Stacks and Queues

Ikjun Yeom

Originated from Prof. Kyungran Kang

Contents

Stacks

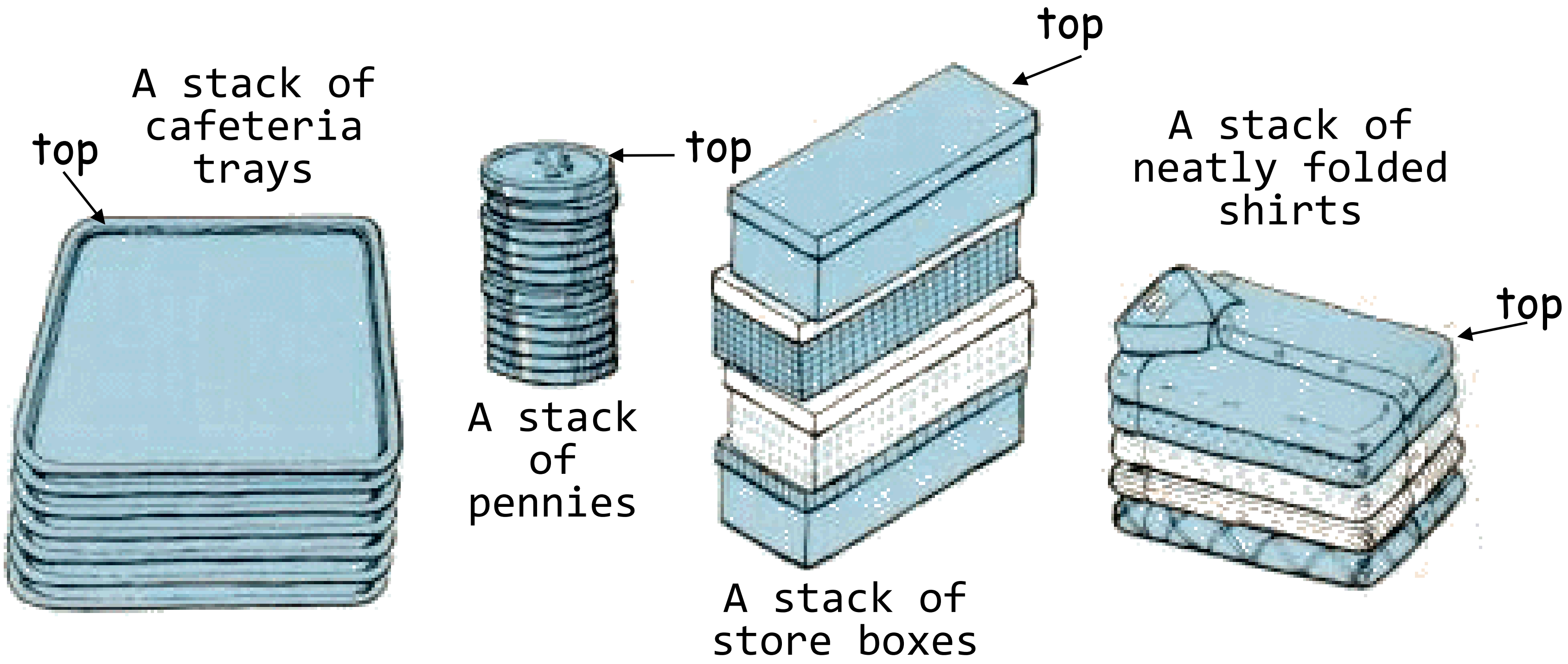
Queues

Circular Queues

A Maze Problem

Evaluation of Expressions

Examples of Stack



top

A stack of
cafeteria
trays

top

A stack
of
pennies

top

A stack of
store boxes

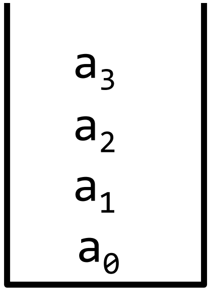
A stack of
neatly folded
shirts

top

Stack

Definition

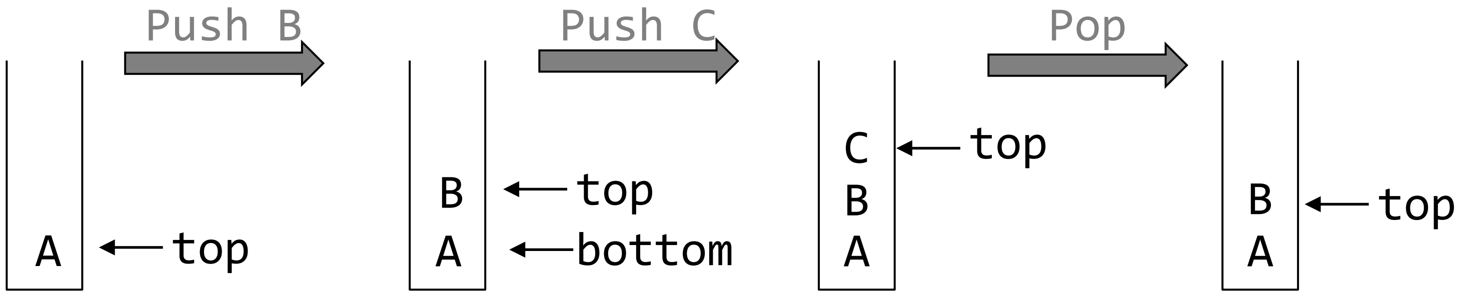
