学号	0122210880104

武汉理工大学

《程序综合设计实验》报告

学　　院	计算机与人工智能学院
专　　业	计算机类
班　　级	计算机类 m2201
姓　　名	吴明洲
指导教师	张蕊

日期　2023-04-26

目录

1 实验目的

通过迭代式开发，深入掌握 C/C++ 语言的文件、链表、结构体、动态内存管理等技术，开发实现一个计费管理软件。

1) 深入理解 C/C++ 语言的基本概念和基本原理，如数据类型等，熟练应用顺序选择和循环结构程序设计、函数、结构体、文件读写等基础技术。

2) 掌握 C/C++ 语言的高级知识，如类、vector、链表等技术。

3) 掌握模块化开发的具体实现方法，深入领会一些 C/C++ 程序设计实用开发和技巧。

4) 了解迭代软件开发的一般过程，领会系统设计、系统实现以及系统测试的方法

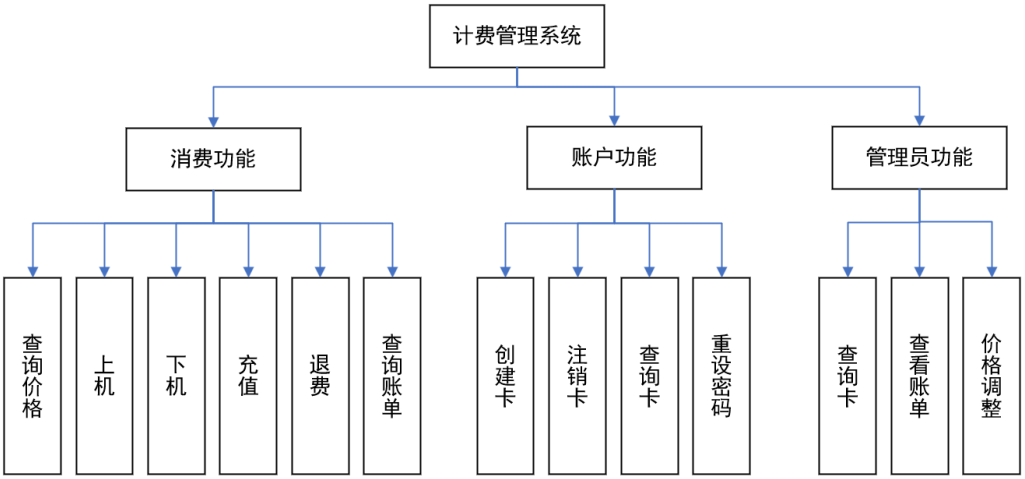
2 系统功能与描述

2.1 系统介绍

本计费管理系统主要是模拟实现网吧收费的基本功能，提供消费功能、账户功能、管理员功能。　　在网吧上机的用户首先在网吧进行开卡，注册一张上级卡。在往卡里充值后，每次上机时，系统根据上下机时间计算费用，从卡中扣除该费用。用户可以要求退回卡中的余额，也可以存放在卡中供下次上机使用。系统会记载用户每次消费、充值、退费的信息，供查询使用。

2.2 功能结构

系统功能结构图如下图所示。



2.3 设计思路

计费管理系统的所涉及到的数据存储在文本文件中。本系统有 3 个文本文件，分别是：

- card.txt 卡信息文件，存储所有上机卡
- billing.txt 消费、充值、退费信息等账单信息记录文件
- price.txt 价格信息文件

1) 计费管理系统启动过程中要完成下列功能

- 从文本文件 card.txt 中读入所有卡信息，每张卡片信息存储在一个结构体变量中
- 从文本文件 billing.txt 中读入所有账单信息，每条账单信息存储在一个结构体变量中
- 用 vector 容器存储所有的卡片及账单信息
- 从文本文件 price.txt 中读入价格信息

2) 计费管理系统退出过程中要完成下列功能

- 保存对卡片信息的更新，即将所有卡片信息写入到文本文件 card.txt

- 保存对账单信息的更新，即将所有账单信息写入到文本文件 billing.txt
- 保存对价格信息的更新，即将价格信息写入到文本文件 price.txt

3 典型算法分析 (完整源码请见光盘)

3.1 二进制文件读写

卡片信息和账单信息保存在容器 vector 中，vector 容器是内存中的变量，当系统退出后，添加的数据就会消失，不能永久保存。要想实现永久保存，必须将信息写入文件中

因为每张卡片信息或每条账单信息存储在结构体中，当我们使用常规的文件读写（文本文件读写）时，不便于我们处理数据。故在此考虑使用二进制文件读写操作，能够按结构体进行读写，便于我们操作数据

