CSCI 262 Lecture 8 – Recursion, part 1

Outline

- Recursion: a function calling itself
- Two important components:
  - Base case – stops the recursion
  - Recursive call – must reduce the problem
- Recursion and the function call stack
  - Each recursive call gets an entry pushed on the stack – arguments move down
  - Return values pass up as we "unwind" the stack

Readings

Google recursion!

Also, anything about fractals

Explore the GCD algorithm more in depth: https://en.wikipedia.org/wiki/Euclidean_algorithm

Self Check

1. What is an example of a recursive mathematical algorithm?
2. What happens if a recursive function lacks a base case?
3. How does the `factorial` function we explored reduce the problem in its recursive call? In what sense is the problem smaller than before?
4. What kinds of things are stored on the function call stack?

For Further Practice

1. Implement a solution for the DigitSum problem (APT 0) using recursion instead of a loop. You could also try the Hailstone problem (APT 1).
2. Pascal's triangle (https://en.wikipedia.org/wiki/Pascal%27s_triangle) is closely related to the binomial coefficient we mentioned in lecture. The top of the triangle contains a single 1. At each subsequent level of the triangle, the number of entries increases by 1. The first and last entries of a level are always 1; every other entry is the sum of the two entries in the line above that bracket the entry. Make a recursive program to print out Pascal's triangle to a specified height (a lopsided triangle is fine!) Here is an example output for height = 7:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

Stack

```
stack<string> s;
s.push("Apple");
s.push("Banana");
s.push("Pear");
s.pop();
s.pop();
s.push("Pineapple");
s.pop();
s.push("Orange");
cout << s.top() << endl;
```

Queue

```
queue<string> q;
q.enqueue("Apple");
q.enqueue("Banana");
q.enqueue("Pear");
q.dequeue();
q.dequeue();
q.enqueue("Pineapple");
q.dequeue();
q.enqueue("Orange");
cout << q.front() << endl;
```

Big O

Simplify the following

- $\mathcal{O}(n^2 + 2n + 4)$
- $\mathcal{O}(5n^2 + 8n^3)$
- $\mathcal{O}(2^n + 3^n)$
- $\mathcal{O}(n! + 1)$

What is the complexity of the following function?

```
void some_function(vector<int> &vec) {
    int n = vec.size();
    for (int left = 0; left < n; left++) {
        int right = left;
        for (int j = left + 1; j < n; j++) {
            if (vec[j] < vec[right]) right = j;
        }
        swap(vec[left], vec[right]);
    }
}
```