## 第2章 C++简单程序设计

**例2-1 一个简单的C++程序**

#include <iostream>

using namespace std;

int main()

{

cout << "Hello, World!" << endl;

return 0;

}

**例2-2 输入一个年份，判断是否闰年**

#include <iostream>

using namespace std;

int main()

{

bool IsLeapYear(int y);

int year;

cout << "Please enter the year: ";

cin >> year;

if (IsLeapYear(year))

cout << year << " is leap year.";

else

cout << year << " is not leap year.";

cout << endl;

return 0;

}

bool IsLeapYear(int y)

{

return ((y % 4 == 0 && y % 100 != 0) || y % 400 == 0);

}

**例2-3 比较两个数的大小。**

#include <iostream>

using namespace std;

int main()

{

int x, y;

cout << "Enter x and y separated by spaces: ";

cin >> x >> y;

if (x > y)

cout << x << ">" << y << endl;

else if (x < y)

cout << x << "<" << y << endl;

else

cout << x << "=" << y << endl;

return 0;

}

**例2-4 输入一个0~6之间的整数，转换成星期输出。**

#include <iostream>

using namespace std;

int main()

{

int day;

cout<<"Please enter an integer(0 - 6): ";

cin >> day;

switch (day) {

case 0:

cout << "Sunday!" << endl;

break;

case 1:

cout << "Monday!" << endl;

break;

case 2:

cout << "Tuesday!" << endl;

break;

case 3:

cout << "Wednesday!" << endl;

break;

case 4:

cout << "Thursday!" << endl;

break;

case 5:

cout << "Friday!" << endl;

break;

case 6:

cout << "Saturday!" << endl;

break;

default:

break;

}

return 0;

}

**例2-5 求自然数1~10之和。**

#include <iostream>

using namespace std;

int main()

{

int i;

int sum = 0;

for (i = 1; i <= 10; i++)

sum += i;

cout << "sum = " << sum << endl;

return 0;

}

**例2-6 输入一个整数，将各位数字反转后输出。**

#include <iostream>

using namespace std;

int main()

{

int num;

cout<< "Please enter an integer: ";

cin >> num;

while (num / 10 != 0)

{

cout << num % 10 << " ";

num = num / 10;

}

cout << num << endl;

return 0;

}

**例2-7 用do…while语句编程，求自然数1~10之和。**

#include <iostream>

using namespace std;

int main()

{

int i = 1;

int sum = 0;

do {

sum += i;

i++;

} while (i <= 10);

cout << "sum = " << sum << endl;

return 0;

}

**例2-8 输入一个整数，求出它的所有因子。**

#include <iostream>

using namespace std;

int main()

{

int m, n;

int i;

cout<< "Please enter an integer: ";

cin >> n;

m = n / 2;

for (i = 1; i <= m; i++)

if (n % i == 0)

cout << i << " ";

return 0;

}

**例2-9 编写程序输出图案**

#include <iostream>

using namespace std;

int main()

{

const int N = 4;

int i, j;

for (i = 1; i <= N; i++) {

for (j = 1; j <= 30; j++)

cout << ' ';

for (j = 1; j <= 8 - 2 \* i; j++)

cout << ' ';

for (j = 1; j <= 2 \* i - 1; j++)

cout << '\*';

cout << endl;

}

for (i = 1; i <= N - 1; i++)

{

for (j = 1; j <= 30; j++)

cout << ' ';

for (j = 1; j <= 7 - 2 \* i; j++)

cout << '\*';

cout << endl;

}

return 0;

}

**例2-10 读入一系列整数，统计出正整数个数i和负整数个数，读入0则结束**

#include <iostream>

using namespace std;

int main()

{

int i = 0, j = 0, n;

cout << "Enter some integers please (enter 0 to quit):";

cin >> n;

while (n != 0)

{

if (n > 0)

i += 1;

if (n < 0)

j += 1;

cout << "next num (enter 0 to quit): ";

cin >> n;

}

cout << endl;

cout << "Count of positive integers:" << i << endl;

cout << "Count of negative integers:" << j << endl;

return 0;

}

**例2-11 设某次体育比赛的结果有4种可能：胜(WIN)、负(LOSE)、平局(TIE)、比赛取消(CANCEL)，编写程序顺序输出这4种情况**

#include <iostream>

using namespace std;

int main()

{

enum GameResult { WIN, LOSE, TIE, CANCEL };

enum GameResult res;

enum GameResult omit = CANCEL;

int count;

for (count = WIN; count <= CANCEL; count++)

{

res = GameResult(count);

if (res == omit)

cout << "The game was cancelled" << endl;

else

{

cout << "The game was played";

if (res == WIN)

cout << "and we won!";

if (res == LOSE)

cout << "and we lost.";

cout << endl;

}

}

return 0;

}

## 第3章 函数

**例3-1 求x的n次方**

#include <iostream>

using namespace std;

int main()

{

double power(double x, int n);

cout << "5 to the power 2 is " << power(5, 2) << endl;

return 0;

}

double power(double x, int n)

{

double val = 1.0;

while (n--)

val \*= x;

return val;

}

**例3-2 输入一个8位二进制数，将其转换为十进制数输出**

#include <iostream>

using namespace std;

int main()

{

double power(double x, int n);

int value = 0;

int i;

char ch;

cout << "Enter an 8 bit binary number:";

for (i = 7; i >= 0; i--)

{

cin >> ch;

if (ch == '1')

value += static\_cast<int>(power(2, i));

}

cout << "Decimal value is " << value << endl;

return 0;

}

double power(double x, int n)

{

double val = 1.0;

while (n--)

val \*= x;

return val;

}

**例3-3 编写程序求π的值**

#include <iostream>

using namespace std;

int main()

{

double arctan(double x);

double a = 16.0 \* arctan(1 / 5.0);

double b = 4.0 \* arctan(1 / 239.0);

cout << "PI = " << a - b << endl;

return 0;

}

double arctan(double x)

{

double sqr = x\*x;

double e = x;

double r = 0;

int i = 1;

double f;

while (e / i > 1e-15)

{

f = e / i;

r = (i % 4 == 1) ? r + f : r - f;

e = e \* sqr;

i += 2;

}

return r;

}

**例3-4 寻找并输出11~999之间的数m，它满足m，m2，和m3均为回文数。**

#include <iostream>

using namespace std;

int main()

{

bool symm(unsigned n);

for(unsigned m=11;m<1000;m++)

if (symm(m) && symm(m \* m) && symm(m \* m \* m))

{

cout << "m=" << m;

cout << " m\*m=" << m \* m;

cout << " m\*m\*m=" << m \* m \* m << endl;

}

return 0;

}

bool symm(unsigned n)

{

unsigned i = n;

unsigned m = 0;

while (i > 0)

{

m = m \* 10 + i % 10;

i /= 10;

}

return m == n;

}

**例3-5 计算如下公式，并输出结果。**



**其中r，s的值由键盘输入。sinx的近似值按如下公式计算：**

****

**计算精度为10-6，当某项的绝对值小于计算精度时，停止累加，累加和即为该精度下的sinx近似值。**

#include <iostream>

#include <cmath>

using namespace std;

const double TINY\_VALUE = 1e-10;

int main()

{

double tsin(double x);

double k, r, s;

cout << "r=";

cin >> r;

cout << "s=";

cin >> s;

if (r \* r <= s \* s)

k = sqrt(tsin(r) \* tsin(r) + tsin(s) \* tsin(s));

else

k = tsin(r \* s) / 2;

cout << k << endl;

return 0;

}

double tsin(double x)

{

double g = 0;

double t = x;

int n = 1;

do {

g += t;

n++;

t = -t \* x \* x / (2 \* n - 1) / (2 \* n - 2);

} while (fabs(t) >= TINY\_VALUE);

return g;

}

**例3-6 投骰子的随机游戏。**

**游戏规则是：每个骰子有6面，点数分别为1，2，3，4，5，6。游戏者在程序开始时输入一个无符号整数，作为产生随机数的种子。**

**每轮投两次骰子，第一轮如果和数为7或11则为胜，游戏结束；和数为2，3或12则为负，游戏结束；和数为其他值则将此值作为自己的点数，继续第二轮、第三轮…直到某轮的和数等于点数则取胜，若在此前出现和数为7则负。**

**由rollDice函数负责模拟投骰子、计算和数并输出和数。**

#include <iostream>

#include <cstdlib>

using namespace std;

enum GameStatus{WIN, LOSE, PLAYING};

int main()

{

int rollDice();

int sum, myPoint;

GameStatus status;

unsigned seed;

cout << "Please enter an unsigned integer:";

cin >> seed;

srand(seed);

sum = rollDice();

switch (sum)

{

case 7:

case 11:

status = WIN;

break;

case 2:

case 3:

case 12:

status = LOSE;

break;

default:

status = PLAYING;

myPoint = sum;

cout << "point is " << myPoint << endl;

break;

}

while (status==PLAYING)

{

sum = rollDice();

if (sum == myPoint)

status = WIN;

else if (sum == 7)

status = LOSE;

}

if (status == WIN)

cout << "player wins" << endl;

else

cout << "player loses" << endl;

return 0;

}

int rollDice()

{

int die1 = 1 + rand() % 6;

int die2 = 1 + rand() % 6;

int sum = die1 + die2;

cout << "player rolled " << die1 << "+" << die2 << "=" << sum << endl;

return sum;

}

**例3-7 输入两个整数，求它们的平方和。**

#include <iostream>

using namespace std;

int fun2(int m);

int fun1(int x, int y);

int main()

{

int a, b;

cout << "Please enter two integers(a and b):";

cin >> a >> b;

cout << "The sum of square of a and b: " << fun1(a, b) << endl;

return 0;

}

int fun2(int m)

{

return m \* m;

}

int fun1(int x, int y)

{

return fun2(x) + fun2(y);

}

**例3-8 求n!。**

#include <iostream>

using namespace std;

int main()

{

unsigned fac(unsigned n);

unsigned n;

cout << "Enter a positive integer:";

cin >> n;

unsigned y = fac(n);

cout << n << "!=" << y << endl;

return 0;

}

unsigned fac(unsigned n)

{

unsigned f;

if (n == 0)

f = 1;

else

f = fac(n - 1) \* n;

return f;

}

**例3-9 用递归法计算从n个人中选择k个人组成一个委员会的不同组合数。**

#include <iostream>

using namespace std;

int main()

{

int comm(int n, int k);

int n, k;

cout << "Please enter two integers n and k:";

cin >> n >> k;

cout << "C(n,k)=" << comm(n, k) << endl;

return 0;

}

int comm(int n, int k)

{

if (k > n)

return 0;

else if (n == k || k == 0)

return 1;

else

return comm(n - 1, k) + comm(n - 1, k - 1);

}

**例3-10 汉诺塔问题。**

#include <iostream>

using namespace std;

void move(char src, char dest);

void hanoi(int n, char src, char medium, char dest);

int main()

{

int m;

cout << "Enter the number of diskes:";

cin >> m;

cout << "the steps to moving " << m << " diskes:" << endl;

hanoi(m, 'A', 'B', 'C');

return 0;

}

void move(char src, char dest)

{

cout << src << "-->" << dest << endl;

}

void hanoi(int n, char src, char medium, char dest)

{

if (n == 1)

move(src, dest);

else

{

hanoi(n - 1, src, dest, medium);

move(src, dest);

hanoi(n - 1, medium, src, dest);

}

}

**例3-11 将两个整数交换次序后输出。**

#include <iostream>

using namespace std;

int main()

{

void swap(int a, int b);

int x = 5, y = 10;

cout << "x=" << x << " y=" << y << endl;

swap(x, y);

cout << "x=" << x << " y=" << y << endl;

return 0;

}

void swap(int a, int b)

{

int t = a;

a = b;

b = t;

}

**例3-12 使用引用传递改写例3-11的程序，使两整数成功地进行交换。**

#include <iostream>

using namespace std;

int main()

{

void swap(int &a, int &b);

int x = 5, y = 10;

cout << "x=" << x << " y=" << y << endl;

swap(x, y);

cout << "x=" << x << " y=" << y << endl;

return 0;

}

void swap(int &a, int &b)

{

int t = a;

a = b;

b = t;

}

**例3-13 值传递与引用传递的比较。**

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

void fiddle(int in1, int& in2);

int v1 = 7, v2 = 12;

cout << "The values are ";

cout << setw(5) << v1;

cout << setw(5) << v2 << endl;

fiddle(v1, v2);

cout << "The values are ";

cout << setw(5) << v1;

cout << setw(5) << v2 << endl;

return 0;

}

void fiddle(int in1, int& in2)

{

in1 = in1 + 100;

in2 = in2 + 100;

cout << "The values are ";

cout << setw(5) << in1;

cout << setw(5) << in2 << endl;

}

**例3-14 内联函数应用举例。**

#include <iostream>

using namespace std;

const double PI = 3.141592665358979;

inline double calArea(double radius)

{

return PI \* radius \* radius;

}

int main()

{

double r = 3.0;

double area = calArea(r);

cout << area << endl;

return 0;

}

**例3-15 带默认形参值的函数举例。**

#include <iostream>

#include <iomanip>

using namespace std;

int getVolume(int length, int width = 2, int height = 3);

int main()

{

const int X = 10, Y = 12, Z = 15;

cout << "Some box data is ";

cout << getVolume(X, Y, Z) << endl;

cout << "Some box data is ";

cout << getVolume(X, Y) << endl;

cout << "Some box data is ";

cout << getVolume(X) << endl;

return 0;

}

int getVolume(int length, int width /\*= 2\*/, int height /\*= 3\*/)

{

cout << setw(5) << length << setw(5) << width << setw(5) << height << '\t';

return length \* width \* height;

}

**例3-16 重载函数应用举例。**

#include <iostream>

#include <iomanip>

using namespace std;

int sumOfSquare(int a, int b)

{

return a \* a + b \* b;

}

double sumOfSquare(double a, double b)

{

return a \* a + b \* b;

}

int main()

{

int m, n;

cout << "Enter two integer: ";

cin >> m >> n;

cout << "Their sum of square: " << sumOfSquare(m, n) << endl;

double x, y;

cout << "Enter two real number: ";

cin >> x >> y;

cout << "Their sum of square: " << sumOfSquare(x, y) << endl;

return 0;

}

**例3-17系统函数应用举例。**

#include <iostream>

#include <cmath>

using namespace std;

const double PI = 3.141592665358979;

int main()

{

double angle;

cout << "Please enter an angle: ";

cin >> angle;

double radian = angle \* PI / 180;

cout << "sin(" << angle << ")=" << sin(radian) << endl;

cout << "cos(" << angle << ")=" << cos(radian) << endl;

cout << "tan(" << angle << ")=" << tan(radian) << endl;

return 0;

}

## 第4章 类与对象

**例4-1 时钟程序。**

class Clock

{

int hour;

int minute;

int second;

public:

void setTime(int newH=0, int newM = 0, int newS = 0);

void showTime();

};

#include "Clock.h"

#include <iostream>

void Clock::setTime(int newH, int newM, int newS)

{

// TODO: 在此处添加实现代码.

hour = newH;

minute = newM;

second = newS;

}

void Clock::showTime()

{

// TODO: 在此处添加实现代码.

std::cout << hour << ": " << minute << ": " << second << std::endl;

}

#include <iostream>

#include "Clock.h"

int main()

{

Clock myClock;

std::cout << "First time set and output: " << std::endl;

myClock.setTime();

myClock.showTime();

std::cout << "Second time set and output: " << std::endl;

myClock.setTime(8, 30, 30);

myClock.showTime();

return 0;

}

**例4-2 Point类程序。**

class Point

{

int x;

int y;

public:

