Übung1:

1. Lösungs-Idee:

Es wird eine Klasse T9CodeConverter entwickelt, welche die Funktionen, welche ich später noch erläutern werde, sowie den Sample-Text und das Wörterbuch in verschiedenen Ausprägungen beinhält.
Der Pfad für das Wörterbuch sowie für den Text, kann beim Konstruktor der Klasse mit gegeben werden. In diesem wird dann das Wörterbuch in zwei verschieden Datenstrukturen geladen. Einmal in ein Set, und einmal in eine Map. In der Map wird das Wörterbuch nach der Länge seiner Wörter gruppiert. Möchte man ein Wort nachsehen, so kann dies sehr beschleunigt werden, da man nur mehr jene durchlaufen muss, welche von der Länge her gleich sind. Alternativ um nicht zwei Datenstrukturen zu laden, könnte man auch im Set mit lower_bound und upper_bound suchen, und die Wörter dazwischen sich ansehen, dies ist aber mit Sicherheit Zeitaufwändiger, als sich das Wörterbuch als Map zu laden. In der Realität, könnte man sich die Version mit dem Set sparen, da sie ja nur für Nummer C für die Veranschaulichung benötigt wird.

Der Text wird Wort für Wort in eine Map gespeichert, wobei der Key das Wort ist und der Value seine Anzahl. Es wird also Wort eingelesen, geprüft schon vorhanden, wenn ja zähle count 1 hoch, nein, füge einen neuen Datensatz mit count=1 ein. Man könnte noch prüfen, ob das Wort überhaupt im Wörterbuch verfügbar ist, das ist aber sehr Zeitaufwändig, und es ist besser einfach alles zu Laden. Beispielsweise das Wort und kommt 100-mal vor, dann muss es 100-mal gegen das Wörterbuch geprüft werden. Das ist sehr aufwändig, und die Wahrscheinlichkeit, dass ein Wort nicht im Wörterbuch vorkommt ist bekanntlich eher gering.

Als Text mit mehr als 20000 Wörter habe ich mir Faust 1 heruntergeladen. Dieser ist als text Datei UTF-8 kodiert frei verfügbar.

Aus dem Untericht wurde das File utils.hpp, sowie utils.cpp verwendet, und dem Projekt hinzugefügt. Es wurde daraus die Hilfsfunktion normalize verwendet.

a. Word2number()

Es kann einfach das eingelesene Wort Buchstabe für Buchstabe auf eine in vom Konstruktor der erstellten T9CodeMap, gemappt werden.

b. Number2String()

Es wird hierbei einfach Ziffer für Ziffer durchgegangen, und die möglichen Buchstaben Kombinationen werden in einem Set gespeichert. Als erstes wandle ich alle Ziffern auf deren mögliche Buchstaben um, danach Bilde ich das Kreuzprodukt dieser einzelnen möglichen Buchstaben.

c. Number2Words()

Hierbei mache ich es mir relativ einfach, indem ich als erstes einfach number2String aufrufe, und das Set welches zurückkommt, gegen mein Wörterbuch prüfe. Kommt ein Wort vor, so kommt es zu einem final Set welches zurückgegeben wird. Die effizienz dieses Verfahren gilt als eher schlecht, da egal wie lang, kurz das Wort ist, immer jedes Wort gegen das gesamte Wörterbuch geprüft wird. Weiters könnte man verbessern, dass man prüft, hat man ein Wort schon gegen das Wörterbuch geprüft, denn eine Prüfung gegen die bereits geprüften Datensätze ist viel weniger Aufwand als gegen das gesamte Wörterbuch mit x tausend Einträgen.

d. Number2WordsByLength()

Hierbei verwende ich wie auch schon oben, die Funktion number2String um einmal alle möglichen Kombinationen erzeugen zu lassen. Danach prüfe ich jedoch nicht mehr gegen das Gesamte Wörterbuch, sondern nur mehr jene Wörter die gleich lange sind. Ich habe mir dazu wie oben schon erwähnt, eine neue Datenstruktur mit einer Map überlegt. Danach kann ich mir das zu prüfende Set mit myMap[number.size()] holen und mit find, versuchen das Wort zu finden.

e. numberPrefix2Word()

Diese Funktion holt sich mit Number2WordsByLength() einmal alle möglichen Wörter. Danach wird das vorhandene Set Wort für Wort durchgelaufen. Bei jedem durchlauf prüfe ich dann alle Wörter des Wörterbuches, welche a) länger sind, und von diesen ob Sub String bis Wort Länge gleich ist. Wenn ja, füge ich dieses Wort ebenfalls zum Ergebnis hinzu.

f. numberPrefix2sortedWords()

In dieser Funktion hole ich mir einfach einmal alle Wörter mit der Funktion numberPrefix2Word(). Anschließend prüfe ich jedes gefundene Wort, ob es in meiner Map mit words und Count vorkommt. Kommt es vor Speichere ich das Ergebnis in eine MultiMap<int, string, greater<int>>, falls es nicht vorkommt ebenfalls jedoch mit key 0. Mittels dieser Map ist sichergestellt, dass die Wörter nun nach ihrer Häufigkeit absteigend Sortiert sind. Um diese Sortierung beizubehalten, jedoch die Info der Anzahl als eigenes Attribut wegzubekommen, füge ich diese Map noch in einen Vector<string>, welcher dann auch zurückgegeben wird.

