

河南工业大学

课 程 报 告

课 程 名 称: 程序设计实践

专 业 班 级: 计算机 1303 班

学 生 姓 名: 田劲锋

学 号: 201316920311

任 课 教 师: 唐建国

学 期: 2013-2014 学年第二学期

课程报告任务书

题目	标准化考试系统							
主要内容	<p>开发出一个标准化考试系统，所谓标准化考试系统即仅支持选择题，也是为方便自动批改的功能的实现。要求实现以下基本功能：</p> <p>(1) 提供给教师添加试题的功能（试题信息用文件保存）——输入；</p> <p>(2) 试题的整体浏览功能；</p> <p>(3) 能够抽取试题组合成一套试卷（组卷的策略：可以是随机的，当然若教师添加的试题时有知识点、章节等信息，亦可以实现按照一定的组卷策略实现出题：如每个知识点抽取若干题目，最终组合一套试卷）；</p> <p>(4) 教师实现题目信息的管理，比如删除、修改等；</p> <p>(5) 查询功能（至少一种查询方式）、排序功能（至少一种排序方式）。</p> <p>扩展功能：可以按照自己的程度进行扩展。比如(1) 简单的权限处理；(2) 成绩报表打印功能；(3) 甚至根据自己情况，可以加上学生信息和考试成绩信息的管理，并扩充为广义的考试系统。即学生输入账号密码登陆，进行考试，交卷后显示成绩；(4) 模糊查询；(5) 综合查询；(6) 统计、分析等功能。总之，可以根据自己需求进行分析功能。</p> <p>特别说明：尽可能地运用自己已经学习过的数据结构的知识去展现。</p>							
任务要求	<p>一、提交材料应包括：(1) 系统源代码(2) 课程报告</p> <p>二、整个设计过程具体要求</p> <p>(1) 需求分析 要求学生对案例系统进行分析，设计出需要完成的功能，完善各个模块的调用关系；</p> <p>(2) 设计过程 要求学生进一步明确各模块调用关系，进一步完善模块函数细节（函数名、参数、返回值等）；</p> <p>(3) 实现过程 要求学生养成良好的编码习惯、完成各个模块并进行测试，最终完成系统整体测试；</p> <p>(4) 总结阶段 按照要求完成系统设计和实现报告，并进行总结、答辩。</p>							
成绩 评定	报告撰写情况（30分）			系统完成情况（30分）		答辩情况（40分）		总分
	内容 20分	规范程度 5分	程序测试 5分	基本功能 20分	扩展功能 10分	自述情况 10分	答辩情况 30分	

成绩评定教师：

目录

1	需求分析	3
1.1	任务探究	3
1.2	界面模块	3
1.2.1	学生功能模块	4
1.2.2	教师功能模块	4
2	概要设计	6
2.1	类的划分	6
2.2	List 类	7
2.3	SList 类	7
2.4	Problem 类	8
2.5	Paper 类	9
2.6	User 类	9
2.7	Score 类	10
2.8	公共回调函数	10
2.9	UI 类	11
2.10	辅助方法	11
3	详细设计	13
3.1	已实现功能	13
3.1.1	插入题目	13
3.1.2	模糊查询	14
3.1.3	生成试卷	14
3.1.4	用户登录	15
3.1.5	学生考试	15
3.2	未实现功能	17
3.2.1	多项选择题	17
3.2.2	词法分析	17
3.2.3	图形界面	17
4	调试分析	19

5	测试结果	21
5.1	教师模块	21
5.1.1	题目增删改	21
5.1.2	题目查询	23
5.1.3	生成试卷	23
5.2	学生模块	23
6	心得总结	29
A	代码清单	31
A.1	许可证	31
A.2	主文件	31
A.3	UI 类	31
A.4	List 类	42
A.5	Slist 类	44
A.6	Problem 类	50
A.7	Paper 类	52
A.8	Score 类	54
A.9	User 类	55
A.10	辅助方法	57
A.11	工程文件	59
	参考文献	60



1 需求分析

1.1 任务探究

大一最后的课程——程序设计实践，目的是加深对 C 语言的理解和使用，为后来的学习打好基础。课程设计要求做一个标准化考试系统。

考虑基本功能，即是对题目数据库的增、删、改、查（用文件存取数据），以及按一定策略生成试卷。

考虑扩展功能，需要实现用户模块和权限处理，还有简单的成绩管理。

1.2 界面模块

对于本程序而言，我们不需要复杂的 GUI（图形用户界面）。因为标准 C 中并没有直接对图形操作的库，直接调用 Windows API 会无法移植到类 UNIX 操作系统，并且程序会相当复杂。如果可能的话，我更喜欢用 Web 的方式呈现，当然这也需要使用相应的编程工具链，不是标准 C 可以简单做到的。

所以我决定采用纯文本命令行界面，如非必要不对终端界面进行特殊设置。本来是要实现命令行参数接口的，限于时间关系，未能完成。目前的界面只是简单地文本状态下的用户交互，提供简单的容错处理。

考虑用户界面，提供学生和教师两个入口。如图 1 所示，主界面提供了两个用户入口，并提供了帮助和关于。

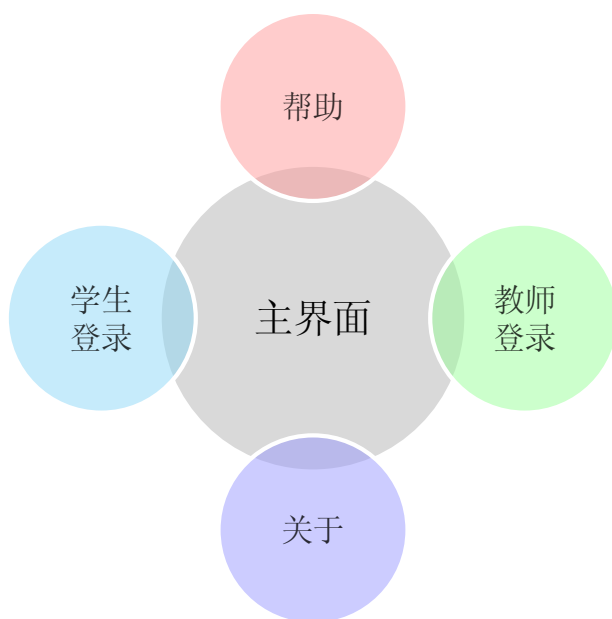


图 1: 主要模块

1.2.1 学生功能模块

学生模块属于扩展功能，如图 2，提供做卷子和看成绩两个模块入口。



图 2: 学生功能模块

1.2.2 教师功能模块

教师模块是本程序的主要部分。如图 3 所示，应该实现试题的增、删、改、查功能，实现一定策略的“智能”组卷算法。作为扩展功能，实现对成绩的查看。

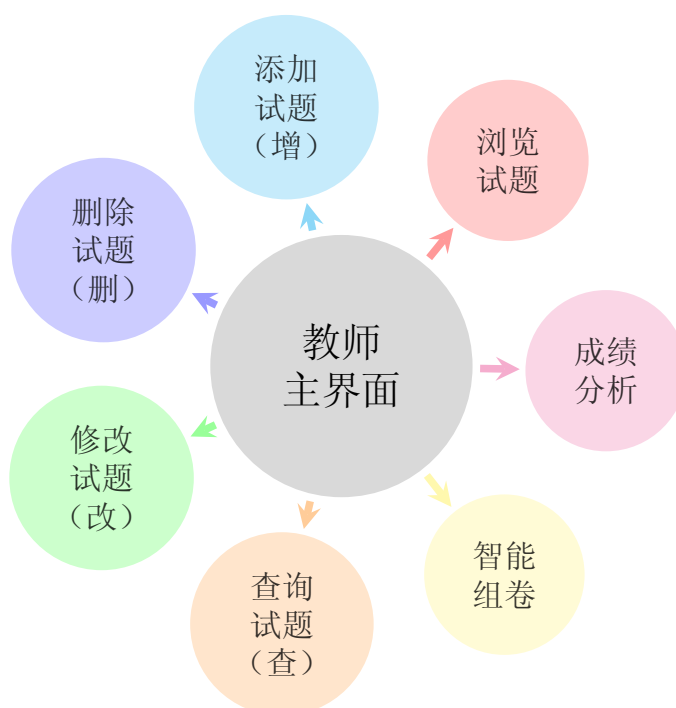


图 3: 教师功能模块

目前不需要做到批量增、删、改，其对象是应当是唯一的，这里我们用题目编号来指定。

对于查询操作，应该提供多种方式。如图 4，可以根据试题某个属性单独查询，也应提供模糊查询以查询所有选项。

对于组卷功能，我们应该提供多种算法。如图 5，可以完全随机生成，也可

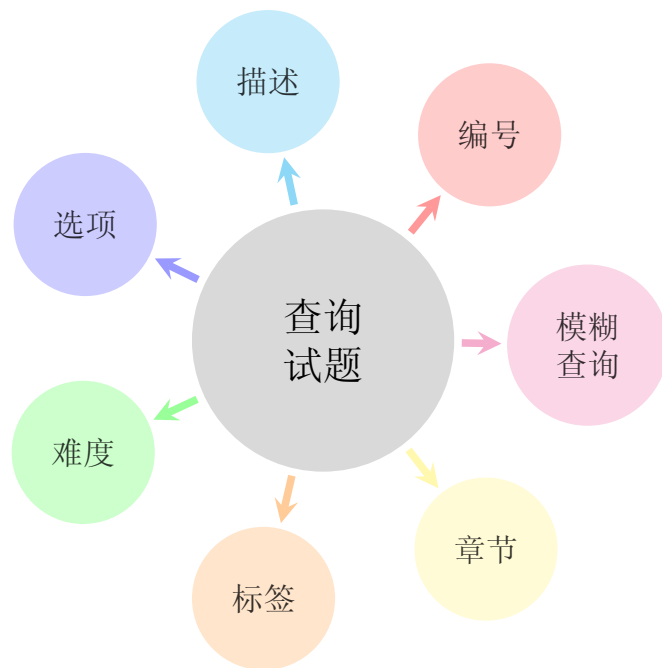


图 4: 试题查询模块

以指定标签、章节、难度对试卷进行生成。另外，这里也可以查看已生成试卷列表和内容。

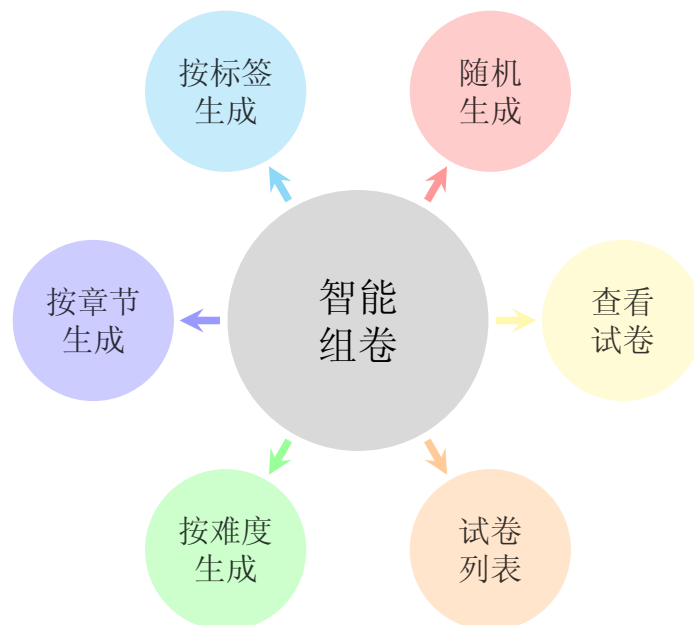


图 5: 试卷模块

2 概要设计

实际上，C 作为系统编程语言，久经考验，使用好的方法便可以写出有良好表现和维护性的代码。按传统的过程式编程方法，对于一个较大型的项目来说，有点难以驾驭。

为了程序维护和修改方便，以及程序通用性，我采用了面向对象的编程方法，参考了面向对象和设计模式的相关书籍资料¹。对于 C 语言来说，实现面向对象有点麻烦²，不过基本上还是模块化编程的思想³。

2.1 类的划分

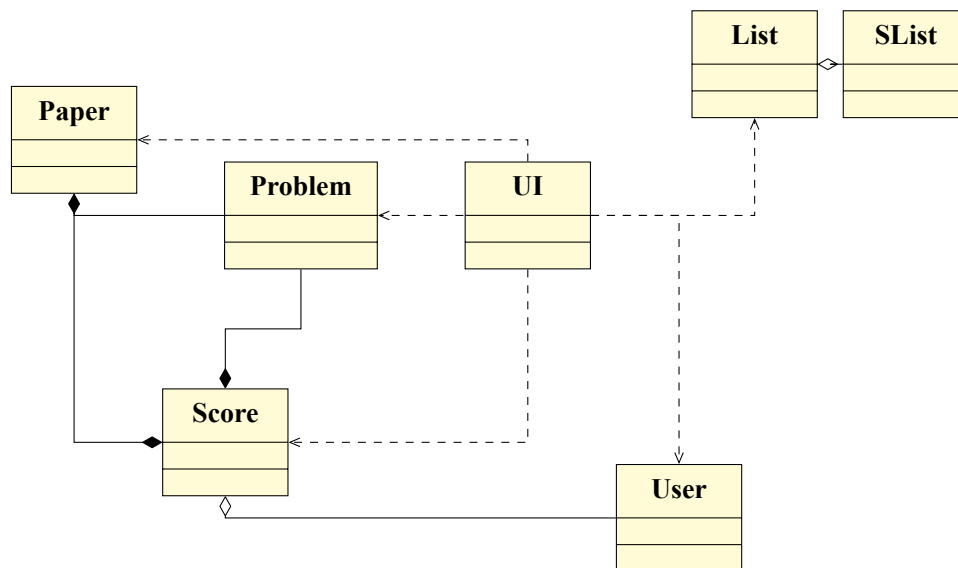


图 6: 各个类之间的关系

通过研究程序中出现的对象，我们可以抽象出若干个类。图 6 展示了各个类之间的关系。

首先，我决定使用 UI 类来控制所有与用户交互有关的操作，所有屏幕可显示字符串都存放在 UI 类中，便于修改和移植。

最重要的类是 Problem 类，提供题目存取和相关操作。Problem 类聚合可以生成试卷，即 Paper 类，用来存取试卷信息。学生和教师共用 User 类来存取，其中学生的成绩使用 Score 类存取。Score 类由 Paper 类和 User 类聚合而成，并且依赖于 Problem 类。

¹主要是《设计模式解析》[1] [2]。

²学习《系统程序员成长计划》[3] 中介绍的经验。

³参考《模块化 C：怎样编写可重用可维护的 C 语言代码》[4]。

由于没有数据库，我设计一个 List 类来实现对文件的读取、写入和增、删、改、查操作。通过 UI 类将 List 类与其他类连接起来操作（这里也是本程序设计的不佳之处）。List 类是对单链表 SList 类的一个封装。

2.2 List 类

List 类（图 7）存储一个链表，记录链表元素个数和元素中最大编号。除了初始化和销毁操作之外，提供从文件中读取、向文件中写入功能。对于链表本身，提供插入、删除、查找和遍历、查找遍历功能，通过传入回调函数来实现具体功能。

List	
- slist: SList *	//单链表头指针
+ max_id: int	//链表中元素最大编号
+ count: int	//链表中元素个数
<hr/>	
+ list_new(): List *	//创建链表
+ list_free(list: List *): void	//销毁链表
+ read_file_to_list(filename: char *, list: List *, size: size_t): int	//读取文件到链表
- list_restore(list: List *): void	//排序整理链表
# file_data_count(file: FILE *, size: size_t): int	//获取文件大小
+ write_list_to_file(filename: char *, list: List *, size: size_t): int	//写出链表到文件
+ list_insert(list: List *, p: void *): int	//插入到链表
+ list_remove(list: List *, find: 回调函数, data: void *): void *	//从列表删除
+ list_find(list: List *, find: 回调函数, data: void *): void *	//在列表中查找
+ list_each_call(list: List *, call: 回调函数, data: void *): void *	//遍历链表
+ list_find_each_call(list: List *, find: 回调, data: void *, call: 回调, userdata: void *): void *	//遍历链表中符合条件的元素

图 7: List 类

List 类实际上是对单链表 SList 类的一个封装，同时提供了文件操作，在程序中实际上是作为数据库类存在的。

2.3 SList 类

SList 类是 GNU Libltdl 库的组成部分之一，经过了长时间的考验，是非常健壮的。尊重 LGPL 协议，我未对其代码做任何改动。本来我是想改动 SList 类来实现现在 List 类的功能的，但是我还是决定尊重版权，并且提高代码重用性。

图 8 描述了 SList 类。每个 SList 对象都是一个链表元素，包含指向下一元素地址的指针和本结点元素的数据指针。

SList 类提供了链表操作的通用方法，包括附加、连接、销毁、删除、查找、

SList	
- next: <i>SList *</i>	//指向下一元素地址的指针
+ userdata: <i>const void *</i>	//指向数据域指针
+ slist_concat(head: <i>SList *</i> , tail: <i>SList *</i>): <i>SList *</i>	//链接两个单链表
+ slist_cons(item: <i>SList *</i> , slist: <i>SList *</i>): <i>SList *</i>	//插入链表元素
+ slist_delete(slist: <i>SList *</i> , delete_fct: 回调函数): <i>SList *</i>	//销毁链表
+ slist_remove(phead: <i>SList **</i> , find: 回调函数, matchdata: <i>void *</i>): <i>SList *</i>	//删除指定元素
+ slist_reverse(slist: <i>SList *</i>): <i>SList *</i>	//反转链表
- slist_sort_merge(left: <i>SList *</i> , right: <i>SList *</i> , compare: 比较函数, userdata: <i>void *</i>): <i>SList *</i>	
+ slist_sort(slist: <i>SList *</i> , compare: 比较函数, userdata: <i>void *</i>): <i>SList *</i>	//快速排序
+ slist_tail(slist: <i>SList *</i>): <i>SList *</i>	//取下一个元素
+ slist_nth(slist: <i>SList *</i> , n: <i>size_t</i>): <i>SList *</i>	//取第 <i>n</i> 个元素
+ slist_find(slist: <i>SList *</i> , find: 回调函数, matchdata: <i>void *</i>): <i>void *</i>	//查找指定元素
+ slist_length(slist: <i>SList *</i>): <i>size_t</i>	//取链表长度
+ slist_foreach(slist: <i>SList *</i> , find: 回调函数, userdata: <i>void *</i>): <i>void *</i>	//遍历每个元素
+ slist_box(userdata: <i>const void *</i>): <i>SList *</i>	//封装数据
+ slist_unbox(item: <i>SList *</i>): <i>void *</i>	//拆包数据