Point(int xx = 0, int yy = 0);

Point(Point &p);

int getX();

int getY();

};

#include "Point.h"

#include <iostream>

Point::Point(int xx, int yy)

{

x = xx;

y = yy;

}

Point::Point(Point &p)

{

x = p.x;

y = p.y;

std::cout << "Calling the copy constructor" << std::endl;

}

int Point::getX()

{

// TODO: 在此处添加实现代码.

return x;

}

int Point::getY()

{

// TODO: 在此处添加实现代码.

return y;

}

#include <iostream>

#include "Point.h"

void fun1(Point p);

Point fun2();

int main()

{

Point a(4, 5);

Point b = a;

std::cout << b.getX() << std::endl;

fun1(b);

b = fun2();

std::cout << b.getX() << std::endl;

return 0;

}

void fun1(Point p)

{

std::cout << p.getX() << std::endl;

}

Point fun2()

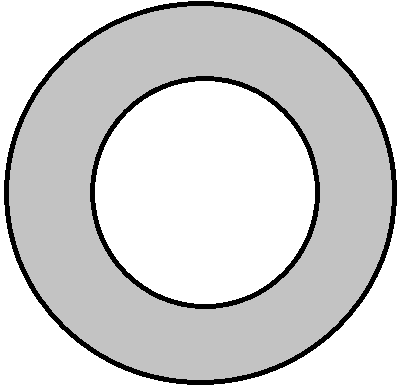
{

Point a(1, 2);

return a;

}

**例4-3 游泳池改造。一圆形游泳池如图所示，现在需在其周围建一圆形过道，并在其四周围上栅栏。栅栏价格为35元/米，过道造价为20元/平方米。过道宽度为3米，游泳池半径由键盘输入。要求编程计算并输出过道和栅栏的造价。**



const float PI = 3.1415926;

const float FENCE\_PRICE = 35;

const float CONCRETE\_PRICE = 20;

class Circle

{

private:

float radius;

public:

Circle(float r);

float circumference();

float area();

};

#include "Circle.h"

Circle::Circle(float r)

:radius(r)

{

}

float Circle::circumference()

{

return 2 \* PI \* radius;

}

float Circle::area()

{

return PI \* radius \* radius;

}

#include <iostream>

#include "circle.h"

int main()

{

float radius;

std::cout << "输入游泳池的半径：";

std::cin >> radius;

Circle pool(radius);

Circle poolRim(radius + 3);

float fenceCost = poolRim.circumference() \* FENCE\_PRICE;

std::cout << "栅栏的造价为￥" << fenceCost<<std::endl;

float concreteCost = (poolRim.area() - pool.area()) \* CONCRETE\_PRICE;

std::cout << "过道的造价为￥" << concreteCost << std::endl;

return 0;

}

**例4-4 类的组合，线段类。**

class Point

{

int x;

int y;

public:

Point(int x1, int y1);

Point(Point& obj);

int getx();

int gety();

};

#include "Point.h"

#include <iostream>

Point::Point(int x1, int y1)

:x(x1), y(y1)

{

}

Point::Point(Point& obj)

: x(obj.x), y(obj.y)

{

std::cout << "Calling the copy constructor of point" << std::endl;

}

int Point::getx()

{

// TODO: 在此处添加实现代码.

return x;

}

int Point::gety()

{

// TODO: 在此处添加实现代码.

return y;

}

#include "Point.h"

class Line

{

Point p0;

Point p1;

double len;

public:

Line(Point xp0, Point xp1);

Line(Line& obj);

int getLen();

};

#include "Line.h"

#include <iostream>

#include <cmath>

Line::Line(Point xp0, Point xp1)

:p0(xp0), p1(xp1)

{

std::cout << "Calling constructor of Line" << std::endl;

double x = static\_cast<double>(p0.getx() - p1.getx());

double y = static\_cast<double>(p0.gety() - p1.gety());

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

}

Line::Line(Line& obj)

:p0(obj.p0), p1(obj.p1)

{

std::cout << "Calling the copy constructor of Line" << std::endl;

len = obj.len;

}

int Line::getLen()

{

// TODO: 在此处添加实现代码.

return len;

}

#include <iostream>

#include "line.h"

int main(int argc, char\* argv[])

{

Point myp0(1, 1);

Point myp1(4, 5);

Line line1(myp0, myp1);

Line line2(line1);

std::cout << "The length of the line1 is:";

std::cout << line1.getLen() << std::endl;

std::cout << "The length of the line2 is:";

std::cout << line2.getLen() << std::endl;

return 0;

}

**例4-9 银行账户管理。**

class SavingsAccount

{

int id; // 账号

double balance; // 余额

double rate; // 利率

int lastDate; // 上次变更余额的日期

double accumulation; // 余额按日累加和

public:

SavingsAccount();

SavingsAccount(int id1, double rate1, int date);

int getId();

double getBalance();

double getRate();

void deposit(int date, double amount); // 存入现金

void withdraw(int date, double amount); // 取出现金

// 结算利息，每年1月1日调用一次该函数

void settle(int date);

void show(); // 显示账户信息

private:

// 记录一笔账，data为日期，amount为金额，desc为说明

void record(int date, double amount);

// 获得到指定日期为止的存款金额按日累积值

double accumulate(int date) const;

};

#include "SavingsAccount.h"

#include <iostream>

SavingsAccount::SavingsAccount()

{

}

SavingsAccount::SavingsAccount(int id1, double rate1, int date)

:id(id1), balance(0), rate(rate1), lastDate(date), accumulation(0)

{

std::cout << date << "\t#" << id << "is created" << std::endl;

}

int SavingsAccount::getId()

{

return id;

}

double SavingsAccount::getBalance()

{

return balance;

}

double SavingsAccount::getRate()

{

return rate;

}

void SavingsAccount::deposit(int date, double amount)

{

record(date, amount);

}

void SavingsAccount::withdraw(int date, double amount)

{

if (amount > balance)

std::cout << "Error: not enough money" << std::endl;

else

record(date, -amount);

}

void SavingsAccount::settle(int date)

{

double interest = accumulate(date) \* rate / 365;

if (interest != 0)

record(date, interest);

accumulation = 0;

}

void SavingsAccount::show()

{

std::cout << "#" << id << "\tBalance: " << balance;

}

void SavingsAccount::record(int date, double amount)

{

accumulation = accumulate(date);

lastDate = date;

amount = floor(amount \* 100 + 0.5) / 100;

balance += amount;

std::cout << date << "\t#" << id << "\t" << amount << "\t" << balance << std::endl;

}

double SavingsAccount::accumulate(int date) const

{

return accumulation + balance \* (date - lastDate);

}

#include <iostream>

#include "savingsaccount.h"

int main(int argc, char\* argv[])

{

SavingsAccount sa0(21325302, 0.015, 1);

SavingsAccount sa1(58320212, 0.015, 1);

sa0.deposit(5, 5000);

sa1.deposit(25, 10000);

sa0.deposit(45, 5500);

sa1.withdraw(60, 4000);

sa0.settle(90);

sa1.settle(90);

sa0.show();

std::cout << std::endl;

sa1.show();

std::cout << std::endl;

return 0;

}

## 第5章 数据的共享与保护

**例5-1 作用域实例。**

#include <iostream>

using namespace std;

int i;

namespace Ns {

    int j;

}

int main()

{

    i = 5;                  //为全局变量i赋值

    Ns::j = 6;              //为全局变量j赋值

    {

        using namespace Ns; //使得在当前模块中可以直接引用Ns命名空间的标识符

        int i;              //局部变量，局部作用域

        i = 7;

        cout << "i=" << i << endl;//输出7

        cout << "j=" << j << endl;//输出6

    }

    cout << "i=" << i << endl;    //输出5

    return 0;

}

**例5-2 变量的生命期。**

#include <iostream>

using namespace std;

int i = 1;

int main()

{

    void other();

    //a为静态局部变量，具有全局寿命，局部可见

    static int a;

    //b, c为局部变量，具有动态生存期

    int b = -10;

    int c = 90;

    cout << "---MAIN---" << endl;

    cout << "i=" << i << ", a=" << a << ", b=" << b << ", c=" << c << endl;

    c += 8;

    other();

    cout << "---MAIN---" << endl;

    cout << "i=" << i << ", a=" << a << ", b=" << b << ", c=" << c << endl;

    i += 10;

    other();

    return 0;

}

void other()

{

    //a, b为静态局部变量，具有全局寿命，局部可见，只第一次进入函数时被初始化

    static int a = 2;

    static int b;

    //c为局部变量，具有动态生存期，每次进入函数都初始化

    int c = 10;

    a += 2;

    i += 32;

    c += 5;

    cout << "---OTHER---" << endl;

    cout << "i=" << i << ", a=" << a << ", b=" << b << ",c=" << c << endl;

    b = a;

}

**例5-3 具有静态和动态生命期的时钟程序。**

#include<iostream>

using namespace std;

class Clock

{

public:

    Clock();

    void setTime(int newH, int newM, int newS);//函数原型作用域

    void showTime();

private:

    int hour;

    int minute;

    int second;

};

Clock::Clock()

:hour(0), minute(0), second(0)

{

}

void Clock::setTime(int newH, int newM, int newS)//局部作用域

{

    hour = newH;

    minute = newM;

    second = newS;

}

void Clock::showTime()

{

    cout << hour << ": " << minute << ": " << second << endl;

}

//具有静态生存期，全局变量，由默认构造函数初始化为0，0，0

Clock globClock;

int main()

{

    cout << "First time output:" << endl;

    //引用具有命名空间作用域的对象

    globClock.showTime();//显示0: 0: 0

    globClock.setTime(8, 30, 30);

    Clock myClock(globClock);//调用复制构造函数

    cout << "Second time output:" << endl;

    myClock.showTime();

    return 0;

}

**例5-4 静态数据成员Point。**

#include <iostream>

using namespace std;

class Point

{

private:

    int x;

    int y;

    static int count;//静态数据成员，用于记录点的个数

public:

    Point();

    Point(int x1, int y1);

    Point(Point& obj);

    ~Point();

    int getx();

    int gety();

    void showCount();

};

Point::Point()

:x(0), y(0)

{

    count++;

}

Point::Point(int x1, int y1)

:x(x1), y(y1)

{

    count++;

}

Point::Point(Point& obj)

:x(obj.x), y(obj.y)

{

    count++;

}

Point::~Point()

{

    count--;

}

int Point::getx()

{

    return x;

}

int Point::gety()

{

    return y;

}

void Point::showCount()

{

    cout << " Object count=" << count << endl;

}

int Point::count = 0;//静态数据成员定义和初始化，使用类名限定

int main(int argc, char\* argv[])

{

    Point a(4, 5);

    cout << "Point A:" << a.getx() << "," << a.gety() << endl;

    a.showCount();

    Point b(a);

    cout << "Point B:" << b.getx() << "," << b.gety() << endl;

    b.showCount();

    return 0;

}

**例5-5 静态数据成员和静态成员函数Point。**

#include <iostream>

using namespace std;

class Point

{

private:

    int x;

    int y;

    static int count;//静态数据成员，用于记录点的个数

public:

    Point();

    Point(int x1, int y1);

    Point(Point& obj);

    ~Point();

    int getx();

    int gety();

    static void showCount();

};

Point::Point()

:x(0), y(0)

{

    count++;

}

Point::Point(int x1, int y1)

:x(x1), y(y1)

{

    count++;

}

Point::Point(Point& obj)

:x(obj.x), y(obj.y)

{

    count++;

}

Point::~Point()

{

    count--;

}

int Point::getx()

{

    return x;

}

int Point::gety()

{

    return y;

}

void Point::showCount()

{

    cout << " Object count=" << count << endl;

}

int Point::count = 0;//静态数据成员定义和初始化，使用类名限定

int main(int argc, char\* argv[])

{

    Point a(4, 5);

    cout << "Point A:" << a.getx() << "," << a.gety() << endl;

    a.showCount();

    Point b(a);

    cout << "Point B:" << b.getx() << "," << b.gety() << endl;

    cout<<"Total object points are ";

    Point::showCount();

    return 0;

}

**例5-6 使用友元函数计算两点距离**

#include <iostream>

#include <cmath>

using namespace std;

class Point

{

public:

    Point();

    Point(int x1, int y1);

    int getx();

    int gety();

    friend float dist(Point& p1, Point& p2);

private:

    int x;

    int y;

};

Point::Point()

:x(0), y(0)

{

}

Point::Point(int x1, int y1)

:x(x1), y(y1)

{

}

int Point::getx()

{

    return x;

}

int Point::gety()

{

    return y;

}

float dist(Point& p1, Point& p2)

{

    int xt = p1.x - p2.x;

    int yt = p1.y - p2.y;

    return static\_cast<float>(sqrt(xt \* xt + yt \* yt));

}

int main()

{

    Point myp1(1, 1);

    Point myp2(4, 5);

    cout << "The distance is:";

    cout << dist(myp1, myp2) << endl;

    return 0;

}

**例5-7 常成员函数。**

#include <iostream>

using namespace std;

class R

{

private:

    int r1, r2;

public:

    R();

    R(int r11, int r21);

    void print();

    void print() const;

};

R::R()

{

}

R::R(int r11, int r21)

:r1(r11), r2(r21)

{

}

void R::print()

{

    cout << r1 << ":" << r2 << endl;

}

void R::print() const

{

    cout << r1 << ":" << r2 << endl;

}

int main()

{

    R a(5, 4);

    a.print();          //调用void print()

    const R b(20, 52);

    b.print();          //调用void print() const

    return 0;

}

**例5-8 常数据成员。**

#include <iostream>

using namespace std;

class A

{

public:

    A(int i);

    void print();

private:

    const int a;

    static const int b; //静态常数据成员

};

const int A::b = 10;    //静态常数据成员在类外说明和初始化

A::A(int i)

    :a(i)

{

}

void A::print()

{

    cout << a << ":" << b << endl;

}

int main()

{

    //建立对象a和b，并以100和0作为初值，分别调用构造函数

    //通过构造函数的初始化表列给对象的常数据成员赋初值

    A a1(100), a2(0);

    a1.print();

    a2.print();

    return 0;

}

**例5-9 常引用作形参**

#include <iostream>

#include <cmath>

using namespace std;

class Point

{

private:

    int x, y;

public:

    Point();

    Point(int x1, int y1);

    friend float dist(const Point& p1, const Point& p2);

};

Point::Point()

:x(0), y(0)

{

}

Point::Point(int x1, int y1)

:x(x1), y(y1)

{

}

float dist(const Point& p1, const Point& p2)

{

    int xp = p1.x - p2.x;

    int yp = p1.y - p2.y;

    return static\_cast<float>(sqrt(xp \* xp + yp \* yp));

}

int main()

{

    const Point myp1(1, 1), myp2(4, 5);

    cout << "The distance is:";

    cout << dist(myp1, myp2) << endl;

    return 0;

}

**例5-11 个人银行账户管理程序**

#include <cmath>

#include <iostream>

using namespace std;

class Account

{

private:

int id; // 账号

double balance; // 余额

double rate; // 存款的年利率

int lastDate; // 上次变更余额的日期

double accumulation;// 余额按日累加之和

static double total;// 所有账户的总金额

// 记录一笔账

void record(int date, double amount);

double accumulate(int date) const;

public:

Account(int date, int id1, double rate1);

int getId() const;

double getBalance() const;

double getRate() const;

static double getTotal();

// 存入现金

void deposit(int date, double amount);

// 去除现金

void withdraw(int date, double amount);

// 结算利息，每年1月1日调用一次该函数

void settle(int date);

// 显示账户信息

void show() const;

};

#include "Account.h"

double Account::total = 0;

Account::Account(int date, int id1, double rate1)

