Assignment 18, Data-Structures

Oleg Sivokon

<2016-06-25 Sat>

Contents

1 Problem Description

Implement a program to manage university library. Below are the operations the program must support. The program should use query language given by ABNF grammar associated to each operation:

1. `join_library(patron, patron_id)`.

   `‘+‘ patron patron_id`

   Adds a new patron to the library.

2. `leave_library(patron, patron_id)`.

   `‘-‘ patron patron_id`

   Removes existing patron from the library.

3. `borrow_book(patron, patron_id, book_id)`.

   `patron patron_id book_id ‘+‘`

   Associates given book to patron.

4. `return_book(patron, patron_id, book_id)`.

   `patron patron_id book_id ‘-‘`

   Dissasocitates the given book and patron.

5. `list_books(patron_id)`.

   `‘?‘ patron_id`

   List all books borrowed by indicated patron.

6. `who_borrows(book_id)`.

   `‘?‘ book_id`

   Displays the patron(s) currnetly holding the indicated book.

7. `borrows_most()`.

   `‘?‘ ‘!‘`

   Displays the patron who currently borrows most books.

2 Discussion

2.1 Reflection on assignment

There are several problems with this specification:

1. It isn't stated whether the same patron can join the library multiple times, but I assumed patrons may only join the library once.
2. It isn't stated whether it is possible to leave the library while borrowing any books. I will treat this as impossible, and will prompt the user to first return the books to the library.
3. It isn't clear whether leaving multiple times should result in an error. I will not treat this as an error.
4. It is not clear whether `book_id` is akin to ISBN (in which case multiple copies of the same book may exist), or that `book_id` is some internal library code uniquely marking the book. I will not treat is as unique.
5. By the same token, it may be possible to return the same book multiple times, viz when the same patron holds multiple copies of the same book.
6. Consequently, `who_borrows` may return the list of patrons (all of which hold the same book, just not the same exact copy).
7. Some queries provide excess information. I am going to make the format more flexible to allow shorter input while preserving the original syntax.

2.2 Data-structures used to support queries

If it wasn't for `borrows_most()`, two `hash-tables` would fit the bill perfectly: `library_books` table would use book identifiers as keys and patrons holding these books as values. Second table, `library_patrons` would use patron identifiers as keys and books held by patrons as values. Thus, adding patron to the library would take constant time, removing patron from the library would also take constant time. Removing a book from the library would take constant time (since we can look up `library_books` and immediately discover whether it is held by anyone. Similarly, `who_borrows` and `list_books` would all run in constant time.

The problematic `borrows_most` would run in linear time (we would need to look at all keys in `library_patrons` table to find the patron who holds most books.

We can mitigate this by adding a special slot in the library for the patron currently holding most books, and then updating this slot any time the library is modified: for an additional constant memory cost and constant cost incured on every destructive operation we would perform every operation in constant time.

2.3 Extension to queries

To make working with the program more convenient, I will add the following queries:

1. `add_book`.

   ```
   ‘.‘ book_id+ ‘+‘ | ‘.‘ file | ‘.‘ num
   ```

   Adds book to the library. Book identifier may be repeated several times, or red from a file or generated by the program.

2. `show_library`.

   ```
   ‘?‘ ‘?‘
   ```

   This prints the current state of the library, both the books and the patrons.

3. `borrow_book` and `return_book` don't need the name of patron returning or borrowing the book (identifier is enough). So the patron's name is optional in the queries.

3 Solution

Below is the `yacc` code for the queries grammar (auxilary code was removed for brevity).

The parser:

```
books    :       BOOKID { $$ = cons($1, NULL); }
         |       BOOKID books { $$ = cons($1, $2); }
         ;

rquery   :       PATRON PATRONID BOOKID PLUS {
                     $$ = query(BORROW, $1, $2, $3); }
         |       PATRON PATRONID BOOKID MINUS {
                     $$ = query(RETURN, $1, $2, $3); }
         |       PLUS PATRON PATRONID {
                     $$ = query(JOIN, $2, $3, NULL); }
         |       MINUS PATRON PATRONID {
                     $$ = query(LEAVE, $2, $3, NULL); }
         |       QUESTION PATRONID {
                     $$ = query(BOOKS, NULL, $2, NULL); }
         |       QUESTION BOOKID {
                     $$ = query(WHO_BORROWS, NULL, NULL, $2); }
         |       QUESTION BANG {
                     $$ = query(BORROWS_MOST, NULL, NULL, NULL); }
         |       QUESTION QUESTION {
                     $$ = query(SHOW, NULL, NULL, NULL); }
         ;

populate :       DOT books { populate_library_list($2); }
         |       DOT QUOTED { populate_library_file($2); }
         |       DOT NUM { populate_library($2); }
         ;

pquery   :       rquery { process_query($1); }
         |       populate {;}
         ;

pqueries :       pquery {;}
         |       pquery pqueries {;}
         ;
```

