Programmierung 1 (PR1)	Abschlussprüfung		Name/Mnr:	1

Jobvermittlung

Implementieren Sie die Klassen **`Qualifikation`**, **`Person`** und **`Stelle`**.

Eine Qualifikation umfasst eine Fähigkeit (die möglichen Fähigkeiten werden in einem vordefinierten enum **`Faehigkeit`** aufgezählt) sowie den Grad, zu welchem die jeweilige Fähigkeit beherrscht wird (eine ganze Zahl zwischen 1 für Grundkenntnisse und 5 für pefekte Beherrschung). In der Klasse Qualifikation sind folgende Operatoren zu implementieren:

- Operator >=: liefert **`true`**, wenn die angebotene Qualifikation (linker Operand) die geforderte Qualifikation (rechter Operand) erfüllt, d.h. beide Qualifikationen beschreiben dieselbe Fähigkeit und der Grad im linken Operanden ist mindestens so groß, wie der Grad im rechten Operanden. Andernfalls ist **`false`** zurückzuliefern.
- Operator ++ (Präfixform): Erhöht den Grad einer Qualifikation um eins (z.B. durch Besuch eines Fortbildungskurses). Der Maximalgrad von 5 darf dabei nicht überschritten werden.
- Operator <<: Eine Qualifikation wird in der Form *Fähigkeitsname* (*Grad*) z.B. „Programmieren (5)“ ausgegeben

Eine Person hat einen Namen (String), eine automatisch zu generierende, fortlaufende Nummer (unsigned, startet mit 0), eine Gehaltsvorstellung (double, >0, Monatsverdienst) sowie eine Liste von Qualifikationen (Vektor, jede Fähigkeit darf nur einmal auftreten. Wird eine Fähigkeit – z.B. im Konstruktoraufruf – mehrmals spezifiziert, so ist jene Qualifikation mit dem höheren Grad zu speichern). Folgende Methoden und Operatoren sind zu implementieren:

- **`void besuche_kurs(Faehigkeit f)`**: Falls eine entsprechende Qualifikation schon vorhanden ist, so wird deren Grad um 1 erhöht (soweit der Grad nicht schon ohnehin 5 war) sonst wird die entsprechende Qualifikation mit Grad 1 in die Liste der Qualifikationen eingetragen.
- Operator <<: Eine Person wird in der Form *Nummer*: *Name* (*Gehaltsvorstellung*) [*Qualifikationen*], z.B.
  12: Maria Meier (2000) [Programmieren (2), English (1)] ausgegeben.

Eine Stelle wird von einer Firma (String) angeboten und umfasst eine Stellenbeschreibung (String), das angebotene Gehalt (double, >=1000, Monatsgehalt) sowie eine Liste der erforderlichen Qualifikationen. Folgende Methoden sind zu implementieren:

- **`vector<Person> suche(const vector<Person>& personen) const`**: (Mindestanforderung) Die Methode liefert eine Liste aller Personen aus der als Parameter übergebenen Liste zurück, die die Anforderungen der Stelle erfüllen (d.h. alle erfordelichen Qualifikationen sind vorhanden und die Gehaltsvorstellungen übersteigen nicht das angebotene Gehalt).
- **`vector<Person> suche_kandidaten(const vector<Person>& personen) const`**: (Zusatz) Die Methode liefert eine Liste aller Personen, die zwar die Qualifikationen haben, aber deren Gehaltsvorstellungen zu hoch sind.
- **`vector<Person> suche_beste(const vector<Person>& personen) const`**: (Zusatz) Liefert eine Liste der am besten geeigneten Personen. Dabei wird eine einfache Bewertung der Qualifikationen herangezogen: Für jede Qualifikation, die übererfüllt wird (wo also der angebotene Grad der Fähigkeit den verlangten übersteigt), wird die Differenz der Grade als Punkteanzahl definiert. Diese Punkteanzahl ist für jede Person über alle verlangten Qualifikationen zu addieren.
  Gibt es Personen, die alle Anforderungen erfüllen (Ergebnis der Methode **`suche`**), so sind jene mit der maximalen Punkteanzahl die am besten geeigneten. Gibt es keine Personen, die alle Anforderungen erfüllen, so sind jene mit zu hohen Gehaltsforderungen (Ergebnis der Methode **`sucheKandidaten`**) zu bewerten. Von diesen sind jene zu wählen, deren Gehaltsvorstellungen minimal sind. Für die dann noch übrigen Personen, ist die Punkteanzahl wie oben zu ermitteln und die Personen mit der höchsten erreichten Punkteanzahl sind die am besten geeigneten.

