Stacks & Queues

Kuan-Yu Chen (陳冠宇)

2018/10/01 @ TR-212, NTUST

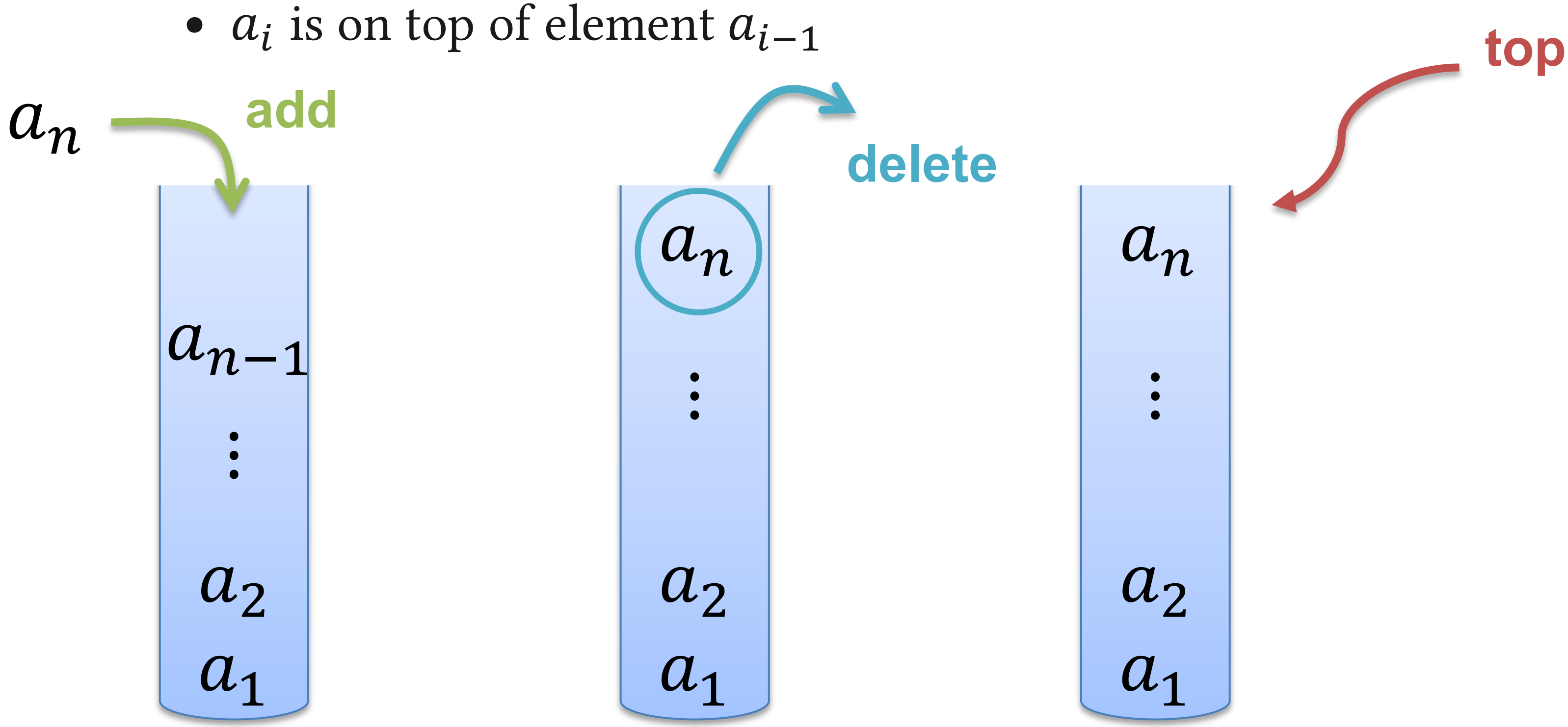
Review

- Stack
  - Stack Permutation

- Expression
  - Infix
  - Prefix
  - Postfix

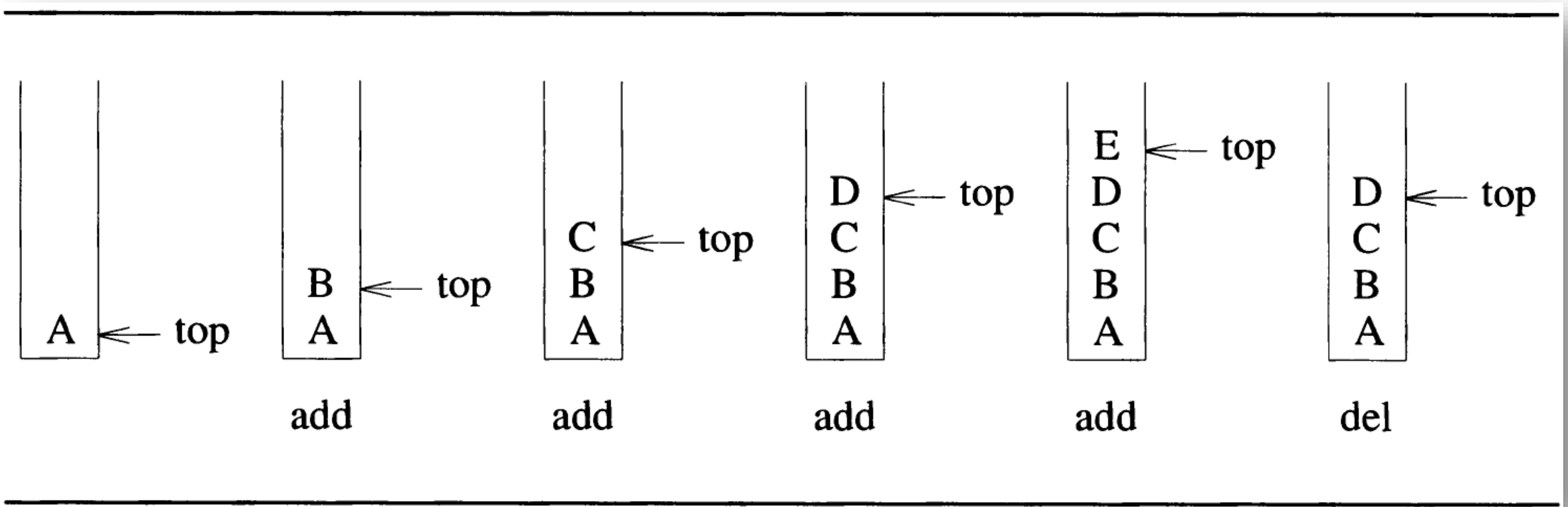
Stacks.

