Applications of Stacks and Queues

CIS*2520, Fall 2011, Lab 4

School of Computer Science (SOCS)
University of Guelph

Mohammad Naeem (TA)
mnaeem@uoguelph.ca

Topics

- general
- applications
  - balancing parentheses (with stacks)
  - prefix to postfix conversion (with stacks)
  - palindrome testing (with stacks)
  - palindrome testing (with stacks and queues)

Stacks and Queues: Characteristics

access restricted (unlike array),
i.e., via special operations:

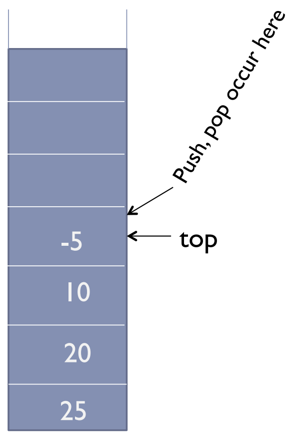
- **push** and **pop** for STACK
- **enqueue** and **dequeue** for QUEUE
- other operations as well

Stacks

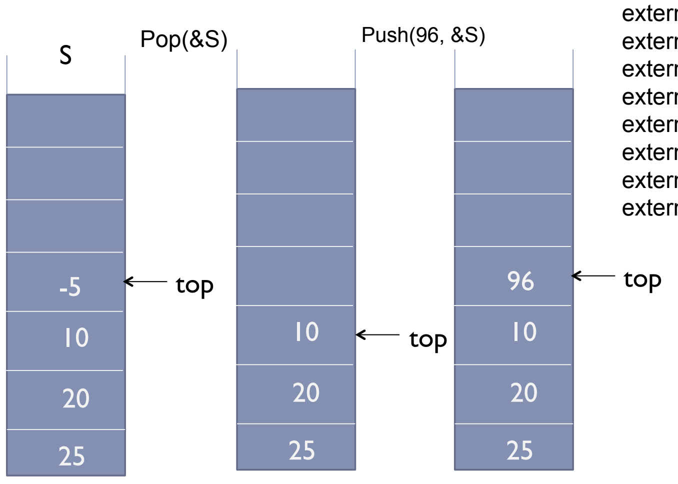
stack S of items of type T is a sequence
of items of type T

operations allowed:

- ***initialize*** S
- determine whether S ***full***
- determine whether S ***empty***
- ***push*** a new item onto the top of the stack
- ***pop*** a new item from the top
- other operations



Stacks



```
extern void Initialize (Stack *S);
extern void Push (Item I, Stack *S);
extern void Pop (Stack *S);
extern int Full (Stack *S);
extern int Empty (Stack *S);
extern int Size (Stack *S);
extern void Top (Stack *S, Item *I);
extern void Destroy (Stack *S);
```

```
void main() {

        Stack S;
        int val;
          …
          …
        Top(&S,&val);
        Pop(&S);
        Push (96, &S);

} //main()
```

Applications of Stacks

A lot!

- checking balanced expressions
- recognizing palindromes
- evaluating algebraic expressions
- call stack (recursion)
- searching networks
- traversing trees (keeping a track where we are)
- parsing (in compilers)
- computational linguistics
- etc.

notation for expressions

Infix

- ❖ the usual notation we use
- ❖ e.g.,

A * (B + C) / D

postfix

- ❖ operators written after operands
- ❖ order of evaluation left-to-right
- ❖ operators apply to immediately preceding ops
- ❖ e.g.,

A B C + * D /

prefix

- ❖ operators before ops
- ❖ e.g.,

/ * A + B C D

- ❖ postfix: no need for parenthesis to specify operator precedence
- ❖ so expressions represented in postfix in computation

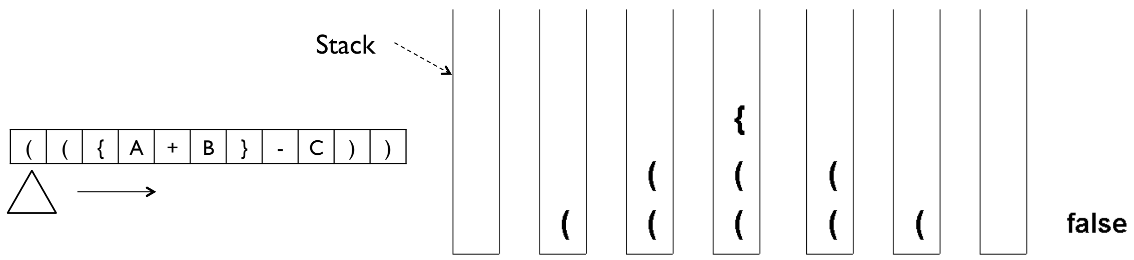
Infix to Postfix Conversion

```
For each token in the input expression do
    If token = operand, append operand to P
    If token = operator, push(token)
    If token = ")", append pop() to P
    If token = "(", ignore
```

INFIX: 3 * (5 + 7) – 9

POSTFIX: 3 5 7 + * 9 -

Stack S, (sideways)	**String P**
empty	3
*	**3**
*	**3 5**
*** +**	**3 5**
*** +**	**3 5 7**
*	**3 5 7 +**
empty	**3 5 7 + ***
-	**3 5 7 + ***
-	**3 5 7 + * 9**
empty	**3 5 7 + * 9 -**



Scan left to right.

