DET102 Data Structures and Algorithms

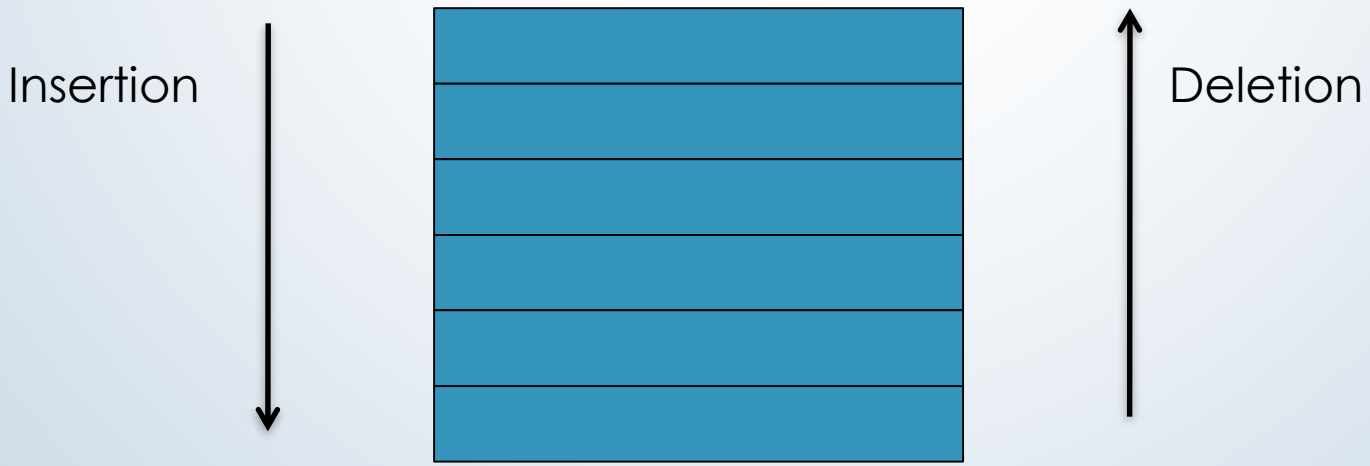
Lecture 03: Stack and Queue

Stack

First In Last Out

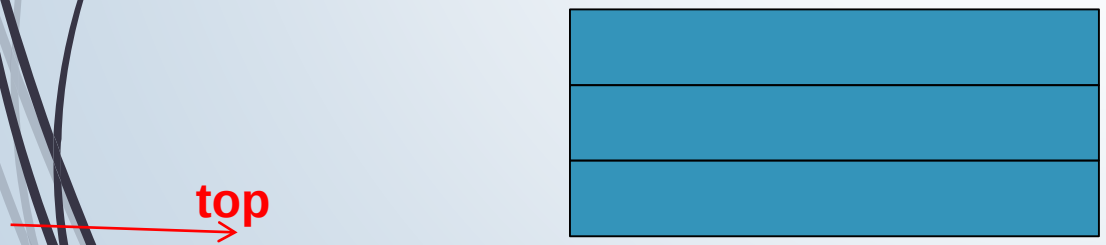
Stack

- Stack is a data structure with the restriction that insertions and deletions (usually all the accesses) can only be performed at one end of the list.
- Rule: Last-in-first-out (LIFO)



ADT of stack

- Value
  - A sequence of items that belong to some data type ITEM_TYPE
- Top pointer
  - Nodes are inserted and removed at the same end of the list.

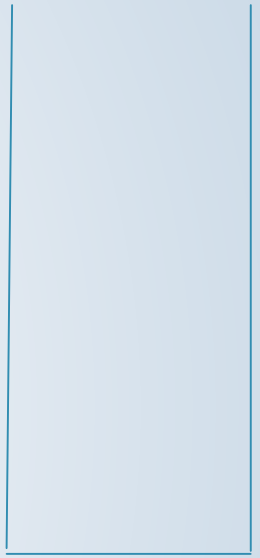


Operations

- stack<Type> s
- push(x)
- pop()
- top()
- size()
- empty()

- A stack is an abstract data type (ADT) such that an instance S supports the following two methods:
  - S.push(x): Add element x to the top of stack S.
  - S.pop(): Remove and return the top element from the stack S; an error occurs if the stack is empty.
- The following accessory methods for convenience:
  - S.top(): Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.
  - S.empty( ): Return True if stack S does not contain any elements.
  - S.size(): Return the number of elements in stack S; in Python, we implement this with the special method len.

Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop( )	3	[5]
S.is_empty( )	False	[5]
S.pop( )	5	[ ]
S.is_empty( )	True	[ ]
S.pop( )	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top( )	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop( )	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop( )	8	[7, 9, 6]



Stack S

Implementation of stack

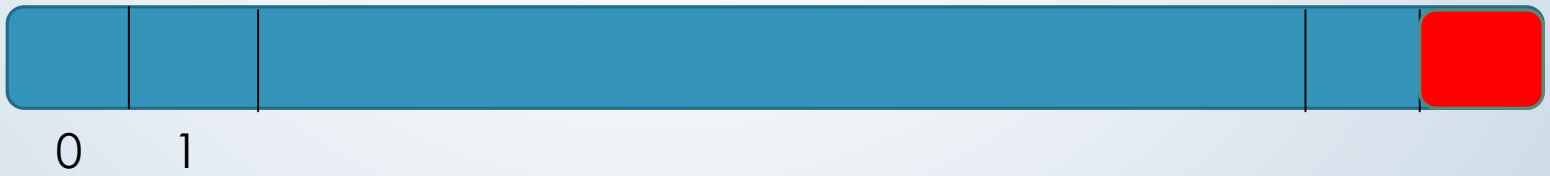
- Array-based stack implementation
  - (List in Python)
  - (Vector in C++)
- Dynamic array
  (Linked list)

there is a problem when the amount of data to be stored in the stack is not known in advance or cannot be accurately predicted.

Implement Stack in C++

Size() and empty() are already included in Vector

```
Template <typename T> class Stack: public Vector<T>{
    public:
        void push(T const & e) { insert ( e );}
        T pop() { return remove( size() -1); }
        T top()  { return (*this)[size() -1];}
};
```



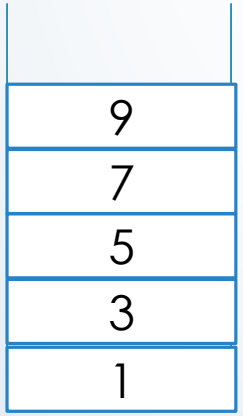
Question: Can we use T[0] as the top?

Analyzing the Array-Based Stack Implementation

Operation	Running Time
push(x)	O(1)
pop()	O(1)
top()	O(1)
size()	O(1)
empty()	O(1)

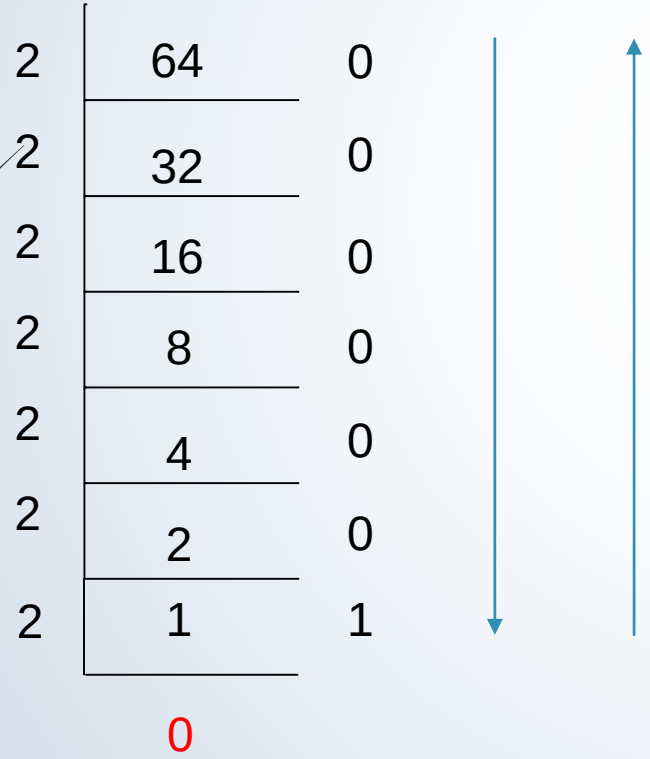
Application 1: reversing data

- input: 1,3,5,7,9
- output: 9,7,5,3,1



Application 2: Number system

- Decimal to binary equivalent



64 (10) =1000000 (2)