- A **stack** is an **ordered** list in which insertions and deletions are made at one end called the **top**
  - Given a stack $S = (a_1, a_2, \dots, a_n)$
    - a_1 is the bottom element
    - a_n is the top element
    - a_i is on top of element a_{i-1}



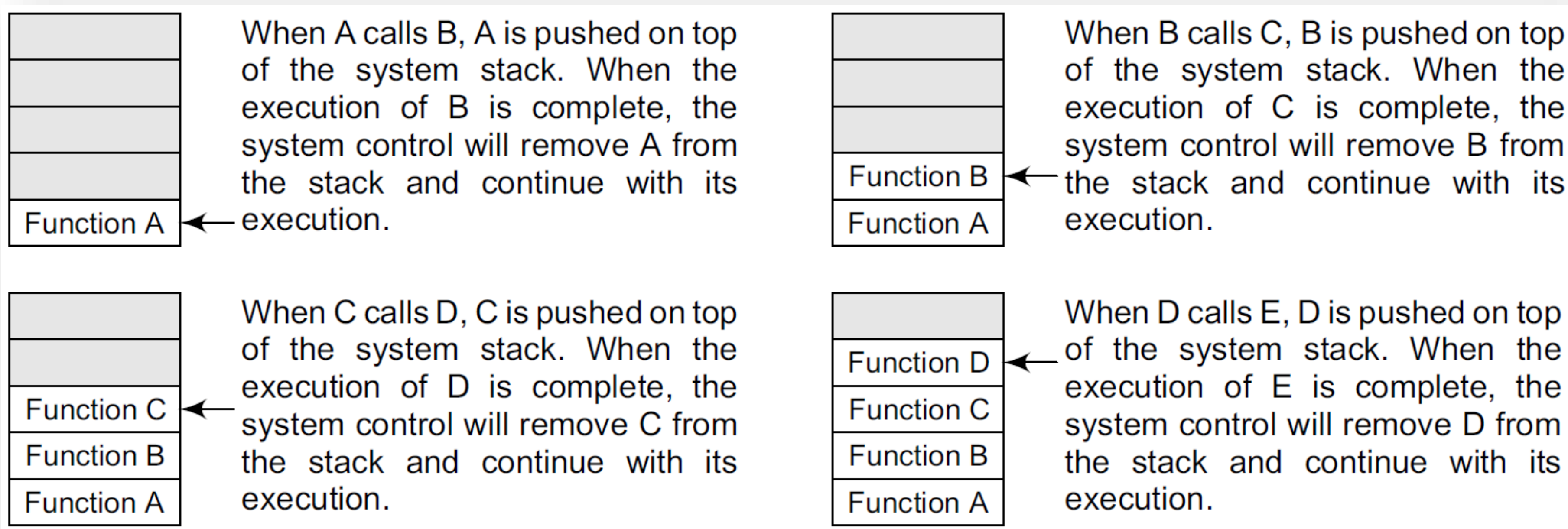
Stacks..