Push each left parenthesis on the stack.

For each right parenthesis,

If the stack is empty, return false (too many right parentheses)

Otherwise, pop off the top parenthesis from the stack:

If the left and right parentheses are of the same type, discard.
Otherwise, return false. (not balanced)

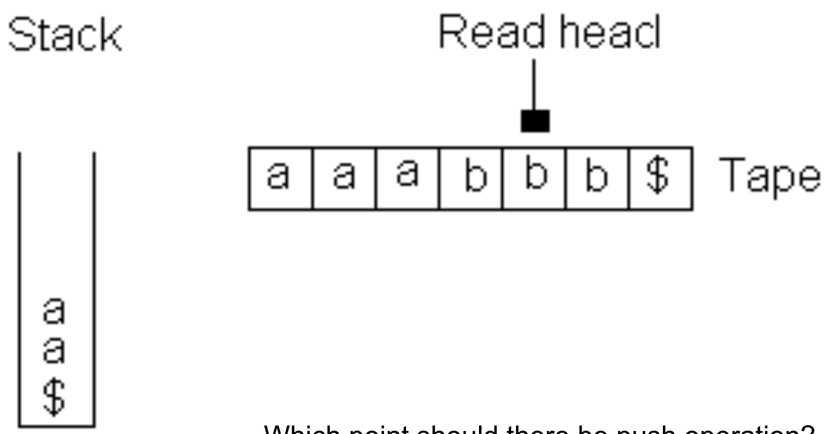
If the stack is empty when the scan is complete, return true.
Otherwise, false. (too many left parentheses)

```
extern void Initialize (Stack *S);
extern void Push (Item I, Stack *S);
extern void Pop (Stack *S);
extern int Full (Stack *S);
extern int Empty (Stack *S);
extern int Size (Stack *S);
extern void Top (Stack *S, Item *I);
extern void Destroy (Stack *S);
```

(({A + B} – C) / D))

```
main () {

  Stack S;
  char exp[] =  "(({A+B}-C) / D))";

  char ch, *exptr = exp;
  ...
   while(*exptr != '\0') {
       if (*exptr == '(' || *exptr == '{')
          Push(*exptr++, &S);
      else
          if (*exptr == ')' || *exptr == '}' ) {
             Top(&S,&ch);
             Pop(&S);
              if (*exptr==')' && ch!='(')
                  printf("parenthesis mismatch\n");
              …
   } //while
   …

 } //main()
```

Pushdown Automata



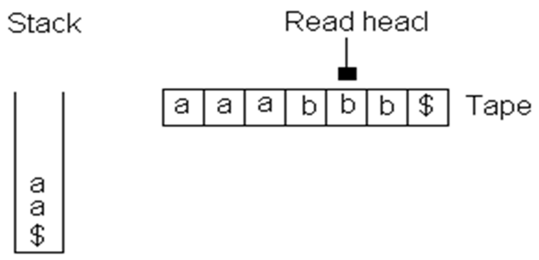
Which point should there be push operation?

Recognizing Balanced Strings and Palindromes

- ❖ Keep pushing “a” on stack until first “b” read from the tape

  then

  - ❖ consecutively read “b” from the tape and match with “a”s popped off the stack



When palindrome: matching “a”s for all “b”s
entire input tape read
stack finally empty

When not palindrome?

Caution: only “a”s pushed on the stack

Implementing Palindrome Recognizer

❖ assume: built-in stack with interface.

```
extern void Initialize (Stack *S);
extern void Push (Item I, Stack *S);
extern void Pop (Stack *S);
extern int Full (Stack *S);
extern int Empty (Stack *S);
extern int Size (Stack *S);
extern void Top (Stack *S, Item *I);
extern void Destroy (Stack *S);
```

❖ function, palindRecog()
  - ❖ called from main()
  - ❖ accepts a string, returns
    - ❖ 1 if palindrome
    - ❖ 0 otherwise

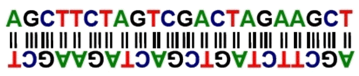
```
deftype ch Item;

void main() {

        Item item[]="aaabbb$";
        palindRecog(item);

} //main ()

int palindRecog (Item *item) {

        Stack S;
         …
         …
         …

} //palindRecog()
```

