上海大学2019～2020学年度秋季学期试卷(A卷)

成

绩

序  
号

课程名**:** 面向对象程序设计 课程号**:** 08305121 学分**:** 5

应试人声明：

我保证遵守《上海大学学生手册》中的《上海大学考场规则》，如有考试违纪、作弊行为，愿意接受《上海大学学生考试违纪、作弊行为界定及处分规定》的纪律处分。

应试人 应试人学号 应试人所在院系

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 题号 | **一(20)** | **二(20)** | **三(30)** | **四(30)** |
| 得分 |  |  |  |  |

**———————————­———————————————————————————**

得

分

**一、判断题（每小题2分，共20分）**

1. C++程序中，任何类都有默认的构造函数。 (**×**)

2. C++程序中，非静态成员函数中的隐含形参**this**是一个指针变量，可以  
执行 **this**++ 运算。 (**×**)

3. C++程序中，类的非静态常量成员函数的隐含**this**指针为常量指针常量。  
即为**const** 类名 \* **const** **this**。 (**√**)

4. C++程序中，某函数的形参为值传递对象，则调用该函数时，程序会自动  
调用拷贝构造函数用实参对象拷贝构造形参对象；函数返回时，自动调用  
析构函数析构形参对象。 (**√**)

5. C++程序中，某函数的形参为引用型形参，则该引用的生命周期为函数调  
用时存在，函数返回后不存在。在引用的生命周期内，该引用绑定的实参  
变量（对象）不可更改。 (**√**)

6. C++程序中，可以用基类的对象初始化派生类的引用声明。 (**×**)

7. C++程序中，基类的所有成员(包括私有成员)在派生类中都可直接访问。 (**×**)

8. C++程序中，可以创建抽象类的对象。 (**×**)

9. C++程序中，重载运算符函数时，不能改变运算符的优先级。 (**√**)

10.C++程序中，重载运算符函数时，不能改变运算符的操作数个数。 (**√**)

**二、填空题（每空2分，共20分）**请根据运行结果，完成程序。

得

分

#**include** <iostream>

#**include** <cstring>

**using** **namespace** ① std ;

**template** <**typename** TYPE> **void** Swap(TYPE &a, TYPE &b)

{ // 函数模板

TYPE temp;

temp = a;

a = b;

b = temp;

}

**void** Swap(**char** \*str1, **char** \*str2) // 重载函数模板

{

**char** \*temp = **new** **char**[strlen(② str1 )+1];

strcpy(temp, str1);

strcpy(str1, str2);

strcpy(str2, temp);

③ **delete**  [] temp;

}

**此处填class，正确！**

**template** <④ **typename**  TYPE>

**void** Show(**const** TYPE &a, **const** TYPE &b)

{

cout << a << ", " << b << endl;

}

**int** main()

{

**char** c1='A', c2='B';

**int** a=3, b=5;

**此处填unsigned char，正确！**

**double** x=3.3, y=5.5;

⑤ **char** str1[80] = "Tom", str2[80] = "Jerry";

**const** **char** \*p1 = "Hello", \*p2 = "你好";

运行结果(2)

B, A

5, 3

5.5, 3.3

Jerry, Tom

你好, Hello

Swap(⑥ c1 , c2); Show(c1, c2);

Swap(⑦ a , b); Show(a, b);

Swap(⑧ x , y); Show(x, y);

Swap(⑨ str1 , str2); Show(str1, str2);

Swap(⑩ p1 , p2); Show(p1, p2);

**return** 0;

}

**三、阅读程序写出运行结果及简答题（共30分）**

得

分

3.1**（5**分**）**

#**include** <iostream>

**using** **namespace** std;

//提示:

//

**char** \*Trans(**unsigned** **int** n, **int** base, **char** \*str)

{

**char** temp, BASE[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

**int** i=0, j=0;

**if**(n==0)

{

str[0] = '0';

str[1] = '\0';

**return** str;

}

**while**(n!=0)

{

str[i] = BASE[n%base];

n /= base;

i++;

}

str[i] = '\0'; // 生成了C-字符串str

**for**(j=0,i--; j<i; j++,i--) // 将C-字符串str的各字符（内容）倒置

{

temp = str[i];

运行结果(3.1)

127 = ( 1111111 )\_2

127 = ( 11201 )\_3

127 = ( 127 )\_10

127 = ( 7F )\_16

127 = ( 3J )\_36

str[i] = str[j];

str[j] = temp;