- An ordered list in which insertions and deletions are made at one end called the *top*
- Given a stack $S=(a_0,\ldots,a_{n-1})$
  - a_0 is bottom element
  - a_{n-1} is top element
  - a_i is on top of element a_{i-1}, $0<i<n$



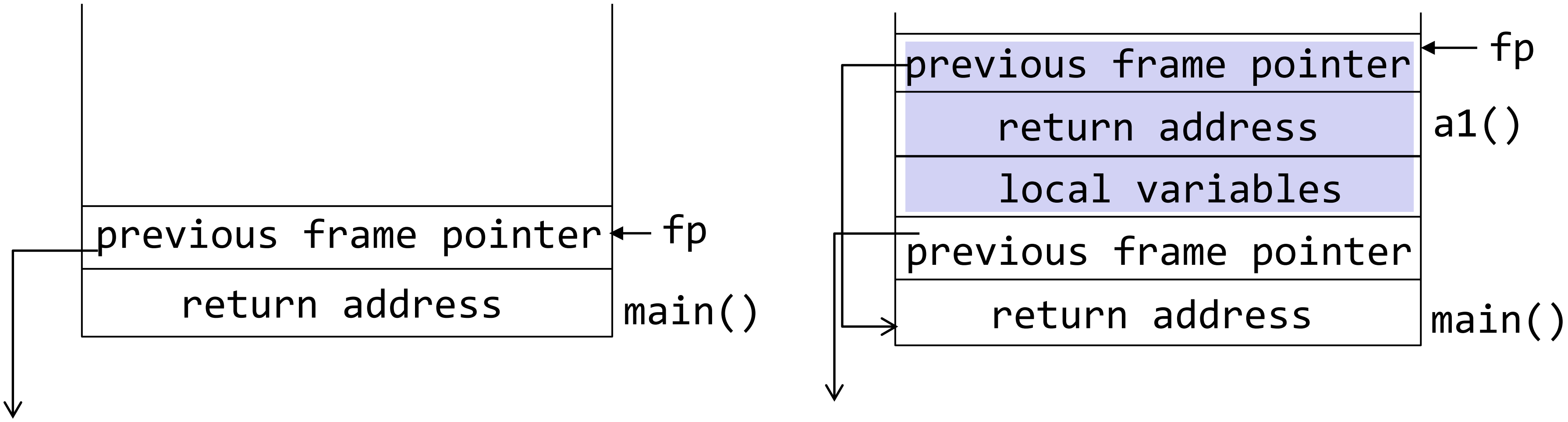
- *Last-In-First-Out (LIFO)*
  - Insert the new element into the stack on the top end
  - We can only delete and get the top element of the stack