Palindromes: Interesting Patterns

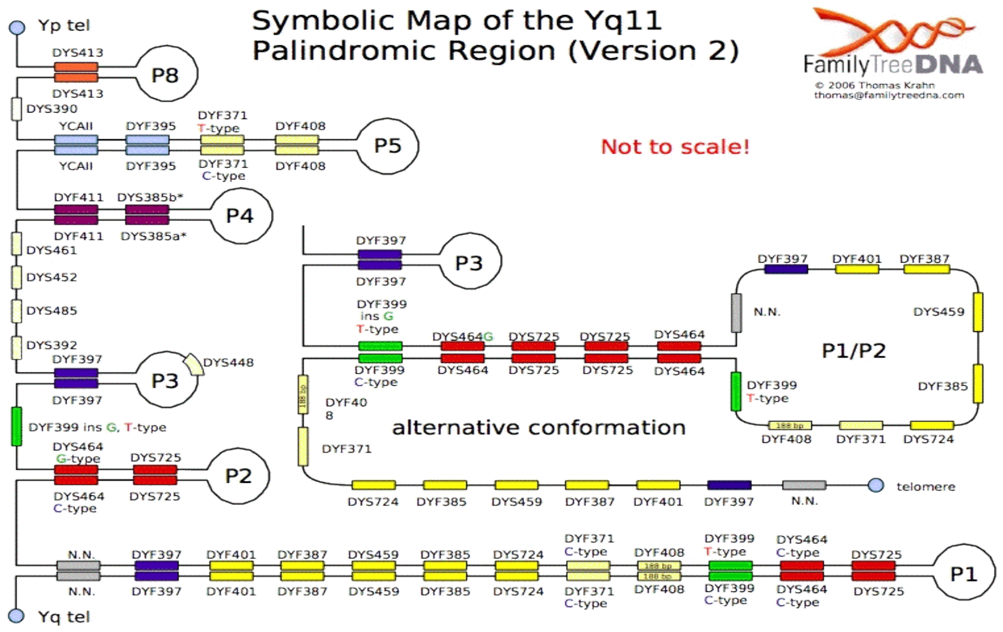
SATOR
AREPO
TENET
OPERA
ROTAS

NAME NO ONE MAN
413245542314

Forward = Reverse Complement (Forward)



From Thomas Krahn's presentation.



Queues

British word for a line (or line-up)

- kind of **front-end data structure**…
- **insertions** at the end
- **deletions** from front
- **FIFO** (First-In-First-Out) data structures

Queue Operations

- Q: a sequence of items of type T
- operations on Q
  - **enqueue** – insert an item at the back of the queue
  - **dequeue** – delete an item from the front
  - **peek** – returns the item at the front of the queue
  - **initialize -** the queue
  - determine whether the queue is **full**
  - determine whether the queue is **empty**
  - ......

