EX1

编辑后Date.h

// Fig. 10.6: Date.h

// Date class definition with overloaded increment operators.

#ifndef DATE\_H

#define DATE\_H

#include <array>

#include <iostream>

class Date

{

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

public:

Date( int m = 1, int d = 1, int y = 1900 ); // default constructor

void setDate( int, int, int ); // set month, day, year

Date &operator++(); // prefix increment operator

Date operator++( int ); // postfix increment operator

Date &operator+=( unsigned int ); // add days, modify object

static bool leapYear( int ); // is date in a leap year?

bool endOfMonth( int ) const; // is date at the end of month?

int getMonth();

private:

unsigned int month;

unsigned int day;

unsigned int year;

static const std::array< unsigned int, 13 > days; // days per month

void helpIncrement(); // utility function for incrementing date

}; // end class Date

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Date.cpp：

// Fig. 10.7: Date.cpp

// Date class member- and friend-function definitions.

#include <iostream>

#include <string>

#include "Date.h"

using namespace std;

// initialize static member; one classwide copy

const array< unsigned int, 13 > Date::days =

{ 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

// Date constructor

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

{

setDate( month, day, year );

} // end Date constructor

// set month, day and 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 >= 1900 && yy <= 2100 )

year = yy;

else

throw invalid\_argument( "Year must be >= 1900 and <= 2100" );

// test for a leap year

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" );

} // end function setDate

// overloaded prefix increment operator

Date &Date::operator++()

{

helpIncrement(); // increment date

return \*this; // reference return to create an lvalue

} // end function operator++

// overloaded postfix increment operator; note that the

// dummy integer parameter does not have a parameter name

Date Date::operator++( int )

{

Date temp = \*this; // hold current state of object

helpIncrement();

// return unincremented, saved, temporary object

return temp; // value return; not a reference return

} // end function operator++

// add specified number of days to date

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

{

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

helpIncrement();

return \*this; // enables cascading

} // end function operator+=

// if the year is a leap year, return true; otherwise, return false

bool Date::leapYear( int testYear )

{

if ( testYear % 400 == 0 ||

( testYear % 100 != 0 && testYear % 4 == 0 ) )

return true; // a leap year

else

return false; // not a leap year

} // end function leapYear

// determine whether the day is the last day of the month

bool Date::endOfMonth( int testDay ) const

{

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

return testDay == 29; // last day of Feb. in leap year

else

return testDay == days[ month ];

} // end function endOfMonth

int Date::getMonth()

{

return this->month;

}

// function to help increment the date

void Date::helpIncrement()

{

// day is not end of month

if ( !endOfMonth( day ) )

++day; // increment day

else

if ( month < 12 ) // day is end of month and month < 12

{

++month; // increment month

day = 1; // first day of new month

} // end if

else // last day of year

{

++year; // increment year

month = 1; // first month of new year

day = 1; // first day of new month

} // end else

} // end function helpIncrement

// overloaded output operator

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

{

static string monthName[ 13 ] = { "", "January", "February",

"March", "April", "May", "June", "July", "August",

"September", "October", "November", "December" };

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

return output; // enables cascading

} // end function operator<<

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

编辑后Employee.h:

// Fig. 12.9: Employee.h

// Employee abstract base class.

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include <string> // C++ standard string class

#include "Date.h"

class Employee

{

public:

Employee( const std::string &, const std::string &,

const std::string & ,const double& bonus = 0);

virtual ~Employee() { } // virtual destructor

void setFirstName( const std::string & ); // set first name

std::string getFirstName() const; // return first name

void setLastName( const std::string & ); // set last name

std::string getLastName() const; // return last name

void setSocialSecurityNumber( const std::string & ); // set SSN

std::string getSocialSecurityNumber() const; // return SSN

void setBirthday(const Date&); // set birthDate

Date getBirthday() const; // return birthDate

void setBonus(const double&);

double getBonus() const;

// pure virtual function makes Employee abstract base class

virtual double earnings() const = 0; // pure virtual

virtual void print() const; // virtual

private:

std::string firstName;

std::string lastName;

std::string socialSecurityNumber;

Date birthDate;

double bonus;

}; // end class Employee

#endif // EMPLOYEE\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Employee.cpp：

// Fig. 12.10: Employee.cpp

// Abstract-base-class Employee member-function definitions.

// Note: No definitions are given for pure virtual functions.

#include <iostream>

#include "Employee.h" // Employee class definition

using namespace std;

// constructor

Employee::Employee(const string& first, const string& last,

const string& ssn, const double& bonu)

: firstName(first), lastName(last), socialSecurityNumber(ssn), bonus(bonu)

{

// empty body

} // end Employee constructor

// set first name

void Employee::setFirstName( const string &first )

{

firstName = first;

} // end function setFirstName

// return first name

string Employee::getFirstName() const

{

return firstName;

} // end function getFirstName

// set last name

void Employee::setLastName( const string &last )

{

lastName = last;

} // end function setLastName

// return last name

string Employee::getLastName() const

{

return lastName;

} // end function getLastName

// set social security number

void Employee::setSocialSecurityNumber( const string &ssn )

{

socialSecurityNumber = ssn; // should validate

} // end function setSocialSecurityNumber

// return social security number

string Employee::getSocialSecurityNumber() const

{

return socialSecurityNumber;

} // end function getSocialSecurityNumber

void Employee::setBirthday(const Date& birth)

{

this->birthDate = birth;

}

Date Employee::getBirthday() const

{

return this->birthDate;

}

void Employee::setBonus(const double& bonu)

{

this->bonus = bonu;

}

double Employee::getBonus() const

{

return this->bonus;

}

// print Employee's information (virtual, but not pure virtual)

void Employee::print() const

{

cout << getFirstName() << ' ' << getLastName() << "\nbirthdate: " << getBirthday()

<< "\nsocial security number: " << getSocialSecurityNumber();

} // end function print

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SalariedEmployee.h：

// Fig. 12.11: SalariedEmployee.h

// SalariedEmployee class derived from Employee.