Try:

98 (10) = _____ (2)

2022 (10) = _____ (5)

AF (16) = _____ (2)

Application 3: Parenthesis checker

- Three sets of grouping symbols:
  - standard parentheses ( )
  - braces { }
  - brackets [ ]
- For each line of input, it verifies that for each left parenthesis, brace, or bracket, there is a corresponding closing symbol and that the symbols are appropriately nested.

Example

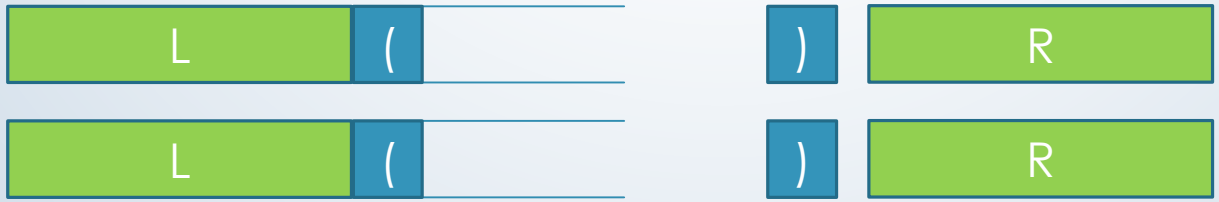
- (a [i-1][j+1]) + a [i+1][j-1])*2
- (a [i-1][j+1] + a [i+1][j-1])*2

- Linear scan
- ([ ][ ]) [ ][ ])
- ([ ][ ] [ ][ ])

How to do next?

- Trivial case: no parenthesis
- E is matched, only if (E) is matched.
- E and F are both matched, only if EF is matched.

Try to cancel a pair of adjacent parenthesis
L( )R is matched, only if LR is matched.



Input examples

- Valid inputs
  - ( )
  - { } [ ]
  - ({ [ ] [ ] })
  - [{ ({ } [ ]) { } [ ] }]
- Invalid inputs
  -) No matching open symbol
  - [Missing closing symbol
  - {] Incorrect nesting of symbols

The Parenthesis-checker Algorithm


```
For each character, c, in the input line:
    if c is a left symbol, push it on the stack
    if c is a right symbol, then
        if the stack is empty, then
            error: "No matching open symbol"
        else
            pop a symbol, s off the stack
            if s doesn't correspond to c, then
                error: "Incorrect nesting of symbols"
If the stack is not empty, then
 error: "Missing closing symbol(s)"
 clear the stack
```

Running example

- Given the input: [ ] ({ })

Buffer pointer	Stack (top is the leftmost item)
[	[^
]	^
(	(^
{	{ (^
}	(^
)	^

Question: if there are only (and), can we simplify the program?

Application 4: Evaluating Expression

In cmd, use 'set /a' to evaluate an expression.
In Excel, use '=' to evaluate an expression

- Expression: p+x/y-a*b+c
- Operands: p, x, y, a, b, c
- Operators: + - * /

Operator	priority
unary -, unary +	1
*, /	2
+, -	3
<, ≤, ≥, >, =, ≠	4
And	5
Or	6

Step 1: convert infix to postfix expression
Step 2: evaluate postfix expression

- Postfix: an operator followed after its operands
- Infix: an operator comes between two operands
- Prefix: an operator comes before its operands

