## ***Project 3: Former Airfield? Married for Life! Testing***

There were 20 test cases. Each test was worth 6.5 points; to run the test cases:

For the first 15 test cases:

1. Remove the main routine from your `anagrams.cpp` file.
2. Change the value of your constant MAXDICTWORDS to be 10.
3. Append the following text to the end of your `anagrams.cpp` file and build the resulting program.
4. For any test case you wish to try, run the program, providing as input the test number.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cassert>
#include <algorithm>

using namespace std;

void testone(int n)
{
    string dictionary[MAXDICTWORDS];
    string results[MAXRESULTS];

    switch (n)
    {
        default: {
            cout << "Bad argument" << endl;
        } break; case 1: {
            istringstream
iss("dog\ncat\nrat\neel\ntar\nart\nlee\nact\ngod");
            int numResults = lexiconBuilder(iss, dictionary);
            sort(dictionary, dictionary + numResults);
            assert(numResults == 9 && dictionary[0] == "act" &&
dictionary[1] == "art");
        } break; case 2: {
            // File is empty, checks that file is empty and
lexiconBuilder returns 0.
            istringstream iss("");
            int numResults = lexiconBuilder(iss, dictionary);
            assert(numResults == 0 && dictionary[0] == "");
        } break; case 3: {
            // Dictionary has been read properly
```

```
            istringstream
iss("dog\ncat\nrat\neel\ntar\nart\nlee\nact\ngod");
            int numResults = lexiconBuilder(iss, dictionary);
            sort(dictionary, dictionary + numResults);
            assert(numResults == 9 && dictionary[0] == "act" &&
dictionary[numResults-1] == "tar");
        } break; case 4: {
            // Input file is larger than the dictionary size
            istringstream
iss("dog\ncat\nrat\neel\ntar\nart\nlee\nact\ngod\ntoo\nmany\nwor
ds");
            int numResults = lexiconBuilder(iss, dictionary);
            sort(dictionary, dictionary + numResults);
            assert(numResults == 10 && dictionary[MAXDICTWORDS-
1] == "too");
        } break; case 5: {
            // If a certain word with repeated letter is shown
in results more than once - still accept.
            string dictionary[] = { "one", "oone", "ne", "e",
"too" };
            int numResults = theJumbler("oto", dictionary, 5,
results);
            assert((numResults == 1 || numResults == 2) &&
results[0] == "too");
        } break; case 6: {
            // Doesn't check numResults nor duplicates.
            string dictionary[] = { "one", "oone", "ne", "e",
"too" };
            theJumbler("oto", dictionary, 5, results);
            assert(results[0] == "too");
        } break; case 7: {
            // If word wasn't found, numResults = 0 and results
array is empty.
            string dictionary[] = { "one", "oone", "ne", "e" };
            int numResults = theJumbler("look", dictionary, 4,
results);
            assert(numResults == 0 && results[0] == "" &&
results[1] == "");
        } break; case 8: {
            // No fraction of a permutation is being searched in
dictionary
            string dictionary[] = { "one", "oone", "non", "oon"
};
            int numResults = theJumbler("on", dictionary, 4,
results);
            assert(numResults == 0 && results[0] == "" &&
results[1] == "");
```