#ifndef SALARIED\_H

#define SALARIED\_H

#include <string> // C++ standard string class

#include "Employee.h" // Employee class definition

class SalariedEmployee : public Employee

{

public:

SalariedEmployee( const std::string &, const std::string &,

const std::string &, double = 0.0 );

virtual ~SalariedEmployee() { } // virtual destructor

void setWeeklySalary( double ); // set weekly salary

double getWeeklySalary() const; // return weekly salary

// keyword virtual signals intent to override

virtual double earnings() const override; // calculate earnings

virtual void print() const override; // print object

private:

double weeklySalary; // salary per week

}; // end class SalariedEmployee

#endif // SALARIED\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SalariedEmployee.cpp:

// Fig. 12.12: SalariedEmployee.cpp

// SalariedEmployee class member-function definitions.

#include <iostream>

#include <stdexcept>

#include "SalariedEmployee.h" // SalariedEmployee class definition

using namespace std;

// constructor

SalariedEmployee::SalariedEmployee( const string &first,

const string &last, const string &ssn, double salary )

: Employee( first, last, ssn )

{

setWeeklySalary( salary );

} // end SalariedEmployee constructor

// set salary

void SalariedEmployee::setWeeklySalary( double salary )

{

if ( salary >= 0.0 )

weeklySalary = salary;

else

throw invalid\_argument( "Weekly salary must be >= 0.0" );

} // end function setWeeklySalary

// return salary

double SalariedEmployee::getWeeklySalary() const

{

return weeklySalary;

} // end function getWeeklySalary

// calculate earnings;

// override pure virtual function earnings in Employee

double SalariedEmployee::earnings() const

{

return getWeeklySalary();

} // end function earnings

// print SalariedEmployee's information

void SalariedEmployee::print() const

{

cout << "salaried employee: ";

Employee::print(); // reuse abstract base-class print function

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

} // end function print

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CommissionEmployee.h:

// Fig. 12.13: CommissionEmployee.h

// CommissionEmployee class derived from Employee.

#ifndef COMMISSION\_H

#define COMMISSION\_H

#include <string> // C++ standard string class

#include "Employee.h" // Employee class definition

class CommissionEmployee : public Employee

{

public:

CommissionEmployee( const std::string &, const std::string &,

const std::string &, double = 0.0, double = 0.0 );

virtual ~CommissionEmployee() { } // virtual destructor

void setCommissionRate( double ); // set commission rate

double getCommissionRate() const; // return commission rate

void setGrossSales( double ); // set gross sales amount

double getGrossSales() const; // return gross sales amount

// keyword virtual signals intent to override

virtual double earnings() const override; // calculate earnings

virtual void print() const override; // print object

private:

double grossSales; // gross weekly sales

double commissionRate; // commission percentage

}; // end class CommissionEmployee

#endif // COMMISSION\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CommissionEmployee.cpp:

// Fig. 12.14: CommissionEmployee.cpp

// CommissionEmployee class member-function definitions.

#include <iostream>

#include <stdexcept>

#include "CommissionEmployee.h" // CommissionEmployee class definition

using namespace std;

// constructor

CommissionEmployee::CommissionEmployee( const string &first,

const string &last, const string &ssn, double sales, double rate )

: Employee( first, last, ssn )

{

setGrossSales( sales );

setCommissionRate( rate );

} // end CommissionEmployee constructor

// set gross sales amount

void CommissionEmployee::setGrossSales( double sales )

{

if ( sales >= 0.0 )

grossSales = sales;

else

throw invalid\_argument( "Gross sales must be >= 0.0" );

} // end function setGrossSales

// return gross sales amount

double CommissionEmployee::getGrossSales() const

{

return grossSales;

} // end function getGrossSales

// set commission rate

void CommissionEmployee::setCommissionRate( double rate )

{

if ( rate > 0.0 && rate < 1.0 )

commissionRate = rate;

else

throw invalid\_argument( "Commission rate must be > 0.0 and < 1.0" );

} // end function setCommissionRate

// return commission rate

double CommissionEmployee::getCommissionRate() const

{

return commissionRate;

} // end function getCommissionRate

// calculate earnings; override pure virtual function earnings in Employee

double CommissionEmployee::earnings() const

{

return getCommissionRate() \* getGrossSales();

} // end function earnings

// print CommissionEmployee's information

void CommissionEmployee::print() const

{

cout << "commission employee: ";

Employee::print(); // code reuse

cout << "\ngross sales: " << getGrossSales()

<< "; commission rate: " << getCommissionRate();

} // end function print

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BasePlusCommissionEmployee.h

// Fig. 12.15: BasePlusCommissionEmployee.h

// BasePlusCommissionEmployee class derived from CommissionEmployee.

