**C++面向对象程序设计（第3版）**

**ISBN：9787302566939**

**例题源代码**

例1.1

编写程序:

#include<iostream>//用cout输出时需要用此头文件

using namespace std; //使用命名空间std

int main()

{ cout<<"Hello world. This is a C++ program.\\n"; //用C++的方法输出一行

　return 0;

例1.2

编写程序:

#include<iostream> //预处理命令

using namespace std; //使用名字空间std

int main() //主函数首部

{ //函数体开始

int a,b,sum; //定义变量

cin>>a>>b; //输入语句

sum=a+b; //赋值语句

cout<<"a+b="<<sum<<endl; //输出语句

return 0; //如程序正常结束,向操作系统返回一个零值

}运行结果: 123456↙ (如果在运行时从键盘输入此两个整数)

a+b=579 (输出结果)

例1.3

编写程序:

#include<iostream>

using namespace std;

int main()

{ int max(int x,int y);//对max函数作声明

int a,b,c;

cin>>a>>b;

c=max(a,b); //调用max函数

cout<<"max="<<c<<endl;

return 0;

}

int max(int x,int y) //定义max函数

{ int z;

if(x>y) z=x;

else z=y;

return(z);

}

例1.4

编写程序:

#include<iostream>

using namespace std;

class Student//声明一个类,类名为Student

{private: //以下为类中的私有部分

int num; //私有变量num

int score; //私有变量score

　public: //以下为类中公用部分

void setdata() //定义公用函数setdata

{ cin>>num; //输入num的值

cin>>score; //输入score的值

}

void display() //定义公用函数display

{ cout<<"num="<<num<<endl; //输出num的值

cout<<"score="<<score<<endl; //输出score的值

};

}; //类的声明结束

Student stud1,stud2; //定义stud1和stud2为Student类的变量,称为对象

int main() //主函数首部

{ stud1.setdata(); //调用对象stud1的setdata函数

stud2.setdata(); //调用对象stud2的setdata函数

stud1.display(); //调用对象stud1的display函数

stud2.display(); //调用对象stud2的display函数

return 0;

}运行结果: 100198.5 ↙ (输入学生1的学号和成绩)

100276.5 ↙ (输入学生2的学号和成绩)

num=1001 (输出学生1的学号)

score=98.5 (输出学生1的成绩)

num=1002 (输出学生2的学号)

score=76.5 (输出学生2的成绩)

例1.5

编写程序:

#include<iostream>

using namespace std;

int main()

{cout<<"please enter your name and age:"<<endl;

char name［10］;

int age;

cin>>name;

cin>>age;

cout<<"your name is "<<name<<endl;

cout<<"your age is "<< age<<endl;

return 0;

}

例1.6

编写程序:

#include<iostream>

using namespace std;

int max(int a,int b,int c)//求3个整数中的最大者

{ if(b>a) a=b;

if(c>a) a=c;

return a;

}

float max(float a,float b, float c) //求3个实数中的最大者

{ if(b>a) a=b;

　if(c>a) a=c;

return a;

}

long max(long a,long b,long c) //求3个长整数中的最大者

{ if(b>a) a=b;

　if(c>a) a=c;

　return a;

}

int main()

{int a,b,c;

float d,e,f;

long g,h,i;

　cin>>a>>b>>c;

　cin>>d>>e>>f;

　cin>>g>>h>>i;

　int m;

　m=max(a,b,c); //函数值为整型

　cout<<"max\_i="<<m<<endl;

　float n;

　n=max(d,e,f); //函数值为实型

　cout<<"max\_f="<<n<<endl;

　long int p;

　p=max(g,h,i); //函数值为长整型

　cout<<"max\_l="<<p<<endl;

}运行结果: 856↙ (输入3个整数给变量a,b,c)

56.990.76543.1↙ (输入3个实数给变量d,e,f)

67543-56778123↙ (输入3个长整数给变量g,h,i)

max\_i=8 (输出3个整数的最大值)

max\_f=90.765 (输出3个实数的最大值)

max\_l=78123 (输出3个长整数的最大值)

例1.7

编写程序:

#include<iostream>

using namespace std;

int max(int a,int b,int c)//求3个整数中的最大者

{if(b>a) a=b;

if(c>a) a=c;

　 return a;

　}

int max(int a, int b) //求两个整数中的最大者

{if(a>b) return a;

　 else return b;

　}

int main()

{int a=7,b=-4,c=9;

　cout<<"max\_3="<<max(a,b,c)<<endl; //输出3个整数中的最大者

　cout<<"max\_2="<<max(a,b)<<endl; //输出两个整数中的最大者

}

例1.8

编写程序:

#include<iostream>

using namespace std;

template<typename T> //模板声明,其中,T为类型参数

T max(T a,T b,T c) //定义一个通用函数,用T作虚拟的类型名

{if(b>a) a=b;

if(c>a) a=c;

return a;

}

int main()

{ int i1=8,i2=5,i3=6,i;

double d1=56.9,d2=90.765,d3=43.1,d;

long g1=67843,g2=-456,g3=78123,g;

i=max(i1,i2,i3); //调用模板函数,此时T被int取代

d=max(d1,d2,d3); //调用模板函数,此时T被double取代

g=max(g1,g2,g3); //调用模板函数,此时T被long取代

cout<<"i\_max="<<i<<endl;

cout<<"f\_max="<<f<<endl;

cout<<"g\_max="<<g<<endl;

return 0;

}

例1.9

编写程序:

#include<iostream>

using namespace std;

int main()

{int a=10;

int &b=a; //声明b是a的引用

a=aa; //a的值变化了,b的值也应一起变化

cout<<a<<b;

b=b/5; //b的值变化了,a的值也应一起变化

cout<<b<<a;

return 0;

}

例1.10

编写程序:

#include<iostream>

using namespace std;

void swap(int a,int b)

{int temp;

temp=a;

a=b;

b=temp; //实现a和b的值互换

}

int main()

{int i=3,j=5;

swap(i,j);

cout<<i<<","<<j<<endl; //i和j的值未互换

return 0;

}

例1.11

编写程序:

#include<iostream>

using namespace std;

void swap(int p1,int p2)

{int temp;

temp=p1;

p1= p2;

p2=temp;

}

int main()

{int i=3,j=5;

swap(&i,&j);

cout<<i<<","<<j<<endl;

return 0;

}

例1.12

编写程序:

#include<iostream>

using namespace std;

void swap(int &a,int &b)

{int temp;

temp=a;

a=b;

b=temp;

}

int main()

{int i=3,j=5;

swap(i,j);

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

return 0;

}

例1.13

编写程序:

#include<iostream>

using namespace std;

inline int max(int a,int b,int c) //这是一个内置函数,求3个整数中的最大者

{if(b>a) a=b;

if(c>a) a=c;

return a;

　}

int main()

{int i=7,j=10,k=25,m;

　 m=max(i,j,k);

cout<<"max="<<m<<endl;

return 0;

　}

例1.14

编写程序:

#include<iostream>

using namespace std;

float a=13.5;

int main()

{int a=5;

cout<<a;

return 0;

}

例1.15

编写程序:

#include<iostream>

#include<string>

using namespace std;

int main()

{string string1,string2,string3,temp;

cout<<"please input three strings:"; //这是对用户输入的提示

cin>>string1>>string2>>string3; //输入3个字符串

if(string2>string3) {temp=string2;string2=string3;string3=temp;}

//使串2≤串3

if(string1<=string2) cout<<string1<<" "<<string2<<" "<<string3<<endl;

//如果串1≤串2,则串1≤串2≤串3

else if(string1<=string3) cout<<string2<<" "<<string1<<" "<<string3<<endl;

//如果串1>串2,且串1≤串3,则串2<串1≤串3

else cout<<string2<<" "<<string3<<" "<<string1<<endl;

//如果串1>串2,且串1>串3,则串2≤串3<串3

return 0;

}

例1.16

编写程序:

#include<iostream>

#include<string.h>

using namespace std;

struct Student

{char name \[10\];

int num;

char sex;

};

int main()

{Student p;

p=new Student;

strcpy(p->name,"Wang Yun");

p->num=10123;

p->sex='M';

cout<<p->name<<" "<<p->num<<" "<<p->sex<<endl;

delete p;

return 0;

}

例2.1

编写程序:

#include<iostream>

using namespace std;

class Time //声明Time类

{public: //数据成员为公用的

int hour;

int minute;

int sec;

};

int main()

{ Time t1; //定义t1为Time类对象

cin>>t1.hour; //以下3行的作用是输入设定的时间

cin>>t1.minute;

cin>>t1.sec;

cout<<t1.hour<<":"<<t1.minute<<":"<<t1.sec<<endl; //输出时间

return 0;

}

例2.2

程序(a): #include<iostream>

using namespace std;

class Time//声明Time类

{public:

int hour;

int minute;

int sec;

};

int main()

{Time t1; //定义对象t1

cin>>t1.hour; //向t1的数据成员输入数据

cin>>t1.minute;

cin>>t1.sec;

cout<<t1.hour<<":"<<t1.minute<<":"<<t1.sec<<endl; //输出t1中数据成员的值

Time t2; //定义对象t2

cin>>t2.hour; //向t2的数据成员输入数据

cin>>t2.minute;

cin>>t2.sec;

cout<<t2.hour<<":"<<t2.minute<<":"<<t2.sec<<endl; //输出t2中数据成员的值

return 0;

}

程序(b): #include<iostream>

using namespace std;

class Time

{public:

int hour;

int minute;

int sec;

};

int main()

{

void set\_time(Time&);//函数声明

void show\_time(Time&); //函数声明

Time t1; //定义t1为Time类对象

set\_time(t1); //调用set\_time函数,向t1对象中的数据成员输入数据

show\_time(t1); //调用show\_time函数,输出t1对象中的数据

Time t2; //定义t2为Time类对象

set\_time(t2); //调用set\_time函数,向t2对象中的数据成员输入数据

show\_time(t2); //调用show\_time函数,输出t2对象中的数据

return 0;

