CMP2003 Data Structures and Algorithms (C++)
Project Report

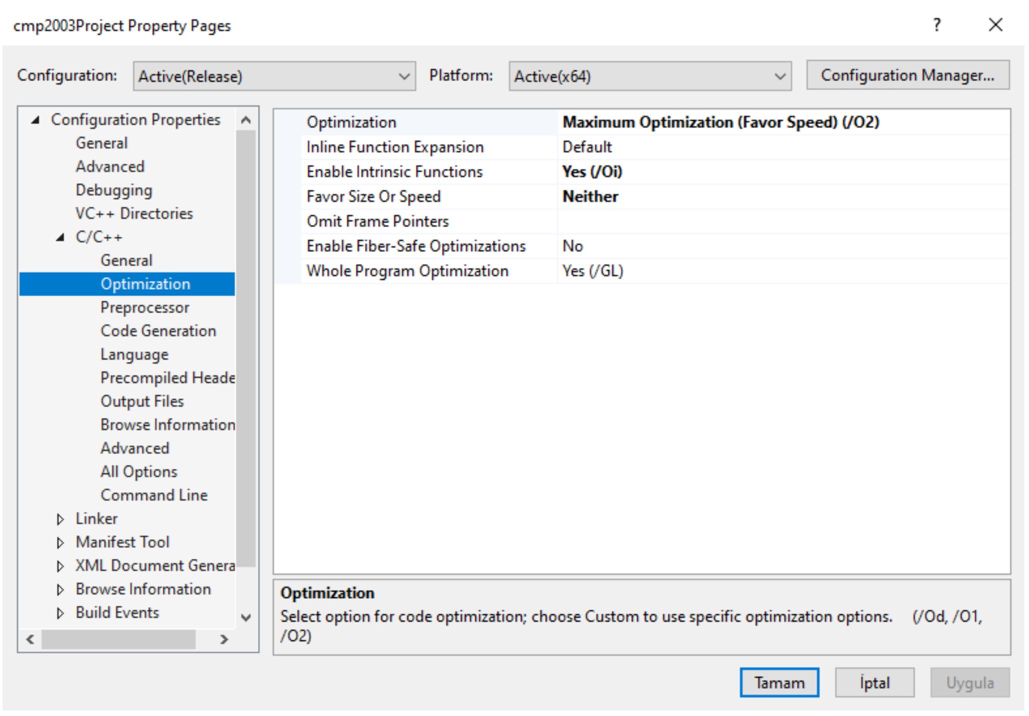
-- Top 10 Frequent Words —

Mücahit Bayram - 2019274, Zeynep Tuba Çalışkan - 1805207

This project consists of counting the most used unigram words and numbers in mathematics, applied sciences and information science books published from 1900 to 2021.

Our aim in the project is to create a console application in C++ by counting the unigram words in the given text file and using the appropriate data structures for the 10 most used words in a minimum of seconds.

During the preparation phase of compiler, we put Visual Studio in release and Active(x64) mode, as stated in the Project report, in order to run the application at maximum performance. At the same time, we have chosen the optimization performance as the maximum value that visual studio offers us. Since the other options have no effect on our run-time, we left it at the values that Visual Studio automatically generates.



Configuration: Active(Release)   Platform: Active(x64)

Optimization	**Maximum Optimization (Favor Speed) (/O2)**
Inline Function Expansion	Default
Enable Intrinsic Functions	**Yes (/Oi)**
Favor Size Or Speed	**Neither**
Omit Frame Pointers	
Enable Fiber-Safe Optimizations	No
Whole Program Optimization	Yes (/GL)

Optimization
Select option for code optimization; choose Custom to use specific optimization options.   (/Od, /O1, /O2)

We have used main data structure and algorithm while making our application. Data structure is hashtable, and sorting algorithm is selection sort. The reason we use hashtables is because it is faster and easier to put words together with the same initials. Although quick sort is faster, generally, the reason we have used selection sort is that we only want to sort the top 10 words. In this way, we got a faster run time compared to quick sort algorithm.

We have avoided using third-party libraries while making the application. The extra libraries we have used to make the application are fstream, for opening the text file, and ctime for calculating the run time.

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <string>
#include <cassert>
```

To calculate the time, we have defined values for the start and end in `clock_t`

```
clock_t startTime, endTime;
```

We have defined a separate class for hashtable. Values set as private are an array for Unigrams and their counts, a second array for Stop Words. A totalCount that holds it to an int value to calculate the sum of the number of unigrams, and indexStatus to calculate the empty state assigning to the hashtable.

```
class hashUnigram {
private:
   string* listUnigrams;
   string stopWords[571];
   int* totalCount;
   int* indexStatus;
```

In addition to the consturctor, destructor and copy constructer, we have created 5 different functions to add words in the hashUnigram class, to control the stop word, to add the stop words, to perform the sort operation and to print the output.

