A&R TECH C++ Chess Challenge

Willkommen zur A&R TECH C++ Chess Challenge! Wir freuen uns, dass du an dieser Herausforderung teilnimmst. Bevor es losgeht, möchten wir dich noch auf einige Dinge aufmerksam machen:

- Die Challenge besteht aus sieben Levels, die nacheinander gelöst werden müssen. Die Levels bauen aufeinander auf. Sobald du Level k korrekt gelöst hast, erhältst du von uns die Angabe zu Level k+1.

- Die Levels können mit Papier und Bleistift gelöst werden (z.B. auf den Rückseiten der Angaben). Du kannst die Lösungen natürlich auch auf deinem mitgebrachten Laptop präsentieren.

- In vielen Levels sind Implementierungen von Klassen oder Funktionen gefragt. Es geht uns nicht um syntaktisch einwandfreie, kompilierfähige Lösungen, sondern um die richtigen Ideen und Lösungsansätze. Solltest du die C++ Syntax nicht oder nur teilweise beherrschen, so kannst du Pseudocode verwenden.

- Es gibt keine Zeitlimits für die einzelnen Levels.

- Einige Levels setzen Grundkenntnisse des Schach-Regelwerks voraus. Falls du dir nicht sicher bist: www.google.at ☺.

- Wir haben uns bemüht, die Aufgabenstellungen so klar wie möglich zu formulieren. Sollte es aber dennoch Unklarheiten zu den Angaben geben, so kannst du uns jederzeit am Messestand besuchen. Wir nehmen uns gerne Zeit um Klarheit zu schaffen.

- Du hast Ideen, Vorschläge oder Anregungen? Vielleicht hast du auch eine Idee, wie man Schach auf eine andere Art und Weise implementieren könnte? Lass‘ uns daran teilhaben, wir diskutieren gerne mit dir!

- Warum solltest du dir überhaupt die Zeit nehmen und diese Challenge absolvieren? Nun ja, dafür kann es mehrere Gründe geben:
    - Dir macht Programmieren Spaß.
    - Du löst gerne komplexe Probleme.
    - Du hast Interesse an unserer Firma gewonnen. Wenn du uns fachlich überzeugen kannst, so bieten wir dir ein weiterführendes Gespräch mit uns an.

A&R TECH C++ Chess Challenge

Level 1

Name:

Aufgabenstellung:
Skizziere eine Klasse **`Square`**, die ein Feld eines Schachbretts darstellt. Ein Feld ist eindeutig durch seine x und y Koordinaten im Schachbrett bestimmt. Stelle Getter und Setter Methoden zur Verfügung. Implementiere außerdem eine `public` Funktion

```
<return_type> Square::name() const,
```

die den Namen des Feldes als String zurückgibt (Beispiel: „A1“, „C3“, …). Verwende als **`<return_type>`** einen gängigen String Datentyp.

A&R TECH C++ Chess Challenge

Level 2

Name:

Aufgabenstellung:
Skizziere eine Klasse **`ChessBoard`**, die ein Schachbrett darstellt. Die Klasse soll in ihrem Konstruktor 64 **`Square`** Objekte erzeugen und diese (genauer: Pointer auf diese) in einem privaten Feld/Container speichern.

Füge der Klasse **`ChessBoard`** eine `public` Funktion

```
Square *ChessBoard::getSquare(int x, int y) const
```

hinzu, die zu einem Koordinatenpaar den Pointer auf das entsprechende **`Square`** Objekt zurückliefert.

Zusatzfragen:

- Was bedeutet das **`const`** keyword in der Deklaration von `getSquare()`?
- Wie könnte die Klasse **`Square`** nun sinnvoll erweitert werden?