}

void set\_time(Time& t) //定义函数set\_time ,形参t是引用变量

{

cin>>t.hour; //输入设定的时间

cin>>t.minute;

cin>>t.sec;

}

void show\_time(Time& t) //定义函数show\_time,形参t是引用变量

{

cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl; //输出对象中的数据

}

例2.3

编写程序:

#include<iostream>

using namespace std;

class Time//声明Time类

{public:

void set\_time(); //公用成员函数

void show\_time(); //公用成员函数

private: //数据成员为私有

int hour;

int minute;

int sec;

};

int main()

{

Time t1; //定义对象t1

t1.set\_time(); //调用对象t1的成员函数set\_time,向t1的数据成员输入数据

t1.show\_time(); //调用对象t1的成员函数show\_time,输出t1的数据成员的值

Time t2; //定义对象t2

t2.set\_time(); //调用对象t2的成员函数set\_time,向t2的数据成员输入数据

t2.show\_time(); //调用对象t2的成员函数show\_time,输出t2的数据成员的值

return 0;

}

void Time::set\_time() //在类外定义set\_time函数

{

cin>>hour;

cin>>minute;

cin>>sec;

}

void Time::show\_time() //在类外定义show\_time函数

{

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

}

例2.4

编写程序:

#include<iostream>

using namespace std;

class Array\_max//声明类

{public: //以下3行为成员函数原型声明

void set\_value(); //对数组元素设置值

void max\_value(); //找出数组中的最大元素

void show\_value(); //输出最大值

private:

int array\[10\]; //整型数组

　 int max; //max用来存放最大值

};

void Array\_max::set\_value() //成员函数定义,向数组元素输入数值

{ int i;

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

cin>>array\[i\];

}

void Array\_max::max\_value() //成员函数定义,找数组元素中的最大值

{int i;

max=array\[0\];

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

if(array\[i\]>max) max=array\[i\];

}

void Array\_max::show\_value() //成员函数定义,输出最大值

{cout<<"max="<<max;}

int main()

{Array\_max arrmax; //定义对象arrmax

arrmax.set\_value(); //调用arrmax的set\_value函数,向数组元素输入数值

arrmax.max\_value(); //调用arrmax的max\_value函数,找出数组元素中的最大值

arrmax.show\_value(); //调用arrmax的show\_value函数,输出数组元素中的最大值

return 0;

}

例3.1

编写程序:

#include<iostream>

using namespace std;

class Time//声明Time类

{public: //以下为公用函数

Time() //定义构造成员函数,函数名与类名相同

{hour=0; //利用构造函数对对象中的数据成员赋初值

minute=0;

sec=0;

}

void set\_time(); //成员函数声明

void show\_time(); //成员函数声明

private: //以下为私有数据

int hour;

int minute;

int sec;

};

void Time::set\_time() //定义成员函数,向数据成员赋新值

{cin>>hour;

cin>>minute;

cin>>sec;

}

void Time::show\_time() //定义成员函数,输出数据成员的值

{

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

}

int main() //主函数

{

Time t1; //建立对象t1,同时调用构造函数t1.Time()

t1.set\_time(); //对t1的数据成员赋新值

t1.show\_time(); //显示t1的数据成员的值

Time t2; //建立对象t2,同时调用构造函数t2.Time()

t2.show\_time(); //显示t2的数据成员的值

return 0;

}

例3.2

编写程序:

#include<iostream>

　 using namespace std;

class Box//声明Box类

{public:

Box(int,int,int); //声明带参数的构造函数

int volume(); //声明计算体积的函数

private:

int height; //高

int width; //宽

int length; //长

};

Box::Box(int h,int w,int len) //在类外定义带参数的构造函数

{height=h;

width=w;

length=len;

}

int Box::volume() //定义计算体积的函数

{return(heightwidthlength);

}

int main()

{Box box1(12,25,30); //建立对象box1,并指定box1的高、宽、长的值

cout<<"The volume of box1 is "<<box1.volume()<<endl;

Box box2(15,30,21); //建立对象box2,并指定box2的高、宽、长的值

cout<<"The volume of box2 is "<<box2.volume()<<endl;

return 0;

}

例3.3

编写程序:

#include<iostream>

using namespace std;

class Box

{public:

Box();//声明一个无参的构造函数Box

Box(int h,int w,int len):height(h),width(w),length(len){ }

//定义一个有参的构造函数,用参数的初始化表对数据成员初始化

int volume();//声明成员函数volume

private:

int height;

int width;

int length;

};

Box::Box() //在类外定义无参构造函数Box

{height=10;

width=10;

length=10;

}

int Box::volume() //在类外定义成员函数volume

{return(heightwidthlength);

}

int main()

{Box box1; //建立对象box1,不指定实参

cout<<"The volume of box1 is "<<box1.volume()<<endl;

Box box2(15,30,25); //建立对象box2,指定3个实参

cout<<"The volume of box2 is "<<box2.volume()<<endl;

return 0;

}

例3.4

编写程序:

#include<iostream>

using namespace std;

class Box

{public:

Box(int h=10,int w=10,int len=10);//在声明构造函数Box时指定默认参数

int volume();

private:

int height;

int width;

int length;

};

Box::Box(int h,int w,int len) //在定义Box函数时可以不指定默认参数

{height=h;

width=w;

length=len;

}

int Box::volume()

{return(heightwidthlength);

}

int main()

{

Box box1; //没有给定实参

cout<<"The volume of box1 is "<<box1.volume()<<endl;

Box box2(15); //只给定一个实参

cout<<"The volume of box2 is "<<box2.volume()<<endl;

Box box3(15,30); //只给定两个实参

cout<<"The volume of box3 is "<<box3.volume()<<endl;

Box box4(15,30,20); //给定3个实参

cout<<"The volume of box4 is "<<box4.volume()<<endl;

return 0;

}

例3.5

编写程序:

#include<string>

#include<iostream>

using namespace std;

class Student//声明Student类

{public:

Student(int n,string nam,char s) //定义有参数的构造函数

{num=n;

name=nam;

sex=s;

cout<<"Constructor called."<<endl; //输出有关信息

}

~Student() //定义析构函数

{cout<<"Destructor called."<<endl; } //输出指定的信息

void display() //定义成员函数

{cout<<"num: "<<num<<endl;

cout<<"name: "<<name<<endl;

cout<<"sex: "<<sex<<endl<<endl; }

　 private:

int num;

char name\[10\];

char sex;

　 };

int main() //主函数

{Student stud1(10010,"Wang\_li",'f'); //建立对象stud1

stud1.display(); //输出学生1的数据

Student stud2(10011,"Zhang\_fan",'m'); //定义对象stud2

stud2.display(); //输出学生2的数据

return 0;

}

例3.6

编写程序:

#include<iostream>

using namespace std;

class Box

{public:

Box(int h=10,int w=12,int len=15) :hetght(h),width(w),length(len){}

　//声明有默认参数的构造函数,用参数初始化表对数据成员初始化

int volume();

private:

int height;

int width;

int length;

};

int Box::volume()

{return(heightwidthlength);

}

int main()

{ Box a\[3\]={//定义对象数组

Box(10,12,15), //调用构造函数Box,提供第1个元素的实参

Box(15,18,20), //调用构造函数Box,提供第2个元素的实参

Box(16,20,26) //调用构造函数Box,提供第3个元素的实参

Return 0;

};

　cout<<"volume of a\[0\] is "<<a\[0\].volume()<<endl; //调用a\[0\]的volume函数

cout<<"volume of a\[1\] is "<<a\[1\].volume()<<endl; //调用a\[1\] 的volume函数

cout<<"volume of a\[2\] is "<<a\[2\].volume()<<endl; //调用a\[2\] 的volume函数

}

例3.7

编写程序:

#include<iostream>

using namespace std;

class Time

{public:

Time(int,int,int);//声明结构函数

int hour;

int minute;

int sec;

void get\_time(); //声明公有成员函数

};

Time::Time(int h,int m,int s) //定义结构函数

{hour=h;

minute=m;

sec=s;

}

void Time::get\_time() //定义公有成员函数

{cout<<hour<<":"<<minute<<":" <<sec<<endl;}

int main()

{Time t1(10,13,56); //定义Time类对象t1并初始化

　int p1=&t1.hour; //定义指向整型数据的指针变量p1,并使p1指向t1.hour

　cout<<p1<<endl; //输出p1所指的数据成员t1.hour

　t1.get\_time(); //调用对象t1的公用成员函数get\_time

　Time p2=&t1; //定义指向Time类对象的指针变量p2,并使p2指向t1

　p2->get\_time(); //调用p2所指向对象(即t1)的get\_time函数

void(Time::p3)(); //定义指向Time类公用成员函数的指针变量p3

　p3=&Time::get\_time; //使p3指向Time类公用成员函数get\_time

(t1.p3)(); //调用对象t1中p3所指的成员函数(即t1.get\_time())

return 0;

}

例3.8

编写程序:

#include<iostream>

using namespace std;

class Time

{public:

Time(int,int,int);

int hour;

int minute;

int sec;

};

Time::Time(int h,int m,int s) //定义构造函数

{hour=h;

minute=m;

sec=s;

}

void fun(Time &t) //形参t是Time类对象的引用

{t.hour=18;}

int main()

{Time t1(10,13,56); //t1是Time类对象

fun(t1); //实参是Time类对象,可以通过引用来修改实参t1的值

cout<<t1.hour<<endl; //输出t1.hour的值为18

return 0;

}

例3.9

编写程序:

#include<iostream>

using namespace std;

class Box

{public:

Box(int=10,int=10,int=10); //声明有默认参数的构造函数

int volume();

private:

int height;

int width;

int length;

};

Box::Box(int h,int w,int len) //定义构造函数

{height=h;

width=w;

length=len;

