Lecture 6: Stacks

Bohyung Han
CSE, POSTECH
bhhan@postech.ac.kr

Stacks

- What is stack?
  - An abstract data type storing arbitrary objects.
  - An ordered list in which all insertions and deletions are made at one end, called top

- Properties
  - Last-In-First-Out (LIFO) scheme
  - Typically implemented with array or linked list

Stack class in Java
- http://docs.oracle.com/javase/7/docs/api/java/util/Stack.html

Stack Operations

- Main operations:
  - push(Object): inserts an element at the top
  - Object pop(): removes an element at the top and returns it

- Others:
  - Object top(): returns the element at the top without removing it
  - int size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored

Stack Interface in Java

```java
public interface Stack {
    public int size();
    public boolean isEmpty();
    public Object top() throws EmptyStackException();
    public void push(Object element) throws FullStackException();
    public Object pop() throws EmptyStackException();
}
```

- Restrictions
  - The execution of pop() and top() on an empty stack throws an EmptyStackException().
  - The execution of push() on a full stack throws a FullStackException().

- Java: interface
  - A group of related methods with empty bodies
  - To implement this interface, the name of your class would change and keyword implements is used in the class declaration.
Stack Implementation in Java

```java
public class BoundedStack implements Stack
{
    private Object[] array;
    private int size = 0;

    public BoundedStack(int capacity)
    {
        array = new Object[capacity];
    }

    public boolean isEmpty()
    {
        return (size == 0);
    }

    public int size()
    {
        return size;
    }

    public void push(Object item)
    {
        if (size == array.length)
            throw new FullStackException();
        array[size++] = item;
    }

    public Object pop()
    {
        if (size == 0)
            throw new EmptyStackException();
        Object result = array[size-1];
        array[--size] = null;
        return result;
    }
}
```

Common Applications of Stack

- Page-visited history in a web browser
- Undo sequence in a text editor
- Parentheses matching
- Evaluating arithmetic expressions

Method Stack in the JVM

- The Java Virtual Machine (JVM) keeps track of the chain of active methods with a stack.
- When a method is called, the JVM pushes on the stack a frame containing
  ▪ Local variables and return value
  ▪ Program counter, keeping track of the statement being executed
- When a method ends, its frame is popped from the stack and control is passed to the method on top of the stack.
An Example of Method Stack

```
public void main(String args[]) {
    int i = 5;
    foo(i);
}
void foo(int j) {
    int k;
    k = j+1;
    bar(k);
}
void bar(int m) {
    ...
}
```

- **Main Method**: `main(String args[])` is called, which invokes `foo(i)`, where `i` is set to 5.
- **Foo Method**: `foo(int j)` is called with `j` set to 5, which initiates the execution.
- **Bar Method**: `bar(int m)` is called with `m` set to 6, which continues the execution.

### Array-based Stack

- A simple way of implementing the Stack ADT uses an array.
- Top points to the last element in stack.
  - We add elements from left to right.
  - A variable keeps track of the index of the top element.

#### Algorithm `size()`

```java
return t + 1
```

#### Algorithm `pop()`

```java
if isEmpty() then
    throw EmptyStackException()
else
    t ← t − 1
    return S[t + 1]
```

- **Performance**
  - `n`: the number of elements in the stack
  - Space complexity: $O(n)$
  - Time complexity of all operations: $O(1)$

- **Limitations**
  - The maximum size of the stack must be defined a priori and cannot be changed.
  - Trying to push a new element into a full stack causes an exception.
  - Use growable array instead
Application of Stack

- Parentheses Matching
  - Each “(”, “{”, and “[” must be paired with “)”, “}”, and “]”, respectively.
    - correct: ( )( )(())
    - correct: (())(()(()))
    - incorrect: )(( )){([ ])}
    - incorrect: ( )
  - ParenMatch
    - Input: (2+3)*(3+6)
    - X: an array of tokens, e.g., {2,+,3,*,3,+}
    - n = 11
    - Output: true

ParenMatch Algorithm

Algorithm ParenMatch(X,n):
Input: An array X of n tokens
Output: true if and only if all the grouping symbols in X match

Let S be an empty stack
for i=0 to n-1 do
  if X[i] is an opening grouping symbol then
    // e.g., (  
    S.push(X[i])
  else if X[i] is a closing grouping symbol then
    if S.isEmpty() then
      return false // nothing to match with
  return false // wrong match
  if S.isEmpty() then
    return true // Every symbol was matched.
  else
    return false // Some symbols were never matched.

Application of Stack

- Evaluating arithmetic expressions
  - Example: 14 – 3 * 2 + 7 = (14 – (3 * 2)) + 7
  - Operator precedence
    - * and / have higher precedence over + and –.
  - Associativity
    - Operators of the same precedence group
    - Evaluated from left to right
    - Example: x – y + z: (x – y) + z rather than x – (y + z)

Application of Stack

- Evaluating arithmetic expressions
  - Idea: push each operator on the stack, but first pop and perform higher and equal precedence operations.
  - Two stacks:
    - Operator stack (opStk) holds operators.
    - Value stack (vaStk) holds values.
  - To clean up the stack at the end, we use $ as special “end of input” token with lowest precedence.
Application of Stack

- Evaluating arithmetic expressions

**Algorithm EvalExp()**

**Input**: a stream of tokens representing an arithmetic expression (with numbers)

**Output**: the value of the expression

while there is another token z
  if isNumber(z) then
    valStk.push(z)  // push number
  else
    repeatOps(z);  // calculate the higher precedence operators
    opStk.push(z);  // before push the operator z
    repeatOps($);  // calculate all the rest operators
  return valStk.top()

Algorithm doOp()

x  valStk.pop();
y  valStk.pop();
op  opStk.pop();
valStk.push( y op x )

Algorithm repeatOps( refOp ):

while ( valStk.size() > 1 ∧ prec(refOp) ≤ prec(opStk.top()) )
doOp()

An Example

14 \leq 4 - 3 * 2 + 7

Operator \leq has lower precedence than +/-.