A&R TECH C++ Chess Challenge

Level 3

Name:

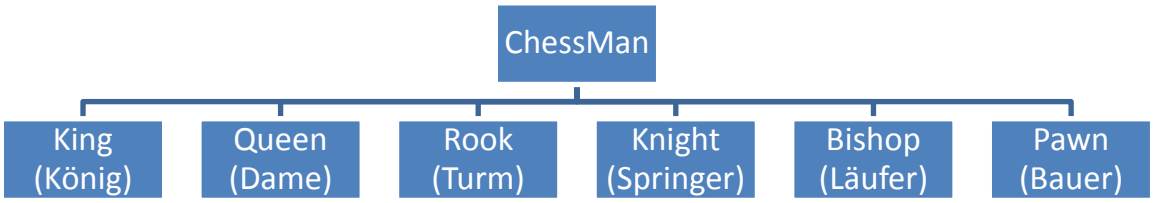
Aufgabenstellung:
An welcher Stelle in einem C++ Schachprogramm bietet sich der Einsatz von öffentlicher Vererbung und Polymorphismus an? Skizziere eine entsprechende Vererbungshierarchie.

A&R TECH C++ Chess Challenge

Level 4

Name:

Aufgabenstellung:
Gehe von folgender Vererbungshierarchie aus:



Skizziere die abstrakte Klasse **`ChessMan`**, die ein Interface für eine Schachfigur darstellt. Die Klasse soll zunächst folgende Elemente enthalten:

- Einen Pointer auf das **`Square`**, auf dem sich der **`ChessMan`** gerade befindet.
- Eine Farbe/Parteizugehörigkeit (Weiß = 0, Schwarz = 1)

Die Initialisierung dieser beiden Variablen soll im Konstruktor stattfinden. Verwende an geeigneten Stellen das C++ keyword **`const`**.

Mache Vorschläge für virtuelle Funktionen, die in **`ChessMan`** rein virtuell sind und in den abgeleiteten Klassen implementiert werden müssen. Welche anderen virtuellen Funktionen könnten eine Default-Implementierung in **`ChessMan`** definieren? (Hier sind nur Ideen und Vorschläge gesucht, nicht die konkreten Implementierungen).

A&R TECH C++ Chess Challenge

Level 5

Name:

Aufgabenstellung:
Sinnvolle rein-virtuelle Funktionen für die Klasse **ChessMan** sind

```
virtual List<ChessDirection> movementDirections() const = 0;
virtual int movementRange() const = 0;
```

Überlege und skizziere sinnvolle Implementierungen dieser Funktionen für die abgeleiteten Klassen **Queen** (Dame) und **Knight** (Springer). Gehe dabei von folgenden Voraussetzungen aus:

- Es existiert bereits eine Klasse **ChessDirection**, die eine Richtung auf einem Schachbrett (ähnlich einem 2D-Richtungsvektor) darstellt. Eine **ChessDirection** besteht aus einem x-Wert und einem y-Wert, die im Konstruktor übergeben werden.

- **List<ChessDirection>** bezeichnet eine geeignete Container Klasse, also eine Liste von **ChessDirection** Objekten.

- Die Funktion `movementDirections()` soll zurückliefern, in welche Richtungen sich ein **ChessMan** bewegen kann.
  Hinweis: Es geht hier nur um die Bewegungsrichtung, nicht um die Schlagrichtung. Beim Bauern (Klasse **Pawn**) würden sich diese unterscheiden.

- Die Funktion `movementRange()` soll zurückliefern, wie weit/oft sich ein **ChessMan** in jede dieser Richtungen pro Zug bewegen kann.

A&R TECH C++ Chess Challenge

Level 6

Name:

Aufgabenstellung:
Implementiere die öffentliche Funktion

`Set<Square*> ChessMan::possibleMovements() const`,

die eine Menge (im mathematischen Sinne) von **`Square`** Pointern zurückliefert, welche mögliche Zugziele dieses **`ChessMan`** darstellen. Gehe dabei von folgenden Voraussetzungen aus:

- Es sind nur jene Ziele gesucht, die frei sind. Es existiert bereits eine öffentliche Funktion
  `bool Square::isFree() const`,
  die du verwenden kannst. Diese Funktion liefert **`true`** zurück, wenn das Feld frei ist, sonst **`false`**.

