## C++简单程序设计

**习题2-26**

//编写一个完整的程序，实现功能

//向用户提问“现在正在下雨吗？”，提示用户输入Y或N

//若输入为Y，显示“现在正在下雨”；若输入为N，显示“现在没有下雨”

//否则继续提问“现在正在下雨吗？”

#include<iostream>

using namespace std;

int main()

{

    bool flag = true;

    char ch;

    while (flag)

    {

        cout<<"现在正在下雨吗？";

        cin>>ch;

        if(ch == 'Y'){

            cout<<"现在正在下雨"<<endl;

            flag = false;

        }

        else if(ch == 'N'){

            cout<<"现在没有下雨"<<endl;

            flag = false;

        }

    }

}

**习题2-27**

//编写一个完整的程序，运行时向用户提问“你考试考了多少分？(0~100)”

//接收输入后判断其等级然后显示出来。规则如下：

//优：90<=分数<=100

//良：80<=分数<90

//中：60<=分数<80

//差：0<=分数<60

#include<iostream>

#include<cstring>

using namespace std;

int main(){

    char \*a;

    bool flag = true;

    int len;

    int x;

    int i;

    while(flag)

    {

        cout<<"请输入考试成绩："<<endl;

        cin>>a;

        len = strlen(a);

        i = 0;

        while(i < len)//对字符串逐个字符进行检查

        {

            if(!isdigit(a[i]))//只要含有一个非数字字符即认为非法输入

            {

                flag = false;

            }

            i++;

        }

        if(!flag)//如果此时标志为false，说明含有非数字字符

        {

            cout<<"您输入有误，请重新输入!"<<endl;//处理异常输入

            flag = true;//标志置成true，继续循环输入

        }

        else//输入全为数字

        {

            flag = false;//输入全为数字，则退出循环，因而标志置成false

            x = atoi(a);

            if(x>100)//如果成绩大于100分，说明输入超出范围，标志置成true，继续循环

            {

                cout<<"输入超出范围!"<<endl;

                flag = true;

            }

            else if(x>=90 && x<=100)

            {

                cout<<"优"<<endl;

            }

            else if(x >= 80 && x<90)

            {

                cout<<"良"<<endl;

            }

            else if(x >= 60 && x<80)

            {

                cout<<"中"<<endl;

            }

            else

                cout<<"差"<<endl;

        }

    }

    return 0;

}

**习题2-28**

//实现一个简单的菜单程序，运行时显示“Menu: A(dd) D(elete) S(ort) Q(uit), Select one:”

//提示用户输入。A表示增加，D表示删除，S表示排序，Q表示退出

//输入为A、D、S时分别提示“数据已经增加、删除、排序”，输入为Q时结束程序

//(1) 要求使用if.. else语句进行判断，用break，continue控制程序流程

//(2) 要求使用switch语句

#include<iostream>

using namespace std;

int main()

{

    char ch;

    bool flag=true;

    while(flag){

        cout<<"Menu: A(dd) D(elete) S(ort) Q(uit), Select one:";

        cin>>ch;

        switch (ch)

        {

        case 'A':

            cout<<"数据已经增加"<<endl;

            break;

        case 'D':

            cout<<"数据已经删除"<<endl;

            break;

        case 'S':

            cout<<"数据已经排序"<<endl;

            break;

        case 'Q':

            flag = false;

            continue;

        default:

            break;

        }

    }

    return 0;

}

**习题2-29**

//用穷举法找出1~100间的质数并显示出来

//分别使用while，do...while，for循环语句实现

bool isPrime(int num);

#include<iostream>

using namespace std;

int main()

{

    int i = 1;

    while(i<=100)

    {

        if(isPrime(i))

            cout<<i<<",";

        i++;

    }

    cout<<endl;

    i=1;

    do

    {

        if(isPrime(i))

            cout<<i<<",";

        i++;

    } while (i<=100);

    cout<<endl;

    for(i=1;i<=100;i++)

    {

        if(isPrime(i))

            cout<<i<<",";

    }

    cout<<endl;

    return 0;

}

bool isPrime(int num)

{

    bool flag = true;

    for(int i=2;i<num;i++){

        if(num%i==0)

            flag = false;

    }

    return flag;

}

**习题2-31**

//声明一个表示时间的结构体，可以精确表示年、月、日、小时、分、秒；

//提示用户输入年、月、日、小时、分、秒的值，然后完整地显示出来

#include<iostream>

using namespace std;

int main()

{

    struct time

    {

        int year;

        int month;

        int day;

        int hour;

        int minute;

        int second;

    };

    struct time t;

    cout<<"Please enter the time(year, month, day, hour, minute, second):";

    cin>>t.year>>t.month>>t.day>>t.hour>>t.minute>>t.second;

    cout<<t.year<<"."<<t.month<<"."<<t.day<<" "<<t.hour<<":"<<t.minute<<":"<<t.second<<endl;

    return 0;

}

**习题2-32**

//在程序中定义一个整型变量，赋予1~100的值

//要求用户猜这个数，比较两个数的大小，把结果提示给用户，直到猜对为止

//分别使用while，do...while语句实现循环

#include<iostream>

using namespace std;

int main()

{

    int x;

    cout<<"Please enter the num:";

    cin>>x;

    bool flag = false;

    int i=0;

    while (i<=100)

    {

        if(i==x)

        {

            flag = true;

            break;

        }

        i++;

    }

    if(flag)

        cout<<"your right! the num is "<<i<<endl;

    else

        cout<<"there is not the num."<<endl;

    return 0;

}

**习题2-33**

//声明枚举类型Weekday，包括SUNDAY到SATURDAY七个元素

//在程序中声明Weekday类型的变量，对其赋值

//声明整型变量，看看能否对其赋Weekday类型的值

#include<iostream>

using namespace std;

int main()

{

    enum Weekday{SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};

    enum Weekday wd;

    wd = WEDNESDAY;

    cout<<(int)wd<<endl;

    return 0;

}

**习题2-34**

//口袋中有红、黄、蓝、白、黑5种颜色的球若干

//每次从口袋中取出3个不同颜色的球，问有多少种取法?

#include<iostream>

using namespace std;

int main()

{

    enum color{red, yellow, blue, white, black};

    color color1;

    int i, j, k, loop;

    int flag = 0; //计数器

    for (i = red; i <= black; i++)

    {

        for (j = red; j <= i; j++)

        {

            for (k = red; k <= j; k++)

            {

                if (i != j && i != k && j != k)

                {

                    flag++;

                    for (loop = 1; loop <= 3; loop++)

                    {

                        switch (loop)

                        {

                        case 1:

                            color1 = color(i);

                            break;

                        case 2:

                            color1 = color(j);

                            break;

                        case 3:

                            color1 = color(k);

                            break;

                        default:

                            break;

                        }

                        switch (color1)

                        {

                        case red:

                            cout << "red"<< ",";

                            break;

                        case yellow:

                            cout << "yellow"<< ",";

                            break;

                        case blue:

                            cout << "blue"<< ",";

                            break;

                        case white:

                            cout << "white"<< ",";

                            break;

                        case black:

                            cout << "black"<< ",";

                            break;

                        default:

                            break;

                        }

                    }

                    cout << endl;

                }

            }

        }

    }

    cout << endl;

    cout << "total: " << flag << endl;

    return 0;

}

**习题2-35**

//输出九九乘法表

#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

    int i,j;

    for (i=1; i<=9; i++)

    {

        for(j=1; j<=i; j++)

        {

            cout<<j<<"\*"<<i<<"="<<setw(2)<<j\*i;

            if(i!=j)

                cout<<", ";

            else

                cout<<endl;

        }

    }

    return 0;

}

## 第3章 函数

**习题3-7**

//完成函数，参数为两个unsigned short int型数

//返回值为第一个参数除以第二个参数的结果，数据类型为short int

//如果第二个参数为0，则返回值为-1

//在主程序中实现输入输出

#include<iostream>

using namespace std;

int main()

{

    short int divide(unsigned short int x, unsigned short int y);

    unsigned short int x, y;

    cout<<"Please enter two integers:";

    cin>>x>>y;

    cout<<divide(x,y)<<endl;

    return 0;

}

short int divide(unsigned short int x, unsigned short int y)

{

    if(y!=0)

        return x/y;

    else

        return -1;

}

**习题3-8**

//编写函数把华氏温度转换为摄氏温度，公式为

//            C=5/9\*(F-32)

//在主程序中提示用户输入一个华氏温度，转化后输出相应的摄氏温度

#include<iostream>

using namespace std;

int main()

{

    float FahrenheitToCentigrade(float FahrenDegree);

    float fa;

    cout<<"Please enter a degree Fahrenheit:";

    cin>>fa;

    cout<<"the degree Centigrade is "<<FahrenheitToCentigrade(fa)<<endl;

    return 0;

}

float FahrenheitToCentigrade(float FahrenDegree)

{

    float C;

    C = (FahrenDegree-32)\*5/9;

    return C;

}

**习题3-9**

//编写函数判别一个数是否质数，在主程序中实现输入输出

#include<iostream>

using namespace std;

int main()

{

    bool isPrime(int num);

    int num;

    cout<<"Please enter an integer:";

    cin>>num;

    if(isPrime(num))

        cout<<num<<" is prime."<<endl;

    else

        cout<<num<<" is not prime."<<endl;

    return 0;

}

bool isPrime(int num)

{

    bool flag = true;

    for(int i=2;i<num;i++){

        if(num%i==0)

            flag = false;

    }

    return flag;

}

**习题3-10**

//编写函数求两个整数的最大公约数和最小公倍数

#include<iostream>

using namespace std;

int greatestCommonDivisor(int a, int b);

int leastCommonMultiple(int a, int b);

int main()

{

    int a,b;

    cout<<"Please enter two integers:";

    cin>>a>>b;

    cout<<"the greatest common divisor with "<<a<<" and "<<b<<" is "<<greatestCommonDivisor(a,b)<<endl;

    cout<<"the least common multiple with "<<a<<" and "<<b<<" is "<<leastCommonMultiple(a,b)<<endl;

    return 0;

}

int greatestCommonDivisor(int a, int b)

{

    if(b==0)

        return 0;

    int c = a%b;

    while (c!=0)

    {

        a = b;

        b = c;

        c = a%b;

    }

    return b;

}

int leastCommonMultiple(int a, int b)

{

    int c;

    c = greatestCommonDivisor(a,b);

    if(c!=0)

        return a\*b/c;

    else

        return 0;