图 8: SList 类

遍历操作，还可以对其进行排序（使用快速排序）、反转，一般的取长度和取指定元素方法也必不可少。

2.4 Problem 类

Problem	
+ id: <i>int</i>	//题目编号
+ des: <i>char[]</i>	//题目描述
+ opt: <i>char[4][]</i>	//选项
+ ans: <i>char</i>	//答案
+ dif: <i>short</i>	//难度系数
+ tag: <i>short</i>	//知识点标签
+ chapter: <i>short</i>	//章
+ section: <i>short</i>	//节
+ problem_new(): <i>Problem *</i>	//实例化题目
+ problem_read_file(list: <i>List *</i>): <i>int</i>	//读取题目数据库
+ problem_write_file(list: <i>List *</i>): <i>int</i>	//写出题目数据库

图 9: Problem 类

图 9 所示的 Problem 类，有着题目的数据域。为了简便起见，采用四个选项的单选形式。

题目可以通过访问数据域直接操作，所以其本身的方法并不是很多，都托管给 UI 类了。

2.5 Paper 类

Paper	
+ id: <i>int</i>	// 试卷编号
+ length: <i>int</i>	// 试卷题目数
+ pid: <i>int</i> []	// 题目编号数组
+ title: <i>char</i> []	// 试卷标题
<hr/>	
+ papar_new(): <i>Papar *</i>	// 实例化试卷
+ papar_free(pa: <i>Paper *</i>): <i>void *</i>	// 析构试卷
+ papar_read_list(list: <i>List *</i>): <i>int</i>	// 读取试卷数据库
+ papar_write_list(list: <i>List *</i>): <i>int</i>	// 写出试卷数据库
+ paper_insert_pid(pa: <i>Paper *</i> , pid: <i>int</i>): <i>int</i>	// 插入题目
+ paper_problem_call(pa: <i>Paper *</i> , list: <i>List *</i> , call: 回调, userdata: <i>void *</i>): <i>void</i>	// 遍历试卷题目
+ paper_generate_random(pa: <i>Paper *</i> , list: <i>List *</i> , n: <i>int</i>): <i>int</i>	// 随机生成试卷
+ paper_generate_tags(pa: <i>Paper *</i> , list: <i>List *</i> , n: <i>int</i> , tags: <i>int</i> [], m: <i>int</i>): <i>int</i>	// 按标签生成试卷
+ paper_generate_secs(pa: <i>Paper *</i> , list: <i>List *</i> , n: <i>int</i> , secs: <i>double</i> [], m: <i>int</i>): <i>int</i>	// 按章节生成试卷
+ paper_generate_dif(pa: <i>Paper *</i> , list: <i>List *</i> , n: <i>int</i> , a: <i>int</i> , b: <i>int</i>): <i>int</i>	// 在难度区间 [a, b] 生成试卷

图 10: Paper 类

图 10 所示的 Paper 类存储试卷，用数组来存储题目数据库中的题目编号（所以试卷依赖于题目）。

在试卷中插入题目，要求题目不重复，方法会遍历题目数组保证题号唯一。生成试卷时指定的题目数量如果过多，会自动缩减到合适大小。

2.6 User 类

User	
+ id: <i>int</i>	// 用户编号
+ username: <i>char</i> []	// 用户名
- passwd: <i>char</i> []	// 密码
+ teacher: <i>bool</i>	// 是否为教师
<hr/>	
+ user_new(): <i>User *</i>	// 实例化用户
+ user_free(u: <i>User *</i>): <i>void *</i>	// 析构用户
- user_init(): <i>int</i>	// 初始化根用户
+ user_reg(u: <i>User *</i>): <i>int</i>	// 用户注册
+ user_login(u: <i>User *</i>): <i>int</i>	// 用户登录

图 11: User 类

图 11 是 User 类，这个类自己在内部实现了文件操作。

密码本来应该是密文存储，但是没有必要为这样一个玩具系统写 Hash 函数，不如明文存储。这里目的是展示用户登录逻辑，没有必有实现特别复杂的用户管理。

2.7 Score 类

Score	
+ id: <i>int</i>	//成绩编号
+ user_id: <i>int</i>	//用户编号
+ username: <i>char[]</i>	//用户名
+ paper_id: <i>int</i>	//试卷编号
+ paper_count: <i>int</i>	//试卷题目数
+ answer: <i>char[]</i>	//用户提交的答案
+ right: <i>int</i>	//用户题目正确个数
+ date: <i>time_t</i>	//开始做题时间
- now: <i>int</i>	//当前做到的题目指针
+ score_new(<i>u: User *</i> , <i>p: Paper *</i>): <i>Score *</i> //实例化成绩	
+ score_read_list(<i>list: List *</i>): <i>int</i> //读取成绩数据库	
+ score_write_list(<i>list: List *</i>): <i>int</i> //写出成绩数据库	
+ score_did(<i>s: Score *</i> , <i>c: char</i> , <i>ans: char</i>): <i>int</i> //记录做题	

图 12: Score 类

图 12 是 Score 类，实现了成绩的记录。

2.8 公共回调函数

Slist 类提供了两个接口，其中回调函数接口提供了对 SList 对象的带参数调用操作，排序比较函数接口用于 SList 对象的大小比较，类似于 `qsort()` 的比较函数。

«interface» SListCompare	
# item1: <i>SList *</i>	
# item2: <i>SList *</i>	
# userdata: <i>void *</i>	
+ cmp_id() //比较编号	

图 13: 比较函数

比较函数用于排序，目前只有按编号排序（图 13），并且隐藏在读入方法里调用。其他的关键字排序可以以此接口来编写调用，因为用途不大，不再编写。

回调函数可以用于查找和遍历，主要传递给 `foreach` 方法进行调用。由于传

«interface» SListCallback	
# item: <i>SList</i> *	
# userdata: <i>void</i> *	
+ write_userdata()	//写出数据到文件 (遍历)
+ by_id()	//各类编号 (查找)
+ by_des()	//题目描述 (查找)
+ by_opt()	//题目选项 (查找)
+ by_difr()	//题目难度区间 (查找)
+ by_difs()	//题目难度 (查找)
+ by_tags()	//题目标签 (查找)
+ by_secs()	//题目章节 (查找)
+ by_mul()	//模糊查询 (查找)
+ by_user_id()	//试卷中的用户编号 (查找)
+ ui_each_problem_show()	//显示单个题目 (遍历)
+ ui_each_paper_show()	//显示单张试卷 (遍历)
+ ui_each_score_show()	//显示单条成绩 (遍历)
+ ui_do_problem()	//做单个题目 (遍历)

图 14: 回调函数

递进去的参数只有一个泛型指针，所以我设计了 `sel_num` 和 `ID`、`Block` 数据类型来存储和传递参数，具体实现比较丑陋，还有待改进。遵循此接口的方法列在图 14 中。

2.9 UI 类

UI 类实际上相当于一个上帝类，掌控全局，并且可以访问其他类的几乎所有方法和属性。其本身没有属性，也不需要被实例化，是唯一的用户界面接口。图 15 是 UI 类的所有方法，这些方法都是与界面有关联的。方法的嵌套层次与界面菜单一致。

2.10 辅助方法

还有一些辅助方法，用于通用的终端控制操作，分散在不同的类中，如图 16。其他一些辅助的数值操作的方法不再表述。

UI	
ui_index(): void	//主界面
ui_do_login(u: User *): int	//登录界面
ui_student_login(): void	//学生登录
ui_student(u: User *): void	//学生主界面
ui_student_test(u: User *): void	//做卷子
ui_student_score(u: User *): void	//看成绩
ui_teacher_login(): void	//教师登录
ui_teacher(): void	//教师主界面
ui_teacher_view(): void	//浏览试题
ui_output_count(list: List *): void	//输出题目数等
ui_output_problem(p: Problem *, show_more: bool): void	//输出试题
ui_teacher_insert(): void	//插入试题 (增)
ui_input_problem(): Problem *	//输入题目
ui_teacher_delete(): void	//删除试题 (删)
ui_teacher_update(): void	//修改试题 (改)
ui_edit_problem(p: Problem *): void	//编辑题目
ui_teacher_select(): void	//查询试题 (查)
ui_select_id(list: List *): Problem *	//按编号查询
ui_select_des(list: List *): int	//按题目描述查询
ui_select_opt(list: List *): int	//按题目选项查询
ui_select_dif(list: List *): int	//按题目难度查询
ui_select_tag(list: List *): int	//按题目标签查询
ui_select_sec(list: List *): int	//按题目章节查询
ui_select_mul(list: List *): int	//模糊查询
ui_select_output(list: List *, 回调find, void *matchdata): int	//查询到时输出题目
ui_teacher_generate(): void	//智能组卷
ui_generate_random(list: List *): void	//随机生成
ui_generate_tags(list: List *): void	//按标签生成
ui_generate_secs(list: List *): void	//按章节生成
ui_generate_dif(list: List *): void	//按难度区间生成
ui_paper_list(): void	//试卷列表
ui_paper_view(problist: List *): void	//查看试卷
ui_paper_save(pa: Paper *): int	//保存试卷
ui_teacher_score(): void	//成绩分析
ui_output_score(s: Score *): void	//输出成绩
ui_help(): void	//帮助
ui_about(): void	//关于

图 15: UI 类

addon	
gotn(): bool	//清除遗留换行符
getpass(prompt: char *): char *	//掩码输入密码
read_string(s: char *, n: size_t): char *	//输入字符串
cls(): void	//清屏
pause(): void	//暂停
dif2star(): char *	//把难度转成星级
ui_input_number(): int	//输入数字
ui_input_ans(): char	//输入选项字母

图 16: 辅助方法

3 详细设计

3.1 已实现功能

整个系统有七个大类，上百个方法。由于时间关系，不能够一一说明，我这里拣几个比较主要的方法来说明其程序流程，其他代码实现还请查看附录 A。

3.1.1 插入题目

图 17 展示了插入题目的流程，首先实例化一个链表，将题目数据库读入，提示用户输入题目信息，将题目编号设置为最大编号加一，插入链表尾部，最后将链表写入文件，释放内存。

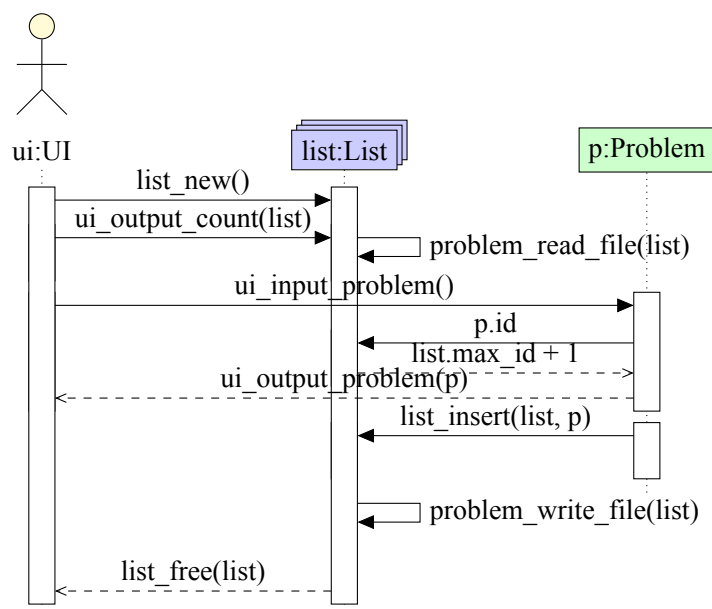


图 17: `ui_teacher_insert()`

删除和修改题目与此同理，都是将文件数据全部读入实例化的链表中，对链表进行操作，再将链表全部写入文件。这对于小数据量是有效且便利的。老师提出，删除时不应读入内存，存在大量数据的情况会使内存溢出。为了换取这样的空间代价，我们不得不用时间上非常慢的对硬盘文件的读取来实现，这也得不偿失。实际上，对文件的读写在操作系统里会映射为对内存的读写，操作系统还是会把文件读入内存（虽然不是全部，但一般文件没有大到足以分页），这种优化实际上是徒劳的。基于这种思想的删除和修改在实验最后一题里有实现（见实验报告），但不适用于本程序。

3.1.2 模糊查询

模糊查询实际上是统合了所有属性的查询，这种查询方式很类似于搜索引擎。当然相对于搜索引擎来说，还是非常简单的，不需要语法分析什么的，只是简单地关键字比对。

在输入关键字 `key` 之后，只需要调用 `ui_select_output(list, by_mul, key)` 即可将所有查找结果输出出来。其中回调函数 `by_mul` 实现如 Listing 1，将关键字与各个域进行比对并返回元素本身。

Listing 1: `by_mul()`

```
1 void *by_mul(SList *item, void *data)
2 {
3     Problem *p = (Problem *)item->userdata;
4     char *key = (char *)data;
5     int a, b;
6     if ('0' <= key[0] && key[0] <= '9') {
7         sscanf(key, "%d", &a);
8         if (p->id == a || p->tag == a || p->dif == a
9             || p->chapter == a || p->section == a) {
10             return item;
11         }
12         if (sscanf(key, "%d.%d", &a, &b) == 2
13             && p->chapter == a && p->section == b) {
14             return item;
15         }
16     }
17     if (by_des(item, data) != NULL) {
18         return item;
19     }
20     if (by_opt(item, data) != NULL) {
21         return item;
22     }
23     return NULL;
24 }
```

3.1.3 生成试卷

生成试卷的算法有多个，其中按标签和按章节生成的算法比较类似。Listing 2 是按标签生成的算法，将题目列表和指定的题目数、标签列表传入，限制每个标签的题目数量不超过 $n/m + 1$ ，遍历题目列表凡是符合条件的加入试卷，直到满足数量或超出限制。

Listing 2: `paper_generate_tags()`

```
1 int paper_generate_tags(Paper *pa, List *list, int n, int tags[],
2     int m)
3 {
```

```

3     if (list->count < n) {
4         n = list->count;
5     }
6     SList *s = list->slist;
7     Problem *p = NULL;
8     int i = 0, j = 0;
9     int k = n / m + 1;
10    for (i = 0; i < m; i++) {
11        j = 0;
12        while ((s = (SList *)slist_find(s, by_tags, tags + i)) !=
13            NULL) {
14            p = (Problem *)s->userdata;
15            if (p->id == paper_insert_pid(pa, p->id)) {
16                j++;
17                if (j >= k) {
18                    break;
19                }
20            }
21            if (pa->length >= n) {
22                goto end;
23            }
24            s = slist_tail(s);
25        }
26    end:
27    return pa->length;
28 }

```

随机生成的算法比较简单，每次生成合法的随机题目编号插入试卷，直到达到指定数量为止。

按难度区间生成的算法，需要在区间中均匀分布题目难度，也有一个控制因子在起作用。

3.1.4 用户登录

用户登录逻辑比较简单，如果没有用户数据库，就注册一个教师用户 root；如果没有该用户名，就注册一个学生类型的账户。注册完后直接登录。

逻辑比较简单，方便用户操作，只是为了简单分模块逻辑。密码明文存储，也不提供修改密码什么的操作，没有必要。

3.1.5 学生考试

学生考试功能是 UI 类最复杂的，一共对三个类进行了操作，实例化了五个对象，逻辑比较复杂。Listing 3 展示了该方法的具体实现代码，首先读取试卷数据库，让用户选择试卷编号，实例化试卷；接着将试卷对应的题目读取出来，用用

户和试卷来实例化成绩类，对每道题目调用 `ui_do_problem` 来做题并记录；最后读取成绩列表，插入成绩再写回硬盘。

Listing 3: `ui_student_test()`

```
1 void ui_student_test(User *u)
2 {
3     List *list = list_new();
4     if (paper_read_list(list) < 0) {
5         perror("读取试卷数据库失败");
6         return;
7     }
8     list_each_call(list, ui_each_paper_show, list);
9     int id = -1;
10    printf("试卷编号:\n$ ");
11    id = ui_input_number();
12    Paper *p = (Paper *)list_find(list, by_id, &id);
13    if (p == NULL) {
14        perror("没有这套试卷");
15        return;
16    }
17    List *problist = list_new();
18    if (problem_read_file(problist) < 0) {
19        perror("读取题目数据库失败");
20        return;
21    }
22    cls();
23    Score *s = score_new(u, p);
24    paper_problem_call(p, problist, ui_do_problem, s);
25    list_free(list);
26    list_free(problist);
27
28    List *scorelist = list_new();
29    if (score_read_list(scorelist) < 0) {
30        perror("读取成绩数据库失败");
31        goto end;
32    }
33    s->id = scorelist->max_id + 1;
34    if (list_insert(scorelist, s) < 0) {
35        perror("插入成绩失败");
36        goto end;
37    }
38    if (score_write_file(scorelist) < 0) {
39        perror("写入成绩数据库失败");
40        goto end;
41    }
42 end:
43    printf("考试完成! \n");
44    ui_output_score(s);
45    list_free(scorelist);
46    pause();
47 }
```

