A5: Sudoku (2 Wochen)

Neue Elemente von C++: Klassen, Attribute, Methoden, Operatorüberladung, Zugriffskontrolle

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

Bei einem Sudoku-Rätsel sucht man für jedes freie Feld eine Ziffer zwischen 1 und 9 so, dass folgende Regeln erfüllt sind: In jeder Zeile, in jeder Spalte und in jedem dick umrandeten Teilquadrat kommt jede der Ziffern von 1 bis 9 genau einmal vor. Wir wollen in dieser Aufgabe eine Klasse `Sudoku` programmieren, die ein solches Sudoku darstellen kann.

Aufgaben

1) Schreiben Sie eine Klasse `Sudoku`. Diese soll in ihrem *privaten* Bereich folgendes Attribut besitzen:

```
vector<int> Data;
```

Der Vektor `Data` speichert die $9 \cdot 9 = 81$ Einträge, wobei ein leeres Feld durch eine 0 repräsentiert wird.

2) Schreiben Sie den Default-Konstruktor

```
Sudoku()
```

der ein leeres Sudoku erstellt.

3) Entscheiden Sie, ob der implizit vorhandene Destruktor ausreicht oder ob Sie ggf. einen geeigneten Destruktor `~Sudoku()` schreiben müssen.

4) Um bei einem Sudoku S auf den Eintrag in Zeile r (*row*) und Spalte c (*column*) zugreifen zu können, möchten wir Code wie diesen schreiben können:

```
S( r, c)= 2;        // (a) Schreibzugriff
cout << S( r, c);   // (b) Lesezugriff
```

Dazu benötigen wir die folgenden Zugriffsoperatoren `operator()`,

```
int& operator() (int r, int c)          // lesen/schreiben
int  operator() (int r, int c) const    // nur lesen
```

die den entsprechenden Eintrag aus dem Vektor `Data` zurückgeben. Implementieren Sie beide Operatoren als Methoden der Klasse `Sudoku`, wobei $r, c \in \{1, \ldots, 9\}$ sein soll. Prüfen Sie, ob (r, c) ein gültiger Index ist, und geben Sie evtl. eine Fehlermeldung aus.

5) Schreiben Sie eine Funktion (keine Methode)

```
ostream& operator<< (ostream& out, const Sudoku& S)
```

also den Ausgabeoperator zur Ausgabe eines Sudoku auf einem Output-Stream `out`[1]. Beachten Sie, dass Sie dabei nur Zugriff auf den `public`-Bereich von `Sudoku` haben. Testen Sie Ihren Code z.B. mit

```
Sudoku S;
S(2,4)= 9;
cout << S << endl;
```

6) Schreiben Sie einen Eingabeoperator

```
istream& operator>> (istream& in, Sudoku& S)
```

zum Einlesen eines Sudoku aus einem Input-Stream `in`. Verwenden Sie zum Testen die Datei `Sudoku1.txt`.

7) Schreiben Sie eine Methode

```
bool validRow( int r) const
```

die testet, ob in der r-ten Zeile keine Zahl (außer der 0) mehrfach auftaucht.

Hinweis: Benutzen Sie dabei die bereits implementierten Zugriffsoperatoren `operator()`, greifen Sie also nicht direkt auf Einträge von `Data` zurück! Dieser Hinweis gilt auch für alle übrigen noch zu implementierenden Methoden.

Fügen Sie analog die Methoden

```
bool validCol( int c) const
bool validSqr( int a, int b) const
```

für Spalten und Quadrate hinzu. Hierbei seien $a, b \in \{1, 2, 3\}$, wobei im Quadrat (a, b) der linke obere Eintrag die Position (r, c) besitzt mit $r = 3(a-1)+1$ und $c = 3(b-1)+1$.

Schließlich soll die Methode

```
bool valid() const
```

zurückgeben, ob alle Zeilen, Spalten und Quadrate gültig sind.

8) Schreiben Sie eine Methode

```
int numEmpty() const
```

die die Anzahl der leeren Felder zurückgibt, sowie eine Methode

```
bool solved() const
```

die prüft, ob keine leeren Felder mehr existieren und das Sudoku gültig ist.

9) Nun soll der Benutzer das Sudoku aus `Sudoku1.txt` schrittweise lösen. Schreiben Sie dazu im Hauptprogramm eine Schleife, in der

- nacheinander vom Benutzer Zeilen- und Spaltenindex r, c sowie eine Zahl $z \in \{0, 1, \ldots, 9\}$ eingelesen wird, die im Sudoku an der Stelle (r, c) eingetragen werden soll,
- und anschließend das modifizierte Sudoku auf dem Bildschirm ausgegeben wird.

[1] Schauen Sie sich ggf. nochmal an, wie Sie die Ein-/Ausgabeoperatoren in A4 implementiert haben.

Die Schleife soll abgebrochen werden, wenn alle Felder belegt sind oder das Sudoku ungültig ist. Nach Abschluss der Schleife soll natürlich dem Benutzter gratuliert werden, falls er das Sudoku gelöst hat.