}

**习题3-12**

//在主程序中提示输入整数n，编写函数

//用递归的方法求1+2+...+n的值

#include<iostream>

using namespace std;

int sum1\_n(int n);

int main()

{

    int n;

    cout<<"Please enter an integer:";

    cin>>n;

    if(n==1)

        cout<<"1 = "<<sum1\_n(n)<<endl;

    else if(n==2)

        cout<<"1+2 = "<<sum1\_n(n)<<endl;

    else

        cout<<"1+2+...+"<<n<<" = "<<sum1\_n(n)<<endl;

    return 0;

}

int sum1\_n(int n)

{

    if(n==1)

        return 1;

    else

        return sum1\_n(n-1)+n;

}

**习题3-13**

//用递归的方法编写函数求Fibonacci级数，公式为

//Fn=Fn-1+Fn-2(n>2), F1=F2=1

//观察递归调用的过程

#include<iostream>

using namespace std;

int Fibonacci(int n);

int main()

{

    int i,n;

    cout<<"Please enter item n:";

    cin>>n;

    cout<<"Fibnoacci:";

    for(i=1;i<=n;i++)

        if(i<n)

            cout<<Fibonacci(i)<<",";

        else

            cout<<Fibonacci(i)<<endl;

    return 0;

}

int Fibonacci(int n)

{

    if(n==1)

        return 1;

    else if(n==2)

        return 1;

    else

        return Fibonacci(n-1)+Fibonacci(n-2);

}

**习题3-14**

//用递归的方法编写函数求n阶勒让德多项式的值

//在主程序中实现输入输出，递归公式为

//pn(x)=1 (n=0)

//pn(x)=x (n=1)

//pn(x)=[(2n-1)x\*pn-1(x) - (n-1)pn-2(x)]/n (n>1)

#include<iostream>

using namespace std;

double legendre(double x, int n);

int main()

{

    double x;

    int n;

    cout<<"Please enter n:";

    cin>>n;

    cout<<"Please enter x:";

    cin>>x;

    cout<<"Legendre("<<x<<", "<<n<<")="<<legendre(x, n)<<endl;

    return 0;

}

double legendre(double x, int n)

{

    if(n==0)

        return 1;

    else if(n==1)

        return x;

    else

        return ((2\*n-1)\*x\*legendre(x, n-1) - (n-1)\*legendre(x,n-2))/n;

}

**习题3-15**

//编写递归函数getPower计算x^y

//在同一个程序中针对整型和实型实现两个重载的函数

//int getPower(int x, int y);       //整型形式，当y<0时，返回0

//double getPower(double x, int y); //实型形式

//在主程序中实现输入输出，分别输入一个整数a和一个实数b作为底数

//再输入一个整数m作为指数，输出a^m和b^m

//另外请读者思考，如果在调用getPower函数计算a^m时希望得到一个实型结果

//(实型结果表示范围更大，而且可以准确表示m<0时的结果)

//该如何调用？

#include<iostream>

using namespace std;

int getPower(int x, int y);       //整型形式，当y<0时，返回0

double getPower(double x, int y); //实型形式

int main()

{

    int a;

    double b;

    int m;

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

    cin>>a;

    cout<<"Please enter a real b:";

    cin>>b;

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

    cin>>m;

    if(m<0)

        cout<<a<<"^"<<m<<" = "<<1/(double)getPower(a,-m)<<endl;

    else

        cout<<a<<"^"<<m<<" = "<<getPower(a,m)<<endl;

    if(m<0)

        cout<<b<<"^"<<m<<" = "<<1/getPower(b,-m)<<endl;

    else

        cout<<b<<"^"<<m<<" = "<<getPower(b,m)<<endl;

    return 0;

}

int getPower(int x, int y)

{

    if(y==0)

        return 1;

    else

        return x\*getPower(x, y-1);

}

double getPower(double x, int y)

{

    if(y==0)

        return 1;

    else

        return x\*getPower(x, y-1);

}

## 第4章 类与对象

**习题4-8 定义一个Dog类，包含了age，weight等属性，以及对这些属性操作的方法。实现并调试这个类。**

#include<iostream>

using namespace std;

class Dog

{

private:

    /\* data \*/

    int age;

    int weight;

public:

    Dog(int age, int weight);

    int getAge();

    int getWeight();

    void setAge(int age);

    void setWeight(int weight);

};

Dog::Dog(int age, int weight)

{

    this->age = age;

    this->weight = weight;

}

int Dog::getAge()

{

    return this->age;

}

int Dog::getWeight()

{

    return this->weight;

}

void Dog::setAge(int age)

{

    this->age = age;

}

void Dog::setWeight(int weight)

{

    this->weight = weight;

}

int main()

{

    Dog dog1(2, 6);

    Dog dog2(3, 10);

    cout<<"The dog1: "<<"age = "<<dog1.getAge()<<" weight = "<<dog1.getWeight()<<endl;

    cout<<"The dog2: "<<"age = "<<dog2.getAge()<<" weight = "<<dog2.getWeight()<<endl;

    return 0;

}

**习题4-9 设计并测试一个名为Rectangle的矩形类，其属性为矩形的左下角与右上角两个点的坐标，根据坐标能计算矩形的面积。**

#include<iostream>

using namespace std;

class Rectangle{

private:

    int leftx, bottomy;

    int rightx, topy;

public:

    Rectangle(int leftx, int bottomy, int rightx, int topy);

    int getleftx();

    int getbottomy();

    int getrightx();

    int gettopy();

    int area();

};

Rectangle::Rectangle(int leftx, int bottomy, int rightx, int topy)

{

    this->leftx = leftx;

    this->bottomy = bottomy;

    this->rightx = rightx;

    this->topy = topy;

}

int Rectangle::getleftx()

{

    return leftx;

}

int Rectangle::getbottomy()

{

    return bottomy;

}

int Rectangle::getrightx()

{

    return rightx;

}

int Rectangle::gettopy()

{

    return topy;

}

int Rectangle::area()

{

    return (rightx - leftx)\*(topy - bottomy);

}

int main()

{

    Rectangle rect1(10, 20, 30, 40);

    cout<<"The rect's area is "<<rect1.area()<<endl;

    return 0;

}

/\*

习题4-10设计一个用于人事管理的“人员”类。

由于考虑到通用性，这里只抽象出所有类型人员都具有的属性：编号、性别、出生日期、身份证号等。

其中“出生日期”声明为一个“日期”类内嵌子对象。

用成员函数实现对人员信息的录入和显示。

要求包括：构造函数和析构函数、赋值构造函数、带默认形参值的成员函数、类的组合。

\*/

#include<iostream>

#include<cstring>

using namespace std;

class Date{

public:

    Date(){}

    Date(int y, int m, int d);

    Date(Date& obj);

    ~Date();

    void show();

private:

    int year;

    int month;

    int day;

};

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

:year(y), month(m), day(d)

{

}

Date::Date(Date& obj)

:year(obj.year), month(obj.month), day(obj.day)

{

}

Date::~Date()

{

}

void Date::show()

{

    cout<<year<<": "<<month<<": "<<day<<endl;

}

class Person{

public:

    Person(){}

    Person(int nm, bool gd, Date& bd, string& id1);

    Person(Person& obj);

    ~Person();

    void input();

    void show();

private:

    int num;

    bool gender;

    Date birthday;

    string id;

};

Person::Person(int nm, bool gd, Date& bd, string& id1)

:num(nm), gender(gd), birthday(bd), id(id1)

{

}

Person::Person(Person& obj)

:num(obj.num), gender(obj.gender), birthday(obj.birthday), id(obj.id)

{

}

Person::~Person()

{

}

void Person::input()

{

    int n;

    string sex;

    int y, m, d;

    string id1;

    cout<<"Please enter a num: ";

    cin>>n;

    cout<<"Please enter the sex(female/male):";

    cin>>sex;

    cout<<"Please enter the birthday(year month day):";

    cin>>y>>m>>d;

    cout<<"Please enter the id:";

    cin>>id1;

    num = n;

    if(sex=="female")

        gender = true;

    else

        gender = false;

    birthday = Date(y, m, d);

    id = id1;

}

void Person::show()

{

    cout<<"Num: "<<num<<endl;

    if(gender)

        cout<<"Gender: female"<<endl;

    else

        cout<<"Gender: male"<<endl;

    cout<<"Birthday: ";

    birthday.show();

    cout<<"ID: "<<id<<endl;

}

int main()

{

    Person p1;

    p1.input();

    p1.show();

    return 0;

}

/\*

习题4-11 定义并实现一个矩形类，有长、宽两个属性，由成员函数计算矩形的面积。

\*/

#include<iostream>

using namespace std;

class Rectangle

{

public:

    Rectangle(int l, int w);

    Rectangle(Rectangle& obj);

    ~Rectangle();

    int area();

private:

    int length;

    int width;

};

Rectangle::Rectangle(int l, int w)

:length(l), width(w)

{

}

Rectangle::Rectangle(Rectangle& obj)

:length(obj.length), width(obj.width)

{

}

Rectangle::~Rectangle()

{

}

int Rectangle::area()

{

    return length\*width;

}

int main()

{

    int len, wid;

    cout<<"Please enter length and width of the rectangle separated by space: ";

    cin>>len>>wid;

    Rectangle rect(len, wid);

    cout<<"The area of Rectangle is "<<rect.area()<<endl;

    return 0;

}

/\*

习题4-12 定义一个DataType（数据类型）类，能处理包含字符型、整型、浮点型3种类型的数据，给出其构造函数。

\*/

#include<iostream>

using namespace std;

class DataType

{

public:

    DataType(char c1);

    DataType(int n1);

    DataType(float rl);

    ~DataType(){};

    char getChar();

    int getInt();

    float getFloat();

private:

    char ch;

    int num;

    float real;

};

DataType::DataType(char c1)

:ch(c1)

{

}

DataType::DataType(int n1)

:num(n1)

{

}

DataType::DataType(float r1)

:real(r1)

{

}

char DataType::getChar()

{

    return ch;

}

int DataType::getInt()

{

    return num;

}

float DataType::getFloat()

{

    return real;

}

int main()

{

    DataType charVar('a');

    DataType intVar(4);

    DataType floatVar(static\_cast<float>(4.5));

    cout<<charVar.getChar()<<endl;

    cout<<intVar.getInt()<<endl;

    cout<<floatVar.getFloat()<<endl;

    return 0;

}

/\*

习题4-13 定义一个Circle类，有数据成员radius（半径），成员函数getArea()，计算圆的面积，构造一个Circle的对象进行测试。

\*/

#include<iostream>

using namespace std;

class Circle

{

public:

    Circle(int r):radius(r){};

    float area()

    {

        return 3.1415926\*radius\*radius;

    }

private:

    int radius;

};

int main()

{

    int r;

    cout<<"Please enter the radius of cirlce:";

    cin>>r;

    Circle c1(r);

    cout<<"The area of circle is "<<c1.area()<<endl;

}

/\*

习题4-14 定义一个Tree（树）类，有成员ages（树龄），成员函数grow(int years)对ages加上years，age()显示Tree对象的ages值。

\*/

#include<iostream>

using namespace std;

class Tree

{

public:

    Tree(int a):ages(a){}

    void grow(int years);

    int age();

private:

    int ages;

};

void Tree::grow(int years)

{

    ages += years;

}

int Tree::age()

{

    return ages;

}

int main()

{

    int a;

    cout<<"Please enter the ages of tree:";

    cin>>a;

    Tree tr(a);

    cout<<"The ages of tree is "<<tr.age()<<endl;

    int g = 2;

    tr.grow(g);

    cout<<g<<" years later, the ages of tree is "<<tr.age()<<endl;

    return 0;

}

**习题4-20 定义一个复数类Complex，使得下面的代码能够工作。**

**Complex c1(3, 5);//用复数3+5i初始化c1**

**Complex c2=4.5;//用实数4.5初始化c2**

**c1.add(c2);//将c1与c2相加，结果保存在c1中**

**c1.show();//将c1输出（这时的结果应该是7.5+5i）**

#include <iostream>

using namespace std;

class Complex

{

private:

double real;

double image;

public:

Complex();

Complex(double r, double i);

Complex(double r);

Complex(const Complex& obj);

Complex add(const Complex& c2);

void show();

};

Complex::Complex()

:real(0), image(0)

{

}

Complex::Complex(double r, double i)

: real(r), image(i)

{

}

Complex::Complex(double r)

: real(r), image(0)

{

}

Complex::Complex(const Complex& obj)

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

{

}

Complex Complex::add(const Complex& c2)

{

real = real + c2.real;

image = image + c2.image;

return \*this;

}

void Complex::show()

{

std::cout << real << "+" << image << " i" << std::endl;

}

#include "complex.h"

int main()

{

Complex c1(3, 5);

Complex c2 = 4.5;

c1.add(c2);

c1.show();

return 0;

}

## 第5章 数据的共享与保护

/\*

习题5-7 定义一个Cat类，拥有静态数据成员numOfCats，记录Cat的个体数目；

静态成员函数getNumOfCats()，读取numOfCats。

设计程序测试这个类，体会静态数据成员和静态成员函数的用法。

\*/

#include<iostream>

using namespace std;

class Cat

{

public:

    Cat()

    {

        numOfCats++;

    }

    ~Cat()

    {

        numOfCats--;

    }

    static int getNumOfCats()

    {

        return numOfCats;

    }

private:

    static int numOfCats;

};

int Cat::numOfCats = 0;

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

{

    Cat a;

    cout<<a.getNumOfCats()<<endl;

    Cat b;

    cout << b.getNumOfCats() << endl;

    Cat c;

    cout << c.getNumOfCats() << endl;

    b.~Cat();

    cout<<a.getNumOfCats()<<endl;

    return 0;

}

/\*

习题5-11 在一个文件中定义一个全局变量n，主函数main()，

在另一个文件中定义函数fn1()，在main()中对n赋值，

再调用fn1()，在fn1()中也对n赋值，显示n最后的值。

\*/

#include<iostream>

using namespace std;

#include"fn1.h"

int n;

int main()

{

    n=1;

    fn1(2);

    cout<<n<<endl;

    return 0;

}

//fn1.h

void fn1(int x)

{

    int n;

    n=x;

}

/\*

习题5-12 在函数fn1()中定义一个静态变量n，fn1()中对n的值加1，

在主函数中，调用fn1()十次，显示n的值。

\*/

#include<iostream>

using namespace std;

int fn1();

int main()

{

    int i;

    const int N = 10;

    for(i=0; i<N; i++)

    {

        cout<<fn1()<<endl;

    }

}

int fn1()

{

    static int n;

    n++;

    return n;

}

/\*

习题5-13 定义类X，Y，Z，函数h(X\*)，满足：

类X有私有成员i，Y的成员函数g(X\*)是X的友元函数，实现对X的成员i加1；

类Z是类X的友元类，其成员函数f(X\*)实现对X的成员i加5；

函数h(X\*)是X的友元函数，实现对X的成员i加10。

在一个文件中定义和实现类，在另一个文件中实现main()函数。

\*/

#include<iostream>

using namespace std;

#include"xyz.h"

int main()

{

    X objX;

    X\* p=&objX;

    cout<<"i="<<objX.geti()<<endl;

    Y objY;

    objY.g(p);

    cout<<"i="<<objX.geti()<<endl;

    Z objZ;

    objZ.f(p);

    cout<<"i="<<objX.geti()<<endl;

    h(p);

    cout<<"i="<<objX.geti()<<endl;

    return 0;

}

//xyz.h

class X;

class Z;

class Y

{

public:

    Y();

    ~Y();

    void g(X\* p);

};

class X

{

private:

    int i;

public:

    X();

    X(int i1);

    ~X();

    int geti();

    friend void Y::g(X\* p);

    friend class Z;

    friend void h(X\* p);

};

class Z

{

public:

    Z();

    ~Z();

    void f(X\* p);

};

void Y::g(X\* p)

{

    p->i++;

}

void Z::f(X\* p)

{

    p->i += 5;

}

void h(X\* p)

{

    p->i +=10;

}

Y::Y()

{

}

Y::~Y()

{

}

X::X()

:i(0)

{

}

X::X(int i1)

:i(i1)

{

}

X::~X()

{

}

Z::Z()

{

}

Z::~Z()

{

}

int X::geti()

{

    return i;

}

/\*

习题5-14 定义Boat和Car两个类，二者都有weight属性，

定义二者的一个友元函数getTotalWeight()，计算二者的重量和。

\*/

#include<iostream>

using namespace std;

class Car;

class Boat

{

private:

    int weight;

public:

    Boat(int w);

    ~Boat();

    friend int getTotalWeight(const Car& ,const Boat&);

};

class Car

{

private:

    int weight;

public:

    Car(int weight2);

    ~Car();

    friend int getTotalWeight(const Car& ,const Boat&);

};

Boat::Boat(int w)

:weight(w)

{

}

Boat::~Boat()

{

}

Car::Car(int w)

:weight(w)

{

}

Car::~Car()

{

}

int getTotalWeight(const Car& c ,const Boat& b)

{

    return c.weight+b.weight;

}

int main()

{

    Car c(2);

    Boat b(5000);

    cout<<"Total weight of car and boat is "<<getTotalWeight(c, b)<<endl;

    return 0;

}

## 第6章 数组指针与字符串

/\*

习题6-20 实现一个名为SimpleCircle的简单圆类，

其数据成员int\* itsRadius为一个指向其半径值的指针，存放其半径值。

设计对数据成员的各种操作，给出这个类的完整实现并测试这个类。

\*/

#include <iostream>

using namespace std;

const double PI = 3.1415926;

class SimpleCircle

{

private:

    int\* itsRadius;

public:

    SimpleCircle();

    SimpleCircle(int\* p);

    SimpleCircle(const SimpleCircle& obj);

    ~SimpleCircle();

    void setRadiusPtr(int\* p);

    int\* getRadiusPtr() const;

    double area() const;

    double circum() const;

};

SimpleCircle::SimpleCircle()

    :itsRadius(nullptr)

{

}

SimpleCircle::SimpleCircle(int\* p)

    : itsRadius(p)

{

}

SimpleCircle::SimpleCircle(const SimpleCircle& obj)

{

    itsRadius = new int;

    if (obj.itsRadius)

        \*itsRadius = \*obj.itsRadius;

    else

        itsRadius = nullptr;

}

SimpleCircle::~SimpleCircle()

{

    if (itsRadius)

        delete itsRadius;

}

void SimpleCircle::setRadiusPtr(int\* p)

{

    if (p) {

        itsRadius = new int;

        \*itsRadius = \*p;

    }

    else

        itsRadius = nullptr;

}

int\* SimpleCircle::getRadiusPtr() const

{

    return itsRadius;

}

double SimpleCircle::area() const

{

    if (itsRadius)

        return PI \* (\*itsRadius) \* (\*itsRadius);

    else

        return 0;

}

double SimpleCircle::circum() const

{

    if (itsRadius)

        return 2 \* PI \* (\*itsRadius);

    else

        return 0;

}

int main()

{

    int radius;

    cout << "Please the radius:";

    cin >> radius;

    int\* p = new int(radius);

    SimpleCircle c1(p);

    cout << "the area is " << c1.area() << endl;

    cout << "the cirumference is " << c1.circum() << endl;

    return 0;

}

/\*

例6-21 编写一个函数，统计一条英文句子中字母的个数，在主程序中实现输入输出。

\*/

#include <iostream>

#include <cstring>

using namespace std;

int countSentenceAlphabets(const string& s);

int main()

{

    string s1;

    cout<<"Please enter a string: ";

    getline(cin, s1);

    cout << countSentenceAlphabets(s1) << endl;

    return 0;

}

int countSentenceAlphabets(const string& s)

{

    int k = 0;

    int len = s.length();

    int i;

    for (i = 0; i < len; i++)

    {

        if (s[i] >= 'A' && s[i] <= 'z')

            k++;

    }

    return k;

}

// 例6-22 编写函数void reverse(string& s)，用递归算法使字符串s倒序

#include<iostream>

#include<cstring>

using namespace std;

void reverse(string& s);

int main()

{

    string str = "abcdefg";

    reverse(str);

    cout<<str;

    return 0;

}

void reverse(string& s)

{

    static int head = 0;

    static int end = s.length()-1;

    static int size = s.length()-2;

    char tmp;

    if(size>=2)

    {

        tmp = s[head];

        s[head] = s[end];

        s[end] = tmp;

        head++;

        end--;

        size -= 2;

        reverse(s);

    }

}

/\*

例6-23 设学生人数N=8，提示用户输入N个人的考试成绩，然后计算出他们的平均成绩并显示出来

\*/

#include <iostream>

using namespace std;

double average(double\* p, int n);

int main()

{

    const int N = 8;

    double score[N];

    int i;

    for (i = 0; i < N; i++)

    {

        cout << "Please enter the score of " << i + 1 << "th student:";

        cin >> score[i];