:id(id1), balance(0), rate(rate1), lastDate(date), accumulation(0)

{

cout << date << "\t#" << id << " is created" << endl;

}

int Account::getId() const

{

return id;

}

double Account::getBalance() const

{

return balance;

}

double Account::getRate() const

{

return rate;

}

double Account::getTotal()

{

return total;

}

void Account::deposit(int date, double amount)

{

record(date, amount);

}

void Account::withdraw(int date, double amount)

{

if (amount > getBalance())

cout << "Error: not enough money"<<endl;

else

record(date, -amount);

}

void Account::settle(int date)

{

double interest = accumulate(date) \* rate / 365; //计算年息

if (interest != 0)

record(date, interest);

accumulation = 0;

}

void Account::show() const

{

cout << "#" << id << "\tBalance:" << balance<<endl;

}

void Account::record(int date, double amount)

{

accumulation = accumulate(date);

lastDate = date;

amount = floor(amount \* 100 + 0.5) / 100;

balance += amount;

total += amount;

cout << date << "\t#" << id << "\t" << amount << "\t" << balance<<endl;

}

double Account::accumulate(int date) const

{

return accumulation + balance \* (date - lastDate);

}

#include "account.h"

int main(int argc, char\* argv[])

{

//建立几个账户

Account sa0(1, 21325302, 0.015);

Account sa1(1, 58320212, 0.015);

//几笔账目

sa0.deposit(5, 5000);

sa1.deposit(25, 10000);

sa0.deposit(45, 5500);;

sa1.withdraw(60, 4000);

//开户后第90天到了银行的计息日，结算所有账户的年息

sa0.settle(90);

sa1.settle(90);

//输出各个账户信息

sa0.show();

sa1.show();

cout << "Total:" << Account::getTotal()<<endl;

return 0;

}

## 第6章 数组指针与字符串

**例6-1 数组的声明与使用**

#include<iostream>

using namespace std;

int main()

{

const int N = 10;

int a[N], b[N];

int i;

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

{

a[i] = i \* 2 + 1;

b[N - i - 1] = a[i];

}

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

{

cout << "a[" << i << "]=" << a[i] << " " << "b[" << i << "]=" << b[i] << endl;

}

return 0;

}

**例6-2 使用数组名作为函数参数**

#include<iostream>

using namespace std;

int main()

{

void rowSum(int a[][4], int nRow);

int table[3][4] = { {1,2,3,4}, {2,3,4,5}, {3,4,5,6} };

int i, j;

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

{

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

cout << table[i][j] << " ";

cout << endl;

}

rowSum(table, 3);

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

cout << "Sum of row " << i << " is " << table[i][0] << endl;

return 0;

}

void rowSum(int a[][4], int nRow)

{

int i, j;

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

for (j = 1; j < 4; j++)

a[i][0] += a[i][j];

}

**例6-3 对象数组应用**

#include <iostream>

using namespace std;

class Point

{

private:

int x, y;

public:

Point();

Point(int x1, int y1);

~Point();

void move(int newx, int newy);

int getx() const;

int gety() const;

};

Point::Point()

:x(0), y(0)

{

cout << "Default Constructor called."<<endl;

}

Point::Point(int x1, int y1)

: x(x1), y(y1)

{

cout << "Constructor called." << endl;

}

Point::~Point()

{

cout << "Destructor called." << endl;

}

void Point::move(int newx, int newy)

{

cout << "Moving the point to (" << newx << ", " << newy << ")" << endl;

x = newx;

y = newy;

}

int Point::getx() const

{

return x;

}

int Point::gety() const

{

return y;

}

int main()

{

cout << "Entering main..."<<endl;

Point a[2];

int i;

for (i = 0; i < 2; i++)

a[i].move(i + 10, i + 20);

cout << "Exiting main..."<<endl;

return 0;

}

**例6-4 利用Point类进行点的线性拟合**

 求最小值









定义密切程度





#include <iostream>

using namespace std;

class Point

{

private:

float x, y;

public:

Point();

Point(float x1, float y1);

float getx() const;

float gety() const;

};

Point::Point()

:x(0), y(0)

{

}

Point::Point(float x1, float y1)

: x(x1), y(y1)

{

}

float Point::getx() const

{

return x;

}

float Point::gety() const

{

return y;

}

#include "Point.h"

#include <cmath>

int main()

{

float lineFit(const Point p[], int n);

Point pt[10] = {

Point(6,10),Point(14,20),Point(26,30),Point(33,40),Point(46,50),

Point(54,60),Point(67,70),Point(75,80),Point(84,90),Point(100,100)

};

float r = lineFit(pt, 10);

cout << "LIne coefficient r= " << r << endl;

return 0;

}

float lineFit(const Point p[], int n)

{

float avgx = 0;

float avgy = 0;

float lxx = 0;

float lyy = 0;

float lxy = 0;

int i;

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

avgx += p[i].getx() / n;

avgy += p[i].gety() / n;

}

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

lxx += (p[i].getx() - avgx) \* (p[i].getx() - avgx);

lyy += (p[i].gety() - avgy) \* (p[i].gety() - avgy);

lxy += (p[i].getx() - avgx) \* (p[i].gety() - avgy);

}

cout << "This line can be fitted by y=ax+b." << endl;

cout << "a= " << lxy / lxx << endl;

cout << "b= " << avgy - lxy \* avgx / lxx << endl;

return static\_cast<float>(lxy / sqrt(lxx \* lyy));

}

**例6-5 指针的定义、赋值与使用**

#include <iostream>

using namespace std;

int main()

{

int i;

int\* ptr = &i;

i = 10;

cout << "i=" << i << endl;

cout << "\*ptr=" << \*ptr << endl;

return 0;

}

**例6-6 void类型指针的使用**

#include <iostream>

using namespace std;

int main()

{

void\* pv;

int i = 5;

pv = &i;

int\* pint = static\_cast<int\*>(pv);

cout << "\*pint=" << \*pint << endl;

return 0;

}

**例6-7 设有一个int型数组，有10个元素，用3种方法输出各元素。**

#include <iostream>

using namespace std;

int main()

{

int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };

int i;

int\* p;

//使用数组名和下标

for (i = 0; i < 10; i++)

cout << a[i] << " ";

cout << endl;

//使用数组名和指针运算

for (i = 0; i < 10; i++)

cout << \*(a + i) << " ";

cout << endl;

//使用指针变量

for (p = a; p < (a + 10); p++)

cout << \*p << " ";

cout << endl;

return 0;

}

**例6-8 利用指针数组输出单位矩阵**

#include <iostream>

using namespace std;

int main()

{

int line1[] = { 1, 0, 0 };

int line2[] = { 0, 1, 0 };

int line3[] = { 0, 0, 1 };

//定义整型指针数组并初始化

int\* pLine[3] = { line1, line2, line3 };

cout << "Matrix test:" << endl;

int i, j;

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

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

cout << pLine[i][j] << " ";

cout << endl;

}

return 0;

}

**例6-9 二维数组举例**

#include <iostream>

using namespace std;

int main()

{

int array2[3][3] = {

{11, 12, 13},

{21, 22, 23},

{31, 32, 33}

};

int i, j;

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

{

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

cout << \*(\*(array2 + i) + j) << " ";

cout << endl;

}

return 0;

}

**例6-10 读入三个浮点数，将整数部分和小数部分分别输出**

#include <iostream>

using namespace std;

int main()

{

void splitfloat(float x, int\* intpart, float\* fracpart);

int i, n;

float x, f;

cout << "Enter three(3) floating point numbers" << endl;

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

{

cin >> x;

splitfloat(x, &n, &f); //变量地址做实参

cout << "Integer Part is " << n << " Fraction Part is " << f << endl;

}

return 0;

}

void splitfloat(float x, int\* intpart, float\* fracpart)

{

\*intpart = int(x); // 取x的整数部分

\*fracpart = x - \*intpart; //取x的小数部分

}

**例6-11 函数指针**

#include <iostream>

using namespace std;

int main()

{

void print\_stuff(float data\_to\_ignore);

void print\_message(float list\_this\_data);

void print\_float(float data\_to\_print);

void (\*function\_pointer)(float);

float pi = (float)3.14159;

float two\_pi = (float)2.0 \* pi;

print\_stuff(pi);

function\_pointer = print\_stuff;

function\_pointer(pi);

function\_pointer = print\_message;

function\_pointer(two\_pi);

function\_pointer(13.0);

function\_pointer = print\_float;

function\_pointer(pi);

print\_float(pi);

return 0;

}

void print\_stuff(float data\_to\_ignore)

{

cout << "This is the print stuff function." << endl;

}

void print\_message(float list\_this\_data)

{

cout << "The data to be listed is " << list\_this\_data << endl;

}

void print\_float(float data\_to\_print)

{

cout << "The data to be printed is " << data\_to\_print << endl;

}

**例6-12 使用指针访问类的成员**

#include <iostream>

using namespace std;

class Point

{

private:

int x, y;

public:

Point();

Point(int x1, int y1);

int getx() const;

int gety() const;

};

Point::Point()

{

}

Point::Point(int x1, int y1)

:x(x1), y(y1)

{

}

int Point::getx() const

{

return x;

}

int Point::gety() const

{

return y;

}

int main()

{

Point a(4, 5);

Point\* p1 = &a;

cout << p1->getx() << endl;

cout << a.getx() << endl;

return 0;

}

**例6-13 访问对象的公有成员函数的不同方式**

#include "point.h"

int main()

{

Point a(4, 5);

Point\* p1 = &a;

int (Point:: \* funcPtr)() const = &Point::getx;

cout << (a.\*funcPtr)() << endl;

cout << (p1->\*funcPtr)() << endl;

cout << a.getx() << endl;

cout << p1->getx() << endl;

return 0;

}

**例6-14 通过指针访问类的静态数据成员**

#include <iostream>

using namespace std;

class Point

{

private:

int x, y;

public:

Point();

Point(int x1, int y1);

~Point();

int getx() const;

int gety() const;

static int count;

};

Point::Point()

:x(0), y(0)

{

count++;

}

Point::Point(int x1, int y1)

: x(x1), y(y1)

{

count++;

}

Point::~Point()

{

count--;

}

int Point::getx() const

{

return x;

}

int Point::gety() const

{

return y;

}

int Point::count = 0;

int main()

{

int\* ptr = &Point::count;

Point a(4, 5);

cout << "Point A:" << a.getx() << "," << a.gety();

cout << " Objec count=" << \*ptr << endl;

Point b(a);

cout << "Point B:" << b.getx() << "," << b.gety();

cout << " Object count=" << \*ptr << endl;

return 0;

}

**例6-15 通过指针访问类的静态函数成员**

#include <iostream>

using namespace std;

class Point

{

private:

int x, y;

static int count;

public:

Point();

Point(int x1, int y1);

~Point();

int getx() const;

int gety() const;

static void showCount();

};

Point::Point()

:x(0), y(0)

{

count++;

}

Point::Point(int x1, int y1)

: x(x1), y(y1)

{

count++;

}

Point::~Point()

{

count--;

}

int Point::getx() const

{

return x;

}

int Point::gety() const

{

return y;

}

void Point::showCount()

{

cout << " Object count=" << count << endl;

}

int Point::count = 0;

int main()

{

void (\*funcPtr)() = Point::showCount;

Point a(4, 5);

cout << "Point A:" << a.getx() << "," << a.gety();

funcPtr();

Point b(a);

cout << "Point B:" << b.getx() << "," << b.gety();

funcPtr();

return 0;

}

**例6-16 动态创建对象**

#include <iostream>

using namespace std;

class Point

{

private:

int x, y;

public:

Point();

Point(int x1, int y1);

~Point();

int getx() const;

int gety() const;

void move(int x1, int y1);

};

Point::Point()

:x(0), y(0)

{

cout << "Default Constructor Called." << endl;

}

Point::Point(int x1, int y1)

: x(x1), y(y1)

{

cout << "Constructor Called." << endl;

}

Point::~Point()

{

cout << "Destructor Called." << endl;

}

int Point::getx() const

{

return x;

}

int Point::gety() const

{

return y;

}

void Point::move(int x1, int y1)

{

x = x1;

y = y1;

}

int main()

{

cout << "Step one:" <<endl;

Point\* ptr1 = new Point;

delete ptr1;

cout << "Step two:" <<endl;

ptr1 = new Point(1, 2);

delete ptr1;

return 0;

}

**例6-17 动态创建对象数组**

#include "point.h"

int main()

{

Point\* ptr = new Point[2];

ptr[0].move(5, 10);

ptr[1].move(15, 20);

cout << "Deleting..." <<endl;

delete[]ptr;

return 0;

}

**例6-18 动态数组类**

#include <iostream>

#include <cassert>

using namespace std;

//点类的定义

class Point

{

    int x, y;

public:

    Point();

    Point(int x1, int y1);

    ~Point();

    int getx() const;

    int gety() const;

    void move(int x1, int y1);

};

Point::Point()

    :x(0), y(0)

{

    cout << "Default Constructor Called." <<endl;

}

Point::Point(int x1, int y1)

    : x(x1), y(y1)

{

    cout << "Constructor Called." << endl;

}

Point::~Point()

{

    cout << "Destructor Called." << endl;

}

int Point::getx() const

{

    return x;

}

int Point::gety() const

{

    return y;

}

void Point::move(int x1, int y1)

{

    x = x1;

    y = y1;

}

//动态数组类定义

class ArrayPoints

{

private:

    int size;//数组当前长度

    Point\* ptr;//数组首地址

public:

    ArrayPoints(int s);

    ~ArrayPoints();

    Point& element(int index);

};

ArrayPoints::ArrayPoints(int s)

:size(s)

{

    ptr = new Point[size];

}

ArrayPoints::~ArrayPoints()

{

    delete[] ptr;

}

Point& ArrayPoints::element(int index)

{

    assert(index >= 0 && index < size);

    return ptr[index];

}

int main()

{

    int count;

    cout << "Please enter the count of points:" <<endl;

    cin >> count;

    ArrayPoints points(count);

    points.element(0).move(5, 0);

    points.element(1).move(15, 20);

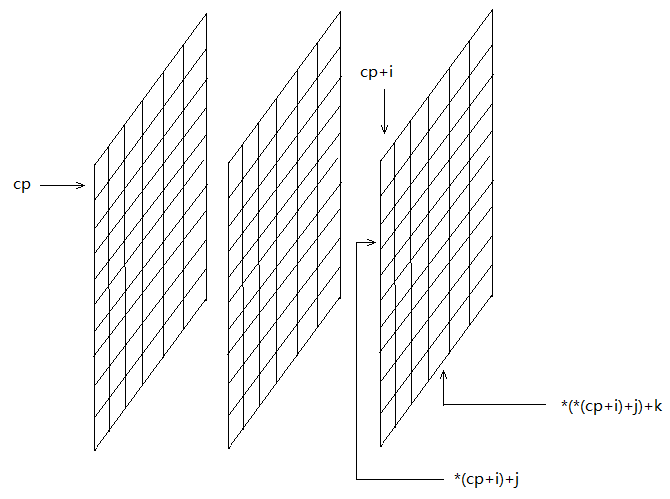
    cout<<"The first point is ("<<points.element(0).getx()<<", "<<points.element(0).gety()<<")"<<endl;

    cout<<"The second point is ("<<points.element(1).getx()<<", "<<points.element(1).gety()<<")"<<endl;

    return 0;

}

**例6-19 动态创建多维数组**



#include <iostream>

using namespace std;

int main()

{

float (\*cp)[9][8] = new float[8][9][8];

int i, j, k;

for (i = 0; i < 8; i++)

for (j = 0; j < 9; j++)

for (k = 0; k < 8; k++)