3.2 未实现功能

3.2.1 多项选择题

系统完成的时候，老师提出了加入多选题功能的要求。由于时间关系，不再具体实现，大概叙述一下实现的思路。

在存储单项选择题的时候，我固定了四个选项，并且用一个字符表示正确选项的编号。实际上，选项可以增长到无限，正确答案也可能不止一个。那么如何存储呢？考虑使用 n 个二进制位来标识每个选项的正确与否，这样一个8位的字符实际上可以保存8个选项的状态。要实现选项答案的分离，只需要用到简单的位操作。Listing 4 实现了取二进制指定位的操作，并且支持用多个字符即字符串来存储更多选项。

Listing 4: get_bit()

```
1 unsigned char get_bit(unsigned char *bits, unsigned long i) {  
2     return (bits[i / 8] >> i % 8) & 1;  
3 }
```

至于多项选择题的输入判断，可以直接读入字符串，进行大小写转换，取其ASCII 编码值直接对应到每个选项中，取对应位进行对比判断即可。

至于评分标准，应该支持自定义。一般地，多选只要有错选就不给分，不选不给分。特殊的，少选的给部分分或者不给分。新课标高考物理的评分标准是“全部选对的得6分，选对但不全的得3分，有选错的得0分”。类似于这样的标准可以通过选项配置来解决。

3.2.2 词法分析

实际上我们的查询可以做得更好，利用状态机来将用户一定规则的输入与数据进行匹配，可以实现更为丰富的查询，甚至实现批量删除和修改。当然我们会发现这些功能都是 SQL 的标准配置。个人实现一个 SQL 还是比较困难的，尤其是没有编译原理的相关基础，所以这部分就放弃不做了。

3.2.3 图形界面

其实不管怎么说，这个系统的难度在于使用了大量的底层操作。终端界面有不少限制，并不适合这种互动性非常高的应用。想象了许多功能，在终端中运营都不会达到非常好的效果。终端的优势在于使用脚本语言处理大量文本数据，进

行系统编程和系统管理维护。以普通用户为中心的应用程序，应当拥有良好的图形用户界面⁴。

比较好的解决方案是采用 C/S 架构，分离底层与用户界面。本程序这点做的非常不好，也是水平所限。

⁴关于用户界面的设计，参考《界面设计模式》[5]。

4 调试分析

基于版本控制的编程开发是非常有效的，Git 和 Github 为我的程序代码管理提供了很大的帮助。

程序的调试时一个需要耐心而艰苦反复的工作。我使用多种方法进行调试：

静态 使用 `fprintf(stderr, ...)` 函数来输出中间变量的值；

GDB 在命令行下使用 `gdb` 来跟踪程序并设置端点等；

IDE 后期换用了 Visual Studio 2013，其调试的功能也是非常强大。

我遵循了类似敏捷的开发方式，保证每个版本可用的情况下进行迭代开发。每增加一个新功能我都会在版本号加上 0.0.1，所以通过版本号可以看出我已经进行了 40 多次的迭代开发了，大概用了一个星期的时间。

有几次迭代实际上不能正常运行，所以没有提交到版本库。以下是 Git 的提交历史记录：

1. 2014-07-02 16:49:30 Init.
2. 2014-07-02 17:42:48 Use
3. 2014-07-03 14:15:42 界面可用
4. 2014-07-03 18:46:13 实现浏览和插入
5. 2014-07-04 16:18:31 兼容 VS 和 nmake
6. 2014-07-04 16:50:27 更好的解耦
7. 2014-07-04 18:29:48 实现修改
8. 2014-07-05 14:16:40 修补内存泄露
9. 2014-07-05 15:12:02 实现删除
10. 2014-07-05 18:10:24 实现条件查询
11. 2014-07-05 21:13:37 实现所有单项查询
12. 2014-07-05 21:53:33 实现模糊查询
13. 2014-07-05 22:54:17 增强输入稳定性
14. 2014-07-05 23:21:13 命令行下跨平台兼容
15. 2014-07-06 16:36:48 开始编写试卷类
16. 2014-07-06 18:20:21 实现随机组题
17. 2014-07-06 19:27:57 布置好主要界面
18. 2014-07-07 11:29:25 实现几个组卷算法

19. 2014-07-07 11:36:28 实现组卷
20. 2014-07-07 16:27:41 不稳定的用户登录功能
21. 2014-07-07 17:40:38 用户登录注册可用
22. 2014-07-07 21:05:32 使用封装的链表操作题目
23. 2014-07-07 22:48:45 对用户界面隐藏单链表
24. 2014-07-08 10:14:52 试卷封装
25. 2014-07-08 11:24:45 做题界面和成绩初始化
26. 2014-07-08 14:22:35 考试成绩读写
27. 2014-07-08 14:45:52 完成所有既定功能
28. 2014-07-09 11:01:06 整理程序架构

程序编写过程中，必然出现过许多 Bug。当然，这些 Bug 都已经被及时修复了，如今复现难度比较大。

课程设计的调试目的是学会使用调试工具和学会修补 Bug，我想目的已经达到，不必要再走形式，详细地把调试过程列出来了。

5 测试结果

由于功能比较多，这里指展示几个比较关键的界面和测试情况。功能测试在编写程序时已有少量工作，暂时没有时间和能力做大量的测试。

程序使用简单的数字选择菜单，要求用户输入的时候会显示美元符号 \$ 当做命令提示符。

5.1 教师模块

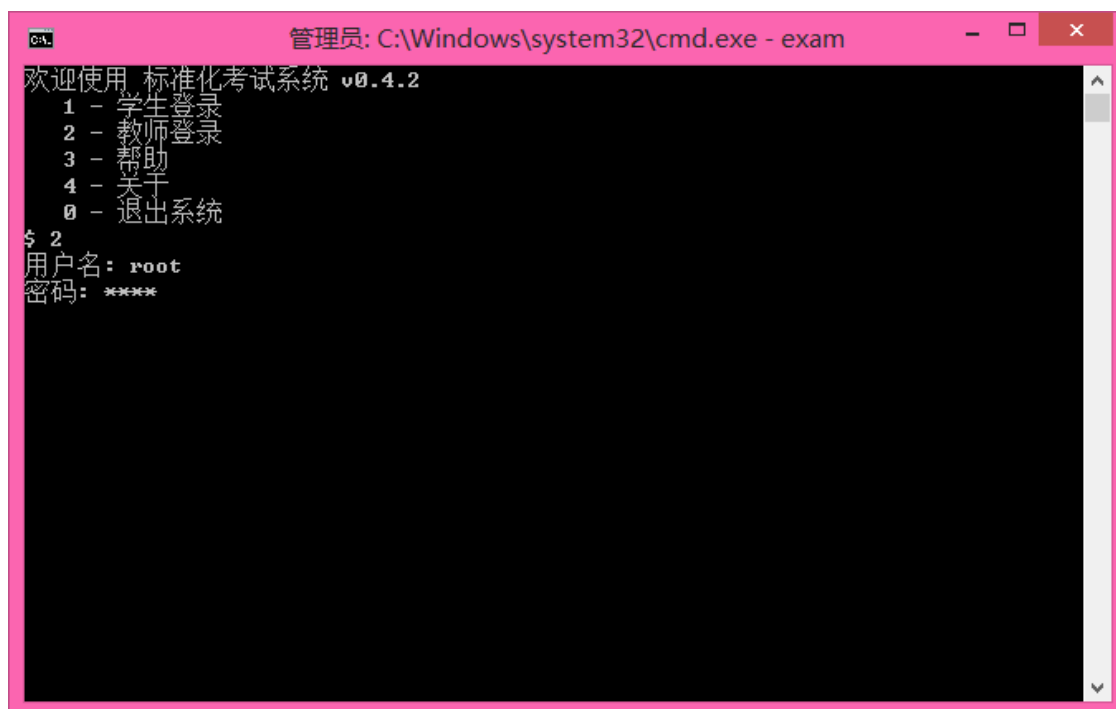


图 18: 教师登录

如图 18，默认教师用户为 root，密码 root，输入之后即可进入教师界面。

5.1.1 题目增删改

进入教师界面，如图 19，选择“添加试题（增）”，可以添加题目。输入题目信息时已经处理了大部分异常输入情况，在输入数字的地方必须输入数字，输入选项字母只需输入 A~D（大小写均可），输入字符串中可以输入空格。

如图 20，选择“浏览试题”，可以显示简短的题目信息列表。

如图 21，选择“修改试题（改）”后输入题号，可以修改已有题目。对于不想修改的信息可以按回车直接跳过，需要修改的若有输入也会有容错处理。

```
管理员: C:\Windows\system32\cmd.exe - exam
欢迎使用 标准化考试系统 v0.4.2
1 - 浏览试题
2 - 添加试题 (增)
3 - 删除试题 (删)
4 - 修改试题 (改)
5 - 查询试题 (查)
6 - 智能组卷
7 - 成绩分析
9 - 返回上一级
0 - 退出系统
$ 2
数据库中有 1 条记录, 最大编号为 1
请输入题目相关信息:
题目描述:
$ 哪一个解释性语言?
[选项 A]:
$ C++
[选项 B]:
$ Python
[选项 C]:
$ Go
[选项 D]:
$ Basic
正确答案字母序号:
$ b
题目难度(1--10):
$ 5
知识点标签<数字编号>:
$ 1
知识点章节<章.节>:
$ 1.5
2. 哪一个解释性语言?
[A]. C++
[B]. Python
[C]. Go
[D]. Basic
答案: B 难度: ★★☆☆ 标签: 1 章节: 1.5
插入题目 2 成功!
请按任意键继续. . .
```

图 19: 插入题目

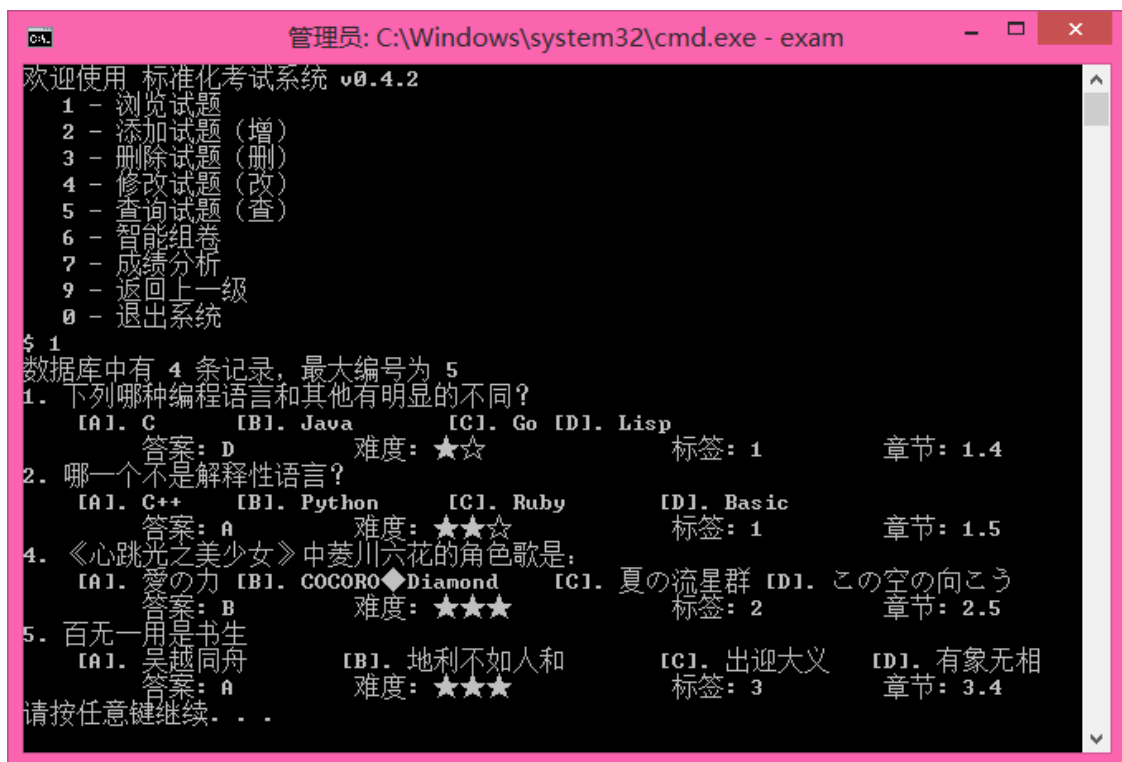


图 20: 查看题目

如图 22，选择“删除试题（删）”后输入题号，可以删除题库中的题目。由于删除操作的危险性，提示确认再删除。

5.1.2 题目查询

如图 23，选择“题目难度”，可以按难度查询试题。你可以输入不等号后跟数字，比如这里输入了 < 5，就能显示出所有难度小于5的题目。不等号可以是 < <= > >= = == != <>，如果输错，默认为等于。

如图 24，选择“模糊查询”，允许以一个关键字来查询所有题目属性。

5.1.3 生成试卷

如图 25，选择“随机生成”，输入题目数量和试卷标题，可以生成试卷。

5.2 学生模块

如图 26，在主界面选择“学生登录”，任意输入用户名和密码，系统会自动建立学生用户并登录学生界面。

```
管理员: C:\Windows\system32\cmd.exe - exam
$ 4
数据库中有 3 条记录, 最大编号为 4
题目编号:
$ 2
2. 哪一个解释性语言?
  [A]. C++
  [B]. Python
  [C]. Go
  [D]. Basic
答案: B 难度: ★★☆☆ 标签: 1 章节: 1.5
修改题目信息 (直接回车表示不修改):
题目描述: 哪一个解释性语言?
$ 哪一个不是解释性语言?
[选项 A]: C++
$
[选项 B]: Python
$
[选项 C]: Go
$ Ruby
[选项 D]: Basic
$
正确答案: B
$ a
题目难度: <5> ★★☆☆
$
知识点标签: 1
$
知识点章节: 1.5
$
2. 哪一个不是解释性语言?
  [A]. C++
  [B]. Python
  [C]. Ruby
  [D]. Basic
答案: A 难度: ★★☆☆ 标签: 1 章节: 1.5
修改题目 2 成功!
请按任意键继续. . .
```

图 21: 修改题目

```
C:\Windows\system32\cmd.exe - exam
欢迎使用 标准化考试系统 v0.4.2
1 - 浏览试题
2 - 添加试题 (增)
3 - 删除试题 (删)
4 - 修改试题 (改)
5 - 查询试题 (查)
6 - 智能组卷
7 - 成绩分析
9 - 返回上一级
0 - 退出系统
$ 3
数据库中有 4 条记录, 最大编号为 5
题目编号:
$ 5
5. 百无一用是书生
   [A]. 吴越同舟
   [B]. 地利不如人和
   [C]. 出迎大义
   [D]. 有象无相
答案: A 难度: ★★★ 标签: 3 章节: 3.4
你确定要删除该题吗? 输入 'y' 或 'Y' 以确认:
$ y
删除题目 5 成功!
请按任意键继续. . .
```

图 22: 删除题目

```
C:\Windows\system32\cmd.exe - exam
可以查询的关键词:
1 - 题目编号
2 - 题目描述
3 - 题目选项
4 - 题目难度
5 - 题目标签
6 - 题目章节
7 - 模糊查询
9 - 返回上一级
0 - 退出系统
$ 4
题目难度 (输入 < > = == <= >= != <> 后空格数字):
$ < 5
查找题目难度 < 5 的题目...
1. 下列哪种编程语言和其他有明显的不同?
   [A]. C [B]. Java [C]. Go [D]. Lisp
   答案: D 难度: ★☆ 标签: 1 章节: 1.4
2. 哪一个不是解释性语言?
   [A]. C++ [B]. Python [C]. Ruby [D]. Basic
   答案: A 难度: ★ 标签: 1 章节: 1.5
共查询到 2 个符合条件的结果。
请按任意键继续. . .
```

