srand((unsigned)time(NULL)); // seed random-number generator

**一、STOCK例：**

**1.1、**// stock10.h ?Stock class declaration with constructors, destructor added

#ifndef STOCK1\_H\_

#define STOCK1\_H\_

#include <string>

class Stock

{

private:

std::string company;

long shares;

double share\_val;

double total\_val;

void set\_tot() { total\_val = shares \* share\_val; }

public:

Stock(); // default constructor

Stock(const std::string & co, long n = 0, double pr = 0.0);

~Stock(); // noisy destructor

void buy(long num, double price);

void sell(long num, double price);

void update(double price);

void show();

};

#endif

// stock20.cpp -- augmented version

#include <iostream>

#include "stock20.h"

using namespace std;

// constructors

/\*

Stock::Stock() // default constructor

{

shares = 0;

share\_val = 0.0;

total\_val = 0.0;

}

\*/

Stock::Stock(const std::string & co, long n, double pr)

{

company = co;

if (n < 0)

{

std::cout << "Number of shares can't be negative; "

<< company << " shares set to 0.\n";

shares = 0;

}

else

shares = n;

share\_val = pr;

set\_tot();

}

// class destructor

Stock::~Stock() // quiet class destructor

{

}

// other methods

void Stock::buy(long num, double price)

{

if (num < 0)

{

std::cout << "Number of shares purchased can't be negative. "

<< "Transaction is aborted.\n";

}

else

{

shares += num;

share\_val = price;

set\_tot();

}

}

void Stock::sell(long num, double price)

{

using std::cout;

if (num < 0)

{

cout << "Number of shares sold can't be negative. "

<< "Transaction is aborted.\n";

}

else if (num > shares)

{

cout << "You can't sell more than you have! "

<< "Transaction is aborted.\n";

}

else

{

shares -= num;

share\_val = price;

set\_tot();

}

}

void Stock::update(double price)

{

share\_val = price;

set\_tot();

}

void Stock::show() const

{

using std::cout;

using std::ios\_base;

// set format to #.###

ios\_base::fmtflags orig =

cout.setf(ios\_base::fixed, ios\_base::floatfield);

std::streamsize prec = cout.precision(3);

cout << "Company: " << company

<< " Shares: " << shares << '\n';

cout << " Share Price: $" << share\_val;

// set format to #.##

cout.precision(2);

cout << " Total Worth: $" << total\_val << '\n';

// restore original format

cout.setf(orig, ios\_base::floatfield);

cout.precision(prec);

}

const Stock & Stock::topval(const Stock & s) const

{

if (s.total\_val > total\_val)

return s;

else

return \*this;

}

// usestok2.cpp -- using the Stock class

// compile with stock20.cpp

#include <iostream>

//#include "stock20.h"

const int STKS = 4;

int main()

{

{

// create an array of initialized objects

Stock stocks[STKS] = { Stock("NanoSmart", 12, 20.0), Stock("Boffo Objects", 200, 2.0), Stock("Monolithic Obelisks", 130, 3.25), Stock("Fleep Enterprises", 60, 6.5) };

std::cout << "Stock holdings:\n";

int st;

for (st = 0; st < STKS; st++)

stocks[st].show();

// set pointer to first element

const Stock \* top = &stocks[0];

for (st = 1; st < STKS; st++)

top = &top->topval(stocks[st]);

// now top points to the most valuable holding

std::cout << "\nMost valuable holding:\n";

top->show();

}

// std::cin.get();

system("pause");

return 0;

}

**2.2、Stock：**

**// stock20.h -- augmented version**

**#ifndef STOCK20\_H\_**

**#define STOCK20\_H\_**

**#include <string>**

**class Stock**

**{**

**private:**

**std::string company;**

**int shares;**

**double share\_val;**

**double total\_val;**

**void set\_tot() { total\_val = shares \* share\_val; }**

**public:**

**// Stock(); // default constructor**

**Stock(const std::string & co, long n = 0, double pr = 0.0);**

**~Stock(); // do-nothing destructor**

**void buy(long num, double price);**

**void sell(long num, double price);**

**void update(double price);**

**void show()const;**

**const Stock & topval(const Stock & s) const;**

**};**

**#endif**

// stock20.cpp -- augmented version

#include <iostream>

#include "stock20.h"

using namespace std;

