13 Reguläre Ausdrücke

13.1 Übungsaufgabe

13.1.1 Aufgabe 1

Schreiben Sie ein C++ Programm, das eine txt-Datei (UTF8 kodiert) öffnet und mit `wcin` alle Wörter einliest, Punktuationszeichen am Anfang und Ende des eingelesenen Wortes entfernt, die Wörter in einer Frequenzliste speichert und anschließend nach der Häufigkeit sortiert ausgibt. Im zweiten Teil der Aufgabe sollen reguläre Ausdrücke eingesetzt werden.

Sie lesen aus der UTF8 Datei die einzelnen Wörter ein, entfernen Punktuationszeichen am Anfang und Ende und tragen Sie in einen Container `unordered_map <wstring> freq_list` ein.

Speichern Sie die gesäuberten Wörter in der Frequenzliste `freq_list` und zählen Sie die Frequenz der Wörter.

Nun wissen Sie, dass der Container `freq_list` zwar die Häufigkeit zählt, aber nicht numerisch aufsteigend sortiert ist. Dies können Sie mit einem Container `multimap` lösen. Der Trick eine `unordered_map` zu sortierten besteht darin alle Elemente mit Ihrer Häufigkeit in einen Container `multimap` zu übertragen.

Folgendermaßen können Sie vorgehen:
Übertragen sie alle Elemente der `unordered_map <wstring> freq_list` zusammen mit Ihrer Häufigkeit in eine `multimap<int,wstring>`. Beachten Sie, dass der Key der `multimap` nun die Häufigkeiten sind. Da eine `multimap` immer nach dem Key geordnet ist, müssen Sie sie nur der Reihe nach ausgeben und schon haben Sie eine sortierte Frequenzliste. Geben Sie von der die sortierten Wortliste die 5 häufigsten aus!

13.1.2 Aufgabe 2

Nun noch eine Aufgabe mit Regulären Ausdrücken: Wiederholen Sie Aufgabe 1

Sie lesen wieder aus einer Datei, aber diesmal lesen Sie zeilenweise mit `getline` und splitten die Zeile nach Wörtern mit Hilfe der split-Methode von Boost. Sie entfernen die Punktuationszeichen der einzelnen Wörter mit Boost und fügen dann die Wörter, die weniger als 4 Buchstaben haben (auch wieder mit Hilfe von boost), in die Frequenzliste ein. Geben Sie von der die sortierten Wortliste die 5 häufigsten aus!

13.2 Wiederholung und zusätzliche Informationen

13.2.1 C++11 in Eclipse verwenden

- Neues C++ Projekt erstellen
- Neue C++ Klasse erstellen und Code einfügen
- Rechtsklick auf das Projekt - Properties
- C/C++ Build - Settings - Tool Settings - GCC C++ Compiler - Miscellaneous - Other Flags: Am Ende Einfügen `-std=c++0x` und mit Apply bestätigen
- C/C++ General - Paths and Symbols - Symbols - GNU C++ - Add `__GXX_EXPERIMENTAL_CXX0X__` und mit Apply bestätigen
- Projekt neu builden (mit Hammer) und ausführen
- Bitte nicht wundern, die Fehler werden immer noch rot markiert, aber es ist ausführbar

13.2.2 Boost auf Cip-Pool-Rechnern installieren

- Leider habt ihr auf Cip Pool Rechnern keine Rechte um sudo zu verwenden, daher müsst ihr Boost manuell installieren
- Neueste Boost Version downloaden: http://www.boost.org/users/history/ und an geeigneter Stelle entpacken
- In der Konsole in den Boost Ordner wechseln
- `./bootstrap.sh --prefix=` eingeben und bei `prefix=` den Pfad eingeben unter dem Boost installiert werden soll
- `./b2 install` eingeben und Kaffee trinken gehen - das dauert eine Weile

13.2.3 Boost in Eclipse korrekt einstellen

- Projekt wählen und mit Rechtsklick Properties auswählen
- Unter C/C++ General - Paths and Symbols - Symbols - GNU C++ - Add: den Pfad zu eurem installierten Boost einfügen (im Tab Includes)
- Unter dem selben Pfad im Tab Libraries mit Add die Bibliothek `boost_regex` eingeben
- Mit Apply bestätigen und Projekt neu builden

