SWE 4x Übung zu Softwareentwicklung mit modernen Plattformen 4 **SS 2015, Übung 1**

Abgabetermin: SA in der KW 12

☑ **Gr. 1, E. Pitzer** **Name** Stefan Kert **Aufwand in h** 10

☐ **Gr. 2, F. Gruber-Leitner**

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

T9 (2 + 4 + 4 + 4 + 4 + 6 Punkte)

Der T9-Code (*Text on 9 keys*, siehe *http://de.wikipedia.org/wiki/Text_on_9_keys*) bildet das Leerzeichen und die 26 Buchstaben des Alphabets auf die neun Dezimalziffern 0 sowie 2 bis 9 (z. B. auf den Tasten eines Mobiltelefons) ab, und zwar gemäß folgender Tabelle:

Buchstaben	*Ziffer*
a, b, c	2
d, e, f	3
g, h, i	4
j, k, l	5
m, n, o	6
p, q, r, s	7
t, u, v	8
w, x, y, z	9

(aus *http://de.wikipedia.org/)*

Damit ist eine eindeutige Abbildung von Wörtern auf ganze Zahlen möglich (z. B. wird das Wort "kiss" auf die Zahl 5477 abgebildet).

Nutzbringend (als Eingabehilfe) verwendet wird dieser Code gerade für die Abbildung in der entgegengesetzten Richtung: Eine ganze Zahl (also eine Ziffernfolge, die z. B. über die Handy-Tastatur eingegeben wird) entspricht nun allerdings mehreren Zeichenketten, wobei aber nur wenige davon tatsächlich Wörter sind (z. B. kann die Zahl 5477 nicht nur auf das Wort "kiss" abgebildet werden, sondern sinnigerweise auch auf das Wort "lips", aber auch auf das Wort "lisp" und viele weitere, leider unsinnige Zeichenketten wie z. B. "lhqr").

Für eine Ziffernfolge der Länge n gibt es mindestens 3^n und höchstens 4^n Zeichenketten, in welche diese Ziffernfolge abgebildet werden kann, je nachdem, welche Ziffern darin vorkommen – nur sehr wenige dieser Zeichenketten sind Wörter.

a) Entwickeln Sie eine Funktion `word2number()`, die ein Wort mittels iTap/T9 auf eine Ziffernfolge abbildet. Um längere Wörter und die sich daraus ergebenden sehr großen Zahlen darstellen zu können, sollen auch die Ziffernfolgen in Form von Zeichenketten dargestellt werden (z. B. soll `word2number("kiss")` die Zeichenkette `"5477"` liefern).

b) Entwickeln Sie eine Funktion `number2strings()`, die für eine Ziffernfolge (in Form einer Zeichenkette) als Funktionsergebnis in einem passenden Behälter alle Zeichenketten liefert, die sich aus der Ziffernfolge bilden lassen. Z. B sollte `number2strings("5477")` insgesamt 144 Zeichenketten von `"jgpp"` bis `"liss"` liefern.

Die Funktion aus b) ist als Eingabehilfe noch nicht geeignet, da sie – wie schon erwähnt – auch viele unsinnige Zeichenketten liefert. Hat man allerdings ein Wörterbuch zur Verfügung, z. B. jenes, das Sie unter http://tug.ctan.org/tex-archive/systems/win32/winedt/dict/de_neu.zip herunterladen können, das über 500.000 deutsche enthält, kann man mit dessen Hilfe "die Spreu vom Weizen trennen".

c) Entwickeln Sie eine Funktion `number2words()`, die für eine Ziffernfolge nur mehr solche Zeichenketten liefert, die sich auch im Wörterbuch befinden, bei denen es sich also um "echte Wörter" handelt. Gehen Sie dabei so vor, dass Sie die Wörterbuchdatei beim Programmstart in einen geeigneten Behälter einlesen, und dass Sie dann jede Zeichenkette, die Sie erzeugen im Wörterbuch suchen: wird sie gefunden, handelt es sich um ein Wort und kann in den Ergebnisbehälter aufgenommen werden. Wie beurteilen Sie die Effizienz dieses Verfahrens?

