题10.7

**SavingAccount Class**

题目要求：

创建一个SavingAccount类，使用静态数据成员annualInterestRate来存储利息的年利率。每一个对象包含一个私有数据成员savingBalance来表示saver目前的存款，提供成员函数calculateMonthlyInterest()来计算每一个月的利息，并且将利息加入到savingBalance中。提供一个静态成员函数modifyInterestRate()来改变静态数据成员annualInterestRate的值。

注：月利息计算公式：MonthlyInterest=savingBalance\*annualInterestRate/12；

解题思路：

在类中申明annualInterestRate为静态存储类型，并在类外进行定义与初始化。通过构造函数来初始化savingBalance（同时提供默认构造函数），通过calculateMonthlyInterest()函数计算月利息，将其加入到savingBalance中，并打印savingBalance，通过modifyInterestRate()函数来更改annualInterestRate的值（static函数才可调用static数据），再进行calculateMonthlyInterest()计算，并打印。

测试数据：

saver1：savingBalance：$ 2000.00 ；

saver2：savingBalance：$ 3000.00 ；

第一次annualInterestRate：0.03；

第二次annualInterestRate：0.04；

源代码：

//10.7.cpp

# include <iostream>

# include "SavingAccount.h"

using namespace std;

int main()

{

SavingAccount save1(2000.00);

SavingAccount save2(3000.00);

SavingAccount::modifyInterestRate(0.03);

save1.calculateMonthlyInterest();

save2.calculateMonthlyInterest();

cout << "annualInterestRate:" << 0.03 << endl;

cout << "save1:";

save1.print();

cout << "save2:";

save2.print();

cout << "\n\n";

SavingAccount::modifyInterestRate(0.04);

save1.calculateMonthlyInterest();

save2.calculateMonthlyInterest();

cout << "annualInterestRate:" << 0.04 << endl;

cout << "save1:";

save1.print();

cout << "save2:";

save2.print();

system("pause");

return 0;

}

//SavingAccount.h

# include <iostream>

# ifndef SavingAccount\_h

# define SavingAccount\_h

class SavingAccount {

private:

static double annualInterestRate;

double savingBalance;

public:

SavingAccount(double a = 0);

void calculateMonthlyInterest();

void print();

static void modifyInterestRate(double);

};

# endif

// SavingAccount.cpp

# include <iostream>

# include "SavingAccount.h"

# include <iomanip>

using namespace std;

double SavingAccount::annualInterestRate;

SavingAccount::SavingAccount(double a)

{

this->savingBalance = a;

}

void SavingAccount::calculateMonthlyInterest()

{

this->savingBalance += this->savingBalance\*annualInterestRate / 12.0;

}

void SavingAccount::modifyInterestRate(double a)

{

SavingAccount::annualInterestRate = a;

}

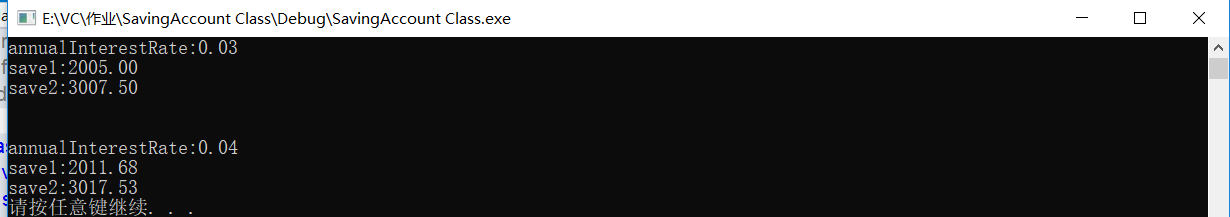
void SavingAccount::print()

{

cout <<fixed<<setprecision(2)<< this->savingBalance << endl;

}

运行结果：



题9.8

**IntegerSet Class**

题目要求：

创建一个IntegerSet类，这个类的每一个对象可以容纳0~100的整数，用bool型的vector数组来存储这个整数集（如果数i再这个集合里，那么Element[i]为True，否则为False）。提供下列功能的成员函数