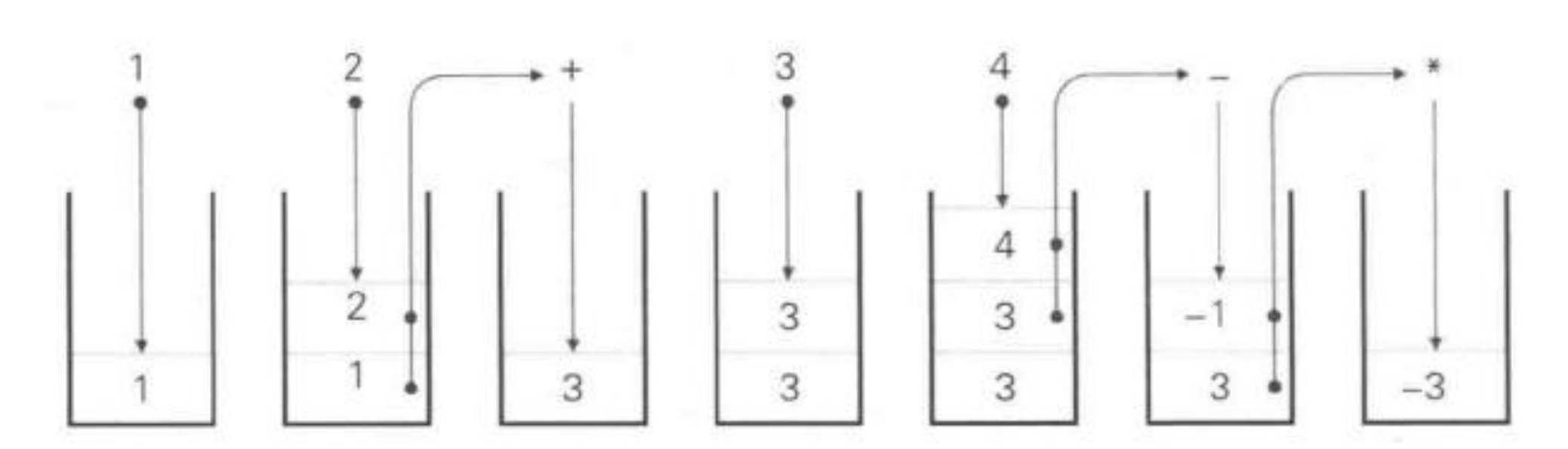
Convert infix to postfix

INFIX: P*Q/R
POSTFIX: PQ*R/

INFIX: **P*Q/R**
Let T=P*Q
POSTFIX PQ*
INFIX T/R
POSTFIX TR/
Substituting value of 'T' in postfix form
POSTFIX: PQ*R/

Postfix Expression Evaluation

- Postfix expression: 1 2 + 3 4 - +



For more examples about postfix expression evaluation, you can read
http://www.btechsmartclass.com/data_structures/postfix-evaluation.html

Other Applications

- Menu: each level
- Maze: find a way from entrance to exit
- Undo function
- Operating system simulation: call a function

Queue

First In First Out

Phenomena on the computer

- In Game, when factory produces units
- See online movies
- The way printer works

Queue

First in, first out (FIFO)

- The queue is the dual of the stack.
- With a queue, data items are placed at the rear and removed from the front.
- The items are removed from a queue in the same order in which they were added.

ADT of Queue

The essence of ADT is that the user can access the queue through these operations without knowing how the queue is actually implemented

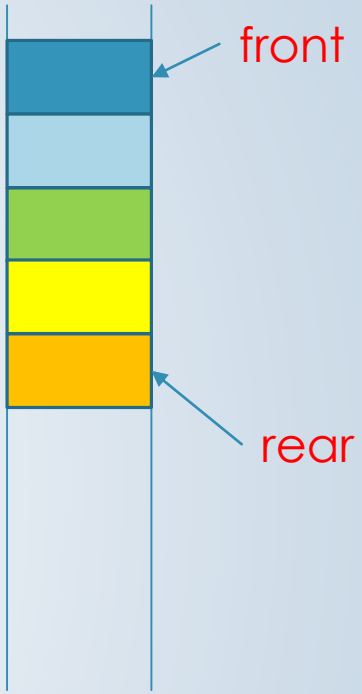
- Value
  - A sequence of items that belong to some data type
- Rule: FIFO
- Fundamental methods for a queue Q:
  - Q.enqueue(e): Add element e to the back of queue Q.
  - Q.dequeue( ): Remove and return the first element from queue Q; an error occurs if the queue is empty.