2. Code:

T9CodeConverter.hpp:

```
#ifndef T9CODECONVERTER_H
#define T9CODECONVERTER_H
#include<iostream>
#include<map>
#include<algorithm>
#include<string>
#include<set>
#include<locale>
#include<list>
#include<fstream>
#include<utils.hpp>


using namespace std;
class T9CodeConverter
{
    private:
        map<int,string> t9CodeBase; // store T9Code map
        map<int,set<string>> dictionaryAdvanced; // dictionary entries
grouped by lenght
        set<string> dictionary; // dictionary basic
        map<string,int> wordsCount; // word and the frequency occuring in
the loaded text

        string dictPath; // path for dictionary
        string textPath; // path for text

        virtual void initT9CodeBase();
        virtual void loadDictionary();
        virtual void loadText();

        virtual bool dictContains(string word);
        virtual bool dictContainsFast(string word);
        inline virtual map<int,string> getT9CodeBase() const{return this-
>t9CodeBase;}

    public:
        T9CodeConverter(string dictionaryPath, string textPath);
        virtual ~T9CodeConverter();

        inline virtual string getDictionaryPath()const{return this-
>dictPath;}
        inline virtual string getTextPath()const{return this->textPath;}

        virtual string digitToString(int digit) const;
        virtual set<string> numberToStrings(string number) const;
        virtual string wordToNumber(string word)const;
        virtual int charToDigit(char c)const;
        virtual set<string>numberToWords(string number);
        virtual set<string>numberToWordsByLength(string number);
        virtual set<string>numberPrefixToWords(string number);
        virtual vector<string>numberPrefixToSortedWords(string number);
};

#endif // T9CODECONVERTER_H
```

