C/C++ Programmierung
Übungsblatt 6

Florian Oetke

1 Wort-Histogramm

Schreiben Sie ein Programm das mit `std::cin` Wörter (d.h. mit Leerzeichen getrennte Zeichenfolgen), bis zur Eingabe `STOP` einliest und zählt wie oft jedes Wort vorkommt. Geben Sie anschließend alle Wörter und ihre Häufigkeit in beliebiger Reihenfolge, sowie das häufigste Wort mit `std::cout` aus.

Bei einer Eingabe von `Wenn wir uns zum Beispiel mal dieses Beispiel anschauen STOP` sollte die Ausgabe z.B. so aussehen:

```
dieses : 1
mal : 1
zum : 1
uns : 1
wir : 1
anschauen : 1
Beispiel : 2
Wenn : 1

Most common word: Beispiel
```

2 Bubblesort

Schreiben Sie eine Funktion `sort`, die eine Referenz auf einen `std::vector<int>` als Parameter bekommt und diesen mittels Bubblesort aufsteigend sortiert.

Das Sortierverfahren funktioniert über zwei verschachtelte Schleifen. Die innere Schleife durchläuft alle Elemente im Vector, prüft ob das linke Element kleiner ist als das rechte und vertauscht sie falls dies nicht der Fall ist. Das wird durch die äußere Schleife solange wiederholt, bis alle Elemente in der richtigen Reihenfolge sind. Das heißt solange bis wir die innere Schleife einmal durchlaufen haben ohne Elemente vertauschen zu müssen

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```
1  #include <iostream>
2  #include <vector>
3
4  // TODO
5
6  int main() {
7      auto data = std::vector<int>{0, 7, 9, 5, 8, 1, 2, 4, 3, 6};
8
9      sort(data);
10
11     for(auto& e : data) {
12         std::cout << e << " ";
13     }
14 }
```

Die Ausgabe des Programms sollte so aussehen:

```
0 1 2 3 4 5 6 7 8 9
```

3 Vektor Statistiken

Schreiben Sie eine Funktion `statistics`, die einen `std::vector<float>` als Parameter erhält und für diesen einige Statistiken berechnet und zurückgibt.

Der Rückgabewert soll ein `struct Statistics` sein, das Sie ebenfalls noch schreiben müssen und das folgende Member Variablen enthält:

- `min`: Der kleinste Wert im Vector
- `max`: Der größte Wert im Vector
- `avg`: Der Durchschnitt aller Werte im Vector
- `range`: Eine `enum class Number_range`, die Sie ebenfalls noch schreiben müssen und die angibt ob alle Werte >=0 (`positive`) bzw. <0 (`negative`) sind oder ob sowohl positive als auch negative Werte im Vector vorkommen (`mixed`)

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```
1  #include <iostream>
2  #include <vector>
3
4  // TODO
5
6  int main() {
7      const auto data = std::vector<float>{-20.f, 14.1f, 9.f, 9.f, 200.f};
8      const auto stats = statistics(data);
9
10     std::cout << "min   : " << stats.min << "\n"
11               << "max   : " << stats.max << "\n"
12               << "avg   : " << stats.avg << "\n"
13               << "range : ";
14
15     switch(stats.range) {
16         case Number_range::positive:
17             std::cout << "positive\n";
18             break;
19         case Number_range::negative:
20             std::cout << "negative\n";
21             break;
22         case Number_range::mixed:
23             std::cout << "mixed\n";
24             break;
25     }
26 }
```

4 Zeitpläne konvertieren

In dem unten (und im Handout) aufgeführten Programm finden Sie zwei Definitionen für unterschiedliche Zeitpläne. Einer der für jedes Fach, die Zeiten und Räume enthält, an denen Vorlesungen stattfinden und einen Raumplan, der für jeden Raum die Zeiten und Vorlesungen enthält, die dort stattfinden. Um die Aufgabe möglichst einfach zu halten wird für alle Angaben `std::string` verwendet.

Implementieren Sie die Funktion `to_room_schedule`, die einen fach-spezifischen Zeitplan in einen Raumplan übersetzt.

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```cpp
#include <iostream>
#include <utility>
#include <map>
#include <unordered_map>

using Subject = std::string;
using Time    = std::string;
using Room    = std::string;

using Class_schedule = std::unordered_map<Subject, std::unordered_map<Time, Room>>;
using Room_schedule  = std::map<Room, std::map<Time, Subject>>;

Room_schedule to_room_schedule(const Class_schedule& class_schedule) {
    // TODO
}

int main() {
    const auto class_schedule = Class_schedule{
        {"C/C++", {
            {"Mo. 20:00", "3B"},
            {"Fr. 14:00", "L402"}
        }},
        {"Java", {
            {"Di. 06:15", "HS1"},
            {"Di. 08:00", "HS1"}
        }},
        {"The Mathematics of Quantum Neutrino Fields", {
            {"Mo. 17:30", "3B"}
        }}
    };

    const auto room_schedule = to_room_schedule(class_schedule);

    for(auto&&[room, timeslots] : room_schedule) {
        std::cout << room << ":\n";
        for(auto&&[time, subject] : timeslots) {
            std::cout << "\t" << time << " " << subject << "\n";
        }
    }
}
```

Die Ausgabe des Programms sollte so aussehen:

```
3B:
    Mo. 17:30 The Mathematics of Quantum Neutrino Fields
    Mo. 20:00 C/C++