d) Eine weitere Möglichkeit dieses Problem zu lösen, besteht darin, die Wörter im Wörterbuch primär nach ihrer Länge zu betrachten: Wählen Sie einen Behälter, der es gestattet, für alle Wortlängen *n* die Wörter dieser Länge so zu speichern, dass man in einer neuen Version Ihrer Funktion `number2wordsByLength()` aufgrund der Länge der übergebenen Ziffernfolge die Wortsuche auf Wörter mit der passenden Länge einschränken kann. Die Ergebnisse sollten gegenüber c) unverändert bleiben, aber wie beurteilen Sie die Effizienz dieses neuen Verfahrens?

e) Eine noch viel nützlichere Eingabehilfe bestünde darin, dass für eine Ziffernfolge (die noch kein vollständiges Wort darstellen muss) nicht nur jene Wörter berechnet werden, die in der Länge genau dazu passen, sondern auch all jene Wörter, die sich zukünftig (durch Verlängerung der aktuellen Ziffernfolge) daraus ergeben könnten, für welche die Ziffernfolge also ein gültiges Präfix darstellt. Entwickeln Sie eine neue Funktion `numberPrefix2word()`, die für eine Ziffernfolge alle Wörter mit passendem Präfix liefert. So soll z. B. für die Ziffernfolge `"4355"` nicht nur (wie schon bisher) das in der Länge genau passende Wort "hell" sondern auch die längeren Wörter wie "hellen" und "hello" geliefert werden. Überlegen Sie gut, ob der/die bisherige/n Behälter für das Wörterbuch auch für diese Aufgabe (noch) geeignet ist/sind und verwenden Sie ev. einen anderen. Wie beurteilen Sie die Effizienz dieses letzten Verfahrens?

f) Die Benutzerfreundlichkeit lässt sich noch weiter steigern, indem Sie die zu einem Präfix gehörende Wortmenge nach der Verwendungshäufigkeit sortieren. Erweitern Sie ihr Wörterbuch um eine Möglichkeit, die Häufigkeit der einzelnen Wörter zu speichern und initialisieren Sie das Wörterbuch einerseits aus der angegeben Quelle und andererseits, indem Sie aus einem repräsentativen Text – der mindestens 20.000 Wörter umfassen sollte und idealerweise in der gleichen Sprache wie das Wörterbuch verfasst wurde – die Worthäufigkeiten ermitteln. Entwickeln Sie dazu eine weiter Funktion `numberPrefix2sortedWords()`, das für einen Präfix dieselbe Wortmenge wie `numberPrefix2word()` ermittelt, die Wortmenge aber nach der Worthäufigkeit absteigend sortiert zurückgibt.

Ein Tipp, um Ihre Nerven (und Ihren Rechner) zu schonen:

Nutzen Sie die STL möglichst gut aus und testen Sie Ihr Programm zu Beginn eventuell mit einer selbst kleinen Wörterbuchdatei (z.B. 100 Wörter), so dass Sie Fehler leichter lokalisieren können. Verwenden Sie die Wörterbuchdatei mit den über 500.000 Einträgen erst wenn Ihr Programm "halbwegs" funktioniert.

1 Einleitung

Bei der ersten Übung sollte ein Decoder entworfen werden, mittels welchem es möglich ist eine Zeichenkette in eine Nummer zu konvertieren und eine solche Nummer auch wieder in einen Text zu konvertieren. Dies ist die Dokumentation für das Programm.

1.1 Lösungsidee

Für den Converter sollten folgende Methoden geschrieben werden:

- *Word2Number*
- *Number2Strings*
- *Number2Words*
- *Number2WordsByLength*
- *NumberPrefix2Word*
- *NumerPrefix2SortedWords*

Diese Methoden sollten in eine Klasse *T9Converter* gekapselt werden. In dieser Klasse gibt es eine Methode zum Initialisieren der T9 – Mappings, welche in einer weiteren Klasse *T9MappingEntry* gekapselt sind. Diese Klasse enthält die Ziffer und die einzelnen Buchstaben die miteinander verbunden werden. Weiters gibt es in der *T9MappingEntry* Klasse eine Methode zum Überprüfen ob der gesuchte Buchstabe vorhanden ist und weitere Methoden zum Zugreifen auf die Ziffer bzw. die Buchstaben. Die Buchstaben sind in dieser Klasse in einem „set<string>“ gespeichert. Weiters existiert in der *T9Converter* Klasse eine Methode zum Überprüfen ob die angegeben Zahl eine valide T9 Zahl ist. Dies ist dann der Fall wenn nur Ziffern im Bereich von 2-9 vorhanden sind.