// constructors

/\*

Stock::Stock() // default constructor

{

shares = 0;

share\_val = 0.0;

total\_val = 0.0;

}

\*/

Stock::Stock(const std::string & co, long n, double pr)

{

company = co;

if (n < 0)

{

std::cout << "Number of shares can't be negative; "

<< company << " shares set to 0.\n";

shares = 0;

}

else

shares = n;

share\_val = pr;

set\_tot();

}

// class destructor

Stock::~Stock() // quiet class destructor

{

}

// other methods

void Stock::buy(long num, double price)

{

if (num < 0)

{

std::cout << "Number of shares purchased can't be negative. "

<< "Transaction is aborted.\n";

}

else

{

shares += num;

share\_val = price;

set\_tot();

}

}

void Stock::sell(long num, double price)

{

using std::cout;

if (num < 0)

{

cout << "Number of shares sold can't be negative. "

<< "Transaction is aborted.\n";

}

else if (num > shares)

{

cout << "You can't sell more than you have! "

<< "Transaction is aborted.\n";

}

else

{

shares -= num;

share\_val = price;

set\_tot();

}

}

void Stock::update(double price)

{

share\_val = price;

set\_tot();

}

void Stock::show() const

{

using std::cout;

using std::ios\_base;

// set format to #.###

ios\_base::fmtflags orig =

cout.setf(ios\_base::fixed, ios\_base::floatfield);

std::streamsize prec = cout.precision(3);

cout << "Company: " << company

<< " Shares: " << shares << '\n';

cout << " Share Price: $" << share\_val;

// set format to #.##

cout.precision(2);

cout << " Total Worth: $" << total\_val << '\n';

// restore original format

cout.setf(orig, ios\_base::floatfield);

cout.precision(prec);

}

const Stock & Stock::topval(const Stock & s) const

{

if (s.total\_val > total\_val)

return s;

else

return \*this;

}

// usestok2.cpp -- using the Stock class

// compile with stock20.cpp

#include <iostream>

//#include "stock20.h"

const int STKS = 4;

int main()

{

{

// create an array of initialized objects

Stock stocks[STKS] = { Stock("NanoSmart", 12, 20.0), Stock("Boffo Objects", 200, 2.0), Stock("Monolithic Obelisks", 130, 3.25), Stock("Fleep Enterprises", 60, 6.5) };

std::cout << "Stock holdings:\n";

int st;

for (st = 0; st < STKS; st++)

stocks[st].show();

// set pointer to first element

const Stock \* top = &stocks[0];

for (st = 1; st < STKS; st++)

top = &top->topval(stocks[st]);(不可以改）

// now top points to the most valuable holding

std::cout << "\nMost valuable holding:\n";

top->show();

}

// std::cin.get();

system("pause");

return 0;

}

# 名称空间：

// namesp.h

#ifndef NAMESP\_H\_

#define NAMESP\_H\_

#include <string>

// create the pers and debts namespaces

namespace pers

{

struct Person

{

std::string fname;

std::string lname;

};

void getPerson(Person &);

void showPerson(const Person &);

}

namespace debts

{

using namespace pers;

struct Debt

{

pers::Person name;

double amount;

};

void getDebt(Debt &);

void showDebt(const Debt &);

double sumDebts(const Debt ar[], int n);

}

// namesp.cpp -- namespaces

#include <iostream>

#include "namesp.h"

namespace pers

{

using std::cout;

using std::cin;

void getPerson(Person & rp)

{

cout << "Enter first name: ";

cin >> rp.fname;

cout << "Enter last name: ";

cin >> rp.lname;

}

void showPerson(const Person & rp)

{

std::cout << rp.lname << ", " << rp.fname;

