C++ Programming
(335)

REGIONAL – 2017

Production Portion:

Natural Language Processing: Named Entities	______________	(350 points)
TOTAL POINTS	______________	***(350 points)***

Judge/Graders: Please double check and verify all scores and answer keys!

Natural Language Processing: Named Entities

Have you chatted with Apple Siri, Google Now, Amazon Alexa or Microsoft Cortana? These amazing intelligent assistants employ Natural Language Processing (NLP). This is a leading edge field of computer science and artificial intelligence, concerned with the interactions between computers and human languages. Programmers like you are enabling computers to derive meaning from human or natural language input, as well as generate human language. For this exercise, you will use computer language (C++) to process human language!

1. Write a program that reads written natural language from provided file "human_jabber.txt". Your program will identify paragraphs, sentences and words. Words are separated by spaces, sentences by periods, and paragraphs are delimited by newlines ("\n"). Hint: most punctuation except periods can be discarded.

2. Your program will also read "named_entities.txt". This is a list of proper *nouns* which are often just capitalized words. Use it to identify named entities.

3. Your program will save to "output.csv" what was parsed (example below for format).

4. The program will output a total count to the **screen** of named entities, words, sentences and paragraphs (example below).

5. If the same word or named entity occurs again in the input, count it again. A name like "Paul Bunyan" counts as two named entities.

6. Congratulations! You've processed text in a way that a program like Siri can begin to interpret.

Steps

1. Build a reusable "readFile" function (to read input files), a "parser" function (to identify paragraphs, sentences, words and named entities) and a "writeFile" function to write the output file. Output totals to screen. The program should gracefully handles improper or missing input files, as well as ignore extra whitespace, punctuation and symbols.

2. The program will read files "human_jabber.txt" and "named_entities.txt" and output formatted csv, generated from the data structure.

Sample Input and Output:

1. Here is an example input file `human_jabber.txt`:

```
I am from Minnesota.  Paul Bunyan lives here.
Florida is warmer.  I might move.
Prince was from here so it's cool.
```

2. The file `named_entities.txt` contains:

```
Minnesota
Paul
Bunyan
Prince
Florida
```

3. Example `output.csv` shown. The output contains csv columns for word #, paragraph #, sentence #, type (word or namedEntity), and parsed word.

```
paragraph, sentence, type, word
w1, p1, s1, word, I
w2, p1, s1, word, am
w3, p1, s1, word, from
w4, p1, s1, namedEntity, Minnesota
w5, p1, s2, namedEntity, Paul
w6, p1, s2, namedEntity, Bunyan
w7, p1, s2, word, lives
w8, p1, s2, word, here
w9, p2, s3, namedEntity, Florida
w10, p2, s3, word, is
w11, p2, s3, word, warmer
w12, p2, s4, word, I
w13, p2, s4, word, might
w14, p2, s4, word, move
w15, p3, s5, namedEntity, Prince
w16, p3, s5, word, was
w17, p3, s5, word, from
w18, p3, s5, word, here
w19, p3, s5, word, so
w20, p3, s5, word, it's
w21, p3, s5, word, cool
```

4. The program will output this summary to the screen:

```
Words: 21
Named Entities: 5
Sentences: 5
Paragraphs: 3
```

5. You will have 90 minutes to complete your work.

6. Your name or school name should *NOT* appear on any work you submit for grading.

Development Standards

- Consistent naming should be used for variables and code.
- Classes, methods, and functions must be documented with comments explaining the purpose, the input parameters (if any), and the output (if any).

Your application will be graded on the following criteria:

Solution and Project

Custom code is present	____ 10 points
All classes and methods/functions are customized	____ 10 points

Program Execution

Program runs	____ 20 points

If program does not execute, then remaining items receive *partial credit* if credible code exists.

The program gracefully handles empty, improper or missing input files	____ 10 points
The program reads “human_jabber.txt” into a data structure	____ 15 points
The program reads “named_entities.txt” into a data structure	____ 15 points
The program saves “output.csv” containing dynamically generated csv	____ 15 points
The program outputs correct totals at end	____ 30 points
The “output.csv” correctly counts Words, Paragraphs and Sentences	____ 15 points
The “output.csv” has correct Words identified	____ 15 points
The “output.csv” has Named Entities correctly identified	____ 15 points
The program ignores input “,” and parenthesis and doesn’t add to csv	____ 10 points
The program correctly handles paragraph, sentence and word delimiters	____ 10 points
The program correctly handles (ignores) extra white space	____ 10 points