3.1.1 文件流

头文件 #include<fstream> 提供了三个类，用来实现 C++ 对文件的操作：

- fstream(读写文件)
- ifstream(读文件)
- ofstream(写文件)

3.1.2 文件打开方式和读写方式

打开方式

读：`ifstream filename(char *buffer, ios::binary | ios::in)`

写：`ofstream filename(char *buffer, ios::binary | ios::out)`

其中`buffer`是一块内存地址，表示文本路径，用来存储或读取数据`ios::binary`用于指定以二进制方式打开文件，`ios::in`和`ios::out`为缺省值，

ios::out打开文件的同时会截断文件内容。还有ios::app不截断文件内容，只在文件末尾追加内容

读写方式

读：read(char *buffer, streamsize size)

写：write(char *buffer, streamsize size)

其中buffer是一块内存地址，表示文本路径，用来存储或读取数据。size表示要从缓存 (buffer) 中读出或写入的字符数

3.1.3 实现代码

向文件中写入卡信息

```
void CardVector::saveCard(const string &cardPATH) {
    ofstream cardfile(cardPATH, ios::out | ios::binary);
    if (!cardfile.is_open()) return;
    for (auto it: vec) cardfile.write((char *) &it, sizeof(Card));
    cardfile.close();
}
```

更新指定的卡信息

```
void CardVector::updateCard(const Card *p, const string &cardPATH, int
    CardIndex) {
    fstream cardfile(cardPATH, ios::in | ios::out);
    if (!cardfile.is_open()) return;
    cardfile.seekp(sizeof(Card) * CardIndex, ios::beg);
    cardfile.write((char *) p, sizeof(Card));
    cardfile.close();
}
```

从文件中读取卡信息

```
CardVector::CardVector(const string &cardPATH) {
    ifstream cardfile(cardPATH, ios::in | ios::binary);
    Card card{};
    if (!cardfile.is_open()) return;
    while (true) {
        cardfile.read((char *) &card, sizeof(Card));
        if (cardfile.eof())break;
```

```
        vec.push_back(card);
    }
    cardfile.close();
}
```

对于账单信息和价格信息的操作类似

3.2 时间类型与字符串类型的互相转换

3.2.1 与时间有关的知识

C++ 提供了与时间处理相关的一些函数和与时间处理有关的数据类型 time_t 和 tm

time_t 是 64 位长整数，精确到秒，表示从 1970 年 1 月 1 日的零点开始到当前时间经过了多少秒

tm 是结构体类型，如下所示：

```
struct tm {
    int tm_sec; /*秒，0-59*/
    int tm_min; /*分钟，0-59*/
    int tm_hour; /*小时，  0-23*/
    int tm_mday; /*日，1-31*/
    int tm_mon; /*月，0-11*/
    int tm_year; /*年，从1900至今已经多少年*/
    int tm_wday; /*星期，从星期日算起，0-6*/
    int tm_yday; /*天数，0-365*/
    int tm_isdst; /*日光节约时间的旗标*/
};
```

常见的时间函数有：

`time_t time(time_t *t)` 取得从 1970 年 1 月 1 日的零点至今的秒数

`time_t t = time(NULL)` 获取本地时间

`struct tm *localtime(const time_t *clock)` 将长整数时间转换为结构体时间，从中得到年月日、星期、时分秒等信息

`time_t mktime(struct tm *timeptr)` 将 tm 结构的时间转换为长整数从 1970 年至今的秒数

由于一般用户能够接受的时间是字符串“2017 年 3 月 8 日 15 时 30 分”这样的形式，因此我们需要编写自己的时间处理函数，将结构体 tm 中相关信息取出来组合成这样的字符串

3.2.2 实现代码

```
// 定义函数timeToString，将时间转换成字符串格式
void timeToString(time_t t, char *Buf) {
    tm *timeinfo;
    timeinfo = localtime(&t);
    strftime(Buf, 20, "%Y-%m-%d %H:%M", timeinfo);
}

// 定义函数stringToTime，将字符串格式的时间转换成time_t类型
time_t stringToTime(char *Time) {
    tm tm1{};
    // 使用sscanf函数将字符串中的时间数据读入tm结构体
    sscanf(Time, "%d-%d-%d %d:%d", &tm1.tm_year, &tm1.tm_mon, &tm1.
        tm_mday, &tm1.tm_hour, &tm1.tm_min);
    tm1.tm_year -= 1900;
    tm1.tm_mon -= 1;
    tm1.tm_sec = 0;
    // 设置tm_isdst为-1，表示使用本地时区
    tm1.tm_isdst = -1;
    return mktime(&tm1);
}
```