\*(\*(\*(cp + i) + j) + k) = static\_cast<float>(i \* 100 + j \* 10 + k);

for (i = 0; i < 8; i++)

{

for (j = 0; j < 9; j++)

{

for (k = 0; k < 8; k++)

cout << cp[i][j][k] << " ";

cout << endl;

}

cout << endl;

}

delete[]cp;

return 0;

}

**例6-20 vector应用**

#include <vector>

#include <iostream>

using namespace std;

double average(const vector<double>& arr);

int main()

{

int n;

cout << "n=";

cin >> n;

vector<double>arr(n);

cout << "Please input " << n << " real numbers:" << endl;

int i;

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

cin >> arr[i];

cout << "Average=" << average(arr) << endl;

return 0;

}

double average(const vector<double>& arr)

{

double sum = 0;

int i;

for (i = 0; i < arr.size(); i++)

sum += arr[i];

return sum / arr.size();

}

**例6-21 对象的浅复制**

#include "arraypoints.h"

int main()

{

int count;

cout << "Please enter the count of points:";

cin >> count;

ArrayPoints pointsArray1(count);

pointsArray1.element(0).move(5, 10);

pointsArray1.element(1).move(15, 20);

ArrayPoints pointsArray2 = pointsArray1;

cout << "Copy of pointsArray1:" << endl;

cout << "Point\_0 of array2:" << pointsArray2.element(0).getx() << "," << pointsArray2.element(0).gety() << endl;

cout << "Point\_1 of array2:" << pointsArray2.element(1).getx() << "," << pointsArray2.element(1).gety() << endl;

pointsArray1.element(0).move(25, 30);

pointsArray1.element(1).move(35, 40);

cout << "After the moving of pointsArray1:" << endl;

cout << "Point\_0 of array2:" << pointsArray2.element(0).getx() << "," << pointsArray2.element(0).gety() << endl;

cout << "Point\_1 of array2:" << pointsArray2.element(1).getx() << "," << pointsArray2.element(1).gety() << endl;

return 0;

}

**例6-22 对象的深复制**

#include "point.h"

#include <cassert>

class ArrayPoints

{

private:

int size;

Point\* ptr;

public:

ArrayPoints(int s);

ArrayPoints(const ArrayPoints& obj);

~ArrayPoints();

Point& element(int index);

};

#include "ArrayPoints.h"

ArrayPoints::ArrayPoints(int s)

:size(s)

{

ptr = new Point[size];

}

ArrayPoints::ArrayPoints(const ArrayPoints& obj)

{

size = obj.size;

ptr = new Point[size];

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

ptr[i] = obj.ptr[i];

}

ArrayPoints::~ArrayPoints()

{

cout << "Deleting..." <<endl;

delete[]ptr;

}

Point& ArrayPoints::element(int index)

{

assert(index >= 0 && index < size);

return ptr[index];

}

#include "arraypoints.h"

int main()

{

int count;

cout << "Please enter the count of points:";

cin >> count;

ArrayPoints pointsArray1(count);

pointsArray1.element(0).move(5, 10);

pointsArray1.element(1).move(15, 20);

ArrayPoints pointsArray2 = pointsArray1;

cout << "Copy of pointsArray1:" << endl;

cout << "Point\_0 of array2:" << pointsArray2.element(0).getx() << "," << pointsArray2.element(0).gety() << endl;

cout << "Point\_1 of array2:" << pointsArray2.element(1).getx() << "," << pointsArray2.element(1).gety() << endl;

pointsArray1.element(0).move(25, 30);

pointsArray1.element(1).move(35, 40);

cout << "After the moving of pointsArray1:" << endl;

cout << "Point\_0 of array2:" << pointsArray2.element(0).getx() << "," << pointsArray2.element(0).gety() << endl;

cout << "Point\_1 of array2:" << pointsArray2.element(1).getx() << "," << pointsArray2.element(1).gety() << endl;

return 0;

}

**例6-23 string类应用举例**

#include <string>

#include <iostream>

using namespace std;

int main()

{

void test(const char\* title, bool value);

string s1 = "EDF";

cout << "s1 is " << s1 << endl;

string s2;

cout << "Please enter s2:";

cin >> s2;

cout << "length of s2:" << s2.length() << endl;

test("s1<=\"ABC\"", s1 <= "ABC");

test("\"def\"<=s1", "DEF" <= s1);

s2 += s1;

cout << "s2=s2+s1:" << s2 << endl;

cout << "length of s2:" << s2.length() << endl;

return 0;

}

void test(const char\* title, bool value)

{

cout << title << " returns " << (value ? "true" : "false") << endl;

}

**例6-24 用getline输入字符串**

#include <string>

#include <iostream>

using namespace std;

int main()

{

int i;

for (i = 0; i < 2; i++)

{

string city, state;

getline(cin, city, ',');

getline(cin, state);

cout << "City:" << city << " State:" << state << endl;

}

return 0;

}

**例6-25 个人银行账户管理程序改进**

改进1：将账户改用字符串表示，这样可以表达诸如002378166374这样开始编号为0的账户。

改进2：将日期用类表示，其属性为（y, m, d）。这样表示直观，方便，但计算两个日期之间的天数复杂一些。可以通过统一基准日期的办法来计算。

* 假设某天对公元1年1月1日的天数为f(y/m/d, 1/1/1)，则其可以分开来计算
* f(y/m/d, 1/1/1) = f(y/1/1, 1/1/1) + f(y/m/1, y/1/1) + f(y/m/d, y/d/1)
* f(y/1/1, 1/1/1)为y年的1月1号和公元1年1月1号之间的总天数。每年365天，闰年多一天，所以总天数为(y-1)\*365 + 闰年数。由于4年一闰，100的倍数免闰，400的倍数再闰，所以
* f(y/1/1, 1/1/1) = (y-1)\*365 + (y-1)/4 – (y-1)/100 + (y-1)/400
* 注意这里的/为整除
* f(y/m/1, y/1/1)为当年的某月1号与当年的1月1号的天数，这个有两种情况，一种是平年，一种是闰年，需要分别计算。比较简单的办法是查表。
* f(y/m/d, y/d/1) = d - 1

#include <iostream>

#include <cstdlib>

using namespace std;

class Date

{

private:

int year;

int month;

int day;

int totalDays;//从公元1/1/1开始的总天数

public:

Date(int y, int m, int d);

int getYear() const;

int getMonth() const;

int getDay() const;

int getTheMonthDays() const;// 获得当月的天数

bool isLeapYear() const;// 判断当年是否闰年

void show() const;// 输出当前日期

// 计算两个日期之间的天数

int distances(const Date& date) const;

};

#include "Date.h"

namespace { // namespace使下面的定义只在当前文件中有效

// 存储平年中的某个月1日之前有多少天，

// 为便于getTheMonthDays函数的实现，该数组多出一项

const int DAYS\_BEFORE\_MONTH[] = {

0,31,59,90,120,151,181,212,243,273,304,334,365

};

}

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

:year(year), month(month), day(day)

{

if (day <= 0 || day > getTheMonthDays())

{

cout<<"Invalid date: ";

show();

cout << endl;

exit(1);

}

int years = year - 1;

totalDays = years \* 365 + years / 100 + years / 400

+ DAYS\_BEFORE\_MONTH[month - 1] + day;

if (isLeapYear() && month > 2)

totalDays++;

}

int Date::getYear() const

{

return year;

}

int Date::getMonth() const

{

return month;

}

int Date::getDay() const

{

return day;

}

// 获得当月的天数

int Date::getTheMonthDays() const

{

if (isLeapYear() && month == 2)

return 29;

else

return DAYS\_BEFORE\_MONTH[month] - DAYS\_BEFORE\_MONTH[month - 1];

}

// 判断当年是否闰年

bool Date::isLeapYear() const

{

return year % 4 == 0 && year % 100 != 0 || year % 400 == 0;

}

// 输出当前日期

void Date::show() const

{

cout << getYear() << "-" << getMonth() << "-" << getDay();

}

// 计算两个日期之间的天数

int Date::distances(const Date& date) const

{

return totalDays - date.totalDays;

}

#include "Date.h"

#include <string>

#include <cmath>

using namespace std;

class Account

{

private:

string id; // 账号

double balance; // 余额

double rate; // 存款的年利率

Date lastDate; // 上次变更余额的日期

double accumulation; // 余额按日累加之和

static double total; // 所有账户的总金额

// 记录一笔账，date为日期，amount为金额，desc为说明

void record(const Date& date, double amount, const string& desc);

// 报告错误信息

void error(const string& msg) const;

// 获得到指定日期为止的存款金额按日累积值

double accumulate(const Date& date) const;

public:

Account(const Date& date, const string& id, double rate);

const string& getId() const;

double getBalance() const;

double getRate() const;

static double getTotal();

// 存入现金

void deposit(const Date& date, double amount, const string& desc);

// 取出现金

void withdraw(const Date& date, double amount, const string& desc);

// 结算利息，每年1月1日调用一次该函数

void settle(const Date& date);

// 显示账户信息

void show() const;

};

#include "Account.h"

double Account::total = 0;

Account::Account(const Date& date, const string& id, double rate)

:id(id), balance(0), rate(rate), lastDate(date), accumulation(0)

{

date.show();

cout << "\t#" << id << " created" << endl;

}

// 记录一笔账，date为日期，amount为金额，desc为说明

void Account::record(const Date& date, double amount, const string& desc)

{

accumulation = accumulate(date);

lastDate = date;

amount = floor(amount \* 100 + 0.5) / 100;

balance += amount;

total += amount;

date.show();

cout << "\t#" << id << "\t" << amount << "\t" << balance << "\t" << desc << endl;

}

// 报告错误信息

void Account::error(const string& msg) const

{

cout << "Error(#" << id << "):" << msg << endl;

}

// 获得到指定日期为止的存款金额按日累积值

double Account::accumulate(const Date& date) const

{

return accumulation + balance \* (date.distances(lastDate));

}

const string& Account::getId() const

{

return id;

}

double Account::getBalance() const

{

return balance;

}

double Account::getRate() const

{

return rate;

}

double Account::getTotal()

{

return total;

}

// 存入现金

void Account::deposit(const Date& date, double amount, const string& desc)

{

record(date, amount, desc);

}

// 取出现金

void Account::withdraw(const Date& date, double amount, const string& desc)

{

if (amount > getBalance())

error("not enouth money");

else

record(date, -amount, desc);

}

// 结算利息，每年1月1日调用一次该函数

void Account::settle(const Date& date)

{

double interest = accumulate(date) \* rate //计算年息

/ date.distances(Date(date.getYear() - 1, 1, 1));

if (interest != 0)

record(date, interest, "interest");

accumulation = 0;

}

// 显示账户信息

void Account::show() const

{

cout << id << "\tBalance:" << balance;

}

#include "Account.h"

int main(int argc, char\* argv[])

{

Date date(2008, 11, 1);

//建立几个账户

Account accounts[] = {

Account(date, "S3755217", 0.015),

Account(date, "02342342", 0.015)

};

const int n = sizeof(accounts) / sizeof(Account); // 账户总数

// 11月份的几笔账目

accounts[0].deposit(Date(2008, 11, 5), 5000, "salary");

accounts[1].deposit(Date(2008, 11, 25), 10000, "sell stock 0323");

// 12月份的几笔账目

accounts[0].deposit(Date(2008, 12, 5), 5500, "salary");

accounts[1].withdraw(Date(2008, 12, 20), 4000, "buy a laptop");

// 结算所有账户并输出各个账户信息

cout << endl;

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

{

accounts[i].settle(Date(2009, 1, 1));

accounts[i].show();

cout << endl;

}

cout << "Total: " << Account::getTotal() << endl;

return 0;

}

## 第7章 继承与派生

**例7-1 Point类公有继承举例**

#include <iostream>

using namespace std;

class Point

{

private:

    float x;

    float y;

public:

    Point();

    Point(float x1, float y1);

    Point(const Point& obj);

    ~Point();

    void move(float offX, float offY);

    float getX() const;

    float getY() const;

};

Point::Point()

:x(0), y(0)

{

}

Point::Point(float x1, float y1)

:x(x1), y(y1)

{

}

Point::Point(const Point& obj)

:x(obj.x), y(obj.y)

{

}

Point::~Point()

{

}

void Point::move(float offX, float offY)

{

    x += offX;

    y += offY;

}

float Point::getX() const

{

    return x;

}

float Point::getY() const

{

    return y;

}

class Rectangle : public Point

{

private:

    float w;

    float h;

public:

    Rectangle();

    Rectangle(float x1, float y1, float w1, float h1);

    Rectangle(const Rectangle& obj);

    ~Rectangle();

    float getH() const;

    float getW() const;

};

Rectangle::Rectangle()

:w(0), h(0)

{

}

Rectangle::Rectangle(float x1, float y1, float w1, float h1)

: Point(x1, y1), w(w1), h(h1)

{

}

Rectangle::Rectangle(const Rectangle& obj)

:Point(obj.getX(), obj.getY()),

w(obj.w), h(obj.h)

{

}

Rectangle::~Rectangle()

{

}

float Rectangle::getH() const

{

    return h;

}

float Rectangle::getW() const

{

    return w;

}

int main(int argc, char\* argv[])

{

    Rectangle rect(2, 3, 20, 10);

    cout << "The data of rect(x,y,w,h):" <<endl;

    cout << rect.getX() << ", "

        << rect.getY() << ", "

        << rect.getW() << ", "

        << rect.getH() << endl;

    cout<<endl;

    rect.move(3, 2);

    Rectangle rect1(rect);

    cout << "The data of rect1(x,y,w,h):" <<endl;

    cout << rect1.getX() << ", "

        << rect1.getY() << ", "

        << rect1.getW() << ", "

        << rect1.getH() << endl;

    return 0;

}

**例7-2 Point私有继承举例**

#include <iostream>

using namespace std;

class Point

{

private:

    float x;

    float y;

public:

    Point();

    Point(float x1, float y1);

    Point(const Point& obj);

    ~Point();

    void move(float offX, float offY);

    float getX() const;

    float getY() const;

};

Point::Point()

:x(0), y(0)

{

}

Point::Point(float x1, float y1)

:x(x1), y(y1)

{

}

Point::Point(const Point& obj)

:x(obj.x), y(obj.y)

{

}

Point::~Point()

{

}

void Point::move(float offX, float offY)

{

    x += offX;

    y += offY;

}

float Point::getX() const

{

    return x;

}

float Point::getY() const

{

    return y;

}

class Rectangle : private Point

{

private:

    float w;

    float h;

public:

    Rectangle();

    Rectangle(float x1, float y1, float w1, float h1);

    Rectangle(const Rectangle& obj);

    ~Rectangle();

    void move(float offX, float offY);

    float getX() const;

    float getY() const;

    float getH() const;

    float getW() const;

};

Rectangle::Rectangle()

:w(0), h(0)

{

}

Rectangle::Rectangle(float x1, float y1, float w1, float h1)

:Point(x1, y1), w(w1), h(h1)

{

}

Rectangle::Rectangle(const Rectangle& obj)

:Point(obj.getX(),obj.getY()),

w(obj.w), h(obj.h)

{

}

Rectangle::~Rectangle()

{

}

void Rectangle::move(float offX, float offY)

{

    Point::move(offX, offY);

}

float Rectangle::getX() const

{

    return Point::getX();

}

float Rectangle::getY() const

{

    return Point::getY();

}

float Rectangle::getH() const

{

    return h;

}

float Rectangle::getW() const

{

    return w;

}

int main(int argc, char\* argv[])

{

    Rectangle rect(2, 3, 20, 10);

    cout << "The data of rect(x,y,w,h):" <<endl;

    cout << rect.getX() << ", "