Examples of Stack

A thin box



Process stack **(p.108, Exmaple 3.1)**



Stack Abstract Data Type

```
ADT Stack is
  objects: A finite ordered list with zero or more elements
  functions: for all stack $\in$Stack,item $\in$ element,
                 max_stack_size $\in$ positive integer
     Stack CreateS(maxStackSize) ::= create an empty stack whose maximum size is maxStackSize
     Boolean IsFull(stack, maxStackSize) ::= …
     Stack Push(stack, item) ::=
         if (isFull(stack) stackFull
         else insert item into top of stack and return
     Boolean IsEmpty(stack) ::=
     Element Pop(stack) ::=
        if (isEmpty(stack) return
        else remove and return the element at the top of the stack
```

Stack Implementation (1)

```
#define MAX_STACK_SIZE 100 /*maximum stack size  */
typedef struct {
  int key;
  /* other fields may be added*/
}  element;
element stack[MAX_STACK_SIZE];
int top = -1;

void push (element item)
{
   /* add an item to the global stack */
   if (top >= MAX_STACK_SIZE-1) {
      stackFull();
      return;
   }
   stack[++top] = item;
}
```

```
void stackFull()
{
   fprintf(stderr, “Stack is full, cannot add element”);
   exit(EXIT_FAILURE);
}
```

Stack Implementation (2)

```
element pop ()
{
/* return the top element from the stack, so called ‘pop’
   */
   if (top == -1)
     return stackEmpty();/*returns an error key */
   return stack[top--];
}
```

```
main()
{
  element e,f;

  e.key=3;   push(e);
  e.key=2;   push(e);
  f=pop();
  printf (“%d  %d\n”, top, f.key);
}
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
stack	3	2						

```
0   2
```

Quiz 13

Name and student ID

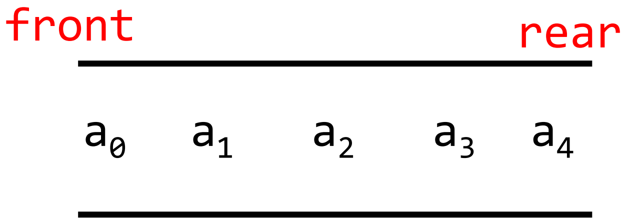
Using the stack implementation in Slide 7 and 8, write a program to generate ten random integer numbers, and push them to a stack. Then print out in the order of first-in-first-out (not last-in-first-out).

Queue Abstract Data Type

Definition

- An ordered list in which all insertions take place at one end (rear) and all deletions take place at the opposite end (front)

- Given a queue Q=(a_0,…, a_{n-1})
  - a_0 is front element
  - a_{n-1} is rear element
  - a_i is behind a_{i-1}, $0<i<n$



front

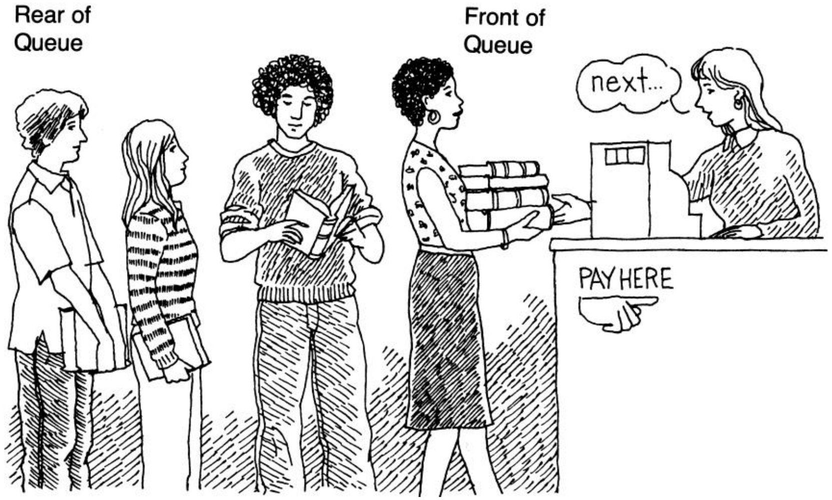
rear

a_0 a_1 a_2 a_3 a_4

- *First-In-First-Out* (*FIFO*)
  - Insert the new element into the queue on the rear side
  - We can only delete/get the front element of the queue

Example of Queue

In a shop



Job scheduling

- Frequently used in computer programming
- Job queue by an operating system
- The jobs are processed in the order they enter the system

Queue Abstract Data Type