}

**return** str;

}

**int** main()

{

**char** str[80];

**unsigned** **int** n = 127;

cout << n << " = (" << Trans(n, 2, str) << ")\_2" << endl;

cout << n << " = (" << Trans(n, 3, str) << ")\_3" << endl;

cout << n << " = (" << Trans(n, 10, str) << ")\_10" << endl;

cout << n << " = (" << Trans(n, 16, str) << ")\_16" << endl;

cout << n << " = (" << Trans(n, 36, str) << ")\_36" << endl;

**return** 0;

}

3.2 **（5**分**）**

运行结果(3.2)

构造 (2, 3)

构造 (3, 3)

返回操作系统

(3, 3)在直线 y = x 上

(2, 3)不在直线 y = x 上

#**include** <iostream>

**using** **namespace** std;

**class** Point

{

**public**:

Point(**int** x=0, **int** y=0) : a(x), b(y)

{

cout << "构造 " << \***this** << endl;

}

**virtual** ~Point()

{

cout << \***this** << (a==b? "":"不") << "在直线 y = x 上" << endl;

}

**friend** ostream & **operator**<<(ostream &out, **const** Point &p)

{

out << "(" << p.a << ", " << p.b << ")";

**return** out;

}

**private**:

**int** a, b;

};

**int** main()

{

Point x(2, 3), y(3, 3);

cout << "返回操作系统" << endl;

**return** 0;

}

3.3 **（5**分**）**如下程序能演奏“致爱丽丝”的前9个音符。程序采用多文件结构，共有一个头文件、两个源程序文件。程序中使用了标准MIDI函数（打开/关闭MIDI设备、选择音色即选择乐器、开音/关音等），暂时不必深究这些函数。

// MIDI.h 文件之一（头文件）

// 本程序需要连接多媒体库libwinmm.a。设置方法:

// Project/Settings.../Link/Libraries项的值为C:\MinGWStudio\MinGW\lib\libwinmm.a

#**ifndef** MIDI\_H

#**define** MIDI\_H

#**include** <windows.h>

#**include** <iostream>

#**include** <string>

**using** **namespace** std;

**const** **int** N = 100;

**struct** Music

{

**int** deltaTime, note, volume; // 时长（拍数）、音符、音量

};

**class** MIDI // 乐器数字化接口(Musical Instrument Digital Interface)

{

**public**:

MIDI(**const** string &Name="", **int** n=0, **const** Music \*data=NULL);

// 构造函数

MIDI(**const** MIDI &m); // 拷贝构造函数

**virtual** ~MIDI(); // 析构函数（虚函数）

**virtual** **void** Play() **const**; // 演奏乐曲（虚函数）

**protected**:

**static** DWORD MidiOutMessage(**int** iStatus, **int** iChannel,

**int** iFlip, **int** iVolume);

// 给MIDI设备发送消息，用于设置音色、发音或停止发音。DWORD为unsigned long

**static** HMIDIOUT hMIDI; // 静态数据成员（MIDI设备句柄）

**static** **int** NumObjs; // 创建对象的个数

string name; // 乐曲名

**int** len; // 乐曲的音符数（不超过N）

Music x[N]; // 乐谱

};

**class** MIDI\_Piano : **public** MIDI // 钢琴类

{

**public**:

MIDI\_Piano(**const** string &Name="", **int** n=0, **const** Music \*data=NULL);

**void** Play() **const**;

**protected**:

**int** timbre; // 音色（对应某种乐器）

};

**class** MIDI\_Violin : **public** MIDI // 小提琴类

{

**public**:

MIDI\_Violin(**const** string&Name="", **int** n=0, **const** Music\*data=NULL);

**void** Play() **const**;

**protected**:

**int** timbre; // 音色（对应某种乐器）

};

**class** MIDI\_ChurchOrgan : **public** MIDI // 教堂管风琴类

{

**public**:

MIDI\_ChurchOrgan(**const** string&Name="",**int** n=0,**const** Music\*data=NULL);

**void** Play() **const**;

**protected**:

**int** timbre; // 音色（对应某种乐器）

};

#**endif**

// MIDI.cpp 文件之二（源程序文件）

#**include** "MIDI.h"

HMIDIOUT MIDI::hMIDI; // 静态数据成员定义（MIDI设备句柄）

**int** MIDI::NumObjs = 0; // 静态数据成员定义（对象的个数）

DWORD MIDI::MidiOutMessage(**int** iStatus,**int** iChannel,**int** iFlip,**int** iVolume)

{

DWORD dwMessage = (iVolume<<16) | (iFlip<<8) | iStatus | iChannel;

**return** midiOutShortMsg(hMIDI, dwMessage);

} // 本函数的功能是向MIDI设备发送消息，实现设置音色、设备发出声音等功能。无须深究。

MIDI::MIDI(**const** string &Name, **int** n, **const** Music \*data)  
 : name(Name), len(n)

