

Recursion

Steven Zeil

November 25, 2013

Contents

1	Recursion	2
2	Example: Compressing a Picture	3
3	Example: Calculator	4

1 Recursion

Recursion

A function is *recursive* if it calls itself or calls some other function that eventually causes it to be called again.

- Recursion, like iteration (looping) is a way to process multiple pieces of data.
- Recursion and iteration are equally powerful
 - but some problems are easier to solve with one and some with the other

.....

A Pattern for Recursive Algorithms

Recursive functions tend to fall into certain familiar patterns.

- Test the input parameters to see if they represent a special case simple enough to be solved without recursion.
 - These called *base cases*.
- When this is not a base case, the recursive function
 - break apart the problem into one or more smaller sub-problems.
 - Solve the sub-problems via recursive calls.
 - Combine the sub-problem solutions into a solution for the original problem

.....

A Silly Example of Recursion

```
template <class T>
int listLength (LinkedListNode<T>* list)
{
    if (list == NULL) // list is empty: base case
        return 0;    // base case solution
    else
    {
        int rest = listLength (list->next); // smaller
                                           // subproblem
        return 1 + rest; // combine into total solution
    }
}
```

Recursion

Why silly?

- An iterative form would be simpler and faster.

.....

How to Get In Trouble With Recursion

- Forget to check for a base class
- Break into subproblems that are bigger or the same size as the current problem
- Break into subproblems that are different kinds of problems

.....

When to Think 'Recursion!'

- Look for a problem or a data structure that has "pieces" that are the same "kind" as the whole. E.g.,
 - organization charts
 - mathematical expressions
 - large images

.....

2 Example: Compressing a Picture

Example: Compressing a Picture

Idea: some styles of picture have large areas of uniform color

- Compress by finding rectangles of uniform color

.....

Compression Code

```
void compress(ostream& out, const Image& img,
             int x, int y, int w, int h)
{
    if (w == 0 || h == 0)
        return;
    if (allOneColor(img, x, y, w, h))
    {
        Color c = img.getColor(x,y);
        out << c << ":" << x << ":" << y
            << ":" << w << ":" << h << endl;
    }
    else
        if (w > h)
        { // split horizontally
            compress (out, img, x, y, w/2, h);
            compress (out, img, x+w/2, y, w/2, h);
        }
        else
        { // split vertically
            compress (out, img, x, y, w, h/2);
            compress (out, img, x, y+h/2, w, h/2);
        }
}
```

An iterative form is possible, but would be much more complicated.

.....

3 Example: Calculator

Example: Calculator

Consider a calculator program to evaluate expressions like $1.5 + (2 / 3)$

- We'll simplify the input by assuming each token is separated by blanks

.....

Main Program

```
int main(int argc, char** argv)
{
    LListHeader<string> tokens;
    string token;
    while (cin >> token)
```

Recursion

```
tokens.addToEnd (token);  
cout << evaluate (tokens) << endl;  
return 0;  
}
```

- The calculator itself is in `evaluate()`

.....

How Do We Evaluate An Expression?

- An expression is a sum (or difference) of one or more products
- A product is formed by multiplying (or dividing) one or more terms
- A term is either a number or a parenthesized expression
 - What's inside a parenthesized expression?
 - * An expression!
 - * A "natural" recursion

.....

Example: $2 / (1 + 3)$

$$2 / (1 + 3)$$

This expression is a sum of one product
The product is the division of two terms

- The first term is a number (2)
- The second term is a parenthesized expression
 - The expression $1 + 3$ is the sum of two products
 - * The first product is a (multiplication of) one term
That term is a number (1)
 - * The second product is one term
That term is a number (3)

.....

Starting the Evaluation

```
double expression (NodePtr& input);
double product (NodePtr& input);
double term (NodePtr& input);

double evaluate (LListHeader<string>& tokens)
{
    LListNode<string>* input = tokens.first;
    return (expression(input));
}
```

.....

Evaluating an Expression

```
double expression (NodePtr& input)
{
    double sum = product(input);
    while (input != 0 &&
           (input->data == "+" || input->data == "-"))
    {
        string op = input->data;
        input = input->next;
        assert (input != 0);
        double value = product(input);
        if (op == "+")
            sum += value;
        else
            sum -= value;
    }
    return sum;
}
```

.....

Evaluating a Product

```
double product (NodePtr& input)
{
    double result = term(input);
    while (input != 0 &&
           (input->data == "*" || input->data == "/"))
    {
        string op = input->data;
        input = input->next;
    }
}
```

Recursion

```
    assert (input != 0);
    double value = term(input);
    if (op == "*")
        result *= value;
    else
        result /= value;
}
return result;
}
```

.....

Evaluating a Term

```
double term (NodePtr& input)
{
    if (input->data == "(")
    {
        input = input->next;
        assert (input != 0);
        double result = expression(input);
        assert (input->data == ")");
        input = input->next;
        return result;
    }
    else
    {
        double result = atof(input->data.c_str());
        input = input->next;
        return result;
    }
}
```

.....

Example: evaluating 2 / (1 + 3)

```
expression:      2 / ( 1 + 3 )
product:        2 / ( 1 + 3 )
term :          2 / ( 1 + 3 )
    returns 2.0  / ( 1 + 3 )
sees /         ( 1 + 3 )
term:          ( 1 + 3 )
    sees (      1 + 3 )
expression:    1 + 3 )
```

Recursion

```
product:      1 + 3 )
  term:       1 + 3 )
    returns 1.0 + 3 )
    returns 1.0 + 3 )
  sees +      3 )
product:      3 )
  term:       3 )
    returns 3.0 )
    returns 3.0 )
  return 4.0  )
sees )
returns 4.0
returns 0.5
returns 0.5
```

.....

Try It!

- Run this program [here](#).
- How would you do this without recursion?"
 - Would be far more complicated

.....