    }

    cout << "the average score is " << average(score, N) << endl;

    return 0;

}

double average(double\* p, int n)

{

    int i;

    double sum = 0.0;

    for (i = 0; i < n; i++)

        sum += p[i];

    return sum / n;

}

/\*

6-26 编写一个矩阵转置的函数，矩阵的行数和列数在程序中由用户输入

\*/

#include <iostream>

#include <cstring>

using namespace std;

class Matrix

{

public:

    Matrix();

    Matrix(int m1, int n1, int \*\*a);

    Matrix(Matrix& obj);

    ~Matrix();

    Matrix transport();

    void show();

private:

    int m;

    int n;

    int \*\*M;

};

Matrix::Matrix()

:m(0), n(0), M(nullptr)

{

}

Matrix::Matrix(int m1, int n1, int \*\*a)

{

    int i,j;

    m = m1;

    n = n1;

    M = new int\*[m];

    for(i=0; i<m; i++)

    {

        M[i] = new int[n];

    }

    for(i=0;i<m1;i++)

    {

        for(j=0;j<n1;j++)

        {

            M[i][j]=a[i][j];

        }

    }

}

Matrix::Matrix(Matrix& obj)

{

    int i, j;

    m = obj.m;

    n = obj.n;

    M = new int\*[m];

    for(i=0; i<m; i++)

        M[i] = new int[n];

    for(i=0; i<m; i++)

        for(j=0; j<n; j++)

            M[i][j] = obj.M[i][j];

}

Matrix::~Matrix()

{

    int i;

    if(M)

    {

        for(i=0;i<m;i++)

            if(M[i])

                delete []M[i];

        delete []M;

    }

}

Matrix Matrix::transport()

{

    int i, j;

    int m1 = n;

    int n1 = m;

    int \*\*p = new int\*[m1];

    for (i = 0; i < m1; i++)

        p[i] = new int[n1];

    for (i = 0; i < m1; i++)

        for (j = 0; j < n1; j++)

            p[i][j] = M[j][i];

    Matrix\* p1 = new Matrix(m1, n1, p);

    return \*p1;

}

void Matrix::show()

{

    int i,j;

    for(i=0;i<m;i++)

    {

        for(j=0;j<n;j++)

        {

            cout<<M[i][j]<<" ";

        }

        cout<<endl;

    }

}

int main()

{

    int m, n;

    cout << "Please enter m and n: ";

    cin >> m >> n;

    int\*\* A;

    int i, j;

    A = new int\* [m];

    for (i = 0; i < m; i++)

        A[i] = new int[n];

    cout << "Please enter matrix A" << endl;

    for (i = 0; i < m; i++)

    {

        cout << "please the element of line " << i + 1 << ":";

        for (j = 0; j < n; j++)

            cin >> A[i][j];

    }

    Matrix M1(m, n, A);

    M1.show();

    cout<<endl;

    M1.transport().show();

    return 0;

}

/\*

6-27 定义一个Employee类，其中包括表示姓名、地址、城市和邮编等属性，

包括setName()和display()等函数。

display()使用cout语句显示姓名、地址、城市和邮编等属性，

函数setName()改变对象的姓名属性，实现并测试这个类。

\*/

#include<iostream>

#include<cstring>

using namespace std;

class Employee

{

public:

    Employee();

    Employee(string& nm, string& ad, string& ct, string& pc);

    Employee(Employee& obj);

    ~Employee();

    void setName(string& nm);

    void display() const;

private:

    string name;

    string address;

    string city;

    string postcode;

};

Employee::Employee()

{

}

Employee::Employee(string& nm, string& ad, string& ct, string& pc)

:name(nm), address(ad), city(ct), postcode(pc)

{

}

Employee::Employee(Employee& obj)

:name(obj.name),

address(obj.address),

city(obj.city),

postcode(obj.postcode)

{

}

Employee::~Employee()

{

}

void Employee::setName(string& nm)

{

    name = nm;

}

void Employee::display() const

{

    cout<<"name: "<<name<<endl;

    cout<<"address: "<<address<<endl;

    cout<<"city: "<<city<<endl;

    cout<<"postcode: "<<postcode<<endl;

}

int main()

{

    string nm = "Zhanglei";

    string ad = "180 Wenhuaxi Road";

    string ct = "Weihai";

    string pc = "264200";

    Employee e1(nm, ad, ct, pc);

    e1.display();

    cout<<endl;

    string nm1 = "Wangyuan";

    e1.setName(nm1);

    e1.display();

    return 0;

}

## 第7章 继承与派生

// 例7-5 定义一个基类Shape，在此基础上派生出Rectangle和Circle，

// 二者都有getArea()函数计算对象的面积。

// 使用Rectangle类创建一个派生类Square

//Shape.h

class Shape

{

public:

   virtual double getArea(void)=0;

};

//Circle.h

#include "Shape.h"

class Circle : public Shape

{

public:

   Circle(double r);

   double getRadius(void);

   void setRadius(double r);

   double getArea(void);

private:

   double r;

};

Circle::Circle(double r)

:r(r)

{

}

double Circle::getRadius(void)

{

   return r;

}

void Circle::setRadius(double r)

{

   this->r = r;

}

double Circle::getArea(void)

{

   return 3.1415926\*r\*r;

}

//Rectangle.h

#include "Shape.h"

class Rectangle : public Shape

{

public:

   Rectangle(double len, double width);

   double getLength(void);

   double getWidth(void);

   void setLength(double len);

   void setWidth(double width);

   double getArea(void);

private:

   double len;

   double width;

};

Rectangle::Rectangle(double len, double width)

:len(len), width(width)

{

}

double Rectangle::getLength(void)

{

   return len;

}

double Rectangle::getWidth(void)

{

   return width;

}

void Rectangle::setLength(double len)

{

   this->len = len;

}

void Rectangle::setWidth(double width)

{

   this->width = width;

}

double Rectangle::getArea(void)

{

   return len\*width;

}

//Square.h

#include"Rectangle.h"

class Square: public Rectangle

{

private:

    double edgeLen;

public:

    Square(double el);

    double getEdgeLen();

    void setEdgeLen(double el);

    double getArea();

};

Square::Square(double el)

:Rectangle(el, el), edgeLen(el)

{

}

double Square::getEdgeLen()

{

    return edgeLen;

}

void Square::setEdgeLen(double el)

{

    edgeLen = el;

}

double Square::getArea()

{

    return edgeLen\*edgeLen;

}

#include"Square.h"

#include<iostream>

using namespace std;

int main()

{

    Square sq(4);

    cout<<sq.getArea()<<endl;

    return 0;

}

//习题7-6 定义一个哺乳动物类Mammal，再由此派生出狗类Dog

//定义一个Dog类对象，观察基类与派生类的构造函数和析构函数的调用顺序

#include<iostream>

using namespace std;

class Mammal

{

public:

    Mammal()

    {

        cout<<"New a mammal object."<<endl;

    }

    ~Mammal()

    {

        cout<<"Destroy a mammal object."<<endl;

    }

};

class Dog: public Mammal

{

public:

    Dog()

    {

        cout<<"New a dog object."<<endl;

    }

    ~Dog()

    {

        cout<<"Destroy a dog object."<<endl;

    }

};

#include"Mammal.h"

int main()

{

    Dog d;

    return 0;

}

//习题7-8 定义一个Document类，有数据成员name，从Document派生出Book类

//增加数据成员pageCount

#include<cstring>

#include<iostream>

using namespace std;

class Document

{

private:

    string name;

public:

    Document()

    {

    }

    Document(string name)

    :name(name)

    {

    }

};

class Book: public Document

{

private:

    int pageCount;

public:

    Book()

    {

    }

    Book(string name, int n)

    : Document(name), pageCount(n)

    {

    }

};

//习题7-9 定义一个基类Base，有两个公有成员函数fn1()，fn2()，私有派生出Derived类，

//如何通过Derived类的对象调用基类的函数fn1()?

#include<iostream>

using namespace std;

class Base

{

public:

    Base();

    void fn1();

    void fn2();

};

Base::Base()

{

    cout<<"New a Base Object."<<endl;

}

void Base::fn1()

{

    cout<<"Call the function fn1()."<<endl;

}

void Base::fn2()

{

    cout<<"Call the function fn2()."<<endl;

}

class Derived: private Base

{

public:

    Derived();

    void callfn1();

};

Derived::Derived()

{

    cout<<"New a Derived Object."<<endl;

}

void Derived::callfn1()

{

    Base::fn1();

}

int main()

{

    Derived dv;

    dv.callfn1();

    return 0;

}

//习题7-10 定义一个Object类，有数据成员weight及相应的操作函数

//由此派生出Box类，增加数据成员height和width及相应的操作函数

//声明一个Box对象，观察构造函数与析构函数的调用顺序

//

#include<iostream>

using namespace std;

class Object

{

private:

    int weight;

public:

    Object();

    Object(int weight);

    ~Object();

    int getWeight() const;

};

Object::Object()

{

    cout<<"New a Default Object."<<endl;

}

Object::Object(int weight)

:weight(weight)

{

    cout<<"New a Object, its weight is "<<weight<<endl;

}

Object::~Object()

{

    cout<<"Destory Object."<<endl;

}

int Object::getWeight() const

{

    return weight;

}

class Box: public Object

{

private:

    int height;

    int width;

public:

    Box();

    Box(int weight, int height, int width);

    ~Box();

};

Box::Box()

{

    cout<<"New a Default Box."<<endl;

}

Box::Box(int weight, int height, int width)

:Object(weight), height(height), width(width)

{

    cout<<"New a Box. its weight is "<<Object::getWeight();

    cout<<", its height is "<<height<<", its width is "<<width<<endl;

}

Box::~Box()

{

    cout<<"Destroy a Box Object."<<endl;

}

int main()

{

    Box b1;

    Box b2(2, 30, 40);

    return 0;

}

//习题7-11 定义一个基类BaseClass，从它派生出类DerivedClass

//BaseClass有成员函数fn1()，fn2()，DerivedClass也有成员函数fn1()，fn2()

//在主函数中声明一个DerivedClass的对象

//分别用DerivedClass的对象以及BaseClass和DerivedClass的指针来调用fn1()，fn2()

//观察运行结果

#include<iostream>

using namespace std;

class BaseClass

{

public:

    BaseClass()

    {

        cout<<"New a BaseClass Object."<<endl;

    }

    ~BaseClass()

    {

        cout<<"Destroy a BaseClass Object."<<endl;

    }

    void fn1()

