Overview

- Name: Library Management Information System (`LMIS` for short)
- Timeline
  - `Start` 2016.3.10
  - `Architecture Design` 2016.3.11
  - `UI Programming` 2016.3.12 - 2016.3.17
  - `DataStore Programming` 2016.3.17 - 2016.3.23
  - `Testing` 2016.3.24
- Development Environment
  - `IDE` Code::Blocks 16.02
  - `Compiler` GCC 4.2.1

Analysis

Description of function

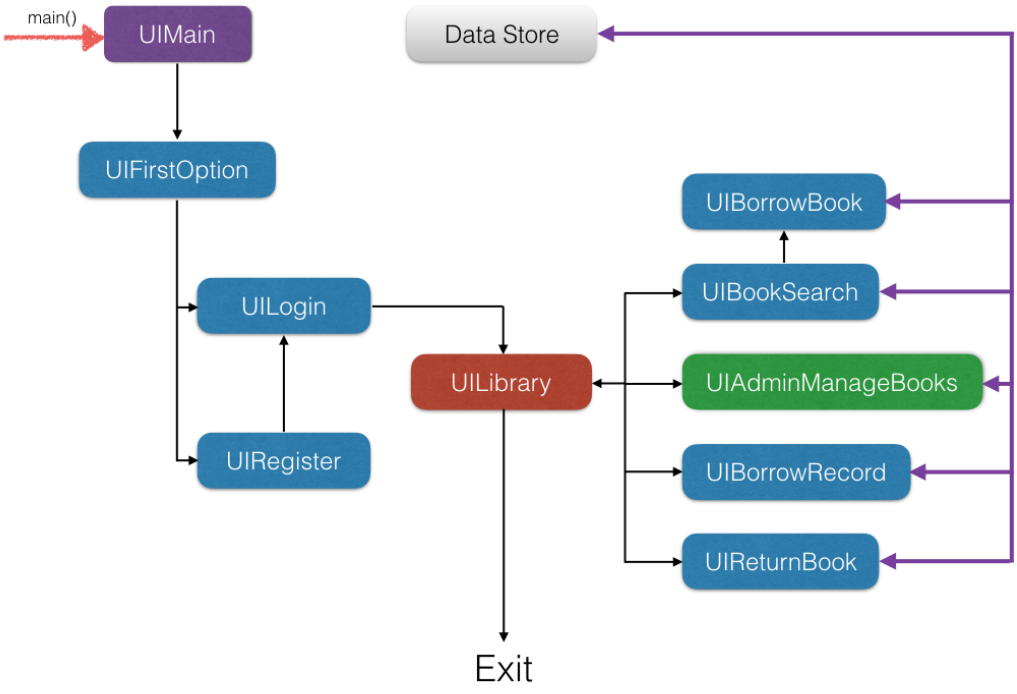
1. Normal users sign in with their name and password.
2. Add, modify or delete books by the `admin`.
3. Search for books.
4. Borrow or return books.

Solution

A simple process-oriented program with some advanced features.

Design

Modules & Procedure



`Data Store` in the picture is especially an independent program, which would be refered to in the `Innovation Points` part.

Data Structure

- **Linked List.** A basic data structure which has already have support of standard library in languages such as `C++` and `Java`. There aren't runtime support of genericity like `C++ template`, so I created linked lists of different kinds of data types, due to which the code would be reused much less.
- **Trie Tree.** Refer to the `Innovation Points` part.

Key Algorithm

- Query of data indexed by `trie tree`.

Debug Record

1. I've spent lots of time on coordinating the header file (`.h`) and the

program file (`.c`). At first all of the global variables in my program are declared as `static` and I've known later that variables declared this way could only be used in per unit of compilation (normally a `.o` file generated by `gcc -o *.c`). However, I detected at the last time that `extern` should be used in another `.c` file, but in the file of first declaration, just declare it commonly. Compilation parts using the `extern` would get access to the external (global) variable.

2. At some times, constant variables declared as `#define` is more convenient to be used than `const` . I put all the strings which would be printed on the screen in the `Resources.h` file. While the `const char []` needs length of array to be provided, `#define` is more easy to be used.

Innovation Points

Global Error Code & Status Centred Programming

All the functions (without a necessary value to return, for example, except of functions returning head of `Linked List`) would return a code of its situation of executing. It's called as `global error code` . It's just like this:

```
enum runtimestatus {

    // Everything is safe and sound.
    Okay = 0,

    // Some error that couldn't be recognized.
    UnrecognizedError,

    // User wants to exit the program.
    UserExit,

    // User forces to exit the program.
    ForcedExit,

    // Go back to library menu.
    BackToLibrary,

    ...

};

typedef enum runtimestatus RuntimeStatus;
```

By the way, this kind of design could also provide the status of program runtime. In case of abnormally exiting the program, the code can be used for rapid check of where is wrong.

Complicated UI Jumping

It could be seen in my `Modules and Procedure` part that the jump between User Interfaces are a little complicated.

I realized this by two of the global variables: `nextUI` and `nextStatus`. Functional pointer is used.

