题13.12

**Payroll System Modification**

题目要求：

对Figs. 13.9-13.17的工资支付系统进行升级，要求在基类Employee中加入一个数据成员birthdate，birthdate使用Figs. 11.6-11.7的Date类来表示。默认工资每个月支付一次。并在员工的生日所在月，给员工加100元的奖金。

解题思路：

在Employee中加入Date型的数据birthday，用于存储生日，并改编Date类，加入一个getMonth()函数，用于获取生日所在月，在Employee中加入getBirthMonth()用于获取生日所在月，打印工资单时，先判断当前月是否为员工生日所在月，若是则工资加100，若不是，则正常输出工资。

源代码：

**//13.12.cpp**

#include<iostream>

#include<iomanip>

#include<vector>

#include"Employee.h"

#include"SalariedEmployee.h"

#include"CommissionEmployee.h"

#include"BasePlusCommissionEmployee.h"

# include "Date.h"

using namespace std;

void virtualViaReference(const Employee&,int);

int main()

{

cout << fixed << setprecision(2);

SalariedEmployee salariedEmployee("John", "Smith", "111-11-1111",Date(1,1,1999), 800.0);

CommissionEmployee commissionEmployee("Sue", "Jones", "333-33-3333", Date(5,20,2000),10000, .06);

BasePlusCommissionEmployee basePlusCommissionEmployee("Bob", "Lewis", "444-44-4444",Date(7,12,2003),5000, .04, 300);

vector <Employee \*> employee(3);

employee[0] = &salariedEmployee;

employee[1] = &commissionEmployee;

employee[2] = &basePlusCommissionEmployee;

int month = 1;

for(;month<=12;month++)

for (size\_t i = 0; i < employee.size() ; i++)

virtualViaReference(\*employee[i], month);

}

void virtualViaReference(const Employee& baseClassRef,int month)

{

baseClassRef.print();

if(baseClassRef.getBirthMonth()==month)

cout << "\nearned $" << baseClassRef.earnings()+100.00 <<" (BirthMonth ! !)"<< "\n\n";

else

cout << "\nearned $" << baseClassRef.earnings() << "\n\n";

}

**//Employee.h**

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include<string>

# include "Date.h"

using namespace std;

class Employee

{

public:

Employee(const string&, const string&, const string&, Date);

void setFirstName(const string&);

string getFirstName()const;

void setLastName(const string&);

string getLastName()const;

void setSocialSecurityNumber(const string&);

string getSocialSecurityNumber()const;

int getBirthMonth() const;

virtual double earnings() const = 0;

virtual void print() const;

private:

string firstName;

string lastName;

string socialSecurityNumber;

Date birthday;

};

#endif

**//Employee.cpp**

#include<iostream>

#include "Employee.h"

using namespace std;

Employee::Employee(const string &first,const string &last,const string &ssn,Date date)

:firstName(first),lastName(last),socialSecurityNumber(ssn),birthday(date)

{

}

void Employee::setFirstName(const string& first)

{

firstName = first;

}

string Employee::getFirstName()const

{

return firstName;

}

void Employee::setLastName(const string& last)

{

lastName = last;

}

string Employee::getLastName()const

{

return lastName;

}

void Employee::setSocialSecurityNumber(const string& ssn)

{

socialSecurityNumber = ssn;

}

string Employee::getSocialSecurityNumber()const

{

return socialSecurityNumber;

}

int Employee::getBirthMonth() const

{

return birthday.getMonth();

}

void Employee::print()const

{

cout << getFirstName() << ' ' << getLastName() << "\nsocial security number: " << getSocialSecurityNumber();

}

**//SalariedEmployee.h**

#ifndef SALARIED\_H

#define SALARIED\_H

#include "Employee.h"

class SalariedEmployee :

public Employee

{

public:

SalariedEmployee(const string&, const string&, const string&, Date,double = 0.0);

void setWeeklySalary(double);

double getWeeklySalary() const;

virtual double earnings()const;

virtual void print()const;

private:

double weeklySalary;

};