图 23: 按难度查询

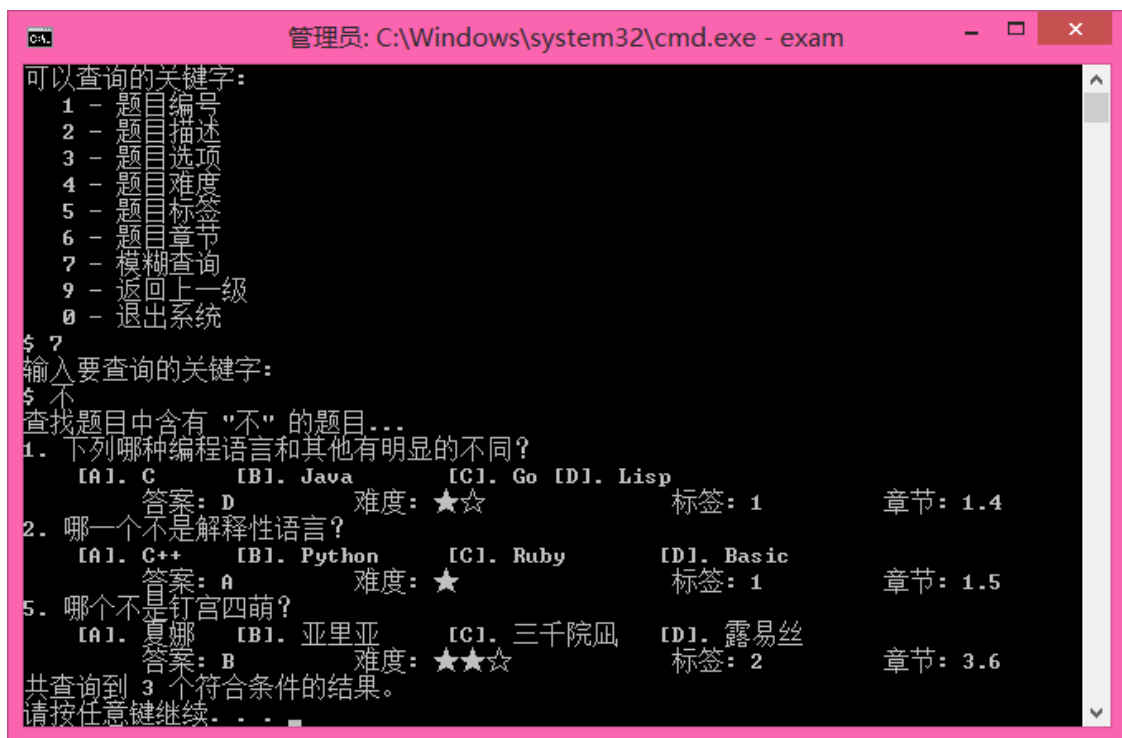


图 24: 模糊查询

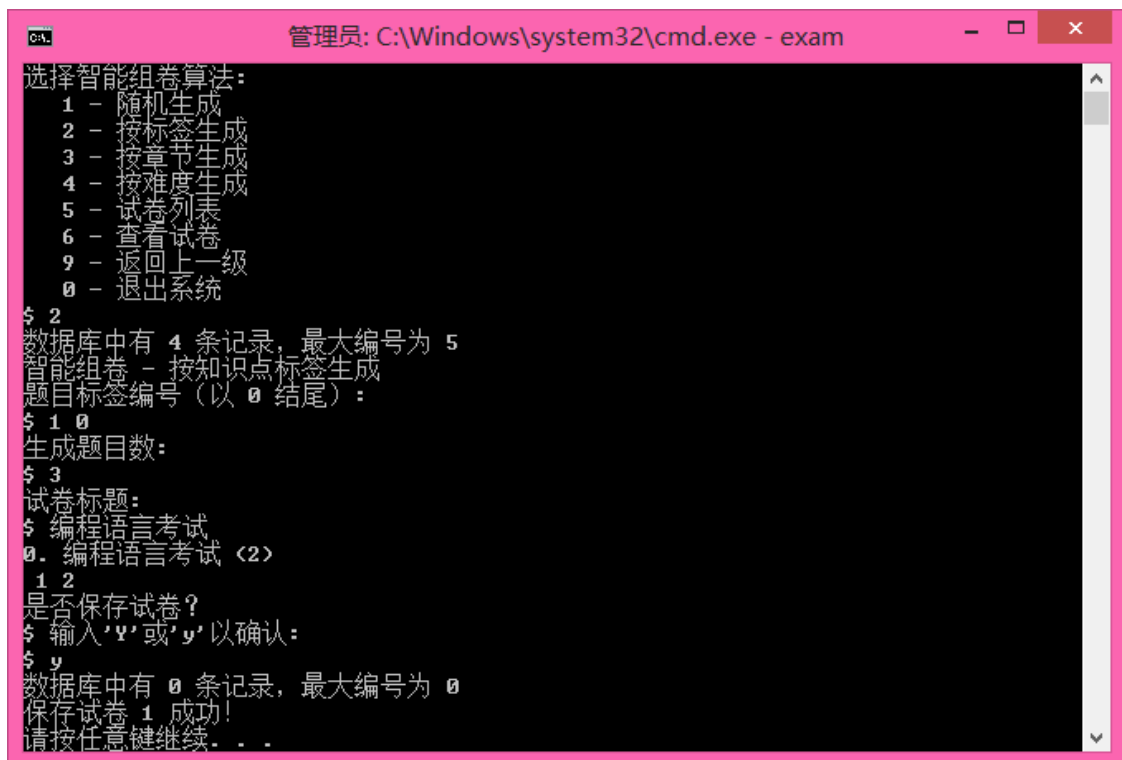


图 25: 随机生成试卷

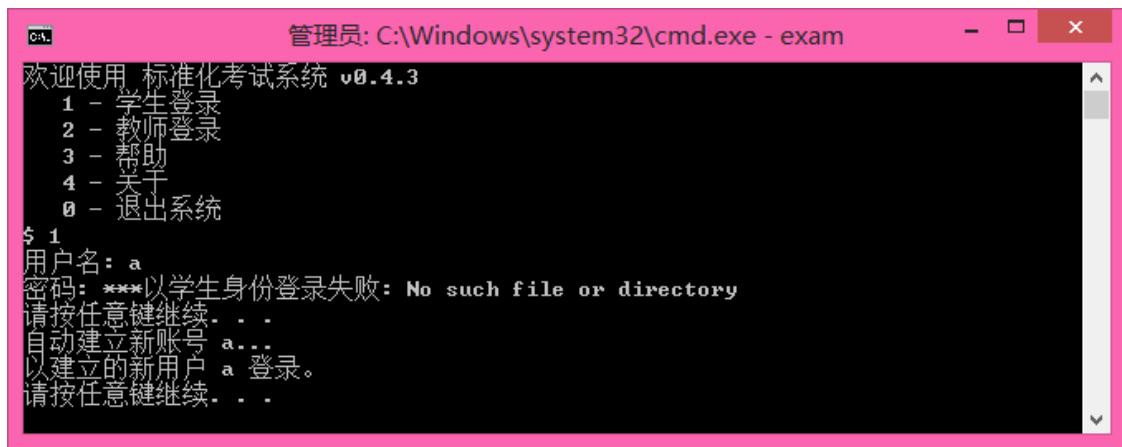


图 26: 学生登录

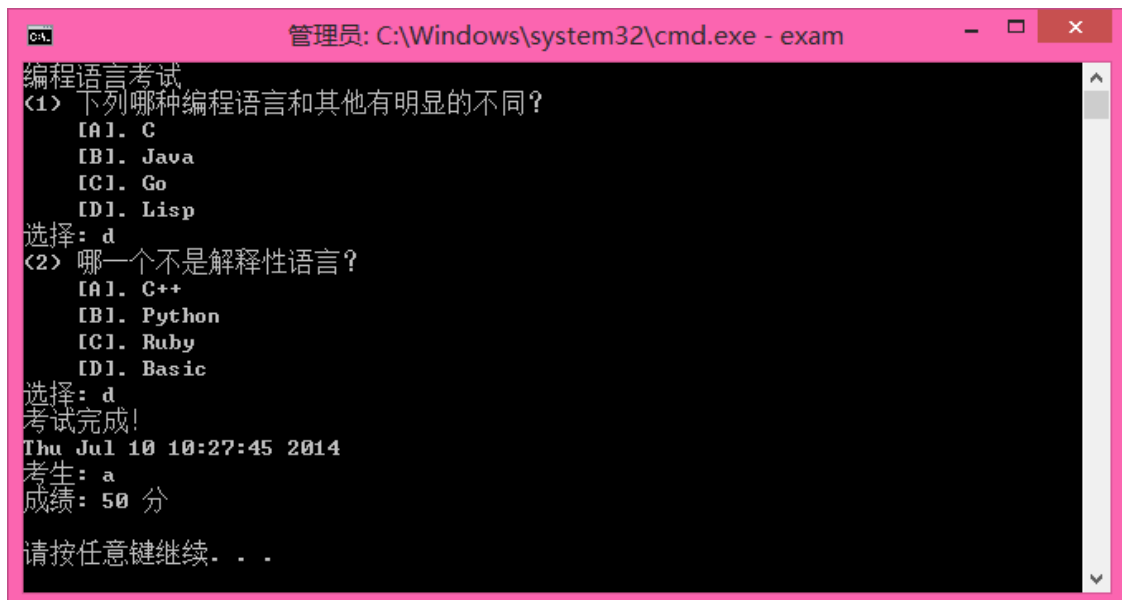
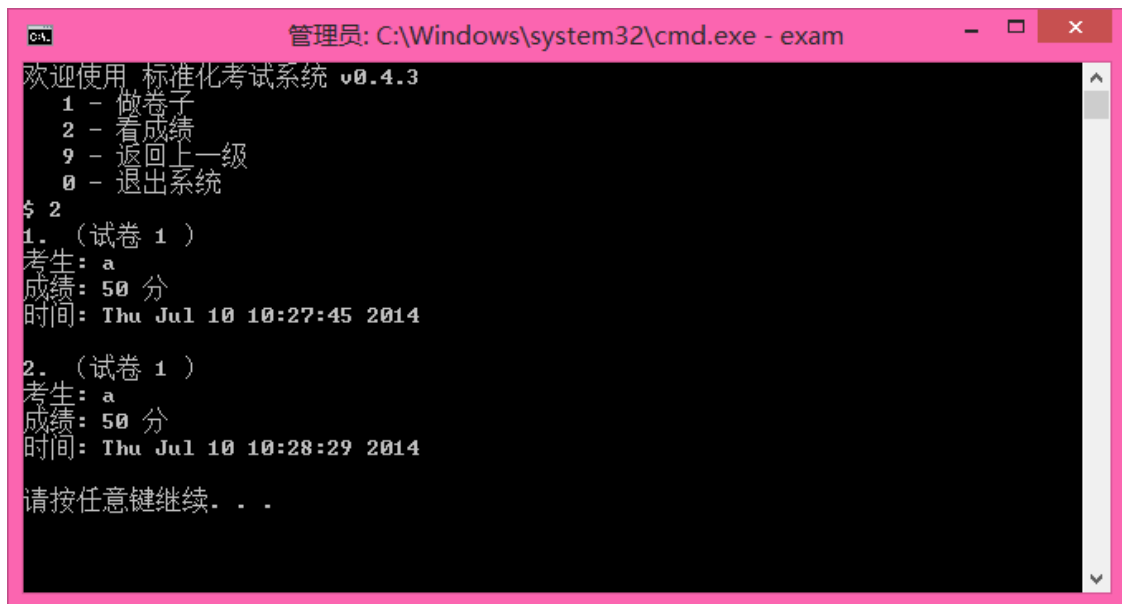


图 27: 学生参与考试

如图 27，进入“做卷子”后，选择试卷后进入考试界面。依次做完所有试题之后会显示成绩并保存成绩到数据库。



```
管理员: C:\Windows\system32\cmd.exe - exam
欢迎使用 标准化考试系统 v0.4.3
1 - 做卷子
2 - 看成绩
9 - 返回上一级
0 - 退出系统
$ 2
1. (试卷 1)
考生: a
成绩: 50 分
时间: Thu Jul 10 10:27:45 2014

2. (试卷 1)
考生: a
成绩: 50 分
时间: Thu Jul 10 10:28:29 2014

请按任意键继续...
```

图 28: 成绩列表

如图 28，学生可以查看自己的成绩列表。

6 心得总结

(废话挺多, 不看也罢。)

我的编程经验, 大概有六年了吧。开始初中学习 Pascal 参加信息学竞赛, 高中开始改用 C/C++, 虽然没有在竞赛里拿到保送资格, 不过还是积累了一定的编程经验。工具语言也从初中的易语言到高中的 Python 和 PHP, 至今又非常喜欢 Ruby, 也学习了各种有趣的编程语言比如 Go、Lisp、JavaScript 之流。这次课程完成后, 预定要重新深入学习一下 C# 和 Java 了。

当然, 语言是很简单的东西。我认为, 重要的是这些语言相关的工具链。语言没有好的生态环境和社区支持, 没有优秀的强大的类库, 是很难发展起来的, 因为会很难用。C 语言之所以能走到今天, 其实完全是 UNIX 的功劳 [6]。

从前受王垠[7]的影响, 非常推崇 Linux 的生活方式, 顺便也接触到了 \TeX ——也就是这篇文档所用的排版软件。我用 \LaTeX 2_ε 已经有四年了吧, 期间这玩意儿一直没怎么成长, 我也只是用而已。

说一下这篇文档吧, 目录啊、交叉引用啊、参考文献啊都是自动生成的。页面格式也是经过精细控制得到的结果。最头疼的部分是画图, 尝试了数十个画图的宏包, 最终选定了 tikz-uml 来画 UML 图。实际上我不会什么 UML, OO 也是一知半解, 临阵磨枪, 反正硬着头皮上也就是这结果了。实在有的地方画不出来图, 就直接上代码了。当然我觉得使用伪代码更好, 不过本程序不怎么偏重于算法, 用伪代码也会显得很单调。倒是 smartdiagram 画出来需求分析的几个彩色圆圈挺好看的, 赏心悦目。这些宏包都是基于 TikZ & PGF 的, 也是足够强大。

为什么不用 Visio 之类的软件呢? 是因为我实在不喜欢这种图形界面的绘图软件画出来的图难以控制, 导出来生成一堆乱七八糟的代码, 实在恶心的受不了。当然话数学几何图形用 GeoGebra 是能生成非常好看的图形的, 可惜 UML 似乎没有这样好用的软件, 也可能是我没发现吧。不过什么东西都是熟能生巧, 多用用就好了。

说说程序本身吧。完全遵循 ANSI C, 可以在 Linux GCC 编译器上编译通过正常运行, 也可以在 Windows 平台上使用 MinGW GCC 编译运行。后来是尝试了 Visual Studio 感觉很好用, 就移植到 Microsoft Visual C++ 2013 平台上, 经过修改终于可以完美运行。于是这个程序就是跨平台的了, 在 Windows 上设置好 vcvars32 环境变量, 就可以直接 `nmake /f Makefile.vs` 来自动编译并连接生成 EXE 可执行文件, 当然用 VS 建立工程后把文件导入也可以正常编译的; 在 Linux

上也是直接 `make` 就可以编译连接生成 ELF 可执行文件。当然，有些平台相关特性，比如清屏、暂停、输入掩码，我都用预编译命令进行了处理，以兼容不同操作系统。

本程序是开源的，以 BSD 许可证发布，是个挺宽容的许可证。以前挺疯狂追崇 GPL 的，不过现在也不那么狂热了，宽容点好。SList 类不是我写的，是 LGPL 协议。代码托管在 Github 上，网址是 <https://github.com/kingfree/haut/tree/master/clang/exam/exam>。

好像说了很多技术上的问题，该说说这次课程应该得到的收获吧。总的来说就是 C 语言的综合应用，目的是数量掌握 C 语言，掌握程序设计的基本方法，对软件工程产生初步的认识，为后来的学习打下基础。

这次程序开发，难度有点超乎想象了。说实话，用 C 语言写这么大的项目确实还是第一次，为了遵循相关开发标准也适应了一段时间。实际上整个开发时间是六个半天，代码有 2400 多行，源文件达到 17 个。连续几日以比较高负荷的状态来编写代码，确实是非常疲累。但实际上编码的过程也是很充实的，因为我本身就喜欢编程，所以倒也没什么厌恶情绪，其实也是乐在其中。

这次课程呢，作为一次锻炼的机会，可以投入很大的精力来专心做一件事，是平常的学习中所没有的。尤其是作为一个准程序员，这样的训练其实应该有更多。既苦逼又充实的生活，其实也是挺想要的。

再有，通过这次实践，我发现了更多不会的东西。我对面向对象和设计模式还是一知半解，计划通过学习 C# 和 Java 来深入学习，至于 C++ 我有点不太喜欢就暂时搁置。另外也要继续深入学习 Linux，包括 TCP/IP、UNIX 编程、套接字、进程间通信等。对于 Web 开发，要继续深入掌握 JavaScript，熟练运用 Python 和 Ruby 配置服务器和编写脚本。在思维方面，要读一读侯世德的《集异璧之大成》。基本上计划如此，不过还是会在途中增加很多内容的，毕竟知识实在太多，我会的实在太少。

学习是一个漫长而复杂的过程，我希望我能够坚持走下去。当然我这个人平常是比较懒的，每天刷刷微博，看看动画片，一天就过去了。所以还是要静下心来，好好看看书，写写程序，多思考、多运动。只有充实自己，才能够有能力月入一狗，成为菊苣。总之朝着目标努力吧。

A 代码清单

A.1 许可证

版权所有 (c) 2014 田劲锋

保留所有权利

这份授权条款，在使用者符合以下三条件的情形下，授予使用者使用及再散播本软件包装原始码及二进制可执行形式的权利，无论此包装是否经改作皆然：

- * 对于本软件源代码的再散播，必须保留上述的版权宣告、此三条件表列，以及下述的免责声明。
- * 对于本套件二进制可执行形式的再散播，必须连带以文件以及 / 或者其他附于散播包装中的媒介方式，重制上述之版权宣告、此三条件表列，以及下述的免责声明。
- * 未获事前取得书面许可，不得使用柏克莱加州大学或本软件贡献者之名称，来为本软件之衍生物做任何表示支持、认可或推广、促销之行为。

免责声明：本软件是由加州大学董事会及本软件之贡献者以现状(as is)提供，本软件包装不负任何明示或默示之担保责任，包括但不限于就适售性以及特定目的的适用性为默示性担保。加州大学董事会及本软件之贡献者，无论任何条件、无论成因或任何责任主义、无论此责任为因合约关系、无过失责任主义或因非违约之侵权（包括过失或其他原因等）而起，对于任何因使用本软件包装所产生的任何直接性、间接性、偶发性、特殊性、惩罚性或任何结果的损害（包括但不限于替代商品或劳务之购用、使用损失、资料损失、利益损失、业务中断等等），不负任何责任，即在该种使用已获事前告知可能会造成此类损害的情形下亦然。

A.2 主文件

```
1  /* exam.c
2  * 入口主程序
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  #include "ui.h"
10
11 int main(int argc, char *argv[])
12 {
13     if (argc > 1) {
14         // 调试模式
15     }
16     ui_index();
17     return 0;
18 }
```

