**实验十二 模块化设计**

1. **实验目的和要求**

设 计 一 个 学 生 成 绩 管 理 程 序 ， 实 现 对 n 个 学 生 的 m 门 课 程 的 成 绩 的 记 录 与 统 计 工 作 。

学 生 信 息 包 括 ： 学 号 、 姓 名 、 数 学 成 绩 、 英 语 成 绩 、 物 理 成 绩 。

程 序 基 本 功 能 要 求 如 下 ：

a. 能 够 新 增 学 生 信 息 ， 并 计 算 总 分 和 平 均 分 。

b. 能 够 分 别 根 据 学 号 和 姓 名 查 询 该 学 生 的 基 本 信 息。

c. 能 够 根 据 学 号 修 改 某 个 学 生 的 信 息

d. 能 够 显 示 所 有 学 生 的 成 绩 信 息

e. 能 够 分 别 按 照 学 号 和 总 分 进 行 排 序 。

程 序 运 行 时 ， 出 现 菜 单 ， 形 式 如 下 ：

欢 迎 使 用 成 绩 管 理 系 统 ！

1 一 添 加 学 生 信 息

2一 修 改 学 生 信 息

3 一 显 示 全 部 学 生 信 息

4 一 按 学 号 查 询 学 生 信 息

5 一 按 姓 名 查 询 学 生 信 息

6 一 按 学 号 升 序 排 序

7 一 按 总 分 降 序 排 序

0 一 退出

**二、实验内容**

1. 实验准备
   1. 理论知识介绍
      1. 数据的输入：

要从用户依次读入int类型的变量a,n，可以使用如下语句:

cin >> a>>n;

* + 1. 数据的输出

要将变量的内容显示在显示器上，可以使用cout和流插入运算符。Cout还可以输出表达式的执行结果，比如:

cout << a << ‘+’ << b << ‘=’ << a+b << endl;

* + 1. 算术表达式

C++中，算术运算里有+（加法），-（减法），/（除法），%（取模），但是没有取平方算数运算。

* + 1. 数据类型 （整型）：

C++中一个整型变量可以储存一个整数，可以直接通过cin和cout输入和输出。

* + 1. 关系表达式

关系表达式用于比较两个值的大小。C++提供了6个关系运算符: <, <=, >, >=,

==, !=。关系运算符左结合。

* + 1. for语句

**for (i = 0; i < n; ++i)**

**需要重复执行的语句**

* + 1. while语句

**while (表达式) {**

**需要重复执行的语句**

**}**

* + 1. switch 语句

**switch (控制表达式) {**

**case 常量表达式1: 语句1； break;**

**case 常量表达式2: 语句2； break;**

**…**

**case 常量表达式n: 语句n； break;**

**default: 语句n+1;**

**}**

* + 1. 函数

编写一个实现某个功能的函数称为函数定义。一旦定义了一个函数，在程序中就可以反复调用这个函数。函数的定义形式如下：

**类型名 函数名 (形式参数表)**

**{**

**变量定义部分**

**语句部分**

**}**

* + 1. 指针

所谓的指针变量就是保存另一个变量地址的变量。指针变量存在的意义在于提供间接访问，即从一个变量访问另一个变量的值，使变量访问更加灵活。

指针变量的定义如下：

**类型名 \*指针变量名**

* + 1. 函数指针

指向函数的指针的定义为：

**返回类型 （\*指针变量） （形式参数列表）；**

指向函数的指针主要有两个用途：作为另一个函数的参数，以及用于实现菜单选择。

利用函数指针我们能将一个函数作为另一个函数的参数。

1. 实验项目
   1. 分析
      1. 由于我们要实现洗牌和发牌这两个功能，所以要设计两个函数分别实现这个功能。
      2. 同时为了灵活地利用这两个函数，我们可以再定义一个函数，将这两个函数作为这个函数的参数，这样就能比较方便地实现选择列表的功能。
   2. 方案
      1. 由用户输入一个标志变量
      2. 通过用户输入的标志来判断将要进行什么操作
      3. 利用函数指针和switch判断语句来进行对应的操作和对应的次数
      4. 洗牌功能的函数利用随机数生成器来进行对应的随机两个位置的牌的交换从而实现简单的洗牌的功能
      5. 发牌功能需要函数从deck里面依次抽取出相应数量的牌并打印出来
   3. 源程序

#include <iostream>

using namespace std;

const char \*suit[4] = { "Spades", "Hearts", "Clubs", "Diamonds" };

const char \*face[13] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",

"Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };

int deck[52];

int index = 0, i = 0;

void Swap(int &x, int &y);

int Func(void(\*p)(int times), int times);

void Shuffle(int times);

void Deal(int times);

int main() {

bool quit = false;

int times = 10;

char choice;

for (i = 0; i < 52; i++) {

deck[i] = i;

}

while (!quit) {

cout << "q: quit; 1: shuffle; 2: deal;" << endl;

cin >> choice;

switch (choice) {

case 'q':

quit = true;

break;

case '1':

cout << "How many times?" << endl;

cin >> times;

Func(Shuffle, times);

break;

case '2':

cout << "How many cards?" << endl;

cin >> times;