Source Code Review

Class code is commented, for each method, and as needed	____ 15 points
Code uses reasonable and consistent variable naming conventions	____ 15 points
The program contains well-formed function for readFile	____ 25 points
The program contains well-formed function for parser	____ 25 points
The program contains well-formed function for writeFile	____ 25 points
Processing exists for counting and displaying totals	____ 15 points
The program has punctuation processing	____ 10 points
The program has whitespace processing	____ 10 points
Code exists to trap for file errors	____ 10 points

Total Points: _____/ 350 points

Solution Key Input and Output

Input File `human_jabber.txt`:

```
Apollo was the spaceflight that landed the first humans on the Moon,
Americans Neil Armstrong and Buzz Aldrin, on July 20, 1969.  Armstrong
became the first to step onto the lunar surface six hours later.

Armstrong spent about two and a half hours outside the spacecraft.
Aldrin spent slightly less. Together they collected 47 pounds (21 kg)
of lunar material for return to Earth.

The third member of the mission, Michael Collins, piloted the command
spacecraft alone in lunar orbit until Armstrong and Aldrin returned to
it just under a day later for the trip back to Earth.
```

Input File `named_entities.txt`:

```
Maxwell
Apollo
Neil
Armstrong
Buzz
Aldrin
Michael
Collins
Americans
Moon
Earth
July
Pacific
Toaster
```

Solution Output to stdout:

```
Words: 98
Named Entities: 18
Sentences: 6
Paragraphs: 3
```

Solution Output File `output.csv` (spot check their file for correct named entities)

```
w1, p1, s1, namedEntity, Apollo
w2, p1, s1, word, was
w3, p1, s1, word, the
w4, p1, s1, word, spaceflight
w5, p1, s1, word, that
w6, p1, s1, word, landed
w7, p1, s1, word, the
w8, p1, s1, word, first
w9, p1, s1, word, humans
w10, p1, s1, word, on
w11, p1, s1, word, the
w12, p1, s1, namedEntity, Moon
w13, p1, s1, namedEntity, Americans
w14, p1, s1, namedEntity, Neil
w15, p1, s1, namedEntity, Armstrong
w16, p1, s1, word, and
w17, p1, s1, namedEntity, Buzz
w18, p1, s1, namedEntity, Aldrin
w19, p1, s1, word, on
w20, p1, s1, namedEntity, July
w21, p1, s1, word, 20
w22, p1, s1, word, 1969
w23, p1, s2, namedEntity, Armstrong
w24, p1, s2, word, became
w25, p1, s2, word, the
w26, p1, s2, word, first
w27, p1, s2, word, to
w28, p1, s2, word, step
w29, p1, s2, word, onto
w30, p1, s2, word, the
w31, p1, s2, word, lunar
w32, p1, s2, word, surface
w33, p1, s2, word, six
w34, p1, s2, word, hours
w35, p1, s2, word, later
w36, p2, s3, namedEntity, Armstrong
w37, p2, s3, word, spent
w38, p2, s3, word, about
w39, p2, s3, word, two
w40, p2, s3, word, and
w41, p2, s3, word, a
w42, p2, s3, word, half
w43, p2, s3, word, hours
w44, p2, s3, word, outside
w45, p2, s3, word, the
w46, p2, s3, word, spacecraft
w47, p2, s4, namedEntity, Aldrin
w48, p2, s4, word, spent
w49, p2, s4, word, slightly
w50, p2, s4, word, less
```

```
w51, p2, s5, word, Together
w52, p2, s5, word, they
w53, p2, s5, word, collected
w54, p2, s5, word, 47
w55, p2, s5, word, pounds
w56, p2, s5, word, 21
w57, p2, s5, word, kg
w58, p2, s5, word, of
w59, p2, s5, word, lunar
w60, p2, s5, word, material
w61, p2, s5, word, for
w62, p2, s5, word, return
w63, p2, s5, word, to
w64, p2, s5, namedEntity, Earth
w65, p3, s6, word, The
w66, p3, s6, word, third
w67, p3, s6, word, member
w68, p3, s6, word, of
w69, p3, s6, word, the
w70, p3, s6, word, mission
w71, p3, s6, namedEntity, Michael
w72, p3, s6, namedEntity, Collins
w73, p3, s6, word, piloted
w74, p3, s6, word, the
w75, p3, s6, word, command
w76, p3, s6, word, spacecraft
w77, p3, s6, word, alone
w78, p3, s6, word, in
w79, p3, s6, word, lunar
w80, p3, s6, word, orbit
w81, p3, s6, word, until
w82, p3, s6, namedEntity, Armstrong
w83, p3, s6, word, and
w84, p3, s6, namedEntity, Aldrin
w85, p3, s6, word, returned
w86, p3, s6, word, to
w87, p3, s6, word, it
w88, p3, s6, word, just
w89, p3, s6, word, under
w90, p3, s6, word, a
w91, p3, s6, word, day
w92, p3, s6, word, later
w93, p3, s6, word, for
w94, p3, s6, word, the
w95, p3, s6, word, trip
w96, p3, s6, word, back
w97, p3, s6, word, to
w98, p3, s6, namedEntity, Earth
```

