Thinking Recursively

CS 106B

Programming Abstractions
Fall 2016
Stanford University
Computer Science Department
Announcements

- Assignment 2 due Saturday
- Remember no LaIR on Friday night
- New texts for random writer
Today’s Goal

1. More practice with recursion
2. Be able to write graphical recursive functions

One of the programs on assignment 3 will use graphical recursion
Recursion:

A problem solving technique in which problems are solved by reducing them into smaller problems of the same form.
The structure of recursive functions is typically like the following:

```plaintext
recursiveFunction:
  if (test for simple case) {
    Compute the solution without recursion
  } else {
    Break the problem into subproblems of the same form
    Call recursiveFunction on each subproblem
    Reassemble the results of the subproblems
  }
```

Recursive paradigm

Recursion Template
Three Musts of Recursion

1. Your code must have a case for all valid inputs.

2. You must have a base case (makes no recursive calls).

3. When you make a recursive call it should be to a simpler instance (forward progress towards base case).
Start with Delicious Example
((1+3)*(2*(4+1)))
Challenge

Implement a function which evaluates an expression string:

“(((1+3)*(2*(4+1))))”

“(7+6)”

“(((4*(1+2))+6)*7)”

Only * or +
Anatomy of an Expression

An expression is always one of these three things:

- **number**
- \((\text{expression} + \text{expression})\)
- \((\text{expression} \times \text{expression})\)
Anatomy of an Expression

Left Exp  \( \text{Op.} \)  Right Exp

\(((1*3)+(4+2))\)

Left Expression  Joining Operator  Right Expression
How do we evaluate \(((1\times 17)+\left(2\times (3+(4\times 9))\right))\)?

\[
(1 \times 17) + (2 \times (3 + (4 \times 9)))
\]
It is Recursive?

\[((1\times3)+(4+2))\]

The big instance of this problem is:
\[((1\times3)+(4+2))\]

The smaller instances are:
\((1\times3)\) and \((4+2)\)
Write this function

```cpp
int evaluate(string exp);
```

“((1*3)+(4+2))” //returns 11

Using these library functions

```cpp
stringIsInteger(exp)
stringToInteger(exp)
```

And these exp helper functions

```cpp
char op = getOperator(exp);
```

//returns ‘+’

```cpp
string left = getLeftExp(exp);
```

//returns “(1*3)”

```cpp
string right = getRightExp(exp);
```

//returns “(4+2)”
(**1*3**)+(4+2))

```c
int evaluate(expression):
    • If **expression** is a number, return **expression**
    • Otherwise, break up **expression** by its **operator**:
      – *leftResult* = evaluate(*leftExpression*)
      – *rightResult* = evaluate(*rightExpression*)
      – Return *leftResult* **operator** *rightResult*
```
Solution

```c
int evaluate(string exp) {
    if (stringIsInteger(expression)) {
        return stringToInteger(expression);
    } else {
        char op = getOperator(exp);
        string left = getLeftExp(exp);
        string right = getRightExp(exp);
        int leftResult = evaluate(left);
        int rightResult = evaluate(right);

        if (op == '+') {
            return leftResult + rightResult;
        } else if (op == '*') {
            return leftResult * rightResult;
        }
    }
}
```
int evaluate(string exp) {
    if (stringIsInteger(expression)) {
        return stringToInteger(expression);
    } else {
        char op = getOperator(exp);
        string left = getLeftExp(exp);
        string right = getRightExp(exp);
        int leftResult = evaluate(left);
        int rightResult = evaluate(right);

        if (op == '+') {
            return leftResult + rightResult;
        } else if (op == '*') {
            return leftResult * rightResult;
        }
    }
    return 0;
}

Solution

exp = "(((1*3)+((4*5)+2))"

op = ‘+’

left = "(1*3)"

right = "((4*5)+2)"

leftResult = 3

rightResult = 22
Here is the key function behind the helper methods:

```java
int getOppIndex(string exp){
    int parens = 0;
    for (int i = 0; i < exp.length(); i++) {
        char c = exp.charAt(i);
        if (c == '(') { parens++; }
        else if (c == ')') { parens--; }
        if (parens == 0 && (c == '+' || c == '*')) {
            return i;
        }
    }
}
```
Recursion you can see
**fractal**: A recurring graphical pattern. Smaller instances of the same shape or pattern occur within the pattern itself.
Many natural phenomena generate fractal patterns:
1. earthquake fault lines
2. animal color patterns
3. clouds
4. mountain ranges
5. snowflakes
6. crystals
7. DNA
8. …
Ready?
Let’s go!
Cantor Fractal
Cantor Fractal
Parts of a Cantor set image... are Cantor set images
Cantor Fractal

Start

Another cantor set

Also a cantor set

End
Levels of Cantor

6 Levels
Levels of Cantor
Levels of Cantor
How to Draw a Level 1 Cantor?
How to Draw a Level n Cantor?

1. Draw a line from start to finish

2. Draw a Cantor of size n-1

3. Draw a Cantor of size n-1
GWindow w;
GPoint a(100, 100);
cout << a.getX() << endl;
GPoint

GPoint a

GPoint b

GWindow w;
GPoint a(100, 100);
GPoint b(20, 20);
cout << a.getX() << endl;
w.drawLine(a, b);
Depth 1 Snowflake Line
Depth 2 Snowflake Line
Depth 3 Snowflake Line (in progress)
Depth 3 Snowflake Line (in progress)
Depth 3 Snowflake Line (in progress)
Depth 3 Snowflake Line (in progress)
Extra Problems Online
Fractal Tree
Fractal Circles
Recursive Ray Tracing
The End