实验四 静态成员和公用数据的保护

1. 实验目的

掌握类中静态成员的定义方法，掌握静态数据成员的初始化方法、静态数据成员和静态成员函数的访问和使用方法。掌握类中常对象、常对象成员、指向对象的常指针、指向常对象的指针变量及常引用的定义和使用，理解对象的动态建立和释放。

1. 实验内容
2. 编写一个歌手类，完成统计歌手大赛参赛人数，计算每名选手的平均得分功能。要求设计一个成员函数输入选手的编号、姓名、3名评委的打分；设计一个成员函数计算选手的平均得分；设计一个成员函数输出参赛总人数，并按平均分由高到低的顺序输出每位选手的名次、姓名、平均得分。在主函数中进行测试验证。

提示：将参赛人数声明为静态数据成员，定义一个歌手对象数组方便排序。

# include <string>

using namespace std;

class Singer

{

public:

Singer(int id,string n,int s1,int s2,int s3);

void average();

void display();

static int getsum();

public:

int ID;

string name;

int score1;

int score2;

int score3;

float aver;

static int sum;

public:

Singer();

virtual ~Singer();

};

#include <iostream>

#include "Singer.h"

#include <string>

using namespace std;

Singer::~Singer()

{

}

Singer::Singer()

{

}

Singer::Singer(int id,string n,int s1,int s2,int s3)

{

ID=id;

name=n;

score1=s1;

score2=s2;

score3=s3;

sum++;

}

void Singer::average()

{

int i,g;

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

{

g=score1+score2+score3;

aver=g/3;

}

}

void Singer::display()

{

cout<<ID<<" "<<name<<" "<<aver<<endl;

}

int Singer::getsum()

{

return sum;

}

int Singer::sum=0;

# include <iostream>

# include <string>

# include "Singer.h"

using namespace std;

int change(Singer a[3])

{

Singer temp;

int i,j;

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

{

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

{

if(a[j].aver<a[j+1].aver)

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

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

a[i].display();

return 0;

}

int main()

{

Singer s[3]={Singer(1,"a",9,8,8),

Singer(2,"b",7,8,8),

Singer(3,"c",9,8,7)};

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

s[i].average();

change(s);

cout<<"参赛人数:"<<Singer::getsum()<<endl;

return 0;

}

1. 设计一个TSample类,它包含一个公有数据成员n，建立该类的4个对象，即s1(n=90), s2(n=20), s3(n=50), s4(n=40)，设计一个成员函数nadd(TSample &s) 实现这些对象n值的累加，要求对象不能与自己相加，形参使用对象的引用；设计一个成员函数输出最后的累加结果，并在主函数中测试验证。

# include <iostream>

using namespace std;

class Sample

{

public:

Sample(int nu);

void add(Sample &s);

void display();

public:

int n;

};

Sample::Sample(int nu)

{

n=nu;

}

void Sample::add(Sample &s)

{

n=n+s.n;

}

void Sample::display()

{

cout<<"n="<<n<<endl;

}

int main()

{

Sample s1(10),s2(20),s3(30),s4(40);

s1.add(s2);

s1.add(s3);

s1.add(s4);

s1.display();

return 0;

}

1. 建立一个对象数组，内放5个学生的数据（学号，成绩），设计一个函数max()，用指向常对象的指针做函数参数，在max() 函数中找出5个学生中成绩最高者，并输出其学号。在主函数中进行测试验证。

# include <iostream>

using namespace std;

class Student

{

public:

Student(int n,int s);

int num;

int score;

};

Student::Student(int n,int s)

{

num=n;

score=s;

}

void max(const Student \*a)

{

int max;

int i,k=0;

max=a[0].score;

for(i=1;i<5;i++)

{

if(a[i].score>max)

{

max=a[i].score;

k=i;

}

}

cout<<"成绩最高:"<<a[k].num<<" "<<a[k].score<<endl;

}

int main()

{

const Student s[5]={Student(1,89),

Student(2,77),

Student(3,95),

Student(4,88),

Student(5,93)};

void max(const Student \*);

const Student \*p=&s[0];

max(s);

return 0;

}

# include <iostream>

using namespace std;

class Student

{

public:

Student(int n,int s);

int num;

int score;

};

Student::Student(int n,int s)

{

num=n;

score=s;

}

void max(const Student \*a)

{

int max;

int i,k=0;

max=a[0].score;

for(i=1;i<5;i++)

{

if(a[i].score>max)

{

max=a[i].score;

k=i;

}

}

cout<<"成绩最高:"<<a[k].num<<" "<<a[k].score<<endl;

}

int main()

{

Student s[5]={Student(1,89),

Student(2,77),

Student(3,95),

Student(4,88),

Student(5,93)};

void max(const Student \*);

Student \*p=&s[0];

max(s);

return 0;

}

实验小结