}

int Box::volume() //定义volume函数

{return(heightwidthlength); //返回体积

}

int main()

{Box box1(15,30,25),box2; //定义两个对象box1和box2

cout<<"The volume of box1 is "<<box1.volume()<<endl;

box2=box1; //将box1的值赋给box2

cout<<"The volume of box2 is "<<box2.volume()<<endl;

return 0;

}

例3.10

编写程序:

#include<iostream>

using namespace std;

class Box

{public:

Box(int,int);

int volume();

static int height; //把height定义为公用的静态的数据成员

int width;

int length;

};

Box::Box(int w,int len) //通过构造函数对width和length赋初值

{width=w;

length=len;

}

int Box::volume() //定义成员函数volume

{return(heightwidthlength);

}

int Box::height=10; //对公用静态数据成员height初始化

int main()

{

Box a(15,20),b(20,30); //建立两个对象

cout<<a.height<<endl; //通过对象名a引用静态数据成员

cout<<b.height<<endl; //通过对象名b引用静态数据成员

cout<<Box::height<<endl; //通过类名引用静态数据成员

cout<<a.volume()<<endl; //调用volume函数,计算体积,输出结果

return 0;

}

例3.11

编写程序:

#include<iostream>

using namespace std;

class Student//定义Student类

{public:

Student(int n,int a,float s):num(n),age(a),score(s){ } //定义构造函数

void total(); //声明成员函数

static float average(); //声明静态成员函数

private:

int num;

int age;

float score;

static float sum; //静态数据成员sum(总分)

static int count; //静态数据成员count(计数)

};

void Student::total() //定义非静态成员函数

{sum+=score; //累加总分

count++; //累计已统计的人数

}

float Student::average() //定义静态成员函数

{return(sum/count);

}

float Student::sum=0; //对静态数据成员初始化

int Student::count=0; //对静态数据成员初始化

int main()

{Student stud\[3\]={ //定义对象数组并初始化

Student(1001,18,70),

Student(1002,19,78),

Student(1005,20,98)

};

int n;

cout<<"please input the number of students:";

cin>>n; //输入需要求前面多少名学生的平均成绩

for(int i=0;i<n;i++) //调用3次total函数

stud\[i\].total();

cout<<"the average score of "<<n<<" students is "<<Student::average()

<<endl; //调用静态成员函数

return 0;

}

例3.12

编写程序:

#include<iostream>

using namespace std;

class Time

{public:

Time(int,int,int); //声明构造函数

friend void display(Time &); //声明display函数为Time类的友元函数

private: //以下数据是私有数据成员

int hour;

int minute;

int sec;

};

Time::Time(int h,int m,int s) //定义构造函数,给hour,minute,sec赋初值

{hour=h;

minute=m;

sec=s;

}

void display(Time& t) //这是普通函数,形参t是Time类对象的引用

{cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl;}

int main()

{ Time t1(10,13,56);

display(t1); //调用display函数,实参t1是Time类对象

　return 0;

}

例3.13

编写程序:

#include<iostream>

using namespace std;

class Date;//对Date类的提前引用声明

class Time //声明Time类

{public:

Time(int,int,int); //声明构造函数

void display(Date &); //display是成员函数,形参是Date类对象的引用

private:

int hour;

int minute;

int sec;

};

class Date //声明Date类

{public:

Date(int,int,int); //声明构造函数

friend void Time::display(Date &);//声明Time中的display函数为本类

//的友元成员函数

private:

int month;

int day;

int year;

};

Time::Time(int h,int m,int s) //定义类Time的构造函数

{hour=h;

minute=m;

sec=s;

}

void Time::display(Date &d) //display的作用是输出年、月、日和时、分、秒

{cout<<d.month<<"/"<<d.day<<"/"<<d.year<<endl; //引用Date类对象中的私有数据

cout<<hour<<":"<<minute<<":"<<sec<<endl; //引用本类对象中的私有数据

}

Date::Date(int m,int d,int y) //类Date的构造函数

{month=m;

day=d;

year=y;

}

例3.14

编写程序:

#include<iostream>

using namespace std;

template<class numtype> //声明类模板,虚拟类型名为numtype

class Compare //类模板名为Compare

{public:

Compare(numtype a,numtype b) //定义构造函数

{x=a;y=b;}

numtype max() //函数类型暂定为numtype

{return(x>y)?x:y;}

numtype min()

{return(x<y)?x:y;}

private:

numtype x,y; //数据类型暂定为numtype

};

int main()

{Compare<int> cmp1(3,7); //定义对象cmp1,用于两个整数的比较

cout<<cmp1.max()<<" is the Maximum of two integer numbers."<<endl;

cout<<cmp1.min()<<" is the Minimum of two integer numbers."<<endl<<endl;

Compare<float> cmp2(45.78,93.6); //定义对象cmp2,用于两个浮点数的比较

cout<<cmp2.max()<<" is the Maximum of two float numbers."<<endl;

cout<<cmp2.min()<<" is the Minimum of two float numbers."<<endl<<endl;

Compare<char> cmp3('a','A'); //定义对象cmp3,用于两个字符的比较

cout<<cmp3.max()<<" is the Maximum of two characters."<<endl;

cout<<cmp3.min()<<" is the Minimum of two characters."<<endl;

return 0;

}

int main()

{Time t1(10,13,56); //定义Time类对象t1

Date d1(12,25,2004); //定义Date类对象d1

t1.display(d1); //调用t1中的display函数,实参是Date类对象d1

return 0;

}

例4.1

编写程序:

#include<iostream>

using namespace std;class Complex//定义Complex类

{public:

Complex(){real=0;imag=0;} //定义构造函数

Complex(double r,double i){real=r;imag=i;} //构造函数重载

Complex complex\_add(Complex &c2); //声明复数相加函数

void display(); //声明输出函数

private:

double real; //实部

double imag; //虚部

};

Complex Complex::complex\_add(Complex &c2) //定义复数相加函数

{Complex c;

c.real=real+c2.real;

c.imag=imag+c2.imag;

return c;}

void Complex::display() //定义输出函数

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

int main()

{Complex c1(3,4),c2(5,-10),c3; //定义3个复数对象

c3=c1.complex\_add(c2); //调用复数相加函数

cout<<"c1="; c1.display(); //输出c1的值

cout<<"c2="; c2.display(); //输出c2的值

cout<<"c1+c2="; c3.display(); //输出c3的值

return 0;

　}

例4.2

编写程序:

#include<iostream>

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

Complex operator+(Complex &c2);//声明重载运算符"+"的函数

void display();

private:

double real;

double imag;

};

Complex Complex::operator+(Complex &c2) //定义重载运算符"+"的函数

{ Complex c;

c.real=real+c2.real; //实现两个复数的实部相加

　c.imag=imag+c2.imag; //实现两个复数的虚部相加

　return c;}

void Complex::display()

{ cout<<"("<<real<<","<<imag<<"i)"<<endl;} //输出复数形式

int main()

{ Complex c1(3,4),c2(5,-10),c3;

c3=c1+c2; //运算符"+"用于复数运算

cout<<"c1=";c1.display(); //输出c1

　cout<<"c2=";c2.display(); //输出c2

cout<<"c1+c2=";c3.display(); //输出c1+c2

　return 0;

}

例4.3

编写程序:

#include<iostream>

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

friend Complex operator+(Complex &c1,Complex &c2);//重载函数作为友元函数

void display();

private:

double real;

double imag;

};

Complex operator+(Complex &c1,Complex &c2) //定义运算符"+"重载函数

{return Complex(c1.real+c2.real,c1.imag+c2.imag);}

void Complex::display()

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

int main()

{Complex c1(3,4),c2(5,-10),c3;

c3=c1+c2;

cout<<"c1="; c1.display();

cout<<"c2="; c2.display();

cout<<"c1+c2="; c3.display();

}

例4.4

编写程序:

#include<iostream>

using namespace std;

class String

{public:

String(){p=NULL;}

String(char str);

friend bool operator>(String &string1,String &string2);

friend bool operator<(String &string1,String &string2);

friend bool operator==(String &string1,String &string2);

void display();

private:

char p;

};

String::String(char str)

{p=str;}

void String::display() //输出p所指向的字符串

{cout<<p;}

bool operator>(String &string1,String &string2)

{if(strcmp(string1.p,string2.p)>0)

return true;

else

return false;

}

bool operator<(String &string1,String &string2)

{if(strcmp(string1.p,string2.p)<0)

return true;

else

return false;

　}

bool operator==(String &string1,String &string2)

{if(strcmp(string1.p,string2.p)==0)

return true;

else

return false;

}

void compare(String &string1,String &string2)

　{if(operator>(string1,string2)==1)

{string1.display();cout<<">";string2.display();}

else

if(operator<(string1,string2)==1)

{string1.display();cout<<"<";string2.display();}

else

if(operator==(string1,string2)==1)

{string1.display();cout<<"=";string2.display();}

cout<<endl;

}

int main()

{String string1("Hello"),string2("Book"),string3("Computer"),

string4("Hello");

compare(string1,string2);

compare(string2,string3);

compare(string1,string4);

return 0;

}

例4. 5

编写程序:

#include<iostream>

using namespace std;

class Time

{public:

Time(){minute=0;sec=0;}//默认构造函数

Time(int m,int s):minute(m),sec(s){ } //构造函数重载

Time operator++(); //声明运算符重载成员函数

void display(){cout<<minute<<":"<<sec<<endl;} //定义输出时间函数

private:

int minute;

int sec;

};

Time Time::operator++() //定义运算符重载成员函数

{if(++sec>=60)

{sec-=60; //满60秒进1分钟

++minute;}

return this; //返回当前对象值

}

int main()

{Time time1(34,0);

for(int i=0;i<61;i++)

{++time1;

time1.display();}

return 0;

}

例4.6

编写程序:

#include<iostream>

using namespace std;

class Time

{public:

Time(){minute=0;sec=0;}

Time(int m,int s):minute(m),sec(s){}

Time operator++(); //声明前置自增运算符"++"重载函数

Time operator++(int); //声明后置自增运算符"++"重载函数

void display(){cout<<minute<<":"<<sec<<endl;}

private:

int minute;

int sec;

};

Time Time::operator++() //定义前置自增运算符"++"重载函数

{if(++sec>=60)

{sec-=60;

++minute;}

return this; //返回自加后的当前对象

}

Time Time::operator++(int) //定义后置自增运算符"++"重载函数

{Time temp(this); //建立临时对象temp

sec++;

if(sec>=60)

{sec-=60;

++minute;}

return temp; //返回的是自加前的对象

}

int main()

{Time time1(34,59),time2;

cout<<" time1 : ";

time1.display();

++time1;

cout<<"++time1: ";

time1.display();

time2=time1++; //将自加前的对象的值赋给time2

cout<<"time1++: ";

time1.display();

cout<<" time2: ";

time2.display(); //输出time2对象的值

}

例4.7

编写程序:

#include<iostream>在Visual C++ 6.0环境下运行时,需将第1行改为#include<iostream.h>,并删去第2行。

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

Complex operator+(Complex &c2); //运算符"+"重载为成员函数

friend ostream& operator <<(ostream&,Complex&); //运算符"<<"重载为友元函数

private:

double real;

double imag;

};

Complex Complex::operator+(Complex &c2) //定义运算符"+"重载函数

{return Complex(real+c2.real,imag+c2.imag);}

ostream & operator <<(ostream & output,Complex& c) //定义运算符"<<"重载函数

{output<<"("<<c.real<<"+"<<c.imag<<"i)"<<endl;

return output;

}

int main()

{Complex c1(2,4),c2(6,10),c3;

c3=c1+c2;

cout<<c3;

return 0;

}

例4.8

编写程序:

#include<iostream>

using namespace std;

class Complex

{public:

friend ostream& operator <<(ostream&,Complex&); //声明友元重载运算符"<<"函数

friend istream& operator >>(istream&,Complex&); //声明友元重载运算符">>"函数

private:

double real;

double imag;

};

ostream& operator <<(ostream& output,Complex& c) //定义重载运算符"<<"函数

{output<<"("<<c.real<<"+"<<c.imag<<"i)";

return output;

}

istream& operator >>(istream& input,Complex& c) //定义重载运算符">>"函数

{cout<<"input real part and imaginary part of complex number:";

input>>c.real>>c.imag;

return input;

}

int main()

{Complex c1,c2;

cin>>c1>>c2;

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

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

return 0;

}

例4.9

编写程序:

#include<iostream>

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

operator double() {return real;}//定义类型转换函数

private:

double real;

double imag;

};

int main()

{Complex c1(3,4),c2(5,-10),c3; //建立3个Complex类对象

double d;

d=2.5+c1; //要求将一个double数据与Complex类数据相加

cout<<d<<endl;

return 0;

}

例4.10

编写程序:

#include<iostream>

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}//默认构造函数,无形参

Complex(double r){real=r;imag=0;} //转换构造函数,一个形参

Complex(double r,double i){real=r;imag=i;} //实现初始化的构造函数,两个形参

friend Complex operator+(Complex c1,Complex c2); //重载运算符"+"的友元函数

void display();

private:

double real;

double imag;

};

Complex operator+(Complex c1,Complex c2) //定义运算符"+"重载函数

{return Complex(c1.real+c2.real,c1.imag+c2.imag);}

void Complex::display() //定义输出函数

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

int main()

{Complex c1(3,4),c2(5,-10),c3; //建立3个对象

c3=c1+2.5; //复数与double数据相加

c3.display();

return 0;

}

例5.1

编写程序:

先写类的声明部分。Class Student//声明基类

{public: //基类公用成员

void get\_value() //输入基类数据的成员函数

{cin>>num>>name>>sex;}

void display() //输出基类数据的成员函数

{cout<<" num: "<<num<<endl;

cout<<" name: "<<name<<endl;

cout<<" sex: "<<sex<<endl;}

private: //基类私有成员

int num;

string name;

char sex;

};

class Student1: public Student //以public方式声明派生类Student1

{public:

void get\_value\_1() //输入派生类数据

{cin>>age>>addr;}

void display\_1()

{ cout<<" num: "<<num<<endl; //企图引用基类的私有成员,错误

cout<<" name: "<<name<<endl; //企图引用基类的私有成员,错误

cout<<" sex: "<<sex<<endl; //企图引用基类的私有成员,错误

cout<<" age: "<<age<<endl; //引用派生类的私有成员,正确

cout<<" address: "<<addr<<endl;} //引用派生类的私有成员,正确

Private:

int age;

string addr;

};

void get\_value\_1()//输入5个数据的函数

{get\_value(); //调用基类的公用函数输入基类3个数据

cin>>age>>addr;} //输入派生类两个数据

void display\_1() //输出5个数据成员的值

{display(): //调用基类的公用成员函数,输出3个数据成员的值

　 cout<<"age: "<<age<<endl;//输出派生类的私有数据成员age

cout<<"address: "<<addr<<endl;} //输出派生类的私有数据成员addrmain函数可改写为int main()

{Student1 stud1;

　 stud1.get\_value\_1();//get\_value\_1是派生类Student1类的公用函数

　 stud1.display\_1(); //display\_1是派生类Student1类的公用函数

　 return 0;

}

例5.2

class Student1: private Student//用私有继承方式声明派生类Student1

{public:

void get\_value\_1() //输入派生类数据

{cin>>num>>name>>sex;}

void display\_1() //输出两个数据成员的值

{ cout<<"age: "<<age<<endl; //引用派生类的私有成员

cout<<"address: "<<addr<<endl;} //引用派生类的私有成员

private:

int age;

string addr;

　}; 　请分析下面的主函数: 　int main()

　{Student1 stud1;//定义一个Student1类的对象stud1

stud1.display(); //错误,私有基类的公用成员函数在派生类中是私有函数

stud1.display\_1(); //正确,display\_1函数是Stutent1类的公用函数

stud1.age=18; //错误,外界不能引用派生类的私有成员

return 0;

}

例5.3

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Student//声明基类

{public: //基类无公用成员

protected:

int num;

string name;

char sex;

};

class Student1: protected Student //用protected方式声明派生类Student1

{public:

void get\_value1(); //派生类公用成员函数

　void display1(); //派生类公用成员函数

private:

int age; //派生类私有数据成员

string addr; //派生类私有数据成员

};

void get\_value1() //定义派生类公用成员函数

{cin>>num>>name>>sex; //输入保护基类数据成员

cin>>age>>addr;} //输入派生类数据成员

void Student1::display1() //定义派生类公用成员函数

{cout<<"num: "<<num<<endl; //引用基类的保护成员

cout<<"name: "<<name<<endl; //引用基类的保护成员

　 cout<<"sex: "<<sex<<endl; //引用基类的保护成员

cout<<"age: "<<age<<endl; //引用派生类的私有成员

cout<<"address: "<<addr<<endl; //引用派生类的私有成员

}

int main()

{Student1 stud1;//stud1是派生类Student1类的对象

stud1.get\_value1(); //get\_value1是派生类中的公用成员函数,输入数据

stud1.display1(); //display1是派生类中的公用成员函数,输出数据

return 0;

}

例5.4

class A //基类

{public:

int i;

protected:

void f1();

int j;

private:

int k;

};

class B: public A //public派生类

{public:

void f2();

protected:

void f3();

private:

int m;

};

class C: protected B //protected派生类

{public:

void f4();

private:

int n;

};

例5.5

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Student //声明基类Student

{public:

Student(int n,string nam,char s) //定义基类构造函数

{num=n;

name=nam;

sex=s; }

~Student(){ } //基类析构函数

protected: //保护部分

int num;

string name;

char sex;

};

class Student1: public Student //声明公用派生类Student1

{public: //派生类的公用部分

Student1(int n,string nam,char s,int a,string ad):Student(n,nam,s)

//定义派生类构造函数

{age=a; //在函数体中只对派生类新增的数据成员初始化

addr=ad;

}

void show()

{cout<<"num: "<<num<<endl;

cout<<"name: "<<name<<endl;

cout<<"sex: "<<sex<<endl;

cout<<"age: "<<age<<endl;

cout<<"address: "<<addr<<endl<<endl;

}

~Student1(){} //派生类析构函数

private: //派生类的私有部分

int age;

string addr;

};

int main() //主函数

{Student1 stud1(10010,"Wang-li",'f',19,"115 Beijing Road,Shanghai");

Student1 stud2(10011,"Zhang-fan",'m',21,"213 Shanghai Road,Beijing");

stud1.show(); //输出第一个学生的数据

stud2.show(); //输出第二个学生的数据

return 0;

}

例5.6

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Student//声明基类

{public: //公用部分

Student(int n,string nam) //基类构造函数,与例5.5相同

{num=n;

name=nam;

}

void display() //成员函数,输出基类数据成员

{cout<<"num:"<<num<<endl<<"name:"<<name<<endl;}

protected: //保护部分

int num;

string name;

};

class Student1: public Student //声明公用派生类student1

{public:

Student1(int n,string nam,int n1,string nam1,int a,string ad)

:Student(n,nam),monitor(n1,nam1) //派生类构造函数

{age=a;

addr=ad;

}

void show()

{cout<<"This student is:"<<endl;

display(); //输出num和name

cout<<"age: "<<age<<endl; //输出age

cout<<"address: "<<addr<<endl<<endl; //输出addr

}

void show\_monitor() //成员函数,输出子对象

{cout<<endl<<"Class monitor is:"<<endl;

monitor.display(); //调用基类成员函数

}

private: //派生类的私有数据

Student monitor; //定义子对象(班长)

int age;

string addr;

};

int main()

{Student1 stud1(10010,"Wang\_li",10001,"Li\_jun",19,"115 Beijing Road,

Shanghai");

stud1.show(); //输出学生的数据

stud1.show\_monitor(); //输出子对象的数据

　return 0;