        << rect.getY() << ", "

        << rect.getW() << ", "

        << rect.getH() << endl;

    cout<<endl;

    rect.move(3, 2);

    Rectangle rect1(rect);

    cout << "The data of rect1(x,y,w,h):" <<endl;

    cout << rect1.getX() << ", "

        << rect1.getY() << ", "

        << rect1.getW() << ", "

        << rect1.getH() << endl;

    return 0;

}

**例7-3 类型兼容规则实例**

#include <iostream>

using namespace std;

class Base1

{

public:

Base1();

void display() const;

};

Base1::Base1()

{

}

void Base1::display() const

{

cout << "Base1::display()" << endl;

}

class Base2 : public Base1

{

public:

Base2();

void display() const;

};

Base2::Base2()

{

}

void Base2::display() const

{

cout << "Base2::display()" << endl;

}

class Derived : public Base2

{

public:

Derived();

void display() const;

};

Derived::Derived()

{

}

void Derived::display() const

{

cout << "Derived::display()" << endl;

}

int main(int argc, char\* argv[])

{

void fun(Base1 \* ptr);//参数为指向基类对象的指针

Base1 base1;//声明Base1类对象

Base2 base2;//声明Base2类对象

Derived derived;// 声明Derived类对象

fun(&base1);//用Base1对象的指针调用fun函数

fun(&base2);//用Base2对象的指针调用fun函数

fun(&derived);//用Derived对象的指针调用fun函数

return 0;

}

void fun(Base1\* ptr)

{

ptr->display();//对象指针->成员名

}

**例7-4 派生类构造函数**

#include <iostream>

using namespace std;

class Base1

{

public:

    Base1();

    Base1(int i);

};

Base1::Base1()

{

}

Base1::Base1(int i)

{

    cout << "Constructing Base1 " << i << endl;

}

class Base2

{

public:

    Base2();

    Base2(int j);

};

Base2::Base2()

{

}

Base2::Base2(int j)

{

    cout << "Constructing Base2 " << j << endl;

}

class Base3

{

public:

    Base3();

};

Base3::Base3()

{

    cout << "Constructing Base3 \*" << endl;

}

class Derived : public Base2, public Base1, public Base3

{

private:

    Base1 member1;

    Base2 member2;

    Base3 member3;

public:

    Derived();

    Derived(int a, int b, int c, int d);

};

Derived::Derived()

{

}

Derived::Derived(int a, int b, int c, int d)

:Base1(a), member2(d), member1(c), Base2(b)

{

}

int main()

{

    Derived obj(1, 2, 3, 4);

    return 0;

}

**例7-5 派生类析构函数举例（多继承、含有嵌入对象）。**

#include <iostream>

using namespace std;

class Base1

{

public:

    Base1(int i);

    ~Base1();

};

Base1::Base1(int i)

{

    cout << "Constructing Base1 " << i << endl;

}

Base1::~Base1()

{

    cout << "Destructing Base1" << endl;

}

class Base2

{

public:

    Base2(int j);

    ~Base2();

};

Base2::Base2(int j)

{

    cout << "Constructing Base2 " << j << endl;

}

Base2::~Base2()

{

    cout << "Destructing Base2" << endl;

}

class Base3

{

public:

    Base3();

    ~Base3();

};

Base3::Base3()

{

    cout << "Constructing Base3 \*" << endl;

}

Base3::~Base3()

{

    cout << "Destructing Base3" << endl;

}

class Derived : public Base2, public Base1, public Base3

{

private:

    Base1 member1;

    Base2 member2;

    Base3 member3;

public:

    Derived(int a, int b, int c, int d);

};

Derived::Derived(int a, int b, int c, int d)

:Base1(a), member2(d), member1(c), Base2(b)

{

}

int main()

{

    Derived obj(1, 2, 3, 4);

    return 0;

}

**例7-6 多继承同名隐藏举例1**

#include <iostream>

using namespace std;

class Base1

{

public:

int var;

void fun();

};

void Base1::fun()

{

cout << "Member of Base1" << endl;

}

class Base2

{

public:

int var;

void fun();

};

void Base2::fun()

{

cout << "Member of Base2" << endl;

}

class Derived : public Base1, public Base2

{

public:

int var;

void fun();

};

void Derived::fun()

{

cout << "Member of Derived" << endl;

}

int main(int argc, char\* argv[])

{

Derived d;

Derived\* p = &d;

d.var = 1;

d.fun();

d.Base1::var = 2;

d.Base1::fun();

p->Base2::var = 3;

p->Base2::fun();

return 0;

}

**例7-7 多继承同名隐藏举例2**

#include <iostream>

using namespace std;

class Base0

{

public:

int var0;

void fun0();

};

void Base0::fun0()

{

cout << "Member of Base0" << endl;

}

class Base1 : public Base0

{

public:

int var1;

};

class Base2 : public Base0

{

public:

int var2;

};

class Derived : public Base1, public Base2

{

public:

int var;

void fun();

};

void Derived::fun()

{

cout << "Member of Derived" << endl;

}

int main(int argc, char\* argv[])

{

Derived d;

d.Base1::var0 = 2;

d.Base1::fun0();

d.Base2::var0 = 3;

d.Base2::fun0();

return 0;

}

**例7-8 虚基类**

#include <iostream>

using namespace std;

class Base0

{

public:

int var0;

void fun0();

};

void Base0::fun0()

{

cout << "Member of Base0" << endl;

}

class Base1 : virtual public Base0

{

public:

int var1;

};

class Base2 : virtual public Base0

{

public:

int var2;

};

class Derived : public Base1, public Base2

{

public:

int var;

void fun();

};

void Derived::fun()

{

cout << "Member of Derived" << endl;

}

int main(int argc, char\* argv[])

{

Derived d;

d.var0 = 2;

d.fun0();

return 0;

}

## 第8章 多态性

**例8-1 复数类运算符重载**

#include <iostream>

using namespace std;

class Complex

{

private:

double real;

double image;

public:

Complex();

Complex(double r, double i);

Complex(const Complex& obj);

Complex operator +(const Complex& c2) const;

Complex operator -(const Complex& c2) const;

void display() const;

};

#include "Complex.h"

Complex::Complex()

:real(0), image(0)

{

}

Complex::Complex(double r, double i)

: real(r), image(i)

{

}

Complex::Complex(const Complex& obj)

: real(obj.real), image(obj.image)

{

}

Complex Complex::operator +(const Complex& c2) const

{

return Complex(real + c2.real, image + c2.real);

}

Complex Complex::operator -(const Complex& c2) const

{

return Complex(real - c2.real, image - c2.real);

}

void Complex::display() const

{

cout << "(" << real << "," << image << ")" << endl;

}

#include "Complex.h"

int main()

{

Complex c1(5, 4), c2(2, 10), c3;

cout << "c1=";

c1.display();

cout << "c2=";

c2.display();

c3 = c1 - c2;

cout << "c3=c1-c2=";

c3.display();

c3 = c1 + c2;

cout << "c3=c1+c2=";

c3.display();

return 0;

}

**例8-2 单目运算符重载**

#include <iostream>

using namespace std;

class Clock

{

private:

int hour;

int minute;

int second;

public:

Clock();

Clock(int h, int m, int s);

Clock& operator ++();//前置增量运算

Clock operator ++(int);//后置增量运算

void show() const;

};

#include "Clock.h"

Clock::Clock()

:hour(0), minute(0), second(0)

{

}

Clock::Clock(int h, int m, int s)

: hour(h), minute(m), second(s)

{

}

Clock& Clock::operator ++()//前置增量运算

{

second++;

if (second >= 60)

{

second -= 60;

minute++;

if (minute >= 60)

{

minute -= 60;

hour = (hour + 1) % 24;

}

}

return \*this;

}

Clock Clock::operator ++(int)//后置增量运算

{

Clock old = \*this;

++(\*this);

return old;

}

void Clock::show() const

{

cout << hour << ":" << minute << ":" << second << endl;

}

#include "Clock.h"

int main()

{

Clock myClock(23, 59, 59);

cout << "First time output:";

myClock.show();

cout << "Show myClock++: ";

(myClock++).show();

cout << "Show ++myClock: ";

(++myClock).show();

return 0;

}

**例8-3 非成员函数重载**

#include <iostream>

using namespace std;

class Complex

{

private:

double real;

double image;

public:

Complex();

Complex(double r, double i);

friend Complex operator+(const Complex& c1, const Complex& c2);

friend Complex operator-(const Complex& c1, const Complex& c2);

friend ostream& operator<<(ostream& out, const Complex& c);

};

#include "Complex.h"

Complex::Complex()

{

}

Complex::Complex(double r, double i)

:real(r), image(i)

{

}

Complex operator+(const Complex& c1, const Complex& c2)

{

return Complex(c1.real + c2.real, c1.image + c2.image);

}

Complex operator-(const Complex& c1, const Complex& c2)

{

return Complex(c1.real - c2.real, c1.image - c2.image);

}

ostream& operator<<(ostream& out, const Complex& c)

{

out << "(" << c.real << "," << c.image << ")";

return out;

}

#include "Complex.h"

int main(int argc, char\* argv[])

{

Complex c1(5, 4), c2(2, 10), c3;

cout << "c1=" << c1 << endl;

cout << "c2=" << c2 << endl;

c3 = c1 - c2;

cout << "c3=c1-c2=" << c3 << endl;

c3 = c1 + c2;

cout << "c3=c1+c2=" << c3 << endl;

return 0;

}

**例8-4 虚函数成员**

#include <iostream>

using namespace std;

class Base1

{

public:

virtual void display() const;

};

void Base1::display() const

{

cout << "Base1: display()" << endl;

}

class Base2 : public Base1

{

public:

void display() const;

};

void Base2::display() const

{

cout << "Base2: display()" << endl;

}

class Derived : public Base2

{

public:

void display() const;

};

void Derived::display() const

{

cout << "Derived: display()" << endl;

}

int main()

{

void fun(Base1 \* ptr);

Base1 base1;

Base2 base2;

Derived derived;

fun(&base1);

fun(&base2);

fun(&derived);

return 0;

}

void fun(Base1\* ptr)

{

ptr->display();

}

**例8-5 虚析构函数**

#include <iostream>

using namespace std;

class Base

{

public:

virtual ~Base();

};

Base::~Base()

{

cout << "Base destructor" << endl;

}

class Derived : public Base

{

private:

int\* p;

public:

Derived();

~Derived();

};

Derived::Derived()

{

p = new int(0);

}

Derived::~Derived()

{

cout << "Derived destructor" << endl;

delete p;

}

int main(int argc, char\* argv[])

{

void fun(Base \* b);

Base\* b = new Derived();

fun(b);

return 0;

}

void fun(Base\* b)

{

delete b;

}

**例8-6 抽象类**

#include <iostream>

using namespace std;

class Base1

{

public:

virtual void display() const = 0;

};

class Base2 : public Base1

{

public:

void display() const;

};

void Base2::display() const

{

cout << "Base2: display()" << endl;

}

class Derived : public Base2

{

public:

void display() const;

};

void Derived::display() const

{

cout << "Derived: display()" << endl;

}

int main()

{

void fun(Base1 \* ptr);

Base2 base2;

Derived derived;

fun(&base2);

fun(&derived);

return 0;

}

void fun(Base1\* ptr)

{

ptr->display();

}

## 第9章 群体类和群体数据的组织

**例9-1 函数模板**

#include <iostream>

using namespace std;

template<typename T>

void outputArray(const T\* array, int count);

int main(int argc, char\* argv[])

{

const int A\_COUNT = 8;

const int B\_COUNT = 8;

const int C\_COUNT = 20;

int a[A\_COUNT] = { 1,2,3,4,5,6,7,8 };

double b[B\_COUNT] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8 };

char c[C\_COUNT] = "Welcome to see you!";

cout << "a array contains:" << endl;

outputArray(a, A\_COUNT);

cout << "b array contains:" << endl;

outputArray(b, B\_COUNT);

cout << "c array contains:" << endl;

outputArray(c, C\_COUNT);

return 0;

}

template<typename T>

void outputArray(const T\* array, int count)

{

int i;

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

cout << array[i] << " ";

cout << endl;

}

**例9-2 类模板应用（类模板要求声明和实现文件放在一起）**

// student.h

class Student

{

private:

    int id;

    float gpa;//平均分

public:

    Student();

    Student(int id, float gpa);

    int getId();

    float getGpa();

};

Student::Student()

{

}

Student::Student(int id, float gpa)

    :id(id), gpa(gpa)

{

}

int Student::getId()

{

    return id;

}

float Student::getGpa()

{

    return gpa;

}

//store.h

#include "student.h"

template <class T>

class Store

{

private:

    T item;//存放任意类型数据

    bool haveValue;//数据存放标志

public:

    Store();

    Store(T item);

    T& getElem();

};

template<class T>

Store<T>::Store()

    :haveValue(false)

{

}

template<class T>

Store<T>::Store(T item)

    :item(item), haveValue(true)

{

}

template<class T>

T& Store<T>::getElem()

{

    return item;

}

//9-3.cpp

#include <iostream>

using namespace std;

#include "Store.h"

int main(int argc, char\* argv[])

{

    Store<int> s1(3);

    Store<int> s2(-7);

    cout << s1.getElem() << " " << s2.getElem() << endl;

    Student g(1000, 23);

    Store<Student> s3(g);

    cout << "The student id is " << s3.getElem().getId() << endl;

    return 0;

}

**例9-3 Array类**

// array.h

#include <cassert>

template<class T>

class Array

{

private:

    T\* list;

    int size;

public:

    Array();

    Array(int sz);

    Array(const Array<T>& a);

    ~Array();

    Array<T>& operator=(const Array<T>& rhs);

    T& operator[](int i);

    const T& operator[](int i) const;

    operator T\* ();

    operator const T\* () const;

    int getSize() const;

    void resize(int sz);

};

template<class T>

Array<T>::Array()

{

    list = NULL;

    size = 0;

}

template<class T>

Array<T>::Array(int sz)

{

    assert(sz >= 0);

    size = sz;

    list = new T[size];

}

template<class T>

Array<T>::~Array()

{

    if (list)

        delete[]list;

}

template<class T>

Array<T>::Array(const Array<T>& a)

{

    size = a.size;

    list = new T[size];

    int i;

    for (i = 0; i < size; i++)

        list[i] = a.list[i];

}

template<class T>

Array<T>& Array<T>::operator=(const Array<T>& rhs)

{

    int i;

    if (&rhs != this)

    {

        if (size != rhs.size)

        {

            delete[]list;

            size = rhs.size;

            list = new T[size];

        }

        for (i = 0; i < size; i++)

            list[i] = rhs.list[i];

    }

    return \*this;

}

template<class T>

T& Array<T>::operator[](int n)

{

    assert(n >= 0 && n < size);

    return list[n];

}

template<class T>

const T& Array<T>::operator[](int n) const

{

    assert(n >= 0 && n < size);

    return list[n];

}

template<class T>

Array<T>::operator T\* ()

{

    return list;

}

template<class T>

Array<T>::operator const T\* () const

{

    return list;

}

template<class T>

int Array<T>::getSize() const

{

    return size;

}

template<class T>

void Array<T>::resize(int sz)

{

    assert(sz >= 0);

    if (sz == size)

        return;

    T\* newList = new T[sz];

    int n = (sz < size) ? sz : size;

    int i;

    for (i = 0; i < n; i++)

        newList[i] = list[i];

    delete[]list;

    list = newList;

    size = sz;

}

**例9-4 Array类的应用**

#include <iostream>

#include <iomanip>

using namespace std;

#include "array.h"

int main(int argc, char\* argv[])