T9CodeConverter.cpp:

```
#include "T9CodeConverter.h"
/// CONSTRUCTOR DESTRUCTOR ///
T9CodeConverter::T9CodeConverter(string dictionaryPath, string
textPath):dictPath(dictionaryPath), textPath(textPath){
    // initialize Dictionary and Text
    this->initT9CodeBase();
    this->loadDictionary();
    this->loadText();
}

T9CodeConverter::~T9CodeConverter(){
    // nothing TODO here
}

/**
* load dictionary into basic and extended Dictionary
**/
void T9CodeConverter::loadDictionary(){
    ifstream file;
    file.open(this->dictPath);
    if(file.fail()){
        cerr << "Cannot Load Dictionary" << flush;
        return;
    }
    string line;
    while (file >> line){
        string help = normalize(line);
        // insert word into advanced dictionary ordered by lenght
            dictionaryAdvanced[help.size()].insert(help);
        // insert word into basic dictionary
        dictionary.insert(help);
    }
    cout << "Dictionary Loaded with: " << this->dictionary.size() << "
entries \n" << flush;
    file.close();
}
/**
* load the 20000 words text
**/
void T9CodeConverter::loadText(){
    ifstream fileStream;
    fileStream.open(this->textPath);
    if(fileStream.fail()){
        cerr << "Cannot Load Text " << flush;
        return;
    }
    string line;
    int count=0;
    while (fileStream >> line){
        string help = normalize(line);
        //if(this->dictContainsFast(help)){ do not check if word is in
dictionary will fast-up loading text
            map<string,int>::iterator it = wordsCount.find(help);
            if(it == wordsCount.end()){
                wordsCount.insert(pair<string,int>(help,1));
            }else{
                (*it).second += 1;
            }
            count ++;
        //}
    }
```

```
    cout << "Text Loaded with: " << count << " entries \n" << flush;
    fileStream.close();
}
/**
* initialize a map with T9Code mapping
**/
void T9CodeConverter::initT9CodeBase(){
    t9CodeBase[2] = "abc";
    t9CodeBase[3] = "def";
    t9CodeBase[4] = "ghi";
    t9CodeBase[5] = "jkl";
    t9CodeBase[6] = "mno";
    t9CodeBase[7] = "pqrs";
    t9CodeBase[8] = "tuv";
    t9CodeBase[9] = "wxyz";
}
/**
* function to check if a word is in dictonary optimized
**/
bool T9CodeConverter::dictContainsFast(string word){
    set<string> help = dictionaryAdvanced[word.size()];
    set<string>::const_iterator got = help.find(word);
    if(got != help.end()){
        return true;
    }
    return false;

}
/**
* function to check if a word is in dictionary
**/
bool T9CodeConverter::dictContains(string word){
    set<string>::const_iterator got = dictionary.find(word);
    if(got != dictionary.end()){
        return true;
    }
    return false;
}
/**
* function to convert a Character into a Number
* e.g. 'a' -> 2
**/
int T9CodeConverter::charToDigit(char c)const{
    int value;
    for_each(t9CodeBase.begin(), t9CodeBase.end(), [&c,
&value](pair<int,string> entry){
        if(entry.second.find(tolower(c),0) != std::string::npos){
            value=entry.first;
        }
    });
    return value;
}
/**
* function to convert a T9Digit into a String
* e.g. 2 -> "abc"
**/
string T9CodeConverter::digitToString(int digit) const{
    string value;
    for_each(t9CodeBase.begin(), t9CodeBase.end(), [&digit,
&value](pair<int,string> entry){
        if(entry.first == digit){
            value.append(entry.second);
        }
```

```
    });
    return value;
}
/**
* function that returns all possible combination of strings for a given
T9Code
**/
set<string> T9CodeConverter::numberToStrings(string number) const{
    list<string> wordsList;
    set<string> words;
    set<string> tmp;
    set<string> tmpWords;
    int sizeOfWord = number.size();

    for_each(number.begin(), number.end(), [&wordsList, this](char c){
        wordsList.push_back(digitToString(atoi(&c)));
    });
    for_each(wordsList.begin(), wordsList.end(),[&words, &tmp, &tmpWords,
&sizeOfWord](string chars){
        for_each(chars.begin(),chars.end(),[ &tmp, &tmpWords](char letter){
            if(tmpWords.empty()){
                tmp.insert(&letter);
            }else{
                for_each(tmpWords.begin(), tmpWords.end(),[&tmp,
&letter](string word){
                    tmp.insert(word.append((&letter)));
                });
            }
        });
        for_each(tmp.begin(),tmp.end(),[&words, &tmp, &tmpWords,
&sizeOfWord](string word){
            if(word.size() == sizeOfWord){
                words.insert(word);
            }
            tmpWords.insert(word);
        });
    });
    return words;
}
/**
* function to convert a word into a T9Code
**/
string T9CodeConverter::wordToNumber(string word)const{
    string retVal;
    for_each(word.begin(), word.end(),[&retVal,this](char c){
        retVal.append(to_string(charToDigit(c)));
    });
    return retVal;
}
/**
* function to convert any T9Code to words
**/
set<string> T9CodeConverter::numberToWords(string number){
    set<string> values = this->numberToStrings(number);
    set<string> finalValues;
    for_each(values.begin(), values.end(),[&finalValues,this](string word){
        if(this->dictContains(word)){
            finalValues.insert(word);
        }
    });
    return finalValues;
}
/**
```