A.3 UI 类

```
1  /* ui.h
2  * 终端用户屏幕交互
3  */
4
5  #ifndef _UI_H_
6  #define _UI_H_
7
8  #include <stdbool.h>
9
10 #include "slist.h"
11 #include "problem.h"
12 #include "paper.h"
13 #include "user.h"
14 #include "score.h"
15
16 void cls();
17 void pause();
18 char *dif2star(int dif);
19
20 void ui_index();
21 int ui_do_login(User *u);
22 void ui_student_login();
23 void ui_teacher_login();
24
25 void ui_student(User *u);
26 void ui_teacher();
```

```

27 void ui_help();
28 void ui_about();
29
30 void ui_student_test(User *u);
31 void ui_student_score(User *u);
32
33 void ui_teacher_view();
34 void ui_teacher_insert();
35 void ui_teacher_delete();
36 void ui_teacher_update();
37 void ui_teacher_select();
38 void ui_teacher_generate();
39 void ui_teacher_score();
40
41 int ui_input_number();
42 char ui_input_ans();
43
44 Problem *ui_select_id(List *list);
45 int ui_select_des(List *list);
46 int ui_select_opt(List *list);
47 int ui_select_dif(List *list);
48 int ui_select_tag(List *list);
49 int ui_select_sec(List *list);
50 int ui_select_mul(List *list);
51 int ui_select_output(List *list, SListCallback *find, void *matchdata);
52 void ui_output_count(List *list);
53
54 void ui_generate_random(List *list);
55 void ui_generate_tags(List *list);
56 void ui_generate_secs(List *list);
57 void ui_generate_dif(List *list);
58
59 Problem *ui_input_problem();
60 void ui_edit_problem(Problem *p);
61 void ui_output_problem(Problem *p, bool show_more);
62 void *ui_each_problem_show(SList *item, void *userdata);
63
64 void ui_paper_list();
65 void ui_paper_view(List *proplist);
66 int ui_paper_save(Paper *pa);
67 void *ui_each_paper_show(SList *item, void *userdata);
68
69 void *ui_do_problem(SList *item, void *userdata);
70 void ui_output_score(Score *s);
71 void *ui_each_score_show(SList *item, void *userdata);
72
73 #endif

```

```

1  /* ui.c
2   * 终端用户屏幕交互
3   */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdbool.h>
9  #include <ctype.h>
10 #include <time.h>
11 #include <assert.h>
12 #include <errno.h>
13
14 #include "addon.h"
15 #include "slist.h"
16 #include "problem.h"
17 #include "paper.h"
18 #include "user.h"
19 #include "score.h"
20 #include "ui.h"
21
22 static char *NAME = "标准化考试系统";
23 static char *VERSION = "0.4.4";
24
25 void cls()
26 {
27 #ifdef WIN32
28     system("cls");
29 #else
30     system("clear");
31 #endif
32 }
33
34 void pause()
35 {
36 #ifdef WIN32
37     system("pause");
38 #else
39     system("read -p 请按任意键继续\". . .\");
40 #endif

```

```

41 }
42
43 char *dif2star(int dif)
44 {
45     assert(0 <= dif && dif <= 10);
46     char *s = calloc(16, sizeof(char));
47     assert(s);
48     char c[8][4] = { "" };
49     int i;
50     for (i = 0; i < dif / 2; i++)
51         strcpy(c[i], " ");
52     if ((i = dif % 2) == 1)
53         strcpy(c[dif / 2 + 1], " ");
54     sprintf(s, "%s%s%s%s", c[0], c[1], c[2], c[3], c[4]);
55     return s;
56 }
57
58 int ui_input_option(char *menu, bool cl)
59 {
60     if (cl) {
61         cls();
62         printf("欢迎使用 %s v%s\n", NAME, VERSION);
63     }
64     printf("%s$ ", menu);
65     int x;
66     scanf("%i", &x);
67     getchar();
68     return x;
69 }
70
71 void ui_index()
72 {
73     while (true) {
74         switch (ui_input_option(
75             " 1 - 学生登录\n"
76             " 2 - 教师登录\n"
77             " 3 - 帮助\n"
78             " 4 - 关于\n"
79             " 0 - 退出系统\n",
80             true)) {
81             case 1: ui_student_login(); break;
82             case 2: ui_teacher_login(); break;
83             case 3: ui_help(); break;
84             case 4: ui_about(); break;
85             case 0: exit(0); break;
86             default: return; break;
87         }
88     }
89 }
90
91 void ui_help()
92 {
93     #ifdef WIN32
94         system("..\\exam.pdf");
95     #else
96         printf("请查阅课程报告文档 exam.pdf \n");
97         pause();
98     #endif
99 }
100
101 void ui_about()
102 {
103     cls();
104     printf("%s v%s \n"
105         "开源的标准化考试系统 (命令行版) \n"
106         "\n"
107         "版权所有 (c) 2014 田劲锋保留所有权利 \n"
108         "遵循 MIT 许可协议, 允许有限的自由使用 \n"
109         "\n", NAME, VERSION);
110     pause();
111 }
112
113 void ui_student_login()
114 {
115     User *u = user_new();
116     int res = 0;
117     if ((res = ui_do_login(u)) > 0) {
118         ui_student(u);
119     } else {
120         perror("以学生身份登录失败");
121         pause();
122         if (res != -2) {
123             printf("自动建立新账号 %s...\n", u->username);
124             if (user_reg(u) < 0) {
125                 perror("建立新用户失败");
126                 goto end;
127             }
128             printf("以建立的新用户 %s 登录.\n", u->username);
129             pause();

```

```

130         ui_student(u);
131     }
132 }
133 end:
134     user_free(u);
135 }
136
137 void ui_student(User *u)
138 {
139     while (true) {
140         switch (ui_input_option(
141             "    1 - 做卷子\n"
142             "    2 - 看成绩\n"
143             "    9 - 返回上一级\n"
144             "    0 - 退出系统\n"
145             , true)) {
146             case 1: ui_student_test(u); break;
147             case 2: ui_student_score(u); break;
148             case 0: exit(0); break;
149             default: return; break;
150         }
151     }
152 }
153
154 void ui_student_test(User *u)
155 {
156     List *list = list_new();
157     if (paper_read_list(list) < 0) {
158         perror("读取试卷数据库失败");
159         return;
160     }
161     list_each_call(list, ui_each_paper_show, list);
162     int id = -1;
163     printf("试卷编号:\n$ ");
164     id = ui_input_number();
165     Paper *p = (Paper *)list_find(list, by_id, &id);
166     if (p == NULL) {
167         perror("没有这套试卷");
168         return;
169     }
170     List *problist = list_new();
171     if (problem_read_file(problist) < 0) {
172         perror("读取题目数据库失败");
173         return;
174     }
175     cls();
176     Score *s = score_new(u, p);
177     paper_problem_call(p, problist, ui_do_problem, s);
178     list_free(list);
179     list_free(problist);
180
181     List *scorelist = list_new();
182     if (score_read_list(scorelist) < 0) {
183         perror("读取成绩数据库失败");
184         goto end;
185     }
186     s->id = scorelist->max_id + 1;
187     if (list_insert(scorelist, s) < 0) {
188         perror("插入成绩失败");
189         goto end;
190     }
191     if (score_write_file(scorelist) < 0) {
192         perror("写入成绩数据库失败");
193         goto end;
194     }
195 end:
196     printf("考试完成! \n");
197     ui_output_score(s);
198     //list_free(scorelist);
199     pause();
200 }
201
202 void ui_student_score(User *u)
203 {
204     List *scorelist = list_new();
205     if (score_read_list(scorelist) < 0) {
206         perror("读取成绩数据库失败");
207         goto end;
208     }
209     list_find_each_call(scorelist, by_user_id, &u->id, ui_each_score_show, NULL);
210 end:
211     list_free(scorelist);
212     pause();
213 }
214
215 int ui_do_login(User *u)
216 {
217     printf("用户名: ");
218     scanf("%s", u->username);

```

```

219     char *s = getpass("密码: ");
220     strncpy(u->passwd, s, 64);
221     return user_login(u);
222 }
223
224 void ui_teacher_login()
225 {
226     User *u = user_new();
227     if (ui_do_login(u) > 0 && u->teacher == true) {
228         user_free(u);
229         ui_teacher();
230     } else {
231         user_free(u);
232         perror("以教师身份登录失败");
233         pause();
234     }
235 }
236
237 void ui_teacher()
238 {
239     while (true) {
240         switch (ui_input_option(
241             " 1 - 浏览试题\n"
242             " 2 - 添加试题 (增) \n"
243             " 3 - 删除试题 (删) \n"
244             " 4 - 修改试题 (改) \n"
245             " 5 - 查询试题 (查) \n"
246             " 6 - 智能组卷\n"
247             " 7 - 成绩分析\n"
248             " 9 - 返回上一级\n"
249             " 0 - 退出系统\n"
250             , true)) {
251             case 1: ui_teacher_view(); break;
252             case 2: ui_teacher_insert(); break;
253             case 3: ui_teacher_delete(); break;
254             case 4: ui_teacher_update(); break;
255             case 5: ui_teacher_select(); break;
256             case 6: ui_teacher_generate(); break;
257             case 7: ui_teacher_score(); break;
258             case 0: exit(0); break;
259             default: return; break;
260         }
261     }
262 }
263
264 void ui_teacher_view()
265 {
266     List *list = list_new();
267     if (problem_read_file(list) < 0) {
268         perror("读取题目数据库失败");
269         return;
270     }
271     ui_output_count(list);
272     list_each_call(list, ui_each_problem_show, NULL);
273     list_free(list);
274     pause();
275 }
276
277 void ui_teacher_insert()
278 {
279     List *list = list_new();
280     if (problem_read_file(list) < 0) {
281         perror("读取题目数据库失败");
282         goto end;
283     }
284     ui_output_count(list);
285     Problem *p = ui_input_problem();
286     p->id = list->max_id + 1;
287     int id = p->id;
288     ui_output_problem(p, true);
289     if (list_insert(list, p) < 0) {
290         perror("插入题目失败");
291         goto end;
292     }
293     if (problem_write_file(list) < 0) {
294         perror("写入题目数据库失败");
295         goto end;
296     }
297     printf("插入题目 %d 成功! \n", id);
298 end:
299     list_free(list);
300     pause();
301 }
302
303 bool ui_get_yes(char *s)
304 {
305     printf("%s输入s'Y或'y以确认':\n$ ", s);
306     gotn();
307     char y = getchar();

```

```

308     if (y == 'Y' || y == 'y') {
309         return true;
310     }
311     return false;
312 }
313
314 void ui_teacher_delete()
315 {
316     List *list = list_new();
317     int n;
318     if ((n = problem_read_file(list)) < 0) {
319         perror("读取题目数据库失败");
320         goto end;
321     }
322     ui_output_count(list);
323     Problem *p = ui_select_id(list);
324     if (p == NULL || p->id < 0) {
325         perror("获取题目失败");
326         goto end;
327     }
328     int id = p->id;
329     if (!ui_get_yes("你确定要删除该题吗? ")) {
330         goto end;
331     }
332     if (list_remove(list, by_id, &id) != p) {
333         perror("删除题目失败");
334         goto end;
335     }
336     free(p);
337     if (problem_write_file(list) < 0) {
338         perror("写入题目数据库失败");
339         goto end;
340     }
341     printf("删除题目 %d 成功!  \n", id);
342 end:
343     list_free(list);
344     pause();
345 }
346
347 void ui_teacher_update()
348 {
349     List *list = list_new();
350     int n;
351     if ((n = problem_read_file(list)) < 0) {
352         perror("读取题目数据库失败");
353         goto end;
354     }
355     ui_output_count(list);
356     Problem *p = ui_select_id(list);
357     if (p == NULL || p->id < 0) {
358         perror("获取题目失败");
359         goto end;
360     }
361     int id = p->id;
362     ui_edit_problem(p);
363     ui_output_problem(p, true);
364     if (problem_write_file(list) < 0) {
365         perror("写入题目数据库失败");
366         goto end;
367     }
368     printf("修改题目 %d 成功!  \n", id);
369 end:
370     list_free(list);
371     pause();
372 }
373
374 void ui_teacher_select()
375 {
376     List *list = list_new();
377     int n;
378     if ((n = problem_read_file(list)) < 0) {
379         perror("读取题目数据库失败");
380         return;
381     }
382     ui_output_count(list);
383     while (true) {
384         cls();
385         switch (ui_input_option(
386             "可以查询的关键词:\n"
387             " 1 - 题目编号\n"
388             " 2 - 题目描述\n"
389             " 3 - 题目选项\n"
390             " 4 - 题目难度\n"
391             " 5 - 题目标签\n"
392             " 6 - 题目章节\n"
393             " 7 - 模糊查询\n"
394             " 9 - 返回上一级\n"
395             " 0 - 退出系统\n"
396             , false)) {

```



```

397         case 1: ui_select_id(list); break;
398         case 2: ui_select_des(list); break;
399         case 3: ui_select_opt(list); break;
400         case 4: ui_select_dif(list); break;
401         case 5: ui_select_tag(list); break;
402         case 6: ui_select_sec(list); break;
403         case 7: ui_select_mul(list); break;
404         case 0: exit(0); break;
405         default: return; break;
406     }
407     pause();
408 }
409 list_free(list);
410 }
411
412 void ui_teacher_generate()
413 {
414     List *list = list_new();
415     if (problem_read_file(list) < 0) {
416         perror("读取题目数据库失败");
417         return;
418     }
419     while (true) {
420         cls();
421         switch (ui_input_option(
422             "选择智能组卷算法:\n"
423             " 1 - 随机生成\n"
424             " 2 - 按标签生成\n"
425             " 3 - 按章节生成\n"
426             " 4 - 按难度生成\n"
427             " 5 - 试卷列表\n"
428             " 6 - 查看试卷\n"
429             " 9 - 返回上一级\n"
430             " 0 - 退出系统\n"
431             , false)) {
432             case 1: ui_generate_random(list); break;
433             case 2: ui_generate_tags(list); break;
434             case 3: ui_generate_secs(list); break;
435             case 4: ui_generate_dif(list); break;
436             case 5: ui_paper_list(); break;
437             case 6: ui_paper_view(list); break;
438             case 0: exit(0); break;
439             default: return; break;
440         }
441         pause();
442     }
443     list_free(list);
444 }
445
446 void ui_generate_random(List *list)
447 {
448     ui_output_count(list);
449     printf("智能组卷 - 随机生成\n");
450     printf("生成题目数:\n$ ");
451     int n = ui_input_number();
452     Paper *pa = paper_new();
453     printf("试卷标题:\n$ ");
454     scanf("%s", pa->title);
455     paper_generate_random(pa, list, n);
456     ui_paper_save(pa);
457 }
458
459 void ui_generate_tags(List *list)
460 {
461     ui_output_count(list);
462     printf("智能组卷 - 按知识点标签生成\n");
463     int tags[64], i = 0;
464     printf("题目标签编号 (以 0 结尾):\n$ ");
465     for (; scanf("%d", &tags[i]), tags[i] > 0; i++);
466     printf("生成题目数:\n$ ");
467     int n = ui_input_number();
468     Paper *pa = paper_new();
469     printf("试卷标题:\n$ ");
470     scanf("%s", pa->title);
471     paper_generate_tags(pa, list, n, tags, i);
472     ui_paper_save(pa);
473 }
474
475 void ui_generate_secs(List *list)
476 {
477     ui_output_count(list);
478     printf("智能组卷 - 按知识点章节生成\n");
479     double secs[64];
480     int i = 0;
481     printf("题目章节编号 (小数点分割章节, 节可省略, 以 0 结尾):\n$ ");
482     for (; scanf("%lf", &secs[i]), secs[i] > 0; i++);
483     printf("生成题目数:\n$ ");
484     int n = ui_input_number();
485     Paper *pa = paper_new();

```

```

486     printf("试卷标题:\n$ ");
487     scanf("%s", pa->title);
488     paper_generate_secs(pa, list, n, secs, i);
489     ui_paper_save(pa);
490 }
491
492 void ui_generate_dif(List *list)
493 {
494     ui_output_count(list);
495     printf("智能组卷 - 按难度区间生成\n");
496     int a, b;
497     printf("题目难度区间 ( 两个 0--10 的数字 ):\n$ ");
498     a = ui_input_number();
499     b = ui_input_number();
500     printf("生成题目数:\n$ ");
501     int n = ui_input_number();
502     Paper *pa = paper_new();
503     printf("试卷标题:\n$ ");
504     scanf("%s", pa->title);
505     paper_generate_dif(pa, list, n, a, b);
506     ui_paper_save(pa);
507 }
508
509 void ui_teacher_score()
510 {
511     List *scorelist = list_new();
512     if (score_read_list(scorelist) < 0) {
513         perror("读取成绩数据库失败");
514         goto end;
515     }
516     list_each_call(scorelist, ui_each_score_show, NULL);
517 end:
518     list_free(scorelist);
519     pause();
520 }
521
522 void *ui_each_problem_show(SList *item, void *userdata)
523 {
524     Problem *p = (Problem *)item->userdata;
525     // fprintf(stderr, "(%p)->p\n", item, p);
526     printf("%d. %s\n",
527           " [A]. %s\t[B]. %s\t[C]. %s\t[D]. %s\n",
528           "", p->id, p->des,
529           p->opt[0], p->opt[1], p->opt[2], p->opt[3]);
530     printf("\t 答案: %c\t 难度: %-10s\t 标签: %i\t 章节: %i.%i\n",
531           p->ans, dif2star(p->dif), p->tag, p->chapter, p->section);
532     return NULL;
533 }
534
535 void *ui_each_score_show(SList *item, void *userdata)
536 {
537     Score *s = (Score *)item->userdata;
538     printf("%d. ( 试卷 %d ) \n",
539           "考生: %s\n",
540           "成绩: %d 分 \n",
541           "时间: %s",
542           "\n",
543           s->id, s->paper_id,
544           s->username,
545           s->right * 100 / s->paper_count,
546           ctime(&s->date)
547           );
548     return NULL;
549 }
550
551 void *ui_do_problem(SList *item, void *userdata)
552 {
553     Problem *p = (Problem *)item->userdata;
554     printf("%s\n",
555           " [A]. %s\n",
556           " [B]. %s\n",
557           " [C]. %s\n",
558           " [D]. %s\n",
559           "选择: ", p->des,
560           p->opt[0], p->opt[1], p->opt[2], p->opt[3]);
561     char c = ui_input_ans();
562     Score *s = (Score *)userdata;
563     score_did(s, c, p->ans);
564     return NULL;
565 }
566
567 void ui_output_score(Score *s)
568 {
569     printf("%s",
570           "考生: %s\n",
571           "成绩: %d 分 \n",
572           "\n",
573           ctime(&s->date),
574           s->username,