{

    Array<int> b(10);

    int count = 0;

    int n;

    cout << "Enter a value>=2 as upper limit for prime numbers:";

    cin >> n;

    int i, j;

    bool isPrime;

    for (i = 2; i <= n; i++)

    {

        isPrime = true;

        for (j = 0; j < count; j++)

        {

            if (i % b[j] == 0)

            {

                isPrime = false;

                break;

            }

        }

        if (isPrime)

        {

            if (count == b.getSize())

                b.resize(count \* 2);

            b[count++] = i;

        }

    }

    for (i = 0; i < count; i++)

        cout << setw(8) << b[i];

    cout << endl;

    return 0;

}

**例9-5 节点类模板**

// node.h

template<class T>

class Node {

private:

    Node<T>\* next;

public:

    T data;

public:

    Node(const T& data, Node<T>\* next = 0);

    void insertAfter(Node<T>\* p);//在本节点后插入一个节点

    Node<T>\* deleteAfter();//删除本节点的后继节点，并返回其地址

    Node<T>\* nextNode();//返回后继节点的地址

    const Node<T>\* nextNode() const;

};

template<class T>

Node<T>::Node(const T& data, Node<T>\* next)

    :data(data), next(next)

{

}

template<class T>

void Node<T>::insertAfter(Node<T>\* p)

{

    p->next = next;

    next = p;

}

template<class T>

Node<T>\* Node<T>::deleteAfter()

{

    Node<T>\* tempPtr = next;

    if (!next)

        return 0;

    next = tempPtr->next;

    return tempPtr;

}

template<class T>

Node<T>\* Node<T>::nextNode()

{

    return next;

}

template<class T>

const Node<T>\* Node<T>::nextNode() const

{

    return next;

}

**例9-6 链表类模板**

//linkedlist.h

#include <cassert>

#include "Node.h"

template<class T>

class LinkedList {

private:

    Node<T>\* front;     //表头指针

    Node<T>\* rear;      //表尾指针

    Node<T>\* prevPtr;   //前驱指针

    Node<T>\* currPtr;   //当前指针

    int size;           //元素个数

    int position;       //元素序号

    Node<T>\* newNode(const T& item, Node<T>\* ptrNext = nullptr);

    void freeNode(Node<T>\* p);

    void copy(const LinkedList<T>& L);

public:

    LinkedList();

    LinkedList(const LinkedList<T>& L);

    ~LinkedList();

    LinkedList<T>& operator=(const LinkedList<T>& L);

    int getSize();

    bool isEmpty() const;

    void reset(int pos = 0);

    void next();

    bool endOfList() const;

    int currentPosition() const;

    void insertFront(const T& item);//在表头插入节点

    void insertRear(const T& item);//在表尾插入节点

    void insertAt(const T& item);//插入前驱节点

    void insertAfter(const T& item);//插入后继节点

    T deleteFront();//删除头节点

    void deleteCurrent();//删除当前节点

    T& data();//返回当前节点的数据引用

    const T& data() const;

    void clear();//清空链表

};

template<class T>

Node<T>\* LinkedList<T>::newNode(const T& item, Node<T>\* ptrNext)

{

    Node<T>\* p;

    p = new Node<T>(item, ptrNext);

    assert(p);

    return p;

}

template<class T>

void LinkedList<T>::freeNode(Node<T>\* p)

{

    delete p;

}

template<class T>

void LinkedList<T>::copy(const LinkedList<T>& L)

{

    if (L.size == 0)

        return;

    front = rear = newNode(L.front->data);

    for (Node<T>\* srcNode = L.front->nextNode(); srcNode != nullptr; srcNode = srcNode->nextNode()) {

        Node<T>\* p = newNode(srcNode->data);

        rear->insertAfter(p);

        rear = p;

    }

    size = L.size;

    reset(position = L.currentPosition());

}

template<class T>

LinkedList<T>::LinkedList()

    :front(nullptr)

    , rear(nullptr)

    , prevPtr(nullptr)

    , currPtr(nullptr)

    , size(0)

    , position(0)

{

}

template<class T>

LinkedList<T>::LinkedList(const LinkedList<T>& L)

    :front(nullptr)

    , rear(nullptr)

    , prevPtr(nullptr)

    , currPtr(nullptr)

    , size(0)

    , position(0)

{

    copy(L);

}

template<class T>

LinkedList<T>::~LinkedList()

{

    clear();

}

template<class T>

LinkedList<T>& LinkedList<T>::operator=(const LinkedList<T>& L)

{

    clear();

    copy();

    return \*this;

}

template<class T>

int LinkedList<T>::getSize()

{

    return size;

}

template<class T>

bool LinkedList<T>::isEmpty() const

{

    return size == 0;

}

template<class T>

void LinkedList<T>::reset(int pos)

{

    if (pos >= 0 && pos <= size)

    {

        position = 0;

        prevPtr = nullptr;

        currPtr = front;

        while (pos--)

            next();

    }

    else

    {

        position = pos;

        prevPtr = nullptr;

        currPtr = nullptr;

    }

}

template<class T>

void LinkedList<T>::next()

{

    position++;

    prevPtr = currPtr;

    if (currPtr)

        currPtr = currPtr->nextNode();

}

template<class T>

bool LinkedList<T>::endOfList() const

{

    return currPtr == nullptr;

}

template<class T>

int LinkedList<T>::currentPosition() const

{

    return position;

}

template<class T>

void LinkedList<T>::insertFront(const T& item)

{

    front = newNode(item, front);

    if (isEmpty())

        rear = front;

    size++;

    reset(++position);

}

template<class T>

void LinkedList<T>::insertRear(const T& item)

{

    Node<T>\* p = newNode(item);

    if (isEmpty())

    {

        front = rear = p;

    }

    else

    {

        rear->insertAfter(p);

        rear = p;

    }

    size++;

    reset(position);

}

template<class T>

void LinkedList<T>::insertAt(const T& item)

{

    if (currPtr)

    {

        Node<T>\* p = newNode(item, currPtr);

        if (prevPtr)

            prevPtr->insertAfter(p);

        else

        {

            front = prevPtr = p;

        }

        size++;

        reset(++position);

    }

}

template<class T>

void LinkedList<T>::insertAfter(const T& item)

{

    if (currPtr)

    {

        Node<T>\* p = newNode(item, currPtr->nextNode());

        currPtr->insertAfter(p);

        if (rear == currPtr)

            rear = p;

        size++;

    }

}

template<class T>

T LinkedList<T>::deleteFront()

{

    assert(!isEmpty());

    Node<T>\* p = front;

    front = front->nextNode();

    if (--size == 0)

        rear = nullptr;

    reset(--position);

    T item = p->data;

    freeNode(p);

    return item;

}

template<class T>

void LinkedList<T>::deleteCurrent()

{

    if (currPtr)

    {

        if (front == currPtr)

            front = currPtr->nextNode();

        if (rear == currPtr)

            rear = prevPtr;

        if (prevPtr)

            prevPtr->deleteAfter();

        freeNode(currPtr);

        size--;

        reset(position);

    }

}

template<class T>

T& LinkedList<T>::data()

{

    if (!currPtr)

    {

        cerr << "Current node is invalid!" << endl;

        exit(1);

    }

    return currPtr->data;

}

template<class T>

const T& LinkedList<T>::data() const

{

    if (!currPtr)

    {

        cerr << "Current node is invalid!" << endl;

        exit(1);

    }

    return currPtr->data;

}

template<class T>

void LinkedList<T>::clear()

{

    while (isEmpty())

        deleteFront();

}

**例9-7 链表类应用举例**

#include <iostream>

using namespace std;

#include "LinkedList.h"

int main(int argc, char\* argv[])

{

    LinkedList<int> list;

    int i;

    int item;

    cout << "Please enter some integer separated by spaces:";

    for (i = 0; i < 10; i++)

    {

        cin >> item;

        list.insertFront(item);

    }

    cout << "List:";

    list.reset();

    while (!list.endOfList())

    {

        cout << list.data() << " ";

        list.next();

    }

    cout << endl;

    // 输入需要删除的整数

    int key;

    cout << "Please enter some integer needed to be deleted: ";

    cin >> key;

    // 查找并删除节点

    list.reset();

    while (!list.endOfList())

    {

        if (list.data() == key)

            list.deleteCurrent();

        list.next();

    }

    // 输出链表

    cout << "List: ";

    list.reset();

    // 输出各结点数据，直到链表尾

    while (!list.endOfList())

    {

        cout << list.data() << " ";

        list.next();

    }

    cout << endl;

    return 0;

}

**例9-8 栈类模板**

// stack.h

#include <cassert>

template<class T, int SIZE = 50>

class Stack {

private:

    T list[SIZE];

    int top;

public:

    Stack();

    void push(const T& item);

    T pop();

    void clear();

    const T& peek() const;//访问栈顶元素

    bool isEmpty() const;

    bool isFull() const;

};

template<class T, int SIZE>

Stack<T, SIZE>::Stack()

    :top(-1)

{

}

template<class T, int SIZE>

void Stack<T, SIZE>::push(const T& item)

{

    assert(!isFull());

    list[++top] = item;

}

template<class T, int SIZE>

T Stack<T, SIZE>::pop()

{

    assert(!isEmpty());

    return list[top--];

}

template<class T, int SIZE>

void Stack<T, SIZE>::clear()

{

    top = -1;

}

template<class T, int SIZE>

const T& Stack<T, SIZE>::peek() const

{

    assert(!isEmpty());

    return list[top];

}

template<class T, int SIZE>

bool Stack<T, SIZE>::isEmpty() const

{

    return top == -1;

}

template<class T, int SIZE>

bool Stack<T, SIZE>::isFull() const

{

    return top == SIZE - 1;

}

**例9-9 栈的应用。一个简单的整数计算器**

// calculator.h

#include <iostream>

#include <sstream>

#include <cmath>

using namespace std;

#include "stack.h"

class Calculator {

private:

    Stack<double> s;

    void enter(double num);//操作数压栈

    //连续将两个操作数退栈，放在opnd1和opnd2中

    bool getTwoOperands(double& opnd1, double& opnd2);

    void compute(char op);//知行由操作符op指定的运算

public:

    void run();

    void clear();

};

void Calculator::enter(double num)

{

    s.push(num);

}

bool Calculator::getTwoOperands(double& opnd1, double& opnd2)

{

    if (s.isEmpty()) {

        cerr << "Missing operand!" << endl;

        return false;

    }

    opnd1 = s.pop();

    if (s.isEmpty()) {

        cerr << "Missing operand!" << endl;

        return false;

    }

    opnd2 = s.pop();

    return true;

}

void Calculator::compute(char op)

{

    double operand1, operand2;

    bool result = getTwoOperands(operand1, operand2);

    if (result) {

        switch (op) {

        case '+':

            s.push(operand2 + operand1);

            break;

        case '-':

            s.push(operand2 - operand1);

            break;

        case '\*':

            s.push(operand2 \* operand1);

            break;

        case '/':

            if (operand1 == 0) {

                cerr << "Divided by 0!" << endl;

                s.clear();

            }

            else

                s.push(operand2 / operand1);

            break;

        case '^':

            s.push(pow(operand2, operand1));

            break;

        default:

            cerr << "Unrecognized operator!" << endl;

            break;

        }

        cout << "=" << s.peek() << " ";

    }

    else

        s.clear();

}

inline double stringTodouble(const string& str)

{

    istringstream stream(str);

    double result;

    stream >> result;

    return result;

}

void Calculator::run()

{

    string str;

    while (cin >> str, str != "q") {

        switch (str[0]) {

        case 'c':

            s.clear();

            break;

        case '-':

            if (str.size() > 1)

                enter(stringTodouble(str));

            else

                compute(str[0]);

            break;

        case '+':

        case '\*':

        case '/':

        case '^':

            compute(str[0]);

            break;

        default:

            enter(stringTodouble(str));

            break;

        }

    }

}

void Calculator::clear()

{

    s.clear();

}

// 9-9.cpp

#include "Calculator.h"

int main(int argc, char\* argv[])

{

    Calculator c;

    c.run();

    return 0;

}

**例9-10 队列类模板**

#include <cassert>

template<class T, int SIZE = 50>

class Queue

{

private:

int front;

int rear;

int count;

T list[SIZE];

public:

Queue();

void insert(const T& item);

T remove();

void clear();

const T& getFront() const;

int getLength() const;

bool isEmpty() const;

bool isFull() const;

};

template<class T, int SIZE>

Queue<T, SIZE>::Queue()

:front(0), rear(0), conut(0)

{

}

template<class T, int SIZE>

void Queue<T, SIZE>::insert(const T& item)

{

assert(count != SIZE);

count++;

list[rear] = item;

rear = (rear + 1) % SIZE;

}

template<class T, int SIZE>

T Queue<T, SIZE>::remove()

{

assert(count != 0);

int temp = front;

count--;

front = (front + 1) % SIZE;

return list[temp];

}

template<class T, int SIZE>

void Queue<T, SIZE>::clear()

{

count = 0;

front = 0;

rear = 0;

}

template<class T, int SIZE>

const T& Queue<T, SIZE>::getFront() const

{

return list[front];

}

template<class T, int SIZE>

int Queue<T, SIZE>::getLength() const

{

return count;

}

template<class T, int SIZE>

bool Queue<T, SIZE>::isEmpty() const

{

return count == 0;

}

template<class T, int SIZE>

bool Queue<T, SIZE>::isFull() const

{

return count == SIZE;

}

**例9-11 排序与查找算法**

template<class T>

void insertionSort(T a[], int n)

{

int i, j;

T temp;

//将下标为1~n-1的元素逐个插入到已排序序列中适当的位置

for (i = 1; i < n; i++) {

//从a[i-1]开始向a[0]方向扫描各元素，寻找适当位置插入a[i]

j = i;

temp = a[i];

while (j > 0 && temp < a[j - 1]) {

//逐个比较，直到temp>=a[j-1]时，j便是应插入的位置

//若达到j==0，则0是应插入位置

a[j] = a[j - 1];

j--;

}

//插入位置已找到，立即插入

a[j] = temp;

}

}

**例9-12 直接选择排序函数模板**

template<class T>

void mySwap(T& x, T& y)

{

T temp = x;

x = y;

y = temp;

}

template<class T>

void selectionSort(T a[], int n)

{

int i, j;

int leastIndex;

for (i = 0; i < n - 1; i++) {

leastIndex = i;

//在元素a[i+1..a[n-1]中逐个比较，找出最小值

for (j = i + 1; j < n; j++)

if (a[j] < a[leastIndex])//leastIndex始终记录当前找到的最小值的下标

leastIndex = j;

mySwap(a[i], a[leastIndex]);//将这一趟找到的最小元素与a[i]交换

}

}

**例9-13 起泡排序函数模板**

template<class T>

void bubbleSort(T a[], int n)

{

int i, j;

int lastExchangeIndex;

i = n - 1;

while (i > 0) {

lastExchangeIndex = 0;

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

if (a[j + 1] < a[j]) {

mySwap(a[j], a[j + 1]);

lastExchangeIndex = j;

}

i = lastExchangeIndex;

}

}

**例9-14 顺序查找函数模板**

template<class T>

int seqSearch(const T list[], int n, const T& key)

{

int i;

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

if (list[i] == key)

return i;

return -1;

}

**例9-15 折半查找函数模板**

template<class T>

int binSearch(const T list[], int n, const T& key)

{

int low = 0;

int high = n - 1;

int mid;

while (low <= high) {

mid = (low + high) / 2;

if (key == list[mid])

return mid;

else if (key < list[mid])

high = mid - 1;

else

low = mid + 1;

}

return -1;

