Informationen zur Prüfung

Randbedingungen und Anforderungen – So oder ähnlich formuliert

- **Beachten Sie:** Bei jeder Aufgabe gibt es **Abzüge**, wenn das Programm nicht compiliert, wenn der Code grob ineffizient, kryptisch oder umständlich bzw. unverständlich ist, oder Best Practices ignoriert.
- **Generell:** Sollten Sie bei einer Teilaufgabe nicht auf die Lösung kommen, haben aber eine **Alternative**, so vermerken Sie das im Code und auf dem Aufgabenblatt! Diese kann, ggf. mit Abzügen, auch gewertet werden und Sie kommen weiter. Gleiches gilt, falls die Aufgabenstellung an einer Stelle unklar sein sollte.
- **Vorbereitung:** Legen Sie ein Verzeichnis mit dem Namen "`cpp_<MATRNR>_<NAME>`" an, wobei "`<MATRNR>`" durch Ihre Matrikelnummer und "`<NAME>`" durch Ihren klein geschriebenen Nachnamen ohne Umlaute ersetzt ist. In diesem Ordner legen Sie je Programmieraufgabe eine *.cpp Datei an (`a1.cpp, a2.cpp`, …).
- **Durchführung:** Geben Sie bitte zur Sicherheit Ihren **Namen und die Matrikelnummer** zu Beginn der jeweiligen, von Ihnen bearbeiteten `cpp`-Datei an.
- **Abgabe Aachen:** Sie laden eine **zip-Datei** namens "`cpp_<MATRNR>_<NAME>.zip`" (`<MATRNR>, <NAME>` s.o.) Ihres gesamten Verzeichnisses in einen vorgegebenen Ordner im Ilias hoch ("Klausurabgabe" o. ä. im Modul "C++") oder kopieren Sie diese auf einen Stick, falls Sie Probleme haben.

Beispielaufgabe A1 – Aufgabenstellung

A1) In dieser Aufgabe geht es um die Anbindung einer Datenbank. Dazu soll eine Klasse `ResultSet`, d.h. eine Art Tabelle, für die Ergebnisse einer Abfrage implementiert werden. Realisieren Sie folgende Anforderungen:

(a) [2P] Die Klasse `ResultSet` erbt von `vector<vector<string>>`.

(b) [6P] Sie besitzt eine Funktion `readFile`, die eine Datei mit Daten einlesen kann, so dass das `ResultSet` gefüllt ist. In der ersten Zeile der Datei stehen die Spaltennamen und in den Zeilen danach die durch Kommata getrennten Werte. Gehen Sie davon aus, dass es keine Leerzeichen dazwischen gibt, keine Kommata in den Texten etc. Der Funktion wird ein Tabellenname (erster Parameter) und ein Dateiname (zweiter Parameter) übergeben. Eine Beispieldatei finden Sie auf der nächsten Seite.

(c) [4P]Darüber hinaus gibt es eine Funktion `readData`, die analog zu `readFile`, Daten einliest. Diese Daten werden als (zweiter) Parameter vom Typ `initializer_list` (Vektor von Vektoren) übergeben. Das Format ist analog zu (b) und der erste Parameter ist wieder ein Tabellenname.

(d) [6P] Nutzen Sie für (b) eine statische Funktion `split`, die einen übergebenen `string` in seine durch Kommata getrennten Teile aufspaltet und diese in einem Vektor von `string`s zurück gibt.

(e) [4P] `ResultSet` kann in einen `ostream` ausgegeben werden.

(f) [2P] Testen Sie Ihren Code aus a) bis e) im Hauptprogramm a1.cpp aussagekräftig.

