

河南工业大学

《面向对象程序设计》实验报告

专业班级： 软件 1305 班 学号： 201316920311 姓名： 田劲锋

实验单元一 类和对象

实验一 标准控制台输入输出

实验时间： 2014 年 11 月 28 日

【实验目的】

- 1、 熟悉 Dev-Cpp 编程环境。
- 2、 编写简单的输入输出语句。
- 3、 熟练使用算术运算符。
- 4、 能够编写简单的判断语句。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

编写 C++ 程序，实现输入两个整数，输出两个整数的加、减、乘、除结果；详细的注释，完整的输出显示。

【详细分析】

实验内容是对两个输入整数的四则运算，即简单的顺序结构。

在此基础上稍作改动，即输入整数和运算符来自动进行计算，流程如图1。

【实验源码】

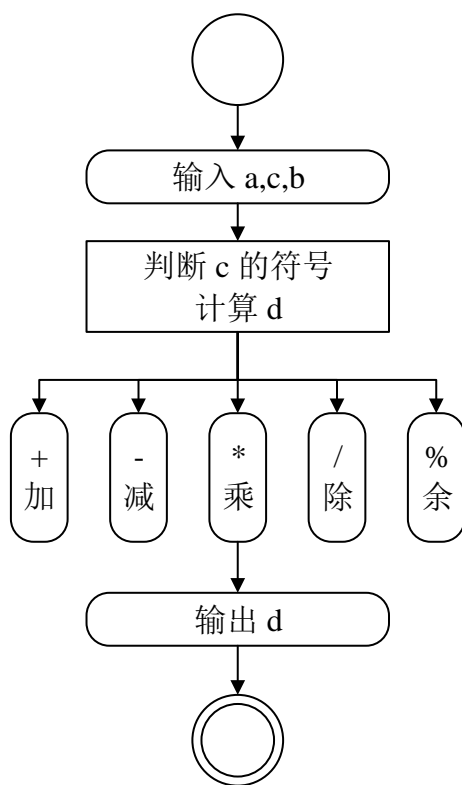


图 1: 程序流程图

Listing 1: exp01.cpp

```

1  #include <iostream>
2  /* 输入输出流 */
3  using namespace std;
4  /* std 命名空间 */
5  int main()
6  /* 主函数 */
7  {
8      int a, b;
9      /* 声明两个整数 */
10     char c;
11     /* 声明一个字符 */
12     cin >> a >> c >> b;
13     /* 从屏幕输入流读入 */
14     int d = 0;
15     /* 初始化结果 */
16     switch(c) {
17     /* 分支判断运算符 */
18         case '+':
19             d = a + b;
20             break;
21             /* break 是 C 遗留的冗余语句 */
22         case '-':
23             d = a - b;
24             break;
25         case '*':
26             d = a * b;
27             break;
28         case '/':
29             d = a / b;
30             break;
31         case '%':
32             d = a % b;
33             break;
34         /* 不需要 default */
35     }
36     cout << d << endl;
37     /* 输出结果并换行 */
38     return 0;
39     /* 正常结束返回 0 */
40 }

```

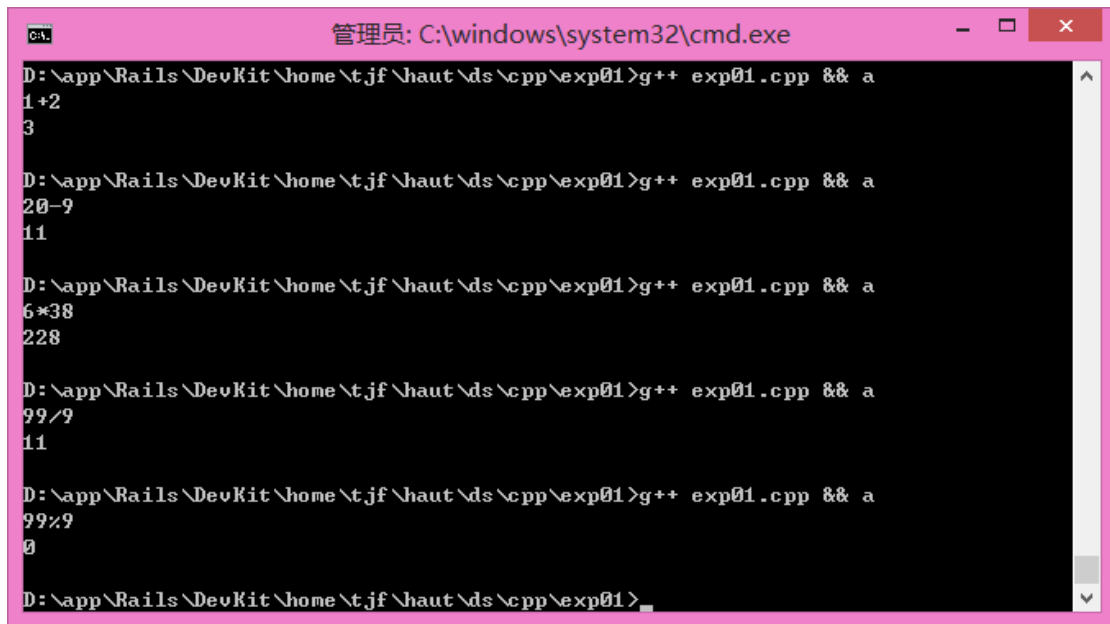
【实验结果】

图2显示了编译、运行、输入、输出的过程。

【实验体会】

这是一个非常基础的简单程序，目的在于熟悉编程和调试环境。程序本身没有任何难度，加上注释也不过40行。

关于编程环境的配置，我倾向于使用编辑器（如 Vim、Sublime Text）编写



```
管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp01>g++ exp01.cpp && a
1+2
3
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp01>g++ exp01.cpp && a
20-9
11
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp01>g++ exp01.cpp && a
6*38
228
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp01>g++ exp01.cpp && a
99/9
11
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp01>g++ exp01.cpp && a
99%9
0
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp01>
```