}

例5.7

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Student//声明基类

{public: //公用部分

Student(int n,string nam) //基类构造函数

{num=n;

name=nam;

}

void display() //输出基类数据成员

{cout<<"num:"<<num<<endl;

cout<<"name:"<<name<<endl;

}

protected: //保护部分

int num; //基类有两个数据成员

string name;

};

class Student1: public Student //声明公用派生类Student1

{public:

Student1(int n,char nam\[10\],int a):Student(n,nam) //派生类构造函数

{age=a;} //在此处只对派生类新增的数据成员初始化

void show() //输出num,name和age

{display(); //输出num和name

cout<<"age: "<<age<<endl;

}

private: //派生类的私有数据

int age; //增加一个数据成员

};

class Student2:public Student1 //声明间接公用派生类student2

{public:

//下面是间接派生类构造函数

Student2(int n,string nam,int a,int s):Student1(n,nam,a)

{score=s;}

void show\_all() //输出全部数据成员

{show(); //输出num和name

cout<<"score:"<<score<<endl; //输出age

}

private:

int score; //增加一个数据成员

};

int main()

{Student2 stud(10010,"Li",17,89);

stud.show\_all(); //输出学生的全部数据

　return 0;

}

例5.8

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Teacher//声明类Teacher(教师)类

{public: //公用部分

Teacher(string nam,int a,string t) //构造函数

{name=nam;

age=a;

title=t;}

void display() //输出教师有关数据

{cout<<"name:"<<name<<endl;

cout<<"age"<<age<<endl;

cout<<"title:"<<title<<endl;

}

protected: //保护部分

string name;

int age;

string title; //职称

};

class Student //定义类Student(学生)

{public:

Student(char nam\[\],char s,float sco) //构造函数

{strcpy(name1,nam);

sex=s;

score=sco;}

void display1() //定义输出学生有关数据的函数

{cout<<"name:"<<name1<<endl;

cout<<"sex:"<<sex<<endl;

　 cout<<"score:"<<score<<endl;

}

protected: //保护部分

string name1;

char sex;

float score; //成绩

};

class Graduate:public Teacher,public Student //声明多重继承的派生类Graduate

{public:

Graduate(string nam,int a,char s,string t,float sco,float w):

Teacher(nam,a,t),Student(nam,s,sco),wage(w) { }

void show() //输出研究生的有关数据

{cout<<"name:"<<name<<endl;

cout<<"age:"<<age<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"score:"<<score<<endl;

cout<<"title:"<<title<<endl;

cout<<"wages:"<<wage<<endl;

}

private:

float wage; //津贴

};

int main()

{Graduate grad1("Wang\_li",24,'f',"assistant",89.5,2400);

grad1.show();

return 0;

}

例5.9

编写程序:

#include<iostream>

#include<string>

using namespace std;

//下面声明公共基类Person

class Person

{public:

Person(string nam,char s,int a)//构造函数

{name=nam;sex=s;age=a;}

protected: //保护成员

string name;

char sex;

int age;

};

//下面声明Person的直接派生类Teacher

class Teacher:virtual public Person //声明Person为公用继承的虚基类

{public:

Teacher(string nam,char s,int a,string t):Person(nam,s,a) //构造函数

{title=t;

}

protected: //保护成员

string title; //职称

};

//下面声明Person的直接派生类Student

class Student:virtual public Person //声明Person为公用继承的虚基类

{public:

Student(string nam,char s,int a,float sco) //构造函数

:Person(nam,s,a),score(sco){} //初始化表

protected: //保护成员

float score; //成绩

};

　//下面声明多重继承的派生类Graduate

class Graduate:public Teacher,public Student //Teacher和Student为直接基类

{public:

Graduate(string nam,char s,int a,string t,float sco,float w) //构造函数

:Person(nam,s,a),Teacher(nam,s,a,t),Student(nam,s,a,sco),wage(w){} //初始化表

void show() //输出研究生的有关数据

{cout<<"name:"<<name<<endl;

cout<<"age:"<<age<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"score:"<<score<<endl;

cout<<"title:"<<title<<endl;

cout<<"wages:"<<wage<<endl;

}

private:

float wage; //津贴

};

//主函数

int main()

{Graduate grad1("Wang\_li",'f',24,"assistant",89.5,1200);

grad1.show();

return 0;

}

例5.10

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Student//声明Student类

{public:

Student(int,string,float); //声明构造函数

void display(); //声明输出函数

private:

int num;

string name;

float score;

};

Student::Student(int n,string nam,float s) //定义构造函数

{num=n;

name=nam;

score=s;

}

void Student::display() //定义输出函数

{cout<<endl<<"num:"<<num<<endl;

cout<<"name:"<<name<<endl;

cout<<"score:"<<score<<endl;

}

class Graduate:public Student //声明公用派生类Graduate

{public:

Graduate(int,string,float,float); //声明构造函数

void display(); //声明输出函数

private:

float wage; //津贴

};

Graduate::Graduate(int n,string nam,float s,float w):Student(n,nam,s),

wage(w){} //定义构造函数

void Graduate::display() //定义输出函数

{Student::display(); //调用Student类的display函数

cout<<"wage="<<wage<<endl;

}

int main()

{Student stud1(1001,"Li",87.5); //定义Student类对象stud1

Graduate grad1(2001,"Wang",98.5,1000); //定义Graduate类对象grad1

Student pt=&stud1; //定义指向Student类对象的指针并指向stud1

pt->display(); //调用stud1.display函数

pt=&grad1; //指针指向grad1

pt->display(); //调用grad1.display函数

}

例6.1

(1) 声明基类Point

可写出声明基类Point的部分: #include<iostream>

//声明类Point

class Point

{public:

Point(float x=0,float y=0);//有默认参数的构造函数

void setPoint(float,float); //设置坐标值的成员函数

float getX() const {return x;} //读x坐标,getX函数为常成员函数

float getY() const {return y;} //读y坐标,getY函数为常成员函数

friend ostream & operator<<(ostream &,const Point &); //友元重载运算符"<<"

protected: //受保护成员

　float x,y;

};

//下面定义Point类的成员函数

//定义Point类的构造函数

Point::Point(float a,float b) //对x,y初始化

　{x=a;y=b;}

//设置x和y的坐标值

void Point::setPoint(float a,float b) //对x,y赋以新值

{x=a;y=b;}

//重载运算符<<,使之能输出点的坐标

ostream & operator<<(ostream &output,const Point &p)

{output<<"\["<<p.x<<","<<p.y<<"\]"<<endl;

return output;

}

int main()

{Point p(3.5,6.4); //建立Point类对象p,对x,y初始化

cout<<"x="<<p.getX()<<",y="<<p.getY()<<endl; //输出p的坐标值x,y

p.setPoint(8.5,6.8); //重新设置p的坐标值

cout<<"p(new):"<<p<<endl; //用重载运算符<<输出p点坐标

}

(2) 声明派生类Circle

class Circle:public Point //circle是Point类的公用派生类

{public:

Circle(float x=0,float y=0,float r=0); //构造函数

void setRadius(float); //设置半径值的函数

float getRadius() const; //读取半径值的函数

float area() const; //计算圆面积的函数

friend ostream &operator<<(ostream &,const Circle &); //重载运算符"<<"

private:

float radius;

};

//定义Circle类的构造函数,对圆心坐标和半径初始化

Circle::Circle(float a,float b,float r):Point(a,b),radius(r){}

//定义设置半径值的函数

void Circle::setRadius(float r)

{radius=r;}

//定义读取半径值的函数

float Circle::getRadius() const {return radius;}

//定义计算圆面积的函数

float Circle::area() const

{return 3.14159radiusradius;}

//重载运算符<<,使之按规定的形式输出圆的信息

ostream &operator<<(ostream &output,const Circle &c)

{output<<"Center=\["<<c.x<<","<<c.y<<"\],r="<<c.radius<<",area="<<c.area()<<endl;

　 return output;

　}为了测试以上Circle类的定义,可以写出下面的主函数: int main()

{Circle c(3.5,6.4,5.2); //建立Circle类对象c并指定圆心坐标和半径

cout<<"original circle:\\nx="<<c.getX()<<",y="<<c.getY()<<",r="

<<c.getRadius()<<",area="<<c.area()<<endl; //输出圆心坐标、半径和面积

c.setRadius(7.5); //设置半径值

c.setPoint(5,5); //设置圆心坐标值x,y

cout<<"new circle:\\n"<<c; //用重载运算符<<输出圆对象的信息

Point &pRef=c; //pRef是Point类的引用,被c初始化

cout<<"pRef:"<<pRef; //输出pRef的信息

return 0;

}

(3) 声明Circle的派生类Cylinder

class Cylinder:public Circle//Cylinder是Circle的公用派生类

{public:

Cylinder(float x=0,float y=0,float r=0,float h=0); //构造函数

void setHeight(float); //设置圆柱高的函数

float getHeight() const; //读取圆柱高的函数

float area() const; //计算圆表面积的函数

float volume() const; //计算圆柱体积的函数

friend ostream& operator<<(ostream&,const Cylinder&); //重载运算符"<<"

protected:

float height; //圆柱高

};

//定义构造函数

Cylinder::Cylinder(float a,float b,float r,float h)

　:Circle(a,b,r),height(h){}

//定义设置圆柱高的函数

void Cylinder::setHeight(float h){height=h;}

//定义读取圆柱高的函数

float Cylinder::getHeight() const {return height;}

//定义计算圆表面积的函数

float Cylinder::area() const

{ return 2Circle::area()+23.14159radiusheight;}

//定义计算圆柱体积的函数

float Cylinder::volume() const

{return Circle::area()height;}

//重载运算符"<<"的函数

ostream &operator<<(ostream &output,const Cylinder& cy)