    {

        cout<<"Call the BaseClass function fn1()."<<endl;

    }

    void fn2()

    {

        cout<<"Call the BaseClass function fn2()."<<endl;

    }

};

class DerivedClass: public BaseClass

{

public:

    DerivedClass()

    {

        cout<<"New a DerivedClass Object."<<endl;

    }

    ~DerivedClass()

    {

        cout<<"Destroy a DerivedClass Object."<<endl;

    }

    void fn1()

    {

        cout<<"Call the DerivedClass function fn1()."<<endl;

    }

    void fn2()

    {

        cout<<"Call the DerivedClass function fn2()."<<endl;

    }

};

int main()

{

    DerivedClass dv;

    dv.fn1();

    cout<<endl;

    BaseClass \*pb = &dv;

    pb->fn1();

    cout<<endl;

    DerivedClass \*pd = &dv;

    pd->fn1();

    return 0;

}

## 第8章 多态性

//习题8-4 请编写一个计数器Counter类，对其重载运算符"+"

#include<iostream>

using namespace std;

class Counter

{

private:

    unsigned int value;

public:

    Counter();

    Counter(unsigned int n);

    Counter(const Counter& obj);

    ~Counter();

    unsigned int getValue() const;

    void setValue(unsigned int n);

    Counter operator+(const Counter& obj);

};

//习题8-5 请编写一个哺乳动物类Mammal，再由此派生出狗类Dog，

//二者都声明speak()成员函数，该函数在基类中被声明为虚函数

//声明一个Dog的对象，通过此对象调用speak函数，观察运行结果

#include<iostream>

using namespace std;

class Mammal

{

public:

    virtual void speak()

    {

        cout<<"Call the mammal speak()."<<endl;

    }

};

class Dog: public Mammal

{

public:

    void speak()

    {

        cout<<"Call the dog speak()."<<endl;

    }

};

int main()

{

    Dog dog;

    dog.speak();

    return 0;

}

//习题8-6 请编写一个抽象类Shape，在此基础上派生出类Rectangle和Circle

//二者都有计算对象面积的函数getArea()、计算对象周长的函数getPerim()

#include<iostream>

using namespace std;

//Shape.h

class Shape

{

public:

   virtual double getArea(void)=0;

   virtual double getPerim(void)=0;

};

class Rectangle : public Shape

{

public:

   double getArea(void);

   double getPerim(void);

};

class Circle : public Shape

{

public:

   double getArea(void);

   double getPerim(void);

};

//习题8-7 对类Point重载"++"、"--"运算符，要求同时重载前缀和后缀的形式

class Point

{

private:

    int x;

    int y;

public:

    Point(int x, int y);

    Point(const Point& obj);

    Point operator++();

    Point operator++(int);

    Point operator--();

    Point operator-(int);

}

//习题8-8 定义一个基类BaseClass，从它派生出类DerivedClass

//BaseClass有成员函数fn1()，fn2()，其中fn1()是虚函数

//DerivedClass也有成员函数fn1()，fn2()

//在主函数中声明一个DerivedClass的对象

//分别用BaseClass和DerivedClass的指针指向DerivedClass的对象

//并通过指针调用fn1()，fn2()，观察运行结果

#include<iostream>

using namespace std;

class BaseClass

{

public:

    virtual void fn1()

    {

        cout<<"Call the BaseClass function fn1()."<<endl;

    }

    void fn2()

    {

        cout<<"Call the BaseClass function fn2()."<<endl;

    }

};

class DerivedClass: public BaseClass

{

public:

      void fn1()

      {

            cout<<"Call the DerivedClass function fn1()."<<endl;

      }

      void fn2()

      {

            cout<<"Call the DerivedClass function fn2()."<<endl;

      }

};

int main()

{

      DerivedClass dv;

      BaseClass \*pb = &dv;

      pb->fn1();

      cout<<endl;

      DerivedClass \*pd = &dv;

      pd->fn1();

      return 0;

}

//习题8-9 请编写程序定义一个基类BaseClass，从它派生出类DerivedClass

//在BaseClass中声明虚析构函数

//在主函数中将一个动态分配的DerivedClass的对象地址赋给一个BaseClass的指针

//然后通过指针释放对象空间，观察程序运行结果

#include<iostream>

using namespace std;

class BaseClass

{

public:

    BaseClass()

    {

        cout<<"New a BaseClass Object."<<endl;

    }

    virtual ~BaseClass()

    {

        cout<<"Destroy a BaseClass Object."<<endl;

    }

};

class DerivedClass: public BaseClass

{

public:

    DerivedClass()

    {

        cout<<"New a DerivedClass Object."<<endl;

    }

    ~DerivedClass()

    {

        cout<<"Destroy a DerivedClass Object."<<endl;

    }

};

int main()

{

    BaseClass \*p = new DerivedClass();

    delete p;

    return 0;

}

//习题8-10 编写程序定义类Point，有数据成员x，y，为其定义友元函数实现重载"+"

class Point

{

private:

    int x, y;

public:

    Point();

    Point(int x, int y);

    Point(const Point& obj);

    ~Point();

    void setX(int x);

    void setY(int y);

    int getX() const;

    int gety() const;

    friend Point operator+(const Point& p1, const Point& p2);

};

//习题8-11 在习题8-6的基础上，通过继承Rectangle得到一个新的类Square

//然后在Shape类中增加一个函数int getVertexCount()

//用来获得当前图形的顶点个数

//用以下几种方法实现，并体会各自的优势：

//(1)使用dynamic\_cast实现Shape::getVertexCount函数

#include<iostream>

using namespace std;

class Shape

{

public:

    virtual int getVertexCount()

    {

        return 0;

    }

};

class Rectangle: public Shape

{

public:

    int getVertexCount()

    {

        return 4;

    }

};

class Square : public Rectangle

{

public:

    int getVertexCount()

    {

        return 4;

    }

};

int main()

{

    Shape\* p;

    Rectangle\* pr;

    Square \*ps;

    Shape sh;

    Rectangle rt;

    Square sq;

    p = &sh;

    cout<<"Shape Vertex: "<<p->getVertexCount()<<endl;

    p = &rt;

    pr = dynamic\_cast<Rectangle\*>(p);

    cout<<"Rectangle Vertex: "<<pr->getVertexCount()<<endl;

    p = &sq;

    ps = dynamic\_cast<Square\*>(p);

    cout<<"Square Vertex: "<<ps->getVertexCount()<<endl;

    return 0;

}

//(2)使用typied实现Shape::getVertexCount函数

#include<iostream>

#include<typeinfo>

using namespace std;

class Shape

{

public:

    virtual int getVertexCount()

    {

        return 0;

    }

};

class Rectangle: public Shape

{

public:

    int getVertexCount()

    {

        return 4;

    }

};

class Square : public Rectangle

{

public:

    int getVertexCount()

    {

        return 4;

    }

};

int main()

{

    Shape ob1;

    Rectangle ob2;

    Square ob3;

    Shape\* p;

    if(typeid(ob1)==typeid(Shape))

    {

        p = &ob1;

        cout<<p->getVertexCount()<<endl;

    }

    if(typeid(ob2)==typeid(Rectangle))

    {

        p = &ob2;

        cout<<dynamic\_cast<Rectangle\*>(p)->getVertexCount()<<endl;

    }

    if(typeid(ob3)==typeid(Square))

    {

        p = &ob3;

        cout<<dynamic\_cast<Square\*>(p)->getVertexCount()<<endl;

    }

    return 0;

}

//(3)将Shape::getVertexCount声明为纯虚函数，在派生类中给出具体实现

#include<iostream>

using namespace std;

class Shape

{

public:

    virtual int getVertexCount()=0;

};

class Rectangle: public Shape

{

public:

    int getVertexCount()

    {

        return 4;

    }

};

class Square : public Rectangle

{

public:

    int getVertexCount()

    {

        return 4;

    }

};

int main()

{

    Shape\* p;

    Rectangle rect;

    p = &rect;

    int n;

    n = p->getVertexCount();

    Square s;

    p=&s;

    n = p->getVertexCount();

    cout<<n<<endl;

    cout<<n<<endl;

    return 0;

}

## 第9章 群体类和群体数据的组织

//习题9-1

//编写程序提示用户输入一个班级中的学生人数n

//再依次提示用户输入n个人在课程A中的考试成绩

//然后计算出平均成绩，显示出来

//请使用数组类模板Array定义浮点型数组存储考试成绩

//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;

}

#include"array.h"

#include<iostream>

using namespace std;

int main()

{

    int n;

    cout<<"Please enter the num of student n: ";

    cin>>n;

    Array<double> score(n);

    int i;

    double x;

    for(i=0; i<score.getSize(); i++)

    {

        cout<<"Please enter the "<<i+1<<"th student score: ";

        cin>>x;

        score[i] = x;

    }

    double sum = 0;

    for(i=0; i<score.getSize(); i++)

        sum += score[i];

    double avg = sum/score.getSize();

    cout<<avg<<endl;

    return 0;

}

//习题9-5

//使用链表类模板，声明两个int类型的链表a和b

//分别插入5个元素，然后把b中的元素加入a的尾部

// 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;

}

//linkedlist.h

#include <cassert>

#include<iostream>

using namespace std;

#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

    //新建一个结点

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

    void freeNode(Node<T>\* p);//删除一个结点

    void copy(const LinkedList<T>& L);//复制一个结点

public:

    LinkedList();//创建空链表

    LinkedList(const LinkedList<T>& L);//链表的复制构造函数

    ~LinkedList();//销毁链表

    LinkedList<T>& operator=(const LinkedList<T>& L);//链表对象赋值操作

    int getSize();//返回链表长度

    bool isEmpty() const;//判断链表是否为空

    void reset(int pos = 0);//初始化游标的位置

    void next();//使游标移动到下一个结点

    bool endOfList() const;//游标是否到了链尾

    int currentPosition() const;//返回游标当前的位置

    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;//返回当前节点数据的常引用

    void clear();//清空链表

};

//新建一个结点

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;

}

//删除一个结点

template<class T>

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

{

    delete p;

}

//复制一个结点

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());

}

//创建空链表

template<class T>

LinkedList<T>::LinkedList()

: front(nullptr)

, rear(nullptr)

, prevPtr(nullptr)

, currPtr(nullptr)

, size(0)

, position(0)

{

}

//链表的复制构造函数

template<class T>

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

: front(nullptr)

, rear(nullptr)

, prevPtr(nullptr)

, currPtr(nullptr)

, size(0)

, position(0)

{

    copy(L);