Implementieren Sie die Klassen **`Qualifikation, Person`** und **`Stelle`** mit den notwendigen Konstruktoren, Methoden und Operatoren, sodass jedenfalls das Rahmenprogramm (qualifikation.h, qualifikation.cpp, person.h, person.cpp, stelle.h, stelle.cpp und testvermittlung.cpp verfügbar unter /home/Xchange/ue10) kompiliert und ausgeführt werden kann und die gewünschten Ergebnisse liefert. Achten Sie in Ihren Konstruktoren darauf, dass nur gültige Objekte erstellt werden können. Werfen Sie gegebenenfalls eine Exception vom Typ **`runtime_error`**.

Für Ihr Programm dürfen Sie **nur** die im vorgegebenen Rahmenprogramm angeführten include-Dateien verwenden!

Instanzvariablen sind **`private`** zu definieren und die Verwendung globaler Variablen ist (abgesehen von im Rahmenprogramm eventuell bereits definierten) nicht erlaubt! Interpretationsspielraum in der Angabe können Sie zu Ihren Gunsten nutzen.

Datei qualifikation.h:

```
#ifndef QUALIFIKATION_H
#define QUALIFIKATION_H
#include<iostream>
#include<vector>
#include<string>
using namespace std;

enum class Faehigkeit{programmieren, englisch, datenbank, web};

//Definition der Klasse Qualifikation

#endif
```

Datei person.h:

```
#ifndef PERSON_H
#define PERSON_H
#include <iostream>
#include <stdexcept>
#include <vector>
#include <string>
#include "qualifikation.h"
using namespace std;

//Definitino der Klasse Person

#endif
```

Datei stelle.h

```
#ifndef STELLE_H
#define STELLE_H
#include<vector>
#include<string>
#include"qualifikation.h"
#include"person.h"
using namespace std;

//Definition der Klasse Stelle

#endif
```

Datei testvermittlung.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <stdexcept>
#include "qualifikation.h"
#include "person.h"
#include "stelle.h"
using namespace std;

int main() {
  try {
    Qualifikation q1(Faehigkeit::englisch); //default Grad ist 1
    cout << q1 << '\n';
  }
  catch (runtime_error e) {
    cout << e.what() << '\n';
  }
  try {
    Qualifikation q2(Faehigkeit::web, -1);
    cout << q2 << '\n';
  }
  catch (runtime_error e) {
    cout << e.what() << '\n';
  }
  try {
    Qualifikation q3(Faehigkeit::datenbank,2);
    ++(++q3);
    cout << q3 << '\n';
  }
  catch (runtime_error e) {
    cout << e.what() << '\n';
  }
  Person p1("Maria Huber"); //default Gehaltswunsch 1500 Qualifikationsliste leer
  cout << p1 << '\n';
  try {
    Person p2("Kevin Huber",0);
  }
  catch (runtime_error e) {
    cout << e.what() << '\n';
  }
  Person p3("Markus Weber", 2000, {{Faehigkeit::englisch,2}, {Faehigkeit::englisch,3}, {Faehigkeit::englisch,1},
{Faehigkeit::web,2}});
  cout << p3 << '\n';
  try {
    Stelle s1("MA 4711","Strassenkehrer",500,{});
  }
  catch (runtime_error e) {
    cout << e.what() << '\n';
  }
  Stelle s2("MA 4711","Strassenkehrer",1780,{});
  vector<Person> kandidaten;
  kandidaten = s2.suche({p3});
  for (Person p : kandidaten) cout << p << '\n';
  cout << "***\n";
  kandidaten = s2.suche_beste({p3});
  for (Person p : kandidaten) cout << p << '\n';
  cout << "***\n";
```