```

```

575         s->right * 100 / s->paper_count);
576     }
577
578     void *ui_each_paper_show(SList *item, void *userdata)
579     {
580         Paper *pa = (Paper *)item->userdata;
581         if (userdata == NULL) {
582             fprintf_paper_pid(pa, stdout);
583         } else {
584             printf("%d. %s (共 %d 题)\n", pa->id, pa->title, pa->length);
585         }
586         return NULL;
587     }
588
589     char ui_input_ans()
590     {
591         char c = ' ';
592         while (scanf("%c", &c), c = toupper(c), !('A' <= c && c <= 'D'));
593         return c;
594     }
595
596     int ui_input_number()
597     {
598         int n = 0;
599         char c = ' ';
600         while (c = getchar(), !('0' <= c && c <= '9'));
601         ungetc(c, stdin);
602         // while (scanf("%s", s), !('0' <= s[0] && s[0] <= '9'));
603         scanf("%d", &n);
604         return n;
605     }
606
607     void ui_output_count(List *list)
608     {
609         printf("数据库中有 %d 条记录, 最大编号为 %d\n", list->count, list->max_id);
610     }
611
612     Problem *ui_input_problem()
613     {
614         int i = 0;
615         Problem *p = problem_new();
616
617         printf("请输入题目相关信息: \n");
618
619         printf("题目描述:\n$ ");
620         read_string(p->des, 256);
621
622         for (i = 'A'; i <= 'D'; i++) {
623             printf("选项[ %c]:\n$ ", i);
624             read_string(p->opt[i - 'A'], 64);
625         }
626
627         printf("正确答案字母序号:\n$ ");
628         p->ans = ui_input_ans();
629
630         printf("题目难度(1--10):\n$ ");
631         p->dif = ui_input_number();
632         if (p->dif < 0) p->dif = 0;
633         if (p->dif > 10) p->dif = 10;
634
635         printf("知识点标签数字编号():\n$ ");
636         p->tag = ui_input_number();
637
638         printf("知识点章节章节():\n$ ");
639         p->chapter = ui_input_number();
640         p->section = ui_input_number();
641
642         return p;
643     }
644
645     void ui_edit_problem(Problem *p)
646     {
647         printf("修改题目信息 (直接回车表示不修改): \n");
648         gotn();
649
650         printf("题目描述: %s\n$ ", p->des);
651         if (!gotn()) {
652             read_string(p->des, 256);
653         }
654
655         int i = 0;
656         for (i = 'A'; i <= 'D'; i++) {
657             printf("选项[ %c]: %s\n$ ", i, p->opt[i - 'A']);
658             if (!gotn()) {
659                 read_string(p->opt[i - 'A'], 64);
660             }
661         }
662
663         printf("正确答案: %c\n$ ", p->ans);

```

```

664     if (!gotn()) {
665         p->ans = ui_input_ans();
666         gotn();
667     }
668
669     printf("题目难度: (%hi) %s\n", p->dif, dif2star(p->dif));
670     if (!gotn()) {
671         p->dif = ui_input_number();
672         if (p->dif < 0) p->dif = 0;
673         if (p->dif > 10) p->dif = 10;
674         gotn();
675     }
676
677     printf("知识点标签: %hi\n", p->tag);
678     if (!gotn()) {
679         p->tag = ui_input_number();
680         gotn();
681     }
682
683     printf("知识点章节: %hi.%hi\n", p->chapter, p->section);
684     if (!gotn()) {
685         p->chapter = ui_input_number();
686         p->section = ui_input_number();
687         gotn();
688     }
689 }
690
691 void ui_output_problem(Problem *p, bool show_more)
692 {
693     printf("%d. %s\n"
694           "    [A]. %s\n"
695           "    [B]. %s\n"
696           "    [C]. %s\n"
697           "    [D]. %s\n"
698           "%s", p->id, p->des,
699           p->opt[0], p->opt[1], p->opt[2], p->opt[3]);
700     if (show_more) {
701         // printf("%p\n", p);
702         printf("答案: %c\t", p->ans);
703         printf("难度: %s\t", dif2star(p->dif));
704         printf("标签: %i\t", p->tag);
705         printf("章节: %i.%i\n", p->chapter, p->section);
706     }
707 }
708
709 Problem *ui_select_id(List *list)
710 {
711     int id = -1;
712     printf("题目编号:\n");
713     while (scanf("%d", &id) != 1);
714
715     Problem *p = (Problem *)list_find(list, by_id, &id);
716     if (p == NULL) {
717         return NULL;
718     }
719     ui_output_problem(p, true);
720     return p;
721 }
722
723 int ui_select_output(List *list, SListCallback *find, void *data)
724 {
725     int n = list_find_each_call(list, find, data, ui_each_problem_show, NULL);
726     printf("共查询到 %d 个符合条件的结果.\n", n);
727     return n;
728 }
729
730 int ui_select_des(List *list)
731 {
732     char des[256] = "";
733     printf("题目描述 (只需输入部分内容):\n");
734     while (scanf("%s", des) != 1);
735     printf("查找题目描述中含有 \"%s\" 的题目...\n", des);
736     return ui_select_output(list, by_des, des);
737 }
738
739 int ui_select_opt(List *list)
740 {
741     char opt[64] = "";
742     printf("题目选项 (只需输入部分内容):\n");
743     while (scanf("%s", opt) != 1);
744     printf("查找题目选项中含有 \"%s\" 的题目...\n", opt);
745     return ui_select_output(list, by_opt, opt);
746 }
747
748 int ui_select_dif(List *list)
749 {
750     sel_num t;
751     printf("题目难度 (输入 < > == <= >= != <> 后空格数字):\n");
752     while (scanf("%s%d", t.mark, &t.num) != 2);

```

```

753     printf("查找题目难度 %s %d 的题目...\n", t.mark, t.num);
754     return ui_select_output(list, by_difs, &t);
755 }
756
757 int ui_select_tag(List *list)
758 {
759     short tags[64], i = 0;
760     printf("题目标签编号 (以 0 结尾):\n$ ");
761     for (; scanf("%hi", &tags[i]), tags[i] > 0; i++);
762     printf("查找题目标签为");
763     for (i = 0; tags[i] > 0; i++) printf(" %d", tags[i]);
764     printf(" 的题目...\n");
765     return ui_select_output(list, by_tags, tags);
766 }
767
768 int ui_select_sec(List *list)
769 {
770     double secs[64];
771     int i = 0;
772     printf("题目章节编号 (小数点分割章节, 节可省略, 以 0 结尾) :\n$ ");
773     for (; scanf("%lf", &secs[i]), secs[i] > 0; i++);
774     printf("查找题目章节为");
775     for (i = 0; secs[i] > 0; i++) printf(" %.2lf", secs[i]);
776     printf(" 的题目...\n");
777     return ui_select_output(list, by_secs, secs);
778 }
779
780 int ui_select_mul(List *list)
781 {
782     char key[64] = "";
783     printf("输入要查询的关键词:\n$ ");
784     while (scanf("%s", key) != 1);
785     printf("查找题目中含有 \"%s\" 的题目...\n", key);
786     return ui_select_output(list, by_mul, key);
787 }
788
789 void ui_paper_list()
790 {
791     List *list = list_new();
792     if (paper_read_list(list) < 0) {
793         perror("读取试卷数据库失败");
794         return;
795     }
796     ui_output_count(list);
797     list_each_call(list, ui_each_paper_show, NULL);
798     list_free(list);
799 }
800
801 void ui_paper_view(List *problist)
802 {
803     List *list = list_new();
804     if (paper_read_list(list) < 0) {
805         perror("读取试卷数据库失败");
806         return;
807     }
808     ui_output_count(list);
809     int id = -1;
810     printf("试卷编号:\n$ ");
811     while (scanf("%d", &id) != 1);
812     Paper *p = (Paper *)list_find(list, by_id, &id);
813     if (p == NULL) {
814         perror("没有这套试卷");
815         return;
816     }
817     cls();
818     paper_problem_call(p, problist, ui_each_problem_show, NULL);
819     list_free(list);
820 }
821
822 int ui_paper_save(Paper *pa)
823 {
824     if (pa->length <= 0) {
825         perror("试卷中没有题目");
826         return -1;
827     }
828     fprintf_paper_pid(pa, stdout);
829     if (!ui_get_yes("是否保存试卷? \n$ ")) {
830         return 0;
831     }
832     List *list = list_new();
833     if (paper_read_list(list) < 0) {
834         perror("读取试卷数据库失败");
835         goto end;
836     }
837     ui_output_count(list);
838     pa->id = list->max_id + 1;
839     int id = pa->id;
840     if (list_insert(list, pa) < 0) {
841         perror("保存试卷失败");

```

```

842         goto end;
843     }
844     if (paper_write_list(list) < 0) {
845         perror("写出试卷数据库失败");
846         goto end;
847     }
848     printf("保存试卷 %d 成功!  \n", id);
849 end:
850     list_free(list);
851     return 0;
852 }

```

A.4 List 类

```

1  /* list.h
2  * 用于操作数据库的链表
3  */
4
5  #ifndef _FILE_H_
6  #define _FILE_H_
7
8  #include <stdio.h>
9
10 #include "slist.h"
11
12 typedef struct Block {
13     FILE *file;
14     size_t size;
15     int max_id;
16     int count;
17 } Block;
18
19 typedef struct List {
20     SList *slist;
21     int max_id;
22     int count;
23 } List;
24
25 typedef struct ID {
26     int id;
27 } ID;
28
29 int cmp_id(const SList *item1, const SList *item2, void *data);
30
31 int file_data_count(FILE *file, size_t size);
32
33 List *list_new();
34 void list_free(List *list);
35
36 int read_file_to_list(char *filename, List *list, size_t size);
37 int write_list_to_file(char *filename, List *list, size_t size);
38 void *write_userdata(SList *item, void *data);
39
40 void *list_each_call(List *list, SListCallback *call, void *data);
41 int list_insert(List *list, void *p);
42 void *list_remove(List *list, SListCallback *find, void *data);
43 void *list_find(List *list, SListCallback *find, void *data);
44 int list_find_each_call(List *list, SListCallback *find, void *data, SListCallback *call, void *userdata);
45
46 #endif

```

```

1  /* list.c
2  * 用于操作数据库的链表
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdarg.h>
9  #include <assert.h>
10
11 #include "list.h"
12
13 void list_restore(List *list);
14
15 int file_data_count(FILE *file, size_t size)
16 {
17     fseek(file, 0, SEEK_END);
18     int n = ftell(file) / size;
19     rewind(file);
20     return n;
21 }
22

```

```

23 void *write_userdata(SList *item, void *data)
24 {
25     Block *b = (Block *)data;
26     if (fwrite(item->userdata, b->size, 1, b->file) == 1) {
27         b->count++;
28     }
29     return NULL;
30 }
31
32 int read_file_to_list(char *filename, List *list, size_t size)
33 /* 返回成功读取的条目数 */
34 {
35     FILE *file = fopen(filename, "rb");
36     if (file == NULL) {
37         if ((file = fopen(filename, "wb")) == NULL) {
38             return -1;
39         }
40         return 0;
41     }
42     int n = file_data_count(file, size);
43     rewind(file);
44     int i = 0;
45     void *p = NULL;
46     for (i = 0; i < n; i++) {
47         if ((p = malloc(size)) == NULL) {
48             goto end;
49         }
50         memset(p, 0, size);
51         if (fread(p, size, 1, file) != 1) {
52             goto end;
53         }
54         if (list_insert(list, p) < 0) {
55             goto end;
56         }
57     }
58 end:
59     fclose(file);
60     list_restore(list);
61     return list->count;
62 }
63
64 int list_insert(List *list, void *p)
65 {
66     list->slist = slist_cons(slist_box(p), list->slist);
67     list->count++;
68     list->max_id++;
69     return 0;
70 }
71
72 int cmp_id(const SList *item1, const SList *item2, void *data)
73 {
74     ID *a = (ID *)item1->userdata;
75     ID *b = (ID *)item2->userdata;
76     return (a->id - b->id);
77 }
78
79 void list_restore(List *list)
80 {
81     list->slist = slist_sort(list->slist, cmp_id, NULL);
82     list->count = slist_length(list->slist);
83     if (list->count <= 0) {
84         list->max_id = 0;
85     } else {
86         SList *s = slist_nth(list->slist, list->count);
87         ID *p = (ID *)s->userdata;
88         list->max_id = p->id;
89         if (list->count > list->max_id) {
90             list->count = list->max_id;
91         }
92     }
93 }
94
95 int write_list_to_file(char *filename, List *list, size_t size)
96 {
97     FILE *file = fopen(filename, "wb");
98     if (file == NULL) {
99         return -1;
100     }
101     Block b = { .file = file, .size = size, .count = 0, .max_id = 0 };
102     rewind(file);
103     slist_foreach(list->slist, write_userdata, &b);
104     fclose(file);
105     return b.count;
106 }
107
108 List *list_new()
109 {
110     List *list = (List *)malloc(sizeof(List));
111     assert(list);

```

```

112     list->slist = NULL;
113     list->max_id = 0;
114     list->count = 0;
115     return list;
116 }
117
118 void userdata_free(void *item)
119 {
120     free(slist_unbox(item));
121 }
122
123 void list_free(List *list)
124 {
125     list->slist = slist_delete(list->slist, userdata_free);
126     free(list);
127 }
128
129 void *list_find(List *list, SListCallback *find, void *data)
130 {
131     SList *s = (SList *)slist_find(list->slist, find, data);
132     if (s == NULL) {
133         return NULL;
134     }
135     void *p = (void *)s->userdata;
136     return p;
137 }
138
139 int list_find_each_call(List *list, SListCallback *find, void *data, SListCallback *call, void *userdata)
140 {
141     int n = 0;
142     SList *s = list->slist;
143     while ((s = (SList *)slist_find(s, find, data)) != NULL) {
144         n++;
145         call(s, userdata);
146         s = slist_tail(s);
147     }
148     return n;
149 }
150
151 void *list_each_call(List *list, SListCallback *call, void *data)
152 {
153     return slist_foreach(list->slist, call, data);
154 }
155
156 void *list_remove(List *list, SListCallback *find, void *data)
157 {
158     return slist_unbox(slist_remove(&list->slist, find, data));
159 }

```

A.5 Slist 类

```

1  /* slist.h -- generalised singly linked lists
2
3  Copyright (C) 2000, 2004, 2009 Free Software Foundation, Inc.
4  Written by Gary V. Vaughan, 2000
5
6  NOTE: The canonical source of this file is maintained with the
7  GNU Libtool package. Report bugs to bug-libtool@gnu.org.
8
9  GNU Libltdl is free software; you can redistribute it and/or
10 modify it under the terms of the GNU Lesser General Public
11 License as published by the Free Software Foundation; either
12 version 2 of the License, or (at your option) any later version.
13
14 As a special exception to the GNU Lesser General Public License,
15 if you distribute this file as part of a program or library that
16 is built using GNU Libtool, you may include this file under the
17 same distribution terms that you use for the rest of that program.
18
19 GNU Libltdl is distributed in the hope that it will be useful,
20 but WITHOUT ANY WARRANTY; without even the implied warranty of
21 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 GNU Lesser General Public License for more details.
23
24 You should have received a copy of the GNU Lesser General Public
25 License along with GNU Libltdl; see the file COPYING.LIB. If not, a
26 copy can be downloaded from http://www.gnu.org/licenses/lgpl.html,
27 or obtained by writing to the Free Software Foundation, Inc.,
28 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
29 */
30
31 /* A generalised list. This is deliberately transparent so that you
32    can make the NEXT field of all your chained data structures first,
33    and then cast them to `(SList *)' so that they can be manipulated

```



```

34 | by this API.
35 |
36 | Alternatively, you can generate raw SList elements using slist_new(),
37 | and put the element data in the USERDATA field. Either way you
38 | get to manage the memory involved by yourself.
39 | */
40 |
41 | #if !defined(SLIST_H)
42 | #define SLIST_H 1
43 |
44 | #if defined(LTDL)
45 | # include <libltdl/lt_glibc.h>
46 | # include <libltdl/lt_system.h>
47 | #else
48 | # define LT_SCOPE
49 | #endif
50 |
51 | #include <stddef.h>
52 |
53 | #if defined(__cplusplus)
54 | extern "C" {
55 | #endif
56 |
57 | typedef struct slist {
58 |     struct slist *next;      /* chain forward pointer*/
59 |     const void *userdata;    /* for boxed `SList' item */
60 | } SList;
61 |
62 | typedef void * SListCallback (SList *item, void *userdata);
63 | typedef int SListCompare (const SList *item1, const SList *item2,
64 |                          void *userdata);
65 |
66 | LT_SCOPE SList *slist_concat (SList *head, SList *tail);
67 | LT_SCOPE SList *slist_cons (SList *item, SList *slist);
68 |
69 | LT_SCOPE SList *slist_delete (SList *slist, void (*delete_fct) (void *item));
70 | LT_SCOPE SList *slist_remove (SList **phead, SListCallback *find,
71 |                             void *matchdata);
72 | LT_SCOPE SList *slist_reverse (SList *slist);
73 | LT_SCOPE SList *slist_sort (SList *slist, SListCompare *compare,
74 |                             void *userdata);
75 |
76 | LT_SCOPE SList *slist_tail (SList *slist);
77 | LT_SCOPE SList *slist_nth (SList *slist, size_t n);
78 | LT_SCOPE void * slist_find (SList *slist, SListCallback *find,
79 |                             void *matchdata);
80 | LT_SCOPE size_t slist_length (SList *slist);
81 |
82 | LT_SCOPE void * slist_foreach (SList *slist, SListCallback *foreach,
83 |                               void *userdata);
84 |
85 | LT_SCOPE SList *slist_box (const void *userdata);
86 | LT_SCOPE void * slist_unbox (SList *item);
87 |
88 | #if defined(__cplusplus)
89 | }
90 | #endif
91 |
92 | #if !defined(LTDL)
93 | # undef LT_SCOPE
94 | #endif
95 |
96 | #endif /*!defined(SLIST_H)*/

