11 Vektoren und Iteratoren

11.1 Übungsaufgabe

Wir arbeiten mit der Standard Template Library und verwenden einen Vector vom Typ `<wstring>` zur Programmierung der Klasse Lexikon. Verwenden Sie bei Schleifen grundsätzlich nur Iteratoren, testen Sie auch `auto` getypte Iteratoren.

Schreiben Sie ein C++ Programm, das Wörter über die Konsole einliest und sie in einem Lexikon `words` speichert und sortiert die ersten `n` Worte ausgeben kann. Dazu definieren Sie sich eine Klasse Lexikon. Die Idee ist, dass diese Klasse in einem `wstring` Vector alle Wörter des Lexikons speichert.

```
public:
    bool is_element(wstring);
    void insert(wstring word);
    void sort_me()
    void print_number_of_words();
    void head(int n);

private:
    void rm_punct(wstring &word);
    vector<wstring> words;
    int number_of_words;
```

- Die Klasse Lexikon definiert eine private Methode `void rm_punct(wstring &word)` die alle Piktuationszeichen am Anfang und Ende des eingegebenen Strings `word` entfernt.
- Die Klasse Lexikon definiert eine public Methode `bool is_element(wstring word)`, die testet, ob der eingebene `wstring word` schon im Lexikon vorkommt. (Entfernen Sie innerhalb dieser Methode zuerst mit `rm_punct(wstring &word)` alle Piktuationszeichen am Anfang und Ende des eingegebenen Strings und verwenden Sie dann die STL-Routine `find`). Falls das Wort nicht vorkommt, wird der Wert `false` zurückgeliefert, ansonsten `true`.
- Schreiben Sie eine public Methode `bool insert(wstring word)`, die zuerst alle Piktuationszeichen am Anfang und Ende des eingegebenen Worts entfernt (verwenden Sie die private Methoden) und das Wort nur dann im Container `words` speichert, wenn es noch nicht vorkommt. Wenn das Wort eingefügt wird, liefern Sie den Wert `true`

zurück. Schreiben Sie eine Methode `print_number_of_words()`, die ausgibt, wieviele Wörter im Lexikon sind.

- Schreiben Sie eine Methode `sort_me()`, die das Lexikon alphabetisch sortiert. Verwenden Sie STL-Routine `sort()`.
- Schreiben Sie eine Methode `void head(int n)`, die Elemente im Container sortiert und die ersten `n` Elemente des sortierten Lexikons ausdrückt.

11.2 Wiederholung und zusätzliche Informationen

11.2.1 Iteratoren

Iteratoren werden benötigt um auf die Elemente in Containern zuzugreifen.

- Ein Iterator beginnt am ersten Element des Containers und merkt sich die Position, an der er sich befindet
- `list<int>::iterator myIter;`: Iterator wird erstellt (`Container::iteratorname;`)
- `myIter = myVector.begin();`: erstes Element des Containers
- `myIter = myVector.end();`: Position hinter dem letzten Element des Containers
- `*myIter`: Zugriff auf das momentane Element mit `*`
- `++myIter`: Gehen zum nächsten Element mit `++`
- `myIter == otherIter`: Prüfen ob 2 Iteratoren auf das selbe Objekt zeigen mit `==` und `!=`

Beispiel:

```
#include <iostream>
#include <vector>
#include <sstream>
using namespace std;

//Beispielprogramm für Vector und Iterator
int main() {
    vector<wstring> myVector;
    wstring zeile;
    wstring word;
    wcout << "Bitte geben Sie eine Zeile ein: " << endl;
    getline(wcin, zeile);
    wstringstream myStream;
    myStream << zeile;

    //Vector wird gefüllt
```

```
while (myStream >> word) {
    myVector.push_back(word);
}

//Iterator wird erstellt und myVector wird durchlaufen
vector<wstring>::iterator myIter;

for (myIter = myVector.begin(); myIter != myVector.end(); ++myIter) {
    wcout << "Aktuelles Element : " << *myIter << endl;
}
}
```