{output<<"Center=\["<<cy.x<<","<<cy.y<<"\],r="<<cy.radius<<",h="<<cy.height<<"\\narea="<<cy.area()<<",volume="<<cy.volume()<<endl;

return output;

　}

可以写出下面的主函数:

int main()

{Cylinder cy1(3.5,6.4,5.2,10); //定义Cylinder类对象cy1,并初始化

cout<<"original cylinder:\\nx="<<cy1.getX()<<",y="<<cy1.getY()<<",r="

<<cy1.getRadius()<<",h="<<cy1.getHeight()<<"\\narea="<<cy1.area()

<<",volume="<<cy1.volume()<<endl; //用系统定义的运算符"<<"输出圆柱

//cy1的数据

cy1.setHeight(15); //设置圆柱高

cy1.setRadius(7.5); //设置圆半径

cy1.setPoint(5,5); //设置圆心坐标值x,y

cout<<"\\nnew cylinder:\\n"<<cy1; //用重载运算符"<<"输出cy1的数据

Point &pRef=cy1; //pRef是Point类对象的引用

cout<<"\\npRef as a point:"<<pRef; //pRef作为一个"点"输出

Circle &cRef=cy1; //cRef是Circle类对象的引用

cout<<"\\ncRef as a Circle:"<<cRef; //cRef作为一个"圆"输出

return 0;

}

例6.2

#include<iostream>

#include<string>

using namespace std;

//声明基类Student

class Student

{public:

Student(int,string,float); //声明构造函数

void display(); //声明输出函数

protected: //受保护成员,派生类可以访问

int num;

string name;

float score;

};

　//Student类成员函数的实现

Student::Student(int n,string nam,float s) //定义构造函数

{num=n;name=nam;score=s;}

void Student::display() //定义输出函数

{cout<<"num:"<<num<<"\\nname:"<<name<<"\\nscore:"<<score<<"\\n\\n";}

//声明公用派生类Graduate

class Graduate:public Student

{public:

Graduate(int,string,float,float); //声明构造函数

void display(); //与基类的输出函数同名

private:

float wage;

};

//Graduate类成员函数的实现

Graduate::Graduate(int n,string nam,float s,float w):Student(n,nam,s),

wage(w){}

void Graduate::display() //定义输出函数

{cout<<"num:"<<num<<"\\nname:"<<name<<"\\nscore:"<<score<<"\\nwage="<<wage<<endl;}

//主函数

int main()

{Student stud1(1001,"Li",87.5); //定义Student类对象stud1

Graduate grad1(2001,"Wang",98.5,1200); //定义Graduate类对象grad1

Student pt=&stud1; //定义指向基类对象的指针变量pt,指向stud1

pt->display(); //输出Student(基类)对象stud1中的数据

pt=&grad1; //pt指向Graduate类对象grad1

pt->display(); //希望输出Graduate类对象grad1中的数据

return 0;

}

例6.3

编写程序：

#include<iostream>

using namespace std;

class Point //定义基类Point类

{public:

Point(){} //Point类构造函数

~Point(){cout<<"executing Point destructor"<<endl;} //Point类析构函数

};

class Circle:public Point //定义派生类Circle类

{public:

Circle(){} //Circle类构造函数

~Circle(){cout<<"executing Circle destructor"<<endl;} //Circle类析构函数

private:

int radus;

};

int main()

{ Point p=new Circle; //用new开辟动态存储空间

delete p; //用delete释放动态存储空间

return 0;

　}

例6.4

第(1)部分

#include<iostream>

using namespace std;

//声明抽象基类Shape

class Shape

{public:

virtual float area() const {return 0.0;}//虚函数

virtual float volume() const {return 0.0;} //虚函数

virtual void shapeName() const=0; //纯虚函数

};

第(2)部分//声明Point类

class Point:public Shape //Point是Shape的公用派生类

{public:

Point(float=0,float=0); //声明构造函数

void setPoint(float,float);

float getX() const {return x;} //设置点的x坐标

float getY() const {return y;} //设置点的y坐标

virtual void shapeName() const {cout<<"Point:";} //对虚函数进行再定义

friend ostream & operator<<(ostream &,const Point &);//运算符重载

protected:

float x,y;

};

//定义Point类成员函数

Point::Point(float a,float b) //定义构造函数

{x=a;y=b;}

void Point::setPoint(float a,float b)

{x=a;y=b;}

ostream & operator<<(ostream &output,const Point &p)

{output<<"\["<<p.x<<","<<p.y<<"\]";

return output;

}

第(3)部分//声明Circle类

class Circle:public Point

{public:

Circle(float x=0,float y=0,float r=0); //声明构造函数

void setRadius(float); //设定半径

float getRadius() const; //取半径的值

virtual float area() const; //对虚函数进行再定义

virtual void shapeName() const {cout<<"Circle:";} //对虚函数进行再定义

friend ostream &operator<<(ostream &,const Circle &); //运算符重载

protected:

float radius;

};

//定义Circle类成员函数

Circle::Circle(float a,float b,float r):Point(a,b),radius(r){} //定义构造函数

void Circle::setRadius(float r):radius(r){}

float Circle::getRadius() const {return radius;}

float Circle::area() const {return 3.14159radiusradius;}

ostream &operator<<(ostream &output,const Circle &c)

{output<<"\["<<c.x<<","<<c.y<<"\],r="<<c.radius;

　 return output;

　}

第(4)部分//声明Cylinder类

class Cylinder:public Circle

{public:

Cylinder(float x=0,float y=0,float r=0,float h=0); //声明构造函数

void setHeight(float); //设定圆柱高

virtual float area() const; //重载虚函数

virtual float volume() const; //重载虚函数

virtual void shapeName() const {cout<<"Cylinder:";} //重载虚函数

friend ostream& operator<<(ostream&,const Cylinder&); //运算符重载

protected:

float height;

};

//定义Cylinder类成员函数

Cylinder::Cylinder(float a,float b,float r,float h)

:Circle(a,b,r),height(h){} //定义构造函数

void Cylinder::setHeight(float h){height=h;} //设定圆柱高

float Cylinder::area() const //计算圆柱表面积

{return 2Circle::area()+23.14159radiusheight;}

float Cylinder::volume() const //计算圆柱体积

{return Circle::area()height;}

ostream &operator<<(ostream &output,const Cylinder& cy)

{output<<"\["<<cy.x<<","<<cy.y<<"\],r="<<cy.radius<<",h="

<<cy.height;

return output;

}

第(5)部分 //main函数

int main()

{Point point(3.2,4.5);//建立Point类对象point

Circle circle(2.4,1.2,5.6); //建立Circle类对象circle

Cylinder cylinder(3.5,6.4,5.2,10.5); //建立Cylinder类对象cylinder

point.shapeName(); //用对象名建立静态关联

cout<<point<<endl; //输出点的数据

circle.shapeName(); //静态关联

cout<<circle<<endl; //输出圆的数据

cylinder.shapeName(); //静态关联

cout<<cylinder<<endl<<endl; //输出圆柱的数据

Shape pt; //定义基类指针

pt=&point; //使指针指向Point类对象

pt->shapeName(); //用指针建立动态关联

cout<<"x="<<point.getX()<<",y="<<point.getY()<<"\\narea="<<pt->area()

<<"\\nvolume="<<pt->volume()<<"\\n\\n"; //输出点的数据

pt=&circle; //指针指向Circle类对象

pt->shapeName(); //动态关联

cout<<"x="<<circle.getX()<<",y="<<circle.getY()<<"\\narea="<<pt->area()

<<"\\nvolume="<<pt->volume()<<"\\n\\n"; //输出圆的数据

pt=&cylinder; //指针指向Cylinder类对象

pt->shapeName(); //动态关联

cout<<"x="<<cylinder.getX()<<",y="<<cylinder.getY()<<"\\narea="<<pt->area()

<<"\\nvolume="<<pt->volume()<<"\\n\\n"; //输出圆柱的数据

return 0;

}

例7.1

编写程序:

#include<iostream>

#include<cmath>

using namespace std;

int main()

{float a,b,c,disc;

cout<<"please input a,b,c:";

cin>>a>>b>>c;

if(a==0)

cerr<<"a is equal to zero,error!"<<endl; //将有关出错信息插入cerr流,在屏幕输出

else

if((disc=bb-4ac)<0)

cerr<<"disc=bb-4ac<0"<<endl; //将有关出错信息插入cerr流,在屏幕输出

else

{cout<<"x1="<<(-b+sqrt(disc))/(2a)<<endl;

cout<<"x2="<<(-b-sqrt(disc))/(2a)<<endl;

}

return 0;

}

例7.2

编写程序:

#include<iostream>

#include<iomanip>//不要忘记包含此头文件

using namespace std;

int main()

{int a;

　cout<<"input a:";

cin>>a;

cout<<"dec:"<<dec<<a<<endl; //以十进制形式输出整数

cout<<"hex:"<<hex<<a<<endl; //以十六进制形式输出整数a

cout<<"oct:"<<setbase(8)<<a<<endl; //以八进制形式输出整数a

char pt="China"; //pt指向字符串"China"

cout<<setw(10)<<pt<<endl; //指定域宽为10,输出字符串

cout<<setfill('')<<setw(10)<<pt<<endl; //指定域宽10,输出字符串,空白处以

//''填充

double pi=22.0/7.0; //计算pi值

cout<<setiosflags(ios∷scientific)<<setprecision(8);//按指数形式输出,8位小数

cout<<"pi="<<pi<<endl; //输出pi值

cout<<"pi="<<setprecision(4)<<pi<<endl; //改为4位小数

cout<<"pi="<<setiosflags(ios∷fixed)<<pi<<endl;//改为小数形式输出

return 0;

　}

例7.3

编写程序:

#include<iostream>

using namespace std;

int main()