}

//销毁链表

template<class T>

LinkedList<T>::~LinkedList()

{

    clear();

}

//链表对象赋值操作

template<class T>

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

{

    clear();

    copy();

    return \*this;

}

//返回链表长度

template<class T>

int LinkedList<T>::getSize()

{

    return size;

}

//判断链表是否为空

template<class T>

bool LinkedList<T>::isEmpty() const

{

    return size == 0;

}

//初始化游标的位置

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;

    }

}

//使游标移动到下一个结点

template<class T>

void LinkedList<T>::next()

{

    position++;

    prevPtr = currPtr;

    if(currPtr)

        currPtr = currPtr->nextNode();

}

//游标是否到了链尾

template<class T>

bool LinkedList<T>::endOfList() const

{

    return currPtr == nullptr;

}

//返回游标当前的位置

template<class T>

int LinkedList<T>::currentPosition() const

{

    return position;

}

//在表头插入节点

template<class T>

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

{

    front = newNode(item, front);

    if(isEmpty())

        rear = front;

    size++;

    reset(++position);

}

//在表尾插入节点

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);

}

//插入前驱节点

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);

    }

}

//插入后继节点

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++;

    }

}

//删除头节点

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;

}

//删除当前节点

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);

    }

}

//返回当前节点数据的引用

template<class T>

T& LinkedList<T>::data()

{

    if(!currPtr)

    {

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

        exit(1);

    }

    return currPtr->data;

}

//返回当前节点数据的常引用

template<class T>

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

{

    if(!currPtr)

    {

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

        exit(1);

    }

    return currPtr->data;

}

//清空链表

template<class T>

void LinkedList<T>::clear()

{

    while(isEmpty())

        deleteFront();

}

#include"linkedlist.h"

int main()

{

    LinkedList<int> a, b;

    a.insertRear(1);

    a.insertRear(2);

    a.insertRear(3);

    a.insertRear(4);

    a.insertRear(5);

    b.insertRear(11);

    b.insertRear(12);

    b.insertRear(13);

    b.insertRear(14);

    b.insertRear(15);

    while(!b.endOfList())

    {

        a.insertRear(b.data());

        b.next();

    }

    int i;

    int n = a.getSize();

    for(i=0; i<n; i++)

    {

        if(i<n-1)

            cout<<a.data()<<", ";

        else

            cout<<a.data()<<endl;

        a.next();

    }

    return 0;

}

//习题9-6 通过对从本章的链表类模板LinkedList进行组合

//编写有序链表模板OrderList，添加成员函数insert实现链表元素的有序递增插入

//声明两个int类型有序链表a和b，分别插入5个元素，然后把b中的元素插入a中

#include"linkedlist.h"

template<class T>

class OrderList

{

private:

    LinkedList<T>\* p;

public:

    OrderList();

    ~OrderList();

    LinkedList<T>\* getLinkList() const;

    void insert(const T& item);

};

template<class T>

OrderList<T>::OrderList()

{

    p = new LinkedList<T>();

}

template<class T>

OrderList<T>::~OrderList()

{

    p->clear();

}

template<class T>

LinkedList<T>\* OrderList<T>::getLinkList() const

{

    return p;

}

template<class T>

void OrderList<T>::insert(const T& item)

{

    p->reset(0);

    if(p->isEmpty())

        p->insertFront(item);

    else{

        while(!p->endOfList() && item>p->data())

            p->next();

        if(p->endOfList())

            p->insertRear(item);

        else

            p->insertAt(item);

    }

}

#include"orderlist.h"

int main()

{

    int i, n;

    OrderList<int> a;

    a.insert(27);

    a.insert(38);

    a.insert(25);

    a.insert(66);

    a.insert(41);

    OrderList<int> b;

    b.insert(100);

    b.insert(20);

    b.insert(80);

    b.insert(50);

    b.insert(30);

    LinkedList<int>\* p = b.getLinkList();

    p->reset(0);

    while(!p->endOfList())

    {

        a.insert(p->data());

        p->next();

    }

    p = a.getLinkList();

    p->reset(0);

    n = p->getSize();

    for(i=0; i<n; i++)

    {

        if(i<n-1)

            cout<<p->data()<<", ";

        else

            cout<<p->data()<<endl;

        p->next();

    }

    return 0;

}

//习题9-10 初始化int类型数组

//data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19,

//         2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

//应用本章的直接插入排序函数模板进行排序

//对此函数模板稍做修改，加入输出语句

//在每插入一个待排序元素后显示整个数组

//观察排序过程中数据的变化，加深对插入排序算法的理解

#include<iostream>

using namespace std;

template<class T>

void insertionSort(T a[], int n)

{

    int i, j, k;

    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;

        //加入输出语句，观察排序过程

        cout<<"第"<<i<<"轮: ";

        for(k=0; k<n; k++)

        {

            if(k<n-1)

                cout<<a[k]<<", ";

            else

                cout<<a[k]<<endl;

        }

    }

}

int main()

{

    int data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    insertionSort(data1, 20);

    return 0;

}

//习题9-12 初始化int类型数组

//data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19,

//         2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

//应用本章的直接选择排序函数模板进行排序

//对此函数模板稍做修改，加入输出语句

//每次从待排序序列中选择一个元素添加到已排序序列后，显示整个数组

//观察排序过程中数据的变化，加深对直接排序算法的理解

#include<iostream>

using namespace std;

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, k;

    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

        //加入输出语句，观察排序过程

        cout<<"第"<<i<<"轮: ";

        for(k=0; k<n; k++)

        {

            if(k<n-1)

                cout<<a[k]<<", ";

            else

                cout<<a[k]<<endl;

        }

    }

}

int main()

{

    int data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    selectionSort(data1, 20);

    return 0;

}

//习题9-14 初始化int类型数组

//data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19,

//         2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

//应用本章的起泡排序函数模板进行排序

//对此函数模板稍做修改，加入输出语句

//每完成一趟起泡排序后显示整个数组

//观察排序过程中数据的变化，加深对直接排序算法的理解

#include<iostream>

using namespace std;

template<class T>

void mySwap(T& x, T& y)

{

    T temp = x;

    x = y;

    y = temp;

}

template<class T>

void bubbleSort(T a[], int n)

{

    int i, j, k, count=0;

    int lastExchangeIndex;

    i = n - 1;

    while (i > 0) {

        lastExchangeIndex = 0;

        for (j = 0; j < i; j++)

            if (a[j] > a[j + 1]) {

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

                lastExchangeIndex = j;

            }

        i = lastExchangeIndex;

        //加入输出语句，观察排序过程

        count++;

        cout<<"第"<<count<<"轮: ";

        for(k=0; k<n; k++)

        {

            if(k<n-1)

                cout<<a[k]<<", ";

            else

                cout<<a[k]<<endl;

        }

    }

}

int main()

{

    int data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    bubbleSort(data1, 20);

    return 0;

}

//习题9-15 请编写降序的起泡排序函数模板

//然后在程序中初始化int类型数组

//data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19,

//         2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

//应用降序的起泡排序函数模板进行排序，加入输出语句

//每完成一趟起泡排序后显示整个数组，观察排序过程中数据的变化

#include<iostream>

using namespace std;

template<class T>

void mySwap(T& x, T& y)

{

    T temp = x;

    x = y;

    y = temp;

}

template<class T>

void bubbleSort(T a[], int n)

{

    int i, j, k, count=0;

    int lastExchangeIndex;

    i = n - 1;

    while (i > 0) {

        lastExchangeIndex = 0;

        for (j = 0; j < i; j++)

            if (a[j] < a[j + 1]) {

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

                lastExchangeIndex = j;

            }

        i = lastExchangeIndex;

        //加入输出语句，观察排序过程

        count++;

        cout<<"第"<<count<<"轮: ";

        for(k=0; k<n; k++)

        {

            if(k<n-1)

                cout<<a[k]<<", ";

            else

                cout<<a[k]<<endl;

        }

    }

}

int main()

{

    int data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    bubbleSort(data1, 20);

    return 0;

}

//习题9-17 初始化int类型数组

//data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19,

//         2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

//提示用户输入一个数字，应用本章的顺序查找函数模板找出它的位置

#include<iostream>

using namespace std;

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;

}

int main()

{

    int data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    int value, index;

    cout<<"Please enter the value: ";

    cin>>value;

    index = seqSearch(data1, 20, value);

    cout<<"元素"<<value<<"的索引号为: "<<index<<endl;

    return 0;

}

//习题9-19 初始化int类型数组

//data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19,

//         2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

//先使用任一种算法对其进行排序，然后提示用户输入一个数字

//应用本章的折半查找函数模板找出它的位置

#include<iostream>

using namespace std;

template<class T>

void mySwap(T& x, T& y)

{

    T temp = x;

    x = y;

    y = temp;

}

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;

    }

}

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;

}

int main()

{

    int data1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    bubbleSort(data1, 20);

    int i;

    for(i=0; i<20; i++)

    {

        if(i<19)

            cout<<data1[i]<<", ";

        else

            cout<<data1[i]<<endl;

    }

    int value, index;

    cout<<"Please enter the value: ";

    cin>>value;

    index = binSearch(data1, 20, value);

    cout<<"元素"<<value<<"的索引号为: "<<index<<endl;

    return 0;

}

## 第10章 泛型程序设计与C++标准模板库

//习题10-3 建立一个向量容器的实例s，不断对s调用push\_back向其中增加新的元素

//观察在此过程中s.capacity()的变化

#include<iostream>

#include<vector>

using namespace std;

int main()

{

    vector<int> s;

    bool flag = true;

    int x;

    while(flag)

    {

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

        cin>>x;

        if(x)

        {

            s.push\_back(x);

            cout<<s.capacity()<<endl;//容器的动态存储容量

        }

        else

            flag = false;

    }

    return 0;

}

//习题10-5 约瑟夫问题：n个骑士编号1, 2, ..., n, 围坐在圆桌旁

//编号为1的骑士从1开始报数，报到m的骑士出列，然后下一个位置再从1开始报数

//找出最后留在圆桌旁的骑士编号

//(1) 编写一个函数模板。以一种顺序容器的类型作为模板参数

//    在模板中使用指定类型的顺序容器求解约瑟夫问题，m, n是该函数模板的形参

//(2) 分别以vector<int>, deque<int>, list<int>作为类型参数调用该函数模板

//    调用时将n设为较大的数，将m设为较小的数(例如令n=10000, m=5)

