A4: Ein einfaches Adressbuch

Neue Elemente von C++: Datei-IO, Vektoren, Strukturen, Datenelemente, Methoden, Operatorüberladung

In dieser Aufgabe wird ein kleines Adressbuch mit Geburtstagskalender programmiert. Zur Erleichterung stellen wir Ihnen mit der Datei `aufgabe4.cpp` im L^2P ein erstes Grundgerüst zur Verfügung. Wenn Sie möchten, kopieren Sie den Inhalt dieser Datei in den Quellcode `main.cpp` Ihres Projektes.

Aufgaben

1) Definieren Sie eine Struktur `Datum` zur Speicherung eines Kalenderdatums; die Datenelemente sind in der nachfolgenden Tabelle beschrieben.

   Außerdem wird eine Struktur `Person` zur Speicherung der Informationen über eine Person benötigt. Verwenden Sie die Variablennamen und Typen aus der folgenden Tabelle:

`struct` **Datum**	
Datenelement	Typ
Tag	ganze Zahl
Monat	ganze Zahl
Jahr	ganze Zahl

`struct` **Person**	
Datenelement	Typ
Nachname	String
Vorname	String
Email	String
MatNr	ganze Zahl
Geburtstag	Datum

   Legen Sie im Hauptprogramm ein Objekt `Ich` des Typs `Person` an und weisen Sie den Datenelementen Ihre entsprechenden Daten zu.

2) Zur Speicherung der Datensätze soll ein `vector<Person>` verwendet werden. Definieren Sie dafür die Typabkürzung `AdressbuchT` mittels einer `typedef`-Anweisung:
   ```
   typedef vector<Person> AdressbuchT;
   ```
   Definieren Sie außerdem eine Typabkürzung `GebListeT`, die eine Liste `list` mit `Datum`-Einträgen repräsentiert.

3) Die Funktionen zum Ausgeben von `Datum`-, `Person`- bzw. `AdressbuchT`-Objekten sind schon vorgegeben[1]. Versuchen Sie diese zu verstehen, bevor Sie weiter fortfahren.

4) Um die Datensätze aus der gegebenen Adressbuchdatei `AB.txt` zu lesen, werden Einlesefunktionen benötigt. Schreiben Sie dazu die folgenden Funktionen und testen Sie diese jeweils:

   a) `void Datum_lesen( istream& datei, Datum& datum)`

      zum Einlesen eines Datums, welches in Form von Dezimalzahlen in der Reihenfolge „Jahr Monat Tag“ gegeben. Zum Testen können Sie die Datei `Datum.txt` verwenden, aus der Sie mit `Datum_lesen` ein `Datum`-Objekt einlesen und anschließend mit `Datum_schreiben` auf dem Bildschirm ausgeben.

   b) `void Person_lesen( istream& datei, Person& person)`

[1] Als Ausgabestream wird in diesen Funktionen `ostream&` verwendet. Deshalb können Sie mit den Funktionen sowohl in Dateien (`ofstreams`) als auch auf den Bildschirm (`cout`) ausgeben.

zum Einlesen eines Personendatensatzes. Dieser hat die Form „Nachname Vorname Email MatNr Geburtstag“.

Beispiel: `Buenner Philippe PhBue@math.tum.de 252367 1984 5 30`

Zum Einlesen des Datums sollte natürlich die Funktion `Datum_lesen` benutzt werden. Zum Testen können Sie den ersten Datensatz der Adressbuchdatei `AB.dat` verwenden.

c) `void AB_lesen( ifstream& datei, AdressbuchT& AB)`

zum Lesen eines Adressbuches aus einer Datei. Da Sie nicht vorher wissen, wie umfangreich das Adressbuch sein wird, bietet es sich an, die `vector`-Methode `push_back` zur Implementierung zu verwenden, mit der eine neue Komponente an einen Vektor angehängt werden kann. Zum Testen können Sie die Adressbuchdatei `AB.dat` verwenden.

5) Fügen Sie Ihren eigenen Eintrag `Ich` dem eingelesenen Adressbuch hinzu.