```
void insertUnigram(string unigram, int count)
void insertStop()
bool isStopWord(string s)
std::string* sortTop10()
void printTop10()
```

`void insertUnigram` function We have defined an int value as a total. This total holds the sum of the ascii values of each letter in a unigram. We have assigned the hashtable according to the total. We have created a key value to be able to insert into the hashtable. The reason why we multiply the key value with the total and the first letter of the unigram is that we want to put them in the hastable by sorting alphabetically and finding the same values, We gave a great value to the hasthtable so that there is no space problem, so that we can process faster. Using the indexStatus array we have created, we check the fullness of the index and place the words in the hashtable.

```
void insertUnigram(string unigram, int count) {
        int total = 0;
        for (int i = 0; i < unigram.length(); ++i) {
            total += (int)unigram[i];
        }
        int key = (total * unigram[0] * 20) % 1000000;
        while (indexStatus[key] == 1 && listUnigrams[key] != unigram) {
            key = (key + 1) % 1000000;
        }
        if (listUnigrams[key] == unigram) {
            totalCount[key] += count;
        }
        if (indexStatus[key] != 1) {
            listUnigrams[key] = unigram;
            totalCount[key] += count;
            indexStatus[key] = 1;
        }
    }
```

`void insertStop()` fonction was created to select the words in the stopWords text and throw them into the stopWords array.

```
   void insertStop() {
        ifstream in;
        in.open("stopwords.txt");
        string stop;
        int i = 0;
        while (!in.eof()) {
            in >> stop;
            stopWords[i] = stop;
            i++;
        }
    }
```

`bool isStopWord(string s)` function checks whether all the words to be added to the hahstable are in the stop words array, and a sequential search is used when checking. Since the number of words in stop words is not too many, it did not harm us in terms of timing.

```
    bool isStopWord(string s) {
        bool found = false;
        for (int i = 0; i < 571; i++) {
            if (s == stopWords[i]) {
                found = true;
            }
            if (found == true) {
                break;
            }
        }
        return found;
    }
```

In the `std::string* sortTop10()` function, we first created 2 temporary arrays with an index of 1 million to assign the listUnigrams array and the totalCount array. we also created another string array called tempList to keep the first 10 words and their count. (`std::string* TempList = new std::string[20],std::string* tempUnigrams = new std::string[1000000], int* temptotalCount = new int[1000000]`). At the same time, we created 2 new int type values to find the maximum value and their index position (`int max, int maxPosition`).

First the words and then the counts are assigned to the created tempList in order. Each discarded word and count is equalized to -1 and " " , so that it does not appear again.

```
std::string* sortTop10() {

    std::string* TempList = new std::string[20];
    std::string* tempUnigrams = new std::string[1000000];
    int* temptotalCount = new int[1000000];
    for (int i = 0; i < 1000000; i++) {

        tempUnigrams[i] = listUnigrams[i];
        temptotalCount[i] = totalCount[i];
    }
    for (int i = 0; i < 20; i++) {
        TempList[i] = "";
    }

    int max;
    int maxPosition;

    for (int i = 0; i < 10; i++) {
        max = 0;
        maxPosition = 0;
        for (int j = 0; j < 1000000; j++) {
            if (temptotalCount[j] > max) {
                max = temptotalCount[j];
                maxPosition = j;
            }
        }
        TempList[i * 2] = tempUnigrams[maxPosition];
        TempList[(i * 2) + 1] = std::to_string(temptotalCount[maxPosition]);
        temptotalCount[maxPosition] = -1;
        tempUnigrams[maxPosition] = "";
    }
    return TempList;
}
```

Finally we have defined a `void printTop10()` function inside the `hashUnigram` class to print the values. function prints values to be " word number ". (The "" sign is representative.)

```
void printTop10() {
    string* PrintList = sortTop10();

    for (int i = 0; i < 10; i++) {
        std::cout << PrintList[i * 2] << " " << PrintList[(i * 2) + 1] <<
std::endl;
    }
}
```

In the int main function, startTime is initialized to calculate the time first.

```
startTime = clock();
```

Object h is created from hashUnigram class.

```
hashUnigram h;
```

All stop words are inserted to a list in h by using the insertStop function created in the hashUnigram class.

```
h.insertStop();
```

We use the ifstream and open methods to open the text file, including the unigrams.

```
ifstream inFile;
inFile.open("PublicationsDataSet.txt");
```

First, we have defined a string named document to retrieve the each document in PublicationsDataSet.txt one by one. We have written a method that reads from the beginning of the list to the end of the list, where we can get the correct words in a while loop. This method firstly, finds the important punctuation marks in our text unigram. These marks the beginning of our unigrams. Since we shouldn't have these punctuation marks, we apply a subtraction method. At the same time, we delete the punctuation marks that we should not include at the end of the text with the pop method. And also because the general structure of each unigram is [,"word":count], we add non-existent punctuation marks to the list. (The [] sign is representative.)

Since we should not take the words with unnecessary punctuation marks except the beginning and the end of the unigrams, we insert the unigrams into the hashtable by checking with the numOfUnwantedChar integer with if else structures.

The firt and last values represent the positions of punctuation marks at the beginning and end of the unigrams.

