浙江大学2011–2012学年春夏学期

《面向对象程序设计》课程期末考试试卷

课程号： \_\_\_\_\_\_\_\_\_\_，开课学院：\_\_\_\_\_\_\_\_\_\_\_

考试试卷：A卷√、B卷（请在选定项上打√）

考试形式：闭√、开卷（请在选定项上打√），允许带\_\_\_\_\_\_\_\_\_\_\_入场

考试日期： 2012 年 06 月 21 日,考试时间： 120 分钟

诚信考试，沉着应考，杜绝违纪。

考生姓名： 学号： 所属院系： \_

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 题序 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总 分 |
| 得分 |  |  |  |  |  |  |  |  |  |
| 评卷人 |  |  |  |  |  |  |  |  |  |

1. Write the output of the code below（36%）

每题6分，对于主要部分正确的可酌情给2-3分。

1）

int &f(int &i)

{

i+=10;

return i;

}

void main()

{

int k=0;

int &m=f(k);

cout<<k<<endl;

m=20;

cout<<k<<endl;

}

[答案]

10

20

2)

int c;

class A{

private:

int a;

staticint b;

public:

A() {a=0;c=0;}

void seta() {a++;}

voidsetb() {b++;}

voidsetc() {c++;}

void display()

{

cout<< a << "\t" << b << "\t" << c;

}

};

int A::b=0;

void main()

{

A a1,a2;

a1.seta();

a1.setb();

a1.setc();

a1.display();

a2.seta();

a2.setb();

a2.setc();

a2.display();

}

[答案]

1 1 1

1 2 2

3)

class A

{

int m;

int n;

public:

A(){cout << "A()"<<endl;}

~A(){cout << "~A()"<<endl;}

A(const A &f){ cout << "A(const A&)" << endl;}

A &operator=(const A &f){ cout << "operator=(const A&)" <<endl;

return \*this;}

A &operator\*(){cout << "operator\*" << endl;return \*this;}

};

void main()

{

A \*ap;

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

ap = new A;

A a2 = \*ap;

delete ap;

A a3;

a3 = a2;

}

[答案]

===

A()

A(const A&)

~A()

A()

operator=(const A&)

~A()

~A()

4)

template<class T>

T f(T\*a,T\*b,int n)

{

T s = (T)0;

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

s += a[i]\*b[i];

return s;

}

void main()

{

double c[3]={1.1,0.22,0.033},d[3]={10.0,100.0,1000.0};

cout<<f(c,d,3)<<endl;

d[0] = 1.0; d[1] = 1.0; d[2] = 1.0;

cout<<f(c,d,3)<<endl;

}

[答案]

66

1.353

5)

class A

{

public:

virtual ~A(){}

};

class B : public A

{

};

void main()

{

B \*bp;

B b;

A a1;

A &a2 = b;

try{

bp = dynamic\_cast<B \*>(&a1);

if (bp)

cout << "Dynamic\_cast (1) OK!"<<endl;

else

cout << "Dynamic\_cast (1) Fail!"<<endl;

bp = dynamic\_cast<B \*>(&a2);

if (bp)

cout << "Dynamic\_cast (2) OK!"<<endl;

else

cout << "Dynamic\_cast (2) Fail!"<<endl;

B &b1 = dynamic\_cast<B &>(a1);

cout << "Dynamic\_cast (3) OK!" <<endl;

}

catch(bad\_cast){

cout << "Dynamic\_cast (3) Fail!"<<endl;

}

}

[答案]

Dynamic\_cast (1) Fail!

Dynamic\_cast (2) OK!

Dynamic\_cast (3) Fail!