Für etwaige Funktionalität zum Auslesen von Dateien und weitere vielseitig nutzbare Methoden gibt es eine „utility“ Datei in der folgende Methoden vorhanden sind:

- *StringToUpper*
  - Diese Methode wandelt den gegebenen Wert in UpperCases um und gibt diesen zurück
- *ReadAllLinesFromFile*
  - Diese Methode liest Zeilen aus einer Datei und ruft danach eine mittels Lambda Expression übergebene sog. Continuation auf, welche von der Aufrufenden Methode gesetzt wird. Dieser Continuation wird die aktuell gelesene Zeile übergeben.

- *ReadAllWordsFromFile*
  - Hier gilt das gleiche wie für *ReadAllLinesFromFile* nur wird hier wortweiße gelesen
- *IncrementString*
  - Diese Methode erhöht den gegeben String um 1

Diese utility Funktionen werden unter anderem in der *main* Funktion und in der Klasse *T9Converter* verwendet.

1.2 Word2Number

1.2.1 Lösungsidee

Die Funktion *Word2Number* sollte für ein gegebenes Wort die T9 Zahl zurückgeben. Das heißt es sollte zum Beispiel für das Wort Bier 2437 zurückgegeben werden. Um zu einer Lösung zu kommen wird das angegeben Wort Zeichen für Zeichen durchlaufen und für jedes Zeichen der dazugehörige T9 Buchstabe gesucht. Es werden dazu die Mappings durchlaufen und wenn das Zeichen im Mapping vorhanden ist wird die Ziffer zurückgegeben. Alle zurückgegebenen Ziffern werden an einen string angehängt und schließlich zurückgegeben. Falls kein Wert angegeben wird sollte eine *invalid_argument* - Exception geworfen werden.

1.2.2 Quellcode

```
string T9Converter::Word2Number(const string& word){
	string retValue = "";
	if(!this->IsWordValid(word)){
       throw invalid_argument("The parameter word was empty or invalid char was given. Param: " + word);
	}
	for (auto character : word){
		for (auto entry : this->_t9MappingEntries){
			if (entry.IsCharAvailableInMappingEntry(character)){
				retValue += entry.GetMappingDigit();
			}
		}
	}
	return retValue;
}
```

1.2.3 Tests

1.2.3.1 Kein Wort übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Word2Number NoInput
Test: Word2Number-NoInput
Invalid Argument exception was caught. Exception text was: The parameter word was empty
```

1.2.4 Ungültiges Wort übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Word2Number InvalidWord
Test: Word2Number-InvalidWord
Invalid Argument exception was caught. Exception text was: The parameter word was empty or invalid char was given. Param: &hae34djfl4@
```

1.2.4.1 Verschiedene Wörter übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Word2Number PrintResult
Test: Word2Number
Bier = 2437
Superman = 78737626
Hausfrauenbingo = 428737283624646
```

1.2.4.2 Kiss übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Word2Number Kiss
Test: Word2Number-Kiss
kiss = 5477
```

1.2.5 Beurteilung des Verfahrens

Grundsätzlich gibt es bei der Funktion *Word2Number* nicht viel zu beurteilen. Der Algorithmus ist ein sehr einfacher und er ist auch nicht sehr zeitaufwendig, wodurch auch die Umwandlung langer Wörter leicht möglich ist.