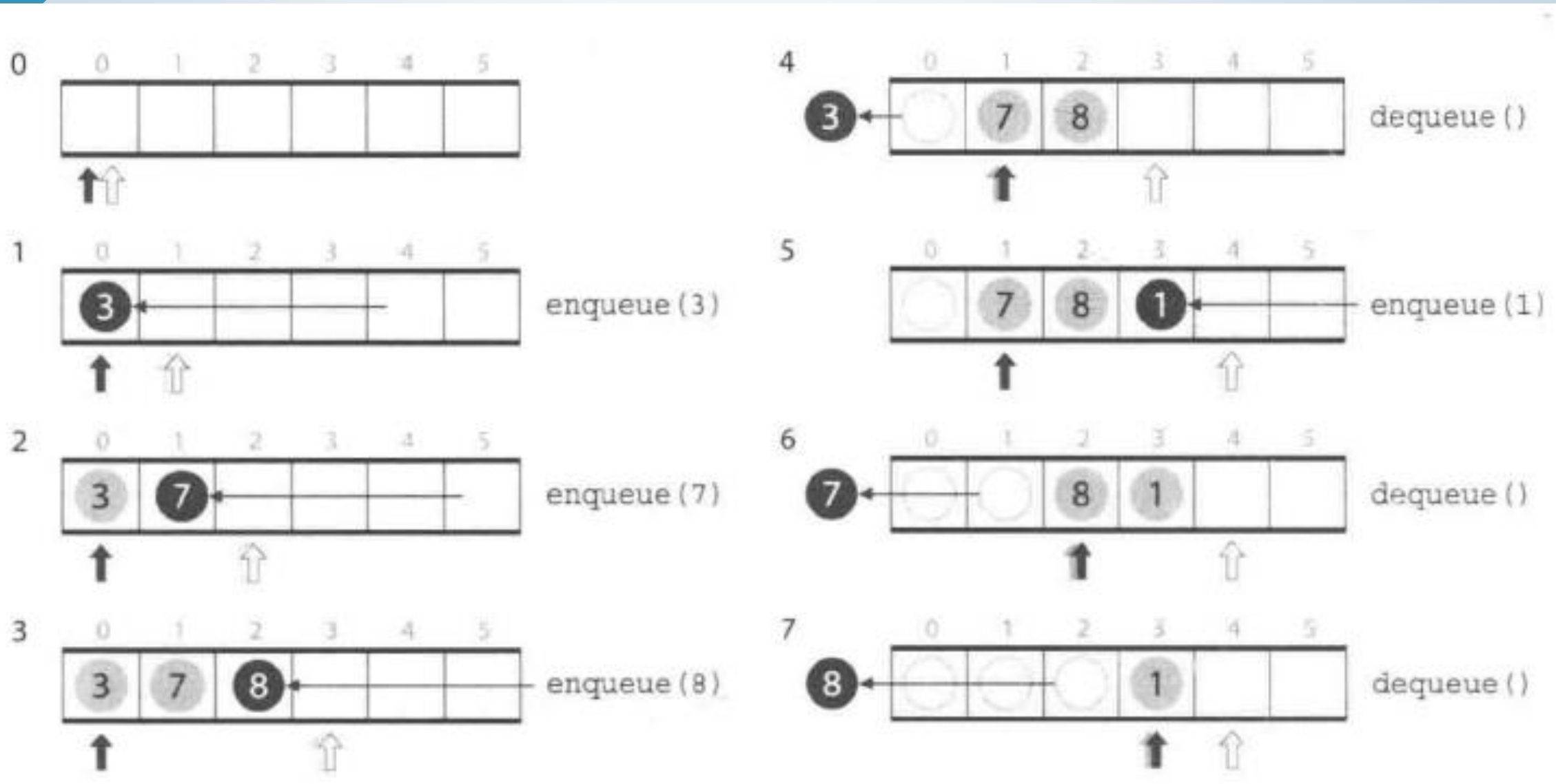
ADT of Queue

Operation	Function
size()	report the size of the queue
empty()	decide whether the queue is empty
enqueue(e)	insert an element e to the end of the queue
dequeue()	remove an element from the front of the queue
front()	get the element at the front