13.3 Lösungsvorschlag

13.3.1 Aufgabe 1

```
/*Autor: Nicola Greth
 * Uebung 12
 * Frequenzliste ohne Regex
 */
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <map>
using namespace std;

void rm_punct(wstring &word);

int main() {
   // Locale Einstellungen
   setlocale(LC_ALL,"de_DE.utf-8");

   //Datei wird eingelesen
   locale utf8("de_DE.utf-8");
   wifstream input("derIdiot.txt");
   input.imbue(utf8);

   //Fehlermeldung wenn Datei nicht existiert
   if(!input){
      wcerr << L"File existiert nicht!!!"<< endl;
      return (-1);
    }

   //freq_list vom Typ unordered_map wird erstellt
   unordered_map<wstring, int> freq_list;
   wstring word;

   //durch >> wird am Space gesplittet
   while(input >> word){

      //Punktuationszeichen werden entfernt
      rm_punct(word);

      //wenn das Wort nicht gefunden wurde
      if (freq_list.find(word)==freq_list.end()){

         //füge das Wort mit der Anzahl 1 ein
         freq_list.insert(make_pair(word,1));
      }

      //wenn das Wort schon enthalten ist
      else {

         //erhöhe die Anzahl
         freq_list.at(word)++;
      }
   }
```

```
    input.close();

    //Iterator über freq_list
    unordered_map< wstring, int>::iterator myIter;

    //Multimap wird erstellt - ACHTUNG: da wir nach den Häufigkeiten sortieren
    //möchten, muss der "Key" die Häufigkeit sein
    multimap<int,wstring> freq_list_multi;

    //Alle Paare des unordered_map werden in die multimap eingetragen
    for(myIter=freq_list.begin(); myIter!=freq_list.end();++myIter){
       freq_list_multi.insert(make_pair(myIter->second, myIter ->first));
     }

    //Reverse Iterator über freq_list_multi wird deklariert um die häufigsten
    //Wörter zuerst auszugeben
    multimap<int,wstring>::reverse_iterator myIterRev=freq_list_multi.rbegin();;
    int counter = 0;

    //freq_list_multi wird von hinten durchlaufen und die 5 häufigsten Wörter
    //werden mit ihren Wahrscheinlichkeiten ausgegeben
    while((myIterRev != freq_list_multi.rend()) and (counter < 5)){
       wcout << myIterRev->first << L" : " << myIterRev->second << endl;
       counter ++;
       myIterRev++;
    }
    return 0;
}

//Methode rm_punct aus Ausgabe 10 und 11
void rm_punct(wstring &word){
    wstring::iterator first=word.begin();

    while(iswpunct(*first)){
       word.erase(first);
    }

    wstring::iterator last=word.end()-1;

    while(iswpunct(*last)){
       word.erase(last);
       last=word.end()-1;
    }
}
```

13.3.2 Aufgabe 2

```
/*Autor: Nicola Greth
 * Uebung 12
 * Frequenzliste mit Regex
 */
```

```
#include <iostream>

//Regex und String werden importiert
#include <boost/regex.hpp>
//Alternative: #include <regex> verwendet Regex von C++ 11
#include <boost/algorithm/string.hpp>

#include <fstream>
#include <unordered_map>
#include <map>
#include <vector>
using namespace std;
//namespace für boost wird gesetzt
using namespace boost;

int main(){
   // Locale Einstellungen
   setlocale(LC_ALL,"de_DE.UTF-8");

   //Textfile wird eingelesen
   locale utf8("de_DE.utf-8");
   wifstream input("derIdiot.txt");
   input.imbue(utf8);

   //Fehlermeldung falls Datei nicht existiert
   if(!input){
      wcerr << L"File existiert nicht!!!"<< endl;
      return (-1);
   }

   //freq_list wird erstellt
   unordered_map<wstring,int> freq_list;
   wstring zeile;

   // input wird zeilenweise eingelesen, diesmal mit getline
   while(getline(input, zeile)){
      vector<wstring> words;

      //Split mit der Split-Methode von boost
      split(words, zeile, is_any_of(" "), token_compress_on);
      //(wohin - Wörter werden im Vektor gespeichert,
      //was - die Zeile,
      //woran - immer am Space wird gesplittet,
      //zusätzliche Eigenschaft: verbindet benachbarte Token)

      //Wort-Vektor wird durchlaufen
      vector<wstring>::iterator v_iter;

      for(v_iter=words.begin(); v_iter!=words.end(); ++v_iter){

         //Regex für Punktuationszeichen an Anfang und Ende
```

```
            wregex punctAnfang(L"^[[:punct:]]*");
            wregex punctEnde(L"[[:punct:]]*$");
            wstring leer(L"");

            //aktuelles Wort wird ausgelesen
            wstring word = *v_iter;

            //Regex punctAnfang und punctEnde wird in word durch den leeren
            //String ersetzt, also gelöscht

            //regex_replace (von was - word, was soll ersetzt werden -
            //punctAnfang, mit was - leer)
            word = regex_replace(word, punctAnfang, leer);
            word = regex_replace(word, punctEnde, leer);

            //wenn das Wort weniger als 4 Buchstaben hat, wird es in die
            //Frequenzliste eingetragen
            wregex vierBuchstaben(L"(\\w{1,3})");

            if (regex_match(word,vierBuchstaben)){

               if (freq_list.find(word)==freq_list.end()){
                  freq_list.insert(make_pair(word,1));
               } else {
                  freq_list.at(word)++;
               }
            }
         }
      }
      input.close();

      //UnorderedMap wird in Multimap konvertiert
      multimap<int,wstring> freq_list_multi;

      for( unordered_map< wstring, int>::iterator iter=freq_list.begin();
      iter!=freq_list.end();++iter){
         freq_list_multi.insert(make_pair(iter->second, iter ->first));
      }

      //Reverse Iterator wird deklariert um die häufigsten Wörter zuerst
      //auszugeben
      multimap<int,wstring>::reverse_iterator myIterRev=freq_list_multi.rbegin();
      int counter = 0;

      while((myIterRev != freq_list_multi.rend()) and (counter < 5)){
         wcout << myIterRev->first << L" : " << myIterRev->second << endl;
         counter ++;
         myIterRev++;
      }
      return 0;
   }
```