

Stack

Read: Chapter 6 of the text book.

Stack is a data structure of items such that items can be inserted and removed only at one end. (Last-in, first-out).

1 Applications using stack

- To reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.
- An “undo” mechanism in text editors; this operation is accomplished by keeping all text changes in a stack: Undo/Redo stacks in Excel or Word.
- Language processing :
 - space for parameters and local variables is created internally using a stack.
 - compiler’s syntax check for matching braces is implemented by using stack.
- A garage that is only one car wide. To remove the first car in we have to take out all the other cars in after it. Wearing/Removing Bangles.
- Back/Forward stacks on browsers.
- Support for recursion
- Activation records of method calls.

2 Stack operations

- push(e): Insert an element e at the top of the stack
- pop(): Remove the top element from the stack; and error occurs if the stack is empty
- top()/peek(): Return a reference to the top element on the stack, without removing it; an error occurs if the stack is empty
- size(): Return the number of elements in the stack
- isEmpty(): Return true if the stack is empty and false otherwise

3 Implementing a stack

- MyStack interface

```
public interface MyStack<E>{
    /* Get the reference to the top element and remove it from the stack
     * Report error if the stack is empty
     * @return top element */
    public E pop();

    /* Insert an object to the stack */
    public void push(E e);

    /* Return a reference to the top element without removing it
     * Report error if the stack is empty
     * @return top element */
    public E top();

    /* @return the number of elements in the stack */
    public int size();

    /* @return true if the stack is empty and false otherwise */
    public boolean isEmpty();
}
```

- Implementation: ArrayStack<E>
- Implementation: LinkStack<E>

4 Using a stack

4.1 Basic usage

```
push(5);
push(3);
pop();
push(7);
pop();
peek();
pop();
pop();
peek();
isEmpty();
push(9);
push(7);
size();
```

```

Data:  $N$ 
1 Initialize an Integer stack  $S$  where we can keep track of our decisions with queens' column ids  $colid$ .
2 Generate the first queen's location:  $Qpos$  with  $colid = 1$  (implicitly it is for row 1)
3 while  $S.size() < N$  do
4   /* Stack is not full; there are  $S.size()$  queens */
5   /* 1.1 Fill in one row */
6   while  $Qpos \leq N$ , i.e., there is room to shift the current queen rightward do
7     boolean conflict = checkConflict( $Qpos, S$ ); /*you need to design this function*/
8     if there are no conflict with the queens then
9       | Push  $Qpos$  to stack  $S$ 
10      | Move to the next row  $Qpos$  with  $colid = 1$  and  $rowid = S.size() + 1$  (implicit)
11     end
12     else
13       | /*there is a conflict*/
14       | Move the current queen rightward
15       | i.e., set  $Qpos$ 's  $colid$  to be  $colid + 1$  ( $rowid$  does not change)
16     end
17   end
18   /*1.2 Tested every column in the row ( $S.size()$ ), no one is working, need to backtrack*/
19   while ( $S$  is not empty and  $col > N$ ) do
20     | //Backtrack!
21     | Keep popping the stack, until you reach a row where the queen can be shifted rightward (loop).
22     | shift this queen right.
23   end
24   //Special case process, no solution
25   if  $Qpos.colid > N$  then
26     | break;
27   end
28 end
29 printStack( $S$ ) /* You need to design this function */

```

Figure 1: Pseudocode for N-Queens

4.2 Application 1: N-Queens problem

Suppose you have 8 chess queens and a chess 8×8 board.

Can the queens be placed on the board so that no two queens are attacking each other?

Two queens are NOT allowed in the **same row, or in the same column, or along the same diagonal**.

Write a program which tries to find a way to place N queens on an $N \times N$ chess board.

4.3 Application 2: Evaluating arithmetic expression

- Infix notation: $2+3$, $3*4$, $5-7$; $6/2$
- Polish prefix notation (prefix notation): $+ 2 3$, $* 3 4$.
 - This is devised by the Polish mathematician Jan Łukasiewicz
 - $* + 2 3 4$
- Postfix notation
 - $2 3 + 4 *$
 - Often used internally for computers because of the ease of evaluation.
- numerator denominator /
- minuend subtrahend –

```
1 Initialize a stack of characters to hold the operation symbols and parentheses;  
2 while there is more of the expression to read do  
3   if (the next input is a left parenthesis) then  
4     | Read the next left parenthesis and push it onto the stack.  
5   else if (the next input is a number or other operand) then  
6     | Read the operand and write it to the output.  
7   else if (the next input is one of the operation symbols) then  
8     | Pop and print operations off the stack until one of three things occurs:  
9     | (1) The stack becomes empty;  
10    | (2) The next symbol on the stack is a left parenthesis;  
11    | or (3) The next symbol on the stack is an operation with lower precedence than the next input  
12    | symbol  
13    | Stop stopping  
14    | Read the next input symbol,  
15    | Push this symbol onto the stack  
16   else  
17     | Read and discard the next input symbol (which should be a “)”)  
18     | Pop and print operations off the stack until the next symbol on the stack is a left parenthesis.  
19     | (If no “(” is encountered, print an error message indicating unbalanced parenthesis.)  
20     | Pop and discard the left parenthesis  
20 Pop and print any remaining operations on the stack. //
```

Figure 2: Converting an Infix Expression to a Postfix Expression (General case)

Example:

$$3 * X + (Y - 12) - Z$$

Output:

$$3X * Y12 - +Z-$$

```

1 Initialize a stack of double numbers;
2 while there is more input in the expression do
3   if the next input is a number then
4     | Read the next input and push it onto the stack
5   else
6     | Read the next input, which is an operation symbol
7     | Pop two numbers off the stack
8     | Combine the two numbers with the operation (using the second number as the left operand)
9     | Push the result onto the stack
10 At this point, the stack contains one number, which is the value of the expression.

```

Figure 3: Using a stack to evaluate postfix expressions

Details of this algorithm see page 347 (Main book 4th Ed.).

Example: $5\ 3\ 2\ * \ +\ 4\ -\ 5\ +$

5 Summary

- Stacks can be implemented using different data structures (e.g., Array, Linked list)
- Stacks have many applications.
- The application which we have shown is called **backtracking**.
 - The key to backtracking: Each choice is recorded in a stack.
 - When you run out of choices for the current decision, you pop the stack, and continue trying different choices for the previous decision.