```
ADT Queue is
  objects: A finite ordered list with zero or more
           elements
  functions: for queue ∈Queue,item ∈ element,
                       maxQueueSize ∈ positive integer
    Queue CreateQ(queue, maxQueueSize)::= …
    Boolean IsFullQ(queue)::=…
    Queue AddQ(queue,item)::=
        if (IsFullQ(queue)) queueFull
        else insert item at rear or queue and return queue
    Boolean IsEmptyQ(queue) ::= …
    Element DeleteQ(queue) ::=
        if (IsEmpty(queue)) return
        else remove item and return the item at the front of
   queue
```

Queue Implementation (1)

```
#define MAX_QUEUE_SIZE 100 /*maximum queue size  */
typedef struct {
  int key;
  /* other fields may be added*/
}  element;
element queue[MAX_QUEUE_SIZE];

int rear = -1; int front = -1;

void addq(element item)
{
/* insert an item into a queue, so called 'enqueue' */
  if (rear == MAX_QUEUE_SIZE-1)
       queueFull();
  queue[++rear] = item;
}
```

Queue Implementation (2)

```
element deleteq()
{
   /* delete an item at the front of the queue, so called
   'dequeue' */
   if (front == rear)
       return queueEmpty(); /* return an error key */
   return queue[++front];
}
```

```
main()
{
  element e,f;

  e.key=3;   addq(e);
  e.key=2;   addq(e);
  f=deleteq();
  printf("%d %d %d\n", front, rear, f.key);
}
```

	[0]	[1]	[2]	[3]	[4]
queue	3	2			

```
0    1    3
```

Job Scheduling Example

Insertion and deletion from a sequential queue

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					Queue is empty
-1	0	J1				Job1 is added
-1	1	J1	J2			Job2 is added
-1	2	J1	J2	J3		Job3 is added
0	2		J2	J3		Job1 is deleted
1	2			J3		Job2 is deleted
1	3			J3	J4	Job4 is added

→ The queue gradually shifts to the right

→ No available space to add a new item when rear is (MAX_QUEUE_SIZE – 1)

→ Circular representation is more efficient to avoid the problem

Circular Queue (1)

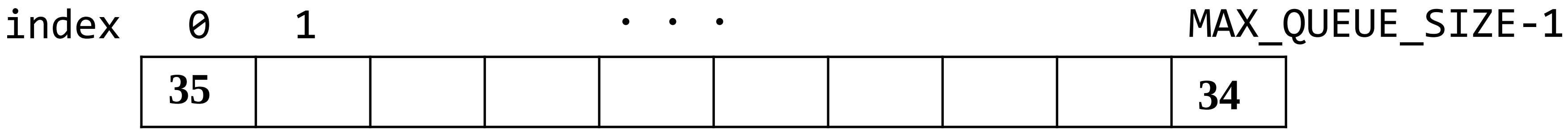
A queue wraps around the end of the array

Array positions are arranged in a circle rather than in a straight line

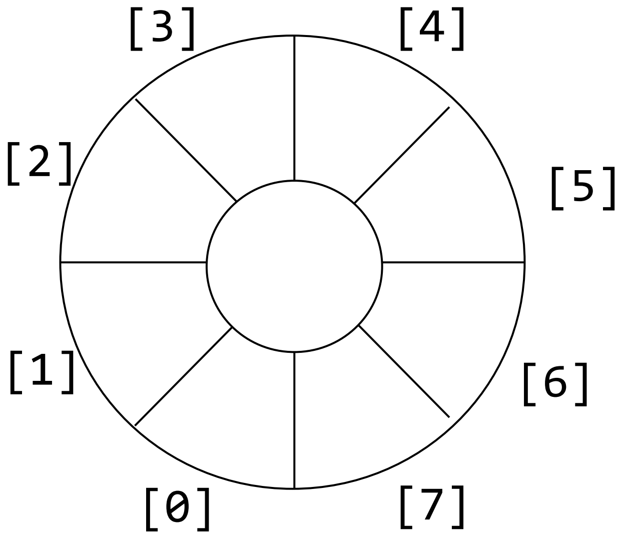
- The position next to position MAX_QUEUE_SIZE-1 is 0
- The position precedes 0 is MAX_QUEUE_SIZE-1

