

Aufgabe 1: Wörter aufräumen

Team-ID: 01144

Team: Legends

Bearbeiter/-innen dieser Aufgabe:

Christoph Waffler

23. November 2020

Lösungsidee.....	1
Umsetzung.....	3
Beispiele.....	5
raetsel0.txt	5
raetsel1.txt	5
raetsel2.txt	5
raetsel3.txt	6
raetsel4.txt	6
Quellcode	7

Lösungsidee

Ziel dieser Aufgabe ist es, einen lückenhaften Satz mithilfe eines nichtsortierten Wörterspeichers zu vervollständigen.

Die lückenhafte Satz besteht aus mehreren Wörtern, deren einzelne Buchstaben größtenteils oder sogar vollständig unbekannt sind und mit „_“ abgebildet sind.

Die Länge eines Wortes kann durch Zählen der Zeichen „_“ zusammen mit den teils vorhandenen Buchstaben bestimmt werden:

So hat das Wort „__t_“ die Länge 5 und an 4. Stelle den Buchstaben „t“. Das Wort könnte bspw. „heute“ bedeuten.

Im Folgenden wird eine Vorgehensweise erläutert, mit der dieses Problem gelöst werden kann.

Die lückenhaften Wörter sind in der Liste *lücken* gespeichert und so sortiert, dass der Satz ausgefüllt einen Sinn ergibt.

Im Wörterspeicher *speicher* sind alle Wörter gespeichert, die für eine Lücke als Lösung eingetragen werden können. *Speicher* ist nicht sortiert.

1. Bestimme für jedes lückenhafte Wort *w* von *lücken* die potenziellen Lösungswörter *pot_lösungen*:
 - a. Bestimme die Länge von *w* und speichere jene Wörter von *speicher* in *pot_lösungen*, die dieselbe Länge wie *w* besitzen.
 - b. Filtere *pot_lösungen*, sodass nur noch jene Wörter am Schluss übrig bleiben, die dieselben Buchstaben (falls vorhanden) von *w* an derselben Stelle wie *w* besitzen.

- c. Falls *pot_lösungen* eines lückenhaften Wortes nur ein Element besitzt, ist dies die Lösung für diese Lücke.
Entferne dieses Element aus *speicher*, da das Wort nun als Lösung für eine Lücke verwendet wird und nicht für zwei verschiedene Lücken benutzt werden kann.
2. Gehe nun alle *pot_lösungen* aller lückenhafter Wörter durch:
Falls *pot_lösungen* nur noch ein Element beinhaltet, ist dies die Lösung für diese Lücke.
Dieses Element wird von allen *pot_lösungen* aller lückenhafter Wörter entfernt.
3. Wiederhole Schritt 2 solange, bis alle *pot_lösungen* aller lückenhafter Wörter leer ist.

Es wird davon ausgegangen, dass alle Eingaben des Problems lösbar sind.

Falls eine Eingabe nicht eindeutig ist, kann es sein, dass die Lösung zu dieser Eingabe nicht korrekt ist.

Da nun allen Lücken ein Lösungswort zugewiesen wurde, wird der vollständige Satz ausgegeben.

Umsetzung

Die Lösungsidee wird in C++ implementiert. (Hinweis zum Ausführen des Programms: die Programmierung erfolgte mit dem OSX Betriebssystem von Apple).

Es wurden zwei Methoden *utf8_to_utf16* und *utf16_to_utf8* implementiert, die die Codierung des Strings von UTF-8 zu UTF-16 und umgekehrt durchführen. Diese sind nicht Gegenstand der Dokumentation.

Die 2 Zeilen der Eingabe im BwInf-Format werden mit *getline(...)* eingelesen.

Es werden zwei Vektoren *spaces* und *spaces_with_punctuations* mit Typ String deklariert. In beiden Vektoren werden die einzelnen Lücken des Satzes gespeichert.

In *spaces* wird die Lücke ohne Satzzeichen und in *spaces_with_punctuations* mit Satzzeichen gespeichert.

Ein Char eines Strings ist ein Satzzeichen, wenn *isalpha(c)* und der Vergleich *c == '_'* beide negativ zurückgeben.

Im Vector *words* werden alle Wörter gespeichert, die in der 2. Zeile vom Benutzer eingegeben werden. Der Int-Wert *n_words* gibt die Größe des Vectors *words* und somit die Anzahl der Wörter an.