1.3 Number2Strings

1.3.1 Lösungsidee

Bei der Methode Number2Strings wird eine Zahl übergeben für die alle möglichen Kombinationen zurückgegeben werden sollten. Als erstes sollte der gegebene Parameter überprüft werden ob die gegebene Zahl valid ist. D.h. die Zahl muss angegeben sein und darf außerdem die Ziffern 0 und 1 nicht enthalten. Wenn die gegebene Zahl valide ist, werden mittels einem rekursiven Algorithmus die gültigen Mappings ermittelt. Hierzu wird die Methode so lange selbst aufgerufen und bei jedem Aufruf ein Teil der Nummer entfernt, solange die Länge der Zahl größer als 1 ist. Wenn nur noch eine Ziffer übrig ist, wird mit der Methode GetPossibleStringsForDigit die gültigen Buchstaben gesucht die zu dieser Ziffer passen und zurückgegeben. Nach dem zurückgeben werden in einer verschachtelten Schleife alle Einträge aneinandergereiht und wiederum zurückgegeben. Hier ein kleines Beispiel zur Veranschaulichung:

Nummer: 123

1. Schritt: Aufruf Number2Strings(123)
    a. Zahl: 123
    b. Mögliche Einträge für die erste Ziffer: a, b, c
2. Schritt: Aufruf Number2Strings(23)
    a. Zahl: 23
    b. Mögliche Einträge für die erste Ziffer: d, e, f
3. Schritt: Aufruf Number2Strings(3)
    a. Zahl: 3
    b. Mögliche Einträge für die erste Ziffer: g, h, i

c. Keine weiteren Ziffern vorhanden daher Rückgabe

d. Rückgabe: {g, h, i}

4. Schritt: Rückgabe zu Schritt 2

a. Bekommt Einträge von Schritt 3 {g, h, i}

b. Reiht Einträge mit den Möglichen Werten für 2 aneinander

c. Rückgabe: {dg, dh, di, eg, eh, ei, fg, fh, fi}

5. Schritt: Rückgabe zu Schritt 1

a. Bekommt Einträge von Schritt 4 {dg, dh, di, eg, eh, ei, fg, fh, fi}

b. Reiht Einträge mit den Möglichen Werten für 1 aneinander

c. Rückgabe: ….

1.3.2 Quellcode

```
set<string> T9Converter::Number2Strings(const string& number){
    if(!IsNumberValid(number)){
        throw invalid_argument("The parameter number is empty or contains a invalid digit. Only from 2-9 allowed.");
    }
         if (number.length() != 1){
        set<string> entries;
        set<string> allChars = Number2Strings(number.substr(1));
        set<string> possibleEntries = GetPossibleStringsForDigit(number[0]);

        for (auto subChar : allChars){
            for (auto entry : possibleEntries){
                entries.insert(entry + subChar);
            }
        }
                return entries;
         }
         else {
        return GetPossibleStringsForDigit(number[0]);
         }
}
```

1.3.3 Tests

1.3.3.1 Keine Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Strings NoInput
Test: Number2Strings-NoInput
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.3.3.2 Ungültige Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Strings InvalidNumber
Test: Number2Strings-InvalidNumber
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.3.3.3 Verschiedene Nummern und ausgeben der Anzahl der Ergebnisse

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Strings ResultCount
Test: Number2Strings-ResultCount
Amount of Results for  '2345'= 81
Amount of Results for  '45374'= 324
Amount of Results for  '59483727'= 15552
```

1.3.3.4 Ausgabe aller Ergebnisse für eine übergebene Zahl

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Strings PrintResult
Test: Number2Strings-ResultCount
Results for '23':
ad
ae
af
bd
be
bf
cd
ce
cf
```

1.3.3.5 Die Nummer 5477 übergeben (nur ein Ausschnitt aller Einträge)

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Strings 5477
Test: Word2Number-5477
Results for '5477':
jgpp
jgpq
jgpr
jgps
jgqp
jgqq
jgqr
jgqs
jgrp
jgrq
jgrr
jgrs
jgsp
jgsq
jgsr
jgss
jhpp
jhpq
jhpr
jhps
```

1.3.4 Beurteilung des Verfahrens

Die Funktion Number2Strings ist ein bisschen komplizierter wie die Funktion Word2Number. Es ist mehr Zeit zum Generieren der einzelnen Einträge und ab einer gewissen Länge der übergebenen Zahl kommt es auch zu „bad_alloc“ Exceptions, da man ca. 3^n rechnen kann für die Einträge, wobei n die Anzahl der Ziffern ist. Es ist nicht genau 3 denn manche Mappings sind auch 4 Buchstaben.

1.4 Number2Words

1.4.1 Lösungsidee

Bei der Funktion Number2Words wird eine Nummer übergeben für die alle sinnvollen Wörter zurückgegeben werden. Hierzu wird vorher eine Wörterbuch Datei gelesen welche anschließend der Methode Number2Words übergeben wird. Dieses Wörterbuch wird in Form einer map übergeben. Der Schlüssel ist das Wort in Großbuchstaben und der Wert ist das Originalwort.
Der erste Schritt besteht darin, alle möglichen Kombinationen für die Nummer auszulesen. Dies geschieht mit Hilfe der Number2Strings Methode. Wenn alle Möglichkeiten gelesen sind wird über

diese iteriert, und jeweils eine Abfrage in der Map mit den Wörtern gemacht. Falls der Eintrag vorhanden ist, wird er der Ergebnismenge hinzugefügt. Am Schluss wird die gesamte Ergebnismenge zurückgegeben.

1.4.2 Quellcode