- By the definition of stack, if we add the elements A, B, C, D, E to the stack, in that order, then E is the first element we delete from the stack
  - **Last-In-First-Out**



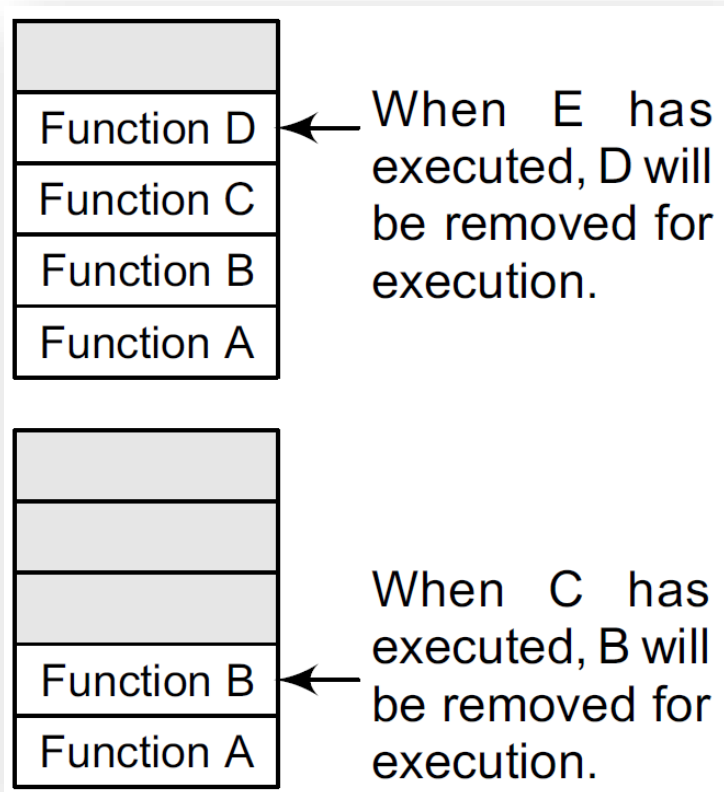
Applications.

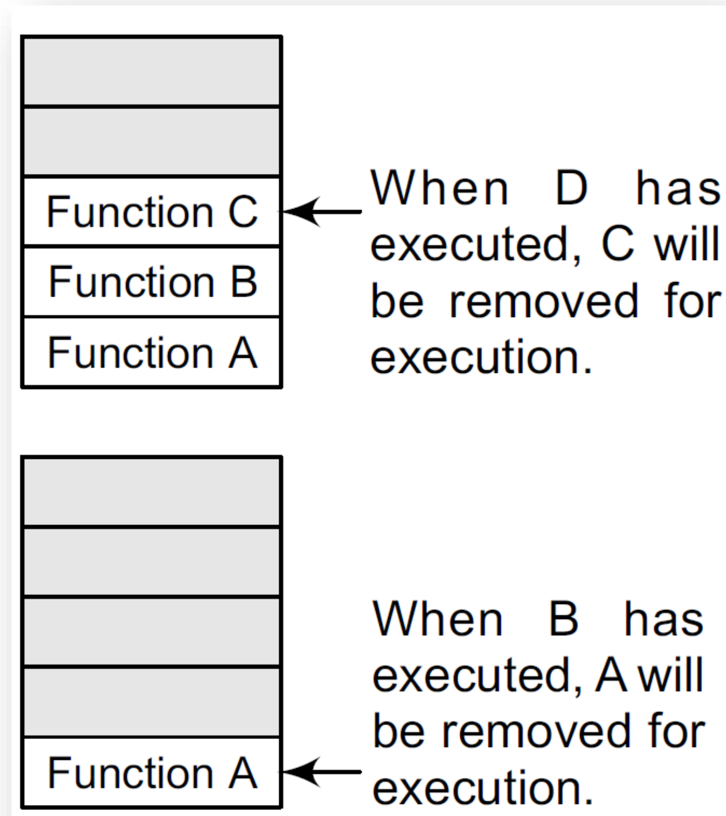
- System stack in the case of function calls