#ifndef BASEPLUS\_H

#define BASEPLUS\_H

#include <string> // C++ standard string class

#include "CommissionEmployee.h" // CommissionEmployee class definition

class BasePlusCommissionEmployee : public CommissionEmployee

{

public:

BasePlusCommissionEmployee( const std::string &, const std::string &,

const std::string &, double = 0.0, double = 0.0, double = 0.0 );

virtual ~BasePlusCommissionEmployee() { } // virtual destructor

void setBaseSalary( double ); // set base salary

double getBaseSalary() const; // return base salary

// keyword virtual signals intent to override

virtual double earnings() const override; // calculate earnings

virtual void print() const override; // print object

private:

double baseSalary; // base salary per week

}; // end class BasePlusCommissionEmployee

#endif // BASEPLUS\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BasePlusCommissionEmployee.cpp:

// Fig. 12.16: BasePlusCommissionEmployee.cpp

// BasePlusCommissionEmployee member-function definitions.

#include <iostream>

#include <stdexcept>

#include "BasePlusCommissionEmployee.h"

using namespace std;

// constructor

BasePlusCommissionEmployee::BasePlusCommissionEmployee(

const string &first, const string &last, const string &ssn,

double sales, double rate, double salary )

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

{

setBaseSalary( salary ); // validate and store base salary

} // end BasePlusCommissionEmployee constructor

// set base salary

void BasePlusCommissionEmployee::setBaseSalary( double salary )

{

if ( salary >= 0.0 )

baseSalary = salary;

else

throw invalid\_argument( "Salary must be >= 0.0" );

} // end function setBaseSalary

// return base salary

double BasePlusCommissionEmployee::getBaseSalary() const

{

return baseSalary;

} // end function getBaseSalary

// calculate earnings;

// override virtual function earnings in CommissionEmployee

double BasePlusCommissionEmployee::earnings() const

{

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

} // end function earnings

// print BasePlusCommissionEmployee's information

void BasePlusCommissionEmployee::print() const

{

cout << "base-salaried ";

CommissionEmployee::print(); // code reuse

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

} // end function print

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

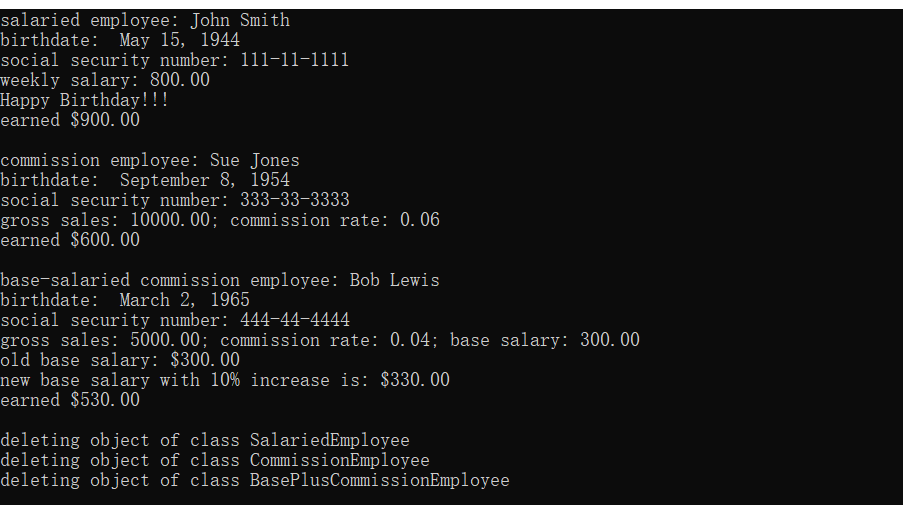
\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

编辑后fig12.19：



// Fig. 12.19: fig12\_19.cpp

// Demonstrating downcasting and runtime type information.

// NOTE: You may need to enable RTTI on your compiler

// before you can compile this application.

#include <iostream>

#include <iomanip>

#include <vector>

#include <typeinfo>

#include <windows.h>

#include "Employee.h"

#include "SalariedEmployee.h"

#include "CommissionEmployee.h"

#include "BasePlusCommissionEmployee.h"

using namespace std;

int main()

{

// set floating-point output formatting

cout << fixed << setprecision( 2 );

// create vector of three base-class pointers

vector < Employee \* > employees( 3 );

// initialize vector with various kinds of Employees

employees[ 0 ] = new SalariedEmployee(

"John", "Smith", "111-11-1111", 800 );

employees[ 1 ] = new CommissionEmployee(

"Sue", "Jones", "333-33-3333", 10000, .06 );

employees[ 2 ] = new BasePlusCommissionEmployee(

"Bob", "Lewis", "444-44-4444", 5000, .04, 300 );

employees[0]->setBirthday(Date(5, 15, 1944));

employees[1]->setBirthday(Date(9, 8, 1954));

employees[2]->setBirthday(Date(3, 2, 1965));

// polymorphically process each element in vector employees

for ( Employee \*employeePtr : employees )