```
vector<string> T9Converter::Number2Words(const string& number, map<string, string> & wordDictionary){
    if(!IsNumberValid(number)){
        throw invalid_argument("The parameter number is empty or contains a invalid digit. Only from 2-9 allowed.");
    }
         vector<string> entries;
         auto allPossibleEntries = this->Number2Strings(number);
         for (auto entry : allPossibleEntries){
                  if (wordDictionary[StringToUpper(entry)] != ""){
                           entries.push_back(wordDictionary[StringToUpper(entry)]);
                  }
         }
         return entries;
}
```

1.4.3 Tests

1.4.3.1 Keine Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Words NoInput
Test: Number2Words-NoInput
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.4.3.2 Ungültige Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Words InvalidNumber
Test: Number2Words-InvalidNumber
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.4.3.3 Verschiedene Nummern und ausgeben der Anzahl der Ergebnisse

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Words ResultCount
Test: Number2Words-ResultCount
Amount of Results for  '2345'= 1
Amount of Results for  '45374'= 0
Amount of Results for  '59483727'= 0
```

1.4.3.4 Ausgabe aller Ergebnisse für eine übergebene Zahl

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Words PrintResult
Test: Number2Words-PrintResult
Results for '23':
ad
Bf
CD
cf
```

1.4.4 Beurteilung des Verfahrens

Auf Grund der Wahl einer Map mit Key ist die Effizienz extrem gestiegen. Durch den direkten Zugriff auf die einzelnen Elemente mittels Key wird das Ergebnis sehr schnell geliefert und dies führt auch zu einer guten Laufzeit. Durch die Tatsache, dass die einzelnen Kombination für die gegebene Zahl ermittelt werden muss und dies mit Hilfe der Funktion Number2Strings geschieht, bestehen hier die gleichen Probleme wie bei Number2Strings. Bei Zahlen mit vielen stellen wird es durch das

bestimmen der einzelnen Möglichkeiten entweder unerträglich langsam oder bricht mit einer „bad_alloc“ Exception ab.

1.5 Number2WordsByLength

1.5.1 Lösungsidee

Bei der Funktion Number2WordsByLength sollte eine effizientere Form der unter Punkt 1.4. beschriebenen Methode implementiert werden. Möglich ist das hier durch das Sortieren der einzelnen Wörter und Gruppieren dieser nach ihrer Länge. Der Funktion Number2WordsByLength wird also wieder eine map übergeben. Die map hat folgende Struktur:

map<int, map<string, string>>

Der Key-Value ist ein Integer und beinhaltet die Länge der Werte die im Value gespeichert sind. Die map welche im Value gespeichert wurde ist hier genau die gleiche wie bei Number2Words, wodurch eine effizientere Verwendung des vorhandenen Quellcodes möglich wird, da in der Funktion Number2WordsByLength nur mittels Länge der übergebenen Zahl auf die map zugegriffen werden muss welche dieser Länge entspricht. Zurückgegeben werden dann, wie erwähnt alle Einträge die von der Länge her genau der gewünschten Nummer entsprechen. Diese Einträge werden schließlich der Funktion Number2Words übergeben welche dann die weitere Verarbeitung übernimmt. Wir erzielen dadurch einen enormen Performancevorteil, da wir nur noch auf wenige Elemente vergleichen müssen bzw. nur noch auf die Elemente welche der Länge der übergebenen Nummer entsprechen.

1.5.2 Quellcode