Solution Source Code

```
// Program to perform Natural Language Processing
// Reads file “human_jabber.txt” and identifies
// paragraphs, sentences, words and named entities.
// It uses "named_entities.txt" to identify proper nouns (often capitalized).
// Program outputs parsed words and their identifcation as csv file
// with count at end.

#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <cctype>
#include <string>

class readFile;
class parser;
class writeFile;

// Reusable class to read file and store in buffer.
// Constructor accepts filename.
// Has getter methods to return buffer and size.
class readFile {

    public:
        readFile(std::string);
        std::stringstream& getBuffer();
        int getBufferSize();

    private:
        std::stringstream buffer;
};
// Constructor
readFile::readFile(std::string fileName) {

    std::ifstream file( fileName );

    if ( file )
    {
        buffer << file.rdbuf();
        file.close();
        // operations on the buffer...
    } else {
        std::cout << "File does not exist.";
    }
};
// Get Buffer
std::stringstream& readFile::getBuffer() {
    return buffer;
};
// Get Buffer Size
int readFile::getBufferSize() {
    return buffer.str().size();
};

// This is a parser class to identify
// paragraphs, sentences, words and named entities.
// Constructor accepts input and named entity buffers.
// Has getter methods for parsed output and its size.
class parser {

    public:
        parser(std::stringstream&, std::stringstream&);
        std::stringstream& getOutput();
        int getOutputSize();

    private:
        std::stringstream *output;
};
// Constructor
parser::parser(std::stringstream& in, std::stringstream& namedEntities) {

    std::string word, type;

    std::size_t sentences = 0;
    std::size_t paragraphs = 0;
    std::size_t words = 0;
    std::size_t nes = 0;

    bool in_sentence = false;
    bool in_paragraph = false;

    char token;

    output = new std::stringstream("paragraph, sentence, type, word\n", std::ios_base::app |
std::ios_base::out);

    // Step through each letter
    while (in.get(token))
    {
        if (std::isspace(token) || token == '.')
        {                               // whitespace

            // new paragraph
            if (token == '\n')
                in_paragraph = false;

            // new word
            if (word != "") {
                type = "word";
                ++words;

                if ((namedEntities.str().find("\n"+word+"\n") != std::string::npos) ||
                    (namedEntities.str().find("\n"+word+" ") != std::string::npos)) {
                    type = "namedEntity";
                    ++nes;
                }
                *output << "w" << words << ", p" << paragraphs << ", s" << sentences << ", "
<< type << ", " << word << "\n";
            }
            // new sentence
            if (token == '.') {
                in_sentence = false;
            }
            word = "";
        }
        else
        {   // non-whitespace and alpha numeric
            if (isalnum(token) == 1) {
                word += token;

                if (!in_paragraph)
                {
                    in_paragraph = true;
                    ++paragraphs;
                }
                if (!in_sentence) {
                    in_sentence = true;
                    ++sentences;
                }
            }
        }
    }
    // Display final counts / summary
    std::cout << "\nWords: " << words << "\nNamed Entities: " << nes
            << "\nSentences: " << sentences << "\nParagraphs: " << paragraphs << "\n";

};
// get parsed results
std::stringstream& parser::getOutput() {
    return *output;
};
// get results size
int parser::getOutputSize() {
    return (*output).str().size();
};

// Reusable class to write file.
// Constructor accepts buffer and filename
class writeFile {
    public:
        writeFile(std::stringstream&, std::string);
};
writeFile::writeFile(std::stringstream& output, std::string fileName) {

    std::ofstream file( fileName );

    if (file.is_open())
    {
        file << output.str();
        file.close();
        // operations on the buffer...
    } else {
        std::cout << "Unaable to write file.";
    }
};

// Main
// Reads file "human_jabber.txt" and identifies paragraphs, sentences and words.
// Saves output
int main()
{
    readFile inputFile("human_jabber.txt");

    readFile namedEntities("named_entities.txt");

    if (inputFile.getBufferSize()) {
        parser parsedInput(inputFile.getBuffer(), namedEntities.getBuffer());

        if (parsedInput.getOutputSize() > 0) {
            writeFile outputFile(parsedInput.getOutput(), "output.csv");
        }
    }
}
```