Im Vector<Vector<String>> *zu_viele* mit der Größe *n_words* wird für jede Lücke eine Liste mit möglich passenden Wörtern gespeichert.

Im Vector<String> *new_sentence* wird der neue vollständige Satz gespeichert.

Nun folgt Schritt 1 der Lösungsidee:

Mithilfe einer for-Schleife (mit Laufvariable *i*) wird für jedes lückenhafte Wort *word* im Satz zuerst die Größe ermittelt.

Anschließend werden im Vector<String> *same_length* alle Wörter mit derselben Länge gespeichert. Falls die Länge gleich der Länge von *word* ist, wird dieses zu *same_length* hinzugefügt.

Für jeden Buchstaben in *word*, wird überprüft, ob dieser nicht , _ ' also einer Lücke entspricht.

Ist dies nicht der Fall so liegt ein Buchstabe vor. Dieser Buchstabe muss beim Lösungswort ebenfalls an derselben Stelle wie bei *word* sein. Daher werden mit der Methode *filter_words(...)* die Liste *same_length* so gefiltert, dass nur noch die Wörter vorhanden sind, bei denen dieses Kriterium zutrifft.

In der Methode *filter_words()* mit einem Vector<String> *list*, *char c* und *int index* als Parameter wird überprüft, ob alle Wörter von *list* den Buchstaben *c* an der Stelle *index* haben.

Ist dies der Fall, so werden diese zu einem neuen Vector<String> *new_list* hinzugefügt.

Der Vector *new_list* wird am Ende zurückgegeben.

Falls nun die Größe von *same_length* größer als 1 ist, so wird dieser Vector zum Vector<Vector<String>> *zu_viele* an der Stelle *i* zugewiesen.

Falls dies jedoch nicht der Fall ist, ist das einzig vorhandene Element in *same_length* die Lösung für diese Lücke und *new_sentence* wird an der Stelle *i* nun dieses Element zugewiesen.

Ebenfalls wird nun aus *zu_viele* dieses Element gelöscht (dies wird mit der Methode *remove_word_vvs(...)* durchgeführt, die jeden „Untervector“ iteriert und dort das Wort löscht).

Auch wird dieses Element vom Vector *words* entfernt, da es nicht mehr als Wort verwendet werden soll.

Nun folgt Schritt 2:

Mit einer for-Schleife wird nun von $i=0$ bis $i < n_words$ iteriert.

Falls der „Untervector“, der in *zu_viele* im Index *i* nicht leer ist, wird folgendes überprüft.

Ist die Größe dieses „Untervectors“ kleiner oder gleich 1, so ist nur noch ein Wort potenziell für diese Lücke an der Stelle *i* möglich. Dieses Wort wird *new_sentence* an der Stelle *i* zugewiesen.

Ebenfalls wird dieses Wort mithilfe der Methode *remove_words_vvs* von *zu_viele* entfernt.

Falls die Größe des „Untervectors“ jedoch größer als 1 ist, so wird überprüft, ob alle Elemente des „Untervectors“ dieselben sind. Falls dies der Fall ist, so ist ebenfalls das Lösungswort für die Lücke an der Stelle *i* gefunden. Dieses Wort wird auch von *zu_viele* entfernt und *new_sentence* an der Stelle *i* zugewiesen.

Schritt 3 der Lösungsidee besagt Schritt 2 solange zu wiederholen, bis *zu_viele* keine Wörter mehr beinhaltet. Dies wird mithilfe der *goto*-Anweisung realisiert, die zu Beginn der for-Schleife in Schritt 2 verweist.

Zum Schluss müssen nun noch die Satzzeichen zu den Wörtern in *new_sentence* hinzugefügt werden. Dafür wird die Methode *merge_vs(spaces_with_punctuations, new_sentence)* verwendet.

In dieser Methode wird der String *s1* von *spaces_with_punctuations* an der Stelle *i* mit dem String *s2* von *new_sentence* an der Stelle *i* verglichen. Falls *s1* länger als *s2* ist, wird das Ende von *s1*, das nicht in *s2* enthalten ist, an das Ende von *s2* hinzugefügt.

Am Ende enthält *new_sentence* nun alle aktualisierten Strings mit gegebenenfalls den entsprechenden Satzzeichen, die der Benutzer bei der Eingabe mit übergeben hat.

