Laboratory 4: Stacks and Queues

CSC205A Data structures and Algorithms Laboratory B. Tech. 2015

Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: vaishali.cs.et@msruas.ac.in
Tel: +91-80-4906-5555 (2212) WWW: www.msruas.ac.in

Introduction and Purpose of Experiment

- Stacks and queues are very important data structures used in many real time applications. This experiment introduces the development of stack and queue ADT and applying them together

Aim and objectives

Aim:

- To develop stack and queue ADT and to use them for string applications

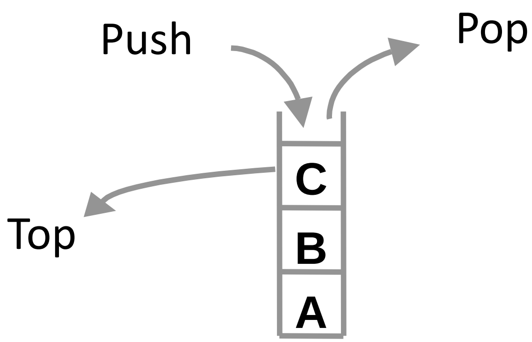
Objectives:

At the end of this lab, the student will be able to

- Design and develop and use stack and demonstrate its operations
- Design and develop and use queue and demonstrate its operations
- Apply stack ADT and Queue ADT to solve simple problems

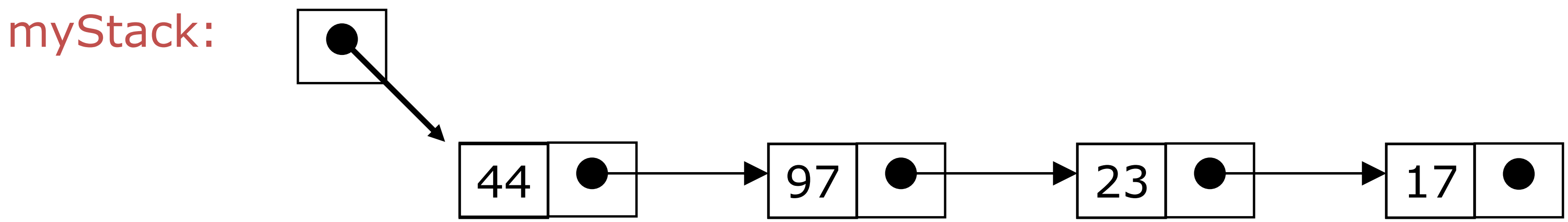
Stacks

- Stacks store arbitrary objects
- Operations
  - push(e): inserts an element to the top of the stack
  - pop(): removes and returns the top element of the stack
  - top(): returns a reference to the top element of the stack, but doesn't remove it
- Optional operations
  - size(): returns the number of elements in the stack
  - empty(): returns a bool indicating if the stack contains any objects



Linked-list implementation of stacks

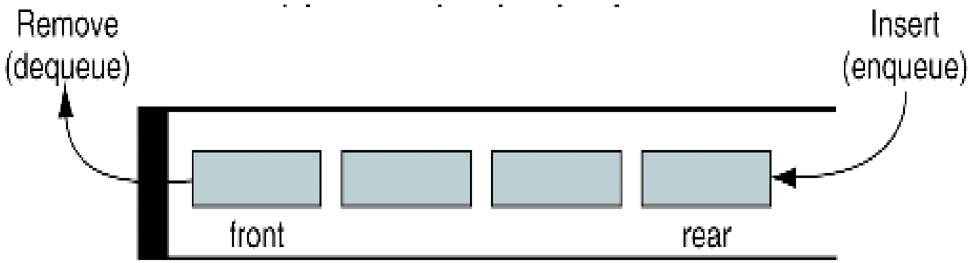
- Since all the action happens at the top of a stack, a singly-linked list (SLL) is a fine way to implement it
- The header of the list points to the top of the stack



- Pushing is inserting an element at the front of the list
- Popping is removing an element from the front of the list

Queues

- Queues store arbitrary objects
- Insertions are at the end of the queue and removals are at the front of the queue
- Main queue operations:
  - enqueue(e): inserts an element at the end of the queue
  - dequeue(): removes and returns the element at the front of the queue



(b) Computer queue

- Auxiliary queue operations:
  - front(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a boolean value indicating if there are no elements in the queue
- Exceptions
  - Attempting to execute dequeue or front on an empty queue throws an EmptyQueueException