```
vector<string> T9Converter::Number2WordsByLength(const string& number, map<int, map<string, string>> &
wordDictionary){
        return this->Number2Words(number, wordDictionary[number.length()]);
}
```

1.5.3 Tests

1.5.3.1 Keine Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2WordsByLength NoInput
Test: Number2WordsByLength-NoInput
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.5.3.2 Ungültige Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2Words InvalidNumber
Test: Number2Words-InvalidNumber
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.5.3.3 Verschiedene Nummern und ausgeben der Anzahl der Ergebnisse

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2WordsByLength ResultCount
Test: Number2WordsByLength-ResultCount
Amount of Results for  '2345'= 1
Amount of Results for  '45374'= 0
Amount of Results for  '59483727'= 0
```

1.5.3.4 Ausgabe aller Ergebnisse für eine übergebene Zahl

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2WordsByLength PrintResult
Test: Number2WordsByLength-PrintResult
Results for '23':
ad
Bf
CD
cf
```

-

1.5.4 Beurteilung des Verfahrens

Die Effizienz dieses Verfahrens zu dem Verfahren unter Number2Words ist um einiges besser. Es arbeitet im Prinzip mit dem gleichen Algorithmus, jedoch wird nie das gesamte Wörterbuch überprüft sondern immer nur die Einträge welche der Länge der gegebenen Zahl entsprechen. Dadurch haben wir einen enormen Performance – Gewinn vor allem bei sehr großen Wörterbuch Dateien.

1.6 NumberPrefix2Word

1.6.1 Lösungsidee

Bei der Funktion NumberPrefix2Word sollten alle Wörter im Wörterbuch zurückgegeben werden, welche mit der gleichen T9 Zahlenkette anfangen. Dies ist vor allem für die Usability besser. Implementiert wird diese Funktion mittels einem set<string>. Da Sets wie binäre Bäume aufgebaut sind kann man mit der Funktion lower_bound alle Einträge im Wörterbuch, welche mit einem bestimmten Prefix beginnen auslesen. Zuerst wird dazu wieder die Funktion Number2Strings aufgerufen und liefert uns alle möglichen Prefixes. Über diese Möglichkeiten wird schließlich iteriert und für jeden Eintrag wird die lower_bound Methode auf das Wörterbuch aufgerufen. Dies hat zur Folge, dass wir den ersten Eintrag im Wörterbuch bekommen welcher dem Prefix entspricht. Der nächste Schritt besteht darin, alle Einträge auszulesen und den Iterator so lange zu erhöhen, bis der Eintrag auf den der Iterator verweist gleich ist wie der Prefix nur der letzte Buchstabe um eins erhöht. Dadurch bekommen wir auf sehr effiziente Art und Weise alle Einträge mit dem gewünschten Prefix.

1.6.2 Quellcode

```
vector<string> T9Converter::NumberPrefix2Word(const string& number, set<string> & wordDictionary){
    if(!IsNumberValid(number)){
        throw invalid_argument("The parameter number is empty or contains a invalid digit. Only from 2-9 allowed.");
    }
         vector<string> entries;
         auto allPossibleEntries = this->Number2Strings(number
         for (auto entry : allPossibleEntries){
        auto iteratorEntry = wordDictionary.lower_bound(entry);
        if(!HasGivenStringDifferentCharsThanZ(entry)){
            while(iteratorEntry != wordDictionary.end()){                entries.push_back(*iteratorEntry);
                ++iteratorEntry;
            }
        }
        else{
            while(iteratorEntry != wordDictionary.lower_bound(IncrementString(entry))){
                 entries.push_back(*iteratorEntry);
                 ++iteratorEntry;
            }
        }
          }
         return entries;
}
```

1.6.3 Tests

1.6.3.1 Keine Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2Word NoInput
Test: NumberPrefix2Word-NoInput
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.6.3.2 Ungültige Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2Word InvalidNumber
Test: NumberPrefix2Word-InvalidNumber
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.6.3.3 Verschiedene Nummern und ausgeben der Anzahl der Ergebnisse

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2Word ResultCount
Test: NumberPrefix2Word-ResultCount
Amount of Results for  '2345'= 131
Amount of Results for  '45374'= 0
Amount of Results for  '59483727'= 0
```