Applications..

- System stack in the case of function calls





Applications...

- For a recursive function, the stack can be used to store the processing status
  - Given an Ackerman's function $A(m,n)$, please calculate $A(1,2)$

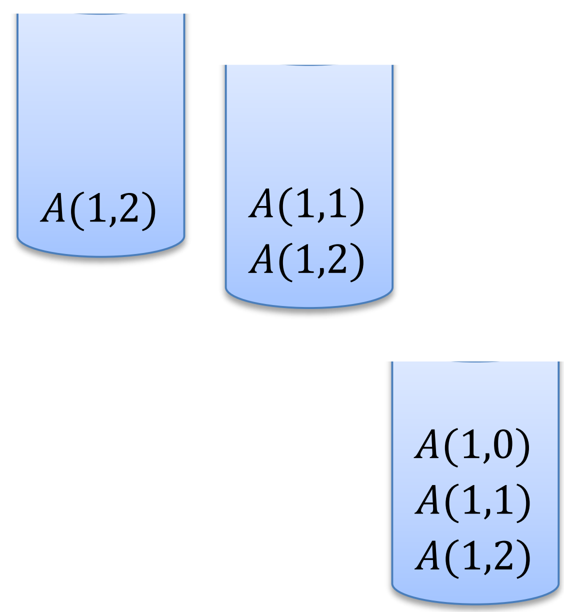
$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$A(1,2) = A\big(0, A(1,1)\big)$

$A(1,1) = A\big(0, A(1,0)\big)$

$A(1,0) = A(0,1)$

$A(0,1) = 2$



Applications....

- For a recursive function, the stack can be used to store the processing status
  - Given an Ackerman's function $A(m,n)$, please calculate $A(1,2)$

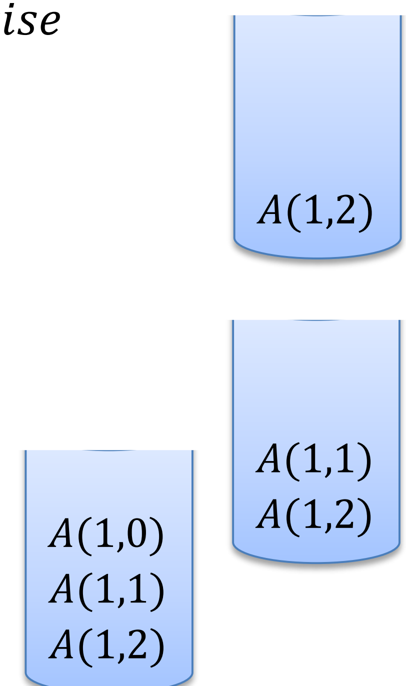
$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big) = A(0,3) = 4$$