Func(Deal, times);

break;

default:

cout << "Please enter a valid value" << endl;

}

}

return 0;

}

void Swap(int &x, int &y) {

int temp = x;

x = y;

y = temp;

}

int Func(void(\*p)(int times), int times) {

p(times);

return 0;

}

void Shuffle(int times) {

for (i = 0; i < times; i++) {

Swap(deck[index + rand() % (52 - index)], deck[index + rand() % (52 - index)]);

}

}

void Deal(int times) {

int old\_index = index;

for (i = index; i < old\_index + times; i++) {

cout << face[deck[i] % 13] << " " << suit[deck[i] / 13] << " ";

if (index == 51) {

index = 0;

cout << "\nAll cards are dealt. Deal from beginning again." << endl;

break;

}

++index;

}

cout << endl;

}

1. 测试数据、运行结果

q: quit; 1: shuffle; 2: deal;

1

How many times?

50

q: quit; 1: shuffle; 2: deal;

2

How many cards?

10

Ace Spades Ten Diamonds Seven Hearts Eight Diamonds Six Hearts Four Diamonds Deuce Spades Three Diamonds Ten Hearts Three Hearts

q: quit; 1: shuffle; 2: deal;

1

How many times?

100

q: quit; 1: shuffle; 2: deal;

2

How many cards?

20

Seven Diamonds Seven Clubs Four Spades King Hearts Nine Clubs Queen Hearts Nine Spades Ace Diamonds Five Hearts Five Diamonds Jack Clubs Three Clubs Three Spades Ace Clubs Deuce Clubs Nine Diamonds Ten Spades Five Spades King Clubs Four Hearts

q: quit; 1: shuffle; 2: deal;

1

How many times?

1000

q: quit; 1: shuffle; 2: deal;

2

How many cards?

30

Seven Spades Ten Clubs Jack Diamonds Jack Spades Six Diamonds King Spades Eight Hearts Nine Hearts King Diamonds Six Spades Eight Spades Queen Spades Queen Clubs Four Clubs Ace Hearts Six Clubs Deuce Diamonds Five Clubs Deuce Hearts Queen Diamonds Jack Hearts Eight Clubs

All cards are dealt. Deal from beginning again.

q: quit; 1: shuffle; 2: deal;

2

How many cards?

40

Ace Spades Ten Diamonds Seven Hearts Eight Diamonds Six Hearts Four Diamonds Deuce Spades Three Diamonds Ten Hearts Three Hearts Seven Diamonds Seven Clubs Four Spades King Hearts Nine Clubs Queen Hearts Nine Spades Ace Diamonds Five Hearts Five Diamonds Jack Clubs Three Clubs Three Spades Ace Clubs Deuce Clubs Nine Diamonds Ten Spades Five Spades King Clubs Four Hearts Seven Spades Ten Clubs Jack Diamonds Jack Spades Six Diamonds King Spades Eight Hearts Nine Hearts King Diamonds Six Spades

q: quit; 1: shuffle; 2: deal;

q

1. 异常现象

初始的发牌函数是这样的

void Deal(int times) {

for (i = index; i < index + times; i++) {

cout << face[deck[i] % 13] << " " << suit[deck[i] / 13] << " ";

if (index == 51) {

index = 0;

cout << "\nAll cards are dealt. Deal from beginning again." << endl;

}

++index;

}

cout << endl;

}

这样写会导致两个问题：

1. 由于index在for循环中每次都要进行自增，所以i < index + times 会一直成立，所以这个for循环并不能得到想要的发一定数量牌的功能。
2. 由于将 ++index 这个语句放在if语句之后，所以如果index 达到51之后重新开始循环的值为1而不是预期的0。
3. 分析

为了解决第一个问题，我们需要新建一个变量来储存发牌之前的index的值，我们把这个变量叫做old\_index，这样for循环就不会一直执行下去从而产生死循环。

解决第二个问题我们可以在if语句里添加一条break语句，从而当index达到51时我们就跳出for循环语句。

1. 总结

我们在for循环的语句中一定要注意终止条件要能够达到， 不能让for循环一直进行下去，同时我们要学会利用break语句来进行灵活的循环的跳出。

**三、实验小结**

1. 完成情况

修改了发牌的函数之后得到了较为理想的结果。

1. 重点

函数指针的定义以及将函数作为指针从而产生选择菜单的效果。

1. 难点

函数指针和函数之间的等价关系以及将函数作为其他函数的参数的时候如何正确地表示该函数。

1. 解决方法

将函数作为其他函数的参数的时候，作为参数的函数不用包含函数体，但需要包含返回值类型和参数列表，而且我们也可以把函数的指针作为函数的参数。

1. 有待改进之处

int Func(void (\*p)(int times), int times);

这个函数其实不太必要，我们可以将其删去，并添加一个指向函数的指针的数组，这样调用函数起来相对就比较简单。

1. 收获、体会

没有必要使用和题目中所给出的数据类型完全相同的数据类型。虽然函数的指针运用起来非常灵活，但是这样的灵活性却让整个程序的可读性得到了一定的减弱，也让函数的定义写起来比较困难且容易出错。