front rear

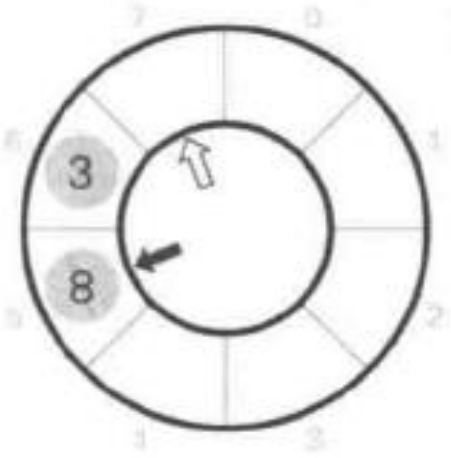
Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue( )	5	[3]
Q.is_empty( )	False	[3]
Q.dequeue( )	3	[ ]
Q.is_empty( )	True	[ ]
Q.dequeue( )	"error"	[ ]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first( )	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue( )	7	[9, 4]

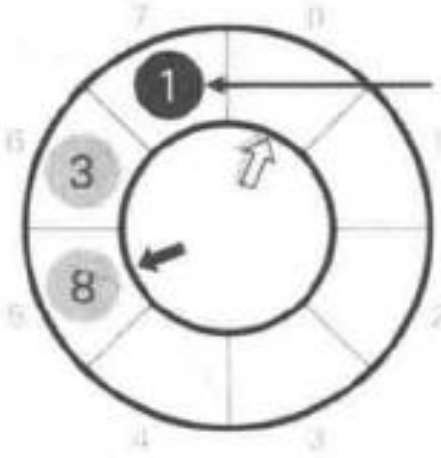



0
0 1 2 3 4 5
1
3
enqueue (3)
2
3 7
enqueue (7)
3
3 7 8
enqueue (8)
4
3
7 8
dequeue ()
5
7 8 1
enqueue (1)
6
7
8 1
dequeue ()
7
8
1
dequeue ()

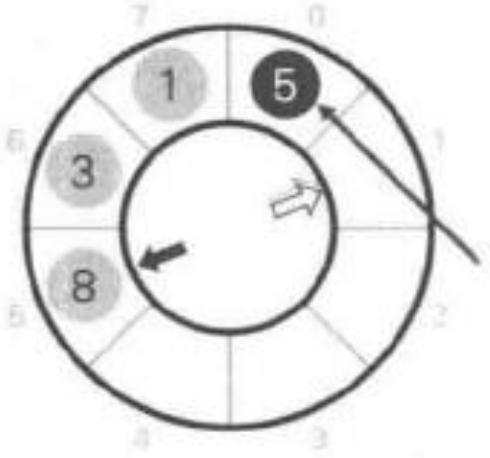
Using an Array Circularly

- When we dequeue an element and want to "advance" the front index, we use the arithmetic = (front + 1) % N.
- Recall that the % operator in C++/Python denotes the ***modulo*** operator, which is computed by taking the remainder after an integral division.

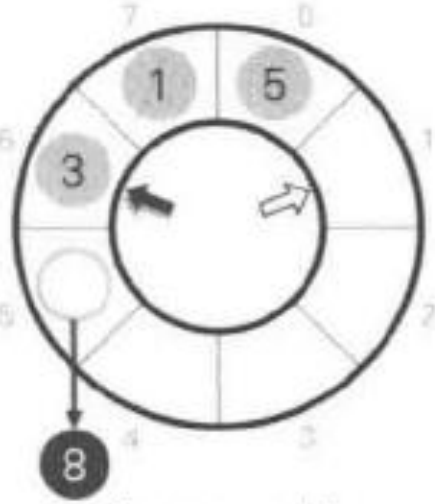




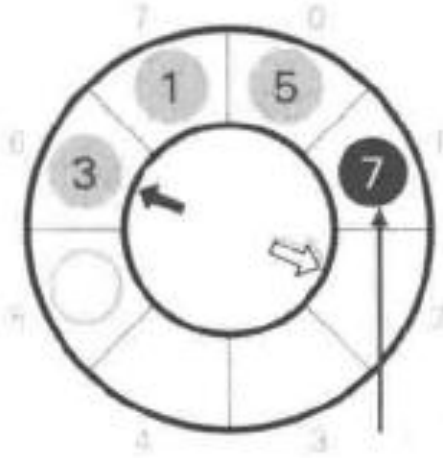
enqueue (1)