图 2: 标准控制台输入输出

源文件，在命令行下编译、运行和调试（使用 gcc/gdb）。当然，Microsoft Visual Studio 作为世界上最好的 IDE，在编写调试程序中也是非常好用的，所以在有条件的时候也会使用 VS。

实验二 类和对象

实验时间： 2014 年 11 月 29 日

【实验目的】

- 1、 掌握类、对象、数据成员、成员函数的基本概念。
- 2、 能够进行类的定义。
- 3、 能够使用成员函数进行相关调用。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

- 1、 编写NumberA类，实现两个整数的加减乘除运算。构造函数实现两整数a，b赋值。
- 2、 编写OperaN类，实现输入1.2.3.4解析成加减乘除符号。
- 3、 P89: 3.11

2.1 NumberA 类

【详细分析】

NumberA 类设有两个成员变量存放两个操作数，提供对这两个操作数进行四则运算的方法。

【实验源码】

Listing 2: exp01.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  class NumberA
5  {
6  private:
7      int a, b;
8
9  public:
10     NumberA(int _a, int _b) : a(_a), b(_b) {}
```

```

11     int plus() { return a + b; }
12     int minus() { return a - b; }
13     int times() { return a * b; }
14     int divide() { return a / b; }
15     int mod() { return a % b; }
16 };
17
18 int main()
19 {
20     int a, b;
21     cin >> a >> b;
22     NumberA t(a, b);
23     cout << t.plus() << endl;
24     cout << t.minus() << endl;
25     cout << t.times() << endl;
26     cout << t.divide() << endl;
27     cout << t.mod() << endl;
28     return 0;
29 }

```

【实验结果】

```

管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp01.cpp && a
9 4
13
5
36
2
1

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp01.cpp && a
12 7
19
5
84
1
5

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>

```

图 3: NumberA 类

2.2 OperaN 类

【详细分析】

用一个整数初始化类，置符号。

【实验源码】

Listing 3: exp02.cpp

```

1 #include <iostream>

```

```

2  #include <string>
3  using namespace std;
4
5  class OperaN
6  {
7  private:
8      int num;
9
10 public:
11     char mark;
12     OperaN(int _x = 1) : num(_x) { getMark(); }
13     char getMark()
14     {
15         static string marks = "+-*/";
16         return mark = marks[num % 4];
17     }
18 };
19
20 int main()
21 {
22     int n;
23     cin >> n;
24     OperaN t(n);
25     cout << t.getMark() << endl;
26     cout << t.mark << endl;
27     return 0;
28 }

```

【实验结果】

```

管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp02.cpp -Wall && a
2
*
*

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp02.cpp -Wall && a
3
/
/

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp02.cpp -Wall && a
4
+
+

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp02.cpp -Wall && a
1
-
-

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ exp02.cpp -Wall && a
5
-
-

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>

```

图 4: OperaN 类

2.3 P89: 3.11

【详细分析】

对 Account 类的修改。

【实验源码】

Listing 4: GradeBook.h

```
1  /* GradeBook.h */
2  #include <string>
3  using std::string;
4
5  class GradeBook
6  {
7  public:
8      GradeBook(string, string);
9      void setCourseName(string);
10     string getCourseName();
11     void setTeacherName(string);
12     string getTeacherName();
13     void displayMessage();
14
15 private:
16     string courseName;
17     string teacherName;
18 };
```

Listing 5: GradeBook.cpp

```
1  /* GradeBook.cpp */
2  #include <iostream>
3  using std::cout;
4  using std::endl;
5
6  #include "GradeBook.h"
7
8  GradeBook::GradeBook(string name, string teacher)
9  {
10     setCourseName(name);
11     setTeacherName(teacher);
12 }
13
14 void GradeBook::setCourseName(string name)
15 {
16     if (name.length() <= 25)
17         courseName = name;
18     if (name.length() > 25) {
19         courseName = name.substr(0, 25);
20         cout << "名称\" << name << "\" 长度超限 (25) 。\n"
21             << "截取前 25 个字符。\" << endl;
22     }
23 }
```



```

24
25 string GradeBook::getCourseName()
26 {
27     return courseName;
28 }
29
30 void GradeBook::setTeacherName(string name)
31 {
32     if (name.length() <= 25)
33         teacherName = name;
34     if (name.length() > 25) {
35         teacherName = name.substr(0, 25);
36         cout << "姓名\" << name << "\" 长度超限 (25) 。\n"
37             << "截取前 25 个字符。\\n" << endl;
38     }
39 }
40
41 string GradeBook::getTeacherName()
42 {
43     return teacherName;
44 }
45
46 void GradeBook::displayMessage()
47 {
48     cout << "欢迎使用 " << getCourseName() << " 课程表! \n"
49         << "任课教师: " << getTeacherName() << endl;
50 }

