

# Test 2: CPS 006L.1

Owen Astrachan

November 13, 2001

Name: \_\_\_\_\_

Login: \_\_\_\_\_

Honor code acknowledgment (signature) \_\_\_\_\_

	value	grade
Problem 1	20 pts.	
Problem 2	15 pts.	
Problem 3	15 pts.	
Problem 4	10 pts.	
TOTAL:	60 pts.	

This test has 8 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 50 minutes.

In writing code you do not need to worry about specifying the proper `#include` header files. Assume that all the header files we've discussed are included in any code you write.

**PROBLEM 1 :** (*Search Me? 20 points*)

**Part A (8 points)**

A struct `Student` stores a student name and social security number.

```
struct Student
{
    string myName;    // student name
    string mySSN;    // student social security number
    Student()
    { }

    Student(const string& name, const string& ssn)
        : myName(name), mySSN(ssn)
    { }
};
```

Write the function `GetSocial` specified below that returns the social security number for the student whose name is `targetName`, and returns "000-00-0000" if no student with the name `targetName` is found in `list`.

```
string GetSocial(const tvector<Student>& list, const string& targetName)
// pre: list contains list.size() elements
// post: returns social security number of student stored in list whose name is targetName
//        but returns "000-00-0000" if no such student found in list
{
```

```
}
```

### Part B (12 points)

You're given a file in the format below, where each line contains a social security number in the format `xxx-xx-xxxx`, followed by one space, followed by a name as shown.

```
123-45-6789 Billy-Bob Wingnut
111-22-3345 Ethel Mertz
666-11-0128 Mary Mary Quite Contrary
890-01-1024 Peter Peter Pumpkin-Eater
```

Write the function `ReadFile` that reads a file in this format, storing all the information read in a vector of `Student` structs. Store social security numbers as strings in the same `"xxx-xx-xxxx"` format as they appear in the file.

```
void ReadFile(tvector<Student>& list, ifstream& input)
// pre: input is open for reading, in proper format
// post: all information in input read, stored in list,
//       list contains list.size() elements
{
```

```
}
```

**PROBLEM 2 :** (*Mean Streets: 15 points*)

A file contains test scores, one-per-line, each score is in the range 0-100 (including both 0 and 100). For example:

```
0
12
89
...
```

Write the function `Stats` that reads such a file and prints both the mean/average and the mode/most-frequently occurring value. The mean is calculated by adding all the scores and dividing by the number of scores. The mode is the value that occurs more frequently than any other value. Assume that the mode is unique in writing your solution. The output should include information as follows:

```
mean = 57.92
mode = 75 occurring 29 times
```

Note: the statement `tvector<string> list(57,"cat")` creates a vector of 57 strings, all equal to "cat", you may want to create a vector in writing your code.

```
void Stats(istream& input)
// pre: input is open for reading, contains one integer value
//       per line, each in the range [0..100]
// post: prints mean and mode of values read from input
{
    int num;

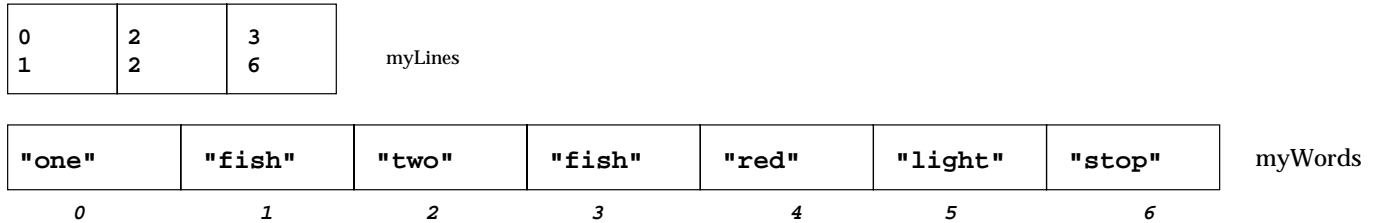
    while (input >> num)    // read a score, process it
    {
```

**PROBLEM 3 :** (*Get in Line (15 points)*)

The class `LineCollection` stores information representing words and lines in a file. For example, consider the file below.

```
one fish
two
fish red light stop
```

The diagram that follows shows how words and lines are stored in the vectors `myWords` and `myLines` (see the class declaration below).



