Overview

- Recursion
Recursion

- Defining a problem in terms of:
  - Some simple trivial case
  - A more complex case which ultimately leads to the trivial case
  - A way to define a problem in terms of itself
Example Problem

• Say we want to calculate the length of a linked list recursively

• A list is represented as a Node*

  • Base case?

  • Length of list besides first element?

  • Recursive case?

  ```
  int length(Node* list);
  ```
Example Problem

```c
int length(Node* list) {
    if (list == NULL) {
        return 0; // base case
    } else {
        return (1 + // this node’s length
                // length of the rest of
                // the list
            length(list->getNext()));
    }
}
```
Revised Problem

- Say we want to determine the length of a list, but with a tweak: we also take the length of the list so far
  
  - Base case?
  
  - Length of list besides first element?
  
  - Recursive case?

- What does the initial call look like?

```c
int firstCall(Node* list);
int length2(Node* list, int soFar);
```
int length2(Node* list, int soFar) {
    if (list == NULL) {
        return soFar; // base case
    } else {
        // get the length of the rest of
        // the list, and say that the
        // length so far is + 1
        return length2(list->getNext(),
                        soFar + 1);
    }
}

int firstCallCall(Node* list) {
    return length2(list, 0);
}
Relationship to Loops

• \texttt{length2} is more similar to an iterative implementation than it may seem at first

• \texttt{while} dynamically inserts \texttt{ifs} as many times as needed

• Recursion dynamically inserts the body of a function as many times as needed

• After doing these expansions, they basically look the same!
Recursion With Arrays
Recursion With Arrays

- If we look at arrays in a similar way as linked lists, operations become more clear.
- The index acts like a pointer to a particular node.
  - What is the base case?
  - Recursive case?
Recursion With Arrays

• If we look at arrays in a similar way as linked lists, operations become more clear

• The index acts like a pointer to a particular node
  • What is the base case?
    • Index out of array
  • Recursive case?
    • Index in array
Example

• Determine the sum of an array of integers, starting from a particular index. An array containing no elements has a sum of 0.

• Base case?

• Recursive case?

```c
int sumFromIndex(int* array, int length, int index);
```
Example

- Determine the sum of an array of integers, starting from a particular index. An array containing no elements has a sum of 0.

- Base case? - index out of bounds (0)

- Recursive case? - index in bounds (current element + sum of rest)

```c
int sumFromIndex(int* array, int length, int index);
```
int sumFromIndex(int* array, int length, int index) {
    if (index >= length) return 0;
    else {
        int restSum = sumFromIndex(array, length, index + 1);
        return restSum + array[index];
    }
}
Recursion Pros

• If your recursive case is always guaranteed to reach a base case, infinite recursion is impossible (appeals to induction)

• No more infinite loops!

• Vital for more complex recursive data structures (e.g., trees)

• Easier to understand :)

Monday, August 4, 14
Recursion Cons

• If you’re not careful, you can run out of stack space (a stack overflow)
• Not written in a tail-recursive way
• Compiler is too stupid to notice it’s tail-recursive
• Very large input
Find the Problem
int length(Node* list) {
  if (list == NULL) {
    return 0;
  } else {
    return 1 + length(list);
  }
}
What’s Wrong?

Recursive case never reaches base case -
infinite recursion
int helper(List* l) {
    if (l->getTail() != NULL) {
        l->getTail()->setNext(head);
    }
    return calcSum(l->getHead());
}

int calcSum(Node* n) {
    if (n == NULL) return 0;
    else return (n->getInt() +
                 calcSum(n->getNext()));
}
int helper(List* l) {
    if (l->getTail() != NULL) {
        l->getTail()->setNext(head);
    }
    return calcSum(l->getHead());
}

int calcSum(Node* n) {
    if (n == NULL) return 0;
    else return (n->getInt() +
    calcSum(n->getNext()));
}

Infinite recursion possible - list may never have NULL in it
Additional Problems
More Array Recursion Examples

• You may add helpers as necessary

```c
bool containsInt(int* array, int size, int what);
int stringLength(char* str);
void setAllTo(int* array, int size, int toWhat);
bool allEqualThis(int* array, int size, int what);
int getProduct(int* array, int size);
int largestElement(int* array, int size); // ??
```
More List Recursion Examples

- You may add helpers as necessary

bool containsInt(Node* head, int what);
void setAllTo(Node* head, int toWhat);
bool allEqualThis(Node* head, int what);
int getProduct(Node* head);

int largestElement(Node* head); // ??