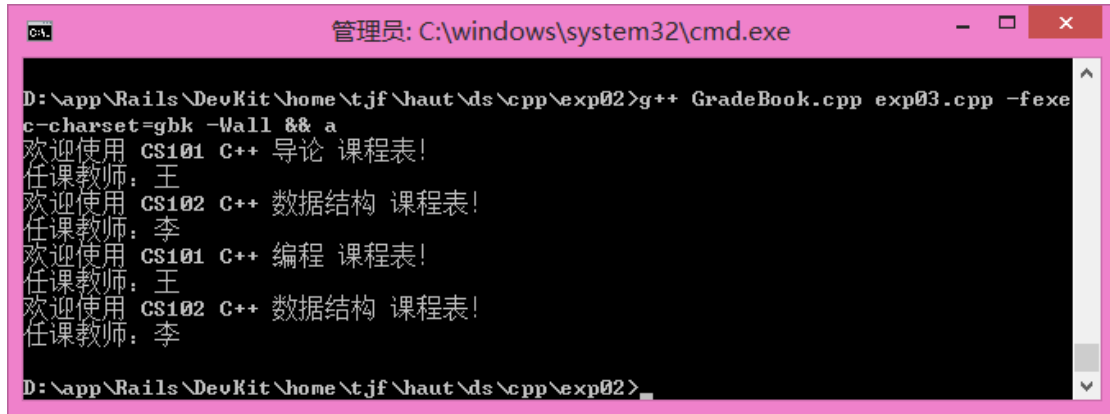
```

Listing 6: 主程序 exp03.cpp

```

1  #include <iostream>
2  using std::cout;
3  using std::endl;
4
5  #include "GradeBook.h"
6
7  int main()
8  {
9      GradeBook gradeBook1("CS101 C++ 导论", "王");
10     GradeBook gradeBook2("CS102 C++ 数据结构", "李");
11
12     gradeBook1.displayMessage();
13     gradeBook2.displayMessage();
14
15     gradeBook1.setCourseName("CS101 C++ 编程");
16
17     gradeBook1.displayMessage();
18     gradeBook2.displayMessage();
19     return 0;
20 }

```



```
管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>g++ GradeBook.cpp exp03.cpp -fexe
c-charset=gbk -Wall && a
欢迎使用 CS101 C++ 导论 课程表!
任课教师: 王
欢迎使用 CS102 C++ 数据结构 课程表!
任课教师: 李
欢迎使用 CS101 C++ 编程 课程表!
任课教师: 王
欢迎使用 CS102 C++ 数据结构 课程表!
任课教师: 李
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp02>
```

图 5: P89: 3.11

【实验结果】

【实验体会】

自己写的代码一般都比较简洁，也不会用特别复杂的变量名和函数名，所以前两个程序是比较简单的——而且没有注释。我觉得这个程序的理解难度还没有必要写注释。最后一个题目来自课本，是对示例中的 GradeBook 类进行修改。

这几个程序的主要目的在于对 C++ 类的熟悉，也是 OO 的基础。

实验三 结构化控制结构

实验时间： 2015 年 1 月 8 日

【实验目的】

- 1、 掌握基本的结构化控制结构。
- 2、 能够熟练进行结构化编程。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

- 1、 编写NumberA类，实现两个整数的加减乘除运算，可以循环计算。构造函数实现两整数a，b赋值。
- 2、 P177： 5.29

3.1 NumberA 类

【详细分析】

NumberA 类设有两个成员变量存放两个操作数，提供对这两个操作数进行四则运算的方法。

主程序循环读入两个整数，进行运算并输出。

【实验源码】

Listing 7: exp01.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  class NumberA
5  {
6  private:
7      int a, b;
8
9  public:
10     NumberA(int _a, int _b) : a(_a), b(_b) {}
11     int plus() { return a + b; }
12     int minus() { return a - b; }
```

```

13     int times() { return a * b; }
14     int divide() { return a / b; }
15     int mod() { return a % b; }
16 };
17
18 int main()
19 {
20     int a, b;
21     while (cin >> a >> b) {
22         NumberA t(a, b);
23         cout << a << '+' << b << '=' << t.plus() << endl;
24         cout << a << '-' << b << '=' << t.minus() << endl;
25         cout << a << '*' << b << '=' << t.times() << endl;
26         cout << a << '/' << b << '=' << t.divide() << endl;
27         cout << a << '%' << b << '=' << t.mod() << endl;
28     }
29     return 0;
30 }

```

【实验结果】

```

管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp03>g++ exp01.cpp -Wall && a
1 2
1+2=3
1-2=-1
1*2=2
1/2=0
1%2=1
5 3
5+3=8
5-3=2
5*3=15
5/3=1
5%3=2
9 14
9+14=23
9-14=-5
9*14=126
9/14=0
9%14=9
^Z
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp03>

```

图 6: NumberA 类

3.2 Peter Minuit 问题

【详细分析】

复利的计算，基于课本的示例程序修改。通过在年份循环之外加上利率的循环，计算出不同利率下的本息变化。

【实验源码】

Listing 8: exp03.cpp

```
1  // 使用 for 语句计算复利
2  #include <iostream>
3  using std::cout;
4  using std::endl;
5  using std::fixed;
6
7  #include <iomanip>
8  using std::setw; // 允许程序设置列宽
9  using std::setprecision;
10
11 #include <cmath> // 标准 C++ 数学库
12 using std::pow; // 允许程序使用幂函数
13
14 int main()
15 {
16     double amount; // 每年结息后的本息合计
17     double principal = 24.0; // 初始化本金
18     double rate = .05; // 利率
19
20     // 显示表头
21     cout << setw(6) << "利率" << setw(4) << "年" << setw(24) << "合计" << endl;
22
23     // 设置小数位
24     cout << fixed << setprecision(2);
25
26     int start = 1626; // 开始年份
27     int now = 2015; // 今年
28
29     // 循环利率
30     for (rate = .05; rate <= .10; rate += .01) {
31         // 计算年间的本息合计
32         cout << setw(5) << rate * 100 << '%' << endl;
33         for (int year = start; year <= now; year++) {
34             // 计算当年本息
35             amount = principal * pow(1.0 + rate, year - start);
36             // 显示当年合计
37             cout << setw(10) << year << setw(24) << amount << endl;
38         } // end for
39         // 显示合计
40         cout << setw(5) << rate * 100 << '%' << setw(28) << amount << endl;
41     } // end for
42
43     return 0; // 程序成功结束
44 } // end main
```