$$A(1,1) = A\big(0, A(1,0)\big) = A(0,2) = 3$$

$$A(1,0) = A(0,1) = 2$$

$$A(0,1) = 2$$

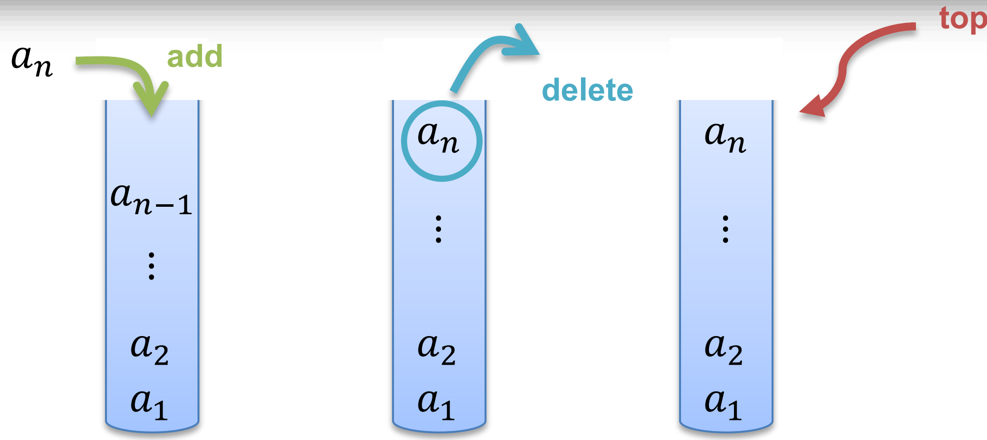


Implementation for Stack by Array.

- Declare

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 3 // Altering this value changes size of stack created

int st[MAX], top=-1;
void push(int st[], int val);
int pop(int st[]);
int peek(int st[]);
void display(int st[]);
```



Implementation for Stack by Array..

- For "push"

```
void push(int st[], int val)
{
        if(top == MAX-1)
        {
                printf("\n STACK OVERFLOW");
        }
        else
        {
                top++;
                st[top] = val;
        }
}
```

Implementation for Stack by Array…

- For “pop”

```
int pop(int st[])
{
        int val;
        if(top == -1)
        {
                printf("\n STACK UNDERFLOW");
                return -1;
        }
        else
        {
                val = st[top];
                top--;
                return val;
        }
}
```

Implementation for Stack by Array….

- For "display"

```
void display(int st[])
{
        int i;
        if(top == -1)
        printf("\n STACK IS EMPTY");
        else
        {
                for(i=top;i>=0;i--)
                printf("\n %d",st[i]);
                printf("\n"); // Added for formatting purposes
        }
}
```

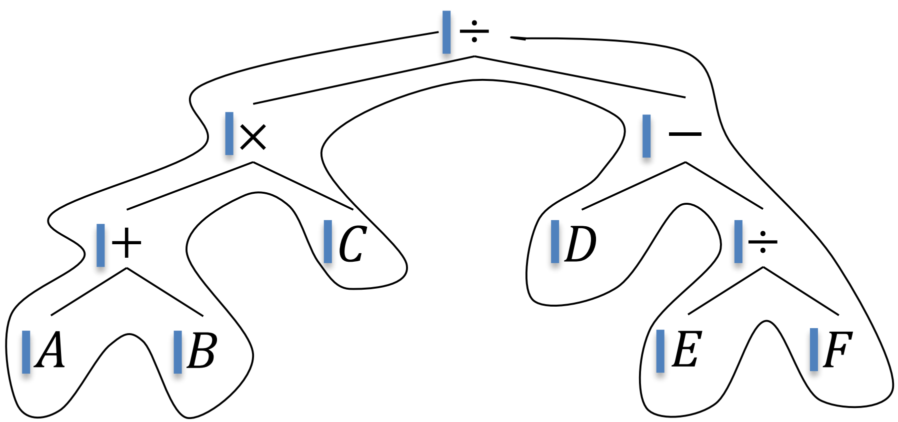
Implementation for Stack by Array…..

- For "peek"

```
int peek(int st[])
{
        if(top == -1)
        {
                printf("\n STACK IS EMPTY");
                return -1;
        }
        else
        return (st[top]);
}
```

Prefix Expression

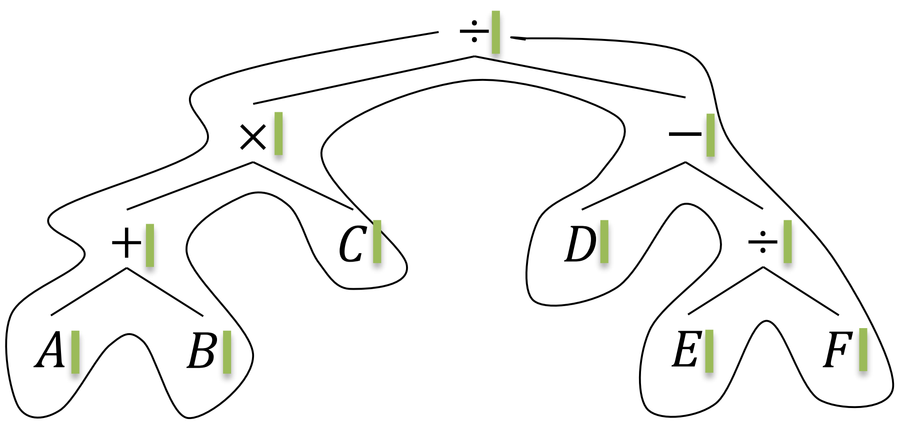
- Given a infix expression $(A + B) \times C \div (D - E \div F)$, please write down the prefix and postfix expressions
  - Prefix



$\div \times +ABC - D \div EF$

Postfix Expression

- Given a infix expression $(A + B) \times C \div (D - E \div F)$, please write down the prefix and postfix expressions
  - Postfix



$AB + C \times DEF \div - \div$

Algorithm to Convert Infix to Postfix.