//    观察3种情况下的调用该函数模板所需花费的时间

#include<iostream>

#include<vector>

using namespace std;

const int M=5;//报到的数

const int N=100;//总人数

template<class T>

T Josephus(vector<T>& a, int m)

{

    int c=-1;

    while(a.size()>1)

    {

        c=(c+m)%a.size();

        cout<<a[c]<<", ";

        a.erase(a.begin()+c);

        c--;

    }

    cout<<endl;

    return a[0];

}

int main()

{

    int i;

    vector<int> v;

    for(i=0; i<N; i++)

        v.push\_back(i);

    int last = Josephus(v, M);

    cout<<"The last num is "<<last<<endl;

    return 0;

}

#include<iostream>

#include<queue>

using namespace std;

const int M=5;//报到的数

const int N=100;//总人数

template<class T>

T Josephus(queue<T>& a, int m)

{

    int c=1;

    while(a.size()>1)

    {

        if(m!=c)

        {

            c++;

            a.push(a.front());

            a.pop();

        }

        else

        {

            cout<<a.front()<<", ";

            a.pop();

            c = 1;

        }

    }

    cout<<endl;

    return a.front();

}

int main()

{

    int i;

    queue<int> v;

    for(i=0; i<N; i++)

        v.push(i);

    int last = Josephus(v, M);

    cout<<"The last num is "<<last<<endl;

    return 0;

}

#include<iostream>

#include<list>

using namespace std;

const int M=5;//报到的数

const int N=100;//总人数

template <class T>

T Josephus(list<T>& a, int m)

{

    int c=0;

    typename list<T>::iterator it, now;

    it=a.begin();

    while(a.size()>1)

    {

        c++;

        now = it;

        if(++it==a.end())

            it = a.begin();

        if(m==c)

        {

            cout<<\*now<<", ";

            a.erase(now);

            c=0;

        }

    }

    cout<<endl;

    return a.front();

}

int main()

{

    int i;

    list<int> v;

    for(i=0; i<N; i++)

        v.push\_back(i);

    int last = Josephus(v, M);

    cout<<"The last num is "<<last<<endl;

    return 0;

}

//习题10-6 编写一个具有以下原型的函数模板

//template<class T>

//void exchange(list<T>& l1, list<T>::iterator p1, list<T>& l2, list<T>::iterator p2);

//该模板用于将l1链表的[p1, l1.end())区间和l2链表[p2, l2.end())区间的内容交换

//在主函数中调用该模板，以测试该模板的正确性

#include<iostream>

#include<iterator>

#include<list>

using namespace std;

const int N = 6;

template<class T>

void exchange(list<T>& l1, typename list<T>::iterator p1, list<T>& l2, typename list<T>::iterator p2)

{

    list<T> temp;

    temp.splice(temp.begin(), l1, p1, l1.end());

    l1.splice(l1.end(), l2, p2, l2.end());

    l2.splice(l2.end(), temp, temp.begin(), temp.end());

}

int main()

{

    list<int> a, b;

    int i;

    for(i=0; i<N; i++)

    {

        a.push\_back(2\*i+1);

        b.push\_back(3\*i+1);

    }

    cout<<"a: ";

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

    cout<<endl;

    cout<<"b: ";

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

    cout<<endl;

    cout<<"after exchanging..."<<endl;

    list<int>::iterator p1 = a.begin();

    list<int>::iterator p2 = b.begin();

    exchange(a, p1, b, p2);

    cout<<"a: ";

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

    cout<<endl;

    cout<<"b: ";

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

    cout<<endl;

    return 0;

}

//习题10-7 分别对stack<int>, queue<int>, priority\_queue<int>的实例执行下面的操作:

//调用push函数分别将5, 1, 4, 6压入;

//调用两次pop函数;

//调用push函数分别将2, 3压入;

//调用两次pop函数。

//请问对于三类容器适配器，每次调用pop函数时弹出的元素分别是什么?

//请编写程序验证自己的判断

#include<iostream>

#include<stack>

using namespace std;

int main()

{

    stack<int> s;

    s.push(5);

    s.push(1);

    s.push(4);

    s.push(6);

    cout<<s.top()<<endl;

    s.pop();

    cout<<s.top()<<endl;

    s.pop();

    s.push(2);

    s.push(3);

    cout<<s.top()<<endl;

    s.pop();

    cout<<s.top()<<endl;

    s.pop();

    return 0;

}

#include<iostream>

#include<queue>

using namespace std;

int main()

{

    queue<int> s;

    s.push(5);

    s.push(1);

    s.push(4);

    s.push(6);

    cout<<s.front()<<endl;

    s.pop();

    cout<<s.front()<<endl;

    s.pop();

    s.push(2);

    s.push(3);

    cout<<s.front()<<endl;

    s.pop();

    cout<<s.front()<<endl;

    s.pop();

    return 0;

}

#include<iostream>

#include<queue>

using namespace std;

int main()

{

    priority\_queue<int> s;

    s.push(5);

    s.push(1);

    s.push(4);

    s.push(6);

    cout<<s.top()<<endl;

    s.pop();

    cout<<s.top()<<endl;

    s.pop();

    s.push(2);

    s.push(3);

    cout<<s.top()<<endl;

    s.pop();

    cout<<s.top()<<endl;

    s.pop();

    return 0;

}

//习题10-8 编写一个程序，从键盘输入一个个单词后

//输出该单词是否曾经出现过以及出现次数

//可以尝试分别用多重集合(multiset)和映射(map)两种途径实现

//将二者进行比较

#include<iostream>

#include<set>

#include<string>

using namespace std;

int main()

{

    set<string> s1;

    multiset<string> s2;

    string word;

    //读入单词并统计其出现次数

    cout<<"Enter some words(Ctrl+Z to end):"<<endl;

    while(cin>>word)

    {

        s1.insert(word);

        s2.insert(word);

    }

    //输出结果

    cout<<"word\t\t"<<"times"<<endl;

    set<string>::iterator iter;

    for(iter=s1.begin(); iter!=s1.end(); ++iter)

    {

        cout<<(\*iter)<<": "<<s2.count(\*iter)<<endl;

    }

    return 0;

}

#include<iostream>

#include<map>

#include<string>

using namespace std;

int main()

{

    map<string , int> wordCount;

    string word;

    //读入单词并统计其出现次数

    cout<<"Enter some words(Ctrl+Z to end):"

        <<endl;

    while(cin>>word)

        ++wordCount[word];  //word的出现次数加1

    //输出结果

    cout<<"word\t\t"<<"times"<<endl;

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

        cout<<(\*iter).first<<"\t\t"

            <<(\*iter).second<<endl;

    return 0;

}

//习题10-9 编写程序对比STL中的3个元素交换函数

//swap, iter\_swap和swap\_ranges对数组中的元素进行的交换操作

#include<iostream>

#include<vector>

#include<algorithm>

#include<iterator>

#include<cassert>

using namespace std;

int main()

{

    int a1[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

    int a2[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    vector<int> v1(a1, a1+sizeof(a1)/sizeof(int));

    vector<int> v2(a2, a2+sizeof(a2)/sizeof(int));

    copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    assert(v1.size()==v2.size());

    swap(v1, v2);

    cout<<"after exchange..."<<endl;

    copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

    return 0;

}

#include<iostream>

#include<vector>

#include<algorithm>

#include<iterator>

#include<cassert>

using namespace std;

int main()

{

    int a1[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

    int a2[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    vector<int> v1(a1, a1+sizeof(a1)/sizeof(int));

    vector<int> v2(a2, a2+sizeof(a2)/sizeof(int));

    copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    assert(v1.size()==v2.size());

    vector<int>::iterator it1, it2;

    for(it1=v1.begin(), it2=v2.begin(); it1!=v1.end(), it2!=v2.end(); it1++, it2++)

        iter\_swap(it1, it2);

    cout<<"after exchange..."<<endl;

    copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

    return 0;

}

#include<iostream>

#include<vector>

#include<algorithm>

#include<iterator>

#include<cassert>

using namespace std;

int main()

{

    int a1[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

    int a2[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    vector<int> v1(a1, a1+sizeof(a1)/sizeof(int));

    vector<int> v2(a2, a2+sizeof(a2)/sizeof(int));

    copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    assert(v1.size()==v2.size());

    swap\_ranges(v1.begin(), v1.end(), v2.begin());

    cout<<"after exchange..."<<endl;

    copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

    return 0;

}

//习题10-10 编写一个程序，从键盘输入两组整数(可以看作是两个集合)

//分别输出同属于两组的整数(即两个集合的交集)

//属于至少一组的整数(即两个集合的并集)

//属于第一组但不属于第二组的整数(即两个集合的差集)

//程序中需要用到set\_intersection, set\_union, set\_difference等算法

#include<iostream>

#include<iterator>

#include<set>

#include<algorithm>

using namespace std;

int main()

{

    set<int> s1, s2;

    int m, n, i, t;

    cout<<"Please enter the element num of set s1:";

    cin>>m;

    cout<<"Please enter the first set:";

    for(i=0; i<m; i++)

    {

        cin>>t;

        s1.insert(t);

    }

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

    cout<<endl;

    cout<<"Please enter the element num of set s2:";

    cin>>n;

    cout<<"Please enter the second set:";

    for(i=0; i<n; i++)

    {

        cin>>t;

        s2.insert(t);

    }

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

    cout<<endl;

    cout<<"its set\_intersection: ";

    set<int> c1;

    set\_intersection(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(c1, c1.begin()));

    copy(c1.begin(), c1.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    cout<<"its set\_union: ";

    set<int> c2;

    set\_union(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(c2, c2.begin()));

    copy(c2.begin(), c2.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    cout<<"its set\_difference: ";

    set<int> c3;

    set\_difference(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(c3, c3.begin()));

    copy(c3.begin(), c3.end(), ostream\_iterator<int>(cout, " "));

    cout<<endl;

    return 0;

}

//习题10-11 下面的程序段首先构造了一个元素按升序排列的向量容器s

//然后试图调用unique算法去掉其中的重复元素，并将结果输出

//int arr[] = {1, 1, 4, 4, 5};

//vector<int> s(arr, arr+5);

//unique(s.begin(), s.end());

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

//(1) 以上的输出结果是什么？是否真正达到了去除重复元素的目的？

//    如果未达到目的，应如何对程序进行修改？

//(2) 如果s是列表，是否有更方便高效的方法？

#include<iostream>

#include<vector>

#include<iterator>

#include<algorithm>

using namespace std;

int main()

{

    int arr[] = {1, 1, 4, 4, 5};

    vector<int> s(arr, arr+5);

    s.erase(unique(s.begin(), s.end()), s.end());

