**学号：202311081015 姓名：杨博文 2024.5.31**

1. **填空题（每空2分，共34分）**

1. 面向对象的程序设计有四大特征：抽象、封装、(继承)、(多态)。

2. 在c++中定义重载函数时，至少使重载函数的(参数类型)、(参数个数)或参数顺序不同。

3. 在基类和派生类中，成员函数的覆盖是指派生类成员函数与基类中被覆盖的成员函数的名字、参数个数、(参数类型) 和(参数顺序返回值类型)均相同。

4. 当一个非静态成员函数被调用时，该成员函数的(this指针)指向调用它的对象。

5. 一个类的(static（静态）)成员被该类的所有对象所共享。

6. 一个类的（成员、（friend）友元）函数可以访问该类的私有数据成员。

7. 在保护继承中，基类的（private）成员在派生类中是不可见的。

8. 在c++中，有一种不能定义对象的类，这样的类只能被继承，这种类被称为（抽象类）， 该类的定义中至少具有一个（纯虚）函数。

9. 建立一个有成员对象的派生类对象时，各构造函数体的执行次序是（先执行基类构造函数，再执行成员对象的构造函数，最后执行派生类的构造函数体本身）。

10. 一个函数功能不太复杂，但要求被频繁调用，则应把它定义为（inline（内联））函数。

11. 设已经正确定义类Person，那么语句Person a = new Person("noah"); 错误的原因是（使用new应当创建一个指针而不是对象，应当更改为Person\* a=new Person(“noah”)）。

12. C++提供两种模板机制：（函数）模板和（类）模板。

1. **程序综合题(每空2分，第1、6、9空4分，共28分)**

1. 设计三角形类Triangle

已知三角形三条边长a,b,c, 其面积为sqrt(s\*(s-a)\*(s-b)\*(s-c)), 其中s=(a+b+c)/2 .

#include <iostream>

#include <cmath>

using namespace std;

class Triangle

{

private:

double \_line1, \_line2, \_line3;

public:

Triangle(double line1=1, double line2=1, double line3=1)

{

if((line1+line2>line3) && (line1+line3>line2) && (line2+line3>line1))

{ \_line1=line1;

\_line2=line2;

\_line3=line3;

}

else

exit(1);

}

/\*\*\*(1)\*\*\*/

Double Area()

{

Double S=\_line1+\_line2+\_line3;

s=s/2;

Return sqrt(s\*(s-\_line1)\*(s-\_line2)\*(s-\_line3));

}

/\*\*\*(2)\*\*\*/

Double Perimeter()

{

Return \_line1+\_line2+\_line3;

}

};

int main()

{

Triangle t(3,4,5);

cout<<"Area: "<<t.Area()<<endl; //输出面积： Area：6

cout<<"Perimeter: "<<t.Perimeter()<<endl; //输出周长： Perimeter：12

return 0;

}

2. static

#include <iostream>

using namespace std;

class Student

{

private:

string \_name;

int \_id;

/\*\*\*(3)\*\*\*/

Static int \_count;

public:

Student(string name="XXX"):\_name(name)

{ \_count++;

\_id=\_count;

}

~Student()

{ \_count--;

}

void Show() const

{ cout<<\_name<<" "<<\_id<<endl;

}

};

/\*\*\*(4)\*\*\*/

Int Student::\_count=0;

int main()

{

Student s1("Lili"), s2("LiuMin"),s[3];

s1.Show(); //输出： Lili 1

s2.Show(); //输出： LiuMin 2

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

s[i].Show(); //输出： XXX 3

// XXX 4

// XXX 5

return 0;

}

3. template

#include <iostream>

using namespace std;

/\*\*\*(5)\*\*\*/

Template <class T>

void Sort(T a[],int n)

{ /\*\*\*(6)\*\*\*/

For(int i=n-1;i>=1;i--)

{

For(int j=0;j<i;j++)

{

If(a[j]>a[j+1])

{

T temp=a[j];

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

A[j+1]=temp;

}

}

}

}

/\*\*\*(7)\*\*\*/

Template <class T>

void Output(T a[],int n)

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

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

cout<<endl;

}

int main()

{ int A[5]={9,8,20,2,5};

double B[6]={12.3,0,-63.5,21,-86,0.03};

string C[4]={"hello","today","Sun","month"};

Sort(A,5);

Output(A,5); //输出： 2 5 8 9 20

Sort(B,6);

Output(B,6); //输出： -86 -63.5 0 0.03 12.3 21

Sort(C,4);

Output(C,4); //输出： Sun hello month today

return 0;

}

4. STL

#include<iostream>

#include<string>

#include<map>

using namespace std;

int main()

{

string name[] = {"Zhang", "Liu", "Li" };

double salary[] = {1200, 2000, 1450 };

map<string, double> sal;

map<string, double>::iterator p;

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

sal.insert(make\_pair(name[i], salary[i]));

/\*\*\*\*(8)\*\*\*/

Sal.insert(make\_pair(“Bob”,2000));