Jeder String von *new_sentence* wird mit *cout* ausgegeben.

Beispiele

Im Folgenden werden die Beispiele von der BwInf-Website mit dem Programm gelöst.
Die Ein- und Ausgabe erfolgt – wie bereits in der Umsetzung erwähnt – über die Kommandozeile.
Es ist die gesamte Ein- und Ausgabe mit abgebildet.

raetsel0.txt

>> Geben Sie in der 1. Zeile den lückenhaften Text ein
und in der 2. Zeile die dazugehörigen Wörter

h, _a_ _r_ _e_ _b_!

arbeit eine für je oh was

>> Folgender Satz wurde vervollständigt:

> oh je, was für eine arbeit!

raetsel1.txt

>> Geben Sie in der 1. Zeile den lückenhaften Text ein
und in der 2. Zeile die dazugehörigen Wörter

m _a_ _e_ _s_ _e_ . D_ _a_ _i_ _u_ _e_ _n_ _u_ _
l _h_ _h_ _e_ .

Am in als das Das die und sehr Leute viele wurde wurde Anfang machte wütend falsche Schritt
Richtung angesehen Universum erschaffen allenthalben

>> Folgender Satz wurde vervollständigt:

> Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde
allenthalben als Schritt in die falsche Richtung angesehen.

raetsel2.txt

>> Geben Sie in der 1. Zeile den lückenhaften Text ein
und in der 2. Zeile die dazugehörigen Wörter

s _e_ _a_ _e_ _n_ _u_ _m_ , _a_ _r_ s_ _n_ _m_ _
t _u_ _m_ _e_ _e_ .

er in zu Als aus Bett fand sich einem eines Samsa Gregor seinem Morgens Träumen erwachte
unruhigen Ungeziefer verwandelt ungeheueren

>> Folgender Satz wurde vervollständigt:

> Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu
einem ungeheueren Ungeziefer verwandelt.

raetsel3.txt

>> Geben Sie in der 1. Zeile den lückenhaften Text ein
und in der 2. Zeile die dazugehörigen Wörter

_____ t d _____ e _____ n e _____ g, e _____,
a _____ n _____ r _____ n o _____, _____ e _____ t _____
a _____ i _____.

der der die ist mit und von von besonders Informatik Darstellung Speicherung Übertragung
Verarbeitung Verarbeitung Wissenschaft Informationen automatischen systematischen
Digitalrechnern

>> Folgender Satz wurde vervollständigt:

> Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung
und Übertragung von Informationen, besonders der automatischen Verarbeitung mit
Digitalrechnern.

raetsel4.txt

>> Geben Sie in der 1. Zeile den lückenhaften Text ein
und in der 2. Zeile die dazugehörigen Wörter

a _____ n _____ t n i _____ e _____ s d _____ e _____ d i _____ e _____ t _____ . s s _____
n _____ e _____ n _____ n _____ e _____, i i i _____ h _____ e _____ b _____ d _____ s _____,
o s s _____ i _____ t _____ e _____ r _____ . _____ n n _____ a _____ s _____ e _____
_____ b _____ d _____ n _____ e _____.

Es in in so aus der die die ein ist Opa sie und und von dass eine eine sind einer Liste schon sowie
einige findet Jürgen Rätsel sollen werden ergeben gegeben lustige Wörtern Apotheke blättert
gebracht richtige Buchstaben Geschichte vorgegeben Leerzeichen Reihenfolge Satzzeichen
Zeitschrift

>> Folgender Satz wurde vervollständigt:

> Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste
von Wörtern gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine
lustige Geschichte ergeben. Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon
vorgegeben.