{

**if**(++NumObjs==1) // 若是第一个对象，则打开MIDI设备

midiOutOpen(&hMIDI, 0, 0, 0, CALLBACK\_NULL);

**if**(len<0) len = 0;

**if**(data==NULL) return;

**for**(**int** i=0; i<len; i++)

x[i] = data[i];

}

MIDI::MIDI(**const** MIDI &m):name(m.name), len(m.len) // 拷贝构造函数

{

++NumObjs;

**for**(**int** i=0; i<len; i++)

x[i] = m.x[i];

}

MIDI::~MIDI()

{

**if**(--NumObjs==0) // 若是最后一个对象，则关闭MIDI设备

midiOutClose(hMIDI);

}

**void** MIDI::Play() **const**

{

cout << name << endl;

**for**(**int** i=0; i<len; i++)

{

cout << x[i].note << " ";

MidiOutMessage(0x90, 0x00, x[i].note, x[i].volume); //开音

Sleep(x[i].deltaTime\*60000/180); // 使开音延时，每分钟180拍

MidiOutMessage(0x80, 0x00, x[i].note, 127); //关音

}

cout << endl;

}

MIDI\_Piano::MIDI\_Piano(**const** string &Name, **int** n,

**const** Music \*data) : MIDI(Name, n, data), timbre(0)

{

}

**void** MIDI\_Piano::Play() **const**

{

cout << "钢琴演奏\t"; // 请注意此处无换行

MidiOutMessage(0xC0, 0x00, timbre, 0); // 设置音色（即乐器）

MIDI::Play();

}

MIDI\_Violin::MIDI\_Violin(**const** string &Name, **int** n,

**const** Music \*data): MIDI(Name, n, data), timbre(40)

{

}

**void** MIDI\_Violin::Play() **const**

{

cout << "小提琴演奏\t"; // 请注意此处无换行

MidiOutMessage(0xC0, 0x00, timbre, 0); // 设置音色（即乐器）

MIDI::Play();

}

MIDI\_ChurchOrgan::MIDI\_ChurchOrgan(**const** string &Name, **int** n,

**const** Music \*data) : MIDI(Name, n, data), timbre(19)

{

}

**void** MIDI\_ChurchOrgan::Play() **const**

{

cout << "教堂管风琴演奏\t"; // 请注意此处无换行

MidiOutMessage(0xC0, 0x00, timbre, 0); // 设置音色（即乐器）

MIDI::Play();

}

// MIDI\_TEST.cpp 文件之三（源程序文件）

#**include** "MIDI.h"

**void** Playing(**const** MIDI &m)

{

m.Play();

}

**int** main()

{

Music data[] = {{1,76,96}, {1,75,96}, {1,76,96}, {1,75,96},

{1,76,96}, {1,71,96}, {1,74,96}, {1,72,96}, {3,69,96}};

**int** n = **sizeof**(data)/**sizeof**(\*data);

MIDI\_Piano p("致爱丽丝", n, data);

MIDI\_Violin v("致爱丽丝", n, data);

MIDI\_ChurchOrgan c("致爱丽丝", n, data);

Playing(v);

Playing(c);

Playing(p);

**return** 0;

}

运行结果(3.3) 部分结果已给出

小提琴演奏 致爱丽丝

76 75 76 75 76 71 74 72 69

教堂管风琴演奏 致爱丽丝

76 75 76 75 76 71 74 72 69

钢琴演奏 致爱丽丝

76 75 76 75 76 71 74 72 69

**此处顺序有错，仅扣少量分，不重复扣分。**

**出题时本无意挖陷阱，修改后变成了陷阱【:(】**

3.4 对于3.3题中的代码，简要回答如下问题**（15**分，每小题3分**）**

(1) 能否创建基类（MIDI类）的对象？为什么？

答: **能**。因为MIDI类中**没有纯虚函数**（即MIDI类**不是抽象类**）。

(2) MIDI类中为什么必须定义拷贝构造函数（尽管本程序尚未测试拷贝构造函数）？

答: 因为在所有方式**创建对象时需要修改静态数据成员NumObjs的值**。若不定义拷贝构造函数，编译系统为该类提供的默认的拷贝构造函数并不修改NumObjs的值。

**关键看回答问题是否涉及到了要点，表达方面的问题不做过高要求**

(3) MIDI类中的静态数据成员NumObjs的作用是什么？

答: 作为一种**状态(或标志)值**，**决定**是否打开或关闭MIDI设备。

(4) MIDI类中，若将Play函数设计成非虚函数，则程序运行会产生什么效果？

答: 则Playing函数调用时，m.Play();语句仅调用基类的成员函数Play()，不会根据实参对象的具体类型调用相应的派生类的成员函数Play()。**不表现为多态性**。