{int a=21

cout.setf(ios::showbase);//显示基数符号(0x或0)

cout<<"dec:"<<a<<endl; //默认以十进制形式输出a

cout.unsetf(ios::dec); //终止十进制的格式设置

cout.setf(ios::hex); //设置以十六进制输出的状态

cout<<"hex:"<<a<<endl; //以十六进制形式输出a

cout.unsetf(ios::hex); //终止十六进制的格式设置

cout.setf(ios::oct); //设置以八进制输出的状态

cout<<"oct:"<<a<<endl; //以八进制形式输出a

char pt="China"; //pt指向字符串"China"

cout.width(10); //指定域宽为10

cout<<pt<<endl; //输出字符串

cout.width(10); //指定域宽为10

cout.fill(''); //指定空白处以''填充

cout<<pt<<endl; //输出字符串

double pi=22.0/7.0; //输出pi值

cout.setf(ios::scientific); //指定用科学记数法输出

cout<<"pi="; //输出"pi="

cout.width(14); //指定域宽为14

cout<<pi<<endl; //输出pi值

cout.unsetf(ios::scientific); //终止科学记数法状态

cout.setf(ios::fixed); //指定用定点形式输出

cout.width(12); //指定域宽为12

cout.setf(ios::showpos); //正数输出"+"号

cout.setf(ios::internal); //数符出现在左侧

cout.precision(6); //保留6位小数

cout<<pi<<endl; //输出pi,注意数符"+"的位置

return 0;

　}

例7.4

编写程序:

#include<iostream>

using namespace std;

int main()

{char p="BASIC";//字符指针指向'B'

for(int i=4;i>=0;i--)

cout.put((p+i)); //从最后一个字符开始输出

cout.put('\\n');

return 0;

}

例7.5

编写程序:

#include<iostream>

using namespace std;

int main()

{float grade;

cout<<"enter grade:";

while(cin>>grade) //如果能从cin流正常读取数据,cin的值为真,执行循环体

{if(grade>=85) cout<<grade<<"GOOD!"<<endl;

if(grade<60) cout<<grade<<"FAIL!"<<endl;

else cout<<grade<<" OK!"<<endl;

cout<<"enter grade:";

　}

cout<<"The end."<<endl;

return 0;

}

例7.6

#include<iostream>

int main()

{int c;

cout<<"enter a sentence:"<<endl;

while((c=cin.get())!=EOF)

cout.put(c);

return 0;

}

例7.7

编写程序:

#include<iostream>

using namespace std;

int main()

{char ch\[20\];

cout<<"enter a sentence:"<<endl;

cin>>ch;

cout<<"The string read with cin is:"<<ch<<endl;

cin.getline(ch,20,'/');//读19个字符或遇到'/'结束

cout<<"The second part is:"<<ch<<endl;

cin.getline(ch,20); //读19个字符或遇到'/n'结束

cout<<"The third part is:"<<ch<<endl;

return 0;

}

例7.8

编写程序:

#include<iostream>

using namespace std;

int main()

{char c;

while(!cin.eof()) //eof()为假表示未遇到文件结束符

if((c=cin.get())!=' ') //检查读入的字符是否空格字符

cout.put(c);

return 0;

}

例7.9

编写程序:

#include<iostream>

using namespace std;

int main()

{char c\[20\];

int ch;

cout<<"please enter a sentence:"<<endl;

cin.getline(c,15,'/');

cout<<"The first part is:"<<c<<endl;

ch=cin.peek(); //观看当前字符

cout<<"The next character(ASCII code) is:"<<ch<<endl;

cin.putback(c\[0\]); //将'I'插入指针所指处

cin.getline(c,15,'/');

cout<<"The second part is:"<<c<<endl;

return 0;

}

例7.10

编写程序：

#include<iostream>

using namespace std;

int main()

{char ch\[20\];

cin.get(ch,20,'/');

cout<<"The first part is:"<<ch<<endl;

cin.get(ch,20,'/');

cout<<"The second part is:"<<ch<<endl;

return 0;

}

(2)

#include<iostream>

using namespace std;

int main()

{char ch\[20\];

cin.get(ch,20,'/');

cout<<"The first part is:"<<ch<<endl;

cin.ignore(); //跳过输入流中一个字符

cin.get(ch,20,'/');

cout<<"The second part is:"<<ch<<endl;

return 0;

}

例7.11

编写程序:

#include<fstream>

using namespace std;

int main()

{int a\[10\];

ofstream outfile("f1.dat",ios::out);//定义输出文件流对象,打开磁盘文件"f1.dat"

if(!outfile) //如果打开失败,outfile返回0值

{cerr<<"open error!"<<endl;

exit(1);

}

cout<<"enter 10 integer numbers:"<<endl;

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

{cin>>a\[i\];

outfile<<a\[i\]<<" "; //向磁盘文件"f1.dat"输出数据

}

outfile.close(); //关闭磁盘文件"f1.dat"

return 0;

}

例7.12

编写程序:

#include<fstream>

int main()

{int a\[10\],max,i,order;

ifstream infile("f1.dat",ios::in|ios::nocreate);

//定义输入文件流对象,以输入方式打开磁盘文件f1.dat

if(!infile)

{cerr<<"open error!"<<endl;

exit(1);

}

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

{infile>>a\[i\]; //从磁盘文件读入10个整数,顺序存放在a数组中

cout<<a\[i\]<<" "; //在显示器上顺序显示10个数

}

cout<<endl;

max=a\[0\];

order=0;

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

if(a\[i\]>max)

{max=a\[i\]; //将当前最大值放在max中

order=i; //将当前最大值的元素序号放在order中

}

cout<<"max="<<max<<endl<<"order="<<order<<endl;

infile.close();

return 0;

}

例7.13

编写程序:

#include<fstream>

using namespace std;

//save\_to\_file函数从键盘读入一行字符并将其中的字母存入磁盘文件

void save\_to\_file()

{ofstream outfile("f2.dat");

//定义输出文件流对象outfile,以输出方式打开磁盘文件f2.dat

if(!outfile)

{cerr<<"open f2.dat error!"<<endl;

exit(1);

}

char c\[80\];

cin.getline(c,80 //从键盘读入一行字符

for(int i=0;c\[i\]!=0;i++) //对字符逐个处理,直到遇'/0'为止

if(c\[i\]>=65 && c\[i\]<=90||c\[i\]>=97 && c\[i\]<=122) //如果是字母字符

{outfile.put(c\[i\]); //将字母字符存入磁盘文件f2.dat

cout<<c\[i\];} //同时送显示器显示

cout<<endl;

outfile.close(); //关闭f2.dat

}

//从磁盘文件f2.dat读入字母字符,将其中的小写字母改为大写字母,再存入f3.dat

void get\_from\_file()

{char ch;

ifstream infile("f2.dat",ios::in|ios::nocreate);

//定义输入文件流outfile,以输入方式打开磁盘文件f2.dat

if(!infile)

{cerr<<"open f2.dat error!"<<endl;

exit(1);

}

ofstream outfile("f3.dat"); //定义输出文件流outfile,以输出方式打开磁盘文件f3.dat

if(!outfile)

{cerr<<"open f3.dat error!"<<endl;

exit(1);

}

while(infile.get(ch)) //当读取字符成功时执行下面的复合语句

{if(ch>=97 && ch<=122) //判断ch是否小写字母

ch=ch-32; //将小写字母变为大写字母

outfile.put(ch); //将该大写字母存入磁盘文件f3.dat

cout<<ch; //同时在显示器输出

}

cout<<endl;

infile.close(); //关闭磁盘文件f2.dat

outfile.close(); //关闭磁盘文件f2.dat

}

int main()

{save\_to\_file();

//调用save\_to\_file(),从键盘读入一行字符并将其中的字母存入磁盘文件f2.dat

get\_from\_file();

//调用get\_from\_file(),从f2.dat读入字母字符,改为大写字母,再存入f3.dat

return 0;

}

例7.14

编写程序:

#include<fstream>

using namespace std;

struct student

{char name\[20\];

int num;

int age;

char sex;

};

int main()

{student stud\[3\]={"Li",1001,18,'f',"Fan",1002,19,'m',"Wang",1004,17,'f'};

　 ofstream outfile("stud.dat",ios::binary);

if(!outfile)

{cerr<<"open error!"<<endl;

abort();//退出程序

}

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

outfile.write((char)&stud\[i\],sizeof(stud\[i\]));

outfile.close();

return 0;

}

例7.15

编写程序:

#include<fstream>

using namespace std;

struct student

{string name;

int num;

int age;

char sex;

};

int main()

{student stud\[3\];

int i;

ifstream infile("stud.dat",ios::binary);

if(!infile)

{cerr<<"open error!"<<endl;

abort();

}

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

infile.read((char)&stud\[i\],sizeof(stud\[i\]));

infile.close();

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

{cout<<"NO."<<i+1<<endl;

cout<<"name:"<<stud\[i\].name<<endl;

cout<<"num:"<<stud\[i\].num<<endl;;

cout<<"age:"<<stud\[i\].age<<endl;

cout<<"sex:"<<stud\[i\].sex<<endl<<endl;

}

return 0;

}

例7.16

编写程序:

#include<fstream>

using namespace std;

struct student

{int num;

char name\[20\];

float score;

};

int main()

{student stud\[5\]={1001,"Li",85,1002,"Fan",97.5,1004,"Wang",54,1006,

"Tan",76.5,1010,"Ling",96};

fstream iofile("stud.dat",ios::in|ios::out|ios::binary);

//用fstream类定义输入输出二进制文件流对象iofile

if(!iofile)

{cerr<<"open error!"<<endl;

abort();

}

for(int i=0;i<5;i++)//向磁盘文件输出5个学生的数据

iofile.write((char )&stud\[i\],sizeof(stud\[i\]));

student stud1\[5\]; //用来存放从磁盘文件读入的数据

for(int i=0;i<5;i=i+2)

{iofile.seekg(isizeof(stud\[i\]),ios::beg); //定位于第0,2,4学生数据开头

iofile.read((char )&stud1\[i/2\],sizeof(stud1\[0\]));