```
string document;
    while (!inFile.eof()) {
        getline(inFile, document);
        size_t pos = document.rfind("\":{\"");
        if (pos != string::npos) {
            document = document.substr(pos + 3);
        }
        document.insert(0, 1, ',');
        document.pop_back();
        document.pop_back();
        document += ",\"\":";
        while (document.length() > 0) {
            size_t first = document.find(",\"");
            if (first != string::npos) {
                document = document.substr(first + 2);
            }
            size_t last = document.find("\":");
            string unigram = document.substr(0, last + 0);

            if (last != string::npos) {
                document = document.substr(last + 2);
            }
            size_t newPos = document.find(",\"");
            string countString = document.substr(0, newPos + 0);
            int digit = 1;
            int count = 0;
            for (int i = countString.length() - 1; i >= 0; i--) {
                count += (countString[i] - 48) * digit;
                digit *= 10;
            }
            int numOfUnwantedChar = 0;
            while (!((unigram[0] >= 'a' && unigram[0] <= 'z') ||
                (unigram[0] >= '0' && unigram[0] <= '9') ||
                unigram[0] >= 'A' && unigram[0] <= 'Z')
                && unigram.length() > 0) {
                unigram = unigram.substr(1);
            }
            while (!((unigram[unigram.length() - 1] >= 'a' && unigram[unigram.length()
- 1] <= 'z') ||
                (unigram[unigram.length() - 1] >= '0' && unigram[unigram.length() - 1]
z<= '9') ||
                unigram[unigram.length() - 1] >= 'A' && unigram[unigram.length() - 1]
<= 'Z') &&
                unigram.length() > 0)
            {
                unigram.pop_back();
            }

            if (unigram.length() > 0) {
                for (int i = 0; i < unigram.length(); i++) {
                    if (!(unigram[i] >= 'a' && unigram[i] <= 'z') && unigram[i] !=
'\'' && !(unigram[i] >= 'A' && unigram[i] <= 'Z')) {
                        numOfUnwantedChar++;
                        break;
                    }
                }
                if (numOfUnwantedChar == 0) {
                    for (int i = 0; i < unigram.length(); i++) {
                        if (unigram[i] >= 'A' && unigram[i] <= 'Z') {
                            unigram[i] = char(unigram[i] + 32);
                        }
                    }
                    if (!h.isStopWord(unigram)) {
                        h.insertUnigram(unigram, count);
                    }
                }
            }
        }
    }
```

Finally, we closed the file we opened.

```
inFile.close();
```

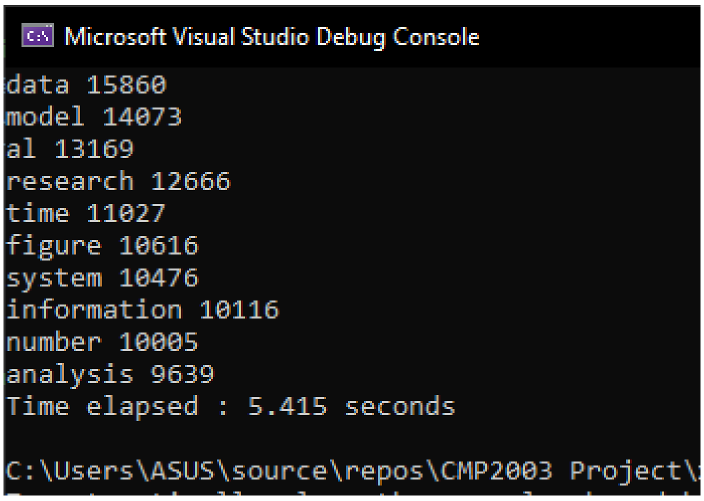
We called the `void printTop10()` function so that we can print the word top ten frequent on the screen.

```
h.printTop10();
```

We used our time calculation method by turning off the timing with endtime.

```
    endTime = clock();
    cout << "Time elapsed : " << ((double)(endTime - startTime)) / CLOCKS_PER_SEC << "
seconds" << endl;
```

The running time of our program is 5.5 seconds on average but the mimimum value we get at the max performance of the computer is 5.415 seconds.



```
data 15860
model 14073
al 13169
research 12666
time 11027
figure 10616
system 10476
information 10116
number 10005
analysis 9639
Time elapsed : 5.415 seconds

C:\Users\ASUS\source\repos\CMP2003 Project\
```

Zeynep Tuba Çalışkan 1805207, Mücahit Bayram 2019274.