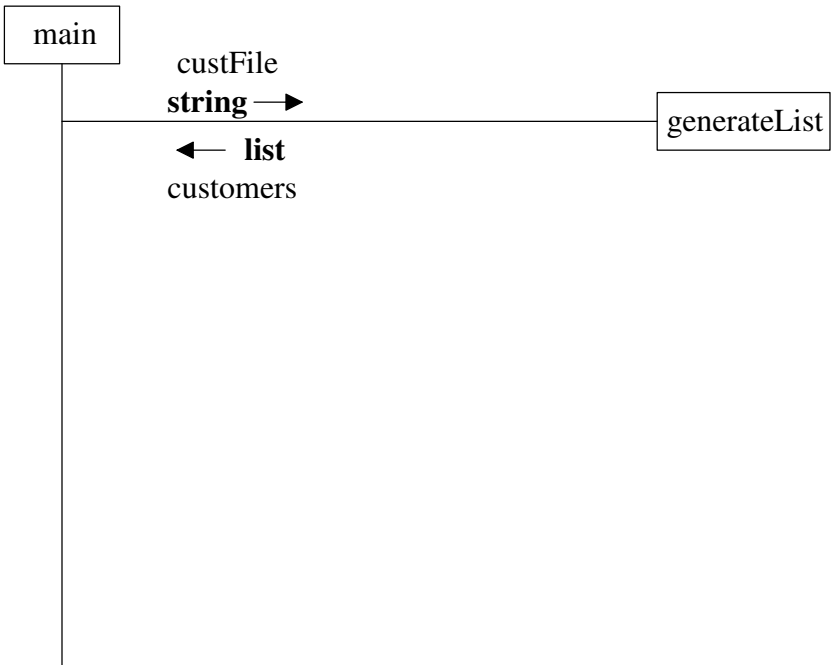
1. Last class we had about completed a function for the problem: "Write a function that takes the list-of-customer-field-lists from the previous lecture (from processing customerData.txt file), and generates a dictionary with a tally of the number of customers from each state. Then, write another function that takes this dictionary as a parameter and prints the number of customers from each state **sorted from most to least**."

```
def main():
    """ Open's file, reads customer information into a list, closes the file"""
    custFile = open('customerData.txt','r')
    customerList = generateList(custFile)
    custFile.close()
    statesCountDictionary = generateStatesCountDictionary(customerList)
    printSortedByState(statesCountDictionary)
    printSortedByCount(statesCountDictionary)

def generateList(custFile):
    """ Reads customer data from file and returns a list of customers"""
    customers = []
    for line in custFile:
        custInfoList = line.strip().split(',')
        customers.append(custInfoList)
    return customers

def generateStatesCountDictionary(customerList):
    """ Tallies the number of customers from each state and returns a dictionary
        with the state as a key and the count as the associated value."""
    statesCountDict = {}
    for customer in customerList:
        if customer[5] in statesCountDict:
            count = statesCountDict.get(customer[5])
            count = count + 1
            statesCountDict[customer[5]] = count
        else:
            statesCountDict[customer[5]] = 1
    return statesCountDict

def printSortedByState(statesCountDictionary):
    """ Prints the tally of the number of customers from each state
        sorted alphabetically by the state abbreviation."""
    tupleList = statesCountDictionary.items()
    tupleList.sort()
    print "\n\n%5s   %-14s" % ("State","Customer Count")
    print "-"*25
    for item in tupleList:
        state, count = item
        print "%5s   %8d" % (state.center(5), count)

def printSortedByCount(statesCountDictionary):
    """ Prints the tally of the number of customers from each state sorted by count."""
```

2. Complete the structure chart for the program on the previous page.



3. A *recursive function* is one that calls itself. The following countDown function is passed a starting value and proceeds to count down to zero and prints "Blast Off!!!".

```
"""
File:  countDown.py
Author:  Mark Fienup
Description:  Demonstrates a simple recursive function that takes a
specific integer and counts down to 1 before printing "Blast Off!!!"""

def main():
    start = input("Enter count down start: ")
    print "\nCount Down:"
    countDown(start)

def countDown(count):
    if count == 0:
        print "Blast Off!!!"
    else:
        print count
        countDown(count - 1)

main()
```

Program Output:

```
Enter count down start: 10

Count Down:
10
9
8
7
6
5
4
3
2
1
Blast Off!!!
```

The `countDown` function, like most recursive functions, solves a problem by splitting the problem into one or more simpler problems of the same type. For example, `countDown(10)` prints the first value (i.e, 10) and then solves the simpler problem of counting down from 9. To prevent "infinite recursion", if-statement(s) are used to check for trivial *base case*(s) of the problem that can be solved without recursion. Here, when we reach a `countDown(0)` problem we can just print "Blast Off!!!".

a) Trace the function call `countDown(5)` on paper by drawing the run-time stack and showing the output.

b) What do you think will happen if your call `countDown(-1)`?

c) Why is there a limit on the depth of recursion?

4. Some mathematical concepts are defining by recursive definitions. One example is the Fibonacci series:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

After the second number, each number in the series is the sum of the two previous numbers. The Fibonacci series can be defined recursively as:

$Fib_0 = 0$

$Fib_1 = 1$

$Fib_N = Fib_{N-1} + Fib_{N-2}$ for $N \geq 2$.

a) Write the recursive function

b) Draw a recursion tree for fib(5).

c) On my office computer, the call to fib(40) takes 22 seconds, the call to fib(41) takes 35 seconds, and the call to fib(42) takes 56 seconds. How long would you expect fib(43) to take?

d) How long would you guess calculating fib(100) would take on my office computer?

e) Why do you suppose this recursive fib function is so slow?

f) How might we speed up the calculation of the Fibonacci series?