CS 106B, Lecture 9
Recursive Data
Plan for Today

- More recursion practice!
- Learning goals for today
  - Understand how to recognize self-similarity in problems and use recursion to solve these problems.
  - See examples of recursively structured data.
  - You should practice: CodeStepByStep, section problems, or examples from the textbook.
Recap: Recursion Tips

- Look for *self-similarity*
- Find the minimum *amount of work*
- Make the problem *simpler* by doing the least amount of work possible
- *Trust* the recursion
- Find a stopping point (*base case*)
power exercise

• Write a function \texttt{power} that accepts integer parameters for a base and exponent and computes base \(^ \text{exponent} \).

  – Write a \texttt{recursive} version of this function (one that calls itself).
  – Solve the problem \texttt{without using any loops}.

  – How is this problem \textit{self-similar}?
  – What is the minimum \textit{amount of work}?
  – How can we make the problem \textit{simpler} by doing the least amount of work?
  – What is our stopping point (\textit{base case})?
• Write a function `power` that accepts integer parameters for a base and exponent and computes base $^\text{exponent}$.

  – Write a recursive version of this function (one that calls itself).
  – Solve the problem without using any loops.

  – How is this problem self-similar? Realize $x^n = x \times x^{n-1}$
  – What is the minimum amount of work?
  – How can we make the problem simpler by doing the least amount of work?
  – What is our stopping point (base case)? $n = 0$
    • Why not $n = 1$?
Initial solution

// Returns base ^ exp.
// Assumes exp >= 1.
int power(int base, int exp) {
    if (exp == 1) {
        return base;
    } else {
        return base * power(base, exp - 1);
    }
}
The call stack

- Each previous call waits for the next call to finish.
  
  ```
  cout << power(5, 3) << endl;
  ```

```cpp
// first call:  5  3
int power(int base, int exp) {
    if (exp == 1) {
        // second call:  5  2
        return base;
    }
    else {
        // third call:  5  1
        return base * power(base, exp - 1);
    }
}
```
Recursion Zen:

The real, even simpler, base case is an exp of 0, not 1:

```c
int power(int base, int exp) {
    if (exp == 0) {
        // base case; base^0 = 1
        return 1;
    } else {
        // recursive case: x^y = x * x^(y-1)
        return base * power(base, exp - 1);
    }
}
```

Recursion Zen: The art of properly identifying the best set of cases for a recursive algorithm and expressing them elegantly.

Opposite is arms-length recursion (our informal term)
• **precondition**: Something your code *assumes is true* when called.
  – Often documented as a comment on the function's header:

    ```c
    // Returns base ^ exp.
    // Precondition: exp >= 0
    int power(int base, int exp) {
    ```

  – Stating a precondition doesn't really "solve" the problem, but it at least documents our decision and warns the client what not to do.

  – What if the caller doesn't listen and passes a negative power anyway? What if we want to actually *enforce* the precondition?
error(expression);

- In Stanford C++ lib's "error.h"
- Generates an exception that will crash the program, unless it has code to handle ("catch") the exception.
- alternative: throw something
  - something can be an int, a string, etc.
- Why would anyone ever want a program to crash?
// Returns base ^ exp.
// Precondition: exp >= 0
int power(int base, int exp) {
    if (exp < 0) {
        throw "illegal negative exponent";
    } else ...  
    ...  
}
An optimization

• Notice the following mathematical property:

\[3^{12} = 9^6 = (3^2)^6 = ((3^2)^2)^3\]

– When does this "trick" work?
– How can we incorporate this optimization into our pow code?
– Why bother with this trick if the code already works?
// Returns base ^ exp.
// Precondition: exp >= 0
int power(int base, int exp) {
    if (exp < 0) {
        throw "illegal negative exponent";
    } else if (exp == 0) {
        // base case; any number to 0th power is 1
        return 1;
    } else if (exp % 2 == 0) {
        // recursive case 1: x^y = (x^2)^(y/2)
        return power(base * base, exp / 2);
    } else {
        // recursive case 2: x^y = x * x^(y-1)
        return base * power(base, exp - 1);
    }
}
• Write a recursive function `convertFromBinary` that accepts an a string of that number's representation in binary (base 2) and returns the base 10 int equivalent.

  – Example: `convertFromBinary ("111")` returns 7
  – Example: `convertFromBinary ("1100")` returns 12
  – Example: `convertFromBinary ("101010")` returns 42

  \[
  42 = 4 \times 10 + 2 \times 1 = 1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1
  \]
convertFromBinary exercise

• How is this problem *self-similar*?
• What is the *smallest amount of work*?
• When should the recursion stop?

