Praktikumsaufgabe 2

Fassung B (für motivierte Studenten)

Lernziele

Wiederholung und tieferes Verständnis der in EPR/OOA behandelten C/C++-Programmierung. Vergleich von embedded SQL und eines nativen Call Level Interfaces. Erarbeiten einer Abstraktion des Zugriffs mit Mitteln der modularen Programmierung.

Vorbereitung

Folgende Informationen müssen Sie zum *Antestat* bereithalten:

- Wie können in einem C++-Programm die Funktionsdefinitionen auf mehrere Sourcedateien verteilt werden? Welche Schritte werden bis zur Erstellung des Executables durchlaufen? (Wiederholung EPR)
- Wie funktioniert Compilieren und Linken mit dem *gcc*? Wie werden Bibliotheken verwendet? ([1] Kap. 12)
- Wie kann die Projektverwaltung mit einem *Makefile* automatisiert werden? ([1] Kap. 12) Wie ruft man es auf?
- Erläutern Sie anhand des bereitgestellten Makefiles, welche Sourcedateien Sie erstellen müssen und wie das Konzept zur Kapselung des DB-Zugriffs ist. Das Hauptprogramm *main.cpp* soll dasselbe sein, unabhängig davon ob der Low-Level Datenbankzugriff über *ecpg* ([3] p. 296 ff) oder *libpq* ([2] Kap. 9.1) erfolgt.

Aufgabe

Es soll ein C++-Programm "dbimp" geschrieben werden, das Daten aus einer Datei in eine Datenbanktabelle *hersteller* einspielt. Parallel sollen zwei Programme entwickelt werden: bei *dbimp-ecpg* erfolgt der DB-Zugriff über embedded SQL (ecpg) und bei *dbimp-libpq* über libpq. Das Hauptprogramm ruft für den Datenbankzugriff nur die Funktionen auf, deren Prototypen in der bereitgestellten Datei *db.h* deklariert sind:

- *db_login*, *db_logout* - Datenbanklogin und -logout
- *db_begin*, *db_commit*, *db_rollback* - Transaktionsbefehle
- *db_findhnr* - Gibt zurück, ob Herstellernummer schon vorhanden
- *db_insert* - Einfügen Datensatz
- *db_delete* - Löscht kompletten Tabelleninhalt

Diese Funktionen sind dann für jede Programmvariante in getrennten Modulen zu implementieren.

Kommandozeilenoptionen Die Programme sollen folgendermaßen aufgerufen werden:

```
Usage:
        dbimp-ecpg  [options] <infile>
        dbimp-libpq [options] <infile>
Options:
        -del  delete table contents before import
        -u  database user
        -p  password
        -h  database host
        -d  database name
For unset options, the usual PostgreSQL environment
(PGUSER, PGPASSWORD, PGHOST, PGDATABASE) takes effect
```

Die Reihenfolge der Optionen ist egal. Bei fehlerhaftem Aufruf (kein *infile* oder unbekannte, mit '-' beginnende Option) wird die obige Meldung ausgegeben und abgebrochen.

Zieltabelle und Dateiformat Die Zieltabelle *hersteller* müssen Sie von Hand per SQL anlegen mit folgenden Feldern:

Feld	hnr#	name	plz	ort
Typ	varchar(3)	varchar(30)	varchar(5)	varchar(30)

Die Import-Datei enthält pro Zeile einen Datensatz, wobei die einzelnen Felder durch `;` getrennt sind. Die Felder stehen in der Reihenfolge *hnr, name, plz, ort*.

Funktionalität *dbimp* soll sich wie folgt verhalten:

- Der ganze Import erfolgt in einer Transaktion: bei Erfolg *commit* und bei einem Fehler Abbruch und *rollback*.
- Vor dem *insert* wird anhand des Schlüsselfelds überprüft, ob der Datensatz schon in der Datenbank vorhanden ist. Wenn ja, wird der Satz nicht importiert.
- Wenn über die Option *-del* gewünscht, wird vor dem Import der Tabelleninhalt gelöscht (innerhalb der Transaktion).

Am Ende gibt das Programm eine Importstatistik aus mit der Gesamtzahl der gelesenen Datensätze und der Anzahl der davon importierten Sätze.

Test Sie können Ihr Programm überprüfen anhand der Testdaten [4]. Hier die Sollergebnisse beginnend mit einer leeren Tabelle *hersteller*:

Kommando	Datensätze/ davon importiert	Anzahl Tabellensätze nach dem Import
dbimp data1	3/3	3
dbimp data2	2/3	5
dbimp -del data2	3/3	3
dbimp data3	Abbruch wegen Fehler in Zeile 2 von *data3*	3

Hinweise zur C++-Programmierung

- Ein Makefile für diese Aufgabe finden Sie als *Makefile.2b* auf der Webseite zu dieser Veranstaltung.
- Um das Makefile portabel zu halten, verwendet es das Programm *pg_config*, das die Verzeichnisse der Include/Library-Files über entsprechende Optionen zurückgibt.
- In embedded SQL können Sie den Login nur über getrennte Übergabe von User und Passwort machen:

```
EXEC SQL CONNECT TO :target USER :user/:password;
```

- Komplette Zeilen können Sie einlesen mit der Funktion *fgets()*. Zum Zerlegen der Inputzeilen können Sie *strtok()* oder -besser- *strsep()* verwenden. Alternativ können Sie auch den Tokenizer aus der OOA-Vorlesung verwenden, wobei Sie für die Liste den STL-Container *list* oder *vector* verwenden.
- Um Buffer-Overflows zu vermeiden, verwenden Sie nach Möglichkeit die STL-Klasse *string*. Diese kann man auch an die libpq-Funktionen übergeben, die ein *const char** als Argument erwarten (wie?).
- Zum Debuggen können Sie den "Data Display Debugger" *ddd* verwenden. *ddd* ist ein grafisches Frontend zum *gdb*. Im unteren Fenster des *ddd* können Sie auch direkt Kommandos an den *gdb* absetzen, ohne sich durch unzählige Menüs hangeln zu müssen. Nützliche *gdb*-Kommandos sind *run arg1 arg2 ...* (Startet Programm mit angegebenen Argumenten), *print var* (druckt Inhalt der Variablen *var* aus) und *list file:line* (öffnet direkt Datei *file* und springt an Zeile *line*).

Referenzen

[1] Welsh, Kaufmann: *Linux - Wegweiser zur Installation&Anwendung.* Semesterapparat (`TWR Wels`)

[2] The PostgreSQL Global Development Group:
PostgreSQL 9.1.8 Dokumentation. `http://www.postgresql.org/docs/` (2013)
Kapitel "Client Interfaces, libpq"

[3] Geschwinde, Schönig: *PostgreSQL Developer's Handbook.* Semesterapparat (`TWY Geschw`)

[4] Die Testdaten unter *http://informatik.hsnr.de/~dalitz/data/lehre/DBS/aufg2dat.tar* können Sie mit dem Befehl *tar xf ...* entpacken. Dort liegt auch die Header-Datei *db.h* und das Makefile *Makefile.2b*.