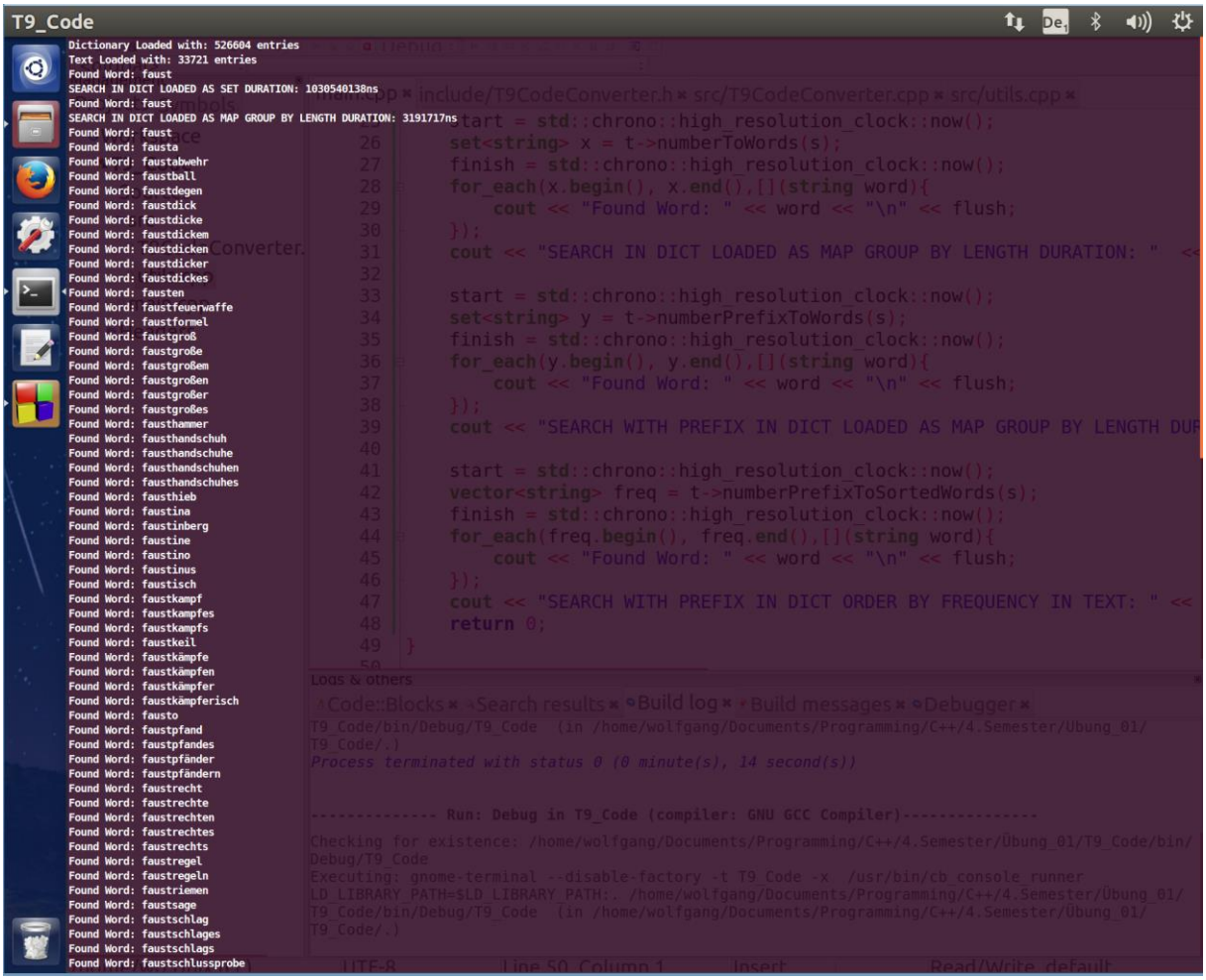
```
* function to Convert any T9Code and Use a Dictionary sorted by length
**/
set<string> T9CodeConverter::numberToWordsByLength(string number){
    set<string> values = this->numberToStrings(number);
    set<string> finalValues;
    for_each(values.begin(), values.end(),[&finalValues,this](string word){
        if(this->dictContainsFast(word)){
            finalValues.insert(word);
        }
    });
    return finalValues;
}
/**
* function to convert any T9Code to words and words
* where the generated word is in prefix
**/
set<string> T9CodeConverter::numberPrefixToWords(string number){
    set<string> values = this->numberToStrings(number);
    set<string> finalValues;
    for_each(values.begin(), values.end(),[&finalValues,this](string word){
        if(this->dictContainsFast(word)){
            finalValues.insert(word);
        }
    });
    for(int i=number.size(); i<dictionaryAdvanced.size(); i++){
        set<string> help = dictionaryAdvanced[i];
        for_each(help.begin(), help.end(), [&finalValues,&number](string
word){
            set<string>::const_iterator got =
finalValues.find(word.substr(0,number.size()));
            if(got != finalValues.end()){
                finalValues.insert(word);
            }
        });
    }
    return finalValues;
}
/**
* function convert T9Code to possible words and sort by count of an text
**/
vector<string> T9CodeConverter::numberPrefixToSortedWords(string number){
    set<string> values = this->numberPrefixToWords(number);
    multimap<int,string, greater_equal<int>> sortValues;
    vector<string> finalValues;
    // loop through generated words and check frequency
    // inserting value in a multimap ensures values get inserted sorted
desc by given comperator greater_equal
    for_each(values.begin(), values.end(), [&sortValues, this](string s){
        map<string,int>::const_iterator got = this->wordsCount.find(s);
        if(got == this->wordsCount.end()){
            // if loaded text does not contain word frequency is 0
            sortValues.emplace(0,s+"Frequency: 0");
        }else{
            sortValues.emplace(((*got).second),s + "Frequency:
"+to_string((*got).second) );
        }
    });
    for_each(sortValues.begin(), sortValues.end(),
[&finalValues](pair<int,string> entry){
        finalValues.push_back(entry.second);
    });
    return finalValues;
}
```