}

## 第10章 泛型程序设计与标准模板库

**例10-1 从标准输入读入几个整数，存入向量容器，用STL输出它们的相反数**

#include <iostream>

#include <vector>

#include <iterator>

#include <algorithm>

#include <functional>

using namespace std;

int main(int argc, char\* argv[])

{

const int N = 5;

vector<int> s(N);

int i;

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

cin >> s[i];

transform(s.begin(), s.end(), ostream\_iterator<int>(cout, " "), negate<int>());

cout << endl;

return 0;

}

**例10-2 从标准输入读入几个实数，分别将它们的平方输出**

#include <iostream>

#include <iterator>

#include <algorithm>

using namespace std;

int main(int argc, char\* argv[])

{

double square(double x);

transform(istream\_iterator<double>(cin), istream\_iterator<double>(),

ostream\_iterator<double>(cout, "\t"), square);

cout << endl;

return 0;

}

double square(double x)

{

return x \* x;

}

**例10-3 综合迭代器示例（输入结束需按下Ctrl+Z）**

#include <algorithm>

#include <iterator>

#include <vector>

#include <iostream>

using namespace std;

template<class T, class InputIterator, class OutputIterator>

void mySort(InputIterator first, InputIterator last, OutputIterator result)

{

    vector<T> s;

    for (; first != last; ++first)

        s.push\_back(\*first);

    sort(s.begin(), s.end());

    copy(s.begin(), s.end(), result);

}

int main(int argc, char\* argv[])

{

    double b[5] = { 1.2, 2.4, 0.8, 3.3, 3.2 };

    mySort<double>(b, b + 5, ostream\_iterator<double>(cout, " "));

    cout << endl;

    cout<<"Please enter some integers separated by space:";

    mySort<int>(istream\_iterator<int>(cin), istream\_iterator<int>(),

        ostream\_iterator<int>(cout, " "));

    cout << endl;

    return 0;

}

**例10-4 顺序容器的基本操作**

#include <list>

#include <deque>

#include <iterator>

#include <iostream>

using namespace std;

template<class T>

void printContainer(const char\* msg, const T& s)

{

    cout << msg << ": ";

    copy(s.begin(), s.end(), ostream\_iterator<int>(cout, " "));

    cout << endl;

}

int main(int argc, char\* argv[])

{

    deque<int> s;

    int i;

    int x;

    for (i = 0; i < 10; i++) {

        cout<<"Please enter "<<i+1<<"th integer:";

        cin >> x;

        s.push\_front(x);

    }

    printContainer("deque at first", s);

    // 用s容器的内容的逆序构造列表容器l

    list<int> l(s.rbegin(), s.rend());

    printContainer("list at first", l);

    // 将列表容器l的每相邻两个元素顺序颠倒

    list<int>::iterator iter = l.begin();

    int v;

    while (iter != l.end()) {

        v = \*iter;

        iter = l.erase(iter);

        l.insert(++iter, v);

    }

    printContainer("list at last", l);

    // 用列表容器l的内容给s赋值，将s输出

    s.assign(l.begin(), l.end());

    printContainer("deque at last", s);

    return 0;

}

**例10-5 奇偶排序**

#include <vector>

#include <deque>

#include <algorithm>

#include <iterator>

#include<iostream>

using namespace std;

int main()

{

    cout<<"Please enter some integers:";

    istream\_iterator<int>i1(cin), i2; // 建立一对输入流迭代器

    vector<int>s1(i1, i2); // 通过输入流迭代器从标准输入流中输入数据

    sort(s1.begin(), s1.end()); // 将输入的整数排序

    copy(s1.begin(), s1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    deque<int>s2;

    // 以下循环遍历s1

    for (vector<int>::iterator iter = s1.begin(); iter != s1.end(); ++iter)

    {

        if (\*iter % 2 == 0) // 偶数放到s2尾部

            s2.push\_back(\*iter);

        else

            s2.push\_front(\*iter);

    }

    // 将s2的结果输出

    copy(s2.begin(), s2.end(), ostream\_iterator<int>(cout, " "));

    cout << endl;

    return 0;

}

**例10-6 列表容器的接合(splice)操作**

#include <list>

#include <iterator>

#include <string>

#include<iostream>

using namespace std;

int main()

{

string names1[] = { "Alice","Helen","Lucy","Suan" };

string names2[] = { "Bob","David","Levin","Mike" };

list<string>s1(names1, names1 + 4); // 用names1数组的内容构造列表s1

list<string>s2(names2, names2 + 4); // 用names2数组的内容构造列表s2

// 将s1的第一个元素放到s2的最后

s2.splice(s2.end(), s1, s1.begin());

list<string>::iterator iter1 = s1.begin(); // iter1指向s1首

advance(iter1, 2); // iter1前进2个元素，它将指向s1第3个元素

list<string>::iterator iter2 = s2.begin(); // iter2指向s2首

++iter2; // iter2前进1个元素，它将指向s2第2个元素

list<string>::iterator iter3 = iter2; // 用iter2初始化iter3

advance(iter3, 2); // iter3前进2个元素，它将指向s2第4个元素

// 将[iter2, iter3)范围内的结点接到s1中iter1指向的结点前

s1.splice(iter1, s2, iter2, iter3);

//分别将s1和s2输出

copy(s1.begin(), s1.end(), ostream\_iterator<string>(cout, " "));

cout << endl;

copy(s2.begin(), s2.end(), ostream\_iterator<string>(cout, " "));

cout << endl;

return 0;

}

**例10-7 利用栈反向输出单词**

#include <stack>

#include <string>

#include<iostream>

using namespace std;

int main()

{

stack<char>s;

string str;

cin >> str; // 从键盘输入一个字符串

// 将字符串的每个元素顺序压入栈中

for (string::iterator iter = str.begin(); iter != str.end(); ++iter)

s.push(\*iter);

// 将栈中的元素顺序弹出并输出

while (!s.empty())

{

cout << s.top();

s.pop();

}

cout << endl;

return 0;

}

**例10-8 细胞分裂模拟**

#include<iostream>

#include<queue>

#include<ctime>

#include<cstdlib>

using namespace std;

const int TIME\_MIN = 500;

const int TIME\_MAX = 2000;

class Cell;

priority\_queue<Cell> cellQueue;

class Cell

{

private:

    static int cnt;

    int time;

    int id;

public:

    Cell(int birth)

    :id(cnt++)

    {

        time = birth + (rand() % (TIME\_MAX - TIME\_MIN)) + TIME\_MIN;

    }

    int getID() const

    {

        return id;

    }

    int gettime() const

    {

        return time;

    }

    bool operator<(const Cell& s) const

    {

        return time > s.time;

    }

    void split() const

    {

        Cell s1(time), s2(time);

        cout << time << "s:Cell #" << id << " splits to #" << s1.getID() << " and #" << s2.getID() << endl;

        cellQueue.push(s1);

        cellQueue.push(s2);

    }

};

int Cell::cnt = 0;

int main()

{

    srand(static\_cast<unsigned>(time(0)));

    int t;

    cout<<"Please enter the simlation time: ";

    cin >> t;

    cellQueue.push(Cell(0));

    while (cellQueue.top().gettime() < t)

    {

        cellQueue.top().split();

        cellQueue.pop();

    }

    return 0;

}

**例10-9 输入一串实数，将重复的去掉，取最大和最小者的中值，分别输出小于或等于此中值和大于等于此中值的实数.**

#include <set>

#include <iterator>

#include <utility>//定义了pair

#include<iostream>

using namespace std;

int main()

{

    set<double>s;

    double v;

    while (true)

    {

        cout<<"Please enter a real(0 exit):";

        cin >> v;

        if (v == 0)

            break; // 输入0表示结束

        // 尝试将v插入。如果v已存在，输出提示信息

        pair<set<double>::iterator, bool>r = s.insert(v);

        if (!r.second)

            cout << v << " is duplicated" << endl;

    }

    set<double>::iterator iter1 = s.begin(); // 得到第一个元素的迭代器

    set<double>::iterator iter2 = s.end(); // 得到末尾元素的迭代器

    double medium = (\*iter1 + \*(--iter2)) / 2; // 得到最小和最大元素的中值

    // 输出小于或等于中值的元素

    cout << "<=medium: ";

    copy(s.begin(), s.upper\_bound(medium), ostream\_iterator<double>(cout, " "));

    cout << endl;

    // 输出大于或等于中值的元素

    cout << ">=medium: ";

    copy(s.lower\_bound(medium), s.end(), ostream\_iterator<double>(cout, " "));

    cout << endl;

    return 0;

}

**例10-10 有5门课程，每门都有相应学分，从中选择3门，输出学分总和。**

#include <map>

#include <string>

#include <utility>

#include <iostream>

using namespace std;

int main()

{

    map<string, int>courses;

    // 将课程信息插入courses映射中

    courses.insert(make\_pair("CSAPP", 3));

    courses.insert(make\_pair("C++", 2));

    courses.insert(make\_pair("CSARCH", 4));

    courses.insert(make\_pair("COMPILER", 4));

    courses.insert(make\_pair("OS", 5));

    int n = 3; // 剩下的可选次数

    int sum = 0;  // 学分总和

    string name; // 课程名称

    while (n > 0)

    {

        cout<<"Please the course name:";

        cin >> name;

        map<string, int>::iterator iter = courses.find(name); // 查找课程

        if (iter == courses.end()) // 判断是否找到

            cout << name << " is not available" << endl;

        else

        {

            sum += iter->second; // 累加学分

            courses.erase(iter); // 将刚选过的课程从映射中删除

            n--;

        }

    }

    cout << "Total credit: " << sum << endl;

    return 0;

}

**例10-11 统计一句话中每个字母出现的次数。**

#include <map>

#include <cctype>

#include <iostream>

using namespace std;

int main()

{

    map<char, int>s; // 用来存储字母出现次数的映射

    char c; // 存储输入字符

    do {

        cin >> c;

        if (isalpha(c)) {  // 判断是否字母

            c = tolower(c); // 将字母转换为小写

            s[c]++; // 将该字母的出现频率加1

        }

    } while (c != '.');

    // 输出每个字母出现的次数

    for (map<char, int>::iterator iter = s.begin(); iter != s.end(); ++iter)

        cout << iter->first << " " << iter->second << " ";

    cout << endl;

    return 0;

}

**例10-12 上课时间查询。**

#include <map>

#include <utility>

#include <string>

#include<iostream>

using namespace std;

int main()

{

    multimap<string, string>courses;

    typedef multimap<string, string>::iterator CourseIter;

    // 将课程上课时间插入courses映射中

    courses.insert(make\_pair("C++", "2-6"));

    courses.insert(make\_pair("COMPILER", "3-1"));

    courses.insert(make\_pair("COMPILER", "5-2"));

    courses.insert(make\_pair("OS", "1-2"));

    courses.insert(make\_pair("OS", "4-1"));

    courses.insert(make\_pair("OS", "5-5"));

    // 输入一个课程名，直到找到该课程为止，记下每周上课次数

    string name;

    int count;

    do {

        cout<<"Please the course name:";

        cin >> name;

        count = courses.count(name);

        if (count == 0)

            cout << "Cannot find this courses!" << endl;

    } while (count == 0);

    // 输出每周上课次数和上课时间

    cout << count << " lesson(s) per week: ";

    pair<CourseIter, CourseIter>range = courses.equal\_range(name);

    for (CourseIter iter = range.first; iter != range.second; ++iter)

        cout << iter->second << " ";

    cout << endl;

    return 0;

}

**例10-13 利用普通函数来定义函数对象。**

#include <numeric>

#include <iostream>

using namespace std;

int mult(int x, int y);

int main()

{

int a[] = { 1,2,3,4,5 };

const int N = sizeof(a) / sizeof(int);

// 将普通函数mult传递给通用算法

cout << "The result by multipling all elements in a is "

<< accumulate(a, a + N, 1, mult) << endl;

return 0;

}

int mult(int x, int y)

{

return x \* y;

}

**例10-14 利用类来定义函数对象。**

#include <numeric>

#include <iostream>

using namespace std;

class MultClass

{

public:

int operator() (int x, int y) const

{

return x \* y;

}

};

int main()

{

int a[] = { 1,2,3,4,5 };

const int N = sizeof(a) / sizeof(int);

// 将类MultClass传递给通用算法

cout << "The result by multipling all elements in a is "

<< accumulate(a, a + N, 1, MultClass()) << endl;

return 0;

}

**例10-15 利用STL标准函数对象**

#include<numeric>

#include<functional> // 标准函数对象头文件

#include<iostream>

using namespace std;

int main()

{

int a[] = { 1,2,3,4,5 };

const int N = sizeof(a) / sizeof(int);

// 将类MultClass传递给通用算法

cout << "The result by multipling all elements in a is "

<< accumulate(a, a + N, 1, multiplies<int>()) << endl;

return 0;

}

**例10-16 利用STL中的二元谓词函数对象**

#include<numeric>

#include<vector>

#include<iterator>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

int main()

{

    int intArr[] = { 30,90,10,40,70,50,20,80 };

    const int N = sizeof(intArr) / sizeof(int);

    vector<int>a(intArr, intArr + N);

    cout << "before sorting: " << endl;

    copy(a.begin(), a.end(), ostream\_iterator<int>(cout, "\t"));

    cout << endl;

    sort(a.begin(), a.end(), greater<int>());

    cout << "after sorting:" << endl;

    copy(a.begin(), a.end(), ostream\_iterator<int>(cout, "\t"));

    cout << endl;

    return 0;

}

**例10-17 bind2dn产生binder2nd函数适配器实例。**

#include<vector>

#include<iterator>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

int main()

{

int intArr[] = { 30,90,10,40,70,50,20,80 };

const int N = sizeof(intArr) / sizeof(int);

vector<int>a(intArr, intArr + N);

vector<int>::iterator p = find\_if(a.begin(), a.end(), bind2nd(greater<int>(), 40));

if (p == a.end())

cout << "no element greater than 40" << endl;

else

cout << "first element greater than 40 is:" << \*p << endl;

return 0;

}

**例10-18 ptr\_fun，not1和not2产生函数适配器实例。**

#include<vector>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

bool g(int x, int y)

{

return x > y;

}

int main()

{

int intArr[] = { 30,90,10,40,70,50,20,80 };

const int N = sizeof(intArr) / sizeof(int);

vector<int>a(intArr, intArr + N);

vector<int>::iterator p;

p = find\_if(a.begin(), a.end(), bind2nd(ptr\_fun(g), 40));

if (p == a.end())

cout << "no element greater than 40" << endl;

else

cout << "first element greater than 40 is: " << \*p << endl;

p = find\_if(a.begin(), a.end(), not1(bind2nd(greater<int>(), 15)));

if (p == a.end())

cout << "no element is not greater than 15" << endl;

else

cout << "first element that is not greater 15 is: " << \*p << endl;

p = find\_if(a.begin(), a.end(), bind2nd(not2(greater<int>()), 15));

if(p==a.end())

cout << "no element is not greater than 15" << endl;

else

cout << "first element that is not greater 15 is: " << \*p << endl;

return 0;

}

**例10-19 成员函数适配器实例。**

#include<vector>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

struct Car {

int id;

Car(int id) { this->id = id; }

void display() const { cout << "Car " << id << endl; }

};

int main()

{

vector<Car\*>pcars;

vector<Car>cars;

int i;

for (i = 0; i < 5; i++)

pcars.push\_back(new Car(i));

for (i = 5; i < 10; i++)

cars.push\_back(Car(i));

cout << "elements in pcars: " << endl;

for\_each(pcars.begin(), pcars.end(), mem\_fun(&Car::display));

cout << endl;

cout << "elements in cars: " << endl;

for\_each(cars.begin(), cars.end(), mem\_fun\_ref(&Car::display));

cout << endl;