And the lexer:

```
[+]                        { yylval.str = str(yytext); return PLUS; }
[-]                        { yylval.str = str(yytext); return MINUS; }
[?]                        { yylval.str = str(yytext); return QUESTION; }
[!]                        { yylval.str = str(yytext); return BANG; }
\.                         { yylval.str = str(yytext); return DOT; }
[0-9]{9}                   { yylval.str = str(yytext); return PATRONID; }
[a-zA-Z][a-zA-Z][0-9]{4}   { yylval.str = str(yytext); return BOOKID; }
[a-zA-Z]{1,32}             { yylval.str = str(yytext); return PATRON; }
[\"][^\"]+[\"]             { yylval.str = quote(yytext); return QUOTED; }
#[0-9]{1,12}               { yylval.num = atoi(yytext + 1); return NUM; }
[ \t\r\n]+                 ;
.                          printf("line: %d, Unexpeced character: %s\n",
                                  yylineno, yytext);
```

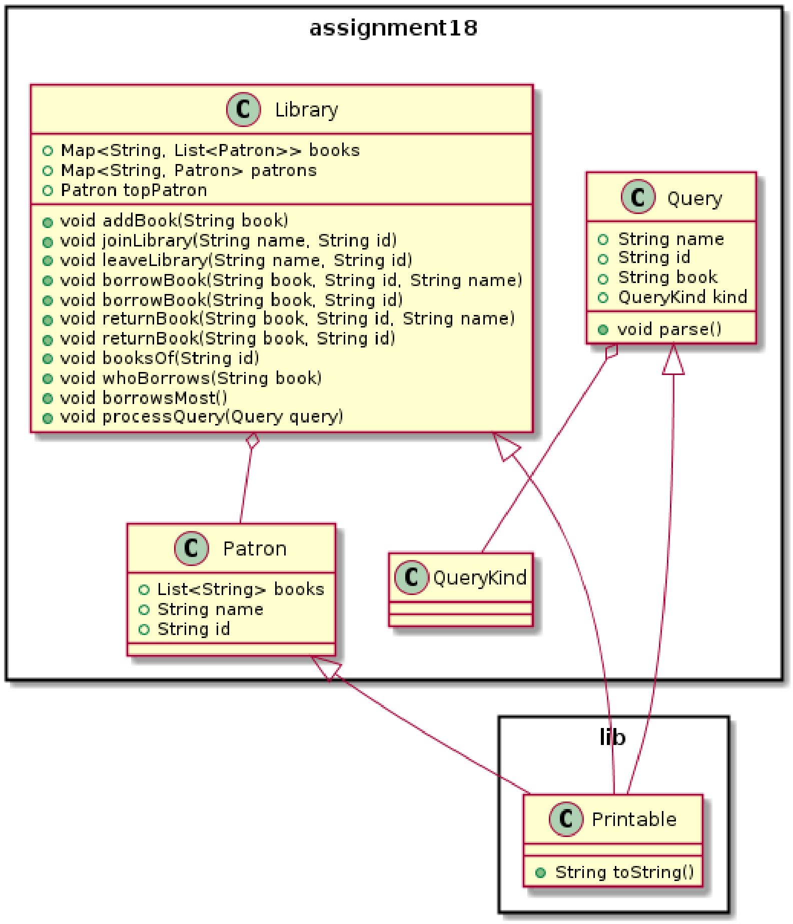
No new data-structures are required for solving this task. The UML diagrams of the objects involved are given below:

The relevant source files are:

1. `assignment18.c` - program's entry point.
2. `query.c` - query interface.
3. `library.c` - library data-structure.
4. `query_grammar.y` - query parser.
5. `queries.l` - query lexer.

Documentation for this code is automatically extracted from header files by `Doxygen` and stored in `../doc` directory.

To compile the code install SCons, GNU Bison (YACC), Flex (LEX). If you aren't interested in generating from YACC and LEX files, this repository contains already generated C files and headers.


assignment18
Library
+ Map<String, List<Patron>> books
+ Map<String, Patron> patrons
+ Patron topPatron
+ void addBook(String book)
+ void joinLibrary(String name, String id)
+ void leaveLibrary(String name, String id)
+ void borrowBook(String book, String id, String name)
+ void borrowBook(String book, String id)
+ void returnBook(String book, String id, String name)
+ void returnBook(String book, String id)
+ void booksOf(String id)
+ void whoBorrows(String book)
+ void borrowsMost()
+ void processQuery(Query query)
Query
+ String name
+ String id
+ String book
+ QueryKind kind
+ void parse()
Patron
+ List<String> books
+ String name
+ String id
QueryKind
lib
Printable
+ String toString()