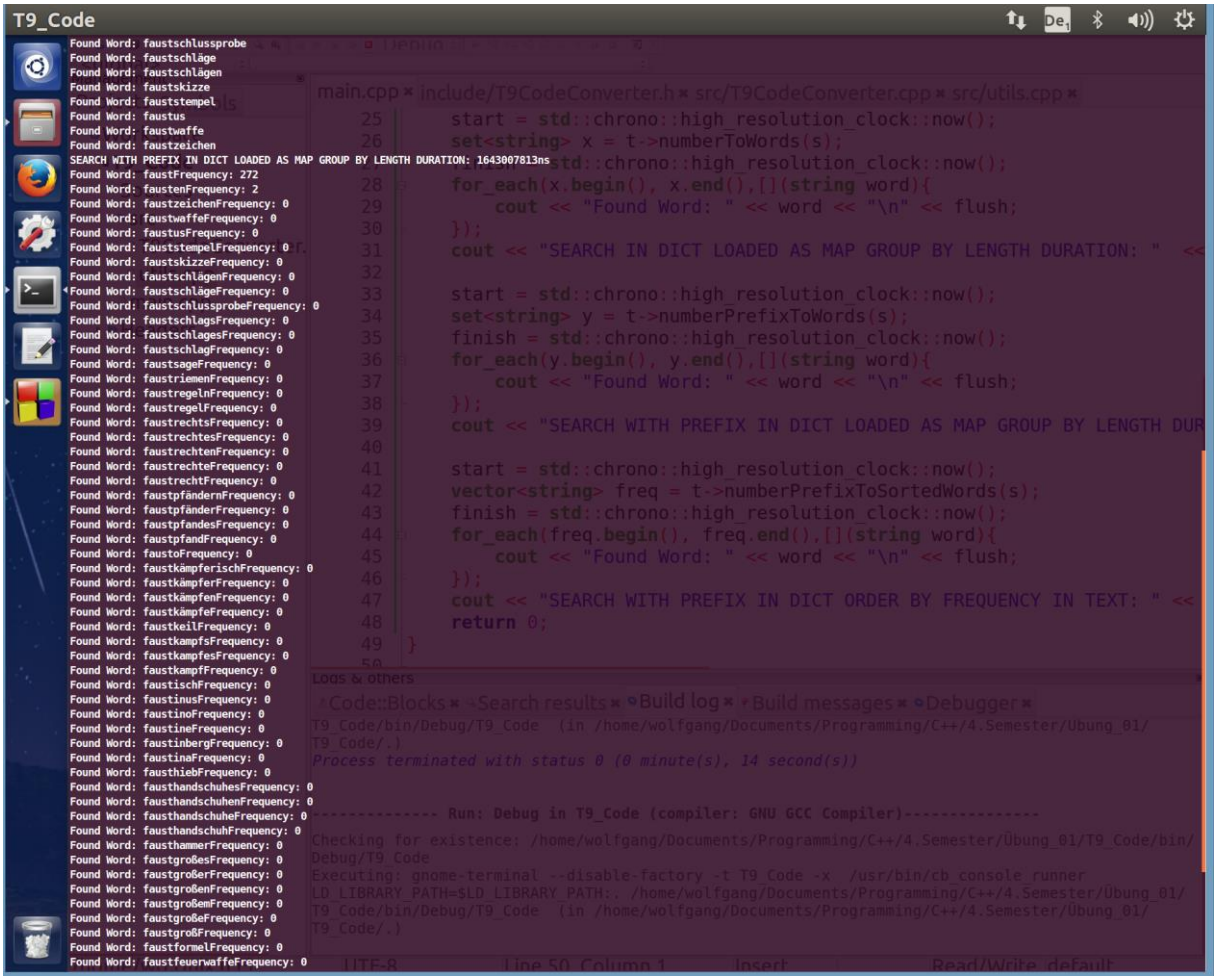
Main.cpp:

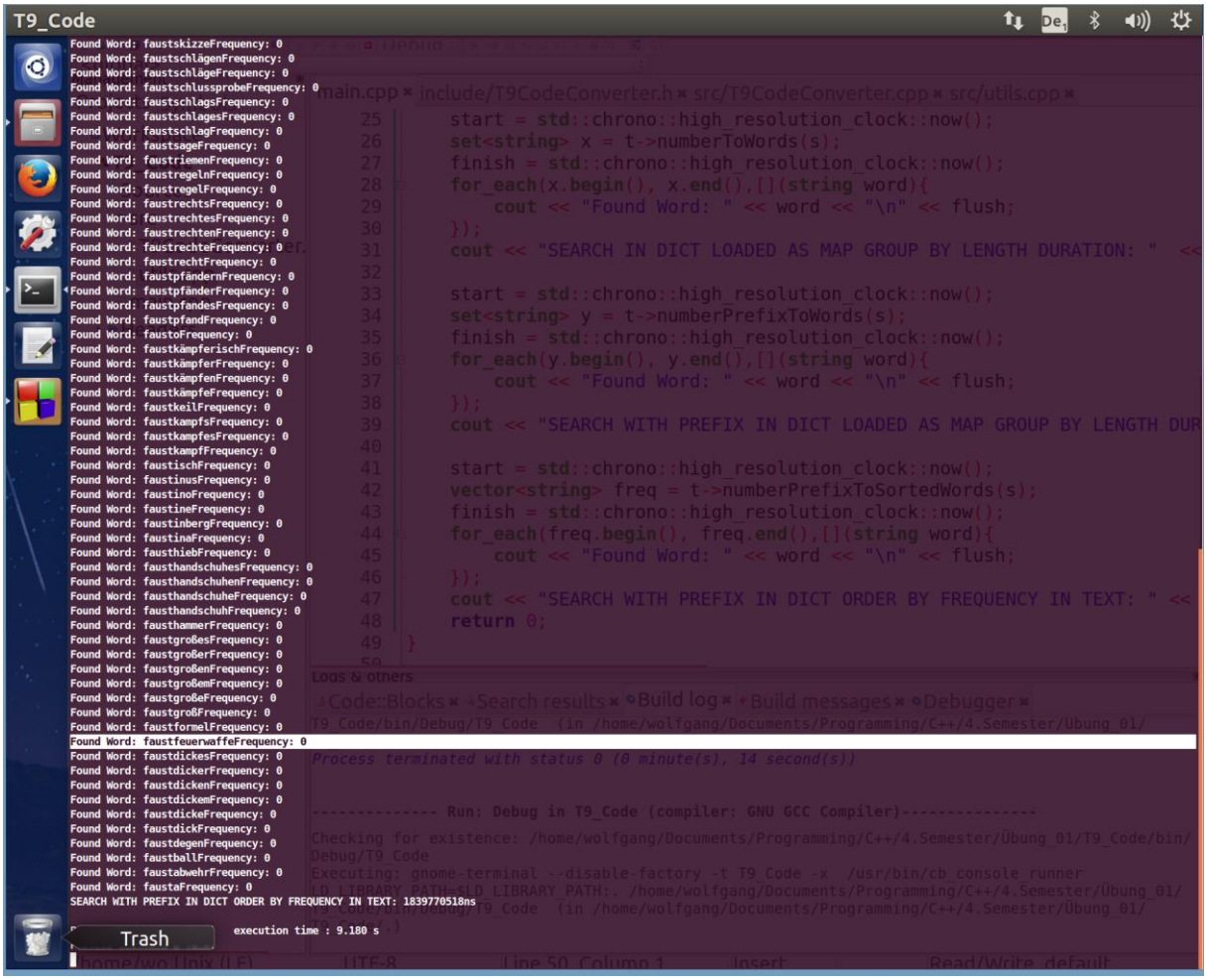
```
#include <iostream>
#include <string>
#include <map>
#include <algorithm>
#include <T9CodeConverter.h>
#include <unordered_set>
#include <chrono>
#include <istream>
#include <iterator>
using namespace std;

int main(){
    T9CodeConverter* t = new T9CodeConverter("de_neu.dic", "faust1.txt");

    string s=  "32878";

    auto start = std::chrono::high_resolution_clock::now();
    set<string> c = t->numberToWordsByLength(s);
    auto finish = std::chrono::high_resolution_clock::now();
    for_each(c.begin(), c.end(),[](string word){
        cout << "Found Word: " << word << "\n" << flush;
    });
    cout << "SEARCH IN DICT LOADED AS SET DURATION: "  <<
std::chrono::duration_cast<std::chrono::nanoseconds>(finish-start).count()
<< "ns" << endl << flush;

    start = std::chrono::high_resolution_clock::now();
    set<string> x = t->numberToWords(s);
    finish = std::chrono::high_resolution_clock::now();
    for_each(x.begin(), x.end(),[](string word){
        cout << "Found Word: " << word << "\n" << flush;
    });
    cout << "SEARCH IN DICT LOADED AS MAP GROUP BY LENGTH DURATION: "  <<
std::chrono::duration_cast<std::chrono::nanoseconds>(finish-start).count()
<< "ns" << endl << flush;

    start = std::chrono::high_resolution_clock::now();
    set<string> y = t->numberPrefixToWords(s);
    finish = std::chrono::high_resolution_clock::now();
    for_each(y.begin(), y.end(),[](string word){
        cout << "Found Word: " << word << "\n" << flush;
    });
    cout << "SEARCH WITH PREFIX IN DICT LOADED AS MAP GROUP BY LENGTH
DURATION: "  <<
std::chrono::duration_cast<std::chrono::nanoseconds>(finish-start).count()
<< "ns" << endl << flush;

    start = std::chrono::high_resolution_clock::now();
    vector<string> freq = t->numberPrefixToSortedWords(s);
    finish = std::chrono::high_resolution_clock::now();
    for_each(freq.begin(), freq.end(),[](string word){
        cout << "Found Word: " << word << "\n" << flush;
    });
    cout << "SEARCH WITH PREFIX IN DICT ORDER BY FREQUENCY IN TEXT: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(finish-start).count()
<< "ns" << endl << flush;
    return 0;
}
```

3. Test-Case:

Es wurden alle Methoden der Klasse während der Entwicklung getestet.
Als Abschlusstest wurde danach noch das oben ersichtliche Hauptprogramm entwickelt, und ausgeführt. Als zusätzliches Feature weist, dieses Programm die Dauern der Suche auf, wodurch schnell ersichtlich ist, welche Methoden effizienter sind als die andern.