{

employeePtr->print(); // output employee information

cout << endl;

// downcast pointer

BasePlusCommissionEmployee \*derivedPtr =

dynamic\_cast < BasePlusCommissionEmployee \* >( employeePtr );

// determine whether element points to a BasePlusCommissionEmployee

if ( derivedPtr != nullptr ) // true for "is a" relationship

{

double oldBaseSalary = derivedPtr->getBaseSalary();

cout << "old base salary: $" << oldBaseSalary << endl;

derivedPtr->setBaseSalary( 1.10 \* oldBaseSalary );

cout << "new base salary with 10% increase is: $"

<< derivedPtr->getBaseSalary() << endl;

} // end if

SYSTEMTIME systm;

GetLocalTime(&systm);

if (systm.wMonth == (employeePtr->getBirthday()).getMonth())

employeePtr->setBonus(100.00);

if (employeePtr->getBonus())

cout << "Happy Birthday!!!\n";

cout << "earned $" << (employeePtr->getBonus() == 100.00 ? employeePtr->earnings() + 100 : employeePtr->earnings()) << "\n\n";

employeePtr->setBonus(0);

} // end for

// release objects pointed to by vector抯 elements

for ( const Employee \*employeePtr : employees )

{

// output class name

cout << "deleting object of "

<< typeid( \*employeePtr ).name() << endl;

delete employeePtr;

} // end for

return 0;

} // end main

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

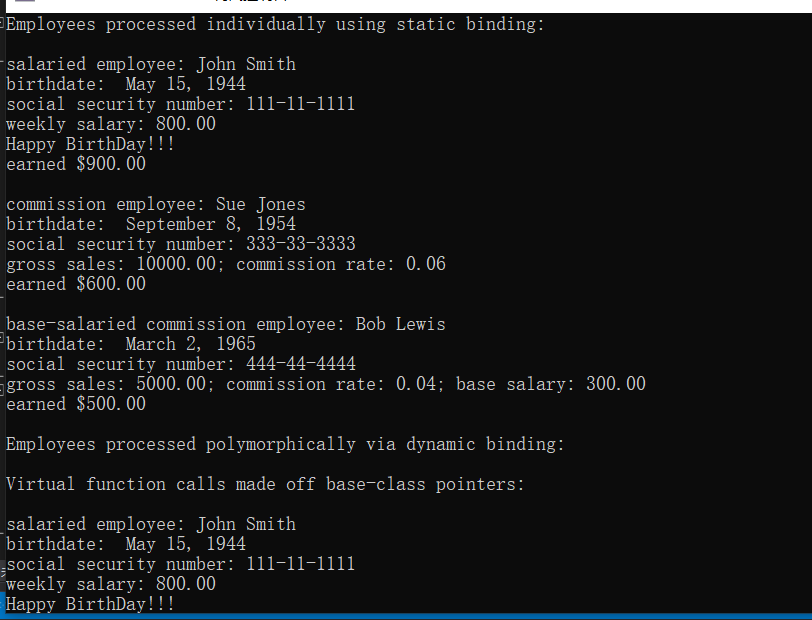
\* and publisher shall not be liable in any event for incidental or \*

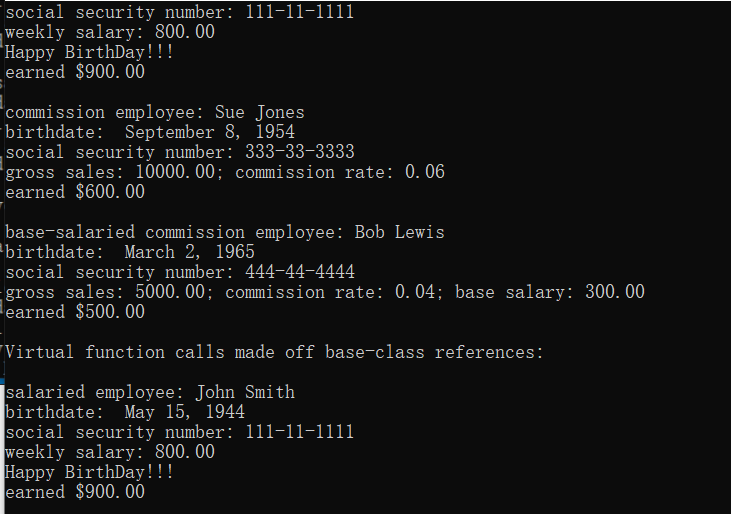
\* consequential damages in connection with, or arising out of, the \*

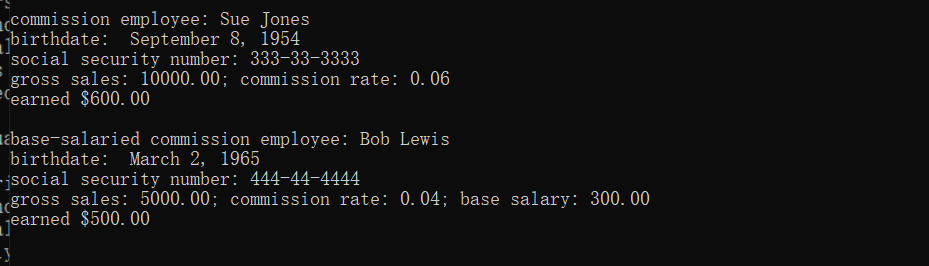
\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

修改12.17：







// Fig. 12.17: fig12\_17.cpp

// Processing Employee derived-class objects individually

// and polymorphically using dynamic binding.

