

Analysis, Stacks & Queues

CSE 373
Data Structures & Algorithms
Ruth Anderson
Autumn 2012

Today's Outline

- Tools of the trade: Analysis, Pseudocode, & Proofs
- Review: Stacks and Queues
- Homework #1

9/26/2012

CSE 373 12au - Stacks & Queues

Algorithm Analysis: Why?

- **Correctness:**
 - › Does the algorithm do what is intended?
- **Performance:**
 - › What is the running time of the algorithm?
 - › How much storage does it consume?
- Multiple algorithms may correctly solve a given task
 - › Analysis will help us determine which algorithm to use

9/26/2012

CSE 373 12au - Stacks & Queues

Pseudocode

- In the lectures algorithms will often be presented in pseudocode.
 - › This is very common in the computer science literature
 - › Pseudocode is usually easily translated to real code.
 - › This is programming language independent

9/26/2012

CSE 373 12au - Stacks & Queues

Pseudocode Example

What does this pseudocode do?

```
mystery(v[ ]: integer array, num: integer): integer {
  temp: integer ;
  temp := 0;
  for i := 0 to num - 1 do
    temp := v[i] + temp;
  return temp;
}
```

9/26/2012

CSE 373 12au - Stacks & Queues

Another Pseudocode Example

```
func(v[ ]: integer array, num: integer): integer {
  if num = 0 then
    return 0
  else
    return v[num-1] + func(v,num-1);
}
```

What does this pseudocode do?

9/26/2012

CSE 373 12au - Stacks & Queues

Iterative Algorithm for Sum

- Find the sum of the first **num** integers stored in an array **v**.

```
sum(v[ ]: integer array, num: integer): integer {
    temp_sum: integer ;
    temp_sum := 0;
    for i = 0 to num - 1 do
        temp_sum := v[i] + temp_sum;
    return temp_sum;
}
```

Note the use of pseudocode

9/26/2012

CSE 373 12au - Stacks & Queues

Programming via Recursion

- Write a *recursive* function to find the sum of the first **num** integers stored in array **v**.

```
sum (v[ ]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
```

9/26/2012

CSE 373 12au - Stacks & Queues

Analysis: How?

- We will use mathematical analysis to examine the efficiency of code (next few lectures)
- How do we prove that an algorithm is correct?

9/26/2012

CSE 373 12au - Stacks & Queues

Proof by Induction

- Basis Step:** The algorithm is correct for the base case (e.g. $n=0$) by inspection.
- Inductive Hypothesis ($n=k$):** Assume that the algorithm works correctly for the first k cases, for any k .
- Inductive Step ($n=k+1$):** Given the hypothesis above, show that the $k+1$ case will be calculated correctly.

9/26/2012

CSE 373 12au - Stacks & Queues

Program Correctness by Induction

- Basis Step:** $\text{sum}(v,0) = 0$. ✓
- Inductive Hypothesis ($n=k$):** Assume $\text{sum}(v,k)$ correctly returns sum of first k elements of v , i.e. $v[0]+v[1]+\dots+v[k-1]$
- Inductive Step ($n=k+1$):** $\text{sum}(v,n)$ returns $v[k]+\text{sum}(v,k)$ which is the sum of first $k+1$ elements of v . ✓

9/26/2012

CSE 373 12au - Stacks & Queues

```
sum (v[ ]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
```

- Basis Step:** $\text{sum}(v,0) = 0$. ✓
- Inductive Hypothesis ($n=k$):** Assume $\text{sum}(v,k)$ correctly returns sum of first k elements of v , i.e. $v[0]+v[1]+\dots+v[k-1]$
- Inductive Step ($n=k+1$):** $\text{sum}(v,n)$ returns $v[k]+\text{sum}(v,k)$ which is the sum of first $k+1$ elements of v . ✓

9/26/2012

CSE 373 12au - Stacks & Queues

Today's Outline

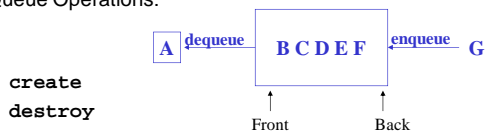
- Tools of the trade: Analysis, Pseudocode, & Proofs
- Review: Stacks and Queues
- Homework #1

9/26/2012

CSE 373 12au - Stacks & Queues

The Queue ADT

Queue Operations:

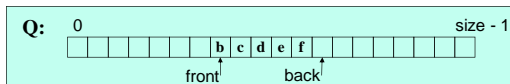


create
destroy
enqueue
dequeue
is_empty

9/26/2012

CSE 373 12au - Stacks & Queues

Circular Array Queue Data Structure



// Basic idea only!

```
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size;
}
```

// Basic idea only!

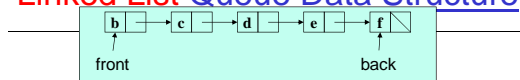
```
obj dequeue() {
  x = Q[front];
  front = (front + 1) % size;
  return x;
}
```

9/26/2012

CSE 373 12au - Stacks & Queues

- What if *queue* is empty?
 - › Enqueue?
 - › Dequeue?
- What if *array* is full?
- How to *test* for empty/full?
- What is the *complexity* of the operations?
- Can you find the k^{th} element in the queue?

Linked List Queue Data Structure



// Basic idea only!

```
enqueue(x) {
  back.next = new Node(x);
  back = back.next;
}
```

// Basic idea only!

```
obj dequeue() {
  x = front.item;
  front = front.next;
  return x;
}
```

9/26/2012

CSE 373 12au - Stacks & Queues

- What if *queue* is empty?
 - › Enqueue?
 - › Dequeue?
- Can *list* be full?
- How to *test* for empty/full?
- What is the *complexity* of the operations?
- Can you find the k^{th} element in the queue?

Circular Array vs. Linked List

9/26/2012

CSE 373 12au - Stacks & Queues

Circular Array vs. Linked List

Array:

- May waste unneeded space or run out of space
- Space per element excellent
- Operations very simple / fast

List:

- Always just enough space
- But more space per element
- Operations very simple / fast

Not in Queue ADT:

- Constant-time access to k^{th} element
- For operation *insertAtPosition*, must shift all later elements

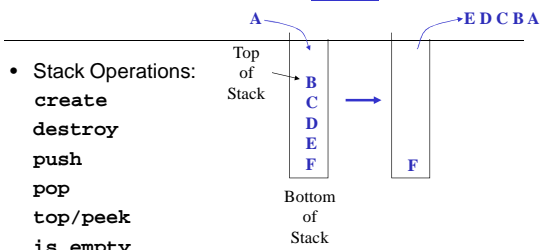
Not in Queue ADT:

- No constant-time access to k^{th} element
- For operation *insertAtPosition*, must traverse all earlier elements

9/26/2012

CSE 373 12au - Stacks & Queues

The Stack ADT



- Stack Operations:
 - `create`
 - `destroy`
 - `push`
 - `pop`
 - `top/peek`
 - `is_empty`
- Can also be implemented with an array or a linked list
 - › This is Project 1!

9/26/2012

CSE 373 12au - Stacks & Queues

Stacks in Practice

- Function call stack
- Removing recursion
- Checking if symbols (parentheses) are balanced
- Evaluating Postfix Notation

9/26/2012

CSE 373 12au - Stacks & Queues

Homework #1 – Sound Blaster!

- Reverse sound clips using a stack!
- Implement a stack interface two ways:
 - › With an array
 - › With linked list nodes (make your own nodes)
- Do NOT use LinkedList or other things from Java Collections

9/26/2012

CSE 373 12au - Stacks & Queues