```

```

1 | /* slist.c -- generalised singly linked lists
2 |
3 | Copyright (C) 2000, 2004, 2007, 2008, 2009 Free Software Foundation, Inc.
4 | Written by Gary V. Vaughan, 2000
5 |
6 | NOTE: The canonical source of this file is maintained with the
7 | GNU Libtool package. Report bugs to bug-libtool@gnu.org.
8 |
9 | GNU Libltdl is free software; you can redistribute it and/or
10 | modify it under the terms of the GNU Lesser General Public
11 | License as published by the Free Software Foundation; either
12 | version 2 of the License, or (at your option) any later version.
13 |
14 | As a special exception to the GNU Lesser General Public License,
15 | if you distribute this file as part of a program or library that
16 | is built using GNU Libtool, you may include this file under the
17 | same distribution terms that you use for the rest of that program.
18 |
19 | GNU Libltdl is distributed in the hope that it will be useful,
20 | but WITHOUT ANY WARRANTY; without even the implied warranty of
21 | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 | GNU Lesser General Public License for more details.
23 |
24 | You should have received a copy of the GNU Lesser General Public

```

```

25 | License along with GNU Libltdl; see the file COPYING.LIB.  If not, a
26 | copy can be downloaded from http://www.gnu.org/licenses/lgpl.html,
27 | or obtained by writing to the Free Software Foundation, Inc.,
28 | 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
29 | */
30 |
31 | #include <assert.h>
32 |
33 | #include "slist.h"
34 | #include <stddef.h>
35 | #include <stdlib.h>
36 |
37 | static SList * slist_sort_merge (SList *left, SList *right,
38 |                                 SListCompare *compare, void *userdata);
39 |
40 |
41 | /* Call DELETE repeatedly on each element of HEAD.
42 |
43 |  CAVEAT: If you call this when HEAD is the start of a list of boxed
44 |          items, you must remember that each item passed back to your
45 |          DELETE function will be a boxed item that must be slist_unbox()ed
46 |          before operating on its contents.
47 |
48 |  e.g. void boxed_delete (void *item) { item_free (slist_unbox (item)); }
49 |  ...
50 |  slist = slist_delete (slist, boxed_delete);
51 |  ...
52 | */
53 | SList *
54 | slist_delete (SList *head, void (*delete_fct) (void *item))
55 | {
56 |     assert (delete_fct);
57 |
58 |     while (head)
59 |     {
60 |         SList *next = head->next;
61 |         (*delete_fct) (head);
62 |         head = next;
63 |     }
64 |
65 |     return 0;
66 | }
67 |
68 | /* Call FIND repeatedly with MATCHDATA and each item of *PHEAD, until
69 |    FIND returns non-NULL, or the list is exhausted.  If a match is found
70 |    the matching item is destructively removed from *PHEAD, and the value
71 |    returned by the matching call to FIND is returned.
72 |
73 |    CAVEAT: To avoid memory leaks, unless you already have the address of
74 |            the stale item, you should probably return that from FIND if
75 |            it makes a successful match.  Don't forget to slist_unbox()
76 |            every item in a boxed list before operating on its contents.  */
77 | SList *
78 | slist_remove (SList **phead, SListCallback *find, void *matchdata)
79 | {
80 |     SList *stale = 0;
81 |     void *result = 0;
82 |
83 |     assert (find);
84 |
85 |     if (!phead || !*phead)
86 |         return 0;
87 |
88 |     /* Does the head of the passed list match? */
89 |     result = (*find) (*phead, matchdata);
90 |     if (result)
91 |     {
92 |         stale = *phead;
93 |         *phead = stale->next;
94 |     }
95 |     /* what about the rest of the elements? */
96 |     else
97 |     {
98 |         SList *head;
99 |         for (head = *phead; head->next; head = head->next)
100 |         {
101 |             result = (*find) (head->next, matchdata);
102 |             if (result)
103 |             {
104 |                 stale = head->next;
105 |                 head->next = stale->next;
106 |                 break;
107 |             }
108 |         }
109 |     }
110 |
111 |     return (SList *) result;
112 | }
113 |

```

```

114 /* Call FIND repeatedly with each element of SLIST and MATCHDATA, until
115 FIND returns non-NULL, or the list is exhausted. If a match is found
116 the value returned by the matching call to FIND is returned. */
117 void *
118 slist_find (SList *slist, SListCallback *find, void *matchdata)
119 {
120     void *result = 0;
121
122     assert (find);
123
124     for (; slist; slist = slist->next)
125     {
126         result = (*find) (slist, matchdata);
127         if (result)
128             break;
129     }
130
131     return result;
132 }
133
134 /* Return a single list, composed by destructively concatenating the
135 items in HEAD and TAIL. The values of HEAD and TAIL are undefined
136 after calling this function.
137
138 CAVEAT: Don't mix boxed and unboxed items in a single list.
139
140 e.g. slist1 = slist_concat (slist1, slist2); */
141 SList *
142 slist_concat (SList *head, SList *tail)
143 {
144     SList *last;
145
146     if (!head)
147     {
148         return tail;
149     }
150
151     last = head;
152     while (last->next)
153         last = last->next;
154
155     last->next = tail;
156
157     return head;
158 }
159
160 /* Return a single list, composed by destructively appending all of
161 the items in SLIST to ITEM. The values of ITEM and SLIST are undefined
162 after calling this function.
163
164 CAVEAT: Don't mix boxed and unboxed items in a single list.
165
166 e.g. slist1 = slist_cons (slist_box (data), slist1); */
167 SList *
168 slist_cons (SList *item, SList *slist)
169 {
170     if (!item)
171     {
172         return slist;
173     }
174
175     assert (!item->next);
176
177     item->next = slist;
178     return item;
179 }
180
181 /* Return a list starting at the second item of SLIST. */
182 SList *
183 slist_tail (SList *slist)
184 {
185     return slist ? slist->next : NULL;
186 }
187
188 /* Return a list starting at the Nth item of SLIST. If SLIST is less
189 than N items long, NULL is returned. Just to be confusing, list items
190 are counted from 1, to get the 2nd element of slist:
191
192 e.g. shared_list = slist_nth (slist, 2); */
193 SList *
194 slist_nth (SList *slist, size_t n)
195 {
196     for (; n > 1 && slist; n--)
197         slist = slist->next;
198
199     return slist;
200 }
201
202 /* Return the number of items in SLIST. We start counting from 1, so

```

```

203     the length of a list with no items is 0, and so on.  */
204 size_t
205 slist_length (SList *slist)
206 {
207     size_t n;
208
209     for (n = 0; slist; ++n)
210         slist = slist->next;
211
212     return n;
213 }
214
215 /* Destructively reverse the order of items in SLIST.  The value of SLIST
216    is undefined after calling this function.
217
218    CAVEAT: You must store the result of this function, or you might not
219           be able to get all the items except the first one back again.
220
221    e.g.    slist = slist_reverse (slist);  */
222 SList *
223 slist_reverse (SList *slist)
224 {
225     SList *result = 0;
226     SList *next;
227
228     while (slist)
229     {
230         next = slist->next;
231         slist->next = result;
232         result = slist;
233         slist = next;
234     }
235
236     return result;
237 }
238
239 /* Call FOREACH once for each item in SLIST, passing both the item and
240    USERDATA on each call.  */
241 void *
242 slist_foreach (SList *slist, SListCallback *foreach, void *userdata)
243 {
244     void *result = 0;
245
246     assert (foreach);
247
248     while (slist)
249     {
250         SList *next = slist->next;
251         result = (*foreach) (slist, userdata);
252
253         if (result)
254             break;
255
256         slist = next;
257     }
258
259     return result;
260 }
261
262 /* Destructively merge the items of two ordered lists LEFT and RIGHT,
263    returning a single sorted list containing the items of both -- Part of
264    the quicksort algorithm.  The values of LEFT and RIGHT are undefined
265    after calling this function.
266
267    At each iteration, add another item to the merged list by taking the
268    lowest valued item from the head of either LEFT or RIGHT, determined
269    by passing those items and USERDATA to COMPARE.  COMPARE should return
270    less than 0 if the head of LEFT has the lower value, greater than 0 if
271    the head of RIGHT has the lower value, otherwise 0.  */
272 static SList *
273 slist_sort_merge (SList *left, SList *right, SListCompare *compare,
274                  void *userdata)
275 {
276     SList merged, *insert;
277
278     insert = &merged;
279
280     while (left && right)
281     {
282         if ((*compare) (left, right, userdata) <= 0)
283         {
284             insert = insert->next = left;
285             left = left->next;
286         }
287         else
288         {
289             insert = insert->next = right;
290             right = right->next;
291         }

```

```

292     }
293
294     insert->next = left ? left : right;
295
296     return merged.next;
297 }
298
299 /* Perform a destructive quicksort on the items in SLIST, by repeatedly
300    calling COMPARE with a pair of items from SLIST along with USERDATA
301    at every iteration. COMPARE is a function as defined above for
302    slist_sort_merge(). The value of SLIST is undefined after calling
303    this function.
304
305    e.g. slist = slist_sort (slist, compare, 0); */
306 SList *
307 slist_sort (SList *slist, SListCompare *compare, void *userdata)
308 {
309     SList *left, *right;
310
311     if (!slist)
312         return slist;
313
314     /* Be sure that LEFT and RIGHT never contain the same item. */
315     left = slist;
316     right = slist->next;
317
318     if (!right)
319         return left;
320
321     /* Skip two items with RIGHT and one with SLIST, until RIGHT falls off
322        the end. SLIST must be about half way along. */
323     while (right && (right = right->next))
324     {
325         if (!right || !(right = right->next))
326             break;
327         slist = slist->next;
328     }
329     right = slist->next;
330     slist->next = 0;
331
332     /* Sort LEFT and RIGHT, then merge the two. */
333     return slist_sort_merge (slist_sort (left, compare, userdata),
334                             slist_sort (right, compare, userdata),
335                             compare, userdata);
336 }
337
338
339 /* Aside from using the functions above to manage chained structures of
340    any type that has a NEXT pointer as its first field, SLISTS can
341    be comprised of boxed items. The boxes are chained together in
342    that case, so there is no need for a NEXT field in the item proper.
343    Some care must be taken to slist_box and slist_unbox each item in
344    a boxed list at the appropriate points to avoid leaking the memory
345    used for the boxes. It is usually a very bad idea to mix boxed and
346    non-boxed items in a single list. */
347
348 /* Return a 'boxed' freshly mallocated 1 element list containing
349    USERDATA. */
350 SList *
351 slist_box (const void *userdata)
352 {
353     SList *item = (SList *) malloc (sizeof *item);
354
355     if (item)
356     {
357         item->next = 0;
358         item->userdata = userdata;
359     }
360
361     return item;
362 }
363
364 /* Return the contents of a 'boxed' ITEM, recycling the box itself. */
365 void *
366 slist_unbox (SList *item)
367 {
368     void *userdata = 0;
369
370     if (item)
371     {
372         /* Strip the const, because responsibility for this memory
373            passes to the caller on return. */
374         userdata = (void *) item->userdata;
375         free (item);
376     }
377
378     return userdata;
379 }

```

A.6 Problem 类

```
1  /* problem.h
2  * 题目数据库的增删改查
3  */
4
5  #ifndef _PROBLEM_H_
6  #define _PROBLEM_H_
7
8  #include "slist.h"
9  #include "list.h"
10
11  typedef struct Problem {
12      int id;
13      char des[256]; // 题目描述
14      char opt[4][64]; // 选项
15      char ans; // 答案
16      short dif; // 难度系数
17      short tag; // 标签 (知识点)
18      short chapter; // 章
19      short section; // 节
20  } Problem;
21
22  Problem *problem_new();
23
24  int problem_read_file(List *list);
25  int problem_write_file(List *list);
26
27  void *by_id(SList *item, void *data);
28  void *by_des(SList *item, void *data);
29  void *by_opt(SList *item, void *data);
30  void *by_difr(SList *item, void *data);
31  void *by_difs(SList *item, void *data);
32  void *by_tags(SList *item, void *data);
33  void *by_secs(SList *item, void *data);
34  void *by_mul(SList *item, void *data);
35
36  #endif
```

```
1  /* problem.c
2  * 题目数据库的增删改查
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <errno.h>
9  #include <assert.h>
10 #include <math.h>
11
12 #include "addon.h"
13 #include "slist.h"
14 #include "list.h"
15 #include "problem.h"
16
17 static char *problem_db_name = "problem.db";
18
19 Problem *problem_new()
20 {
21     Problem *p = (Problem *)malloc(sizeof(Problem));
22     assert(p);
23     p->id = 0;
24     memset(p->des, 0, sizeof(p->des));
25     memset(p->opt, 0, sizeof(p->opt));
26     p->ans = 'A';
27     p->dif = 0;
28     p->tag = 0;
29     p->chapter = 0;
30     p->section = 0;
31     return p;
32 }
33
34 int problem_read_file(List *list)
35 {
36     return read_file_to_list(problem_db_name, list, sizeof(Problem));
37 }
38
39 int problem_write_file(List *list)
40 {
41     return write_list_to_file(problem_db_name, list, sizeof(Problem));
42 }
43
44 void *by_id(SList *item, void *data)
45 {
46     ID *p = (ID *)item->userdata;
47     int *id = (int *)data;
48     return p->id == *id ? item : NULL;
```

```

49 }
50
51 void *by_des(SList *item, void *data)
52 {
53     Problem *p = (Problem *)item->userdata;
54     char *des = (char *)data;
55     return strstr(p->des, des) ? item : NULL;
56 }
57
58 void *by_opt(SList *item, void *data)
59 {
60     Problem *p = (Problem *)item->userdata;
61     char *opt = (char *)data;
62     int i = 0;
63     for (i = 0; i < 4; i++) {
64         if (strstr(p->opt[i], opt) != NULL) {
65             return item;
66         }
67     }
68     return NULL;
69 }
70
71 void *by_difs(SList *item, void *data)
72 {
73     Problem *p = (Problem *)item->userdata;
74     sel_num *t = (sel_num *)data;
75     return select_cond_number(*t, p->dif) ? item : NULL;
76 }
77
78 void *by_difr(SList *item, void *data)
79 {
80     Problem *p = (Problem *)item->userdata;
81     short *dif = (short *)data;
82     //fprintf(stderr, "%d in [%d, %d]\n", p->dif, dif[0], dif[1]);
83     if (dif[0] <= p->dif && p->dif <= dif[1]) {
84         return item;
85     }
86     return NULL;
87 }
88
89 void *by_tags(SList *item, void *data)
90 {
91     Problem *p = (Problem *)item->userdata;
92     short *tags = (short *)data;
93     int i = 0;
94     for (i = 0; tags[i] > 0; i++) {
95         if (p->tag == tags[i]) {
96             return item;
97         }
98     }
99     return NULL;
100 }
101
102 void *by_secs(SList *item, void *data)
103 {
104     Problem *p = (Problem *)item->userdata;
105     double *secs = (double *)data;
106     int i = 0;
107     double a, b;
108     for (i = 0; secs[i] > 0; i++) {
109         b = modf(secs[i], &a);
110         //fprintf(stderr, "%lf %lf\n", a, b);
111         if (p->chapter == (int)a) {
112             if (zero(b) || psame(p->section, b)) {
113                 return item;
114             }
115         }
116     }
117     return NULL;
118 }
119
120 void *by_mul(SList *item, void *data)
121 {
122     Problem *p = (Problem *)item->userdata;
123     char *key = (char *)data;
124     int a, b;
125     if ('0' <= key[0] && key[0] <= '9') {
126         sscanf(key, "%d", &a);
127         if (p->id == a || p->tag == a || p->dif == a || p->chapter == a || p->section == a) {
128             return item;
129         }
130         if (sscanf(key, "%d.%d", &a, &b) == 2 && p->chapter == a && p->section == b) {
131             return item;
132         }
133     }
134     if (by_des(item, data) != NULL) {
135         return item;
136     }
137     if (by_opt(item, data) != NULL) {

```

```

138         return item;
139     }
140     return NULL;
141 }

```

A.7 Paper 类

```

1  /* paper.h
2  * 试卷类
3  */
4
5  #ifndef _PAPER_H_
6  #define _PAPER_H_
7
8  #include "addon.h"
9  #include "slist.h"
10 #include "list.h"
11 #include "problem.h"
12
13 typedef struct Paper {
14     int id;
15     int length;
16     int pid[254];
17     char title[128];
18 } Paper;
19
20 Paper *paper_new();
21 void paper_free(Paper *pa);
22
23 int paper_read_list(List *list);
24 int paper_write_list(List *list);
25
26 int paper_insert_pid(Paper *pa, int pid);
27 void fprintf_paper_pid(Paper *pa, void *file);
28
29 void paper_problem_call(Paper *pa, List *list, SListCallback *call, void *userdata);
30
31 int paper_generate_random(Paper *pa, List *list, int n);
32 int paper_generate_tags(Paper *pa, List *list, int n, int tags[], int m);
33 int paper_generate_secs(Paper *pa, List *list, int n, double secs[], int m);
34 int paper_generate_dif(Paper *pa, List *list, int n, int a, int b);
35
36 #endif