1.6.3.4 Ausgabe der Ergebnisse für eine übergebene Zahl (nur ein Ausschnitt der Ergebnisse)

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2Word PrintResult
Test: NumberPrefix2Word-PrintResult
Results for '4355':
geklafft
geklagt
geklammert
geklammerte
geklammertem
geklammerten
geklammerter
geklammertes
geklappert
geklappt
geklatscht
geklatschte
geklaut
geklaute
geklautem
geklauten
geklauter
geklautes
geklebt
```

1.6.4 Beurteilung des Verfahrens

Die Funktion NumberPrefix2Words ist auf Grund der Tatsache, dass ein set() wie ein binärer Baum aufgebaut ist ausgezeichnet. Es wird direkt auf das erste gewünschte Element zugegriffen und der Range kann leicht bestimmt werden in dem sich der Prefix befindet.

1.7 NumberPrefix2SortedWords

1.7.1 Lösungsidee

Die Funktion NumberPrefix2SortedWords funktioniert ähnlich wie die NumberPrefix2Word. Es sollte in dieser Funktion also nur die NumberPrefix2Word aufgerufen werden und danach neu sortiert werden. Das sortieren wird mittels der sort() Funktion aus der STL erledigt. Dazu wird eine Lambda Expression übergeben welche in einem Wörterbuch, in welchem alle Einträge mit einem Wert versehen sind, wie oft der Eintrag vorkommt, überprüft ob der aktuelle Eintrag größer ist als der nächste. Die sort() – Funktion übernimmt dabei die gesamte Sortierung. Die Anzahl der Einträge wird an Hand des Romans „Das Parfum“ von Patrick Süßkind erfasst. Der Text hat ca. 79.000 Wörter und sollte somit ein guter Indikator sein, wie oft welche Wörter vorkommen. An Hand der Tests sollte man dann den Vergleich zwischen NumberPrefix2Word und NumberPrefix2SortedWords gut erkennen, da die Sortierung eine andere ist.

1.7.2 Quellcode

```
vector<string> T9Converter::NumberPrefix2SortedWords(const string& number, set<string>
& wordDictionary, map<string,int, IgnoreCaseCmp> & wordDictionaryWithCount){
	vector<string> entries = this->NumberPrefix2Word(number, wordDictionary);
	sort(entries.begin(), entries.end(), [&](const string& a, const string& b) ->
bool {
		return wordDictionaryWithCount[a] > wordDictionaryWithCount[b];
    });
    return entries;
}
```

1.7.3 Tests

1.7.3.1 Keine Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test Number2WordsByLength NoInput
Test: Number2WordsByLength-NoInput
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.7.3.2 Ungültige Nummer übergeben

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2SortedWords InvalidNumber
Test: NumberPrefix2SortedWords-InvalidNumber
Invalid Argument exception was caught. Exception text was: The parameter number is empty or contains a invalid digit. Only from
2-9 allowed.
```

1.7.3.3 Verschiedene Nummern und ausgeben der Anzahl der Ergebnisse

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2SortedWords ResultCount
Test: NumberPrefix2SortedWords-ResultCount
Amount of Results for  '2345'= 131
Amount of Results for  '45374'= 0
Amount of Results for  '59483727'= 0
```

1.7.3.4 Ausgabe der Ergebnisse für eine übergebene Zahl (nur ein Ausschnitt der Ergebnisse)

```
stefan@ubuntu:~/Documents/Uebungen/Uebung1/bin/Release$ ./Uebung1 Test NumberPrefix2SortedWords PrintResult
Test: NumberPrefix2SortedWords-PrintResult
Results for '4355':
heller
hell
hellem
gellte
hellsten
geklopft
hellstem
helllichten
geklagt
```

1.7.4 Beurteilung des Verfahrens

Im Gegensatz zu Number2Words liefert diese Funktion Werte welche auch oft verwendet werden, daher ist diese Funktion der Funktion Number2Words vorzuziehen. Wenn man z.B. wirklich einen Konverter für ein Handy implementieren würde, könnte man statt der fixen Datei zum Bestimmen der Anzahl der Elemente eine variable Datei verwenden in der bereits verwendete Wörter gespeichert werden und dadurch ein noch persönlichere und individuellere Sortierung der Wörter gewährleistet werden kann.