```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
       IF a "(" is encountered, push it on the stack
       IF an operand (whether a digit or a character) is encountered, add it to the
       postfix expression.
       IF a ")" is encountered, then
         a. Repeatedly pop from stack and add it to the postfix expression until a
            "(" is encountered.
         b. Discard the "(". That is, remove the "(" from stack and do not
            add it to the postfix expression
       IF an operator O is encountered, then
         a. Repeatedly pop from stack and add each operator (popped from the stack) to the
            postfix expression which has the same precedence or a higher precedence than O
         b. Push the operator O to the stack
       [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```

Algorithm to Convert Infix to Postfix..

- Take $A - (B \div C + (D\%E \times F) \div G) \times H$ for example

Infix Scanned	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
(	(- (	A
B	(- (	A B
/	(- (/	A B
C	(- (/	A B C
+	(- (+	A B C /
(	(- (+ (	A B C /
D	(- (+ (	A B C / D
%	(- (+ (%	A B C / D
E	(- (+ (%	A B C / D E

```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
        IF a "(" is encountered, push it on the stack
        IF an operand (whether a digit or a character) is encountered, add it to the
        postfix expression.
        IF a ")" is encountered, then
          a. Repeatedly pop from stack and add it to the postfix expression until a
             "(" is encountered.
          b. Discard the "(". That is, remove the "(" from stack and do not
             add it to the postfix expression
        IF an operator O is encountered, then
          a. Repeatedly pop from stack and add each operator (popped from the stack) to the
             postfix expression which has the same precedence or a higher precedence than O
          b. Push the operator O to the stack
        [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```

Algorithm to Convert Infix to Postfix...

- Take $A - (B \div C + (D\%E \times F) \div G) \times H$ for example

Infix Scanned	Stack	Postfix Expression
E	(- (+ (%	A B C / D E
*	(- (+ (*	A B C / D E %
F	(- (+ (*	A B C / D E % F
)	(- (+	A B C / D E % F *
/	(- (+ /	A B C / D E % F *
G	(- (+ /	A B C / D E % F * G
)	(-	A B C / D E % F * G / +
*	(- *	A B C / D E % F * G / +
H	(- *	A B C / D E % F * G / + H
)		A B C / D E % F * G / + H * -

Algorithm to Convert Infix to Prefix – 1

```
Step 1: Scan each character in the infix
        expression. For this, repeat Steps
        2-8 until the end of infix expression
Step 2: Push the operator into the operator stack,
        operand into the operand stack, and
        ignore all the left parentheses until
        a right parenthesis is encountered
Step 3: Pop operand 2 from operand stack
Step 4: Pop operand 1 from operand stack
Step 5: Pop operator from operator stack
Step 6: Concatenate operator and operand 1
Step 7: Concatenate result with operand 2
Step 8: Push result into the operand stack
Step 9: END
```

Algorithm to Convert Infix to Prefix – 2.