11.3 Lösungsvorschlag

```
/*Autor: Nicola Greth
 * Uebung 10
 * Lexikon.hpp
 */
#ifndef LEXIKON_HPP
#define LEXIKON_HPP
#include <iostream>
#include <vector>
#include <locale>
#include <algorithm>
using namespace std;

class Lexikon {
public:
    Lexikon();
    bool is_element(wstring word);
    bool insert(wstring word);
    void sort_me();
    void print_number_of_words();
    void head(int n);

private:
    void rm_punct(wstring &word);
    vector<wstring> words;
    int number_of_words;
};

//Konstruktor, erstellt ein leeres Lexikon
Lexikon::Lexikon() {
    number_of_words = 0;
}

;

//Funktion, die Punctuationszeichen aus am Anfang und Ende von word entfernt
void Lexikon::rm_punct(wstring &word) {
```

```
//solange am Anfang Punctuationszeichen sind, wird der Anfang gelöscht
wstring::iterator first = word.begin();

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

//solange am Ende Punctuationszeichen sind, wird das Ende gelöscht
wstring::iterator last = word.end() - 1;

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

//Funktion insert, die word unter bestimmten Bedingungen ins Lexikon einträgt
bool Lexikon::insert(wstring word) {
    rm_punct(word);

    //wenn word nicht schon im Lexikon ist
    if (!is_element(word)) {

        //trägt das Wort ins Lexikon ein
        words.push_back(word);
        number_of_words++;
        return true;
    }
    return false;
}
;

//Funktion is_element, die testet ob word schon element im Lexikon ist
bool Lexikon::is_element(wstring Wort) {
    rm_punct(Wort);

    //sucht in words nach Wort
    vector<wstring>::iterator pos = find(words.begin(), words.end(), Wort);

    //wenn die Iterator-Position hinter dem letzten Element ist,
    //ist es nicht enthalten
    if (pos == words.end()) {
        return false;
    }

    //sonst ist es enthalten
    else {
        return true;
    }
}
}
```

```
;

//Funktion, die die Anzahl der Wörter im Lexikon ausdrückt
void Lexikon::print_number_of_words() {
    wcout << L"Ihr Lexikon enthält " << number_of_words << L" Wörter."
    << endl;
}
;

//Funktion, die die Daten des Lexikons sortiert
void Lexikon::sort_me() {
    //Achtung sort sortiert nach ASCII Sortierung
    sort(words.begin(), words.end());
}
;

//Funktion, die die ersten n Elemente des sortieren Lexikons ausgibt
void Lexikon::head(int n) {
    sort_me();

    //gibt die Element aus bis n = 0 ist
    vector<wstring>::iterator myIter;

    for (myIter = words.begin(); myIter != words.end() && n != 0; ++myIter) {
        wcout << L"Element: " << *myIter << endl;
        n--;
    }
}
;

#endif

/*Autor: Nicola Greth
 * Uebung 10
 * Lexikon.cpp
 */
#include "Lexikon.hpp"
#include <locale>
using namespace std;

int main() {
    //setzt Codierung auf utf8
    setlocale(LC_ALL, "de_DE.UTF-8");

    //neues Lexikon
    Lexikon myLexikon;

    //Lexikon sollte 0 Elemente enthalten
    myLexikon.print_number_of_words();

    //Lexikon wird gefüllt
```

```
wstring word;
wcout << L"Bitte geben Sie ein Wort ein:" << endl;

while (getline(wcin, word) and word != L"") {
    myLexikon.insert(word);
    wcout << L"Bitte geben Sie ein Wort ein oder beenden Sie die Eingabe
mit Enter" << endl;
    word = L"";
}

//druckt Anzahl aus
myLexikon.print_number_of_words();

//Lexikon wird sortiert
myLexikon.sort_me();

//die ersten fünf Wörter des Lexikons werden ausgegeben
wcout << endl << L"Die ersten fünf Wörter des sortierten Lexikons: " <<
endl;

myLexikon.head(5);
}
```