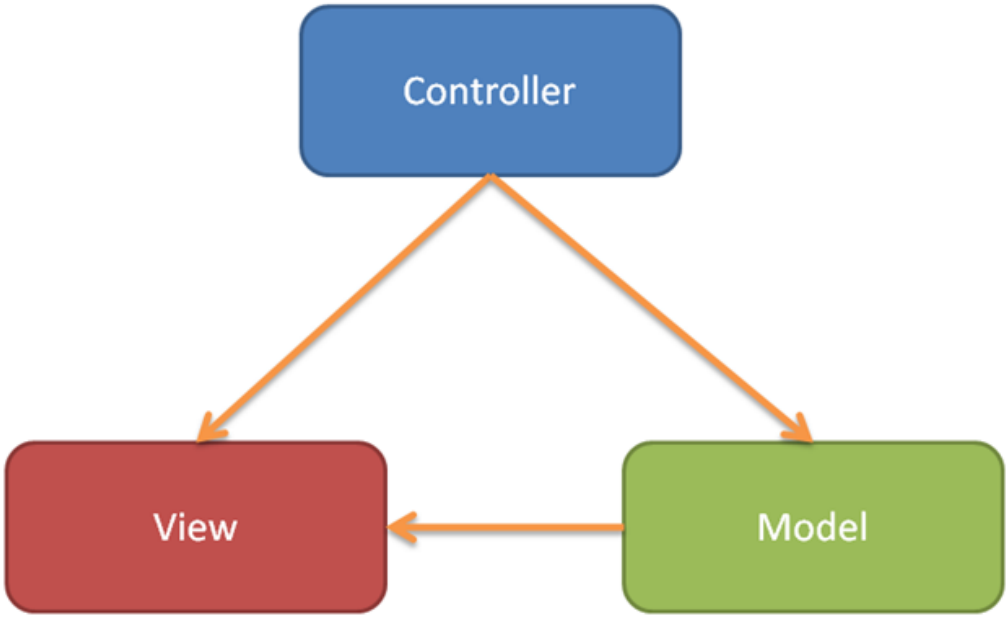
```
typedef RuntimeStatus (* vc)(RuntimeStatus);
(extern) vc nextUI;
(extern) RuntimeStatus nextStatus;
```

These statements are in either `ProgramMain.c`, near `main()` function, or in `GlobalUsage.h`.

When a `controller` (refered to in `An MVC-like Design Pattern` part) gets the returning value of `printView` (also `MVC`) function, the value will be handled, parse which UI is the next one, and use the function of `setNextVC(vc, RuntimeStatus)` to set the two value of `nextUI` and `nextStatus`. Then, on returning to the `main()`, `nextUI(nextStatus)` is called to wake up the next UI. Compared with the main menu centred programs, complicated UI jump can be realized in this way.

An MVC-like Design Pattern

This is an example picture for teaching `MVC` (Model-View-Controller design pattern):



Traditionally, the `controller` passes the information separately to `view` and `model`, while on the presentation of `view`, `model` is used to provide dynamic data.

However, this kind of design pattern is a little difficult to be applyed in this program.

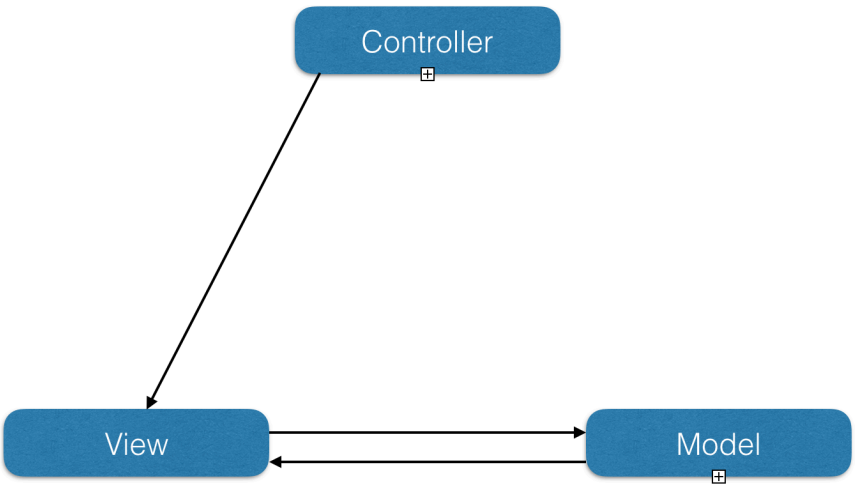
- The world's first valuable practice of MVC is the `Struts` open-source framework of JSP & Servlet, while `JavaBean` is for creating model, `JSP` for view, and `Servlet` for controller. We can say that the

system of light application of `Java EE` is designed like this and usage `MVC` is not difficult for others to understand. Except for `Struts`, there are many different examples alike.

- However, our `C` program is aimed for interaction with human beings by character interfaces, during which the user always need to communicate with the console to use the program. If I use the common `MVC`, the model can only pass the values to the view function by calling the function, pass the formal argument, but that's not easy to be dynamic.

So I tried a new design pattern: **Let view function call the model function.** I call that as an `MVC-like` design pattern.

It can be seen in my source code that the `model` function is managed unitedly. Each function uses `socket` to get data from `DataStore`, which will be talked about later.



Design of Preventing File Operating Error: Socket & Independent Data Storage Process

The main program gets data from file indirectly by usng `socket` to communicate with the `DataStore` program.
The inspirition of this idea started because of the real database. Databases

like `MySQL` and `Microsoft SQL Server` supports mutiple threads of data request. However, if I read the file data directly by the program, meanwhile open another program to get access to the file, then one of them will crash.

To support multiple process getting the data, I chose to move `DataStore` to an independent program. To communicate with socket, a protocol is defined like this:

```
N Gone with the Wind
C Love Romantic
I 2342984747623
```

(N = Name, C = Classify, I = ISBN)

Feeling & Suggestion

Nothing. A good experiment. :)