(5) MIDI类中，若将Play函数设计成非常量成员函数，程序的其他部分不修改，则程序能否通过编译？若不能通过编译，请指出哪个文件的哪条语句出错，为什么？

答: **不能**通过编译。MIDI\_TEST.cpp文件中定义的Playing函数中的m.Play();语句有错。因为**常对象m不能调用非常量成员函数**。

**四、编写程序（共30分）**

得

分

有如下尚未完成的代码，其中已完成构成单向链表类模板所需的结点类模板(**template** <**typename** T> **class** Node;)的设计；单向链表类模板(**template** <**typename** T> **class** LinkList;)中数据成员、成员函数原型已设计。请在类体外完成①~⑧函数的定义（①~⑥每个函数4分，⑦~⑧每个函数3分）。

// LinkList.h

#**ifndef** LINKLIST\_H

#**define** LINKLIST\_H

#**include** <iostream>

**using** **namespace** std;

**template** <**typename** T> **class** LinkList; // 提前声明

**template** <**typename** T> **class** Node // 结点类模板

{

**public**:

Node() : next(NULL) {}

Node(**const** T &t) : data(t), next(NULL) {}

Node(**const** Node<T> &node) : data(node.data), next(NULL) {}

Node<T> & **operator**=(**const** Node<T> &node)

{

data = node.data;

**return** \***this**;

}

**friend** **class** LinkList<T>;

**private**:

T data;

Node<T> \*next;

};

**template** <**typename** T> **class** LinkList // 单向链表类模板

{

**public**:

LinkList(**int** n=0, **const** T \*x=NULL); // ① 构造函数

LinkList(**const** LinkList<T> &link); // ② 拷贝构造函数

**virtual** ~LinkList(); // ③ 析构函数

LinkList<T> & **operator**=(**const** LinkList<T> &link);

// ④ 赋值运算符函数

**void** FreeList(); // ⑤ 释放所有结点

**void** Append(**const** T &t); // ⑥ 追加新的尾结点

**virtual** **void** ShowList(ostream &out)**const**; // ⑦ 输出

**protected**:

Node<T> \*head;

}; // ⑧ 重载**operator**<< 运算符

**template** <**typename** T>

LinkList<T>::LinkList(**int** n, **const** T \*x) : head(NULL)

{

**1. 函数首部及返回1~2分；**

**2. 函数功能实现过程（如:循环、条件判断、new、delete等）剩余分数；**

**3. 实现方案可能有多种，不带头结点/带头结点皆可，成员函数定义时，只要功能基本正确。大小写的问题，符号[]、\*误用等小错误略扣少量分或可不扣分。**

**4. 相同的错误不重复扣分（例如:template <typename T> 未写等）。**

Node<T> \*p;

**for**(**int** i=n-1; i>=0; i--)

{

p = **new** Node<T>;

**if**(x!=NULL)

p->data = x[i];

p->next = head;

head = p;

}

}

**template** <**typename** T>

LinkList<T>::LinkList(**const** LinkList<T> &link) : head(NULL)

{

\***this** = link;

}

**template** <**typename** T>

LinkList<T>::~LinkList()

{

FreeList();

}

**template** <**typename** T>

LinkList<T> & LinkList<T>::**operator**=(**const** LinkList<T> &link)

{

**if**(**this** == &link) **return** \***this**;

FreeList();

**for**(Node<T> \*p=link.head; p!=NULL; p=p->next)

Append(p->data);

**return** \***this**;

}

**template** <**typename** T>

**void** LinkList<T>::FreeList()

{

Node<T> \*p;

**while**(head!=NULL)

{

p = head;

head = head->next;

**delete** p;

}

}

**template** <**typename** T>

**void** LinkList<T>::Append(**const** T &t)

{

**if**(head==NULL)

{

head = **new** Node<T>(t);

head->next = NULL;

**return**;

}

Node<T> \*p=head;

**while**(p->next!=NULL)

p = p->next;

p->next = **new** Node<T>(t);

p->next->next = NULL;

}

**template** <**typename** T>

**void** LinkList<T>::ShowList(ostream &out) **const**

{

out << "head ";

**for**(Node<T> \*p=head; p!=NULL; p=p->next)

out << " -> " << p->data;

out << " -> NULL";

}

//////////////////

**template** <**typename** T>

ostream & **operator**<<(ostream &out, **const** LinkList<T> &link)

{

link.ShowList(out);

**return** out;

}