#include <iostream>

#include <iomanip>

#include <vector>

#include <windows.h>

#include "Employee.h"

#include "SalariedEmployee.h"

#include "CommissionEmployee.h"

#include "BasePlusCommissionEmployee.h"

using namespace std;

void virtualViaPointer( const Employee \* const ); // prototype

void virtualViaReference( const Employee & ); // prototype

int main()

{

// set floating-point output formatting

cout << fixed << setprecision( 2 );

// create derived-class objects

SalariedEmployee salariedEmployee(

"John", "Smith", "111-11-1111", 800 );

CommissionEmployee commissionEmployee(

"Sue", "Jones", "333-33-3333", 10000, .06 );

BasePlusCommissionEmployee basePlusCommissionEmployee(

"Bob", "Lewis", "444-44-4444", 5000, .04, 300 );

// create vector of three base-class pointers

vector < Employee\* > employees(3);

// initialize vector with Employees

employees[0] = &salariedEmployee;

employees[1] = &commissionEmployee;

employees[2] = &basePlusCommissionEmployee;

salariedEmployee.setBirthday(Date(5, 15, 1944));

commissionEmployee.setBirthday(Date(9, 8, 1954));

basePlusCommissionEmployee.setBirthday(Date(3, 2, 1965));

SYSTEMTIME systm;

GetLocalTime(&systm);

for (Employee\* target : employees)

if (systm.wMonth == (target->getBirthday()).getMonth())

target->setBonus(100.00);

cout << "Employees processed individually using static binding:\n\n";

// output each Employee抯 information and earnings using static binding

salariedEmployee.print();

if (salariedEmployee.getBonus()) cout << "\nHappy BirthDay!!!";

cout << "\nearned $" << (salariedEmployee.getBonus() == 100.00 ? salariedEmployee.earnings() + 100 : salariedEmployee.earnings()) << "\n\n";

if (commissionEmployee.getBonus()) cout << "\nHappy BirthDay!!!";

commissionEmployee.print();

cout << "\nearned $" << (commissionEmployee.getBonus() == 100.00 ? commissionEmployee.earnings() + 100 : commissionEmployee.earnings()) << "\n\n";

if (basePlusCommissionEmployee.getBonus()) cout << "\nHappy BirthDay!!!";

basePlusCommissionEmployee.print();

cout << "\nearned $" << (basePlusCommissionEmployee.getBonus() == 100.00 ? basePlusCommissionEmployee.earnings() + 100 : basePlusCommissionEmployee.earnings())

<< "\n\n";

cout << "Employees processed polymorphically via dynamic binding:\n\n";

// call virtualViaPointer to print each Employee's information

// and earnings using dynamic binding

cout << "Virtual function calls made off base-class pointers:\n\n";

for ( const Employee \*employeePtr : employees )

virtualViaPointer( employeePtr );

// call virtualViaReference to print each Employee's information

// and earnings using dynamic binding

cout << "Virtual function calls made off base-class references:\n\n";

for ( const Employee \*employeePtr : employees )

virtualViaReference( \*employeePtr ); // note dereferencing

for (Employee\* target : employees)

target->setBonus(0);

} // end main

// call Employee virtual functions print and earnings off a

// base-class pointer using dynamic binding

void virtualViaPointer( const Employee \* const baseClassPtr )

{

baseClassPtr->print();

if (baseClassPtr->getBonus()) cout << "\nHappy BirthDay!!!";

cout << "\nearned $" << (baseClassPtr->getBonus() == 100 ? baseClassPtr->earnings() + 100 : baseClassPtr->earnings()) << "\n\n";

} // end function virtualViaPointer

// call Employee virtual functions print and earnings off a

// base-class reference using dynamic binding

void virtualViaReference( const Employee &baseClassRef )

{

baseClassRef.print();

if (baseClassRef.getBonus()) cout << "\nHappy BirthDay!!!";

cout << "\nearned $" << (baseClassRef.getBonus() == 100 ? baseClassRef.earnings() + 100 : baseClassRef.earnings()) << "\n\n";

} // end function virtualViaReference

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* (C) Copyright 1992-2014 by Deitel & Associates, Inc. and \*

\* Pearson Education, Inc. All Rights Reserved. \*

\* \*

\* DISCLAIMER: The authors and publisher of this book have used their \*

\* best efforts in preparing the book. These efforts include the \*

\* development, research, and testing of the theories and programs \*

\* to determine their effectiveness. The authors and publisher make \*

\* no warranty of any kind, expressed or implied, with regard to these \*

\* programs or to the documentation contained in these books. The authors \*

\* and publisher shall not be liable in any event for incidental or \*

\* consequential damages in connection with, or arising out of, the \*

\* furnishing, performance, or use of these programs. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