}

}

namespace debts

{

void getDebt(Debt & rd)

{

getPerson(rd.name);

std::cout << "Enter debt: ";

std::cin >> rd.amount;

}

void showDebt(const Debt & rd)

{

showPerson(rd.name);

std::cout <<": $" << rd.amount << std::endl;

}

double sumDebts(const Debt ar[], int n)

{

double total = 0;

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

total += ar[i].amount;

return total;

}

}

// usenmsp.cpp -- using namespaces

#include <iostream>

//#include "namesp.h"

void other(void);

void another(void);

int main(void)

{

using debts::Debt;

using debts::showDebt;

Debt golf = { {"Benny", "Goatsniff"}, 120.0 };

showDebt(golf);

other();

another();

// std::cin.get();

// std::cin.get();

return 0;

}

void other(void)

{

using std::cout;

using std::endl;

using namespace debts;

debts::Person dg = {"Doodles", "Glister"};

showPerson(dg);

cout << endl;

Debt zippy[3];

int i;

for (i = 0; i < 3; i++)

getDebt(zippy[i]);

for (i = 0; i < 3; i++)

showDebt(zippy[i]);

cout << "Total debt: $" << sumDebts(zippy, 3) << endl;

return;

}

void another(void)

{

using pers::Person;;

Person collector = { "Milo", "Rightshift" };

pers::showPerson(collector);

std::cout << std::endl;

}

# 三、TIME实例：

**3.1、原先的**

// mytime0.h -- Time class before operator overloading

#ifndef MYTIME0\_H\_

#define MYTIME0\_H\_

class Time

{

private:

int hours;

int minutes;

public:

Time();

Time(int h, int m = 0);

void AddMin(int m);

void AddHr(int h);

void Reset(int h = 0, int m = 0);

const Time Sum(const Time & t) const;

void Show() const;

};

#endif

// mytime0.cpp -- implementing Time methods

#include <iostream>

#include "mytime0.h"

Time::Time()

{

hours = minutes = 0;

}

Time::Time(int h, int m )

{

hours = h;

minutes = m;

}

void Time::AddMin(int m)

{

minutes += m;

hours += minutes / 60;

minutes %= 60;

}

void Time::AddHr(int h)

{

hours += h;

}

void Time::Reset(int h, int m)

{

hours = h;

minutes = m;

}

const Time Time::Sum(const Time & t) const

{

Time sum;

sum.minutes = minutes + t.minutes;

sum.hours = hours + t.hours + sum.minutes / 60;

sum.minutes %= 60;

return sum;

}

void Time::Show() const

{

std::cout << hours << " hours, " << minutes << " minutes";

}

// usetime0.cpp -- using the first draft of the Time class

// compile usetime0.cpp and mytime0.cpp together

#include <iostream>

#include "mytime0.h"

int main()

{

using std::cout;

using std::endl;

Time planning;

Time coding(2, 40);

Time fixing(5, 55);

Time total;

cout << "planning time = ";

planning.Show();

cout << endl;

cout << "coding time = ";

coding.Show();

cout << endl;

cout << "fixing time = ";

fixing.Show();

cout << endl;

total = coding.Sum(fixing);

cout << "coding.Sum(fixing) = ";

total.Show();

cout << endl;

// std::cin.get();

return 0;

}

**3.2、友元函数**

// mytime3.h -- Time class with friends

#ifndef MYTIME3\_H\_

#define MYTIME3\_H\_

#include <iostream>

class Time

{

private:

int hours;

int minutes;

public:

Time();

Time(int h, int m = 0);

void AddMin(int m);

void AddHr(int h);

void Reset(int h = 0, int m = 0);

Time operator+(const Time & t) const;

Time operator-(const Time & t) const;

Time operator\*(double n) const;

friend Time operator\*(double m, const Time & t)

{ return t \* m; } // inline definition

friend std::ostream & operator<<(std::ostream & os, const Time & t);

};

#endif

// mytime3.cpp -- implementing Time methods

#include "mytime3.h"

Time::Time()

