Report for Homework 3

Amirhossein Alian

January 26, 2024

1 First Project

1.1 Code

```cpp
// Amirhossein Alian
// Computer Engineering
// Date: 1402-11-05
// 4021120017
// Project 1

#include <iostream>

using namespace std;

bool isLower(char);
bool isUpper(char);
bool isAlphabet(char);
int strlen(string);
bool haveDot(string);
string strip(string);
bool isMember(string, string[]);
bool isExclusion(string word);

const int lenExclusion = 115;
string exclusion[] = {
        "A", "An", "The", "Am", "Is", "Are", "For", "And", "Nor", "But",
        "Our", "Yet", "So", "As", "If", "From", "For", "About", "But",
        "Or", "And", "On", "At", "In", "Above", "Of", "By", "With",
        "Than", "Around", "Under", "To", "I", "Yot", "He", "She", "It",
        "We", "They", "Me", "You", "Him", "Her", "It", "Us", "Them",
        "Mine", "Yours", "His", "Hers", "Its", "Ours", "Theirs", "Who",
        "Whom", "Which", "That", "Where", "Whose", "This", "That",
        "These", "Those", "Myself", "Yourself", "Himself", "Herself",
        "Itself", "Ourselves", "Themselves", "Can", "Could", "May",
```

```
        "Might", "Will", "Would", "Shall", "Should", "Must", "Ought to",
        "Needn't", "Mustn't", "Had better", "Be", "Have", "Do", "Not",
        "Was", "Were", "Very", "Much", "Many", "Too", "Little", "Few",
        "Such", "Other", "Lot", "Lots", "What", "Who", "How", "When",
        "Your", "There", "Then", "Use", "Has", "Each", "Any", "Thus"
        "He", "His", "One", "My"
};


const int SIZE = 250;
string finals[SIZE];
int f_counter = 0;
string block[SIZE];
int b_counter = 0;
string temp[SIZE];
int t_counter = 0;
string capitals[SIZE];
int c_counter = 0;

// Store Words separately
string text[SIZE];

int length = 0;

int main() {
        // Read Input From file (for Test Stage)
        //freopen("input.txt", "r", stdin);

        cout << "[in-str] Enter Text: ";
        string word;
        int i = 0;
        while (cin >> word) {
                // break with string code "EOF"
                if (word == "EOF")
                        break;
                // add word to text array
                text[i] = word;
                i++;
        }

        length = i;
        /* check for first word
         * (it can not be done on next loop because i-1 will be out of␣
 ↪range) */
```

```
        if (isUpper(text[0][0])) {
                capitals[c_counter++] = text[0];
                temp[t_counter++] = text[0];
        }
        for (int i = 1; i < length; i++) {
                char first_letter = text[i][0];
                if (isUpper(first_letter)) {
                        capitals[c_counter++] = text[i];
                        if (isExclusion(strip(text[i]))) {
                                block[b_counter++] = text[i];
                                        } else {
                                                if (haveDot(text[i-1])) {
                                        temp[t_counter++] = text[i];
                                                } else {
                                                        finals[f_
 ↪counter++] = text[i];
                                                }
                                        }
                }
        }

        for (int i = 0; i < t_counter; i++) {
                if (!isMember(temp[i], finals)) {
                        if (isExclusion(temp[i])) {
                                block[b_counter++] = temp[i];
                        }
                }
        }

    cout << "[out]: \n";
        for (int i = 0; i < c_counter; i++) {
                if (!isMember(capitals[i], block))
                        cout << strip(capitals[i]) << endl;
        }

        return 0;
}

bool isLower(char letter)
{
        return (letter >= 97 && letter <= 122);
}

bool isUpper(char letter)
```

```
{
        return (letter >= 65 && letter <= 90);
}

bool isAlphabet(char letter)
{
        return (isLower(letter) || isUpper(letter));
}

bool haveDot(string word)
{
        int last_char_index = strlen(word) - 1;
        if (word[last_char_index] == '.')
                return true;
        return false;
}

int strlen(string word) {
        int i = 0;
        while (word[i] != '\0')
                i++;
        return i;
}

string strip(string word)
{
        bool reachedAlpha = false;
        int start = 0, end = -1;
        int i = 0;
        while(word[i] != '\0') {
                if (isAlphabet(word[i]) && (!reachedAlpha)) {
                        start = i;
                        reachedAlpha = true;
                }
                if (isAlphabet(word[i]) && reachedAlpha)
                        end = i;
                i++;
        }
        string stripped;
        for (int i = start; i <= end; i++)
                stripped += word[i];
        return stripped;
}
```

```
bool isMember(string word, string list[])
{
        for (int i = 0; i < length; i++)
                if (list[i] == word)
                        return true;
        return false;
}

bool isExclusion(string word)
{
        for (int i = 0; i < lenExclusion; i++)
                if (exclusion[i] == word)
                        return true;
        return false;
}
```