：unionOfSets() ：求两个集合的并集；

intersectionOfSets() ：求两个集合的交集；

intsertElement(int k) ：用于将整数k插入到集合中；（使Element[k]为True）

deleteElement(int k) ：用于从集合中删除k；（使Element[k]为False）

printSet() ：打印这个集合（如果为空集，则打印“---“）

isEqualTo() ：用于判断两个集合是否相等；

同时提供另一个构造函数，接收的参数为一个数组以及数组长度，使用这个数组来初始化对象。

解题思路：

在private中申明一个const static size=101，用于确定数组大小，并申明

vector <bool> set（bool型的vector数组set）；在构造函数中使用初始化成员列表初始化set的大小为101；重载的构造函数中使用for循环来给set赋值（若数组中的元素k符合0~100的范围，则使set[k]为True）；求并集是使用if条件判断句，只要两个集合一个的元素True，则并集中该元素为True（交集原理类似）；调用insertElement(int k)函数是，将set[k]赋值为True即可，deleteElement(int k)函数则反之；isEqualTo()函数用于比较两个集合是否相同，使用for循环从第一位开始判断，若中间出现不一样的元素，则返回false；

源代码：

//9.8.cpp

# include <iostream>

# include "IntegerSet.h"

using namespace std;

int main()

{

int a[5] = { 15,23,45,34,55 };

int b[5] = { 17,23,43,12,55 };

IntegerSet set1(a, sizeof(a) / 4);

IntegerSet set2(b, sizeof(b) / 4);

cout << "set1:";

set1.printSet();

cout << "set2:";

set2.printSet();

cout << "set1 is equal to set2 ? " << set1.isEqualTo(set2);

cout << "\n\n";

IntegerSet set3 = set1.unionOfSets(set2);

IntegerSet set4 = set1.intersectionOfSets(set2);

cout << "union of set1 and set2:";

set3.printSet();

cout << "intersection of set1 and set2:";

set4.printSet();

cout << "\n\n";

set3.insertElement(100);

cout << "set3 is(insert 100):";

set3.printSet();

set4.deleteElement(55);

cout << "set4 is (delete 55):";

set4.printSet();

system("pause");

return 0;

}

//IntegerSet.h

# include <iostream>

# include <vector>

# ifndef IntegerSet\_h

# define IntegerSet\_h

using namespace std;

class IntegerSet {

private:

const static int size = 101;

vector<bool> set;

public:

IntegerSet();

IntegerSet(int \*a, int l);

IntegerSet unionOfSets(IntegerSet a);

IntegerSet intersectionOfSets(IntegerSet a);

void insertElement(int k);

void deleteElement(int k);

void printSet();

bool isEqualTo(IntegerSet a);

};

# endif

// IntegerSet.cpp

# include <iostream>

# include "IntegerSet.h"

using namespace std;

IntegerSet::IntegerSet()

:set(size)

{

for (int i = 0; i < sizeof(set); i++)

set[i] = 0;

}

IntegerSet::IntegerSet(int \*a, int l)

:set(size)

{

for (int i = 0; i < l; i++)

{

if (a[i] >= 0 && a[i] <= 100)

set[a[i]] = true;

}

}

IntegerSet IntegerSet::unionOfSets(IntegerSet a)

{

IntegerSet c;

for (int i = 0; i <= 100; i++)

{

if (this->set[i] || a.set[i])

c.set[i] = true;

else

c.set[i] = false;

}

return c;

}

IntegerSet IntegerSet::intersectionOfSets(IntegerSet a)

{

IntegerSet c;

for (int i = 0; i <= 100; i++)

{

if (this->set[i] && a.set[i])

c.set[i] = true;

else

c.set[i] = false;

}

return c;

}

void IntegerSet::insertElement(int k)

{

set[k] = true;

}

void IntegerSet::deleteElement(int k)

{

set[k] = false;

}

void IntegerSet::printSet()

{

bool k = 0;

cout << "{";

for (int i = 0; i <= 100; i++)

if (set[i])

{

cout << i<<" ";

k = 1;

}

if (!k)

cout << "---";

cout << "}";

cout << endl;

}

bool IntegerSet::isEqualTo(IntegerSet a)

{

for (int i = 0; i <= 100; i++)

{

if (this->set[i] != a.set[i])