HS1:
    Di. 06:15 Java
    Di. 08:00 Java

L402:
    Fr. 14:00 C/C++
```

5 Sudoku

In dieser Aufgabe soll ein Algorithmus implementiert werden, der zulässige Lösungen für ein Sudoku-Rätsel bestimmt. Diese besten aus einem Raster von 9 mal 9 Feldern, die so mit den Zahlen von 1 bis 9 befüllt werden müssen das jede Zahl pro Zeile, Spalte und 3x3-Block nur einmal vorkommt.

Hierzu finden Sie im Handout bereits ein vorbereitetes Programm das ein Sudoku-Feld, wie in der Vorlesung gezeigt, als 3D-Array speichert und wie unten abgebildet einlesen und ausgeben kann.

Das erste was Sie implementieren müssen ist eine Funktion `valid_move`, die prüft ob es zulässig wäre in ein bestimmtes Feld eine festgelegte Zahl einzutragen. Das heißt ob es die Zahl bereits in der entsprechenden Zeile, Spalte oder in dem 3x3-Block[1] gibt.

Der Algorithmus selbst soll in der Funktion `solve` implementiert werden und funktioniert indem rekursiv die folgenden Schritte durchgeführt werden:

1. Zuerst müssen Sie alle Felder durchlaufen, bis Sie ein freies Feld (mit dem Wert 0) finden. Falls es kein freies Feld mehr gibt, haben wir eine Lösung gefunden, können diese mit `std::cout << sudoku` ausgeben und die Funktion mit `return true;` verlassen.

2. Wenn wir ein freies Feld gefunden haben, durchlaufen wir alle Zahlen von 1 bis 9 und prüfen mit `valid_move` ob wir sie dort einsetzen können. Wenn keine der Zahlen erlaubt ist, gibt es keine Lösung und wir brechen die Funktion mit `return false;` ab.

3. Wenn wir die Zahl einsetzen dürfen, schreiben wir sie an die entsprechende Stelle im Feld und rufen `solve` für das neue Sudoku-Feld auf.

4. Falls der rekursive Aufruf von `solve` nicht erfolgreich war, gibt es keine Lösung mehr, wenn wir die Zahl an dieser Stelle einsetzen und wir müssen die Änderung rückgängig machen (d.h. wieder 0 an die Stelle schreiben) und es mit der nächsten Zahl versuchen.

5. Falls der rekursive Aufruf von `solve` erfolgreich war, wurde eine Lösung gefunden und wir können die Funktion ebenfalls mit `return true;` verlassen.

Das Programm erwartet als Eingabe ein Sudoku-Feld (wie unten) und sollte die erste gefundene Lösung (wie darunter) ausgeben:

```
 3 1 8 | 4 0 0 | 5 0 0
 0 0 0 | 6 0 0 | 2 0 3
 0 4 6 | 9 0 5 | 0 0 0
---------------------
 4 0 5 | 0 0 0 | 0 0 0
 0 0 0 | 0 4 0 | 1 7 6
 0 0 0 | 0 8 6 | 0 5 0
---------------------
 0 0 0 | 3 6 9 | 0 0 5
 0 0 0 | 0 0 0 | 3 0 7
 9 0 2 | 0 0 4 | 0 8 0
```

```
 3 1 8 | 4 7 2 | 5 6 9
 5 9 7 | 6 1 8 | 2 4 3
 2 4 6 | 9 3 5 | 7 1 8
---------------------
 4 6 5 | 1 9 7 | 8 3 2
 8 2 9 | 5 4 3 | 1 7 6
 1 7 3 | 2 8 6 | 9 5 4
---------------------
 7 8 1 | 3 6 9 | 4 2 5
 6 5 4 | 8 2 1 | 3 9 7
 9 3 2 | 7 5 4 | 6 8 1
```

[1] Hinweis: Um den Anfang vom entsprechenden Block pro Koordinate zu finden, können Sie z.B. mit $x - x\%3$ das nächst kleinere Vielfache von drei berechnen.

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden (den Sie ebenfalls in der Handout-ZIP-Datei finden):

```cpp
#include <iostream>
#include <array>
#include <sstream>

struct Sudoku {
    std::array<std::array<int, 9>, 9> field = {};
};

std::ostream& operator<<(std::ostream& out, const Sudoku& s) {
    for(int y=0; y<9; y++) {
        if(y%3==0 && y>0) {
            std::cout << "-----------------------\n";
        }
        for(int x=0; x<9; x++) {
            if(x%3==0 && x>0) {
                std::cout << " |";
            }
            std::cout << " " << s.field[x][y];
        }
        std::cout << "\n";
    }
    return out;
}

std::istream& operator>>(std::istream& in, Sudoku& s) {
    auto ignore = std::string();

    for(int y=0; y<9; y++) {
        if(y%3==0 && y>0) {
            in >> ignore;
        }
        for(int x=0; x<9; x++) {
            if(x%3==0 && x>0) {
                in >> ignore;
            }
            in >> s.field[x][y];
        }
    }
    return in;
}


// TODO: bool valid_move(const Sudoku& sudoku, int x, int y, int number)
// TODO: bool solve(Sudoku sudoku)

int main() {
    auto sudoku = Sudoku{};

    std::cin  >> sudoku;
    std::cout << sudoku << "\n";

    if(!solve(sudoku)) {
        std::cout << "Keine Lösung gefunden\n";
    }
}
```