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

delete pcars[i];

return 0;

}

**例10-20 不可变序列算法应用实例。**

#include<vector>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

int main()

{

int iarray[] = { 0,1,2,3,4,5,6,6,6,7,8 };

vector<int> ivector(iarray, iarray + sizeof(iarray) / sizeof(int));

int iarray1[] = { 6,6 };

vector<int> ivector1(iarray1, iarray1 + sizeof(iarray1) / sizeof(int));

int iarray2[] = { 5,6 };

vector<int> ivector2(iarray2, iarray2 + sizeof(iarray2) / sizeof(int));

int iarray3[] = { 0,1,2,3,4,5,7,7,7,9,7 };

vector<int> ivector3(iarray3, iarray3 + sizeof(iarray3) / sizeof(int));

// 找出ivector之中相邻元素值相等的第一个元素

cout << \*adjacent\_find(ivector.begin(), ivector.end()) << endl;

// 找出ivector之中小于7的元素的个数

cout << count\_if(ivector.begin(), ivector.end(), bind2nd(less<int>(), 7))<<endl;

// 找出ivector之中大于2的第一个元素所在位置的元素

cout << \*find\_if(ivector.begin(), ivector.end(), bind2nd(greater<int>(), 2))<<endl;

// 子序列ivector2在ivector中出现的起点位置元素

cout << \*search(ivector.begin(), ivector.end(), ivector2.begin(), ivector2.end()) << endl;

// 查找连续出现3个6的起点位置元素

cout << \*search\_n(ivector.begin(), ivector.end(), 3, 6, equal\_to<int>()) << endl;

// 判断两个区间ivector和ivector3相等否(0为假，1为真)

cout << equal(ivector.begin(), ivector.end(), ivector3.begin()) << endl;

// 查找区间ivector3在ivector中不匹配点的位置

pair<vector<int>::iterator, vector<int>::iterator> result =

mismatch(ivector.begin(), ivector.end(), ivector3.begin());

cout << result.first - ivector.begin() << endl;

return 0;

}

**例10-21 以可变序列算法对数据序列进行复制、生成、删除、替换、倒序、旋转等可变性操作。**

#include<vector>

#include<iterator>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

class evenByTwo {

private:

int x;

public:

evenByTwo() :x(0) {}

int operator()() { return x += 2; }

};

int main()

{

int iarray1[] = { 0,1,2,3,4,4,5,5,6,6,6,6,6,7,8 };

int iarray2[] = { 0,1,2,3,4,5,6,6,6,7,8 };

vector<int> ivector1(iarray1, iarray1 + sizeof(iarray1) / sizeof(int));

vector<int> ivector2(iarray2, iarray2 + sizeof(iarray2) / sizeof(int));

vector<int> ivector3(2);

ostream\_iterator<int> output(cout, " "); // 定义输出流迭代器用于输出数据

// 迭代遍历ivector3区间，每个元素填上-1

fill(ivector3.begin(), ivector3.end(), -1);

copy(ivector3.begin(), ivector3.end(), output); // 使用copy进行输出

cout << endl;

// 迭代遍历ivector3区间，对每一个元素进行evenByTwo操作

generate(ivector3.begin(), ivector3.end(), evenByTwo());

copy(ivector3.begin(), ivector3.end(), output);

cout << endl;

// 将删除元素6后的ivector2序列置于另一个容器ivector4之中

vector<int> ivector4;

remove\_copy(ivector2.begin(), ivector2.end(), back\_inserter(ivector4), 6);

copy(ivector4.begin(), ivector4.end(), output);

cout << endl;

// 删除小于6的元素

ivector2.erase(remove\_if(ivector2.begin(), ivector2.end(), bind2nd(less<int>(), 6)), ivector2.end());

copy(ivector2.begin(), ivector2.end(), output);

cout << endl;

// 将所有的元素值6，改为元素值3

replace(ivector2.begin(), ivector2.end(), 6, 3);

copy(ivector2.begin(), ivector2.end(), output);

cout << endl;

// 逆向重排每一个元素

reverse(ivector2.begin(), ivector2.end());

copy(ivector2.begin(), ivector2.end(), output);

cout << endl;

// 将[ivector2.begin(), ivector2.begin()+3)

// 和[ivector2.begin()+3, ivector2.end())

// 两个区间的元素互换结果直接输出

rotate\_copy(ivector2.begin(), ivector2.begin() + 3, ivector2.end(), output);

cout << endl;

return 0;

}

**例10-22 排序与搜索算法示例。**

#include<vector>

#include<iterator>

#include<functional>

#include<algorithm>

#include<iostream>

using namespace std;

int main()

{

int iarray[] = { 26,17,15,22,23,33,32,40 };

vector<int> ivector(iarray, iarray + sizeof(iarray) / sizeof(int));

// 查找并输出第一个最大值元素及其位置

vector<int>::iterator p = max\_element(ivector.begin(), ivector.end());

int n = p - ivector.begin();

cout << "max element: " << \*p << " found at " << n << endl;

// 局部排序并复制到别处

vector<int> ivector1(5);

partial\_sort\_copy(ivector.begin(), ivector.end(), ivector1.begin(), ivector1.end());

copy(ivector1.begin(),ivector1.end(),ostream\_iterator<int>(cout," "));

cout << endl;

// 排序，默认为递增

sort(ivector.begin(), ivector.end());

copy(ivector.begin(), ivector.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

// 返回小于等于24和大于等于24的元素的位置

cout << \*lower\_bound(ivector.begin(), ivector.end(), 24) << endl;

cout << \*upper\_bound(ivector.begin(), ivector.end(), 24) << endl;

// 对于有序区间，可以用二分查找方法寻找某个元素

cout << binary\_search(ivector.begin(), ivector.end(), 33) << endl;

// 合并两个序列ivector和ivector1，并将结果放到ivector2中

vector<int> ivector2(13);

merge(ivector.begin(), ivector.end(), ivector1.begin(), ivector1.end(), ivector2.begin());

copy(ivector2.begin(), ivector2.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

// 将小于\*(ivector.begin()+5)的元素放置在该元素之左

// 其余置于该元素之右。不保证维持原有的相对位置

nth\_element(ivector2.begin(), ivector2.begin() + 5, ivector2.end());

copy(ivector2.begin(), ivector2.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

// 排序，并保持原来相对位置

stable\_sort(ivector2.begin(), ivector2.end());

copy(ivector2.begin(), ivector2.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

// 合并两个有序序列，然后将结果保存到原区间内

int iarray3[] = { 1,3,5,7,2,4,6,8 };

vector<int> ivector3(iarray3, iarray3 + sizeof(iarray3) / sizeof(int));

inplace\_merge(ivector3.begin(), ivector3.begin() + 4, ivector3.end());

copy(ivector3.begin(), ivector3.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

// 以字典顺序比较序列ivector3和ivector4

int iarray4[] = { 1,3,5,7,1,5,9,3 };

vector<int> ivector4(iarray4, iarray4 + sizeof(iarray4) / sizeof(int));

cout << lexicographical\_compare(ivector3.begin(), ivector3.end(), ivector4.begin(), ivector4.end());

cout << endl;

return 0;

}

**例10-23 数值算法示例。**

#include<iostream>

#include<numeric>

#include<vector>

#include<iterator>

using namespace std;

int main()

{

    int iarray[] = { 1, 2, 3, 4, 5 };

    vector<int> ivector(iarray, iarray + sizeof(iarray) / sizeof(int));

    //元素的累计

    cout<<accumulate(ivector.begin(), ivector.end(), 0)<<endl;

    //向量的内积

    cout<<inner\_product(ivector.begin(), ivector.end(), ivector.begin(), 10)<<endl;

    //向量容器中元素局部求和

    partial\_sum(ivector.begin(), ivector.end(), ostream\_iterator<int>(cout," "));

    cout<<endl;

    //向量容器中相邻元素的差值

    adjacent\_difference(ivector.begin(), ivector.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    return 0;

}

## 第11章 流类库与输入输出

**例11-1 使用width函数控制输出宽度**

#include<iostream>

using namespace std;

int main()

{

    double values[] = {1.23, 3.36, 63.7, 4538.24};

    for(int i = 0; i<4; i++){

        cout.width(10);

        cout<<values[i]<<endl;

    }

    return 0;

}

**例11-2 使用setw操作符指定宽度**

#include<iostream>

#include<iomanip>

#include<string>

using namespace std;

int main()

{

    double values[] = {1.23, 3.36, 63.7, 4538.24};

    string names[] = {"Zoot", "Jimmy", "Al", "Stan"};

    for(int i=0; i<4; i++)

        cout<<setw(6)<<names[i]<<setw(10)<<values[i]<<endl;

    return 0;

}

**例11-3 设置对齐方式**

#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

    double values[] = {1.23, 3.36, 63.7, 4538.24};

    string names[] = {"Zoot", "Jimmy", "Al", "Stan"};

    for(int i=0; i<4; i++)

        cout<<setiosflags(ios\_base::left)<<setw(6)<<names[i]

            <<resetiosflags(ios\_base::left)<<setw(10)<<values[i]<<endl;

    return 0;

}

**例11-4 控制输出精度**

#include<iomanip>

#include<iostream>

#include<string>

using namespace std;

int main()

{

    double values[] = {1.23, 3.36, 63.7, 4538.24};

    string names[] = {"Zoot", "Jimmy", "Al", "Stan"};

    for(int i=0; i<4; i++)

        cout<<setiosflags(ios\_base::left)

            <<setw(6)<<names[i]

            <<resetiosflags(ios\_base::left)

            <<setw(10)<<setprecision(1)<<values[i]<<endl;

    return 0;

}

**例11-5 向文件输出**

#include<fstream>

using namespace std;

struct Date{

    int mondy, day, year;

};

int main()

{

    Date dt = {6, 10, 92};

    ofstream file("date.dat", ios\_base::binary);

    file.write(reinterpret\_cast<char\*>(&dt), sizeof(dt));

    file.close();

    return 0;

}

**例11-6 用ostringstream将数值转换为字符串**

#include<iostream>

#include<sstream>

#include<string>

using namespace std;

template<class T>

inline string toString(const T &v){

    ostringstream os;//创建字符串输出流

    os<<v;          //将变量v的值写入字符串流

    return os.str();//返回输出流生成的字符串

}

int main()

{

    string str1 = toString(5);

    cout<<str1<<endl;

    string str2 = toString(1.2);

    cout<<str2<<endl;

    return 0;

}

**例11-7 get函数应用举例**

#include<iostream>

using namespace std;

int main()

{

    char ch;

    while ((ch=cin.get())!=EOF)

        cout.put(ch);

    return 0;

}

**例11-8 为输入流指定一个终止字符**

#include<iostream>

#include<string>

using namespace std;

int main()

{

    string line;

    cout<<"Type a line terminated by 't' "<<endl;

    getline(cin, line, 't');

    cout<<line<<endl;

    return 0;

}

**例11-9 从一个payroll文件读一个二进制记录到一个结构中**

#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

struct SalaryInfo

{

    unsigned id;

    double salary;

};

int main()

{

    SalaryInfo employee1 = {600001, 8000};

    ofstream os("payroll", ios\_base::out|ios\_base::binary);

    os.write(reinterpret\_cast<char\*>(&employee1), sizeof(employee1));

    os.close();

    ifstream is("payroll", ios\_base::in|ios\_base::binary);

    if(is){

        SalaryInfo employee2;

        is.read(reinterpret\_cast<char\*>(&employee2), sizeof(employee2));

        cout<<employee2.id<<" "<<employee2.salary<<endl;

    }

    else{

        cout<<"ERROR: Cannot open file 'payroll'."<<endl;

    }

    is.close();

    return 0;

}

**例11-10 用seekg函数设置位置指针**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

    int values[] = {3, 7, 0, 5, 4};

    ofstream os("integers", ios\_base::out|ios\_base::binary);

    os.write(reinterpret\_cast<char\*>(values), sizeof(values));

    os.close();

    ifstream is("integers", ios\_base::in|ios\_base::binary);

    if(is){

        is.seekg(3\*sizeof(int));

        int v;

        is.read(reinterpret\_cast<char\*>(&v), sizeof(int));

        cout<<"The 4th integers in the file 'integers' is "<<v<<endl;

    }

    else{

        cout<<"ERROR: Cannot open file 'integers'."<<endl;

    }

    return 0;

}

**例11-11 读一个文件并显示出其中0元素的位置**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

    ifstream file("integers", ios\_base::in|ios\_base::binary);

    int v;

    if(file){

        while (file)

        {

            streampos here = file.tellg();

            file.read(reinterpret\_cast<char\*>(&v), sizeof(int));

            if(file && v == 0)

                cout<<"Position "<<here<<" is 0"<<endl;

        }

    }

    else{

        cout<<"ERROR: Cannot open file 'integers'."<<endl;

    }

    file.close();

    return 0;

}

**例11-12 用ostringstream将字符串转换为数值**

#include<iostream>

#include<sstream>

#include<string>

using namespace std;

template<class T>

inline T fromString(const string &str)

{

    istringstream is(str);

    T v;

    is>>v;

    return v;

}

int main()

{

    int v1 = fromString<int>("5");

    cout<<v1<<endl;

    double v2 = fromString<double>("1.2");

    cout<<v2;

    return 0;

}

## 第12章 异常处理

**例12-1 处理除零异常**

#include<iostream>

using namespace std;

int divide(int x, int y)

{

    if(y==0)

        throw x;

    return x/y;

}

int main()

{

    try{

        cout<<"5/2="<<divide(5,2)<<endl;

        cout<<"8/0="<<divide(8,0)<<endl;

        cout<<"7/1="<<divide(7,1)<<endl;

    }

    catch(int e){

        cout<<e<<" is divideed by zero!"<<endl;

    }

    cout<<"That is ok!"<<endl;

    return 0;

}

**例12-2 使用带析构语义类的C++异常处理**

#include<iostream>

#include<string>

using namespace std;

class MyException

{

private:

    string message;

public:

    MyException(const string &message)

    : message(message)

    {

    }

    ~MyException()

    {

    }

    const string &getMessage() const

    {

        return message;

    }

};

class Demo

{

public:

    Demo()

    {

        cout<<"Constructor of Demo"<<endl;

    }

    ~Demo()

    {

        cout<<"Destructor of Demo"<<endl;

    }

};

void func()

{

    Demo d;

    cout<<"Throw MyException in func()"<<endl;

    throw MyException("exception thrown by func()");

}

int main()

{

    cout<<"In main function"<<endl;

    try{

        func();

    }

    catch(MyException& e){

        cout<<"Caught an excepton: "<<e.getMessage()<<endl;

    }

    cout<<"Resume the execution of main()"<<endl;

    return 0;

}

**例12-3 三角形面积计算**

#include<iostream>

#include<cmath>

#include<stdexcept>

using namespace std;

//输出三角形三边长，计算三角形面积

double area(double a, double b, double c) //throw (invalid\_argument)

{

    //判断三角形边长是否为正

    if(a<=0 || b<=0 || c<=0)

        throw invalid\_argument("the side length should be positive");

    //判断三边长是否满足三角不等式

    if(a+b<=c || b+c<=a || c+a<=b)

        throw invalid\_argument("the side length should fit the triangle inequation");

    //由Heron公式计算三角形面积

    double s = (a+b+c)/2;

    return sqrt(s\*(s-a)\*(s-b)\*(s-c));

}

int main()

{

    double a, b, c;

    cout<<"Please input the side lengths of a triangle:";

    cin>>a>>b>>c;

    try{

        double s=area(a, b, c);

        cout<<"Area: "<<s<<endl;

    }

    catch (exception &e){

        cout<<"Error: "<<e.what()<<endl;

    }

    return 0;

}