```
Step 1: Reverse the infix string. Note that
        while reversing the string you must
        interchange left and right parentheses.
Step 2: Obtain the postfix expression of the
        infix expression obtained in Step 1.
Step 3: Reverse the postfix expression to get
        the prefix expression
```

- Take $(A - B \div C) \times (A \div K - L)$ for example
  - Step1: $(L - K \div A) \times (C \div B - A)$
  - Step2: $LKA \div -CB \div A - \times$
  - Step3: $\times -A \div BC - \div AKL$

Homeork1: Expression Convertor.

- Given a infix expression, please convert the expression to **both** prefix and postfix expressions
  - The implementation **must** base on stack
  - Please show the results **step-by-step**
  - Please upload your source codes and a paper report to moodle
  - TA will ask you to demo your program
  - The **hard deadline** is 2018/10/15 8:00

Infix Scanned	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
(	(- (	A
B	(- (	A B
/	(- (/	A B
C	(- (/	A B C

Homeork1: Expression Convertor..

- Given a infix expression, please convert the expression to **both** prefix and postfix expressions
  - The maximum length of the input expression will always less than 30
  - Only five operators need to be considered
    - +, −, ×, ÷, %
  - The operands are capital letters (i.e., A~Z)

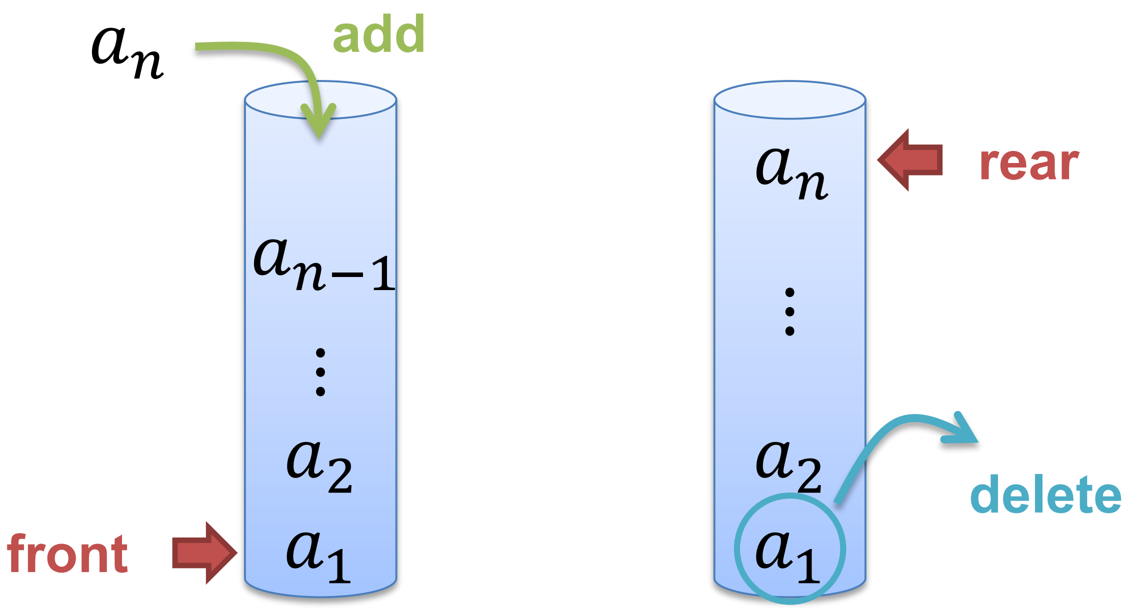
Infix Scanned	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
(	(- (	A
B	(- (	A B
/	(- (/	A B
C	(- (/	A B C

Homeork1: Expression Convertor..

- Given a infix expression, please convert the expression to **both** prefix and postfix expressions
  - $(A - B \div C) \times (A \div K - L)$
    - Prefix: $\times -A \div BC -\div AKL$
    - Postfix: $ABC \div -AK \div L -\times$
  - $A - (B \div C + (D\%E \times F) \div G) \times H$
    - Prefix: $-A \times + \div BC \div \times \%DEFGH$
    - Postfix: $ABC \div DE\%F \times G \div +H \times -$

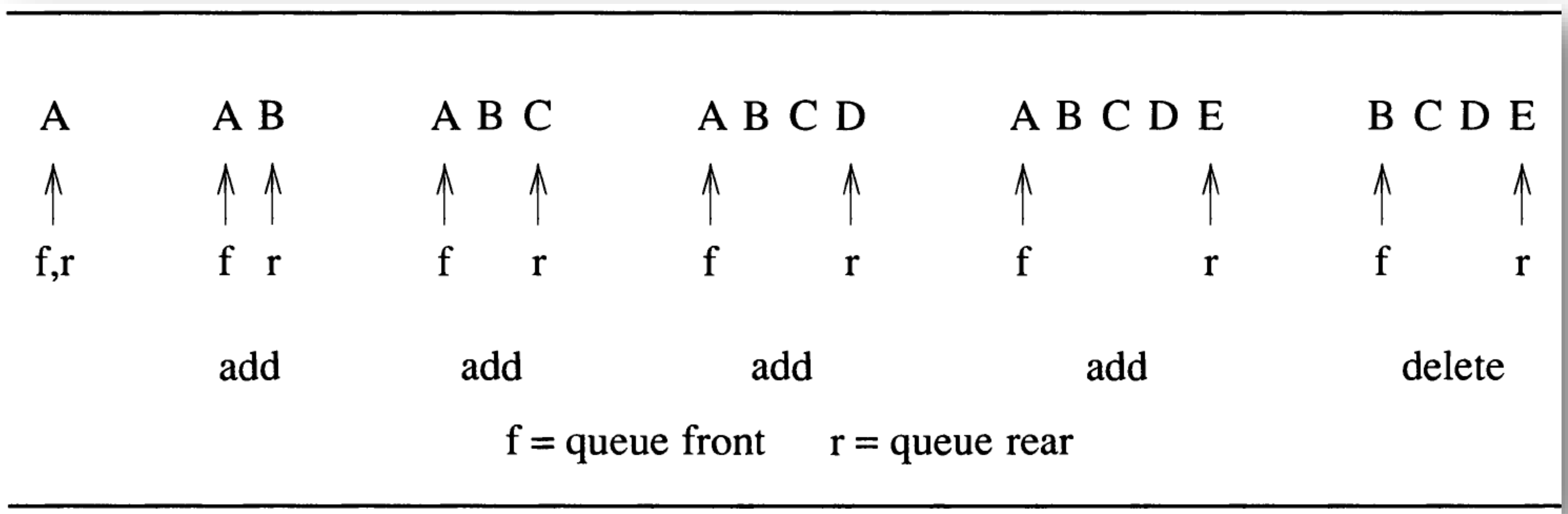
Queue.

- A **queue** is an **ordered** list in which insertions take place at one end (**rare**) and deletions are made at the opposite end (**front**)
