Week 2 Section Solutions

---

1. RecursionMysteryComma - Recursion Trace

	Output:
`recursionMysteryComma(4, 1);`	4
`recursionMysteryComma(4, 2);`	8, 4, 8
`recursionMysteryComma(8, 2);`	16, 8, 16
`recursionMysteryComma(4, 3);`	12, 8, 4, 8, 12
`recursionMysteryComma(3, 4);`	12, 9, 6, 3, 6, 9, 12

---

2. Rarest - Map

```
// Map-based, concise solution
string rarest(const Map<string, string>& map) {
      if (map.isEmpty()) {
      throw "Empty map";
      }

      Map<string, int> counts;
      for (string key : map) {
          counts[map[key]]++;
      }

      string result = "";
      for (string s : counts) {
          if (result == "" || counts.get(s) < counts.get(result)) {
              result = s;   // because of Map (not HashMap), don't need
to break ties
          }
      }
      return result;
}
```

Thanks to Marty Stepp and other CS106B and X instructors and TAs for contributing problems on this handout.

3. biggestFamily - Map, File I/O

```
void biggestFamily(string filename) {
    ifstream input;
    input.open(filename);
    if (input.fail()) {
        throw "No input data.";
    }

    // read file data into map
    int bestCount = 0;
    Map<string, Set<string> > houses;
    string line;
    while (getline(input, line)) {
        Vector<string> tokens = stringSplit(line, " ");
        string first = tokens[0];
        string last = tokens[1];
        houses[last].add(first);
        bestCount = max(bestCount, houses[last].size());
    }

    // find biggest family/ies
    for (string last : houses) {
        if (houses[last].size() == bestCount) {
            cout << last << " family:";
            for (string first : houses[last]) {
                cout << " " << first;
            }
            cout << endl;
        }
    }
}
```

---

4. isHappyNumber - Set

```
int sumOfSquares(int n) {
    int sum = 0;
    while (n > 0) {
        int lastDigit = n % 10;
        sum += lastDigit * lastDigit;
        n = n / 10;
    }
    return sum;
}

bool isHappyNumber(int n) {
    Set<int> seen;
    while (true) {
        int sum = sumOfSquares(n);
        if (seen.contains(sum)) {
            return false;
        }
        if (sum == 1) {
            break;
        } else {
            n = sum;
            seen.add(sum);
        }
    }
    return true;
}
```

---

5. stutterStack - Recursion

```
void stutterStack(Stack<int>& s) {
    if (!s.isEmpty()) {
        int n = s.pop();
        stutterStack(s);
        s.push(n);
        s.push(n);
    }
}
```

---

6. reverseLines - Recursion, File I/O

```
void reverseLines(ifstream& input) {
   string line;
   if (getline(input, line)) {
       // recursive case
       reverseLines(input);
       cout << line << endl;
   } // else implicit base case, do nothing
}
```

---

7. evaluateMathExpression - Recursion

```
int evaluate(const string& s, int& i) {
   if (i >= s.length()) {
       return 0;
   }
   char ch = s[i++];
   if (isdigit(ch)) {
       return ch - '0';
   } else {
       // ch == '('
       int left = evaluate(s, i);
       char op = s[i++];
       int right = evaluate(s, i);
       i++;   // ')'
       if (op == '+') {
           return left + right;
       } else {
           return left * right;   // op == '*'
       }
   }
}
int evaluateMathExpression(const string& s) {
   int i = 0;
   return evaluate(s, i);
}
```