Quellcode*(Anmerkung: Es wurde der gesamte Quellcode eingefügt.)*

```
//  
  
// Created by Christoph Waffler on 01.09.20.  
  
//  
  
#include <bits/stdc++.h>  
  
#include <codecvt>  
  
#include <locale>  
  
using namespace std;  
  
  
#define vs vector<string>  
#define vvs vector<vector<string>>  
#define vi vector<int>  
  
  
// Konvertiert einen UTF-8 String zu UTF-16  
std::u16string utf8_to_utf16(std::string const& utf8)  
{  
    std::wstring_convert<std::codecvt_utf8_utf16<char16_t, 0x10ffff,  
        std::codecvt_mode::little_endian>, char16_t> cnv;  
    std::u16string s = cnv.from_bytes(utf8);  
    if(cnv.converted() < utf8.size())  
        throw std::runtime_error("incomplete conversion");  
    return s;  
}  
  
  
// Konvertiert einen UTF-16 String zu UTF-8  
std::string utf16_to_utf8(std::u16string const& s)  
{  
    std::wstring_convert<std::codecvt_utf8_utf16<char16_t, 0x10ffff,  
        std::codecvt_mode::little_endian>, char16_t> cnv;  
    std::string utf8 = cnv.to_bytes(s);  
}
```

```
    if(cnv.converted() < s.size())
        throw std::runtime_error("incomplete conversion");
    return utf8;
}

// Strings werden in einem Vector als Paramater übergeben.
// Die Methode gibt eine neuen Vector zurück, die all jene Strings enthält,
// die den char c and der Stelle index besitzen.
vs filter_words(const vs& list, char c, int index) {
    vs new_list;
    for (const string& word : list) {
        u16string w = utf8_to_utf16(word);
        if (w[index] == c) {
            new_list.push_back(word);
        }
    }
    return new_list;
}

vs merge_vs(vs punctuations, vs without_punctuations) {
    vs new_one;
    for (int i = 0; i < punctuations.size(); ++i) {
        u16string w = utf8_to_utf16(without_punctuations[i]);
        u16string ww = utf8_to_utf16(punctuations[i]);
        u16string cut (ww.end()-(ww.size()-w.size()), ww.end());
        u16string s = w+cut;
        string new_string = utf16_to_utf8(s);
        new_one.push_back(new_string);
    }
    return new_one;
}
```


// aus dem gegebenen 2D-Vector mit Strings, wird das mit übergebene Wort entfernt
// falls breakFirst = true ist, bricht die Suche in dem aktuellen Vector ab und der nächste wird
bearbeitet

```
vvs remove_word_vvs(vvs liste, const string& word, bool breakFirst=false) {  
    for (int i = 0; i < liste.size(); ++i) {  
        vs list = liste[i];  
        for (int j = 0; j < list.size(); ++j) {  
            string wort = list[j];  
            if (wort == word) {  
                liste[i].erase(liste[i].begin()+j);  
  
                if (breakFirst) {  
                    return liste;  
                }  
  
                // Einmal gefunden --> Abbruch aus dieser Unterliste --> nächste Unterliste  
                break;  
            }  
        }  
    }  
    return liste;  
}
```

// Prüfen, ob die Vektoren im Vector leer sind

```
bool check_vvs_empty(vvs liste) {  
    return all_of(liste.begin(), liste.end(), [](const vs& a) {return a.empty();});  
}
```

```
int main() {  
    ios::sync_with_stdio(false);
```

```
cin.tie(0);

cout << ">> Geben Sie in der 1. Zeile den lückenhaften Text ein \n";
cout << "und in der 2. Zeile die dazugehörigen Wörter" << endl;

// Einlesen der 1. und 2. Zeile
string lineOne, lineTwo;
getline(cin, lineOne);
getline(cin, lineTwo);

// Es werden zum einen die Lücken und die Lücken mit Satzzeichen gespeichert
vs spaces, spaces_with_punctuations;

// Vector zum Speichern aller Wörter
vs words;

// nun werden beide eingelesenen Zeilen verarbeitet
istringstream iss1(lineOne);
istringstream iss2(lineTwo);

// 1. Zeile
for (string s; iss1 >> s;) {
    // String mit Satzzeichen wird gespeichert
    spaces_with_punctuations.push_back(s);

    // alle nicht alphabetischen Zeichen oder '_' (=Lücken) werden entfernt
    string new_s;
    for (char c : s) {
        if (isalpha(c) or c == '_') {
            new_s.push_back(c);
        }
    }
}
```

```
    }  
}  
  
    // im String s befinden sich keine Satzzeichen  
    // speichern im Vector spaces  
    spaces.push_back(new_s);  
}  
  
    // alle Wörter aus der 2. Zeile werden im Vector words hinzugefügt  
    for (string s; iss2 >> s;) {  
        words.push_back(s);  
    }  
  
    // Anzahl der gesamten Wörter ist die Größe des Vector spaces  
    const int n_words = spaces.size();  
  
    // Vector zu_viele, um für alle Lücken die möglich passenden Wörter zu speichern  
    vvs zu_viele(n_words);  
  
    // Vector zum Speichern des neuen vollständigen Satzes  
    vs new_sentence(n_words);  
  
    // ein Durchgang der for-loop für jedes Wort im Satz  
    for (int i = 0; i < spaces.size(); ++i) {  
        string word = spaces[i];  
        int length = word.length();  
  
        // Speichern aller Wörter mit derselben Länge  
        vs same_length;  
        for (const string& w : words) {  
            u16string ww = utf8_to_utf16(w);
```

```
    if (ww.length() == length) {  
        same_length.push_back(w);  
    }  
}
```