Beispielaufgabe A1 – gestelltes Material (Ilias)

Vorgabe Quellcode `a1.cpp`

```
class ResultSet { // a)
public:
    string name;
    // b) readFile(tableName, fileName)
    // c) readData(tableName, data)
    // d) vector strings split(string s)
    // e) Ausgabeoperator
};
```

```
int main() {
    ResultSet TPerson, TAbteilung;
// b) Datei einlesen
    // TPerson.readFile("person","person.txt");
// c) Daten einlesen
    // TAbteilung.readData("abteilung",
    //    {{"id","Beschreibung"},{"M","Marketing"}});
// d) split-Test
    // auto v = ResultSet::split("A,B,C");
    // cout << "v=[" << v[0] << "...]" << << endl;
// e) Ausgabe
    // cout << "TPerson: " << TPerson << endl;
    return EXIT_SUCCESS;
}
```

Vorgabe Datei `person.csv`

```
id,Name,Vorname,Abteilung
1,Meier,Arne,M
2,Schnulze,Thomas,M
3,Wilde,Hilde,B
```

Beispielaufgabe A2 – Aufgabenstellung

A2) In dieser Aufgabe geht es um eine generische Tupel-Datenstruktur, die Tupel der Form (x_1, …, x_n) unterschiedlicher Länge speichern und verarbeiten kann. Dabei soll der Datentyp der Elemente x_i beliebig gewählt werden können. Intern sollen die Elemente in einem `vector` mit dem generischem Typ gespeichert werden.

(a) [4P] Schreiben Sie einen Konstruktor für die Klasse `Tupel` so, dass er mit einer variablen Anzahl von Konstruktor-Parametern aufgerufen werden kann. Dabei soll der Standard-Konstruktor erhalten bleiben!

(b) [4P] Schreiben Sie einen Ausgabeoperator, mit dem ein Tupel in der Form `( 5 6 7 8 )` in einen `ostream` ausgegeben werden kann.

(c) [6P] Zwei Tupel gleichen Typs und gleicher Größe sollen nun addiert werden können. Dabei enthält das Ergebnis die elementweisen Summen der Tupel (Vektoraddition). Der Fall unterschiedlicher Längen braucht hier noch nicht geprüft zu werden!
Implementieren Sie den `operator+=` sowie (darauf aufbauend) den `operator+` .

(d) [6P] Bei der Addition unter c) sollen nun die Längen der Tupel überprüft werden. Wenn die Argumente unterschiedliche Längen besitzen, soll eine eigene Ausnahme vom Typ `SizeException` erzeugt werden, und die Addition nicht durchgeführt werden. Die Methode `what()` soll einen geeigneten Fehlertext zurückgeben (ggf. mit Angabe der beiden inkompatiblen Längen).

Beispielaufgabe A2 – Aufgabenstellung

A2) Fortsetzung

(e) [4P] Schreiben Sie einen Index-Operator, der eine konstante Referenz auf das n’te Element des Tupels zurück liefert.

(f) [4P] Spezialisieren Sie das Template für den Datentyp bool so, dass keine Instanzen von `Tupel<bool>` erzeugt werden können!

Beispielaufgabe A2 – gestelltes Material (Ilias)

Vorgabe Quellcode `a2.cpp`

```
class sizeException : public exception { };

class Tupel { };

int main() {
    // Teilaufgabe a)
    //    Tupel<int>    t1 = {1,2,3,4};
    //    Tupel<int>    t2 = {5,6,7,8};
    //    Tupel<int>    t3 = {10,12,14};
    //    Tupel<long>   t4 = {1,2};
    //    Tupel<double> t5;
    // Teilaufgabe b)
    //    cout << "t1: " << t1 << endl;
    //    cout << "t2: " << t2 << endl;
    // Teilaufgabe c)
    //    t1 += t2;
    //    Tupel<int> t6 = t1 + t2;
    //    cout << t1 << " + " << t2 << " = " << t6 << endl;
    // Teilaufgabe d)
    //    try {
    //      t1 += t3;
    //    } catch (sizeException & e) { cout << e.what(); }
    //    try {
    //      Tupel<int> t7 = t1 + t3;
    //    } catch (sizeException & e) { cout << e.what(); }
    // Teilaufgabe e)
    //    cout << "t6[0]: " << t6[0] << endl;
    //    cout << "t6[1]: " << t6[1] << endl;
}
```