3.3 密码型文本输入

系统在要求用户输入密码时，出于保护用户隐私和账户安全的考虑，一般将用户输入的密码显示为 * 号或 • 号

运用到了_getch()函数，其特点为从控制台读取一个字符，但不显示在屏幕上，因此我们在此基础上根据用户操作进行输出 * 号或退格即可

3.3.1 代码实现

```
// 输入密码，以星号代替，并返回字符串
string CardVector::cinPwd() {
    char password[20];
    char ch;
    int i = 0;
    while ((ch = _getch()) != '\r') {
        if (ch == '\b') {
            if (i > 0) {
                i--;
                putchar('\b');
                putchar(' ');
                putchar('\b');
            }
        } else {
            password[i] = ch;
            i++;
            putchar('*');
        }
    }
    password[i] = '\0';
    return password;
}
```

3.4 模糊搜索 (计算编辑距离)

用户在进行查询卡信息时，可能因为不记得确切卡号或按错键而导致找不到卡，利用模糊搜索功能可在找不到卡时向用户提供最接近的卡号，以便于用户搜索

3.4.1 计算编辑距离算法

编辑距离（Edit Distance）最常用的定义就是 Levenstein 距离，是由俄国科学家 Vladimir Levenshtein 于 1965 年提出的，所以编辑距离一般又称 Levenshtein 距离。它主要作用是测量两个字符串的差异化程度，表示字符串 a 至少要经过多少个操作才能转换为字符串 b，这里的操作包括三种：增加、删

除、替换

先从一个问题谈起：对于字符串”xyz” 和”xcz”，它们的最短距离是多少？我们从两个字符串的最后一个字符开始比较，它们都是’z’，是相同的，我们可以不用做任何操作，此时二者的距离实际上等于”xy” 和”xc” 的距离，即 d(xyz,xcz) = d(xy,xc)。也即是说，如果在比较的过程中，遇到了相同的字符，那么二者的距离是除了这个相同字符之外剩下字符的距离。即 d(i，j) = d(i - 1,j-1)

接着，我们把问题拓展一下，最后一个字符不相同的情况：字符串 A(”xyzab”) 和字符串 B(”axyzc”)，问至少经过多少步操作可以把 A 变成 B

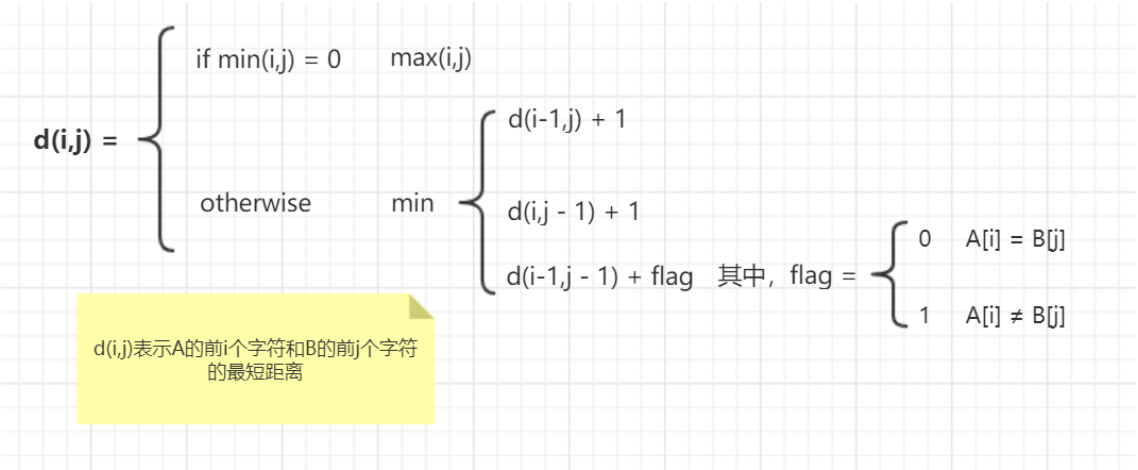
我们还是从两个字符串的最后一个字符来考察即’b’ 和’c’。显然二者不相同，那么我们有以下三种处理办法：

(1) 增加：在 A 末尾增加一个’c’，那么 A 变成了”xyzabc”，B 仍然是”axyzc”，由于此时末尾字符相同了，那么就变成了比较”xyzab” 和”axyz” 的距离，即 d(xyzab,axyzc) = d(xyzab,axyz) + 1。可以写成 d(i,j) = d(i,j - 1) + 1。表示下次比较的字符串 B 的长度减少了 1，而加 1 表示当前进行了一次字符的操作