- **`Set<Square*>`** bezeichnet eine geeignete Container Klasse, also eine Menge von **`Square`** Pointern.

- Es existiert bereits eine Funktion
  `ChessBoard* ChessMan::board() const`,
  die du verwenden kannst und die einen Pointer auf das **`ChessBoard`** zurückliefert, auf dem sich der **`ChessMan`** befindet.
  Hinweis: Du hast also indirekt auch Zugriff auf die Funktion
  `Square *ChessBoard::getSquare(int x, int y) const`
  aus Level 2.

- Es existiert bereits eine Funktion
  `bool ChessMan::canJump() const`,
  die du verwenden kannst. Die Funktion liefert polymorph zurück, ob eine Schachfigur springen kann oder nicht.

- Verwende außerdem die Funktionen
  `List<ChessDirection> ChessMan::movementDirections() const`,
  `int ChessMan::movementRange() const`,
  aus Level 5.

A&R TECH C++ Chess Challenge

Level 7

Name:

Aufgabenstellung:
Gratulation! Du hast den letzten Level erreicht. Jetzt wird es knifflig: Die fertig implementierte Funktion

```
Set<Square*> ChessMan::possibleTargets() const
```

liefert alle möglichen Ziele eines **ChessMan** zurück (also nicht nur Bewegungen, sondern auch „Schläge“) und sieht folgendermaßen aus:

```
Set<Square*> ChessMan::possibleTargets() const
{
    Set<Square*> resultSet;
    resultSet.unite(possibleMovements());
    resultSet.unite(possibleStrikes());
    resultSet.subtract(forbiddenTargets(resultSet));
    return resultSet;
}
```

Implementiere die darin aufgerufene Funktion

```
Set<Square*> ChessMan::forbiddenTargets(const Set<Square*>
&targets),
```

die aus einer gegebenen Menge an Pointern auf **Square** Objekte all jene zurückliefert, auf die der **ChessMan** nicht fahren darf, weil danach der eigene König im Schach stehen würde.

Gehe dabei von folgenden Voraussetzungen aus:

- **Set<Square*>** bezeichnet eine geeignete Container Klasse, also eine Menge von **Square** Pointern.

- Es existiert bereits eine Funktion
  ChessBoard* **ChessMan::board() const**,
  die du verwenden kannst und die einen Pointer auf das **ChessBoard** zurückliefert, auf dem sich der **ChessMan** befindet.

- Es existiert bereits eine öffentliche Funktion
  bool ChessBoard::isCheck(int color) const,
  die du verwenden kannst. Die Funktion liefert **true** zurück, wenn die Partei mit Farbe `color` (Weiß = 0, Schwarz = 1) im Schach steht, sonst **false**.

A&R TECH C++ Chess Challenge

- Es existiert bereits eine öffentliche Funktion
  `ChessMan``* ` **`Square::`**`chessman() ` **`const`**,
  die du verwenden kannst. Die Funktion liefert einen Pointer auf den **`ChessMan`** zurück, der gerade auf diesem **`Square`** steht, oder `NULL`, falls das **`Square`** leer ist.

- Es existiert bereits eine Funktion
  `void ChessMan::``moveTo(`**`Square`**` *pos)`,
  die den **`ChessMan`** auf Position `pos` bewegt (d.h. der Pointer auf sein aktuelles **`Square`** wird auf `pos` gesetzt). Achtung: Ist `pos` bereits von einer gegnerischen Figur besetzt, so wird diese durch `moveTo(pos)` „geschlagen“ (d.h. der Pointer auf deren aktuelles **`Square`** wird auf NULL gesetzt).

Zusatzfrage:

- Die Funktionen `possibleTargets()` und `forbiddenTargets()` würden – so wie sie in der Aufgabenstellung deklariert sind – zu einem Kompilierfehler führen. Warum?