```
 if (rear==MAX_QUEUE_SIZE-1) rear = 0;
   else rear++;
```

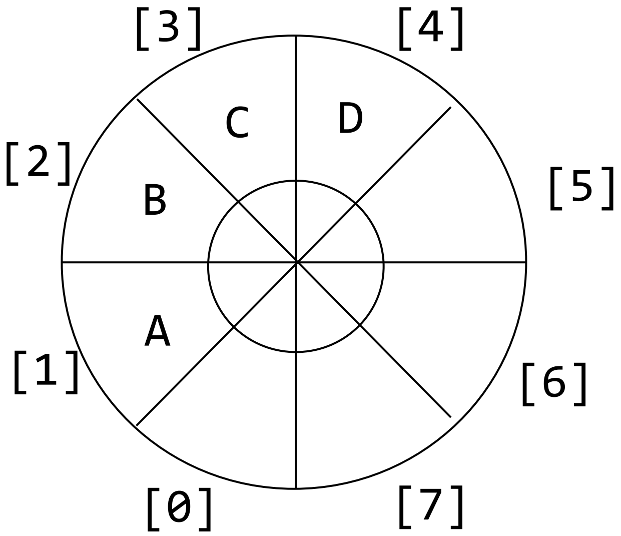
```
→ rear = (rear + 1) % MAX_QUEUE_SIZE
```



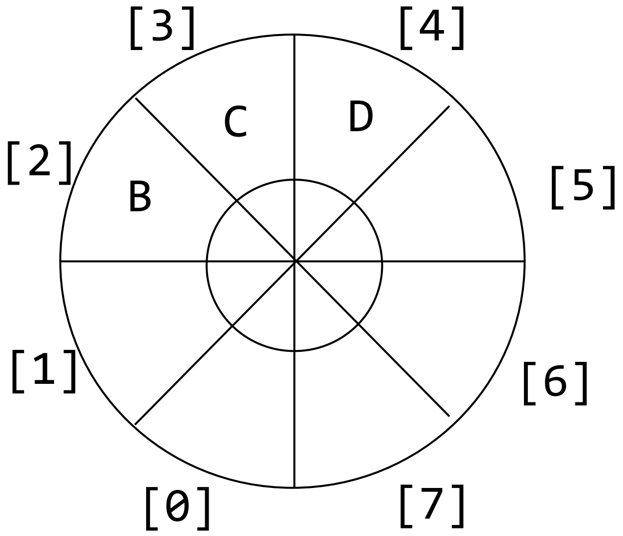