// aus den vorher gespeicherten Wörtern mit derselben Länge

// werden nun diejenigen rausgefiltert, die dieselben Buchstaben am selben Index haben

vi index_chars;

```
for (int j = 0; j < word.length(); ++j){  
    char w = word[j];  
    if (w != '_' ) {  
        same_length = filter_words(same_length, w, j);  
    }  
}
```

```
if (same_length.size() > 1) {
```

// die potenziell passenden Wörter werden im Vector zu_viele an der Stelle der Lücke
im Satz gespeichert

```
    zu_viele[i] = same_length;  
} else {
```

// falls nur ein Wort passt, wird dies als "Lösung" für diese Lücke übernommen

```
string wort = same_length[0];
```

// Entfernen des Wortes aus dem Speicher für alle potenziell passenden Wörter

```
zu_viele = remove_word_vvs(zu_viele, wort);
```

// Zuweisen der Lösung

```
new_sentence[i] = same_length[0];
```

// Entfernen des Wortes aus dem Wörterspeicher

```
remove(words.begin(), words.end(), same_length[0]);
```

```
}
```

```
}
```

```
bool wiederholen;
```

```
LOOP:
```

```
for (int i = 0; i < n_words; ++i) {
```

```
    wiederholen = false;
```

```
    // falls der Speicher der potenziellen Wörter für diese Lücke nicht leer ist
```

```
    if (!zu_viele[i].empty()) {
```

```
        if (zu_viele[i].size() <= 1) {
```

```
            // falls nur noch ein Wort potenziell für diese Lücke möglich ist
```

```
            string wort = zu_viele[i][0];
```

```
            // das Wort wird nicht mehr als potenzielles Wort für andere Lücken angesehen und entfernt
```

```
            zu_viele = remove_word_vvs(zu_viele, wort);
```

```
            // Wort wird als Lösung für diese Lücke gespeichert
```

```
            new_sentence[i] = wort;
```

```
        } else {
```

```
            // check ob alle dasselbe Wort sind
```

```
            string s = zu_viele[i][0];
```

```
            for (const string& wort : zu_viele[i]) {
```

```
                if (s != wort) {
```

```
                    wiederholen = true;
```

```
                }
```

```
            }
```

```
    if (wiederholen) {
        continue;
    } else {
        // falls alle potenziell möglichen Wörter für diese Lücke dasselbe sind,
        // wird das erste als Lösung verwendet
        string wort = zu_viele[i][0];
        // das Wort wird von anderen Speichern für ein potenzielles Wort entfernt
        zu_viele = remove_word_vvs(zu_viele, wort);
        zu_viele = remove_word_vvs(zu_viele, wort, true);

        // Zuweisen des Lösungswortes für diese Lücke
        new_sentence[i] = wort;
        wiederholen = false;
    }
}

// die Loop (Schleife) wird solange wiederholt bis kein Wort mehr im Vector zu_viele
// und die boolsche Variable wiederholen = true ist
if (wiederholen or !check_vvs_empty(zu_viele)) {
    goto LOOP;
}

// der neue Satz mit den passenden Lösungen für die Lücken wird noch mit den
Satzzeichen zusammengefügt,
// sodass an der passenden Stelle jene Satzzeichen sind, die dort auch mit der Eingabe
übergeben wurden
new_sentence = merge_vs(spaces_with_punctuations, new_sentence);
```

```
// Ausgabe des neuen Satzes
cout << "\n\n>> Folgender Satz wurde vervollständigt: " << endl;
cout << "> ";

for (const string& w : new_sentence) {
    cout << w << " ";
}

}
```