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10100</td>
</tr>
<tr>
<td>40</td>
<td>101000</td>
</tr>
<tr>
<td>41</td>
<td>101001</td>
</tr>
</tbody>
</table>
convertFromBinary solution

// Returns the given int's binary representation.
// Precondition: n >= 0
int convertFromBinary(string binary) {
    int length = binary.length();
    if (length == 1) {
        // base case: binary is same as base 10
        return stringToStringInteger(binary);
    }
    // recursive case: break number apart
    string lastCharacter = binary.substr(length - 1);
    string beginning = binary.substr(0, length - 1);
    return 2 * convertFromBinary(beginning) +
           convertFromBinary(lastCharacter);
}
```cpp
int main() {
    cout << convertFromBinary("110") << endl;
}

int convertFromBinary(string binary) {
    int length = binary.length();
    if (length == 1) return stringToInteger(binary);
    string lastCharacter = binary.substr(length - 1);
    string beginning = binary.substr(0, length - 1);
    return 2 * convertFromBinary(beginning) +
    convertFromBinary(lastCharacter);
}
```
Announcements

• Homework 2 due on Wednesday at 5PM
• Homework 1 grades will be released by your section leader on or before Wednesday
• Your partner (if you choose to have one) must be in your section, and you should submit together through Paperless
• Shreya's OH changeup
  – Tuesday, 8:30-10:30AM
  – Wednesday, 9:30-10:30AM
  – Both open to SCPD and non-SCPD students, sign up on QueueStatus (link on sidebar of website), be prepared to use Google Hangouts
reverseLines exercise

• Write a recursive function `reverseLines` that accepts a file input stream and prints the lines of that file in reverse order.

  – Example input file:
  
  ```
  Roses are red,
  Violets are blue.
  All my base
  Are belong to you.
  ```
  
  Expected console output:
  
  ```
  Are belong to you.
  All my base
  Violets are blue.
  Roses are red,
  ```

  – What are the cases to consider?
    
    • How can we solve a small part of the problem at a time?
    • What is the *self-similarity* of this problem?
    • What is a file that is very easy to reverse?
Reversal pseudocode

• Reversing the lines of a file:
  – Read a line L from the file.
  – Print the rest of the lines in reverse order.
  – Print the line L.

• If only we had a way to reverse the rest of the lines of the file....
void reverseLines(ifstream& input) {
    string line;
    if (getline(input, line)) {
        // recursive case
        reverseLines(input);
        cout << line << endl;
    }
}

– Where is the base case?
• Write a function `crawl` accepts a file name as a parameter and prints information about that file.
  – If the name represents a normal file, just print its name.
  – If the name represents a directory, print its name and information about every file/directory inside it, indented.

```plaintext
course
  handouts
    syllabus.doc
    lecture-schedule.xls
homework
  1-gameoflife
    life.cpp
    life.h
    GameOfLife.pro
```

– **recursive data**: A directory can contain other directories.
#include "filelib.h"

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDirectory(name)</td>
<td>creates a new directory with given path name</td>
</tr>
<tr>
<td>deleteFile(name)</td>
<td>removes file from disk</td>
</tr>
<tr>
<td>fileExists(name)</td>
<td>whether this file exists on the disk</td>
</tr>
<tr>
<td>getCurrentDirectory()</td>
<td>returns directory the current C++ program runs in</td>
</tr>
<tr>
<td>getExtension(name)</td>
<td>returns file's extension, e.g. &quot;foo.cpp&quot; → &quot;.cpp&quot;</td>
</tr>
<tr>
<td>getHead(name), getTail(name)</td>
<td>separate a file path into the directory and file part; for &quot;a/b/c/d.txt&quot;, head is &quot;a/b/c&quot;, tail is &quot;d.txt&quot;</td>
</tr>
<tr>
<td>isDirectory(name)</td>
<td>returns whether this file name represents a directory</td>
</tr>
<tr>
<td>.isFile(name)</td>
<td>returns whether this file name represents a regular file</td>
</tr>
<tr>
<td>listDirectory(name)</td>
<td>returns a Vector&lt;string&gt; with the names of all files contained in the given directory</td>
</tr>
<tr>
<td>readEntireFile(name, v)</td>
<td>reads lines of the given file into a vector of strings</td>
</tr>
<tr>
<td>renameFile(old, new)</td>
<td>changes a file's name</td>
</tr>
</tbody>
</table>
Optional parameters

• We cannot vary the indentation without an extra parameter:

```java
void crawl(string filename, string indent) {
```

• Often the parameters we need for our recursion do not match those the client will want to pass.

  One solution is to use a default parameter value:

```java
void crawl(string filename, string indent = "") {
```

  – The client can call `crawl` passing only one parameter.
  – The recursive calls can pass the second parameter to indent.
// Prints information about this file, 
// and (if it is a directory) any files inside it.
void crawl(string filename, string indent = "") {
    cout << indent << getTail(filename) << endl;
    if (isDirectory(filename)) {
        // recursive case; print contained files/dirs
        Vector<string> filelist;
        listDirectory(filename, filelist);
        for (string subfile : filelist) {
            crawl(filename + "/", subfile,
                  indent + "	"");
        }
    }
}
evenDigits exercise

• Write a recursive function evenDigits that accepts an integer and returns a new number containing only the even digits, in the same order. If there are no even digits, return 0.

  – Example: evenDigits(8342116) returns 8426
  – Example: evenDigits(40109) returns 400
  – Example: evenDigits(8) returns 8
  – Example: evenDigits(-163505) returns -60
  – Example: evenDigits(35179) returns 0

– Write the function recursively and without using any loops.
evenDigits solution

// Returns a new integer containing only the even-valued digits from the given integer, in the same order.
// Returns 0 if there are no even digits.

int evenDigits(int n) {
    if (n < 0) {
        return -evenDigits(-n);
    } else if (n == 0) {
        return 0;
    } else if (n % 2 == 0) {
        return 10 * evenDigits(n / 10) + n % 10;
    } else {
        return evenDigits(n / 10);
    }
}