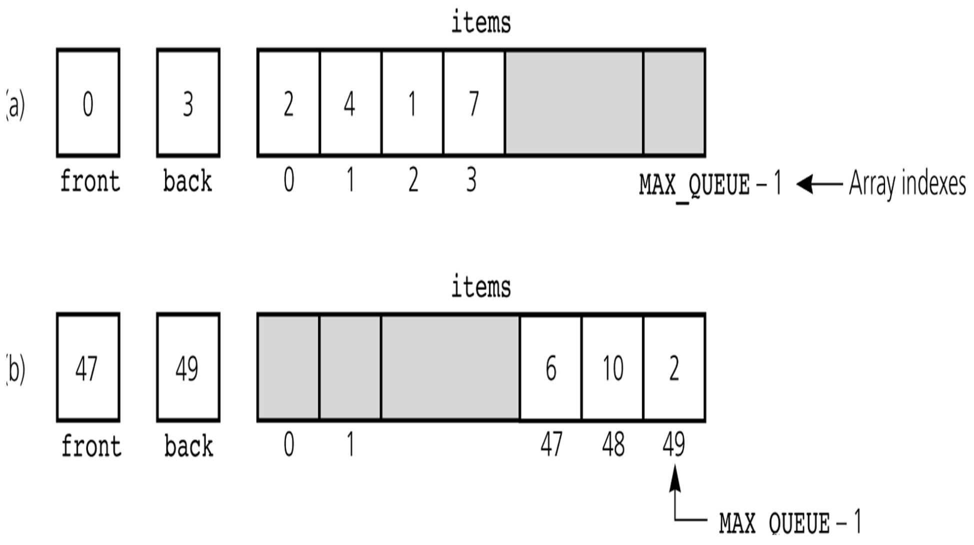
Queue Operations: Example

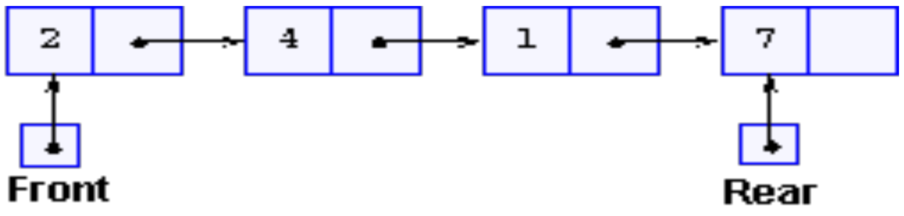
Operation		***Output*** *Q*
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	*5*	(3)
enqueue(7)	–	(3, 7)
dequeue()	*3*	(7)
front()	*7*	(7)
dequeue()	*7*	()
dequeue()	*“error”*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	*9*	(7, 3, 5)

Queue operations:

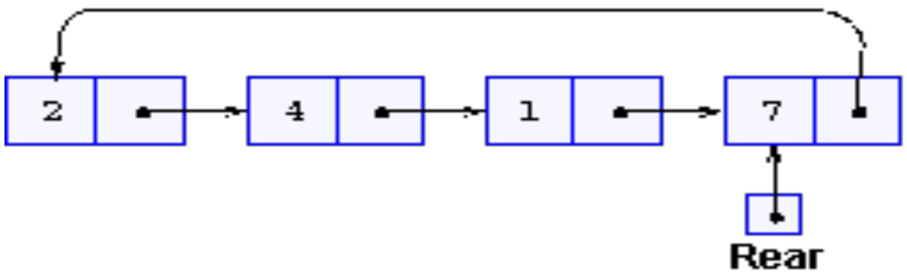
```
extern void Initialize (Queue *Q);
extern void Enqueue (Item I, Queue *Q);
extern void Dequeue (Queue *Q);
extern int Full (Queue *Q);
extern int Empty (Queue *Q);
extern int Size (Queue *Q);
extern void Head (Queue *Q, Item *I);
extern void Tail (Queue *Q, Item *I);
extern void Destroy (Queue *Q);
```

An Array-Based Implementation





Linked List



Applications of Queues

- inputs and outputs to screen
- messaging server: instant messages queue
- DBMS: database requests queue
- print queue: one printer for many computers
- Job scheduler (OS): job queue for CPU..
- simulations
- etc.

Palindrome with Queue & Stack

NAME NO ONE MAN

4 1 3 2 4 5 5 4 2 3 1 4

	Stack S: top ⟶	⟵ front **queue (Q)** rear ⟶	
push('N', S)	N	N	enque('N', Q)
push('A', S)	NA	NA	enque('A', Q)
	NAM	NAM	
	NAME	NAME	
	NAMEN	NAMEN	
	NA MENO	NA MENO	
	NA MENO O	NA MENO O	
	NA MENO ON	NA MENO ON	
	NA MENO ONE	NA MENO ONE	
	NA MENO ONEM	NA MENO ONEM	
	NA MENO ONEMA	NA MENO ONEMA	
push('N', S)	NA MENO ONEMAN ⟵ top	front ⟶ NA MENO ONEMAN	enque('N', Q)

↖ rear

palindrome with queue & stack

N	N
NA	NA
NAM	NAM
NAME	NAME
NAMEN	NAMEN
NA MENO	NA MENO
NA MENO O	NA MENO O
NA MENO ON	NA MENO ON
NA MENO ONE	NA MENO ONE
NA MENO ONEM	NA MENO ONEM
NA MENO ONEMA	NA MENO ONEMA
NA MENO ONEMAN ← top	front → NA MENO ONEMAN ← rear

pop deque

Keep on popping from
S and dequeuing from **Q** matching symbols along

Queue operations:

```
extern void Initialize (Queue *Q);
extern void Enqueue (Item I, Queue *Q);
extern void Dequeue (Queue *Q);
extern int Full (Queue *Q);
extern int Empty (Queue *Q);
extern int Size (Queue *Q);
extern void Head (Queue *Q, Item *I);
extern void Tail (Queue *Q, Item *I);
extern void Destroy (Queue *Q);
```

Stack operations:

```
extern void Initialize (Stack *S);
extern void Push (Item I, Stack *S);
extern void Pop (Stack *S);
extern int Full (Stack *S);
extern int Empty (Stack *S);
extern int Size (Stack *S);
extern void Top (Stack *S, Item *I);
extern void Destroy (Stack *S);
```

```
void main(){
 ….
 Stack stack;
 Queue queue;
 Item item[]="NAME NO ONE MAN";
 i=palind_Recog(item, stack, queue);
 ….

} //main()
```

```
int palind_Recog(Item *it, Stack *S, Queue *Q ){

   ….
   ….
   ….
  return flag;
} //palind_Recog()
```