CS106B Chris Gregg
Summer 2017

Section Handout #3 Solutions

If you have any questions about the solutions to the problems in this handout, feel free to reach out to your section leader, Jason, or Chris for more information.

1. Tracing a Mystery

Call	Output
`mystery1(4, 1)`	`4`
`mystery1(8, 2)`	`16, 8, 16`
`mystery1(3, 4)`	`12, 9, 6, 3, 6, 9, 12`

2. Sum of Squares

```
int sumOfSquares(int n) {
  if (n == 0) return 0;
  return n * n + sumOfSquares(n – 1);
}
```

3. Reverse

```
string reverse(string &s) {
  if (s == "") return "";
  return reverse(s.substr(1)) + s[0];
}
```

4. Star String

There are multiple ways to handle invalid input. In our solution below, we chose to throw an exception. However, you can also have a well-defined value that you return in the event of invalid input. It's important that whatever you choose, you document it well.

```
string starString(int n) {
  if (n < 0) {
    throw "Invalid input.";
  } else if (n == 0) {
    return "*";
  } else {
    string stars = starString(n - 1);
    return stars + stars;
  }
}
```

Note that another possible solution is to return the result `starString(n – 1) + starString(n – 1)`. Why might you not want to use this solution?

5. Subsequence

```
bool isSubsequence(string &big, string &small) {
  if (small == "") {
    return true;
```

```
  } else if (big == "") {
    return false;
  } else {
    if (big[0] == small[0]) {
      return isSubsequence(big.substr(1), small.substr(1));
    } else {
      return isSubsequence(big.substr(1), small);
    }
  }
}
```

6. Make Change

```
void makeChangeHelper(int amount, Vector<int> &coins, Vector<int> &chosen) {
  if (coins.isEmpty()) {
    if (amount == 0) {
      cout << chosen << endl;
    }
  } else {
    int coin = coins[0];
    coins.remove(0);                                  // choose a coin
    for (int i = 0; i <= (amount / coin); i++) {    // explore all quantities of this coin
      chosen += i;
      makeChangeHelper(amount - (i * coin), coins, chosen);
      chosen.remove(chosen.size() - 1);
    }
    coins.insert(0, coin);                            // un-choose a coin
  }
}

void makeChange(int amount, Vector<int> &coins) {
  Vector<int> chosen;
  makeChangeHelper(amount, coins, chosen);
}
```

7. Print Squares

```
void printSquaresHelper(int n, int min, Set<int> &chosen) {
  if (n < 0) {
    return;
  } else if (n == 0) {
    cout << chosen << endl;
  } else {
    int max = (int) sqrt(n); // valid choices go up to sqrt(n)
    for (int i = min; i <= max; i++) {
      chosen.add(i);                                  // choose
      printSquaresHelper(n - (i * i), i + 1, chosen);   // explore
      chosen.remove(i);                               // un-choose
    }
  }
}

void printSquares(int n) {
  Set<int> chosen;
  printSquaresHelper(n, 1, chosen);
}
```

8. Longest Common Subsequence

```
string longestCommonSubsequence(string &s1, string &s2) {
  if (s1.length() == 0 || s2.length() == 0) {
    return "";
  } else if (s1[0] == s2[0]) {
    return s1[0] + longestCommonSubsequence(s1.substr(1), s2.substr(1));
  } else {
    string choice1 = longestCommonSubsequence(s1, s2.substr(1));
    string choice2 = longestCommonSubsequence(s1.substr(1), s2);
    if (choice1.length() >= choice2.length()) {
      return choice1;
    } else {
      return choice2;
    }
  }
}
```

9. Ways to Climb

```
void waysToClimbHelper(int stairs, Stack<int> &chosen) {
  if (stairs < 0) {
    return
  } else if (stairs == 0) {
    cout << chosen << endl;
  } else {
    chosen.push(1);                         // choose 1
    waysToClimbHelper(stairs - 1, chosen); // explore
    chosen.pop();                           // un-choose

    chosen.push(2);                         // choose 2
    waysToClimbHelper(stairs - 2, chosen); // explore
    chosen.pop();                           // un-choose
  }
}


void waysToClimb(int stairs) {
  Stack<int> chosen;
  waysToClimbHelper(stairs, chosen);
}
```

10. Twiddle

```
void listTwiddlesHelper(const string& prefix, const string& str, int index,
        const Lexicon& lex) {

    if (!lex.containsPrefix(prefix)) {
        return;   // optimization; not strictly necessary but good to do
    }

    if (index >= str.size()) {
        if (lex.contains(prefix)) {
            cout << prefix << endl;
        }
    } else {
        for (char ch = str[index] - 2; ch <= str[index] + 2; ch++) {
            if (isalpha(ch)) {
                listTwiddlesHelper(prefix + ch, str, index + 1, lex);
            }
```

```
        }
    }
}

void listTwiddles(const string& str, const Lexicon& lex) {
    string prefix = "";
    listTwiddlesHelper(prefix, str, /* index */ 0, lex);
}
```

11. Hacking and Cracking

```
string findPassword(string soFar, int maxLength) {
    if (login(soFar)) return soFar;
    if (soFar.size() == maxLength) return "";

    for (char c = 'a'; c <= 'z'; c++) {
        if (findPassword(soFar + c, maxLength) != "") {
            return password;
        }
    }

    for (char c= 'A'; c <= 'Z'; c++) {
        if (findPassword(soFar + c, maxLength) != "") {
            return password;
        }
    }
    return "";
}

string crack(int maxLength) {
    if (maxLength < 0) {
        throw maxLength;
    } else if (maxLength == 0) {
        return "";
    }

    return findPassword("", maxLength);
}
```