Circular Queue Operation



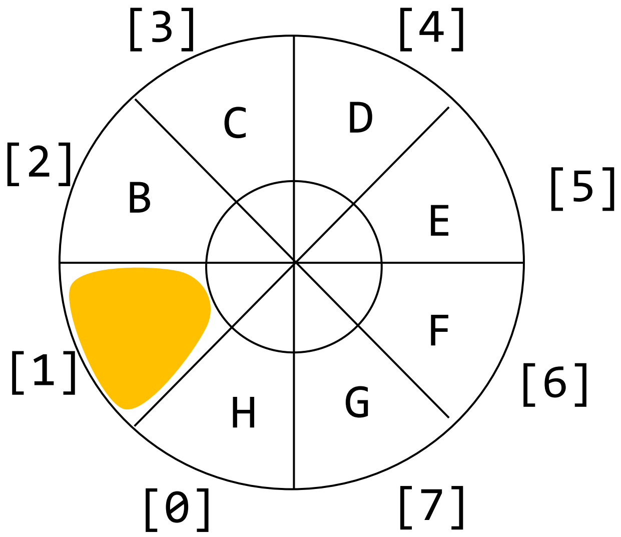
front=0
rear=0



front=0
rear=4



front=1
rear=4



front=1
rear=0

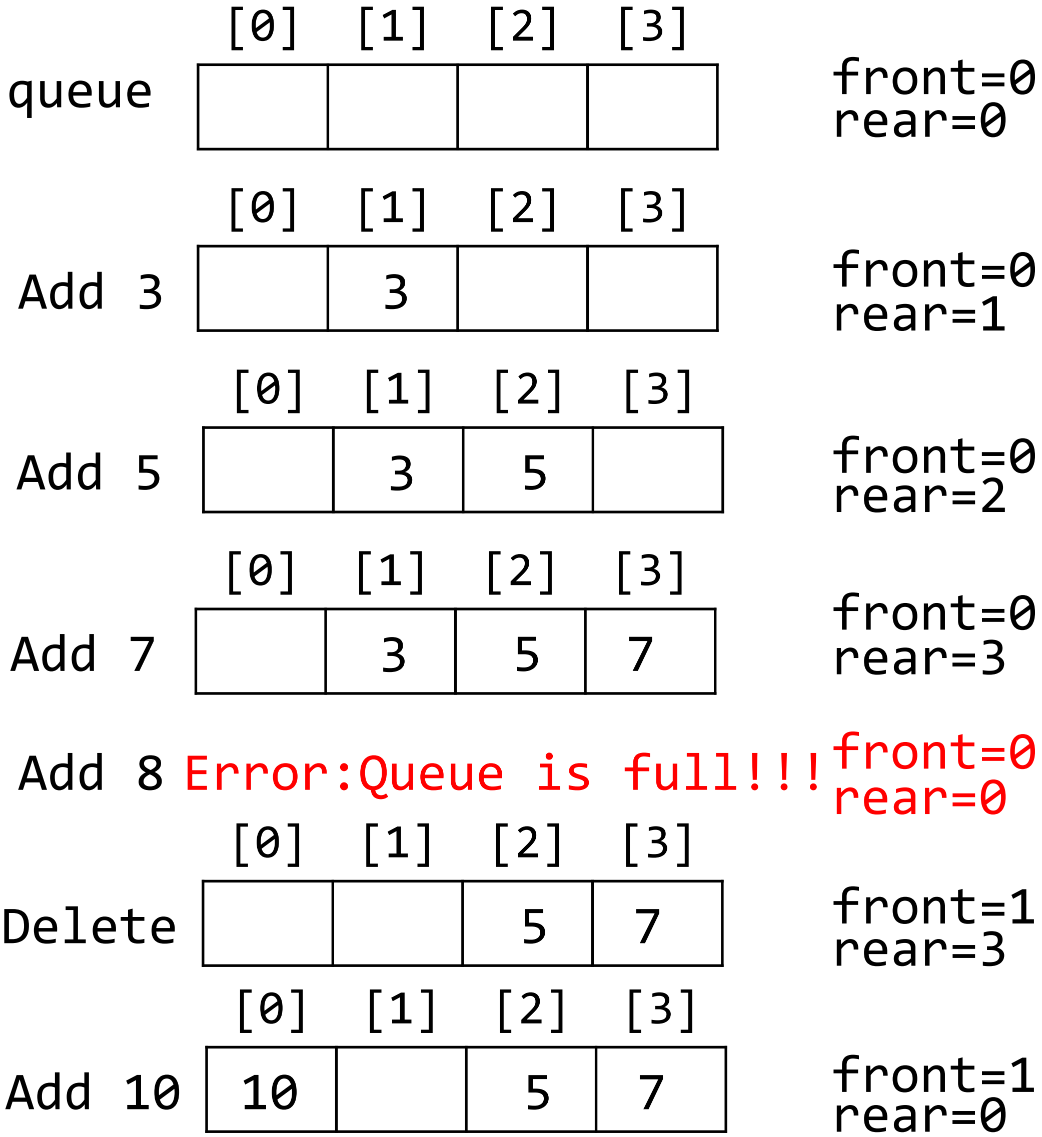
Circular Queue (2)