#endif

**//SalariedEmployee.cpp**

#include<iostream>

#include "SalariedEmployee.h"

using namespace std;

SalariedEmployee::SalariedEmployee(const string &first,const string &last,const string&ssn,Date date,double salary)

:Employee(first,last,ssn,date)

{

setWeeklySalary(salary);

}

void SalariedEmployee::setWeeklySalary(double salary)

{

weeklySalary = (salary < 0.0) ? 0.0 : salary;

}

double SalariedEmployee::getWeeklySalary()const

{

return weeklySalary;

}

double SalariedEmployee::earnings()const

{

return getWeeklySalary();

}

void SalariedEmployee::print()const

{

cout << "salaried employee: ";

Employee::print();

cout << "\nweekly salary: " << getWeeklySalary();

}

**//CommissionEmployee.h**

#ifndef COMMISSION\_H

#define COMMISSION\_H

#include "Employee.h"

class CommissionEmployee :

public Employee

{

public:

CommissionEmployee(const string&, const string&, const string&, Date , double = 0.0, double = 0.0);

void setCommissionRate(double);

double getCommissionRate()const;

void setGrossSales(double);

double getGrossSlaes()const;

virtual double earnings()const;

virtual void print()const;

private:

double grossSales;

double commissionRate;

};

#endif // !COMMISSION\_H

**//CommissionEmployee.cpp**

#include<iostream>

#include "CommissionEmployee.h"

using namespace std;

CommissionEmployee::CommissionEmployee(const string& first, const string& last, const string& ssn, Date date, double sales, double rate)

:Employee(first,last,ssn,date)

{

setGrossSales(sales);

setCommissionRate(rate);

}

void CommissionEmployee::setCommissionRate(double rate)

{

commissionRate = ((rate > 0.0 && rate < 1.0) ? rate : 0.0);

}

double CommissionEmployee::getCommissionRate()const

{

return commissionRate;

}

void CommissionEmployee::setGrossSales(double sales)

{

grossSales = ((sales < 0.0) ? 0.0 : sales);

}

double CommissionEmployee::getGrossSlaes()const

{

return grossSales;

}

double CommissionEmployee::earnings()const

{

return getCommissionRate()\* getGrossSlaes();

}

void CommissionEmployee::print()const

{

cout << "commission employee: ";

Employee::print();

cout << "\ngross sales: " << getGrossSlaes() << "; commission rate: " << getCommissionRate();

}

**//BasePlusCommissionEmployee.h**

#ifndef BASEPLUS\_H

#define BASEPLUS\_H

#include "CommissionEmployee.h"

class BasePlusCommissionEmployee :

public CommissionEmployee

{

public:

BasePlusCommissionEmployee(const string&, const string&, const string&, Date , double = 0.0, double = 0.0, double = 0.0);

void setBaseSalary(double);

double getBaseSalary()const;

virtual double earnings()const;

virtual void print()const;

private:

double baseSalary;

};

#endif // !BASEPLUS\_H

**//BasePlusCommissionEmployee.cpp**

#include<iostream>

#include "BasePlusCommissionEmployee.h"

using namespace std;

BasePlusCommissionEmployee::BasePlusCommissionEmployee(const string &first,const string &last,const string &ssn, Date date, double sales,double rate,double salary)

:CommissionEmployee(first,last,ssn,date,sales,rate)

{

setBaseSalary(salary);

}

void BasePlusCommissionEmployee::setBaseSalary(double salary)

{

baseSalary = ((salary < 0.0) ? 0.0 : salary);

}

double BasePlusCommissionEmployee::getBaseSalary()const

{

return baseSalary;

}

double BasePlusCommissionEmployee::earnings()const

{

return getBaseSalary() + CommissionEmployee::earnings();

}

void BasePlusCommissionEmployee::print()const

{

cout << "base-salaried ";

CommissionEmployee::print();

cout << "; base salary: " << getBaseSalary();

}

**//Date.h**

# ifndef Date\_h

# define Date\_h

# include <iostream>

using namespace std;

class Date

{

friend ostream &operator<<(ostream &, const Date &);

public:

Date(int m = 1, int d = 1, int y = 1990);

void setDate(int, int, int);

Date &operator++();

Date operator++(int);

const Date &operator+=(int);

static bool leapYear(int);

bool endOfMonth(int) const;

int getMonth() const;

private:

int month;

int day;

int year;

static const int days[];

void helpIncrement();

};

# endif

**//Date.cpp**

# include <iostream>

# include <string>

# include "Date.h"

using namespace std;

const int Date::days[] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };

Date::Date(int month, int day, int year)

{

setDate(month, day, year);

}

void Date::setDate(int mm, int dd, int yy)

{

if (mm >= 1 && mm <= 12)

month = mm;

else

throw invalid\_argument("Month must be 1-12");

if (yy >= 1990 && yy <= 2100)

year = yy;

else

throw invalid\_argument("Year must be >=1990 and <=2100");

if ((month == 2 && leapYear(year) && dd >= 1 && dd <= 29) || (dd >= 1 && dd <= days[month]))

day = dd;

else

throw invalid\_argument("Day is out of range for current month and year");

}

Date &Date::operator++()

{

helpIncrement();

return \*this;

}

Date Date::operator++(int)

{

Date temp = \*this;

helpIncrement();

return temp;

}

const Date &Date::operator+=(int additionalDays)

{

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

helpIncrement();

return \*this;

}

bool Date::leapYear(int testYear)

{

if (testYear % 400 == 0 || (testYear % 100 != 0 && testYear % 4 == 0))

return true;

else

return false;

}

int Date::getMonth() const

{

return month;

}

bool Date::endOfMonth(int testDay) const

{

if (month == 2 && leapYear(year))

return testDay == 29;

else

return testDay == days[month];

}

void Date::helpIncrement()

{

if (!endOfMonth(day))

++day;

else

if (month < 12)

{

++month;

day = 1;

}

else

{

++year;

month = 1;

day = 1;

}

}

ostream &operator<<(ostream &output, const Date &d)

{

static string monthName[13] = { "","January","February","March","April","May","June","July",

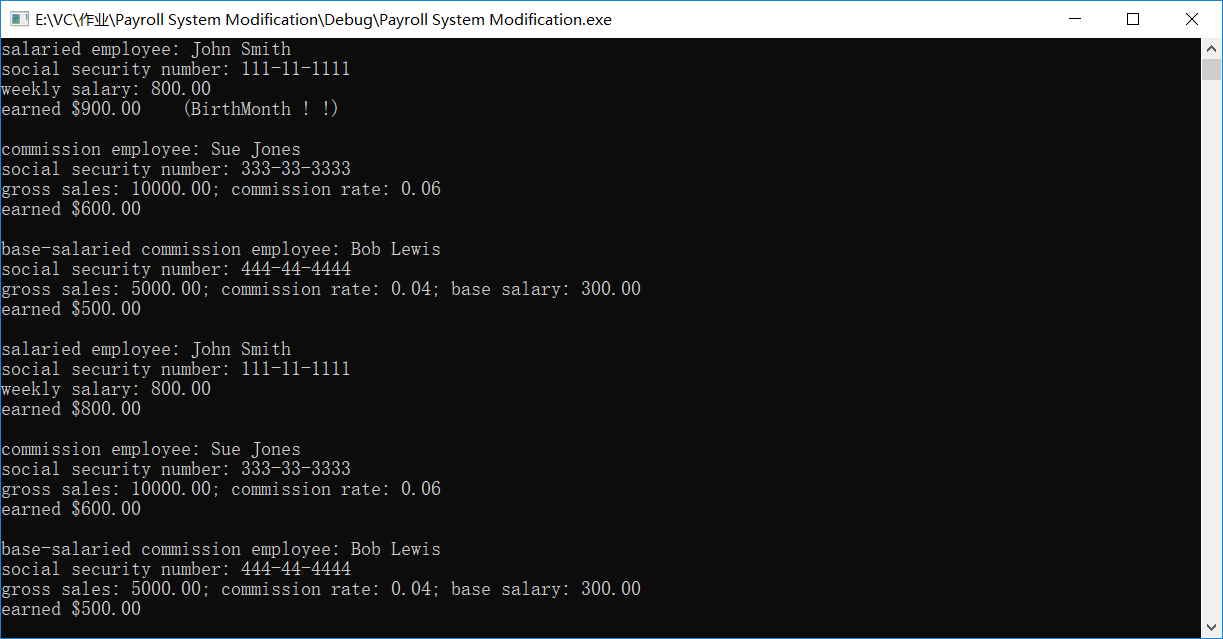
"August","September","October","Novebmer","December" };