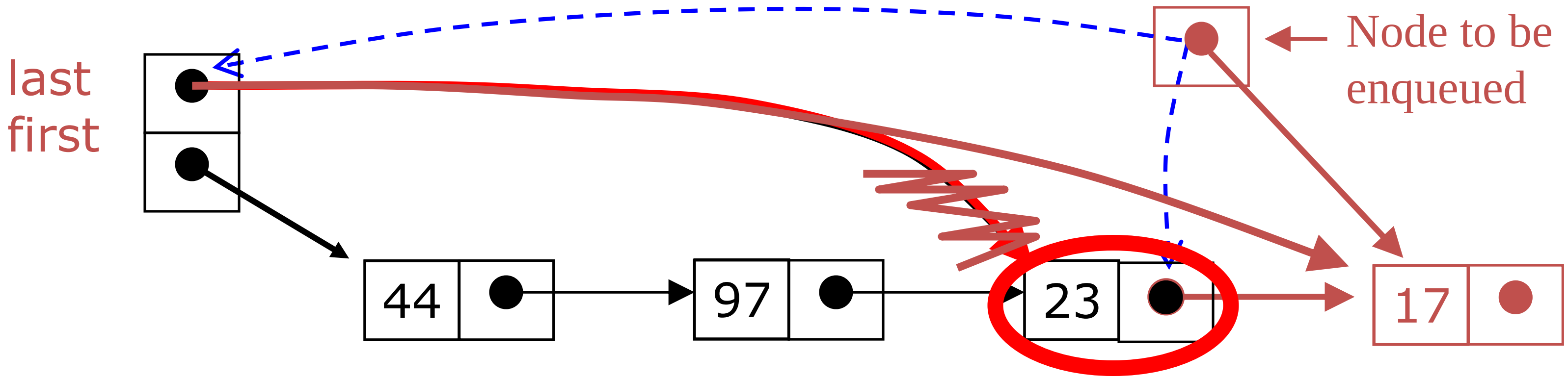
Linked-list implementation of queues

- In a queue, insertions occur at one end, deletions at the other end
- Operations at the front of a singly-linked list (SLL) are O(1), but at the other end they are O(n)
  - Because you have to find the last element each time
- BUT: there is a simple way to use a singly-linked list to implement both insertions and deletions in O(1) time
  - You always need a pointer to the first thing in the list
  - You can keep an additional pointer to the *last* thing in the list

SLL implementation of queues

- In an SLL you can easily find the successor of a node, but not its predecessor
  – pointers are one-way
- If you know where the *last* node in a list is, it's hard to remove that node, but it's easy to add a node after it
- Hence,
  – Use the *first* element in an SLL as the *front* of the queue
  – Use the *last* element in an SLL as the *rear* of the queue
  – Keep pointers to *both* the front and the rear of the SLL

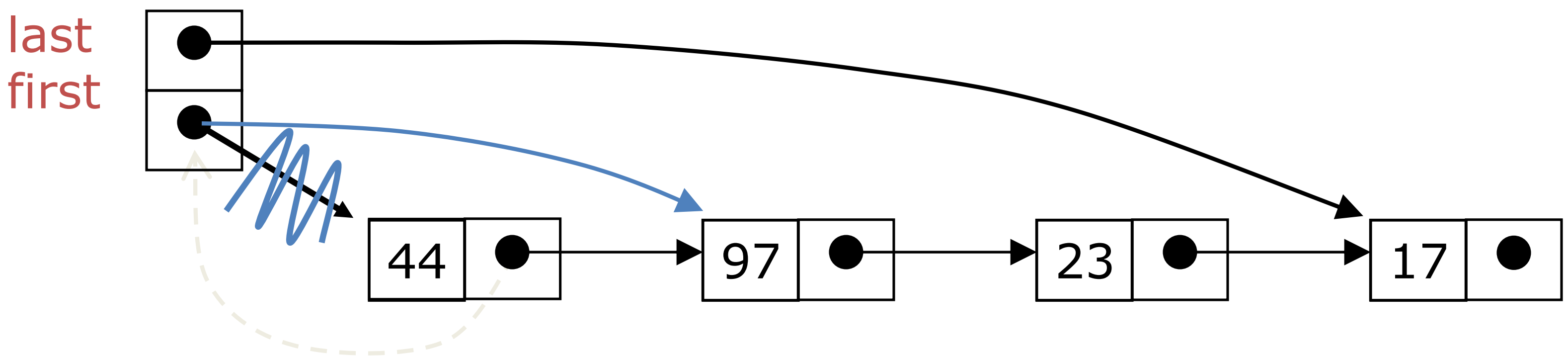
Enqueueing a node



To enqueue (add) a node:

- Find the current last node
- Change it to point to the new last node
- Change the last pointer in the list header

Dequeueing a node



- To dequeue (remove) a node:
  - Copy the pointer from the first node into the header

Queue implementation details

- With an array implementation:
  - you can have both overflow and underflow
  - you should set deleted elements to null
- With a linked-list implementation:
  - you can have underflow
  - overflow is a global out-of-memory condition
  - there is no reason to set deleted elements to null

Linked-list implementation details

- With a linked-list representation, overflow will not happen (unless you exhaust memory, which is another kind of problem)
- Underflow can happen, and should be handled the same way as for an array implementation
- When a node is popped from a list, and the node references an object, the reference (the pointer in the node) does *not* need to be set to null
  - Unlike an array implementation, it really *is* removed--you can no longer get to it from the linked list
  - Hence, garbage collection can occur as appropriate

Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in C language
- Compile the C program
- Design test cases and test the implemented program
- Document the Results
- Analyse and discuss the outcomes of your experiment

Exercise

- Design Stack and Queue data structure ADT using linked list, apply them to check whether a given string is palindrome or not. Write an algorithm and implement the same. Tabulate the output for various inputs and verify against expected values. Analyse the efficiency of the algorithm. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

Key factors and discussion

- Check whether a given string is palindrome or not.
  - A palindrome is a string by reversing the string we get the same string. This has to be verified using both stacks and queues.
  - An integer number can also checked if it is a palindrome number.
  - The ADT should allow appropriate data type conversion depending on the input
  - If void pointer are used the above conversations are possible.

Results and Presentations

- Calculations/Computations/Algorithms

  The calculations/computations/algorithms involved in each program has to be presented

- Presentation of Results

  The results for all the valid and invalid cases have to be presented

- Analysis and Discussions

  how the data is manipulated or transformed, what are the key operations involved. Errors encounters and how they are resolved.

- Conclusions

  Summary

Comments

- Limitations of Experiments

  Outline the loopholes in the program, data structures or solution approach.

- Limitations of Results

  Present the test cases; justify if the program is tested correctly considering all the outcomes. Mention what is not tested, if any.

- Learning happened

  What is the overall learning happened

- Conclusions

  Summary

References

- Gilberg, R. F., and Forouzan, B. A. (2007): A Pseudocode Approach With C, 2nd edn. Cengage Learning