Sal.insert(make\_pair(“Tom”,3000));

for (p = sal.begin(); p != sal.end(); p++)

cout << p->first << "\t" << p->second << endl;

return 0;

}

程序的输出是：

Bob 2000

Li 1450

Liu 2000

Tom 3000

Zhang 1200

5. 拷贝构造函数

#include<iostream>

using namespace std;

class MyArray

{

private:

int\* \_data;

int \_sz;

public:

MyArray(int\* a, int sz)

{ if(sz<=0)

exit(1);

\_sz=sz;

\_data=new int[\_sz];

for(int i=0;i<\_sz; i++)

\_data[i]=a[i];

}

MyArray(const MyArray& arr)

{

/\*\*\*(9)\*\*\*/

\_sz=arr.\_sz;

\_data=new int[\_sz];

For(int i=0;i<\_sz;i++)

{

\_data[i]=arr.\_data[i];

}

}

~MyArray()

{ if(\_data)

{ delete[] \_data;

cout<<"delete...\n";

}

}

void Show() const

{ for(int i=0;i<\_sz;i++)

cout<<\_data[i]<<" ";

cout<<endl;

}

};

int main()

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

MyArray A(a,10);

MyArray B(A);

B.Show(); //输出： 1 2 3 4 5 6 7 8 9 10

return 0; //输出： delete...

} // delete...

6. 友元

#include <iostream>

using namespace std;

class StuScore

{

private:

string \_name;

float \_phy,\_che, \_bio; //物理、化学、生物成绩

public:

StuScore(string name, float ph=0, float ch=0, float bi=0): \_name(name), \_phy(ph), \_che(ch), \_bio(bi)

{}

/\*\*\*(10)\*\*\*/

Friend float Total(StuScore s);

void Show() const

{ cout<<\_name<< " "<<\_phy<< " "<<\_che<< " "<<\_bio;

}

};

float Total(StuScore s)

{ /\*\*\*(11)\*\*\*/

Return s.\_phy+s.\_che+s.\_bio;

}

int main()

{

StuScore stu("WangYan", 90.5, 88, 92.5);

stu.Show();

cout<<"\tTotal:"<< Total(stu)<<endl; //输出： WangYan 90.5 88 92.5 Total:271

return 0;

}

**三、程序设计题(第1小题16分，第2小题22分，共38分)**

1. 设计并实现父亲Father-儿子Son类。

父亲擅长弹琴，儿子擅长弹琴和唱歌。

父亲Father类包含姓名、年龄，能力函数（虚函数）、输出函数（虚函数）;

儿子Son类公有继承父亲类，重写能力函数（虚函数）、输出函数（虚函数）;

下面给出了general(Father& f) 函数 和main()函数，以及程序的输出。

请补充儿子类的设计和实现。

#include <iostream>

using namespace std;

class Father

{

protected:

string \_name;

int \_age;

public:

Father(string name="XXX",int age=0);

virtual void Ability();

virtual void Show();

virtual ~Father(){}

};

Father::Father(string name, int age):\_name(name),\_age(age)

{}

void Father::Ability()

{ cout<<"He plays the piano well."<<endl;

}

void Father::Show()

{ cout<<"Father: "<<\_name<<" "<<\_age<<endl;

Ability();

cout<<endl;

}

void general(Father& f)

{ f.Show();

}

int main()

{

Father Tom("Tom",60);

Son Jack("Jack",30);

general(Tom);

general(Jack);

return 0;

}

程序输出：

Father: Tom 60

He plays the piano well.

Son: Jack 30

He plays the piano well.

He sings very well.

class Son:public Father

{

public:

Son(string name="XXX",int age=0);

void Ability();

void Show();

~Son(){}

};

Son::Son(string name,int age):Father(name,age){}

void Son::Ability()

{

cout<<"He plays the piano well.\nHe sings very well."<<endl;

}

void Son::Show()

{

cout<<"Son: "<<\_name<<" "<<\_age<<endl;

Ability();

cout<<endl;

}

2. 设计并实现简单矩阵Matrix类的+ ,-, \*, <<, >>运算。(这里简单矩阵的行列值都不大于100 .)

给出了Matrix的类声明和部分实现，给出了mian()函数，以及一次运行的结果。

请补充5个运算符重载的实现。

#include <iostream>

using namespace std;

const int N=100;

class Matrix

{

private:

int \_row,\_line; //行数、列数

int \_data[N][N];

public:

Matrix(int row=0,int line=0);

int getRow() const;

int getLine() const;

Matrix operator+(const Matrix& m) const; //#4分

Matrix operator-(const Matrix& m) const; //#4分

Matrix operator\*(const Matrix& m) const; //#6分

friend ostream& operator<<(ostream& out,const Matrix& m); //#4分

friend istream& operator>>(istream& in,Matrix& m); //#4分

};

Matrix::Matrix(int row,int line):\_row(row),\_line(line)

{}

int Matrix::getRow() const

{ return \_row;

}

int Matrix::getLine() const

{ return \_line;

}

int main()