【实验结果】

由于程序输出非常多，图7 只给出了程序的前后十行的输出。

```
管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp03>g++ exp02.cpp -Wall -fexec-charset=gbk && a | head
利率 年 合计
5.00%
1626 24.00
1627 25.20
1628 26.46
1629 27.78
1630 29.17
1631 30.63
1632 32.16
1633 33.77

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp03>g++ exp02.cpp -Wall -fexec-charset=gbk && a | tail
2007 141522107394756192.00
2008 155674318134231808.00
2009 171241749947655008.00
2010 188365924942420544.00
2011 207202517436662592.00
2012 227922769180328896.00
2013 250715046098361792.00
2014 275786550708197984.00
2015 303365205779017792.00
10.00% 303365205779017792.00

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp03>
```

图 7: Peter Minuit 问题

【实验体会】

这次实验主要是对控制语句的应用。C++ 的流程控制语句与 C 一脉相承，并没有太大差别。对于循环语句，三大循环语句其实是可以互相转换的，虽然如此，也要视场合选用合适的语句，不能因为 for 语句的强大而滥用，导致代码变得难以理解。分支语句也是以 if/else if/else 为主的。总之基本流程就是这么多了，搞清楚其中的逻辑关系和嵌套、作用域就好。

河南工业大学

《面向对象程序设计》实验报告

专业班级： 软件 1305 班 学号： 201316920311 姓名： 田劲锋

实验单元二 运算符重载

实验四 创建 Date 类

实验时间： 2015 年 1 月 8 日

【实验目的】

- 1、 掌握创建类的方法。
- 2、 熟悉成员函数的使用方法。
- 3、 掌握函数和指针的概念
- 4、 掌握函数和指针的使用方法。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

- 1、 P89: 3.15
- 2、 P348: 8.12
- 3、 P354: 8.20

4.1 Date 类

【详细分析】

创建一个名为 Date 的类，包括了作为数据成员的三部分信息：年月日，都为 int 类型。包括一个具有三个参数的构造函数，用以初始化年月日。假定给出

的年、日是正确的，对于不在 1–12 的月，默认设置为 1。对每个数据成员都提供 set/get 函数。提供 displayDate 功能显示格式化后的日期。

【实验源码】

Listing 9: Date.h

```
1  // Date.h
2  class Date
3  {
4  private:
5      int year;
6      int month;
7      int day;
8
9  public:
10     Date(int, int, int);
11     void displayDate();
12
13     void setYear(int y) { year = y; };
14     void setMonth(int); // 需要特殊处理
15     void setDay(int d) { day = d; };
16
17     int getYear() const { return year; };
18     int getMonth() const { return month; };
19     int getDay() const { return day; };
20 };
```

Listing 10: Date.cpp

```
1  // Date.cpp
2  #include <iostream>
3  using std::cout;
4  using std::endl;
5
6  #include "Date.h"
7
8  Date::Date(int y, int m, int d)
9  {
10     setYear(y);
11     setMonth(m);
12     setDay(d);
13 }
14
15 void Date::setMonth(int m)
16 {
17     if (m < 1 || m > 12) {
18         m = 1;
19     }
20     month = m;
21 }
22
23 void Date::displayDate()
24 {
```



```

25     cout << year << '/' << month << '/' << day << endl;
26 }

```

Listing 11: exp01.cpp

```

1  #include <iostream>
2  using namespace std;
3
4  #include "Date.h"
5
6  int main()
7  {
8      Date t(2014, 12, 25);
9      t.displayDate();
10
11     t.setYear(2015);
12     cout << "改年为" << 2015 << ": " << t.getYear() << endl;
13     t.displayDate();
14
15     t.setMonth(2);
16     cout << "改月为" << 2 << ": " << t.getMonth() << endl;
17     t.displayDate();
18
19     t.setMonth(35); // 这里会自动设置为 1
20     cout << "改月为" << 35 << ": " << t.getMonth() << endl;
21     t.displayDate();
22
23     t.setDay(18);
24     cout << "改日为" << 18 << ": " << t.getDay() << endl;
25     t.displayDate();
26
27     return 0;
28 }

```

【实验结果】

```

管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>g++ Date.cpp exp01.cpp -Wall -fex ^
ec-charset=gbk && a
2014/12/25
改年为2015: 2015
2015/12/25
改月为2: 2
2015/2/25
改月为35: 1
2015/1/25
改日为18: 18
2015/1/18
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>

```

图 8: Date 类

4.2 P348: 8.12

【详细分析】

修改课本上的程序，使发牌函数发一手 5 张牌，完成任务：

- a) 确定手上是否有一对牌
- b) 确定手上是否有两对牌
- c) 确定手上是否有 3 张同号牌
- d) 确定手上是否有 4 张同号牌
- e) 确定手上是否有同花
- f) 确定手上是否有顺子

程序对花色和点数分别进行累计，然后循环数出数量。

【实验源码】

Listing 12: DeckOfCards.h

```
1 // DeckOfCards.h
2 #include <string>
3 using std::string;
4
5 // DeckOfCards 类定义
6 class DeckOfCards
7 {
8 public:
9     DeckOfCards(); // 初始化
10    void shuffle(); // 洗牌
11    void deal(); // 发牌
12    void shuffleAndDeal(); // 用同一个函数实现
13
14    void faPai(int); // 发牌到手上
15    string showCard(const int, const int) const; // 显示牌面
16    void showHand(); // 显示手牌
17
18    int hasDuiZi(); // 判断对子数量
19    // 约定传入参数为数量，返回值为最小号码或花色的值
20    int hasTongHao(int); // 判断同号
21    int hasTongHua(int); // 判断同花
22    int hasShunzi(int); // 判断顺子
23
24 private:
25     int deck[4][13]; // 存放牌元素
26     int n; // 手中牌数
27     int hana[52], suzi[52]; // 存放手中的牌
28 }; // end class DeckOfCards
```