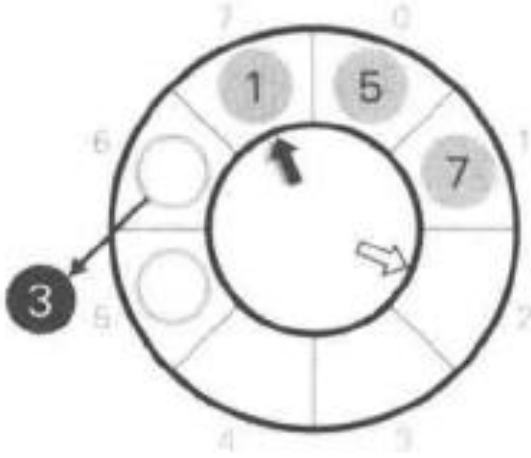
enqueue (5)



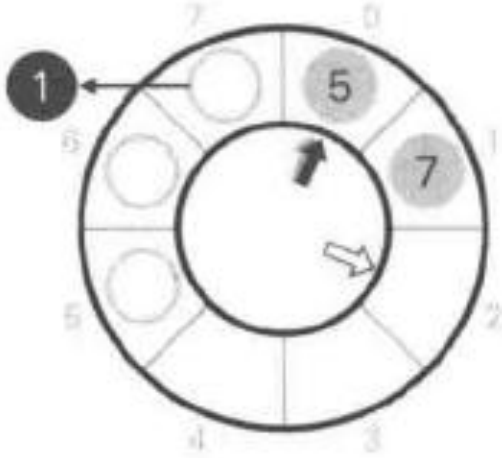
dequeue ()



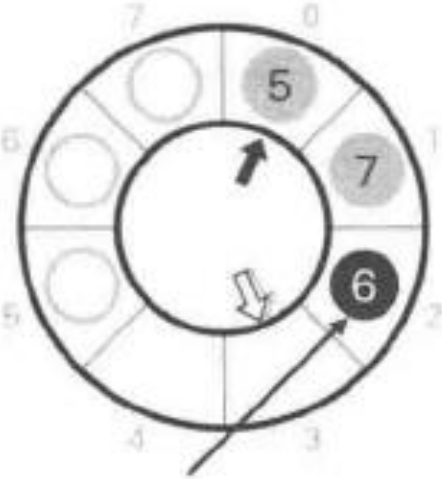
enqueue (7)



dequeue ()



dequeue ()