```
element queue[MAX_QUEUE_SIZE];
int front=0, rear=0;

void addq(element item)
{
   rear = (rear+1) % MAX_QUEUE_SIZE;
   if (front == rear)
      queueFull(rear); /* print error and exit */
   queue[rear] = item;
}

element deleteq(){
   if (front == rear)
       return queueEmpty();
   front = (front+1) % MAX_QUEUE_SIZE;
   return queue[front];
}
```

Circular Queue Operation Example



	[0]	[1]	[2]	[3]	
queue					front=0 rear=0
Add 3		3			front=0 rear=1
Add 5		3	5		front=0 rear=2
Add 7		3	5	7	front=0 rear=3
Add 8	Error:Queue is full!!!				front=0 rear=0
Delete			5	7	front=1 rear=3
Add 10	10		5	7	front=1 rear=0

Quiz 14

Name and student ID

Modify the circular queue implementation in Slide 18 so that we can utilize all the space.

Applications of Stack

Many application areas use stacks:

- Line editing
- Bracket matching
- Maze problem
- Expression evaluation

a b c d ←

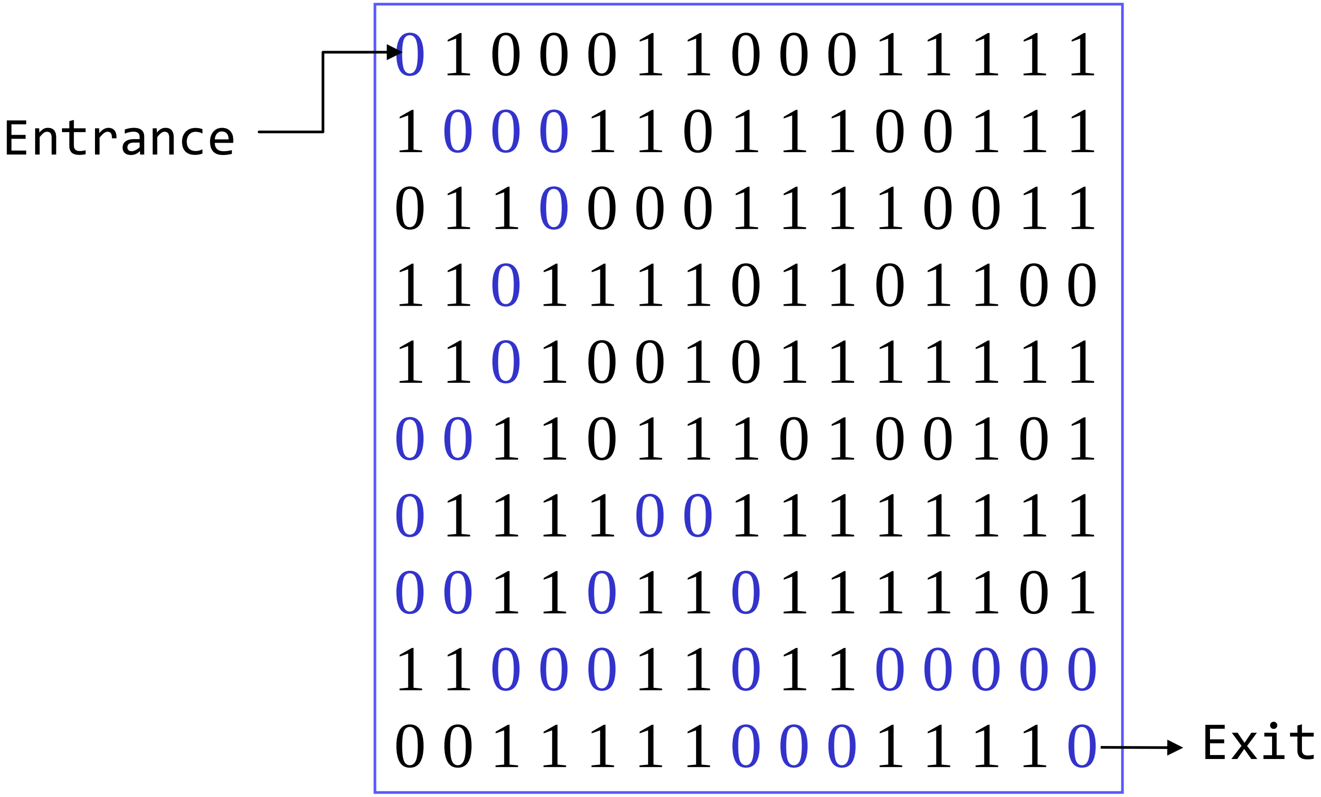
{a,(b+f[4])*3,d+f[5]}

A Maze Problem

Problem

- A value 1 implies a blocked path, and 0 means one can walk right on through.
- Find the way to go out

