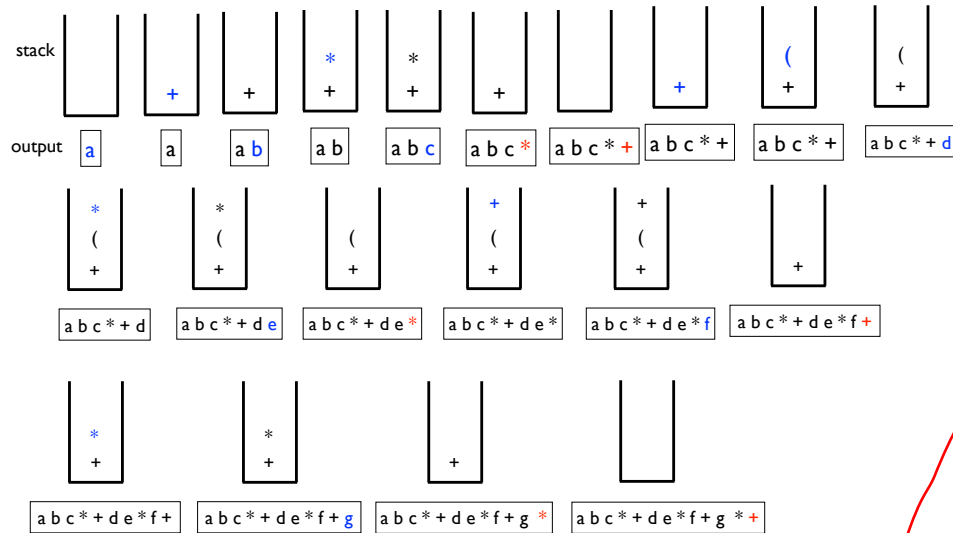


Stack ADT: translation of infix to postfix

$a/b-c+d*e-a*c$

Stack ADT: translation of infix to postfix

$a + b * c + (d * e + f) * g \rightarrow a b c * + d e * f + g * +$



Stack ADT: translation of infix to postfix

infix	postfix
2 + 3 * 4	2 3 4 * +
a * b + 5	a b * 5 +
(1 + 2) * 7	1 2 + 7 *
a * b / c	a b * c /
((a / (b - c + d)) * (e - a)) * c	a b c - d + / e a - * c *
a/b-c+d*e-a*c	a b / c - d e * + a c *

Quiz for... pdf

Q. Performance Analysis of Stack DS

we implement STACK using Array implementation, Linked list implementation, respectively.

Now, we assume the size of input data is N and the ADT of Stack is written in this pdf slide. Write your answers for each question below.

(Should answer your solution using Asymptotic Analysis)

Q1. in case of using array implementation, What is the running time of Push and Pop Operation? and What is the worst-case of POP operation?

1) Push :

2) Pop :

3) Worst-case :

Q2. in case of using Linked List implementation, What is the running time of Push and Pop Operation?

1) Push :

2) Pop :

3) Worst-case :

Q3. What is the space-complexity of Array implementation and Linked List?

Performance Analysis

Asymptotic Analysis()

2

100%.

2

PPT

Source Code

Time complexit

Space Complexity