Listing 13: DeckOfCards.cpp

```

1  // DeckOfCards.cpp
2  #include <iostream>
3  using std::cout;
4  using std::endl;
5  using std::left;
6  using std::right;
7
8  #include <iomanip>
9  using std::setw;
10
11 #include <string>
12 using std::string;
13
14 #include <cstdlib> // 随机函数原型
15 using std::rand;
16 using std::srand;
17
18 #include <ctime> // 时间函数原型
19 using std::time;
20
21 #include "DeckOfCards.h" // DeckOfCards 类定义
22
23 // DeckOfCards 默认初始化
24 DeckOfCards::DeckOfCards()
25 {
26     // 循环行
27     for (int row = 0; row <= 3; row++) {
28         // 循环列
29         for (int column = 0; column <= 12; column++) {
30             deck[row][column] = 0; // 置 0
31         } // end 内层 for
32     } // end 外层 for
33
34     srand(time(0)); // 初始化随机数种子
35 } // end DeckOfCards
36
37 // 洗牌
38 void DeckOfCards::shuffle()
39 {
40     int row; // 表示花色
41     int column; // 表示数字
42
43     // 对这 52 张牌进行随机排列
44     for (int card = 1; card <= 52; card++) {
45         do { // 随机选一张牌
46             row = rand() % 4; // 随机花色
47             column = rand() % 13; // 随机数字
48         } while (deck[row][column] != 0); // 直到找到一张没被选过的牌
49
50         // 标记牌已经选中
51         deck[row][column] = card;
52     } // end for
53 } // end 函数 shuffle
54
55 // 显示牌

```

```

56 string DeckOfCards::showCard(const int row, const int column) const
57 {
58     // 初始化花色数组
59     static string suit[4] = { "红桃", "方块", "黑桃", "梅花" };
60
61     // 初始化数字数组
62     static string face[13] = {
63         " A", " 2", " 3", " 4", " 5", " 6", " 7",
64         " 8", " 9", "10", " J", " Q", " K" };
65
66     return suit[row] + face[column];
67 }
68
69 // 发牌
70 void DeckOfCards::deal()
71 {
72     // 对 52 张牌中的每张牌
73     for (int card = 1; card <= 52; card++) {
74         // 循环花色
75         for (int row = 0; row <= 3; row++) {
76             // 循环数字
77             for (int column = 0; column <= 12; column++) {
78                 // 如果该元素包含需要的牌号
79                 if (deck[row][column] == card) {
80                     cout << showCard(row, column)
81                         << (card % 2 == 0 ? '\n' : '\t');
82                 } // end if
83             } // end 最内层 for
84         } // end 内层 for
85     } // end 外层 for
86 } // end 函数 deal
87
88 void DeckOfCards::faPai(int m)
89 {
90     if (m < 1 || m > 52) {
91         m = 1;
92     }
93     n = m;
94     for (int card = 0; card < n; card++) {
95         for (int row = 0; row < 4; row++) {
96             for (int column = 0; column < 13; column++) {
97                 if (deck[row][column] == card + 1) {
98                     hana[card] = row;
99                     suzi[card] = column;
100                 }
101             }
102         }
103     }
104 }
105
106 void DeckOfCards::showHand()
107 {
108     cout << "手上有 " << n << " 张牌: " << endl;
109     for (int i = 0; i < n; i++) {
110         cout << setw(9) << showCard(hana[i], suzi[i]) << endl;

```

```

111     }
112 }
113
114 // 判断同号
115 int DeckOfCards::hasTongHao(int m)
116 {
117     int num[13] = { 0 };
118     for (int i = 0; i < n; i++) {
119         num[suzi[i]]++;
120     }
121     for (int i = 0; i < 13; i++) {
122         if (num[i] >= m) {
123             return i;
124         }
125     }
126     return -1;
127 }
128
129 // 判断对子数量
130 int DeckOfCards::hasDuizi()
131 {
132     int num[13] = { 0 };
133     for (int i = 0; i < n; i++) {
134         num[suzi[i]]++;
135     }
136     int sum = 0;
137     for (int i = 0; i < 13; i++) {
138         if (num[i] >= 2) {
139             sum++;
140         }
141     }
142     return sum;
143 }
144
145 // 判断同花
146 int DeckOfCards::hasTongHua(int m)
147 {
148     int flower[4] = { 0 };
149     for (int i = 0; i < n; i++) {
150         flower[hana[i]]++;
151     }
152     for (int i = 0; i < 4; i++) {
153         if (flower[i] >= m) {
154             return i;
155         }
156     }
157     return -1;
158 }
159
160 // 判断顺子
161 int DeckOfCards::hasShunzi(int m)
162 {
163     int num[13] = { 0 };
164     for (int i = 0; i < n; i++) {
165         num[suzi[i]]++;

```

```

166     }
167     for (int i = 0; i < 13; i++) {
168         int j = i;
169         for (; j < 13; j++) {
170             if (num[j] <= 0) {
171                 break;
172             }
173         }
174         if (j - i + 1 >= m) {
175             return i;
176         }
177     }
178     return -1;
179 }