{

hours = minutes = 0;

}

Time::Time(int h, int m )

{

hours = h;

minutes = m;

}

void Time::AddMin(int m)

{

minutes += m;

hours += minutes / 60;

minutes %= 60;

}

void Time::AddHr(int h)

{

hours += h;

}

void Time::Reset(int h, int m)

{

hours = h;

minutes = m;

}

Time Time::operator+(const Time & t) const

{

Time sum;

sum.minutes = minutes + t.minutes;

sum.hours = hours + t.hours + sum.minutes / 60;

sum.minutes %= 60;

return sum;

}

Time Time::operator-(const Time & t) const

{

Time diff;

int tot1, tot2;

tot1 = t.minutes + 60 \* t.hours;

tot2 = minutes + 60 \* hours;

diff.minutes = (tot2 - tot1) % 60;

diff.hours = (tot2 - tot1) / 60;

return diff;

}

Time Time::operator\*(double mult) const

{

Time result;

long totalminutes = hours \* mult \* 60 + minutes \* mult;

result.hours = totalminutes / 60;

result.minutes = totalminutes % 60;

return result;

}

std::ostream & operator<<(std::ostream & os, const Time & t)

{

os << t.hours << " hours, " << t.minutes << " minutes";

return os;

}

//usetime3.cpp -- using the fourth draft of the Time class

// compile usetime3.cpp and mytime3.cpp together

#include <iostream>

#include "mytime3.h"

int main()

{

using std::cout;

using std::endl;

Time aida(3, 35);

Time tosca(2, 48);

Time temp;

cout << "Aida and Tosca:\n";

cout << aida<<"; " << tosca << endl;

temp = aida + tosca; // operator+()

cout << "Aida + Tosca: " << temp << endl;

temp = aida\* 1.17; // member operator\*()

cout << "Aida \* 1.17: " << temp << endl;

cout << "10.0 \* Tosca: " << 10.0 \* tosca << endl;

// std::cin.get();

return 0;

}

# 矢量类：

// vect.h -- Vector class with <<, mode state

#ifndef VECTOR\_H\_

#define VECTOR\_H\_

#include <iostream>

namespace VECTOR

{

class Vector

{

public:

enum Mode {RECT, POL};

// RECT for rectangular, POL for Polar modes

private:

double x; // horizontal value

double y; // vertical value

double mag; // length of vector

double ang; // direction of vector in degrees

Mode mode; // RECT or POL

// private methods for setting values

void set\_mag();

void set\_ang();

void set\_x();

void set\_y();

public:

Vector();

Vector(double n1, double n2, Mode form = RECT);

void reset(double n1, double n2, Mode form = RECT);

~Vector();

double xval() const {return x;} // report x value

double yval() const {return y;} // report y value

double magval() const {return mag;} // report magnitude

double angval() const {return ang;} // report angle

void polar\_mode(); // set mode to POL

void rect\_mode(); // set mode to RECT

// operator overloading

Vector operator+(const Vector & b) const;

Vector operator-(const Vector & b) const;

Vector operator-() const;

Vector operator\*(double n) const;

// friends

friend Vector operator\*(double n, const Vector & a);

friend std::ostream & operator<<(std::ostream & os, const Vector & v);

};

} // end namespace VECTOR

#endif

// vect.cpp -- methods for the Vector class

#include <cmath>

#include "vect.h" // includes <iostream>

using std::sqrt;

using std::sin;

using std::cos;

using std::atan;

using std::atan2;

using std::cout;

namespace VECTOR

{

// compute degrees in one radian

const double Rad\_to\_deg = 45.0 / atan(1.0);

// should be about 57.2957795130823

// private methods

// calculates magnitude from x and y

void Vector::set\_mag()

{

mag = sqrt(x \* x + y \* y);

}

void Vector::set\_ang()

{

if (x == 0.0 && y == 0.0)

ang = 0.0;

else

ang = atan2(y, x);

}

// set x from polar coordinate

void Vector::set\_x()

{

x = mag \* cos(ang);