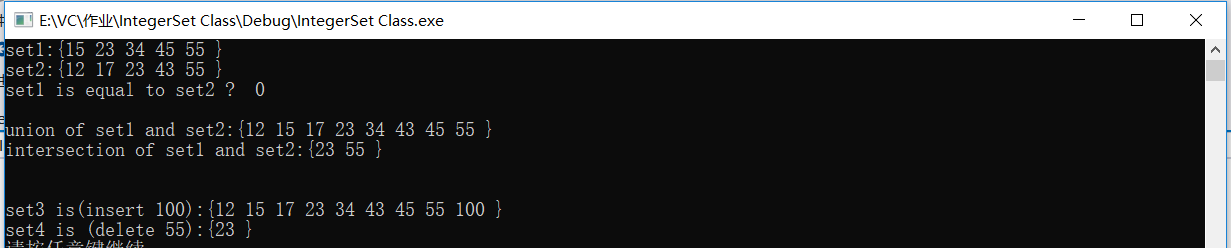
return false;

}

return true;

}

运行结果：



题10.9

**Time Class Modification**

题目要求

对Time类进行改进，只使用一个totalSeconds来存储时间（替代原来使用Hour，Munite,Second的方法）。这样的目的使得用户使用同样的方法而且得到同样的结果。并且用户没有显示更改对象的权力。

解题思路：

用totalSeconds来代替三个数据，所以首先要明白totalSeconds的算法：

因为类中没有Hour，Munite，Second，所以问题在于如何获取这三个数据，因而在getHour()中使用totalSeconds/3600来计算Hour并返回，其余两个数据的获取也类似。

源代码：

//10.9.cpp

# include <iostream>

# include "Time.h"

using namespace std;

int main()

{

Time t;

t.setTime(20, 12, 32);

t.printUniversal();

t.printStandard();

t.setHour(10).setMinute(23).setSecond(5);

t.printUniversal();

t.printStandard();

return 0;

}

//Time.h

# include <iostream>

class Time

{

public:

Time(int = 0, int = 0, int = 0);

//set functions

Time &setTime(int, int, int);

Time &setHour(int);

Time &setMinute(int);

Time &setSecond(int);

// get functions

int getHour() const;

int getMinute() const;

int getSecond() const;

//print functions

void printUniversal() const;

void printStandard() const;

private:

int totalSeconds;

};

//Time.cpp

# include <iostream>

# include <iomanip>

# include "Time.h"

using namespace std;

Time::Time(int a, int b, int c)

{

setTime(a, b, c);

}

Time & Time::setTime(int a, int b, int c)

{

setHour(a);

setMinute(b);

setSecond(c);

return \*this;

}

Time & Time::setHour(int h)

{

totalSeconds = h \* 3600 + getMinute() \* 60 + getSecond();

return \*this;

}

Time &Time::setMinute(int m)

{

totalSeconds = getHour() \* 3600 + m \* 60 + getSecond();

return \*this;

}

Time &Time::setSecond(int s)

{

totalSeconds = getHour() \* 3600 + getMinute() \* 60 + s;

return \*this;

}

int Time::getHour() const

{

int h = totalSeconds / 3600;

return h;

}

int Time::getMinute() const

{

int m = (totalSeconds % 3600) / 60;

return m;

}

int Time::getSecond() const

{

int s = totalSeconds % 60;

return s;

}

void Time::printUniversal() const

{

cout << setfill('0') << setw(2) << getHour() << ":" << setw(2) << getMinute() << ":" << setw(2) << getSecond() << endl;

}

void Time::printStandard() const

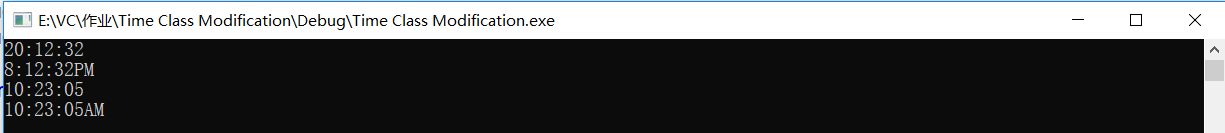
{

cout << ((getHour() == 12) ? 12 : getHour() % 12)

<< ":" << setfill('0') << setw(2) << getMinute() << ":" << setw(2) << getSecond() << (getHour()< 12 ? "AM" : "PM") << endl;