1.2 Input and Output

Test Case 1

```
[in-str] Enter Text: My friend, Zahra is a good student
EOF
[out]:
Zahra
```

Test Case 2

```
[in-str] Enter Text: Sara and I, like pizza
EOF
[out]:
Sara
```

Test Case 3

```
[in-str] Enter Text: Ali went to Tehran yesterday. He has a good friend.␣
 ↪His name is Reza.
EOF
[out]:
Ali
Tehran
Reza
```

1.3 Explanations

```
Our definition of a known word is "a word that starts with a capital␣
 ↪letter"
But this definition has an error and causes some words to be considered␣
 ↪wrongly as defined word.

While they may have started with a capital letter due to the fact that␣
 ↪they are at the beginning of the sentence, and they are not necessarily␣
 ↪defined words.

Therefore, we must use methods to reduce the error and reach the maximum␣
 ↪possible accuracy

In this way, we first consider an array of exception words (words that␣
 ↪may come at the beginning of the sentence and therefore start with a␣
 ↪capital letter while they aren't considered as defined word)

If the word entered by the user is among the words in this array, that␣
 ↪word is stored in an array called "block".

(Finally, when printing words in capital letters, if there is a word in␣
 ↪the "block" array, the program ignores that word and does not print it)

Then, if the word starting with a capital letter was not in this array,␣
 ↪it depends on whether that word is located at the beginning of the␣
 ↪sentence or not at the beginning of the sentence.

If it is located in a place other than the beginning of the sentence,␣
 ↪that word is definitely considered defined and stored in an array␣
 ↪called "final".

And if it is located at the beginning of the sentence, it becomes unclear␣
 ↪for us whether the word is really defined or because it is the␣
 ↪beginning of the sentence, it starts with a capital letter, so we store␣
 ↪that word in the "temp" array.

Then we iterate once over the "temp" array and if the word is not in the␣
 ↪"final" array (that is, there was no other place in the text where that␣
 ↪word started with a capital letter other than the beginning of the␣
 ↪sentence), we add that word to the "block" array so that the program␣
 ↪when printing defined words; Ignore that word too
```

```
Finally, words that passed these filters and were not included in the␣
 ↪"block" array are printed


Functions:
A couple of functions have also been developed for easy and clean work

"strlen" function,
It is a function that receives a string word and returns its length


"strip" function,
It is a function that receives a string word and removes non-English␣
 ↪characters from it, such as .,:-=
(The idea of this function is inspired by Python built-in str functions)


"isUpper" function,
It receives the character and checks whether it is among the characters␣
 ↪A-Z or not

"isLower" function,
It receives the character and checks whether it is among the a-z␣
 ↪characters or not

"isAlphabet" function,
It receives the character and checks whether it is at least a small or an␣
 ↪uppercase English letter


"haveDot" function,
This is to check whether we have reached the end of the sentence or not


"isMember" and "isExclusion" functions,
These functions are written to check the existence of the word in the␣
 ↪array

(isExclusion It is specifically written to check whether an array member␣
 ↪is an exception or not)
```

2 Third Project - Case Four

2.1 Code

```cpp
// Amirhossein Alian
// Computer Engineering
// Date: 1402-11-05
// 4021120017
// Project 3 - Case 4

#include <iostream>

using namespace std;

int main()
{
        // Get N as input
        int n;
        cout << "( Calculate Sum of First N Sequence for E ) ";
        cout << "\nEnter N: ";
        cin >> n;

        // Calculate Factoriels
        int fac[n-1];
        fac[0] = 1;
        for (int i = 1; i < n; i++)
                fac[i] = i * fac[i-1];

        double sum = 0;
        for (int i = 1; i <= n; i++)
                sum += (1) / (double) (fac[i-1]);
        cout << "E: " << sum << endl;

        return 0;
}
```

2.2 Input and Output

Test Case 1

```
( Calculate Sum of First N Sequence for E )
[in-int] Enter N: 1
[out] E: 1
```

Test Case 2

```
( Calculate Sum of First N Sequence for E )
[in-int] Enter N: 2
[out] E: 2
```

Test Case 3

```
( Calculate Sum of First N Sequence for E )
[in-int] Enter N: 3
[out] E: 2.5
```

Test Case 4

```
( Calculate Sum of First N Sequence for E )
[in-int] Enter N: 33
[out] E: 2.71828
```

Test Case 5

```
( Calculate Sum of First N Sequence for E )
[in-int] Enter N: 60
[out] E: inf
```

2.3 Explanations

```
The program receives a number as N

Since we need factorial calculation to calculate this expression

We Must calculate 0! up to (N-1)!

We can use recursive function or normal function to calculate factorials

But for the program to be efficient, we use the array and from 0! We␣
 ↪start and get help from the previous sequence to calculate the next␣
 ↪sequence

For example, to calculate 4! From the result of 3! We will use of what we␣
 ↪calculated before and multiply it by 4 to achieve 4!

Finally, use the for loop to calculate each sequence and find sum for all␣
 ↪in the expansion of e
```