```
        } break; case 9: {
            // No fraction of a permutation is being searched in
dictionary
            string dictionary[] = { "one", "oone", "ne", "e",
"neoo", "oneo" };
            int numResults = theJumbler("neo", dictionary, 6,
results);
            assert(numResults == 1 && results[0] == "one" &&
results[1] == "");
        } break; case 10: {
            // Checking that no error occurs if more than
MAXRESULTS are found.
            string dictionary[] = { "true",  "treu", "teru",
"teur", "ture",
             "tuer", "rtue", "rteu", "retu","reut", "ruet",
"rute", "utre",
             "uter", "uetr", "uert", "urte", "uret", "etru",
"etur", "ertu",
             "erut", "eurt", "eutr" };
            // All 24 permutations
            int numResults = theJumbler("true", dictionary, 24,
results);
            assert(numResults == MAXRESULTS);
        } break; case 11: {
            // Checking one word was found, no duplicates.
            string dictionary[] = { "ppp" };
            int numResults = theJumbler("ppp", dictionary, 1,
results);
            assert(numResults == 1 && results[0] == "ppp" &&
results[1] == "");
        } break; case 12: {
            string dictionary[] = { "run", "dodge", "break",
"urn", "defeat" };
            int numResults = theJumbler("nru", dictionary, 5,
results);
            sort(results, results + numResults);
            assert(numResults == 2 && results[0] == "run" &&
results[1] == "urn");
        } break; case 13: {
            streambuf* oldCoutStreamBuf = cout.rdbuf();
            ostringstream strCout;
            cout.rdbuf(strCout.rdbuf());
            string results[] = { "cat", "act"};
            divulgeSolutions(results, 2);
            cout.rdbuf(oldCoutStreamBuf);
            string temp = strCout.str();
            bool match1 = temp.find("act") != string::npos;
```

```
            bool match2 = temp.find("cat") != string::npos;
            assert(match1 && match2);
        } break; case 14: {
            istringstream iss ("tier\nrite\nbate\ntire\nttir");
            int numWords = lexiconBuilder(iss, dictionary);
            sort(dictionary, dictionary + numWords);
            assert(numWords == 5 && dictionary[0] == "bate");
            int numResults = theJumbler("tier", dictionary,
numWords, results);
            assert(numResults == 3 && (results[2] == "tire" ||
results[2] == "tier" || results[2] == "rite"));
        } break; case 15: {
            string example[] = { "kool", "moe", "dee" };
            int numResults = theJumbler("look", example, 3,
results);
            assert(numResults == 1 && results[0] == "kool");
        } break;
    }
}

int main()
{
    for (int n = 1; n <= 15; n++) {
        testone(n);
        cout << n << " passed" << endl;
    }

    return 0;
}
```

For the final 5 test cases:

1. Rename wordsMax.txt to be words.txt.
2. Run the test code on the SEASnet server. Make sure that anagrams.cpp and words.txt are in the same directory.
3. Replace your main program with that from testmain.cpp.
4. Run g32 in this fashion: g32 anagrams.cpp -DTEST# (where #, is 1, 2, 3, 4, or 5)

```
#include <iostream>
#include <fstream>
#include <istream>
#include <cstring>
#include <string>
#include <cassert>
using namespace std;

const int MAXRESULTS = 20;    // Max matches that can be found
```

```
const int MAXDICTWORDS = 30000; // Max words that can be read in

int main()
{
	string results[MAXRESULTS];
	string dict[MAXDICTWORDS];
	ifstream dictfile;		// file containing the list of
words
	int nwords;			// number of words read from
dictionary
	string word;

	dictfile.open("words.txt");
	if (!dictfile) {
		cout << "File not found!" << endl;
		return (1);
	}

	nwords = lexiconBuilder(dictfile, dict);

#ifdef TEST1
	word = "rat";
#endif
#ifdef TEST2
	word = "babe";
#endif
#ifdef TEST3
	word = "plane";
#endif
#ifdef TEST4
	word = "maiden";
#endif
#ifdef TEST5
	word = "arrogant";
#endif

	int numMatches = theJumbler(word, dict, nwords, results);
	if (!numMatches)
		cout << "No matches found" << endl;
	else
		divulgeSolutions(results, numMatches);
#ifdef TEST1
	assert(numMatches == 3 && (results[0] == "rat" ||
results[0] == "art"
		|| results[0] == "tar"));
#endif
#ifdef TEST2
```

```
	assert(numMatches == 2 && (results[0] == "abbe" ||
results[0] == "babe"));
#endif
#ifdef TEST3
	assert(numMatches == 3 && (results[0] == "plane" ||
results[0] == "panel"
		|| results[0] == "penal"));
#endif
#ifdef TEST4
	assert(numMatches == 2 && (results[0] == "maiden" ||
results[0] == "median"));
#endif
#ifdef TEST5
	assert(numMatches == 2 && (results[0] == "arrogant" ||
results[0] == "tarragon"));
#endif

	return 0;
}
```