```

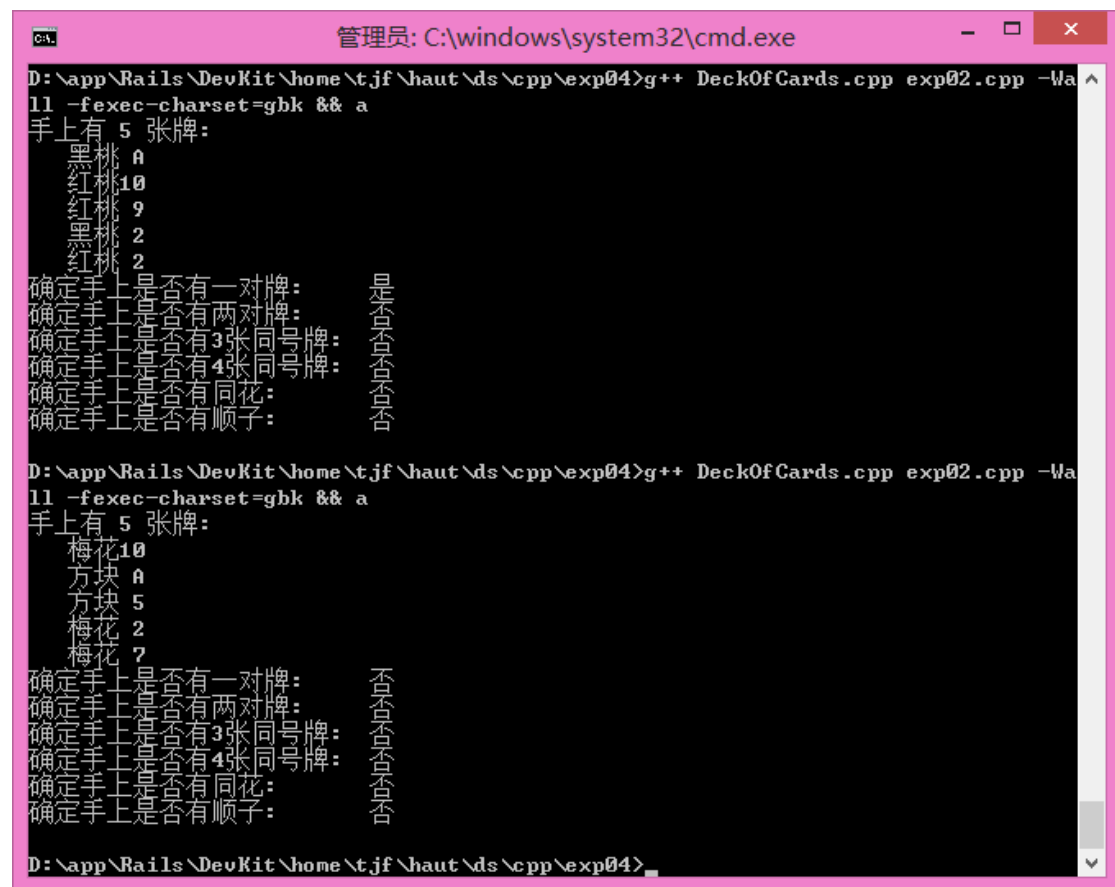
Listing 14: exp02.cpp

```

1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  #include "DeckOfCards.h"
6
7  int main()
8  {
9      DeckOfCards d;
10
11      d.shuffle();
12      d.faPai(5);
13      d.showHand();
14
15      cout << setw(26) << left << "确定手上是否有一对牌:";
16      int a = d.hasDuizi();
17      cout << (a >= 1 ? "是" : "否") << endl;
18
19      cout << setw(26) << left << "确定手上是否有两对牌:";
20      cout << (a >= 2 ? "是" : "否") << endl;
21
22      cout << setw(26) << left << "确定手上是否有 3 张同号牌:";
23      int b = d.hasTongHao(3);
24      cout << (b >= 0 ? "是" : "否") << endl;
25
26      cout << setw(26) << left << "确定手上是否有 4 张同号牌:";
27      int c = d.hasTongHao(4);
28      cout << (c >= 0 ? "是" : "否") << endl;
29
30      cout << setw(26) << left << "确定手上是否有同花:";
31      int e = d.hasTongHua(5);
32      cout << (e >= 0 ? "是" : "否") << endl;
33
34      cout << setw(26) << left << "确定手上是否有顺子:";
35      int f = d.hasShunzi(5);
36      cout << (f >= 0 ? "是" : "否") << endl;
37
38      return 0;
39 }

```

【实验结果】



```
管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>g++ DeckOfCards.cpp exp02.cpp -Wa ^
ll -fexec-charset=gbk && a
手上有 5 张牌:
黑桃 A
红桃 10
红桃 9
黑桃 2
红桃 2
确定手上一是否有一对牌: 是
确定手上一是否有两对牌: 否
确定手上一是否有三张同号牌: 否
确定手上一是否有四张同号牌: 否
确定手上一是否有同花: 否
确定手上一是否有顺子: 否

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>g++ DeckOfCards.cpp exp02.cpp -Wa ^
ll -fexec-charset=gbk && a
手上有 5 张牌:
梅花 10
方块 A
方块 5
梅花 2
梅花 7
确定手上一是否有一对牌: 否
确定手上一是否有两对牌: 否
确定手上一是否有三张同号牌: 否
确定手上一是否有四张同号牌: 否
确定手上一是否有同花: 否
确定手上一是否有顺子: 否

D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>
```

图 9: 发牌和判断程序

4.3 P354: 8.20

【详细分析】

修改洗牌和发牌程序，使之由同一个函数实现。

【实验源码】

Listing 15: DeckOfCards_plus.cpp

```
1 // DeckOfCards_plus.cpp
2 #include <iostream>
3 using std::cout;
4
5 #include <cstdlib>
6 using std::rand;
7
8 #include "DeckOfCards.h"
9
10 // 洗牌和发牌
```

```

11 void DeckOfCards::shuffleAndDeal()
12 {
13     int row;
14     int column;
15
16     for (int card = 1; card <= 52; card++) {
17         do {
18             row = rand() % 4; // 随机花色
19             column = rand() % 13; // 随机数字
20         } while (deck[row][column] != 0);
21
22         deck[row][column] = card;
23         cout << showCard(row, column)
24              << (card % 2 == 0 ? '\n' : '\t');
25     }
26 }

```

Listing 16: exp03.cpp

```

1 #include "DeckOfCards.h"
2
3 int main()
4 {
5     DeckOfCards d;
6
7     d.shuffleAndDeal();
8
9     return 0;
10 }

