CS 106B
Lecture 8: Fractals

Wednesday, April 18, 2018

Programming Abstractions
Spring 2018
Stanford University
Computer Science Department

Nick Troccoli

reading:
Programming Abstractions in C++, Chapter 5.4-5.6
Today's Topics

• Logistics:
  • Serafini Due Thursday April 19th, noon
  • Towers of Hanoi video: https://www.youtube.com/watch?v=2SUvWfNJSsM

• Assignment 3: Recursion —posted at 12PM tomorrow, due next Thursday.
  • Fractals
  • Grammar Solver
  • YEAH Hours: Thurs. 4/19 4:30PM-5:20PM in Skilling Auditorium

• Recursion example: expressions
• Fractals
Three Musts of Recursion

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

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

3. When you make a recursive call it should be to a simpler instance and make forward progress towards the base case.
The base case represents the simplest possible instance of the problem you are solving.

- How many people are behind you? There is no one behind me 😊
- What is $x^n$? $x^0 = 1$
- Solve the Towers of Hanoi? Towers of Hanoi with 1 ring
- Hailstone problem starting at $n$? $n = 1$
An Observation

The **recursive** case represents how you can break down the problem into smaller instances of the same problem.

How many people are behind you? \(1 + \# \text{ behind the person behind me}\)

What is \(x^n\)? \(x \times x^{n-1}\)

Solve the Towers of Hanoi? \(N-1\) disks to aux, 1 disk to target, \(N-1\) disks to target

Hailstone problem starting at \(n\)? Even? Hailstone \((n/2)\). Odd? Hailstone \((3n+1)\)
Recursion Example

\[((1+3)*(2*(4+1)))\]
Recursion Example

$((1*17)+(2*(3+(4*9))))$
Challenge

Implement a function which evaluates an expression string:

```
"((1+3)*(2*(4+1)))"

"(7+6)"

"(((4*(1+2))+6)*7)"
```

(only needs to implement * or +)
Anatomy of an Expression

An expression is always one of these three things

- number
- expression
- (expression + expression)
- (expression * expression)
Anatomy of an Expression

$$((1 \times 3) + (4 \times 2))$$

left expression  \rightarrow  \text{joining operator}  \rightarrow  \text{right expression}
Anatomy of an Expression

How do we evaluate \(( (1 \times 17) + (2 \times (3 + (4 \times 9))) )\)?

\[
( (1 \times 17) + (2 \times (3 + (4 \times 9))) )
\]

\[
= (1 \times 17) + (2 \times (3 + (4 \times 9)))
\]

\[
= 17 + (2 \times (3 + (4 \times 9)))
\]

\[
= 17 + (2 \times (3 + 36))
\]

\[
= 17 + (2 \times 39)
\]

\[
= 17 + 78
\]

\[
= 95
\]
Is it Recursive? Yes!

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

The big instance of this problem is:

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

The smaller instances are:

3 and 2
Task

Write this function: `int evaluate(string exp);`

```
"((1*3)+(4+2))" // returns 9
```

Using these library functions:

- `stringIsInteger(exp)`
- `stringToInteger(exp)`

And these exp helper functions:

- `char op = getOperator(exp);` //returns ‘+’
- `string left = getLeftExp(exp);` //returns “(1*3)”
- `string right = getRightExp(exp);` //returns “(4+2)”
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(exp)) {
        return stringToInteger(exp);
    } 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 {
            return leftResult * rightResult;
        }
    }
}
```

```
exp = "(((1*3)+((4*5)+2))"
    op = '+'
    left = "((1*3)"
    right = "((4*5)+2)"
    leftResult = 3
    rightResult = 22
```
Helper Methods

Here is the key function behind the helper methods:

```cpp
int getOppIndex(string exp) {
    int parens = 0;
    // ignore first left paren
    for (int i = 1; i < exp.length(); i++) {
        char c = exp[i];
        if (c == '(') {
            parens++;
        } else if (c == ')') {
            parens--;
        }
        if (parens == 0 && (c == '+' || c == '*')) {
            return i;
        }
    }
    return -1;
}
```
Recursion you can see
**Fractal**

**fractal**: A recurring graphical pattern. Smaller instances of the same shape or pattern occur within the pattern itself.
Fractal

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. ...
The Cantor Fractal
Cantor Fractal

Parts of a cantor set image ... are Cantor set images
Cantor Fractal

Start

Another cantor set

End

Also a cantor set
Levels of Cantor

6 levels
Levels of Cantor

1 level
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 $n-1$
Graphics in C++ with the Stanford Libs: GPoint

GWindow w;
GPoint a(100, 100);
cout << a.getX() << endl;
Graphics in C++ with the Stanford Libs: GPoint

```
GWindow w;
GPoint a(100, 100);
GPoint b(20, 20);
w.drawLine(a, b);
```
Cantor Fractal
Snoflake Fractal
Snowflake Fractal
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)
Another Example On the Website
Fractals
• Fractals are self-referential, and that makes for nice recursion problems!
• Break the problem into a smaller, self-similar part, and don't forget your base case!
References and Advanced Reading

• **References:**
  - [http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html](http://www.cs.utah.edu/~germain/PPS/Topics/recursion.html)
  - Why is iteration generally better than recursion? [http://stackoverflow.com/a/3093/561677](http://stackoverflow.com/a/3093/561677)

• **Advanced Reading:**
  - Interesting story on the history of recursion in programming languages: [http://goo.gl/P6Einb](http://goo.gl/P6Einb)