6)

class A

{

public:

A(){cout << "A()" << endl;}

~A(){cout << "~A()" << endl;}

virtual void F1(){cout << "A::F1()" << endl;}

virtual void F2(){cout << "A::F2()" << endl;}

void F3(){F1(); cout <<"A::F3()" << endl; F2();}

};

class B : public A

{

public:

B(){cout << "B()" << endl;}

virtual ~B(){cout <<"~B()" << endl;}

void F1( ){cout << "B::F1()" << endl;}

};

class C : public B

{

public:

C(){cout << "C()" << endl;}

~C(){cout << "~C()" << endl;}

void F2(){cout << "C::F2()" << endl;}

void F3(){cout << "C::F3()" << endl;}

};

void main()

{

B b1;

b1.F3();

C c;

c.F3();

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

A \*ap = new B;

ap->F3();

delete ap;

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

B \*bp = new C;

bp->F3();

delete bp;

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

}

[答案]

A()

B()

B::F1()

A::F3()

A::F2()

A()

B()

C()

C::F3()

==================

A()

B()

B::F1()

A::F3()

A::F2()

~A()

==================

A()

B()

C()

B::F1()

A::F3()

C::F2()

~C()

~B()

~A()

=====================

~C()

~B()

~A()

~B()

~A()

2. Please correct the following programs（point out the errors and correct them）(12%)

每题4分，对于主要部分正确的可酌情给1-3分。

1) template<class T>

T add(T x, T y){

return x + y;

}

void main()

{

add(5, 3);

add(5.2, 3.1);

add<int>(5.2, 3);//最好改为add<int>(5, 3);或add<int>((int)5.2, 3);

add(5.2, 3);//改为add<doule>(5.2, 3);或者add(5.2, 3.0);

//或者add(5.2,double(3))

//这道题两个地方这些改法都可接受吧.

}

2)

#include <iostream.h>

class A

{

int m;

static int k;

public:

void SetM(int m)const{ //没有const

A::m = m;

}

static void SetK(int k)const{ //没有const

A::k = k;

}

int GetM()const{

return m;

}

static int GetK()const{ //没有static或者没有const

return k;

}

};

Int A::k;// 加上静态定义

void main()

{

A::SetM(222); //去掉这句 不可以访问

A::SetK(333);

A a1,a2;

a1.SetM(222);//去掉第一句 改为这两句

a2.SetM(222);

cout << a1.GetM()+a2.GetM()<<endl;

cout << a1.GetK()+a2.GetK()<<endl;

}

[答案]

class A

{

int m;

static int k;

public:

void SetM(int m){ //没有const

A::m = m;

}

static void SetK(int k){ //没有const

A::k = k;

}

int GetM()const{

return m;

}

int GetK() const{ ////没有static或者没有const

return k;

}

};

int A::k; // 加上静态定义

void main()

{

A::SetK(333);

A a1,a2;

a1.SetM(222); //去掉第一句 改为这两句

a2.SetM(222);

cout << a1.GetM()+a2.GetM()<<endl;

cout << a1.GetK()+a2.GetK()<<endl;

}

3)

#include <iostream.h>

#include <math.h>

class A

{

double m\_a;

double m\_b;

public:

A(double a = 0.0,double b =0.0){m\_a = a; m\_b = b; }

double operator double(){ //改为operator double() const

return m\_a;

}

double &Abs(){ //最好去掉引用 改为double Abs()

double temp = sqrt(m\_a \* m\_a + m\_b \* m\_b);

return temp;

}

};

void F(const A &a){

cout << a << endl;

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

}

void main()

{

A a(3.0,4.0);

F(a);

}

[答案]

#include <iostream.h>

#include <math.h>

class A

{

double m\_a;

double m\_b;

public:

A(double a = 0.0,double b =0.0){m\_a = a; m\_b = b; }

operator double()const{

return m\_a;

}

double Abs()const{

double temp = sqrt(m\_a \* m\_a + m\_b \* m\_b);

return temp;

}

};

void F(const A &a){

cout << a << endl;

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

}

void main()

{

A a(3.0,4.0);

F(a);

}

3. Fill in the blanks（27%）

//每空一分

1)