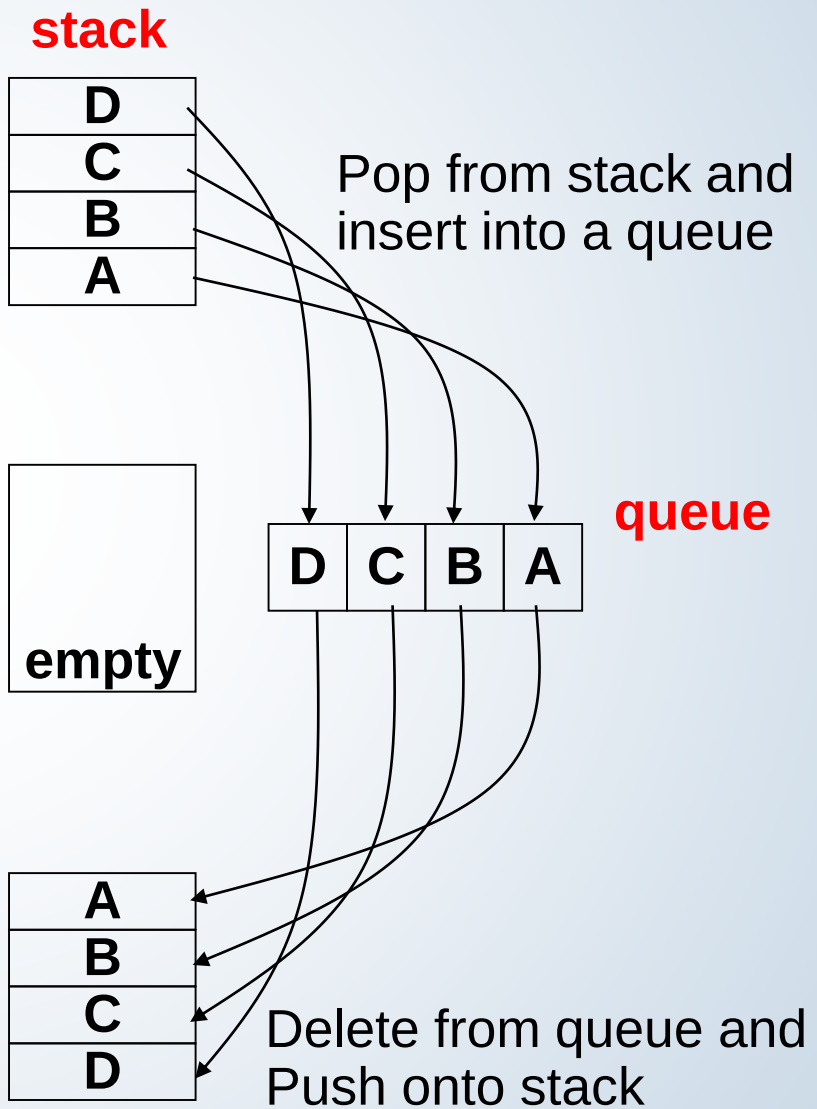
enqueue (6)

Analyzing the Array-Based Queue Implementation

Operation	Running Time
enqueue(x)	O(1)
dequeue( )	O(1)
first( )	O(1)
size( )	O(1)
front( )	O(1)

Application 1: Reversing a stack


```
Stack *s;
Queue *p;
…
while(s.IsEmpty()==0)
{   x = s.pop();
    p.Enqueue(x);
}
while(p.IsEmpty()==0)
{   x =p.Dequeue();
    s.Push(x);
}
```



Application 2

- Phenomena on the computer
  - See online movies
  - The way printer works
  - Round Robin Schedule
    - Establish a queue for current jobs
    - Whenever a time slot is used up
      - Insert the current job into the queue
      - Begin executing the job fetched from the queue

Round Robin Schedule

- job 1 : 4 time slots; job 2 : 3 time slots
- job 3 : 1 time slot; job 4 : 2 time slots

1(4 left)	2(3 left)	3(1 left)	4(2 left)				
	2(3 left)	3(1 left)	4(2 left)	1(3 left)			
		3(1 left)	4(2 left)	1(3 left)	2(2 left)		
			4(2 left)	1(3 left)	2(2 left)		
				1(3 left)	2(2 left)	4(1 left)	
					2(2 left)	4(1 left)	1(2 left)
2(1left)						4(1 left)	1(2 left)
2(1left)							1(2 left)
2(1left)	1(1 left)						
	1(1 left)						

Queue in STL

- #include<queue>

queue<Type> q;

q.push(x); //Add data to the end of the queue.

q.pop(); //Removes first element. Note that **no data** is returned, and if the first element's data is needed, it should be retrieved before **pop()** is called.

q.size(); //Returns the number of elements in the queue.

q.front(); //Returns a read/write reference to the data at the first element of the queue.

q.back(); //Returns a read/write reference to the data at the last element of the queue.

q.empty(); //Returns true if the queue is empty.

Deque

Double ended queue

ADT of Deque in STL

- #include<deque>

deque<type>q;

```
q.front();          // return the element at the front
q.back();           // return the element at the back
q.pop_back();       // delete the element at the back
q.pop_front();      // delete the element at the front
q.push_back(x);     // insert element x at the back of the queue
q.push_front(x);    // insert element x at the front of the queue
```