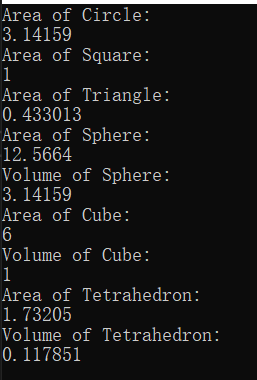
问题与思考：

里面typeinfo头文件提供了一种判断类型的方法，在EX3中采用了这一点。

这个For循环遍历的语法糖前面的作业问题与思考有所提到，这里代码中出现了。

题目里涉及的内容挺有意思：STL array，异常，nullptr，增强for循环，typeinfo，作为课堂补充挺好。

EX2



#include<iostream>

#include<vector>

using namespace std;

#define Pi 3.1415926

#define Root3 1.73205

#define Root2 1.41421

/\*----基类----\*/

class Shape

{

public:

virtual double getArea() = 0;

virtual double getVolume() = 0;

};

/\*----派生类----\*/

class TwoDimensionalShape:public Shape

{

public:

virtual double getArea() = 0;

};

class ThreeDimensionalShape:public Shape

{

public:

virtual double getArea() = 0;

virtual double getVolume() = 0;

};

/\*----二维形状子类----\*/

class Circle:public TwoDimensionalShape

{

double Radius;

public:

Circle(const double& R = 1):Radius(R) {}

virtual ~Circle() {}

virtual double getArea()

{

printf("Area of Circle:\n");

return Pi \* Radius \* Radius;

}

virtual double getVolume() { return 0; }

};

class Square:public TwoDimensionalShape

{

double length;

public:

Square(const double& l = 1):length(l) {}

virtual ~Square() {}

virtual double getArea()

{

printf("Area of Square:\n");

return length \* length;

}

virtual double getVolume() { return 0; }

};

class Triangle :public TwoDimensionalShape

{

/\*----视作等边三角形----\*/

double length;

public:

Triangle(const double& l = 1) :length(l) {}

virtual ~Triangle() {}

virtual double getArea()

{

printf("Area of Triangle:\n");

return 0.5 \* length \* length \* 0.5 \* Root3;

}

virtual double getVolume() { return 0; }

};

/\*\*\*\*\*\*\*\*\*\* 三 维 子 类 \*\*\*\*\*\*\*\*\*\*\*\*/

class Sphere:public ThreeDimensionalShape

{

double radius;

public:

Sphere(const double& R = 1):radius(R) {}

virtual ~Sphere() {}

virtual double getArea()

{

printf("Area of Sphere:\n");

return 4 \* Pi \* radius \* radius;

}

virtual double getVolume()

{

printf("Volume of Sphere:\n");

return (4 / 3) \* Pi \* radius \* radius \* radius;

}

};

class Cube:public ThreeDimensionalShape

{

double length;

public:

Cube(const double& l = 1):length(l) {}

virtual ~Cube() {}

virtual double getArea()

{

printf("Area of Cube:\n");

return 6 \* length \* length;

}

virtual double getVolume()

{

printf("Volume of Cube:\n");

return length \* length \* length;

}

};

class Tetrahedron:public ThreeDimensionalShape

{

/\*----视作正四面体----\*/

double length;

public:

Tetrahedron(const double& l = 1):length(l) {}

virtual ~Tetrahedron() {}

virtual double getArea()

{

printf("Area of Tetrahedron:\n");

return Root3 \* length \* length;

}

virtual double getVolume()

{

printf("Volume of Tetrahedron:\n");

return (Root2 / 12)\* length\* length\* length;

}

};

int main()

{

vector<Shape\*> Col;

Col.push\_back(new Circle());

Col.push\_back(new Square());

Col.push\_back(new Triangle());

Col.push\_back(new Sphere());

Col.push\_back(new Cube());

Col.push\_back(new Tetrahedron());

double tempA = 0, tempV = 0;

for (vector<Shape\*>::iterator it = Col.begin(); it != Col.end(); it++)

{

tempA = (\*it)->getArea(); //运算符问题

cout << tempA << endl;

tempV = (\*it)->getVolume();

if (tempV)

cout << tempV << endl;

}

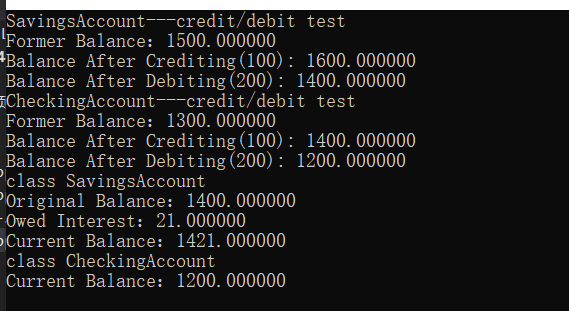
}

问题与思考：

这个因为迭代器指向的成员为指针，又有运算符优先级问题，所以先括号解引用再->调用成员函数。

提交的代码采用了纯虚函数实现，理解上觉得Shape和二维三维派生类不应该用来创建对象，采用让Shape及两个一级派生类变为不可实例化的抽象类，override纯虚函数实现多态。不过亦可静态实现多态，在基类中定义一个状态变量，并让每个类中状态变量取不同值，可静态实现多态。

EX3



#include<iostream>

#include<vector>

#include<typeinfo>

using namespace std;

class Account

{

double balance;

public:

Account(const double& b = 0)

{

if (b < 0)

{

balance = 0;

printf("Invalid value for balance!!!\n");

}

else this->balance = b;