The idea is that all words are stored in `myWords`, and a vector of `Line` structs holds information about where each line is, with the first and last index of the words on each line stored in a `Line` struct. For example, the third line of the file is represented by the third `Line` struct shown in the diagram as (3,6) since the words with indexes 3,4,5,6 in `myWords` represent the words on the third line of the file.

The header for the class `LineCollection` and the struct `Line` are shown below. A function to read a file is shown on the next page as are two member functions you must write.

```
struct Line
{
    int myFirstIndex; // index in myWords for first word on a line
    int myLastIndex; // index in myWords for last word on a line
    Line();
    Line(int first, int last);
};

class LineCollection
{
public:
    LineCollection();
    void open(const string& filename);

    void printLine(int lineNumber) const; // print words on specified line
    void printLineFromIndex(int index) const; // print words on line with index-th word

private:
    tvector<Line> myLines;
    tvector<string> myWords;
};
```

The member function `LineCollection::open` reads a file storing information in `myWords` and `myLines` as shown. It is correct.

```
void LineCollection::open(const string& filename)
// pre: myWords.size() == 0 myLines.size() == 0
// post: reads filename, stores information in myWords and myLines
{
    ifstream input(filename.c_str());
    string line, word;
    int first, last;

    while (getline(input,line))
    {
        istringstream linestream(line.c_str());
        first = myWords.size();    // index of first word on current line

        while (linestream >> word)
        {
            myWords.push_back(word);
        }
        last = myWords.size()-1;    // index of last word on current line
        myLines.push_back(Line(first,last));
    }
}
```

#### Part A 6 points

Write the function `LineCollection::printLine` specified below. For example, after reading the file shown on the previous page into a variable `lines`, the call `lines.printLine(2)` will print

```
fish red light stop
```

and the call `lines.printLine(1)` will print the single word "two" on a line by itself.

```
void LineCollection::printLine(int lineNumber) const
// pre: 0 <= lineNumber and lineNumber < myLines.size()
// post: prints on one line all the words on the line specified by lineNumber
{

}
}
```

(continued)

## Part B 9 points

Write the function `LineCollection::printLineFromIndex` specified below. For example, the calls `lines.printLineFromIndex(3)` and `lines.printLineFromIndex(5)` both print the line below.

```
fish red light stop
```

The call `lines.printLineFromIndex(0)` would print the line

```
one fish
```

Since the word whose index is zero is on the first line of the file.

In writing `printLineFromIndex` you can call `LineCollection::printLine` from the previous problem, assume it works as specified.

```
void LineCollection::printLineFromIndex(int index) const
// pre: 0 <= index and index < myWords.size()
// post: prints entire line on which the index-th word occurs
```

**PROBLEM 4 : (Born to Run (10 points))**

In this question you're given a function that's correct and then asked to write another function based on the one given to you.

A *run* is a sequence of the same value, e.g., 3,3,3,3 is a run of four 3's, 5,5 is a run of two 5's, and 9 is a run of one 9. The function `runCount` correctly returns the number of runs in a sorted vector. For example, the vector (2,3,3,7,8,8,8,8,9,9,9) contains 5 runs: one 2, two 3's, one 7, four 8's, and three 9's.

```
int runCount(const tvector<int>& a)
// pre: a is sorted, a[0] <= a[1] <= ... <= a[a.size()-1]
// post: returns the number of runs in a
{
    int runVal = a[0];    // the value of the current run
    int count = 1;       // the number of runs so far

    // invariant: runVal is the value of the run ending at a[k-1]

    for(int k=1; k < a.size(); k++) {
        if (a[k] != runVal) {
            count++;
            runVal = a[k];
        }
    }
    return count;
}
```

Write the function `maxRun` that returns the *length* of the longest run in a sorted vector. For example, the length of the longest run in (2,3,3,7,8,8,8,8,9,9,9) is four since the longest run consists of four 8's.

```
int maxRun(const tvector<int>& a)
// pre: a is sorted, a[0] <= a[1] <= ... <= a[a.size()-1]
//       a contains a.size() elements, 0 < a.size()
// post: returns the length of the longest run in a
```