{

Matrix A,B,C;

char con('Y');

do

{ cout<<"Input Matrix A: "<<endl;

cin>>A;

cout<<"Input Matrix B: "<<endl;

cin >>B;

cout<<"Test C=A+B :"<<endl;

C=A+B;

cout<<A<<" + \n"<<B<<" = \n"<<C<<endl;

cout<<"Test C=A-B :"<<endl;

C=A-B;

cout<<A<<" - \n"<<B<<" = \n"<<C<<endl;

cout<<"Test C=A\*B :"<<endl;

C=A\*B;

cout<<A<<" \* \n"<<B<<" = \n"<<C<<endl;

cout<<"Continue?(Y/N): ";

cin>>con;

}while(con=='Y'||con=='y');

}

以下是程序的一次输出，下划线表示用户的输入：

Input Matrix A:

Row(int, >=2, <=100) : 3

Line(int, >=2, <=100): 3

1 2 3

2 3 4

3 4 5

Input Matrix B:

Row(int, >=2, <=100) : 3

Line(int, >=2, <=100): 3

9 8 7

6 5 4

3 2 1

Test C=A+B :

1 2 3

2 3 4

3 4 5

+

9 8 7

6 5 4

3 2 1

=

10 10 10

8 8 8

6 6 6

Test C=A-B :

1 2 3

2 3 4

3 4 5

-

9 8 7

6 5 4

3 2 1

=

-8 -6 -4

-4 -2 0

0 2 4

Test C=A\*B :

1 2 3

2 3 4

3 4 5

\*

9 8 7

6 5 4

3 2 1

=

30 24 18

48 39 30

66 54 42

Continue?(Y/N): y

Input Matrix A:

Row(int, >=2, <=100) : 2

Line(int, >=2, <=100): 3

1 2 3

2 3 4

Input Matrix B:

Row(int, >=2, <=100): 3

Line(int, >=2, <=100): 4

9 8 7 6

6 5 4 3

3 2 1 0

Test C=A+B :

Can not Matrix(2,3) + Matrix(3,4) !

1 2 3

2 3 4

+

9 8 7 6

6 5 4 3

3 2 1 0

=

No data

Test C=A-B :

Can not Matrix(2,3) - Matrix(3,4) !

1 2 3

2 3 4

-

9 8 7 6

6 5 4 3

3 2 1 0

=

No data

Test C=A\*B :

1 2 3

2 3 4

\*

9 8 7 6

6 5 4 3

3 2 1 0

=

30 24 18 12

48 39 30 21

Continue?(Y/N): n

Byebye.

Matrix Matrix::operator+(const Matrix &m) const

{

if(\_row!=m.\_row || \_line!=m.\_line)

{

cout<<"Can not Matrix("<<\_row<<","<<\_line<<") + Matrix("<<m.\_row<<","<<m.\_line<<")!"<<endl;

return Matrix();

}

Matrix answer(\_row,\_line);

for(int i=0;i<\_row;i++)

{

for(int j=0;j<\_line;j++)

{

answer.\_data[i][j]=\_data[i][j]+m.\_data[i][j];

}

}

return answer;

}

Matrix Matrix::operator-(const Matrix &m) const

{

if(\_row!=m.\_row || \_line!=m.\_line)

{

cout<<"Can not Matrix("<<\_row<<","<<\_line<<") - Matrix("<<m.\_row<<","<<m.\_line<<")!"<<endl;

return Matrix();

}

Matrix answer(\_row,\_line);

for(int i=0;i<\_row;i++)

{

for(int j=0;j<\_line;j++)

{

answer.\_data[i][j]=\_data[i][j]-m.\_data[i][j];

}

}

return answer;

}

Matrix Matrix::operator\*(const Matrix &m) const

{

if(\_line!=m.\_row)

{

cout<<"Can not Matrix("<<\_row<<","<<\_line<<") \* Matrix("<<m.\_row<<","<<m.\_line<<")!"<<endl;

return Matrix();

}

Matrix answer(\_row,m.\_line);

for(int i=0;i<\_row;i++)

{

for(int j=0;j<m.\_line;j++)

{

answer.\_data[i][j]=0;

for(int k=0;k<\_line;k++)

{

answer.\_data[i][j]=answer.\_data[i][j]+\_data[i][k]\*m.\_data[k][j];

}

}

}

return answer;

}

ostream& operator<<(ostream& out,const Matrix& m)

{

If(m.\_row==0 || m.\_line==0)

{ return out<<”No data\n”;

}

for(int i=0;i<m.\_row;i++)

{

for(int j=0;j<m.\_line;j++)

{

out<<m.\_data[i][j]<<" ";

}

out << endl;

}

return out;

}

istream& operator>>(istream& in,Matrix& m)

{

cout <<"Row(int, >=2, <=100) :";

in>>m.\_row;

cout <<"Line(int, >=2, <=100):";

in>>m.\_line;

for(int i=0;i<m.\_row;i++)

{

for(int j=0;j<m.\_line;j++)

{

in>>m.\_data[i][j];

}

}

return in;

}