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

    return 0;

}

#include<iostream>

#include<list>

#include<iterator>

#include<algorithm>

using namespace std;

int main()

{

    int arr[] = {1, 1, 4, 4, 5};

    list<int> s(arr, arr+5);

    s.erase(unique(s.begin(), s.end()), s.end());

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

    return 0;

}

//习题10-12 编写一个产生器，用来产生0~9范围内的随机数

//建立一个顺序容器，使用该产生器和generate算法为该容器的元素赋值

#include<iostream>

#include<algorithm>

#include<iterator>

#include<vector>

#include<ctime>

using namespace std;

class SrandNum{

private:

    int x;

public:

    SrandNum();

    int operator()();

};

SrandNum::SrandNum()

:x(0)

{

}

int SrandNum::operator()()

{

    static unsigned int seq;//设置定增数，以保证每次种子不同

    srand((unsigned)time(NULL)+(++seq));

    return rand()%10;

}

int main()

{

    vector<int> v(6);

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

    generate(v.begin(), v.end(), SrandNum());

    copy(v.begin(), v.end(), output);

    return 0;

}

//习题10-13 编写一个二元函数对象，用来计算x的y次方，其中x和y都是整数

//利用该函数对象和transform算法，并结合适当的函数适配器

//对于10-12所生成的整数序列中的每个元素n, 分别输出5^n, n^7和n^n

#include<iostream>

#include<algorithm>

#include<iterator>

#include<vector>

#include<ctime>

#include<cmath>

using namespace std;

class SrandNum{

private:

    int x;

public:

    SrandNum();

    int operator()();

};

SrandNum::SrandNum()

:x(0)

{

}

int SrandNum::operator()()

{

    static unsigned int seq;//设置定增数，以保证每次种子不同

    srand((unsigned)time(NULL)+(++seq));

    return rand()%10;

}

template<class T>

class A:public binary\_function<T, T, T>

{

public:

    T operator()(T x, T y) const

    {

        return pow(x, y);

    }

};

template<class T>

class B:public unary\_function<T, T>

{

public:

    T operator()(T x) const

    {

        return pow(x, x);

    }

};

int main()

{

    vector<int> v(6);

    vector<int>::iterator it;

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

    generate(v.begin(), v.end(), SrandNum());

    copy(v.begin(), v.end(), output);

    cout<<endl;

    transform(v.begin(), v.end(), output, bind1st(A<int>(), 5));

    cout<<endl;

    transform(v.begin(), v.end(), output, bind2nd(A<int>(), 7));

    cout<<endl;

    transform(v.begin(), v.end(), output, B<int>());

    cout<<endl;

    return 0;

}

//习题10-17 登陆Boost官方网站，下载最新的Boost程序库，阅读文档，完成以下任务

//(1) 修改例10-3，将主函数中的静态数组a变为array<double, 5>类型的对象

//    对与a相关的代码进行相应修改，比较修改前后的程序

//(2) 修改例10-17，使用Bind程序库的bind代替bind2nd

//(3) 用另一种方式修改例10-17，

//    使用Lambda程序库的相关功能来生成判断一个数是否大于40的函数对象

//(4) 改用unordered\_map实现例10-10的功能

## 第11章 流类库与输入输出

/\*

习题11-3 使用I/O流以文本方式建立一个文件test1.txt，写入字符"已成功写入文件!"

用其他字处理程序打开，看看是否正确写入

\*/

#include<fstream>

#include<cstring>

using namespace std;

int main()

{

    ofstream file1("test1.txt");//使用code runner

    file1<<"已成功写入文件!";

    file1.close();

    return 0;

}

/\*

习题11-4 使用I/O流以文本方式打开11-3题建立的文件test1.txt

读出其内容并显示出来，看看是否正确

\*/

#include<iostream>

#include<fstream>

#include<cstring>

#include<cassert>

using namespace std;

int main()

{

    string a;

    ifstream file1("test1.txt");

    assert(file1);

    file1>>a;

    file1.close();

    cout<<a<<endl;

    return 0;

}

/\*

如出现乱码，点击右下角utf8，将其切换到GBK模式

在终端输入 chcp 查看编码 936代表GBK,65001代表utf8,然后输入chcp 65001即可切换

\*/

/\*

习题11-5 使用I/O流以文本方式打开11-3题建立的文件test1.txt

在文件后面添加字符"已成功添加字符!"

然后读出整个文件的内容显示出来，看看是否正确

\*/

#include<fstream>

#include<cstring>

using namespace std;

int main()

{

    ofstream file1("test1.txt", ios\_base::app);

    file1<<"已成功写入文件!";

    file1.close();

    return 0;

}

/\*

习题11-6 定义一个Dog类，包含体重和年龄两个成员变量及相应的成员函数

声明一个实例dog1，体重为5，年龄为10，使用I/O流把dog1的状态写入磁盘文件

再声明一个实例dog2，通过读文件把dog1的状态赋给dog2

分别使用文本方式和二进制方式操作文件，看看结果有何不同

再看看磁盘文件的ASCII码有何不同

\*/

#include<iostream>

#include<fstream>

#include<cstring>

#include<cassert>

using namespace std;

class Dog

{

private:

    int weight;

    int age;

public:

    Dog();

    ~Dog();

    Dog(int w, int a);

    Dog(const Dog& dog);

    int getWeight() const;

    int getAge() const;

};

Dog::Dog()

{

}

Dog::~Dog()

{

}

Dog::Dog(int w, int a)

:weight(w), age(a)

{

}

Dog::Dog(const Dog& dog)

:weight(dog.weight), age(dog.age)

{

}

int Dog::getWeight() const

{

    return weight;

}

int Dog::getAge() const

{

    return age;

}

int main()

{

    Dog dog1(5, 10);

    string filename;

    cout<<"请输入文件名: ";

    cin>>filename;

    ofstream file1(filename, ios\_base::binary);

    file1.write(reinterpret\_cast<char\*>(&dog1), sizeof(dog1));

    file1.close();

    ifstream file2(filename, ios\_base::binary);

    assert(file2);

    Dog dog2 = dog1;

    file2.read(reinterpret\_cast<char\*>(&dog2), sizeof(dog2));

    cout<<dog2.getWeight()<<", "<<dog2.getAge()<<endl;

    return 0;

}

//习题11-7 观察下面的程序，说明每条语句的作用，写出程序执行的结果

#include<iostream>

using namespace std;

int main()

{

    ios\_base::fmtflags original\_flags=cout.flags();

    cout<<812<<"|";

    cout.setf(ios\_base::left, ios\_base::adjustfield);

    cout.width(10);

    cout<<813<<815<<'\n';

    cout.unsetf(ios\_base::adjustfield);

    cout.precision(2);

    cout.setf(ios\_base::uppercase|ios\_base::scientific);

    cout<<831.0;

    cout.flags(original\_flags);

    return 0;

}

/\*

习题11-8 编写程序提示用户输入一个十进制整数，分别用十进制、八进制和十六进制形式输出

\*/

#include<iostream>

using namespace std;

int main()

{

    int n;

    cout<<"Please enter an integer: ";

    cin>>n;

    cout<<"Dec: "<<dec<<n<<endl;

    cout<<"Oct: "<<oct<<n<<endl;

    cout<<"Hex: "<<hex<<n<<endl;

    return 0;

}

/\*

习题11-9 编写程序实现如下功能：

打开指定的一个文本文件，在每一行前加行号后将其输出到另一个文本文件中

\*/

#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

int main()

{

    ifstream file1("test1.txt");

    ofstream file2("test2.txt");

    string line;

    int i=1;

    while (! getline(file1,line).eof())

    {

        file2<<i++<<" "<<line<<endl;

    }

    file2<<i++<<" "<<line<<endl;

    return 0;

}

/\*

习题11-10 使用宽输入流从一个有中文字符的文本文件中读入所有字符

统计每个字符出现的次数，将统计结果用宽输出流输出到另一个文本文件中

\*/

#include <iostream>

#include <fstream>

#include <cstring>

#include <map>

using namespace std;

int main()

{

    ifstream file1("test1.txt");

    ofstream file2("test2.txt");

    string line, temp;

    int i;

    map<string, int> counter;//UTF-8使用三个字节，所以将宽字符换成三字符

    map<string, int>::iterator iter;

    while(getline(file1, line))

    {

        for(i=0; i<line.size(); i+=3)

        {

            temp = "";

            temp += line[i];

            temp += line[i+1];

            temp += line[i+2];

            counter[temp]++;

        }

    }

    for(iter=counter.begin(); iter!=counter.end(); iter++)

    {

        file2<<iter->first<<", "<<iter->second<<endl;

    }

    file1.close();

    file2.close();

    return 0;

}

## 异常处理

/\*

习题12-4 设计一个异常抽象类Exception，在此基础上派生一个OutOfMemory类响应内存不足

一个RangeError类响应输入的数不在指定范围内，实现并测试这几个类

\*/

#include<iostream>

using namespace std;

class Exception

{

public:

    virtual void show()=0;

};

class OutOfMemory: public Exception

{

public:

    OutOfMemory(){}

    ~OutOfMemory(){}

    virtual void show()

    {

        cout<<"Out of memory."<<endl;

    }

};

class RangeError: public Exception

{

private:

    int num;

public:

    RangeError(){}

    ~RangeError(){}

    RangeError(int n):num(n){}

    virtual void show()

    {

        cout<<"Exception: Range Error."<<endl;

    }

};

int main()

{

    try{

        int n;

        cout<<"Please enter n(1-100): ";

        cin>>n;

        if(n<1 || n>100)

            throw RangeError(n);

        cout<<"n: "<<n<<endl;

    }

    catch(Exception& e)

    {

        e.show();

    }

    try

    {

        bool flag;

        char ch;

        cout<<"Please enter the choice of memory(y/n): ";

        cin>>ch;

        if(ch == 'y' || ch == 'Y')

            cout<<"No memory exception."<<endl;

        else

            throw OutOfMemory();

    }

    catch(Exception& e)

    {

        e.show();

    }

}

/\*

习题12-5 练习使用try, catch语句，在程序中用new分配内存时

如果操作未成功，则用try语句触发一个char类型异常，用catch语句捕获此异常

\*/

#include <iostream>

using namespace std;

int main()

{

    int \*p;

    try

    {

        p = new int[512];

        p = NULL;

        if(!p)

            throw "内存分配错误!";

    }

    catch(const char\* str)

    {

        cout <<"有异常产生："<< str << endl;

    }

    return 0;

}