T9_Code
Found Word: faustschlussprobe
Found Word: faustschläge
Found Word: faustschlägen
Found Word: faustskizze
Found Word: fauststempel
Found Word: faustus
Found Word: faustwaffe
Found Word: faustzeichen
SEARCH WITH PREFIX IN DICT LOADED AS MAP GROUP BY LENGTH DURATION: 1643007813ns
Found Word: faustFrequency: 272
Found Word: faustenFrequency: 2
Found Word: faustzeichenFrequency: 0
Found Word: faustwaffeFrequency: 0
Found Word: faustusFrequency: 0
Found Word: fauststempelFrequency: 0
Found Word: faustskizzeFrequency: 0
Found Word: faustschlägenFrequency: 0
Found Word: faustschlägeFrequency: 0
Found Word: faustschlussprobeFrequency: 0
Found Word: faustschlagsFrequency: 0
Found Word: faustschlagesFrequency: 0
Found Word: faustschlagFrequency: 0
Found Word: faustsageFrequency: 0
Found Word: faustriemenFrequency: 0
Found Word: faustregelnFrequency: 0
Found Word: faustregelFrequency: 0
Found Word: faustrechtsFrequency: 0
Found Word: faustrechtesFrequency: 0
Found Word: faustrechtenFrequency: 0
Found Word: faustrechteFrequency: 0
Found Word: faustrechtFrequency: 0
Found Word: faustpfändernFrequency: 0
Found Word: faustpfänderFrequency: 0
Found Word: faustpfandesFrequency: 0
Found Word: faustpfandFrequency: 0
Found Word: faustoFrequency: 0
Found Word: faustkämpferischFrequency: 0
Found Word: faustkämpferFrequency: 0
Found Word: faustkämpfenFrequency: 0
Found Word: faustkämpfeFrequency: 0
Found Word: faustkeilFrequency: 0
Found Word: faustkampfsFrequency: 0
Found Word: faustkampfesFrequency: 0
Found Word: faustkampfFrequency: 0
Found Word: faustischFrequency: 0
Found Word: faustinusFrequency: 0
Found Word: faustinoFrequency: 0
Found Word: faustineFrequency: 0
Found Word: faustinbergFrequency: 0
Found Word: faustinaFrequency: 0
Found Word: fausthiebFrequency: 0
Found Word: fausthandschuhesFrequency: 0
Found Word: fausthandschuhenFrequency: 0
Found Word: fausthandschuheFrequency: 0
Found Word: fausthandschuhFrequency: 0
Found Word: fausthammerFrequency: 0
Found Word: faustgroßesFrequency: 0
Found Word: faustgroßerFrequency: 0
Found Word: faustgroßenFrequency: 0
Found Word: faustgroßemFrequency: 0
Found Word: faustgroßeFrequency: 0
Found Word: faustgroßFrequency: 0
Found Word: faustformelFrequency: 0
Found Word: faustfeuerwaffeFrequency: 0
main.cpp
include/T9CodeConverter.h
src/T9CodeConverter.cpp
src/utils.cpp
start = std::chrono::high_resolution_clock::now();
set<string> x = t->numberToWords(s);
finish = std::chrono::high_resolution_clock::now();
for_each(x.begin(), x.end(),[](string word){
cout << "Found Word: " << word << "\n" << flush;
});
cout << "SEARCH IN DICT LOADED AS MAP GROUP BY LENGTH DURATION: "
start = std::chrono::high_resolution_clock::now();
set<string> y = t->numberPrefixToWords(s);
finish = std::chrono::high_resolution_clock::now();
for_each(y.begin(), y.end(),[](string word){
cout << "Found Word: " << word << "\n" << flush;
});
cout << "SEARCH WITH PREFIX IN DICT LOADED AS MAP GROUP BY LENGTH DUR
start = std::chrono::high_resolution_clock::now();
vector<string> freq = t->numberPrefixToSortedWords(s);
finish = std::chrono::high_resolution_clock::now();
for_each(freq.begin(), freq.end(),[](string word){
cout << "Found Word: " << word << "\n" << flush;
});
cout << "SEARCH WITH PREFIX IN DICT ORDER BY FREQUENCY IN TEXT: " <<
return 0;
}
Logs & others
Code::Blocks
Search results
Build log
Build messages
Debugger
T9_Code/bin/Debug/T9_Code (in /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/
T9_Code/.)
Process terminated with status 0 (0 minute(s), 14 second(s))
-------------- Run: Debug in T9_Code (compiler: GNU GCC Compiler)---------------
Checking for existence: /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/T9_Code/bin/
Debug/T9_Code
Executing: gnome-terminal --disable-factory -t T9_Code -x /usr/bin/cb_console_runner
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/
T9_Code/bin/Debug/T9_Code (in /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/
T9_Code/.)
UTF-8
Line 50, Column 1
Insert
Read/Write default