```

【实验结果】

【实验体会】

这次的实验是 Date 类的简单实现，以及对 DeckOfCards 类的修改和增添功能。

对于初始化函数，也就是构造函数，可能会用冒号句法来简化成员对象的赋值。这里因为涉及到特殊判断，所以调用了三个设置函数，而对设置值的特殊处理则分别在设置函数中实现。访问函数有 const 属性，直接返回其值，并不改变值的大小。

判断发牌的花色和点数，则使用了循环查找的暴力算法，由于花色 $n = 4$ ，点数 $m = 13$ ，那么 $O(n)$ 和 $O(m)$ 的算法实际上可以算作常数级别的了。为了简化查找，增设了两个数组来存储花色和点数。函数具有一定的通用性。

最后一个程序删掉了本来的注释，程序显得简练很多了。


```
管理员: C:\windows\system32\cmd.exe
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>g++ DeckOfCards.cpp DeckOfCards_p
lus.cpp exp03.cpp -Wall -fexec-charset=gbk && a
红桃 2 梅花 3
梅花 K 梅花 10
红桃 8 方块 J
梅花 Q 黑桃 3
黑桃 8 红桃 10
方块 8 红桃 K
红桃 3 梅花 J
黑桃 5 梅花 6
梅花 A 方块 A
黑桃 2 方块 2
方块 9 红桃 5
方块 4 红桃 7
红桃 A 红桃 Q
梅花 5 红桃 J
方块 3 红桃 9
梅花 2 黑桃 K
黑桃 9 梅花 9
方块 5 黑桃 10
方块 Q 黑桃 A
红桃 4 黑桃 J
方块 7 梅花 8
方块 K 梅花 4
黑桃 4 方块 6
梅花 7 黑桃 Q
黑桃 7 红桃 6
黑桃 6 方块 10
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>
```

图 10: 发牌程序

实验五 类 Date 的属性

实验时间： 2015 年 1 月 9 日

【实验目的】

- 1、 掌握重载的概念
- 2、 能够进行运算符重载。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

日期类设计

定义Date类，参照实现：

- (1) 日期的加、减运算
- (2) 根据日期计算一年中的第几周星期几、一年中第几天为几月几日、该年是否为闰年
- (3) 输出日期对象

完成相应应用程序设计

【详细分析】

（此项由学生自己完成）

【实验源码】

Listing 17: Date.h

```
1 // Date.h
2 #include <string>
3
4 class Date
5 {
6 private:
7     int year;
8     int month;
9     int day;
10
```

```

11     const static int MONTHS[2][13];
12     const static std::string WEEKS[8];
13
14 public:
15     Date(int, int, int);
16     Date(int, int);
17     void displayDate();
18
19     void setYear(int);
20     void setMonth(int);
21     void setDay(int);
22
23     int getYear() const { return year; };
24     int getMonth() const { return month; };
25     int getDay() const { return day; };
26
27     Date& operator+(const int);
28     int operator-(const Date&);
29     Date& operator-(const int);
30
31     int getWeekday() const;
32     int getWeek() const
33
34     bool isLeap() const;
35
36     int getDays() const;
37     void setDays(int d);
38
39     static std::string weekday2s(int weekday);
40     static bool yearIsLeap(int year);
41 };

```

Listing 18: Date.cpp

```

1  // Date.cpp
2  #include <iostream>
3  using std::cout;
4  using std::endl;
5
6  #include <string>
7
8  #include <algorithm>
9  using std::min;
10 using std::max;
11
12 #include "Date.h"
13
14 const int Date::MONTHS[2][13] = {
15     { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
16     { 0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
17 };
18 const std::string Date::WEEKS[8] = { "星期日",
19     "星期一", "星期二", "星期三",
20     "星期四", "星期五", "星期六"
21 };
22

```

```

23 Date::Date(int y, int m, int d)
24 {
25     setYear(y);
26     setMonth(m);
27     setDay(d);
28 }
29
30 Date::Date(int year, int days)
31 {
32     setYear(year);
33     setDays(days);
34 }
35
36 void Date::setYear(int year)
37 {
38     this->year = year;
39 }
40
41 void Date::setMonth(int m)
42 {
43     if (m < 1 || m > 12) {
44         m = 1;
45     }
46     this->month = m;
47 }
48
49 void Date::setDay(int day)
50 {
51     if (day < 1 || day > Date::MONTHS[isLeap()][this->month]) {
52         day = 1;
53     }
54     this->day = day;
55 }
56
57 void Date::displayDate()
58 {
59     cout << year << "年" << month << "月" << day << "日";
60 }
61
62 Date& Date::operator+(const int day)
63 {
64     return Date(this->year, this->getDays() + day);
65 }
66
67 int Date::operator-(const Date& rhs)
68 {
69     int d = 0;
70     for (int i = rhs.year; i < this->year; i++) {
71         d += 365 + (int) yearIsLeap(i);
72     }
73     d += this->getDays();
74     return d;
75 }
76
77 Date& Date::operator-(const int day)

```

```

78 {
79     return Date(this->year, this->getDays() - day);
80 }
81
82 int Date::getWeekday() const
83 {
84     int y = this->year;
85     int m = this->month;
86     int d = this->day;
87     if (m < 3) {
88         m += 12;
89         y -= 1;
90     }
91     int week = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7;
92     return (week + 1) % 7;
93 }
94
95 int Date::getWeek() const
96 {
97     return this->getDays() / 7;
98 }
99
100 bool Date::isLeap() const
101 {
102     return Date::yearIsLeap(this->year);
103 }
104
105 bool Date::yearIsLeap(int year)
106 {
107     return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
108 }
109
110 void Date::setDays(int d)
111 {
112     int y = 365;
113     while (d > y) {
114         d -= y + (int) yearIsLeap(++this->year);
115     }
116     while (d < 0) {
117         d += y + (int) yearIsLeap(--this->year);
118     }
119
120     int i;
121     for (i = 0; i <= 12; i++) {
122         if (d <= Date::MONTHS[isLeap()][i]) {
123             break;
124         }
125         d -= Date::MONTHS[isLeap()][i];
126     }
127     this->setMonth(i);
128     this->setDay(d);
129 }
130
131 int Date::getDays() const
132 {

