### 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

### Pseudocode Example

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

What does this pseudocode do?

### Another Pseudocode Example

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

What does this pseudocode do?
Iterative Algorithm for Sum

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

\begin{verbatim}
sum(v[\cdot]: 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;
}
\end{verbatim}

Programming via Recursion

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

\begin{verbatim}
sum [\cdot]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
\end{verbatim}

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} (\texttt{n=k}): Assume that the algorithm works correctly for the first \texttt{k} cases, for any \texttt{k}.
- \textbf{Inductive Step} (\texttt{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} (\texttt{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} (\texttt{n=k+1}): \texttt{sum(v,n)} returns \texttt{v[k]+sum(v,k)} which is the sum of 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

Circular Array Queue Data Structure

```java
// 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;
}
```

- 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 kth element in the queue?

Linked List Queue Data Structure

```java
// 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;
}
```

- 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 kth element in the queue?

Circular Array vs. Linked List

- Array:
  - May waste unneeded space or run out of space
  - Space per element excellent
  - Operations very simple / fast
  - Constant-time access to kth element
  - For operation insertAtPosition, must shift all later elements
    - Not in Queue ADT
- List:
  - Always just enough space
  - But more space per element
  - Operations very simple / fast
  - No constant-time access to kth element
  - For operation insertAtPosition must traverse all earlier elements
    - Not in Queue ADT

Circular Array vs. Linked List

- Can also be implemented with an array or a linked list
  - This is Project 1!

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!
Stacks in Practice

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

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

When did you take cse 143 (what quarter)?

<table>
<thead>
<tr>
<th>Number</th>
<th>Answer</th>
<th>Frequency</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>- summer 11</td>
<td>5</td>
<td>6.25%</td>
</tr>
<tr>
<td>2</td>
<td>- spring 11</td>
<td>7</td>
<td>8.75%</td>
</tr>
<tr>
<td>3</td>
<td>- winter 11</td>
<td>23</td>
<td>28.75%</td>
</tr>
<tr>
<td>4</td>
<td>- autumn 10</td>
<td>10</td>
<td>12.50%</td>
</tr>
<tr>
<td>5</td>
<td>- summer 10</td>
<td>20</td>
<td>25.00%</td>
</tr>
<tr>
<td>6</td>
<td>- spring 10</td>
<td>18</td>
<td>22.50%</td>
</tr>
<tr>
<td>7</td>
<td>- before spring 10</td>
<td>17</td>
<td>21.25%</td>
</tr>
<tr>
<td>8</td>
<td>- I did not take cse 143 at UW (AP or transfer credit)</td>
<td>4</td>
<td>5.00%</td>
</tr>
<tr>
<td>9</td>
<td>Other:</td>
<td>1</td>
<td>1.25%</td>
</tr>
</tbody>
</table>