Today’s Outline

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

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

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
  › It allows for future coding in C++, etc.

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;
}

Another Pseudocode Example

What does this pseudocode do?

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

• Find the sum of the first \texttt{num} integers stored in an array \texttt{v}.

\[
\text{sum}(v[:], \text{num}: \text{integer}) : \text{integer} \\
\begin{align*}
\text{temp}_\text{sum} : & \text{integer} \\
\text{temp}_\text{sum} := & 0; \\
\text{for } i = 0 \text{ to } \text{num} - 1 \text{ do} \quad \text{ temp}_\text{sum} := v[i] + \text{temp}_\text{sum}; \\
\text{return } & \text{temp}_\text{sum};
\end{align*}
\]

Note the use of pseudocode

Programming via Recursion

• Write a recursive function to find the sum of the first \texttt{num} integers stored in array \texttt{v}.

\[
\text{sum}(v[:], \text{num}: \text{integer}) : \text{integer} \\
\begin{align*}
\text{if } & \text{num} = 0 \text{ then} \\
\text{return } & 0 \\
\text{else} & \text{return } v[\text{num}-1] + \text{sum}(v, \text{num}-1); \\
\end{align*}
\]

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?

Proof by Induction

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

Program Correctness by Induction

• \textbf{Basis Step}: \texttt{sum(v,0)} = 0. 
• \textbf{Inductive Hypothesis (n=k)}: Assume \texttt{sum(v,k)} correctly returns sum of first \texttt{k} elements of \texttt{v}, i.e. \texttt{v[0]+v[1]+…+v[k-1]}
• \textbf{Inductive Step (n=k+1)}: \texttt{sum(v,n)} returns \texttt{v[k]+sum(v,k)} which is the sum of the \texttt{k+1}st element plus the sum of the first \texttt{K} elements, giving the first \texttt{k+1} elements of \texttt{v}. 


Today’s Outline

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

The Queue ADT

Queue Operations:

create
destroy
enqueue
dequeue
is_empty

The Queue ADT

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;
}

Linked List Queue Data Structure

// Basic idea only!
enqueue(x) {
    x = front.item;
    front = front.next;
    return x;
}

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 kth element
- For operation insertAtPosition, must shift all later elements

Circular Array vs. Linked List

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

Not in Queue ADT:
- Constant-time access to kth element
- For operation insertAtPosition, must shift all earlier elements

Do we care?
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! It should be mostly review.

Stacks in Practice

- Function call stack
- Removing recursion
- Checking if symbols (parentheses) are balanced
- Evaluating postfix notation
- Reversing a sound wave

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
- Has been posted and is due April 12.