```

```

1  /* paper.c
2  * 试卷类
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdarg.h>
9  #include <math.h>
10 #include <assert.h>
11
12 #include "addon.h"
13 #include "paper.h"
14
15 static char *paper_db_name = "paper.db";
16
17 Paper *paper_new()
18 {
19     Paper *pa = (Paper *)malloc(sizeof(Paper));
20     assert(pa);
21     pa->id = 0;
22     pa->length = 0;
23     memset(pa->title, 0, sizeof(pa->title));
24     memset(pa->pid, 0, sizeof(pa->pid));
25     return pa;
26 }
27
28 int paper_read_list(List *list)
29 {
30     return read_file_to_list(paper_db_name, list, sizeof(Paper));
31 }
32
33 int paper_write_list(List *list)
34 {
35     return write_list_to_file(paper_db_name, list, sizeof(Paper));
36 }
37
38 void fprintf_paper_pid(Paper *pa, void *file)
39 {

```



```

40     fprintf(file, "%d. %s (%d)\n", pa->id, pa->title, pa->length);
41     if (pa->length > sizeof(pa->pid)) {
42         return;
43     }
44     int i = 0;
45     for (i = 0; i < pa->length; i++) {
46         fprintf(file, " %d", pa->pid[i]);
47     }
48     fprintf(file, "\n");
49 }
50
51 void paper_problem_call(Paper *pa, List *list, SListCallback *call, void *userdata)
52 {
53     printf("%s\n", pa->title);
54     int i = 0;
55     for (i = 0; i < pa->length; i++) {
56         printf("(%d) ", i + 1);
57         list_find_each_call(list, by_id, pa->pid + i, call, userdata);
58     }
59 }
60
61 int paper_insert_pid(Paper *pa, int pid)
62 {
63     int i = 0;
64     for (i = 0; i < pa->length; i++) {
65         if (pa->pid[i] == pid) {
66             return -1;
67         }
68     }
69     return pa->pid[pa->length++] = pid;
70 }
71
72 int paper_generate_random(Paper *pa, List *list, int n)
73 {
74     int id = 0;
75     SList *s = NULL;
76     if (list->count < n) {
77         n = list->count;
78     }
79     while (pa->length < n) {
80         id = random(1, list->max_id + 1);
81         s = (SList *)slist_find(list->slist, by_id, &id);
82         if (s != NULL) {
83             paper_insert_pid(pa, id);
84         }
85     }
86     return pa->length;
87 }
88
89 int paper_generate_tags(Paper *pa, List *list, int n, int tags[], int m)
90 {
91     if (list->count < n) {
92         n = list->count;
93     }
94     SList *s = list->slist;
95     Problem *p = NULL;
96     int i = 0, j = 0;
97     int k = n / m + 1;
98     for (i = 0; i < m; i++) {
99         j = 0;
100         while ((s = (SList *)slist_find(s, by_tags, tags + i)) != NULL) {
101             p = (Problem *)s->userdata;
102             if (p->id == paper_insert_pid(pa, p->id)) {
103                 j++;
104                 if (j >= k) {
105                     break;
106                 }
107             }
108             if (pa->length >= n) {
109                 goto end;
110             }
111             s = slist_tail(s);
112         }
113     }
114 end:
115     return pa->length;
116 }
117
118 int paper_generate_secs(Paper *pa, List *list, int n, double secs[], int m)
119 {
120     if (list->count < n) {
121         n = list->count;
122     }
123     SList *s = list->slist;
124     Problem *p = NULL;
125     int i = 0, j = 0;
126     int k = n / m + 1;
127     for (i = 0; i < m; i++) {
128         j = 0;

```

```

129     while ((s = (SList *)slist_find(s, by_secs, secs + i)) != NULL) {
130         p = (Problem *)s->userdata;
131         if (p->id == paper_insert_pid(pa, p->id)) {
132             j++;
133             if (j >= k) {
134                 break;
135             }
136         }
137         if (pa->length >= n) {
138             goto end;
139         }
140         s = slist_tail(s);
141     }
142 }
143 end:
144     return pa->length;
145 }
146
147 int paper_generate_dif(Paper *pa, List *list, int n, int a, int b)
148 {
149     if (a > b) {
150         swap(&a, &b);
151     }
152     if (list->count < n) {
153         n = list->count;
154     }
155     SList *s = list->slist;
156     Problem *p = NULL;
157     int j[11] = { 0 };
158     int k = n / (b - a + 1) + 1;
159     short r[2] = { a, b };
160     //fprintf(stderr, "k = %d\n", k);
161     while ((s = (SList *)slist_find(s, by_difr, &r)) != NULL) {
162         p = (Problem *)s->userdata;
163         if (j[p->dif]++ >= k) {
164             break;
165         }
166         paper_insert_pid(pa, p->id);
167         if (pa->length >= n) {
168             goto end;
169         }
170         s = slist_tail(s);
171     }
172 end:
173     return pa->length;
174 }

```

A.8 Score 类

```

1  /* score.h
2  * 成绩类
3  */
4
5  #ifndef _SCORE_H_
6  #define _SCORE_H_
7
8  #include <time.h>
9
10 #include "user.h"
11 #include "paper.h"
12
13 typedef struct Score {
14     int id;
15     int user_id;
16     char username[128];
17     int paper_id;
18     int paper_count;
19     int right, now;
20     time_t date;
21     char answer[256];
22 } Score;
23
24 Score *score_new(User *u, Paper *p);
25
26 int score_read_list(List *list);
27 int score_write_file(List *list);
28
29 int score_did(Score *s, char c, char ans);
30
31 void *by_user_id(SList *item, void *data);
32
33 #endif

```

```

1  /* score.c
2   * 成绩类
3   */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <assert.h>
9
10 #include "score.h"
11
12 static char *score_db_name = "score.db";
13
14 Score *score_new(User *u, Paper *p)
15 {
16     assert(u);
17     assert(p);
18     Score *s = (Score *)malloc(sizeof(Score));
19     assert(s);
20     s->id = 0;
21     s->user_id = u->id;
22     memset(s->username, 0, sizeof(s->username));
23     strcpy(s->username, u->username);
24     s->paper_id = p->id;
25     s->paper_count = p->length;
26     s->right = 0;
27     s->now = 0;
28     time(&s->date);
29     memset(s->answer, 0, sizeof(s->answer));
30     return s;
31 }
32
33 int score_read_list(List *list)
34 {
35     return read_file_to_list(score_db_name, list, sizeof(Score));
36 }
37
38 int score_write_file(List *list)
39 {
40     return write_list_to_file(score_db_name, list, sizeof(Score));
41 }
42
43 int score_did(Score *s, char c, char ans)
44 {
45     assert(s->now <= s->paper_count);
46     s->answer[s->now++] = c;
47     s->right += (c == ans);
48     return s->now;
49 }
50
51 void *by_user_id(SList *item, void *data)
52 {
53     Score *s = (Score *)item->userdata;
54     int *user_id = (int *)data;
55     return s->user_id == *user_id ? item : NULL;
56 }

```

A.9 User 类

```

1  /* user.h
2   * 用户类 (学生和教师数据)
3   */
4
5  #ifndef _USER_H_
6  #define _USER_H_
7
8  #include <stdbool.h>
9
10 typedef struct User {
11     int id;
12     char username[32];
13     char passwd[64];
14     bool teacher;
15 } User;
16
17 User *user_new();
18 void user_free(User *u);
19
20 int user_reg(User *u);
21 int user_login(User *u);
22
23 #endif

```

```

1  /* user.c
2  * 用户类 (学生和教师数据)
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdbool.h>
9  #include <errno.h>
10 #include <assert.h>
11
12 #include "list.h"
13 #include "user.h"
14
15 static char *user_list_name = "user.db";
16
17 User *user_new()
18 {
19     User *u = (User *)malloc(sizeof(User));
20     assert(u);
21     u->id = 0;
22     memset(u->username, 0, sizeof(u->username));
23     memset(u->passwd, 0, sizeof(u->passwd));
24     u->teacher = false;
25     return u;
26 }
27
28 void user_free(User *u)
29 {
30     free(u);
31 }
32
33 int user_init()
34 {
35     User u = {
36         .id = 1,
37         .username = "root",
38         .passwd = "root",
39         .teacher = true
40     };
41     if (user_reg(&u) < 0) {
42         return -1;
43     }
44     return 0;
45 }
46
47 int user_reg(User *u)
48 {
49     FILE *file = fopen(user_list_name, "ab");
50     if (file == NULL) {
51         return -1;
52     }
53     int n = file_data_count(file, sizeof(User));
54     u->id = n + 1;
55     if (fseek(file, 0, SEEK_END) != 0) {
56         fclose(file);
57         return -1;
58     }
59     if (fwrite(u, sizeof(User), 1, file) != 1) {
60         fclose(file);
61         return -1;
62     }
63     if (fclose(file) != 0) {
64         return -1;
65     }
66     //fprintf(stderr, "%d %s %s %d\n", u->id, u->username, u->passwd, u->teacher);
67     return u->id;
68 }
69
70 int user_login(User *u)
71 {
72     FILE *file = fopen(user_list_name, "rb");
73     if (file == NULL) {
74         if (user_init() < 0) {
75             return -1;
76         }
77         file = fopen(user_list_name, "rb");
78         if (file == NULL) {
79             return -1;
80         }
81     } else if (file_data_count(file, sizeof(User)) < 1) {
82         fclose(file);
83         if (user_init() < 0) {
84             return -1;
85         }
86     }
87     User v;
88     rewind(file);

```

```

89     while (!feof(file)) {
90         fread(&v, sizeof(v), 1, file);
91         //fprintf(stderr, "%d %s %s %d\n", v.id, v.username, v.passwd, v.teacher);
92         if (strcmp(v.username, u->username) == 0) {
93             if (strcmp(v.passwd, u->passwd) == 0) {
94                 u->id = v.id;
95                 u->teacher = v.teacher;
96                 goto end;
97             }
98             errno = EACCES;
99             return -2;
100         }
101     }
102 end:
103     if (fclose(file) == EOF) {
104         return -1;
105     }
106     return u->id;
107 }

```

A.10 辅助方法

```

1  /* addon.h
2  * 附加函数库
3  */
4
5  #ifndef _ADDON_H_
6  #define _ADDON_H_
7
8  #include <stdbool.h>
9
10 #ifndef max
11 #define max(a, b) ((a) > (b) ? (a) : (b))
12 #endif
13
14 #ifndef min
15 #define min(a, b) ((a) < (b) ? (a) : (b))
16 #endif
17
18 void swap(int *a, int *b);
19
20 typedef struct sel_num {
21     int num;
22     char mark[8]; // < > = == <= >= != <>
23 } sel_num;
24
25 bool select_cond_number(sel_num cond, int data);
26
27 bool zero(double d);
28 bool psame(int n, double d);
29
30 bool gotn();
31
32 int random(int a, int b);
33
34 char *getpass(char *prompt);
35
36 char *read_string(char *s, size_t n);
37
38 #endif

```

```

1  /* addon.h
2  * 附加函数库
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <math.h>
9
10 #include "addon.h"
11
12 #ifdef WIN32
13 #include <conio.h>
14 #else
15 #include <termios.h>
16 #endif
17
18 void swap(int *a, int *b)
19 {
20     int t = *a;
21     *a = *b;
22     *b = t;

```

```

23 }
24
25 bool gotn()
26 /* 清理遗留的回车换行符 */
27 {
28     char c;
29     c = getchar();
30     if (c == '\n' || c == '\r') {
31         return true;
32     }
33     ungetc(c, stdin);
34     return false;
35 }
36
37 bool select_cond_number(sel_num cond, int data)
38 /* 查询条件 */
39 {
40     //fprintf(stderr, "%d - [%d %s]\n", data, cond.num, cond.mark);
41     if (strcmp(cond.mark, "<") == 0) {
42         return data < cond.num;
43     } else if (strcmp(cond.mark, ">") == 0) {
44         return data > cond.num;
45     } else if (strcmp(cond.mark, "<=") == 0) {
46         return data <= cond.num;
47     } else if (strcmp(cond.mark, ">=") == 0) {
48         return data >= cond.num;
49     } else if (strcmp(cond.mark, "!=") == 0 || strcmp(cond.mark, "<>") == 0) {
50         return data != cond.num;
51     } else { // "==" "==" ""
52         return data == cond.num;
53     }
54 }
55
56 bool zero(double d)
57 /* 浮点数是否为零 */
58 {
59     return fabs(d - 0) <= 1e-6;
60 }
61
62 bool psame(int n, double d)
63 /* 浮点数 d 的小数位是否等于整数 n */
64 {
65     int i = 0;
66     double m = n;
67     for (i = 0; i < 6; i++) {
68         if (zero((m /= 10) - d)) {
69             return true;
70         }
71     }
72     return false;
73 }
74
75 int random(int a, int b)
76 /* 返回 [a, b) 之间的随机数 */
77 {
78     return rand() % (b - a) + a;
79 }
80
81 char *read_string(char *s, size_t n)
82 {
83     fgets(s, n, stdin);
84     int l = strlen(s);
85     while (--l) {
86         if (s[l] == '\n' || s[l] == '\r') {
87             s[l] = '\0';
88             return s;
89         }
90     }
91     return s;
92 }
93
94 #define PWD_MAX 256
95
96 char *getpass(char *prompt)
97 {
98     static char passwd[PWD_MAX] = "";
99     printf("%s", prompt);
100 #ifdef WIN32
101     int i = 0;
102     char c;
103     while ((c = _getch()) != '\n' && c != '\r' && i < PWD_MAX) {
104         if (c == '\b' && i > 0) {
105             passwd[i--] = '\0';
106             printf("\b \b");
107         } else {
108             passwd[i++] = c;
109             printf("*");
110         }
111     }

```

```

112     passwd[i] = '\0';
113 #else
114     struct termios oldflags, newflags;
115     tcgetattr(fileno(stdin), &oldflags);
116     newflags = oldflags;
117     newflags.c_lflag &= -ECHO;
118     newflags.c_lflag |= ECHONL;
119     if (tcsetattr(fileno(stdin), TCSANOW, &newflags) != 0) {
120         perror("tcsetattr");
121         return NULL;
122     }
123     fgets(passwd, PWD_MAX, stdin);
124     if (tcsetattr(fileno(stdin), TCSANOW, &oldflags) != 0) {
125         perror("tcsetattr");
126         return NULL;
127     }
128 #endif
129     return passwd;
130 }

```

A.11 工程文件

```

1  RM=rm -f
2  CC= gcc
3  DEFS=
4  PROGRAM= exam
5  INCLUDES= -I.
6  LIBS=
7
8  ifdef WIN32
9  OSDEF= -fexec-charset=gbk
10 else
11 OSDEF= -DSYS_UNIX=1
12 endif
13
14 DEFINES= $(INCLUDES) $(DEFS) $(OSDEF)
15 CFLAGS= -Wall -g $(DEFINES)
16
17 SRCS = addon.c user.c exam.c list.c paper.c problem.c score.c slist.c ui.c
18
19 OBJS = addon.o user.o exam.o list.o paper.o problem.o score.o slist.o ui.o
20
21 .c.o:
22     $(RM) $@
23     $(CC) $(CFLAGS) -c $.c
24
25 all: $(PROGRAM)
26
27 $(PROGRAM) : $(OBJS)
28     $(CC) $(CFLAGS) -o $(PROGRAM) $(OBJS) $(LIBS)
29
30 clean:
31     $(RM) $(OBJS) $(PROGRAM) core *~

```

```

1  CC= cl
2  #DEFS= -nologo -DSTRICT -G3 -Ow -W3 -Zp -Tp
3  DEFS= -nologo
4  PROGRAM= exam.exe
5  LINKER=link -nologo
6
7  INCLUDES= -I.
8
9  DEFINES= $(INCLUDES) $(DEFS) -DWINNT=1 -DWIN32=1
10
11 CFLAGS= $(DEFINES)
12
13 SRCS = addon.c user.c slist.c list.c paper.c problem.c score.c ui.c exam.c
14
15 OBJS = addon.obj user.obj slist.obj list.obj paper.obj problem.obj score.obj ui.obj exam.obj
16
17 .c.obj:
18     $(CC) $(CFLAGS) -c $< -Fo$@
19
20 all: $(PROGRAM)
21
22 $(PROGRAM) : $(OBJS)
23     $(LINKER) $(OBJS) /OUT:$(PROGRAM) $(LIBS)
24
25 clean:
26     del $(OBJS) $(PROGRAM) core

```

参考文献

- [1] A. Shalloway and J. R. Trott, *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Pearson Education, 2 ed., 2005.
- [2] 徐言生, ed., 设计模式解析: 第二版 (修订版). 北京: 人民邮电出版社, 2 ed., 2013.
- [3] 李先静, 系统程序员成长计划. 北京: 人民邮电出版社, 2010.
- [4] R. Strandh, *Modular C: How to Write Reusable and Maintainable Code using the C Language*. <http://dept-info.labri.u-bordeaux.fr/~idurand/enseignement/PFS/Common/Strandh-Tutorial/reuse.pdf>, 1994.
- [5] J. Tidwell and D. Dream, 界面设计模式. 北京: 电子工业出版社, 2013.
- [6] B. W. Kernighan and D. M. Ritchie, eds., *C 程序设计语言*. 北京: 机械工业出版社, 2 ed., 2004.
- [7] 王垠, “完全用 linux 工作,” 2004.
- [8] 殷建平, 徐云, 王刚, 刘晓光, 苏明, 邹恒明, and 王宏志, eds., 算法导论 (第三版). 北京: 机械工业出版社, 3 ed., 2012.
- [9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (Third Edition)*. London, England: The MIT Press, 3 ed., 2009.
- [10] D. E. Knuth, *The Art Of Computer Programming*. Pearson Education, 1968–2011.
- [11] 高德纳, 计算机程序设计艺术. 北京: 国防工业出版社, 1992–2010.
- [12] 邓建松, 彭冉冉, and 陈长松, *L^AT_EX 2_ε 科技排版指南*. 北京: 科学出版社, 2001.
- [13] M. Kerrisk, *Linux/UNIX 系统编程手册*. 北京: 人民邮电出版社, 2014.