  - Given a queue $Q = (a_1, a_2, \dots, a_n)$
    - a_1 is the front element
    - a_n is the rear element
    - a_i is behind element a_{i-1}



Queue..

- By the definition of queue, if we insert the elements A, B, C, D, E in the order, then A is the first element deleted from the queue
  - **First-In-First-Out**



Applications – Queue

- Job scheduling
  - A fair method

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

Array Representation of Queues

- Queues can be easily represented using arrays
  - Given a queue

12	9	7	18	14	36				
0	1	2	3	4	5	6	7	8	9

  - Insert an element

12	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

  - Delete an element

	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

Implementation for Queue by Array.

- Declare

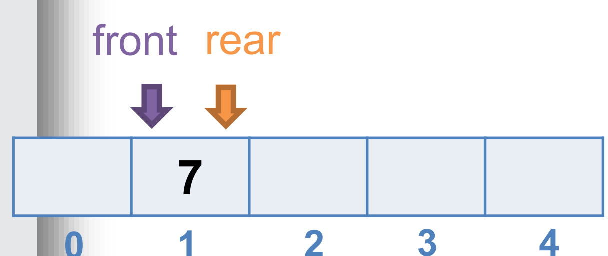
```
#define MAX 10 // Changing this value will change length of array
int queue[MAX];
int front = -1, rear = -1;
void insert(void);
int delete_element(void);
void display(void);
```

Implementation for Queue by Array..

```
void insert()
{
        int num;
        printf("\n Enter the number to be inserted in the queue : ");
        scanf("%d", &num);
        if(rear == MAX-1)
        printf("\n OVERFLOW");
        else if(front == -1 && rear == -1)
        front = rear = 0;
        else
        rear++;
        queue[rear] = num;   ?
}
```

Implementation for Queue by Array...

```
int delete_element()
{
        int val;
        if(front == -1 || front>rear)
        {
                printf("\n UNDERFLOW");
                return -1;
        }
        else
        {
                val = queue[front];
                front++;
                if(front > rear)
                front = rear = -1;
                return val;
        }
}
```



Implementation for Queue by Array....

```
void display()
{
        int i;
        printf("\n");
        if(front == -1 || front > rear)
        printf("\n QUEUE IS EMPTY");
        else
        {
                for(i = front;i <= rear;i++)
                printf("\t %d", queue[i]);
        }
}
```

Questions?

kychen@mail.ntust.edu.tw