(2) 删除：删除 A 末尾的字符’b’，考察 A 剩下的部分与 B 的距离。即 d(xyzab,axyzc) = d(xyza,axyzc) + 1。可以写成 d(i,j) = d(i - 1,j) + 1。表示下次比较的字符串 A 的长度减少了 1

(3) 替换：把 A 末尾的字符替换成’c’，这样就与 B 的末尾字符一样了，那么接下来就要考察出了末尾’c’ 部分的字符，即 d(xyzab,axyzc) = d(xyza,axyz) + 1。写成 d(i,j) = d(i -1,j-1) + 1 表示字符串 A 和 B 的长度均减少了 1。

按照以上思路，我们很容易写出下面的方程：



可以使用递归或动态规划完成

3.4.2 代码实现 (使用动态规划)

```
// 计算两个字符串之间的编辑距离
int CardVector::editDistance(string str1, string str2) {
    int len1 = str1.size(), len2 = str2.size();
    vector<vector<int>> dp(len1 + 1, vector<int>(len2 + 1, 0));
    for (int i = 0; i <= len1; i++) dp[i][0] = i;
    for (int j = 0; j <= len2; j++) dp[0][j] = j;
    for (int i = 1; i <= len1; i++)
        for (int j = 1; j <= len2; j++)
            if (str1[i - 1] == str2[j - 1]) dp[i][j] = dp[i - 1][j - 1];
            else dp[i][j] = min(dp[i - 1][j], min(dp[i][j - 1], dp[i -
                1][j - 1])) + 1;
    return dp[len1][len2];
}

// 返回最接近给定卡号的卡片
vector<Card> CardVector::getTopMatches(const string &CardName) {
    priority_queue<pair<int, int>, vector<pair<int, int>>,
        compareDistance> pq;
    vector<Card> res;
    for (int i = 0; i < vec.size(); i++) {
        int dist = editDistance(CardName, vec[i].CardName);
        pq.push({dist, i});
    }
    for (int i = 0; i < 5 && !pq.empty(); i++) {
```

```
        int idx = pq.top().second;
        res.push_back(vec[idx]);
        pq.pop();
    }
    return res;
}
```

4 开发难点与体会

- **难点一：**文件读写出现问题。在结构体中使用了 string 类型变量，写进文件时并未发现异常，但导致在初始化文件时读取文件异常，程序不能正常执行。原因是 string 类型变量实际上是一个指针，写入文件时，实际上写入的是 string 分配的动态地址而非 string 类型变量所指的内容

  解决方法：把 string 类型变量改为用字符数组表示即可

  体会：很难找到错误的原因，耗费大量时间

- **难点二：**函数重复声明问题。由于系统是分文件写的，当不同的文件互相引用时，很容易就出现函数的重复声明，导致系统不能运行

  解决方法：理清不同文件的包含关系，在.h 文件中使用`#ifndef`或`#pragma once`

  体会：调试难度极大，耗费心神

- **难点三：**不同的类之间方法的调用问题。不同的类之间可以实现友元类或友元函数以方便调用，但使用友元函数是总是出现不明原因的函数

  解决方法：放弃使用友元，用老方法定义空对象再使用该对象调用方法

  体会：C++ 中类的知识深奥复杂，任有很大一部分需要深入学习

5 实验总结

通过本次高级语言程序设计，使我对网吧计费管理系统有了进一步的认识和了解，也对开发系统的需求分析的步骤更加熟悉，并且能够利用 Clion 创建该系统，从而实现了对网吧计费的管理

但是本系统只是一个初步的实现，而且，它还有一定的缺陷，比如还没有实现添加管理员服务，只能默认系统里固定的管理员，所以需要进一步分析以及进一步深入，使其更加的完善。在今后的学习过程中我会更加努力的学习这类知识，正确做出更好的系统

《程序综合设计实验》成绩评定表

班级：计算机类 m2201　　姓名：吴明洲　　学号：0122210880104

序号	评分项目	满分	实得分
1	学习态度认真、遵守纪律	10	
2	迭代开发进度合理，提交结果正确	40	
3	代码规范、注释清晰、可读性好	10	
4	软件功能完善、运行正确	20	
5	验收情况良好	10	
6	报告规范、描述清晰准确	10	
		总评分	
评语:			

指导教师签名：

2023 年　　月　　日