#include <iostream.h>

class Fraction

{

int a;

int b;

public:

Fraction(int x=0,int y=0): a(x) , b(y){}

friend Fraction operator+( const Fraction &f1, const Fraction &f2);

friend ostream& operator<<( ostream &out, const Fraction &f);

};

void main()

{

Fraction c1(2,5),c2(3,7);

Fraction c3;

c3 = c1+c2;

cout << c3 << endl;

}

2）

#include <iostream>

using namespace std;

const int MAXSIZE = 3;

class Empty{};

class Full{};

template <class T>

class Queue

{

T \*m\_data; //dynamically allocated array of T

int m\_front; //head of the queue

int m\_rear; //tail of the queue

int m\_size; //Size of the queue

int m\_number; //number of elements in the current queue

public:

Queue(int size = MAXSIZE){

m\_size = size;

m\_number = 0;

m\_front = m\_rear = 0;

m\_data = new T[m\_size];

}

~Queue(){

delete [] m\_data;

}

Queue( const Queue & q){

m\_size = q.m\_size;

m\_number = q.m\_number;

m\_front = q.m\_front;

m\_rear = q.m\_rear;

m\_data = new T[m\_size];

memcpy(m\_data,q.m\_data,sizeof(T)\*m\_size);

}

void EnQueue(const T &t); //append the element at the tail of queue

T DeQueue(); //remove the head element of the queue

bool isFull() const { return m\_number == m\_size; }

bool isEmpty() const { return m\_number == 0; }

};

//The dynamic array *data* will be a circular queue

template <class T>

void Queue<T>:: EnQueue(const T &t)

{

if (isFull())

throw Full();

m\_data[m\_rear++] = t;

m\_number++;

if ( m\_rear == m\_size )

m\_rear = 0;

}

template <class T>

T Queue<T>:: DeQueue(){

if (isEmpty())

throw Empty();

T t = m\_data[m\_front];

if ( ++m\_front == m\_size)

m\_front = 0;

m\_number--;

return t;

}

void F(Queue<int> queue)

{

queue.EnQueue(35),queue.EnQueue(62),queue.EnQueue(55);

queue.DeQueue(),queue.EnQueue(111);

cout << queue.DeQueue() << endl;

cout << queue.DeQueue() << endl;

cout << queue.DeQueue() << endl;

}

void main()

{

Queue <int> queue;

try{

F(queue);

}

catch(Full) {

cout << "The queue is full." << endl;

}

catch(Empty) {

cout << "The queue is Empty." << endl;

}

}

4. Program Design（25%）

Programs use log libraries to write logs. Suppose we are writing a very simple log library that sends log message both to a log file and the console. To write logs to file, we already have implemented a class FileWriter:

class FileWriter

{

public:

static void writeStringToFile(const char \*strMessage);

};

To display logs in console, we already have implemented a class ConsoleDisplayer:

class ConsoleDisplayer

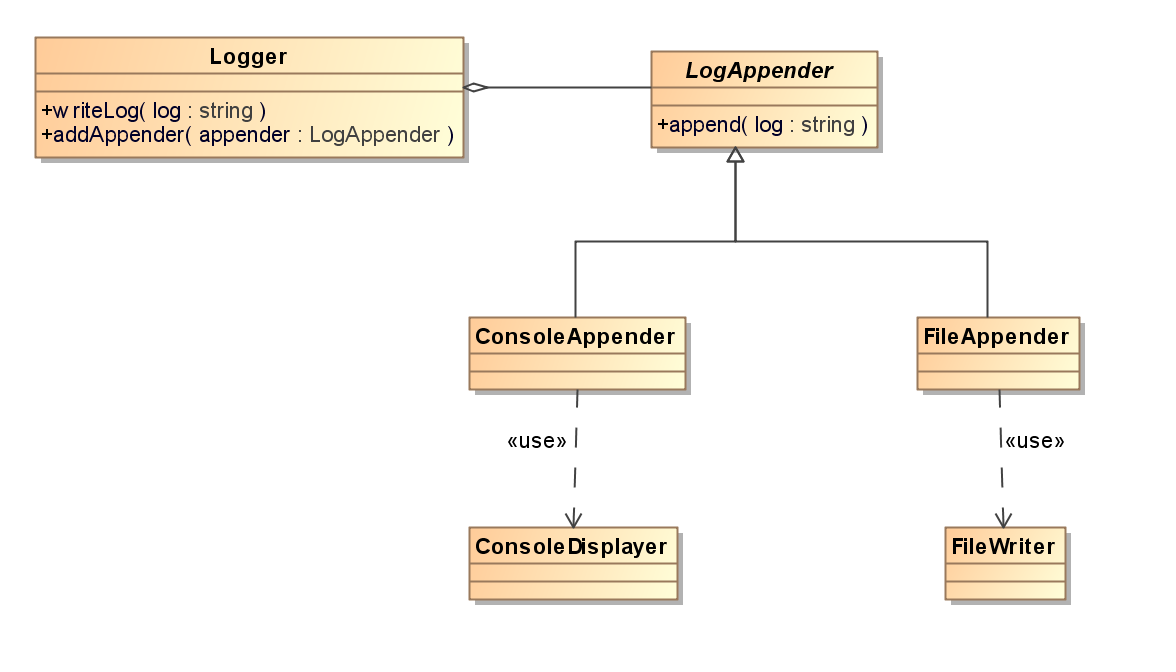
{

public:

static void displayString(const char \*strMessage);

};

Our log library contains four classes: Logger, LogAppender, FileAppender, and ConsoleAppender.



The Logger is used to write logs. The LogAppender is an *abstract* class which shall write log messages to files or consoles. The ConsoleAppender inherits from LogAppender and displays log messages in console using ConsoleDisplayer. The FileAppender also inherits from LogAppender but writes log messages to files using FileWriter. The Logger can attach several LogAppenders to record log messages to different targets at the same time.

Logger is declared as:

class Logger

{

public:

Logger() : m\_count(0){}

void addAppender(LogAppender \*appender);

void writeLog(const char \*log);

private:

LogAppender \*m\_appenders[256];

int m\_count;

};

LogAppender is declared as

class LogAppender

{

public:

virtual void writeLog(cons char \*log) = 0;

};

The log library is supposed to be used as the following example:

Logger logger;

logger.addAppender(new ConsoleAppender());

logger.addAppender(new FileAppender());

logger.write("Hello world.");

Please finish the log library:

1. write the unimplemented methods of class Logger.

2. write the declaration and implementation of class FileAppender.

3. write the declaration and implementation of class ConsoleAppender.

[答案] 共25分 第一题共9分，第二题第三题各8分

**1.**

**//这个函数5分**

**void Logger::addAppender(LogAppender \*appender) //1分**

**{**

**if(appender && m\_count < 256) //1分**

**{**

**m\_appenders[m\_count ++] = appender; //3分**

**}**

**}**

**//这个函数4分**

**void Logger::writeLog(const char \*log) // 1分**

**{**

**for(int i = 0; i < m\_count; i ++) //1分**

**{**

**m\_appenders[i]->writeLog(log); //2分**

**}**

**}**

**2. // 8分**

**class FileAppender : public LogAppender //继承 2分**

**{**

**public: //public: 1分**

**void writeLog(const char \*log) //1分**

**{**

**FileWriter::writeStringToFile(log); //4分: 静态2分 调用2分**

**}**

**};**

**3. //8分**

**class ConsoleAppender : public LogAppender //继承 2分**

**{**

**public: //public: 1分**

**void writeLog(const char \*log) //1分**

**{**

**ConsoleDisplayer::displayString(log); //4分: 静态2分 调用2分**

**}**

**};**