```
 vector<Person> pliste {{"Regina Bauer", 1760, {{Faehigkeit::programmieren,2}, {Faehigkeit::englisch,1},
{Faehigkeit::web,2}}},
         {"Herbert Bauer", 1900, {{Faehigkeit::programmieren,3}, {Faehigkeit::englisch,1},{Faehigkeit::web,3}}},
         {"Eva Bauer", 2000, {{Faehigkeit::programmieren,2}, {Faehigkeit::englisch,1}, {Faehigkeit::web,4}}},
         {"Hubert Mayer", 2050, {{Faehigkeit::programmieren,3}, {Faehigkeit::englisch,3}, {Faehigkeit::web,3}}},
         {"Markus Mayer", 2050, {{Faehigkeit::programmieren,3}, {Faehigkeit::englisch,3}, {Faehigkeit::web,4}}},
         {"Birgit Mayer", 2050, {{Faehigkeit::programmieren,4}, {Faehigkeit::englisch,3}, {Faehigkeit::web,3}}},
         {"Hans Kremml", 3000, {{Faehigkeit::programmieren,5}, {Faehigkeit::englisch,5}, {Faehigkeit::web,5}}}};
 pliste.push_back(p3);
  pliste.at(0).besuche_kurs(Faehigkeit::englisch);
  pliste.at(6).besuche_kurs(Faehigkeit::web);
  Stelle s3("UniWien", "Webprogrammierer", 2000, {{Faehigkeit::programmieren,2}, {Faehigkeit::englisch,1},
{Faehigkeit::web,2}});
  kandidaten = s3.suche(pliste);
  for (Person p : kandidaten) cout << p << '\n';
  cout << "***\n";
//Dekommentieren fuer Zusatzaufgabe suche_kandidaten
/*
  kandidaten = s3.suche_kandidaten(pliste);
  for (Person p : kandidaten) cout << p << '\n';
  cout << "***\n";
*/
//Dekommentieren fuer Zusatzaufgabe suche_beste
/*
  kandidaten = s3.suche_beste(pliste);
  for (Person p : kandidaten) cout << p << '\n';
  cout << "***\n";
  Stelle s4("Uni Wien", "Webprogrammierer", 2000, {{Faehigkeit::programmieren,2}, {Faehigkeit::englisch,3},
{Faehigkeit::web,2}});
  kandidaten = s4.suche_beste(pliste);
  for (Person p : kandidaten) cout << p << '\n';
*/
  return 0;
}
```

Gewünschte Ausgabe:

```
Englisch(1)
grad nicht im erlaubten Bereich
Datenbank(4)
0: Maria Huber (1500) []
Gehaltsvorstellung muss groeszer 0 sein
1: Markus Weber (2000) [Englisch(3), Web(2)]
Mindestgehalt unterschritten
***
1: Markus Weber (2000) [Englisch(3), Web(2)]
***
2: Regina Bauer (1760) [Programmieren(2), Englisch(2), Web(2)]
3: Herbert Bauer (1900) [Programmieren(3), Englisch(1), Web(3)]
4: Eva Bauer (2000) [Programmieren(2), Englisch(1), Web(4)]
***
```

```
5: Hubert Mayer (2050) [Programmieren(3), Englisch(3), Web(3)]
6: Markus Mayer (2050) [Programmieren(3), Englisch(3), Web(4)]
7: Birgit Mayer (2050) [Programmieren(4), Englisch(3), Web(3)]
8: Hans Kremml (3000) [Programmieren(5), Englisch(5), Web(5)]
***
```

```
3: Herbert Bauer (1900) [Programmieren(3), Englisch(1), Web(3)]
4: Eva Bauer (2000) [Programmieren(2), Englisch(1), Web(4)]
***
6: Markus Mayer (2050) [Programmieren(3), Englisch(3), Web(4)]
7: Birgit Mayer (2050) [Programmieren(4), Englisch(3), Web(3)]
```

(die mit Strichen abgetrennten Teile werden erst nach Dekommentieren der entsprechend gekennzeichneten Programmteile ausgegeben)