　//先后读入3个学生的数据,存放在stud1\[0\],stud\[1\]和stud\[2\]中

cout<<stud1\[i/2\].num<<" "<<stud1\[i/2\].name<<" "<<stud1\[i/2\]

.score<<endl;

//输出stud1\[0\],stud\[1\]和stud\[2\]各成员的值

}

cout<<endl;

stud\[2\].num=1012; //修改第3个学生(序号为2)的数据

strcpy(stud\[2\].name,"Wu");

stud\[2\].score=60;

iofile.seekp(2sizeof(stud\[0\]),ios::beg); //定位于第3个学生数据的开头

iofile.write((char )&stud\[2\],sizeof(stud\[2\])); //更新第3个学生数据

iofile.seekg(0,ios::beg); //重新定位于文件开头

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

{iofile.read((char )&stud\[i\],sizeof(stud\[i\])); //读入5个学生的数据

cout<<stud\[i\].num<<" "<<stud\[i\].name<<" "<<stud\[i\].score<<endl;

}

iofile.close();

　return 0;

}

例7.17

编写程序:

#include<strstream>

using namespace std;

struct student

{int num;

char name\[20\];

float score;

}

int main()

{student stud\[3\]={1001,"Li",78,1002,"Wang",89.5,1004,"Fan",90};

char c\[50\]; //用户定义的字符数组

ostrstream strout(c,30); //建立输出字符串流,与数组c建立关联,缓冲区长30

for(int i=0;i<3;i++) //向字符数组c写3个学生的数据

strout<<stud\[i\].num<<stud\[i\].name<<stud\[i\].score;

strout<<ends; //ends是C++的I/O操作符,插入一个'\\0'

cout<<"array c:"<<c<<endl; //显示字符数组c中的字符

}

例7.18

编写程序:

#include<strstream>

using namespace std;

int main()

{char c\[50\]="12 34 65 -23 -32 33 61 99 321 32";

int a\[10\],i,j,t;

cout<<"array c:"<<c<<endl;//显示字符数组中的字符串

istrstream strin(c,sizeof(c)); //建立输入串流对象strin并与字符数组c关联

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

strin>>a\[i\]; //从字符数组c读入10个整数赋给整型数组a

cout<<"array a:";

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

cout<<a\[i\]<<" "; //显示整型数组a各元素

cout<<endl;

for(i=0;i<9;i++) //用起泡法对数组a排序

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

if(a\[j\]>a\[j+1\])

{t=a\[j\];a\[j\]=a\[j+1\];a\[j+1\]=t;}

ostrstream strout(c,sizeof(c)); //建立输出串流对象strout并与字符数组c关联

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

strout<<a\[i\]<<" "; //将10个整数存放在字符数组c

strout<<ends; //加入'\\0'

cout<<"array c:"<<c<<endl; //显示字符数组c

return 0;

}

例8.1

先写出没有异常处理时的程序:

#include<iostream>

#include<cmath>

using namespace std;

int main()

{double triangle(double,double,double);//函数声明

double a,b,c;

cin>>a>>b>>c; //输入3个边

while(a>0 && b>0 && c>0)

{cout<<triangle(a,b,c)<<endl; //输出三角形的面积

cin>>a>>b>>c; //输入3个边

}

return 0;

}

double triangle(double a,double b,double c)

{double area;

double s=(a+b+c)/2;

area=sqrt(s(s-a)(s-b)(s-c));

return area;

}运行结果: 6 5 4↙(输入a,b,c的值)

9.92157 (输出三角形的面积)

1 1.5 2↙ (输入a,b,c的值)

0.726184 (输出三角形的面积)

1 2 1↙ (输入a,b,c的值)

0 (输出三角形的面积,此结果显然不对,因为不是三角形)

1 0 6↙ (输入a,b,c的值)

(输入数据不合理,程序结束)

修改后的程序如下:

#include<iostream>

#include<cmath>

int main()

{double triangle(double,double,double);

double a,b,c;

cin>>a>>b>>c;

try//在try块中包括要检查的函数

{while(a>0 && b>0 && c>0)

{cout<<triangle(a,b,c)<<endl;

cin>>a>>b>>c;

}

}

catch(double) //用catch捕捉异常信息并作相应处理

{cout<<"a="<<a<<",b="<<b<<",c="<<c<<",that is not a traingle!"<<endl;

}

　cout<<"end"<<endl; //最后输出"end"

return 0; //返回0,程序正常结束

}

double triangle(double a,double b,double c)//定义计算三角形的面积的函数

{double s=(a+b+c)/2;

if(a+b<=c||b+c<=a||c+a<=b) throw a; //当不符合三角形条件抛出异常信息a

return sqrt(s(s-a)(s-b)(s-c));

}

例8.2

编写程序:

#include<iostream>

using namespace std;

int main()

{void f1();

try

{f1();}//调用f1()

catch(double)

{cout<<"ERROR0!"<<endl;}

cout<<"end0"<<endl;

return 0;

}

void f1() //定义f1函数

{void f2();

try

{f2();} //调用f2()

catch(char)

{cout<<"ERROR1!";}

cout<<"end1"<<endl;

}

void f2() //定义f2函数

{void f3();

try

{f3();} //调用f3()

catch(int)

{cout<<" ERROR2!"<<endl;}

cout<<"end2"<<endl;

}

void f3() //定义f3函数

{double a=0;

try

{throw a;} //抛出double类型异常

catch(float)

{cout<<" ERROR3!"<<endl;}

cout<<"end3"<<endl;

}

例8.3

编写程序:

#include<iostream>

#include<string>

using namespace std;

class Student

{public:

Student(int n,string nam)//定义构造函数

{cout<<"constructor-"<<n<<endl;

num=n;name=nam;}

~Student(){cout<<"destructor-"<<num<<endl;}//定义析构函数

void get\_data(); //成员函数声明

private:

int num;

string name;

};

void Student::get\_data() //定义成员函数

{if(num==0) throw num; //如num=0,抛出int型变量num

else cout<<num<<" "<<name<<endl; //若num≠0,输出num,name

cout<<"in get\_data()"<<endl; //表示目前在get\_data函数中

}

void fun()

{Student stud1(1101,"Tan"); //建立对象stud1

stud1.get\_data(); //调用stud1的get\_data函数

Student stud2(0,"Li"); //建立对象stud2

stud2.get\_data(); //调用stud2的get\_data函数

}

int main()

{cout<<"main begin"<<endl; //表示主函数开始了

cout<<"call fun()"<<endl; //表示调用fun函数

try

{fun();} //调用fun函数

catch(int n)

{cout<<"num="<<n<<",error!"<<endl;} //表示num=0出错

cout<<"main end"<<endl; //表示主函数结束

return 0;

}

例8.4

编写程序:

1)

#include<string>

#include<cmath>

class Student//声明Student类

{public:

Student(int n,string nam,int a)

{num=n;name=nam;age=a;}

void get\_data();

private:

int num;

string name;

int age;

};

void Student::get\_data() //成员函数定义

{cout<<num<<" "<<name<<" "<<age<<endl;

}

double fun(double a,double b) //定义全局函数(即外部函数)

{return sqrt(a+b);}在main函数所在的主文件中包含头文件header1.h: #include<iostream>

#include "header1.h" //注意要用双引号,因为文件一般是放在用户目录中的

int main()

{Student stud1(101,"Wang",18); //定义类对象stud1

stud1.get\_data();

cout<<fun(5,3)<<endl;

return 0;

}

2)

#include<string>

#include<cmath>

class Student //声明Student类

{public:

Student(int n,string nam,char s) //参数与header1中的student不同

{num=n;name=nam;sex=s;}

void get\_data();

private:

int num;

string name;

char sex; //此项与header1不同

};

void Student::get\_data() //成员函数定义

{cout<<num<<" "<<name<<" "<<sex<<endl;

}

double fun(double a,double b) //定义全局函数

{return sqrt(a-b);} //返回值与header1中的fun函数不同

主文件如下:

//main file

#include<iostream>

#include "header1.h" //包含头文件header1.h

#include "header2.h" //包含头文件header2.h

int main()

{Student stud1(101,"Wang",18);

stud1.get\_data();

cout<<fun(5,3)<<endl;

return 0;

}

例8.5

//header1.h (头文件1)

#include<string>

#include<cmath>

using namespace std;

namespace ns1//声明命名空间ns1

{class Student //在命名空间ns1内声明Student类

{public:

Student(int n,string nam,int a)

{num=n;name=nam;age=a;}

void get\_data();

private:

int num;

string name;

int age;

};

void Student::get\_data() //定义成员函数

{cout<<num<<" "<<name<<" "<<age<<endl;

}

double fun(double a,double b) //在命名空间ns1内定义fun函数

{return sqrt(a+b);}

　 }//命名空间ns1结束

//header2.h(头文件2)

#include<string>

#include<cmath>

namespace ns2 //声明命名空间ns2

{class Student //在命名空间ns2内声明Student类

{public:

Student(int n,string nam,char s)

{num=n;name=nam;sex=s;}

void get\_data();

private:

int num;

char name\[20\];

char sex;

};

void Student::get\_data()

{cout<<num<<" "<<name<<" "<<sex<<endl;

}

double fun(double a,double b)

{return sqrt(a-b);}

}//命名空间ns2结束

//main file(主文件)

#include<iostream>

#include "header1.h" //包含头文件1

#include "header2.h" //包含头文件2

int main()

{ns1::Student stud1(101,"Wang",18); //用命名空间ns1中的Student类定义stud1

stud1.get\_data(); //不要写成ns1::stud1.get\_data();

cout<<ns1::fun(5,3)<<endl; //调用命名空间ns1中的fun函数

ns2::Student stud2(102,"Li",'f'); //用命名空间ns2中的Student类定义stud2

stud2.get\_data();

cout<<ns2::fun(5,3)<<endl; //调用命名空间ns1中的fun函数

return 0;

}