}

virtual void credit(const double& delta = 0)

{

if (delta <= 0)

{

printf("Invalid value to credit!!!\n");

return;

}

else this->balance += delta;

}

virtual void debit(const double& delta = 0)

{

if (delta <= 0)

{

printf("Invalid value to debit!!!\n");

return;

}

else if (delta > this->balance)

{

printf("Debit amount exceeded account balance.");

return;

}

else this->balance -= delta;

}

double getBalance()

{

return balance;

}

virtual double calculateInterest()

{

return 0;

}

};

class SavingsAccount:public Account

{

double rate;

public:

SavingsAccount(const double& b, const double& r) :Account(b), rate(r) {}

double calculateInterest()

{

return getBalance() \* rate \* 0.01;

}

};

class CheckingAccount:public Account

{

double fee;

public:

CheckingAccount(const double& b,const double& f):Account(b),fee(f) {}

void credit(const double& f = 0)

{

Account::credit(f);

}

void debit(const double& i = 0)

{

Account::debit(fee);

}

};

int main()

{

vector<Account\*> target;

target.push\_back(new SavingsAccount(1500, 1.5));

printf("SavingsAccount---credit/debit test\n");

printf("Former Balance：%lf\n", target[0]->getBalance());

target[0]->credit(100);

printf("Balance After Crediting(100): %lf\n", target[0]->getBalance());

target[0]->debit(200);

printf("Balance After Debiting(200): %lf\n", target[0]->getBalance());

target.push\_back(new CheckingAccount(1300, 200));

printf("CheckingAccount---credit/debit test\n");

printf("Former Balance：%lf\n", target[1]->getBalance());

target[1]->credit(100);

printf("Balance After Crediting(100): %lf\n", target[1]->getBalance());

target[1]->debit(200);

printf("Balance After Debiting(200): %lf\n", target[1]->getBalance());

const char\* typeA = "class CheckingAccount";

const char\* typeB = "class SavingsAccount";

for (vector<Account\*>::iterator it = target.begin(); it != target.end(); ++it)

{

const char\* temp = typeid(\*\*it).name();

if (!strcmp(temp, typeB))

{

printf(typeB);

printf("\n");

printf("Original Balance：%lf\n",(\*it)->getBalance());

printf("Owed Interest：%lf\n",(\*it)->calculateInterest());

(\*it)->credit((\*it)->calculateInterest());

printf("Current Balance：%lf\n", (\*it)->getBalance());

}

else

{

printf(typeA);

printf("\n");

printf("Current Balance：%lf\n", (\*it)->getBalance());

}

}

}

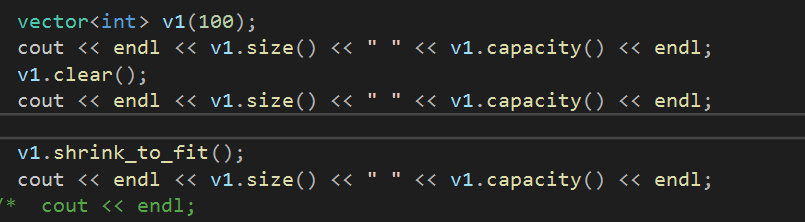
问题与思考：

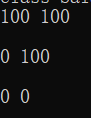
对类型的判断：

回头做第一题的时候发现了typeinfo这个头文件，看到里面用到了typeid，了解到其中name()函数可以用来来返回类型名，本次作业采用了这种方法处理类型判断，但也可以像EX1中转换指针判断nullptr来进行类型判断。

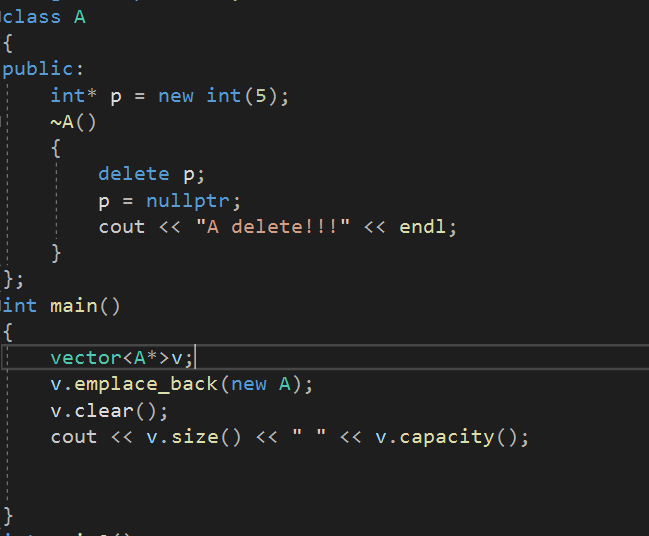
EXTRA（一些小发现）：

Vector的清空：调用clear()（将size变为0）后调用shrink\_to\_fit()（C++11）（将capacity调至与size同等大小）不过取决于编译器实现是否释放内存，另一种利用拷贝构造与临时对象生命周期的方法课上已经讲述，这里不再赘述。

