CS112 Lab 01, Jan 21, 24 2010
http://cs-people.bu.edu/deht/cs112_spring11/lab01/

Diane H. Theriault
deht@cs.bu.edu
http://cs-people.bu.edu/deht/
Preliminaries

• I want to help you.
  – Please put “CS112” in the subject line when you email

• Don’t get stuck!
  – Technical details (development environment, compiler errors) can be as much work as the actual task at hand

• Don’t procrastinate.
  – Even professionals can get stuck on little things. Give yourself enough time to finish.

• Don’t get behind.
  – You will need to understand previous material in order to go forward
Recursion

• Define the solution to a big problem in terms of the solution to a similar, slightly smaller problem

• You know how to functionally decompose a “normal” problem using functions

```java
Boolean solveProblem(parameter x){
    int result = Step1(x);
    return Step2(result);
}
```
A Cartoon Recursive Function

```c
int recursiveProblem(int x)
{
    if (x == 1) return 1;
    return recursiveProblem(x-1) + 1;
}
```
A Cartoon Recursive Function

• Recursive solutions have two important parts:
  – Base Case (what to do with trivial inputs)
  – Recursive Call (how to solve the problem)

```c
int recursiveProblem(int x)
{
    if(x == 1) return 1;  //base case
    return recursiveProblem(x-1) + 1;  //recursive call
}
```
A Classic Recursive Problem

• Fibonacci numbers (you’ve seen this on the SAT!)
• Each number in the sequence is the sum of the previous two numbers

```cpp
int Fibonacci(int x)
{
    if(x == 0) return 0;
    if(x == 1) return 1;
    return Fibonacci(x-1) + Fibonacci(x-2);
}
```
Recursive functions and Stack frames

• Every time you call any function, the local variables, arguments, etc. are saved, before jumping to the next function.
• When a function returns, the caller’s state is restored before continuing execution
• The place where this information is saved is called “the stack” or “the call stack.”
• Practical Upshot: Each call to a function has its own copy of its arguments and local variables (a.k.a. a stack frame)
Normal functions and Stack frames

Void doWork(int x)
{
    int y = 5;
    int z = StepOne(x, y);
}

int StepOne(int x, int y)
{
    int q = StepTwo(x * y, x);
    return q;
}

int StepTwo(int q, int r)
{
    int x = q / r;
    return x;
}
Normal functions and Stack frames

Void doWork(int x)
{
    int y=5;
    int z = StepOne(x,y);
}
int StepOne(int x, int y)
{
    int q = StepTwo(x*y, x)
    return q;
}
int StepTwo(int q, int r)
{
    int x = q/r;
    return x;
}
Normal functions and Stack frames

Void doWork(int x)
{
    int y = 5;
    int z = StepOne(x, y);
}
int StepOne(int x, int y)
{
    int q = StepTwo(x*y, x);
    return q;
}
int StepTwo(int q, int r)
{
    int x = q/r;
    return x;
}
Normal functions and Stack frames

Void doWork(int x)
{
    int y=5;
    int z = StepOne(x,y);
}

int StepOne(int x, int y)
{
    int q = StepTwo(x*y, x)
    return q;
}

int StepTwo(int q, int r)
{
    int x = q/r;
    return x;
}

<table>
<thead>
<tr>
<th>doWork()</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 2</td>
</tr>
<tr>
<td>y=5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StepOne()</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 2</td>
</tr>
<tr>
<td>y = 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StepTwo()</th>
</tr>
</thead>
<tbody>
<tr>
<td>q = 10</td>
</tr>
<tr>
<td>r = 2</td>
</tr>
<tr>
<td>x = 5</td>
</tr>
</tbody>
</table>
Normal functions and Stack frames

Void doWork(int x)
{
    int y=5;
    int z = StepOne(x,y);
}

int StepOne(int x, int y)
{
    int q = StepTwo(x*y, x)
    return q;
}

int StepTwo(int q, int r)
{
    int x = q/r;
    return x;
}
Normal functions and Stack frames

Void doWork(int x)
{
   int y=5;
   int z = StepOne(x,y);
}
int StepOne(int x, int y)
{
   int q = StepTwo(x*y, x)
   return q;
}
int StepTwo(int q, in r)
{
   int x = q/r;
   return x;
}
Recursive functions and Stack frames

Void doWork(int x)
{
    int z = recursive(x)
}
int recursive(int x)
{
    int y;
    if(x == 1)
        y = 5;
    else
        y = recursive(x-1)+1;
    return y;
}
Recursive functions and Stack frames

```java
void doWork(int x) {
    int z = recursive(x);
}

int recursive(int x) {
    int y;
    if(x == 1) {
        y = 5;
    } else {
        y = recursive(x-1)+1;
    }
    return y;
}
```

doWork()
Recursive functions and Stack frames

Void doWork(int x)
{
    int z = recursive(x)
}

int recursive(int x)
{
    int y;
    if(x == 1)
        y = 5;
    else
        y = recursive(x-1)+1;
    return y;
}
Recursive functions and Stack frames

Void doWork(int x)
{
    int z = recursive(x)
}

int recursive(int x)
{
    int y;
    if(x == 1)
        y = 5;
    else
        y = recursive(x-1)+1;
    return y;
}
Recursive functions and Stack frames

Void doWork(int x)
{
    int z = recursive(x)
}
int recursive(int x)
{
    int y;
    if(x == 1)
        y = 5;
    else
        y = recursive(x-1)+1;
    return y;
}
Recursive functions and Stack frames

Void doWork(int x)
{
    int z = recursive(x)
}
int recursive(int x)
{
    int y;
    if(x == 1)
        y = 1;
    else
        y = recursive(x-1)+1;
    return y;
}
How to Think about Recursive Functions

• Use wishful thinking:
  
  “If only I knew the answer to _____, solving this problem would be easy.”

• Understand which variables are changing, and which are constant

• Think about the easiest version(s) of your problem (define your base case)

• Run your program in your head (or on paper) using some simple, but not trivial inputs
Practical Lab: Implementing Exponentiation

- Write a function to recursively compute the function \( f(x,n) = x^n \)
- Compare your results with the built-in java function `Math.pow(x,n);`
- This can be implemented by repeated squaring: \( f(x,n) = x^{n/2} * x^{n/2} \)
  - Assume that \( n \) is an integer
  - What do you need to do if \( n \) is odd?
Practical Lab:
Counting the number of recursive calls

• Add features to your exponentiation function to count the number of recursive calls.
• How can you improve your efficiency by caching your results?
• Count the number of recursive calls with and without your optimization
Things you will need for HW1

- Derivative of a polynomial function:
  - $F(x) = x^n + c \rightarrow F'(x) = n \times x^{n-1}$

- Java exponentiation function:
  - `Math.pow(x,n); // x^n`

- Declaring an array in Java
  - `int [] intArray = new int[10];`

- Input and output
  - (Google “standard out” and “standard in”)
  - `System.out.println(…)`
  - `BufferedReader in = new BufferedReader(new InputStreamReader(System.in));`