T9_Code
Found Word: faustskizzeFrequency: 0
Found Word: faustschlägenFrequency: 0
Found Word: faustschlägeFrequency: 0
Found Word: faustschlussprobeFrequency: 0
Found Word: faustschlagsFrequency: 0
Found Word: faustschlagesFrequency: 0
Found Word: faustschlagFrequency: 0
Found Word: faustsageFrequency: 0
Found Word: faustriemenFrequency: 0
Found Word: faustregelnFrequency: 0
Found Word: faustregelFrequency: 0
Found Word: faustrechtsFrequency: 0
Found Word: faustrechtesFrequency: 0
Found Word: faustrechtenFrequency: 0
Found Word: faustrechteFrequency: 0
Found Word: faustrechtFrequency: 0
Found Word: faustpfändernFrequency: 0
Found Word: faustpfänderFrequency: 0
Found Word: faustpfandesFrequency: 0
Found Word: faustpfandFrequency: 0
Found Word: faustoFrequency: 0
Found Word: faustkämpferischFrequency: 0
Found Word: faustkämpferFrequency: 0
Found Word: faustkämpfenFrequency: 0
Found Word: faustkämpfeFrequency: 0
Found Word: faustkeilFrequency: 0
Found Word: faustkampfsFrequency: 0
Found Word: faustkampfesFrequency: 0
Found Word: faustkampfFrequency: 0
Found Word: faustischFrequency: 0
Found Word: faustinusFrequency: 0
Found Word: faustinoFrequency: 0
Found Word: faustineFrequency: 0
Found Word: faustinbergFrequency: 0
Found Word: faustinaFrequency: 0
Found Word: fausthiebFrequency: 0
Found Word: fausthandschuhesFrequency: 0
Found Word: fausthandschuhenFrequency: 0
Found Word: fausthandschuheFrequency: 0
Found Word: fausthandschuhFrequency: 0
Found Word: fausthammerFrequency: 0
Found Word: faustgroßesFrequency: 0
Found Word: faustgroßerFrequency: 0
Found Word: faustgroßenFrequency: 0
Found Word: faustgroßemFrequency: 0
Found Word: faustgroßeFrequency: 0
Found Word: faustgroßFrequency: 0
Found Word: faustformelFrequency: 0
Found Word: faustfeuerwaffeFrequency: 0
Found Word: faustdickesFrequency: 0
Found Word: faustdickerFrequency: 0
Found Word: faustdickenFrequency: 0
Found Word: faustdickemFrequency: 0
Found Word: faustdickeFrequency: 0
Found Word: faustdickFrequency: 0
Found Word: faustdegenFrequency: 0
Found Word: faustballFrequency: 0
Found Word: faustabwehrFrequency: 0
Found Word: faustaFrequency: 0
SEARCH WITH PREFIX IN DICT ORDER BY FREQUENCY IN TEXT: 1839770518ns
execution time : 9.180 s
Trash
main.cpp
include/T9CodeConverter.h
src/T9CodeConverter.cpp
src/utils.cpp
start = std::chrono::high_resolution_clock::now();
set<string> x = t->numberToWords(s);
finish = std::chrono::high_resolution_clock::now();
for_each(x.begin(), x.end(),[](string word){
cout << "Found Word: " << word << "\n" << flush;
});
cout << "SEARCH IN DICT LOADED AS MAP GROUP BY LENGTH DURATION: "
start = std::chrono::high_resolution_clock::now();
set<string> y = t->numberPrefixToWords(s);
finish = std::chrono::high_resolution_clock::now();
for_each(y.begin(), y.end(),[](string word){
cout << "Found Word: " << word << "\n" << flush;
});
cout << "SEARCH WITH PREFIX IN DICT LOADED AS MAP GROUP BY LENGTH DUR
start = std::chrono::high_resolution_clock::now();
vector<string> freq = t->numberPrefixToSortedWords(s);
finish = std::chrono::high_resolution_clock::now();
for_each(freq.begin(), freq.end(),[](string word){
cout << "Found Word: " << word << "\n" << flush;
});
cout << "SEARCH WITH PREFIX IN DICT ORDER BY FREQUENCY IN TEXT: "
return 0;
}
Logs & others
Code::Blocks
Search results
Build log
Build messages
Debugger
T9_Code/bin/Debug/T9_Code (in /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/
Process terminated with status 0 (0 minute(s), 14 second(s))
Run: Debug in T9_Code (compiler: GNU GCC Compiler)
Checking for existence: /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/T9_Code/bin/
Debug/T9_Code
Executing: gnome-terminal --disable-factory -t T9_Code -x /usr/bin/cb_console_runner
LIBRARY_PATH:. /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/
T9_Code (in /home/wolfgang/Documents/Programming/C++/4.Semester/Ubung_01/