```
int maze[MAX_ROWS][MAX_COLS];
```



```
0 1 0 0 0 1 1 0 0 0 1 1 1 1 1
1 0 0 0 1 1 0 1 1 1 0 0 1 1 1
0 1 1 0 0 0 0 1 1 1 1 0 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 0 0
1 1 0 1 0 0 1 0 1 1 1 1 1 1 1
0 0 1 1 0 1 1 1 0 1 0 0 1 0 1
0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
1 1 0 0 0 1 1 0 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 1 1 1 1 0
```

Strategy

We may have the chance to go in several directions

Pick one and save our current position and the direction of the next move in the list (stack)

If we have taken a false path, we can return and try another direction by getting the top element of the stack

Allowable Moves (1)

Allowable moves

NW [i-1][j-1]	N [i-1][j]	NE [i-1][j+1]
W [i][j-1]	X [i][j]	E [i][j+1]
SW [i+1][j-1]	S [i+1][j]	SE [i+1][j+1]

```
typedef struct {
    short int vert;  /* -1, 0, +1 */
    short int horiz; /* -1, 0, +1 */
    } offsets;
```

Allowable Moves (2)

Table of moves

Name	Dir	Move[dir].vert	Move[dir].horiz
N	0	-1	0
NE	1	-1	1
E	2	0	1
SE	3	1	1
S	4	1	0
SW	5	1	-1
W	6	0	-1
NW	7	1	-1

```
offsets move[8];
nextRow = row + move[dir].vert;
nextCol = col + move[dir].horiz;
```

Movement history

```
#define MAX_STACK_SIZE 100
typedef struct {
       short int row; /* current position */
       short int col; /* current position */
       short int dir; /* direction of next move */
       } element;
element stack[MAX_STACK_SIZE]
```

Pick one and **save our current position and the direction of the next move** in the list (stack).

If we have taken a false path, we can return and try another direction by getting the top element of the stack

Quiz 15

Name and student ID

What is the maximum path length from start to finish for any maze of dimensions rows x columns?