}

// set y from polar coordinate

void Vector::set\_y()

{

y = mag \* sin(ang);

}

// public methods

Vector::Vector() // default constructor

{

x = y = mag = ang = 0.0;

mode = RECT;

}

// construct vector from rectangular coordinates if form is r

// (the default) or else from polar coordinates if form is p

Vector::Vector(double n1, double n2, Mode form)

{

mode = form;

if (form == RECT)

{

x = n1;

y = n2;

set\_mag();

set\_ang();

}

else if (form == POL)

{

mag = n1;

ang = n2 / Rad\_to\_deg;

set\_x();

set\_y();

}

else

{

cout << "Incorrect 3rd argument to Vector() -- ";

cout << "vector set to 0\n";

x = y = mag = ang = 0.0;

mode = RECT;

}

}

// reset vector from rectangular coordinates if form is

// RECT (the default) or else from polar coordinates if

// form is POL

void Vector:: reset(double n1, double n2, Mode form)

{

mode = form;

if (form == RECT)

{

x = n1;

y = n2;

set\_mag();

set\_ang();

}

else if (form == POL)

{

mag = n1;

ang = n2 / Rad\_to\_deg;

set\_x();

set\_y();

}

else

{

cout << "Incorrect 3rd argument to Vector() -- ";

cout << "vector set to 0\n";

x = y = mag = ang = 0.0;

mode = RECT;

}

}

Vector::~Vector() // destructor

{

}

void Vector::polar\_mode() // set to polar mode

{

mode = POL;

}

void Vector::rect\_mode() // set to rectangular mode

{

mode = RECT;

}

// operator overloading

// add two Vectors

Vector Vector::operator+(const Vector & b) const

{

return Vector(x + b.x, y + b.y);

}

// subtract Vector b from a

Vector Vector::operator-(const Vector & b) const

{

return Vector(x - b.x, y - b.y);

}

// reverse sign of Vector

Vector Vector::operator-() const

{

return Vector(-x, -y);

}

// multiply vector by n

Vector Vector::operator\*(double n) const

{

return Vector(n \* x, n \* y);

}

// friend methods

// multiply n by Vector a

Vector operator\*(double n, const Vector & a)

{

return a \* n;

}

// display rectangular coordinates if mode is RECT,

// else display polar coordinates if mode is POL

std::ostream & operator<<(std::ostream & os, const Vector & v)

{

if (v.mode == Vector::RECT)

os << "(x,y) = (" << v.x << ", " << v.y << ")";

else if (v.mode == Vector::POL)

{

os << "(m,a) = (" << v.mag << ", "

<< v.ang \* Rad\_to\_deg << ")";

}

else

os << "Vector object mode is invalid";

return os;

}

} // end namespace VECTOR

// randwalk.cpp -- using the Vector class

// compile with the vect.cpp file

#include <iostream>

#include <cstdlib> // rand(), srand() prototypes

#include <ctime> // time() prototype

#include "vect.h"

int main()

{

using namespace std;

using VECTOR::Vector;

srand((unsigned)time(NULL)); // seed random-number generator

double direction;

Vector step;

Vector result(0.0, 0.0);

unsigned long steps = 0;

double target;

double dstep;

cout << "Enter target distance (q to quit): ";

while (cin >> target)

{

cout << "Enter step length: ";

if (!(cin >> dstep))

break;

while (result.magval() < target)

{

direction = rand() % 360;

step.reset(dstep, direction, Vector::POL);

result = result + step;

steps++;

}

cout << "After " << steps << " steps, the subject "

"has the following location:\n";

cout << result << endl;

result.polar\_mode();

cout << " or\n" << result << endl;

cout << "Average outward distance per step = "

<< result.magval()/steps << endl;

steps = 0;

result.reset(0.0, 0.0);

cout << "Enter target distance (q to quit): ";

}

cout << "Bye!\n";

/\* keep window open

cin.clear();

while (cin.get() != '\n')

continue;

cin.get();

\*/

return 0;

}