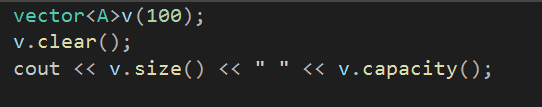


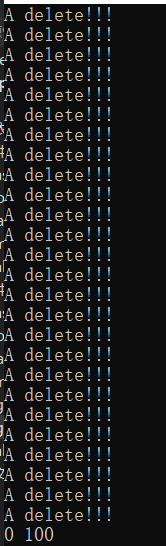


注：若为指向堆中内存的指针作为成员，需要for循环delete（不改变size）后clear，此时clear不会调用她的析构

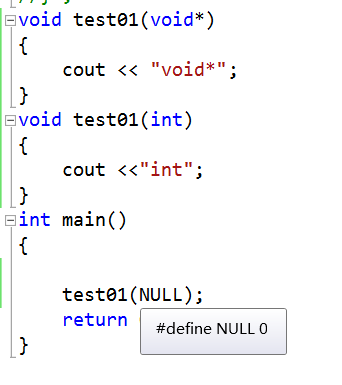






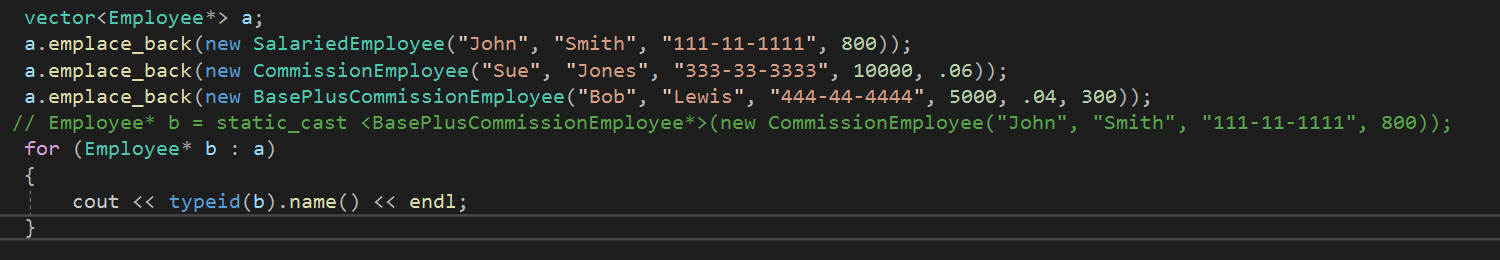


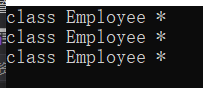
C++中NULL与nullptr的差异（vs2010支持此特性）：C++NULL为宏定义可视作0（应区分于C语言中void\*（0）的宏定义），因为C++（强类型语言）中不支持（void\*）隐式转换为别的类型指针，所以宏定义实际上为0（不妨尝试如下函数重载），而nullptr才是C++中真正的“NULL”：



可参考此文：<https://blog.csdn.net/qq_18108083/article/details/84346655>

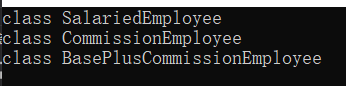
Typeinfo的使用：C++ typeinfo头文件中定义了typeid,使用其name()函数可以获得对应类型



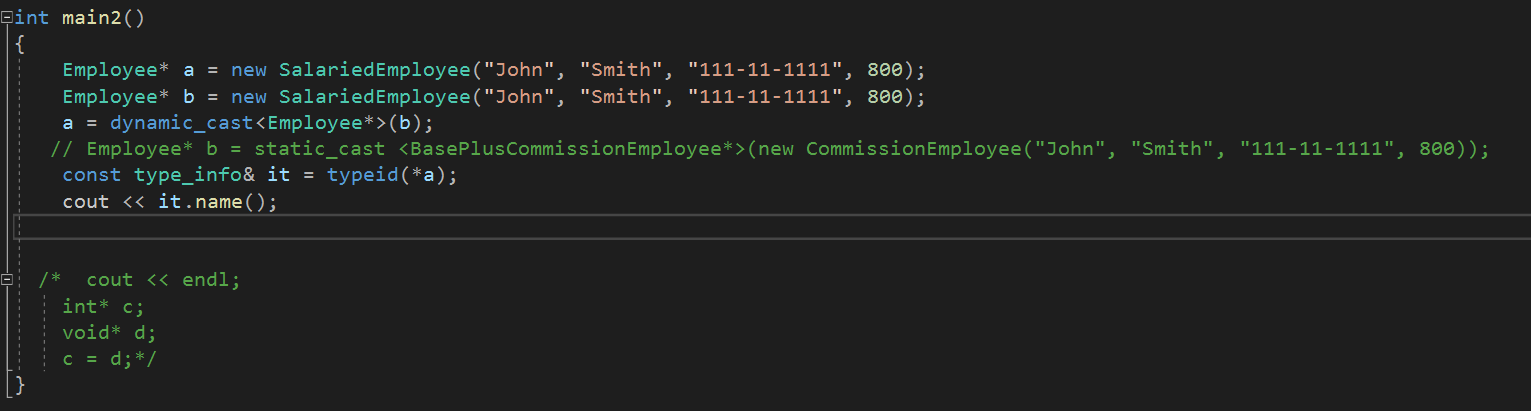
a中均为Employee\*——正确

改为解引用，看看指向的对象



支持多态，完全一致√

也能typeinfo定义对象



对于传引用的默认参数： 前加const修饰即可使用默认值