The Stack ADT

A Stack ADT is a list that allows only operations at one end of the list (called the top).

Main stack operations:
- push(object): inserts an element at the top of the stack
- object pop(): removes and returns the object at the top of the stack
- object top(): returns the last inserted element without removing it (N.B. In Java, this is called peek() )
- integer size(): returns the number of elements stored
- boolean isEmpty(): indicates whether no elements are stored

Applications of Stacks

Direct applications
- Page-visited history in a Web browser
- Undo sequence in a text editor
- Chain of method calls in the Java Virtual Machine

Indirect applications
- Auxiliary data structure for algorithms
- Component of other data structures

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.

Allows for recursion

main()
  int i = 5;
  foo(i);

foo(int j)
  int k;
  k = j+1;
  bar(k);

bar(int m)

 Array-based Stack in Java

public class ArrayStack {
  // holds the stack elements
  private Object[] S;

  // index to top element
  private int top = -1;

  // constructor
  public ArrayStack(int capacity) {
    S = new Object[capacity];
  }

  // returns size of the stack
  public int size() {
    return top + 1;
  }

  // removes and returns the element at the top
  public Object pop() {
    if (isEmpty())
      throw new EmptyStackException();
    else
      return S[top--];
  }

  // puts an element at the top of the stack
  public void push(Object element) {
    if (top == S.length - 1)
      throw new StackOverflowError;
    else
      S[++top] = element;
  }

  // returns true if the stack is empty
  public boolean isEmpty() {
    return top == -1;
  }

  // returns the top element of the stack without removing it
  public Object top() {
    if (isEmpty())
      throw new EmptyStackException();
    else
      return S[top];
  }
}

Array-based Stack

A simple way of implementing the Stack ADT uses an array.

We add elements from left to right.

A variable keeps track of the index of the top element.

Algorithm size() {
  return t;
}

Algorithm pop() {
  if (isEmpty())
    throw new EmptyStackException();
  else
    t = t - 1;
  return S[t+1];
}
Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException

Algorithm `push(o)`

```java
if t = S.length - 1 then throw FullStackException
else
t ← t + 1
S[t] ← o
```

Limitation of the array-based implementation
- Not intrinsic to the Stack ADT

S
0 1 2 ...

Performance and Limitations

- Performance
  - Let \( n \) be the number of elements in the stack
  - The space used is \( O(n) \)
  - Each operation runs in time \( O(1) \)

- 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 implementation-specific exception

Stack using a Singly Linked List

- We can implement a stack with a singly linked list
- The top element is stored at the first node of the list
- The space used is \( O(n) \) and each operation of the Stack ADT takes \( O(1) \) time

Parentheses Matching

- Each "(", "{", or "[" must be paired with a matching ")", "}", or "]"
  - correct: ( )(( ))(())
  - incorrect: ( ))(( ))(())
  - incorrect: )(( ))(())
  - incorrect: ( )
  - incorrect: ( )

Parentheses Matching Algorithm

Algorithm `ParenMatch(X,n)`

Input: An array \( X \) of \( n \) tokens, each of which is either a grouping symbol.

Output: true if and only if \( X \) is well-balanced.

- for \( i = 0 \) to \( n-1 \) do
  - ...

© 2004 Goodrich, Tamassia
Model

Well-bracketed words:
- Each open bracket is associated to one and only one close bracket.
- Open a bracket before to close it.
- A close bracket matches the closest (i.e. last seen) available open bracket.

A stack is used to:
- Store the open brackets not already matched.
- Last in, first out. We want to match the last open bracket that has not been match yet (i.e. the last inserted in the stack).

Example

Push("\(\)\)

Empty stack at the end of the scan.
All match were valid and all brackets are matched. OK!

Example

List non empty.
Brackets without match. Fail!
Parentheses Matching Algorithm

Algorithm ParenMatch(X, n):
Input: An array X of n tokens, each of which is either a grouping symbol.
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
    S.push(X[i])
  else if X[i] is a closing grouping symbol then
    if S.isEmpty() then
      return false \{nothing to match with\}
    if S.pop() does not match the type of X[i] then
      return false \{wrong type\}
    if S.isEmpty() then
      return true \{every symbol matched\}
  else
    return false \{some symbols were never matched\}

Example

(  )  )  )  )  )

Empty stack! Nothing to match.