```

```

133     int d = this->day;
134     for (int i = 1; i < this->month; i++) {
135         d += Date::MONTHS[isLeap()][i];
136     }
137     return d;
138 }
139
140 std::string Date::weekday2s(int weekday)
141 {
142     return Date::WEEKS[weekday % 7];
143 }

```

Listing 19: exp01.cpp

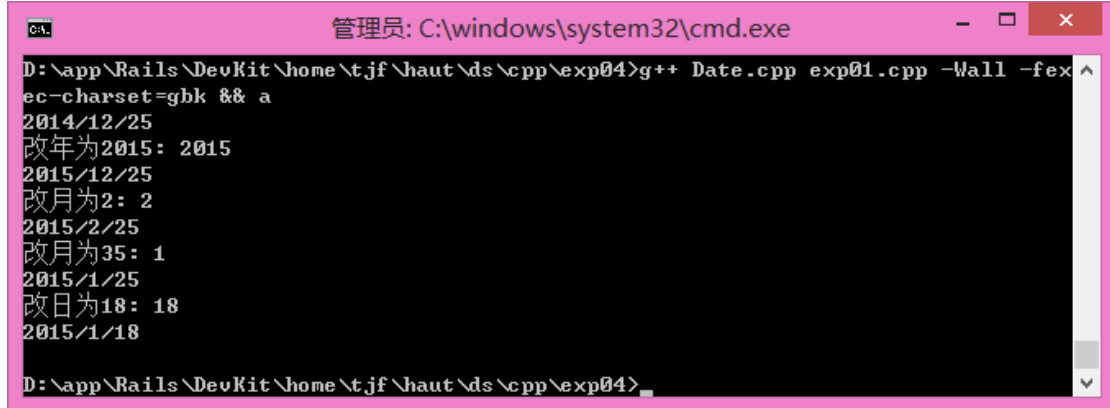
```

1  #include <iostream>
2  using namespace std;
3
4  #include "Date.h"
5
6  int main()
7  {
8      Date t(2012, 12, 25);
9      t.displayDate();
10     cout << Date::weekday2s(t.getWeekday()) << endl;
11     cout << (t.isLeap() ? "闰年" : "平年") << endl;
12
13     Date s(2015, 1, 10);
14     s.displayDate();
15     cout << Date::weekday2s(s.getWeekday()) << endl;
16     cout << (s.isLeap() ? "闰年" : "平年") << endl;
17
18     cout << "相差 " << s - t << " 天" << endl;
19
20     Date a = s - 40;
21     s.displayDate();
22     cout << " 减去 40 天 = ";
23     a.displayDate();
24     cout << endl;
25
26     Date b = t + 20;
27     t.displayDate();
28     cout << " 加上 20 天 = ";
29     b.displayDate();
30     cout << endl;
31
32     Date c(2015, -1);
33     cout << "2015 年的第 -1 天 ";
34     c.displayDate();
35     cout << endl;
36
37     Date d(2015, 128);
38     cout << "2015 年的第 128 天 ";
39     d.displayDate();
40     cout << endl;
41
42     getchar();

```

```
43  
44     return 0;  
45 }
```

【实验结果】



```
管理员: C:\windows\system32\cmd.exe  
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>g++ Date.cpp exp01.cpp -Wall -fexceptions -charset=gbk && a  
2014/12/25  
改年为2015: 2015  
2015/12/25  
改月为2: 2  
2015/2/25  
改月为35: 1  
2015/1/25  
改日为18: 18  
2015/1/18  
D:\app\Rails\DevKit\home\tjf\haut\ds\cpp\exp04>
```

图 11: Date 类

【实验体会】

(至少150字)

河南工业大学

《面向对象程序设计》实验报告

专业班级： 软件 1305 班 学号： 201316920311 姓名： 田劲锋

实验单元三 继承

实验六 创建 Account 继承层次

实验时间： 2015 年 1 月 7 日

【实验目的】

- 1、 掌握继承的概念。
- 2、 掌握不同继承方式的继承特性。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

P522: 12.10

【详细分析】

（此项由学生自己完成）

【实验源码】

（此项由学生自己完成）

【实验结果】

（截图给出实验结果）

【实验体会】

（至少150字）

河南工业大学

《面向对象程序设计》实验报告

专业班级： 软件 1305 班 学号： 201316920311 姓名： 田劲锋

实验单元四 多态性和虚函数

实验七 创建 AHAPE 抽象类

实验时间： 2015 年 1 月 7 日

【实验目的】

- 1、 掌握多态性的概念。
- 2、 掌握虚函数概念及其与多态性的关系。

【实验环境】

gcc version 4.9.2 (tdm64-1)

【实验内容】

功能要求：定义一个抽象类SHAPE，抽象方法SHAPE包含X和Y两个属性的访问方法，VOLUME 方法，AREA抽象方法和GETNAME方法。不同的形状类，如POINT 类实现SHAPE 类，RECTANGLE类继承PIONT ， ELLIPSE类继承RECTANGLE 类。CIRCLE 类继承ELLIPSE 类，CYLINDER类继承CIRCLE类。创建每个类的实例，并将每个类的实例存放于类型为SHAPE的数组中。以该SHAPE的数组作为参数，调用参数的类型为SHAPE 的数组的SHOWSHAPINFO方法，通过调用重写的方法为相应得图形对象计算表面积，体积并输出图形的名称

【详细分析】

（此项由学生自己完成）

【实验源码】

（此项由学生自己完成）

【实验结果】

（截图给出实验结果）

【实验体会】

（至少150字）