CS 161, Lecture 13: Recursion

YOUR PARTY ENTERS THE TAVERN.

I GATHER EVERYONE AROUND A TABLE. I HAVE THE ELVES START WHITTLING DICE AND GET OUT SOME PARCHMENT FOR CHARACTER SHEETS.

HEY, NO RECURSING.
What is Recursion?

• When a function calls itself one or more times (directly or indirectly)
• Form of repetition
• Typically used to perform same operation on a smaller subset and then build the result based on what is returned from the smaller case
• Typically has at least one base case for stopping
• Based on inductive logic
Iteration vs. Recursion

• Anything that can be done iteratively can be done recursively and vice versa
  • Not always a good idea, some problems naturally lend themselves to one mode of thinking or the other
How it works on a high level

```
summation(listOfNumbers[0…n])
  if n == 0
    return listOfNumbers[0]
  return listOfNumbers[0] + summation(listOfNumbers[1…n])
```

```
summation([1,2,3])
  if 2 == 0
    return listOfNumbers[0]
  return 1 + summation([2,3])
```

```
  if 1 == 0
    return listOfNumbers[0]
  return 2 + summation([3])
```

```
    if 0 == 0
      return 3
    return listOfNumbers[0] + summation(listOfNumbers[1…n])
```

```
5
```

```
6
```

```
3
```
Pros and Cons

• Pros
  • Readable
  • Sometimes easier to conceptualize for problems that have many moving parts

• Cons
  • Efficiency
  • Memory usage
    • Each call to the function makes a new function stack frame (see previous slide)
Example: Factorial

• The product of an integer and all that come before it
• \( n! = n \times (n-1) \times (n-2) \times \ldots \times (n-(n-1)) \times 1 \) for all \( n > 0 \)
• Base Case: \( 0! = 1 \)
Iterative Factorial

\[ n = 4 \]

```c
int factorial(int n) {
    int fact;
    if (n == 0) {
        fact = 1;
    } else {
        for (fact = n; n > 1; n--)
            fact = fact * (n-1);
    }
    return fact;
}
```
Recursive Factorial

```c
int factorial (int n) {
    if (n == 0)
        return 1;
    return n * factorial(n-1);
}
```
#include <iostream>

using namespace std;

int fact_itr(int n) {
    cout << "Entered the function for n = " << n << endl;
    int fact;
    if (n==0)
        fact = 1;
    else{
        for(fact = n; n > 1; n--){
            fact = fact * (n-1);
            cout << "Value in for loop: " << fact << endl;
        }
    }
    return fact;
}

int fact_rec(int n) {
    cout << "Function made for n = " << n << endl;
    if (n == 0)
        return 1;
    int fact = n*fact_rec(n-1);
    cout << "Returning " << fact << endl;
}
10     else{
11         for(fact = n; n > 1; n--){
12             fact = fact * (n-1);
13             cout << "Value in for loop: " << fact << endl;
14         }
15     }
16     return fact;
17 }
18
19 int fact_rec(int n) {
20     cout << "Function made for n = " << n << endl;
21     if (n == 0)
22         return 1;
23     int fact = n*fact_rec(n-1);
24     cout << "Returning " << fact << endl;
25     return fact;
26 }
27
28 int main() {
29
30     cout << "Fact iter: " << fact_itr(4) << endl;
31     cout << "Fact rec: " << fact_rec(4) << endl;
32     return 0;
33 }
Exercise: What is the recursive solution to produce the following design?

```
if (n == 1) {
    cout << "*" << endl;
}

func (n, offset) {
    for (i = 0; i < n; i++) {
        for (j = 0; j < offset; j++) {
            cout << " ";
        }
        for (j = 0; j < n - offset; j++) {
            cout << "*");
        }
        cout << endl;
    }
}
```
print (stars, col)
for # of cols
   cout << " "
for # of stars
   cout << "* 
   (cout << endl)
if (stars == 2)
   print (stars, col)
else
```cpp
#include <iostream>

using namespace std;

void print(int stars, int col) {
    for(int i=0; i<col; i++)
        cout << " ";
    for(int i=0; i<stars; i++)
        cout << "*";
    cout << endl;
}

void pattern(int stars, int col) {
    if (stars == 2){
        print((stars/2), col);
        cout << "Top if 2" << endl;
    } else{
        cout << "Top else" << endl;
        pattern((stars/2), col);
        cout << "returned" << endl;
    }
    cout << "Mid" << endl;
    print(stars, col);
```
void pattern(int stars, int col) {
    if (stars == 2) {
        print((stars/2), col);
        cout << "Top if 2" << endl;
    }
    else {
        cout << "Top else" << endl;
        pattern((stars/2), col);
        cout << "returned" << endl;
    }
    cout << "Mid" << endl;
    print(stars, col);
    if (stars == 2) {
        cout << "Bottom if 2" << endl;
        print((stars/2), col+(stars/2));
    }
    else {
        cout << "Bottom else" << endl;
        pattern((stars/2), col+(stars/2));
        cout << "returned" << endl;
    }
}
22  }
23  cout << "Mid" << endl;
24  print(stars, col);
25  if (stars == 2){
26      cout << "Bottom if 2" << endl;
27      print((stars/2), col+(stars/2));
28  }
29  else{
30      cout << "Bottom else" << endl;
31      pattern((stars/2), col+(stars/2));
32      cout << "returned" << endl;
33  }
34 }
35
36 int main () {
37      pattern(8, 0);
38      return 0;
39  }