6) Schreiben Sie für den neuen Datentyp `Datum` einen Ein- und Ausgabeoperator

```
istream& operator>> ( istream& in,  Datum& dat);
ostream& operator<< ( ostream& out, const Datum& dat);
```

um Code wie diesen schreiben zu können:

```
Datum dat;
ifstream ifs( "Datum.txt");
ifs >> dat;
cout << dat;
```

Das sollte mit Hilfe der Funktionen `Datum_lesen` bzw. `Datum_schreiben` kein Problem sein. Fügen Sie die entsprechenden Operatoren auch für `Person` und `AdressbuchT` hinzu.

7) Schreiben Sie eine Funktion

```
void geboren_in( int monat, const AdressbuchT& AB, AdressbuchT& MonatAB)
```

die das Adressbuch `AB` nach Personen durchsucht, die im Monat `monat` Geburtstag haben. All diese Personen werden in das Adressbuch `MonatAB` kopiert. Testen Sie diese Funktion für Ihren Geburtsmonat.

8) Definieren Sie in der Struktur `Datum` Ihre erste *Methode*

```
bool gueltig() const
```

welche für ein gültiges Datum `true` und für ein fehlerhaftes `false` liefert. Die Methode ist const-qualifiziert, da sie das Objekt nicht verändert. Ein Datum mit Tag t, Monat m und Jahr j wird genau dann als gültig betrachtet, wenn

$$m \in \{1, 2, \ldots, 11, 12\} \quad \text{und} \quad t \in \{1, 2, \ldots, 30, 31\}$$

gilt.

Schreiben Sie damit eine Funktion

```
void listeGueltigeGeb( const AdressbuchT& AB, GebListeT& geb)
```

die alle gültigen Geburtstage eines Adressbuches in einer Liste `geb` ablegt, und wenden Sie diese auf Ihr Adressbuch an.

9) Schreiben Sie einen Vergleichsoperator `operator<` für `Datum`. Sortieren Sie anschließend die im vorigen Aufgabenpunkt erhaltene Geburtstagsliste mit Hilfe der `list`-Methode `sort`, die automatisch Ihren geschriebenen `Datum`-Vergleichsoperator benutzt.

Erweiterungen

1*) Benutzen Sie den Sortieralgorithmus `sort` aus der Standardbibliothek `algorithm`[2], um eine nach Geburtstagen sortierte Version des Adressbuches auszugeben:

```
sort( AB.begin(), AB.end(), geboren_vor);
```

Dazu muss eine Vergleichsfunktion

```
bool geboren_vor( const Person& a, const Person& b)
```

für die Struktur `Person` geschrieben werden.

2*) Sortieren Sie das Adressbuch nach Matrikelnummern.

3*) Geben Sie das Adressbuch nach Nachname (bei gleichem Nachnamen: nach Vornamen) sortiert aus.

4*) Schreiben Sie für die Struktur `Person` eine Methode

```
int Alter( Datum heute)
```

welche das Alter der Person (in Jahren) am übergebenen Tag `heute` zurückgibt. Testen Sie diese Funktion für Ihren Geburtsmonat.

5*) Geben Sie die Person aus dem Adressbuch aus, die als nächstes Geburtstag hat.

6*) Schreiben Sie eine präzisere Version der `Datum`-Methode `gueltig`, indem Sie die unterschiedlichen Tage pro Monat berücksichtigen (oder sogar die Schaltjahrregeln).

7*) Entwerfen Sie ein textbasiertes Benutzermenü; dieses soll dem Benutzer des Programmes die Funktionen aus den vorherigen Aufgabenteilen als Liste anzeigen und ihm zum Beispiel durch Eingabe einer Zahl ermöglichen, eine der Funktionen auszuführen. Überlegen Sie sich als erstes, wie sich diese Aufgabe sinnvoll strukturieren lässt.

Lernziele

- **Datei-IO:** File-Streams benutzen, `ostream`, `istream` als Basisklassen zu File-Streams und Character-Streams `cout`, `cin`
- **Sequentielle STL-Container:** `vector` und `list` benutzen.
- **Strukturen:** Definition einer Struktur mit `struct`, Zugriff auf Datenelemente mit '.'
- **Methoden:** Definition in der Struktur, Aufruf mit '.'
- **Operatorüberladung:** Ein-und Ausgabeoperatoren, Vergleichsoperatoren

[2]Diese muss zuvor mit `#include <algorithm>` eingebunden werden.