output << monthName[d.month] << ' ' << d.day << ", " << d.year;

return output;

}

运行结果：



题13.16

**Polymorphic Banking Program Using Account Hierarchy**

题目要求：

利用多态改进12.10的继承类，创建一个Account型的指针vector型数组，用来存储各个账户，包括SavingsAccount类型和CheckingAccount类型的，在每次对账户执行操作时判断账户类型，如果时SavingsAccount类型的账户，则计算它的利息，并将利息加入到balance余额中。

解题思路：

采用typeinfo中的dynamic\_cast来判断账户类型，并根据判断结果进行不同操作；利用到多态中的虚函数，以及纯虚函数。

源代码：

**//13.16.cpp**

# include <iostream>

# include <typeinfo>

# include <vector>

# include <iomanip>

# include "Account.h"

# include "SavingsAccount.h"

# include "CheckingAccount.h"

using namespace std;

int main()

{

cout << fixed << setprecision(2);

SavingsAccount num0(2000.0, 0.05);

SavingsAccount num2(3000.0, 0.05);

CheckingAccount num1(2000.0, 10.0);

CheckingAccount num3(3000.0, 10.0);

vector <Account\*> account(4);

account[0] = &num0;

account[1] = &num1;

account[2] = &num2;

account[3] = &num3;

for (size\_t i = 0; i < account.size(); ++i)

{

cout << "Balance of num" << i << ": ";

account[i]->print();

}

cout << "\n\n";

cout << "credit 1000.00 to each account:" << endl;

for (size\_t i = 0; i < account.size(); ++i)

{

account[i]->credit(1000.0);

SavingsAccount \*temp = dynamic\_cast<SavingsAccount\*>(account[i]);

if (temp!=0)

{

temp->calculateInterest();

}

cout << "Balance of num" << i << ": ";

account[i]->print();

}

system("pause");

return 0;

}

**//SavingsAccount.h**

# include <iostream>

# include "Account.h"

# ifndef SavingsAccount\_h

# define SavingsAccount\_h

class SavingsAccount : public Account

{

public:

SavingsAccount(double = 0, double = 0);

double calculateInterest();

private:

double interestRate;

};

# endif

**//SavingsAccount.cpp**

# include <iostream>

# include "SavingsAccount.h"

SavingsAccount::SavingsAccount(double a, double b)

:Account(a)

{

interestRate = b;

}

double SavingsAccount::calculateInterest()

{

double Interest = getbalance()\*interestRate;

credit(Interest);

return Interest;

}

**//CheckingAccount.h**

# include <iostream>

# include "Account.h"

# ifndef CheckingAccount\_h

# define CheckingAccount\_h

class CheckingAccount :public Account

{

public:

CheckingAccount(double a = 0, double b = 0);

virtual void credit(double);

virtual bool debit(double);

private:

double Checkingfee;

};

# endif

**//CheckingAccount.cpp**

# include <iostream>

# include "CheckingAccount.h"

using namespace std;

CheckingAccount::CheckingAccount(double a, double b)

:Account(a)

{

Checkingfee = b;

}

void CheckingAccount::credit(double a)

{

Account::credit(a - Checkingfee);

}

bool CheckingAccount::debit(double a)

{

if (Account::debit(a))

{

Account::debit(Checkingfee);

return true;

}

else

{

cout << "Error ! (Balance is not enough)";

return false;

}

}

运行结果：

