Stacks

Tuesday, September 25, 2007
11:00 PM

- 7.1 Introduction
  - stacks are last-in, first-out (LIFO)
  - can only access top element
    - no traversal operations
  - Standard operations
    - construct empty stack
    - check if stack is empty
    - push - add item to stack
    - pop - remove item from stack
    - top - just get value at top, do not remove top
  - data storage
    - array
    - linked list
- 7.2 Array-Based Implementation
  - Design for static array
    - constructor - create empty stack
    - empty() - check if stack is empty
    - push() - add value to stack
    - pop() - remove value at top of stack
    - top() - retrieve valve at top of stack
    - display() - print debugging into
  - Data Storage
    - if top is always index 0, have to shift elements each push & pop
    - instead grow array at end & track which index is the top
    - So have an array to store elements & an int to track the top index
  - Implementation for static array
    - Constructor
      - set top index to -1 to indicate empty stack
    - empty()
      - check if top index is -1
        - if it is, stack is empty
    - push(element)
      - if top index is less than stack capacity - 1
        - increment top index
        - store element in array[top index]
      - else
        - give "full stack" error
    - elementType top()
      - if top index is -1
        - give "empty stack" error
      - else
        - return array[top index]
    - Two methods for pop()
      - void pop()
        - if top index is -1
          - give "empty stack" error
        - else

else
decrement top index
elementType pop()
if top index is -1
give "empty stack" error
else
decrement top index
return array[top index +1]
display()
print top index
for i = top index down to 0
print array[i]
Dynamic Army changes
have to allocate & deallocate array
alter default constructor to allocate array
add
constructor the takes a parameter for capacity
destructor to deallocate array
copy constructor & assignment operator to create a copy
Pseudocode for dynamic array implementation
empty, push, pop, top & display stay the same
Default constructor
set top index to -1
set capacity to default capacity
try to allocate capacity elements to array
if allocation fails
set capacity to 0
Constructor that takes parameter
set top index to -1
set capacity to value given by the parameter
try to allocate capacity elements to array
if allocation fails
set capacity to 0
Destructor
if capacity is 0
deallocate array
Creating a copy from source object
set top index to source's top index
set capacity to source's capacity
try to allocate capacity elements to array
if allocation fails
set top index to -1
set capacity to 0
else
for i=0 to top index
array[i] = Source's array[i]
Copy Constructor
just call creating copy
Assignment Operator
if capacity is not 0
deallocate array
call creating copy steps

7.3 Linked Stacks

Like linked list, linked stack grows & shrinks in response to number of elements currently stored

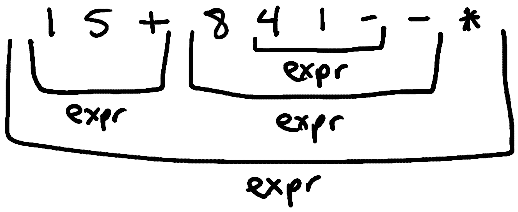
Stacks are really just specialized lists

only allows head insertion & head deletion

only allows head insertion & head deletion
Can use the same node class used for linked list
Stack class contains one member var for head/top node
Operator Pseudocode
Default constructor
set top to NULL
Destructor
while not empty
pop off the top element
-or-
set ptr to top
while ptr is not NULL
set tmp to ptr
set ptr to ptr->getNext()
delete tmp
Creating a copy
can use same traversal from linked list
empty()
if top == NULL
return true
else
return false
push(elementType)
allocate new node & set data
if allocation fails
issue "out of mem" error & return
if empty()
set new node's next to NULL
else
set new node's next to top
set top to new node
Two methods to do pop
Method 1 - just delete top element
void pop()
if empty()
issue "empty Stack" error & return
set tmp to top
set top to top->getNext()
delete tmp
Method 2 - delete top element & return its value
elementType pop()
if empty()
issue "empty stack" error & return
set data to top->getData()
set tmp to top
set top to top->getNext()
delete tmp
return data
elementType top()
if empty()
issue "empty stack" error & return
return top->getData()
Uses for Stacks (7.4 & 7.5)
run-time stack
when a function call occurs, must save the state of that function
allows execution to continue when function call is complete
state contains variables & instruction to return back to

state is saved on the run-time stack
overhead for function calls comes from push/pop on runtime stack
function inlining replaces function call w/ actual function body
inlining avoids using the stack

evaluating expressions
infix format: a + b
postfix format: a b +
postfix expression syntax is:
expr expr operator
where expr is another postfix expression to be evaluated first or an operand
operator is a mathematical operator
postfix example:



corresponds to (1+5)* (8-(4-1))
stack evaluation:
if an operand
push on stack
if an operator
try to pop top two operands
if failed, issue "invalid expr "error
else
calculate result of operation
push result on stack
when done, stack should contain one value that is the result of the whole expr