}

运行结果：



题10.10

**Card Shuffling and Deal**

题目要求：

编写一个洗牌于发牌的项目，项目中包括Card类、DeckOfCard类与一个测试程序。具体要求如下：

Card类：

1. 两个int型数据成员，分别表示face和suit；
2. 一个构造函数，接送两个int参数，分别给face和suit初始化；
3. 两个static型的string数组，分别表示face和suit；
4. 包括toString()函数，用来将card以string类型返回；

DeckOfCard类：

1. 一个Card型的vector数组—deck[]，用来存储cards；
2. 一个整形数currentCard用来表示下一张要处理的牌；
3. 一个默认构造函数用于初始化deck[]中的每一个Card对象，使用vector中的push\_back()函数将每一个初始化完成的Card对象加入到deck[]中；
4. 一个洗牌函数shuffle()，shuffle()函数应该在deck[]中反复做，每一次都随机抽取一张牌与当前牌进行交换；
5. 一个dealCard()函数，用于返回下一张要处理的牌；
6. 一个bool型的moreCard()函数，用来判断是否还有牌需要处理；

解题思路：

在DeckOfCard的构造函数中，采用for循环嵌套，来对deck[]中的52个元素进行初始化（注意：在初始化过程并不需要进行随机初始化，进行规则初始化即可），初始化完毕后，使用shuffle()函数进行洗牌操作，打乱牌的排列，最后在测试文件中使用for循环，调用每一个deck[]元素的toString()函数来输出。

源代码：

//10.10.cpp

# include <iostream>

# include <string.h>

# include <vector>

# include <ctime>

# include <cstdlib>

# include <iomanip>

# include "Card.h"

# include "DeckOfCards.h"

using namespace std;

int main()

{

DeckOfCard a;

a.shuffle();

cout << fixed << left;

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

{

if (!(i % 2))

{

cout << setw(20)<< a.deck[i].toString();

}

else

{

cout << setw(20)<< a.deck[i].toString();

cout << endl;

}

}

cout << "\n\n";

a.shuffle();

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

{

if (!(i % 2))

{

cout << setw(20) << a.deck[i].toString();

}

else

{

cout << setw(20) << a.deck[i].toString();

cout << endl;

}

}

return 0;

}

//Card.h

# include <iostream>

# include <string>

# ifndef Card\_h

# define Card\_h

using namespace std;

class Card{

private:

int face;

int suit;

public:

Card(int, int);

Card();

static const int totalFaces = 13;

static const int totalSuits = 4;

static string faceName[totalFaces];

static string suitName[totalSuits];

string toString();

};

# endif

//Card.cpp

# include <iostream>

# include <cstdlib>

# include <iomanip>

# include "Card.h"

# include <string>

using namespace std;

string Card::faceName[] = { "One","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten","Jack","Queen","King" };

string Card::suitName[] = { "Heart","Diamond","Club","Spade" };

Card::Card(int a, int b)

{

face = a;

suit = b;

}

Card::Card()

{

face = 1;

suit = 1;

}

string Card::toString()

{

return faceName[face] +" of " + suitName[suit];

}

//DeckOfCard.h

# include <iostream>

# include <vector>

# ifndef DeckOfCards\_h

# define DeckOfCards\_h

# include "Card.h"

class DeckOfCard {

public:

DeckOfCard();

void shuffle();

vector <Card> deck;

Card dealCard();

bool moreCards();

private:

int currentCard;

};

# endif

//DeckOfCard.cpp

# include <iostream>

# include "Card.h"

# include "DeckOfCards.h"

# include <ctime>

DeckOfCard::DeckOfCard()

:currentCard(0)

{

for(int i=0;i<13;i++)

for (int j = 0; j < 4; j++)

{

Card card(i, j);

deck.push\_back(card);

}

}

void DeckOfCard::shuffle()

{

currentCard = 0;

srand(unsigned (time(0)));

for (currentCard; currentCard < 52; currentCard++)

{

int j = rand() % 52;

Card card=deck[j];

deck[j] = deck[currentCard];

deck[currentCard] = card;

}

}

Card DeckOfCard::dealCard()

{

return deck[currentCard +1];

}

bool DeckOfCard::moreCards()

{

if (currentCard == 52)

return 0;

else

return 1;

}

运行结果：