10) Versuchen Sie nun einmal, Ihr Programm im Terminal zu übersetzen[2], wie in der Vorlesung erklärt. Falls Sie unter Linux oder MacOS arbeiten, können Sie das Terminal direkt lokal auf Ihrem Rechner starten. Andernfalls können Sie sich *remote* auf dem Mathepool anmelden, wie in der Kurzeinführung zum CIP-Pool beschrieben (z.B. via MobaXterm). Falls Sie auf dem Mathepool-Server arbeiten, müssen Sie zunächst die benötigten Dateien `main.cpp` und `Sudoku1.txt` von Ihrem Rechner auf den Mathepool-Server kopieren wie im Abschnitt „Dateitransfer“ beschrieben.

    Bei Problemen in einem der folgenden Schritte melden Sie sich bitte bei den Tutoren.

    a) Wechseln Sie im Terminal zunächst evtl. mithilfe von `cd <path/to/file>` in den Ordner, in dem sich die Datei `main.cpp` befindet.

    b) Rufen Sie im Terminal mithilfe des Befehls

    ```
    g++ -c main.cpp
    ```

    den Compiler `g++` auf, um die Objektdatei `main.o` zu erzeugen. Die Compileroption `-c` steht hierbei für „*compile*“. Durch Eingabe des Befehls `ls`, der alle Dateien im aktuellen Verzeichnis auflistet, können Sie sich davon überzeugen, dass tatsächlich eine neue Datei `main.o` erzeugt wurde.

    c) Um aus der Objektdatei `main.o` eine ausführbare Datei, z.B. `meinProg`, zu erzeugen, muss der Linker durch den Terminal-Befehl

    ```
    g++ -o meinProg main.o
    ```

    aufgerufen werden. Nach der Compileroption `-o` (für „*output*“) wird dabei der Name der ausführbaren Datei angegeben. Mit `ls` können Sie sich wiederum davon überzeugen, dass die neue Datei `meinProg` angelegt wurde.

    d) Um Ihr Programm `meinProg` aufzurufen, geben Sie im Terminal

    ```
    ./meinProg
    ```

    ein. Der Vorsatz „`./`“ zeigt an, dass sich die ausführbare Datei `meinProg` im aktuellen Verzeichnis befindet.

    e) Wenn Sie Änderungen an Ihrem Code vornehmen wollen, öffnen Sie die Quelldatei `main.cpp` in einem Editor, z.B. `gedit`, indem Sie

    ```
    gedit main.cpp &
    ```

    eingeben. Nachdem der Code geändert wurde (Speichern nicht vergessen!), muss jedesmal neu übersetzt werden. Gehen Sie dazu vor, wie unter b) und c) beschrieben. Um Kompilieren und Linken in einem Befehl auszuführen, können Sie alternativ statt der zwei g++-Befehle in b) und c) auch einfach

    ```
    g++ -o meinProg main.cpp
    ```

    im Terminal eingeben.

[2]Das ist eine wichtige Vorbereitung auf das Mathematische Praktikum im nächsten Semester!

Erweiterungen

1*) Ein n-Sudoku ($n \in \mathbb{N}$) ist ein quadratisches Gitter mit n^2 Zeilen, Spalten und Quadraten (also insgesamt n^4 Felder), in die die Zahlen 1 bis n^2 eingetragen werden sollen, so dass alle Zeilen/Spalten/Quadrate gültig sind, d.h. jede Zahl kommt genau einmal vor. Für $n = 3$ ergibt sich also das klassische Sudoku.

Verallgemeinern Sie Ihre Klasse `Sudoku` so, dass ein n-Sudoku dargestellt werden kann. Übergeben Sie n dabei im Konstruktor `Sudoku( int n=3 )`. Welche Methoden müssen wie angepasst werden?

2*) *Kreativaufgabe:* Schreiben Sie einen Sudoku-Löser in Form einer Klasse `SudokuSolver`, die im Konstruktor ein Objekt vom Typ `Sudoku` entgegennimmt und eine Methode

```
Sudoku getSolution()
```

besitzt, die die Lösung zurückgibt. Hilfreich ist z.B. für jedes freie Feld zu prüfen, welche Zahlen überhaupt noch mögliche Werte darstellen, damit das Sudoku gültig bleibt. Man sollte also als Attribute der Klasse `SudokuSolver` für jedes Feld z.B. einen `bool`-Vektor `possibleValue` mit 9 Einträgen haben, der diese Information speichert. Man beginnt dann mit einem freien Feld, das die kleinste Anzahl von Möglichkeiten besitzt und probiert (rekursiv) alle diese Möglichkeiten durch.

Lernziele

- **Klassen:** Definition einer Klasse mit `class`, Zugriffskontrolle durch `public`- und `private`-Bereiche
- **Attribute:** Deklaration in der Klasse (in der Regel im `private`-Bereich), Aufruf mit '.'
- **Methoden:** Deklaration in der Klasse (in der Regel im `public`-Bereich), Aufruf mit '.', besitzen Zugriff auf `private`-Bereich der Klasse
- **Konstruktoren/Destruktor:** zum Erzeugen bzw. automatischen Zerstören von Objekten der Klasse
- **implizite Elemente**: jede Klasse besitzt per se einen impliziten Default-Konstruktor, Kopierkonstruktor, Destruktor und Zuweisungsoperator; können durch benutzerdefinierte Varianten ersetzt werden
- **Operatorüberladung:** Ein-und Ausgabeoperatoren, Zugriffsoperatoren