一、单项选择题（每题2分，共20分）

1-5 BDBDD 6-10 CCCCA

二、程序分析题

1.读程序，写运行结果。

B::~B() called

A::~A() called

2.改错题。

①using namespace std -> using namespace std;

②a\_ST(const a\_ST aRef){ -> a\_ST(const a\_ST &aRef){

③静态数据成员未在类外初始化 -> int a\_ST:: KST;

④st->display(); -> st.display();

⑤new完了未加delete -> new 后面 加上 delete sp;

3.读程序，写运行结果。

n1: 随机值，n2: 0

n1:1024,n2:1024

12

12

4

三、编程题。

1.按照下列类声明，实现Point类。

#include <iostream>

#include <cmath>

using namespace std;

class Point

{

public:

Point(int aX,int aY);

Point(const Point &aRef);

Point& operator=(const Point &aRef);

bool operator==(const Point &aRef);

~Point();

int GetX() const;

int GetY() const;

void SetX(int aX);

void SetY(int aY);

int Diatance(Point &aRef);

private:

int x;

int y;

};

Point::Point(int aX, int aY)

{

x = aX;

y = aY;

}

Point::Point(const Point &aRef)

{

x = aRef.x;

y = aRef.y;

}

Point& Point::operator=(const Point &aRef)

{

x = aRef.x;

y = aRef.y;

return \*this;

}

bool Point::operator==(const Point &aRef)

{

if (x == aRef.x && y == aRef.y)

{

return true;

}

return false;

}

Point::~Point(){}

int Point::GetX() const

{

return x;

}

int Point::GetY() const

{

return y;

}

void Point::SetX(int aX)

{

x = aX;

}

void Point::SetY(int aY)

{

y = aY;

}

int Point::Diatance(Point &aRef)

{

return sqrt((x - aRef.x)\*(x - aRef.x)

+ (y - aRef.y)\*(y - aRef.y));

}

int main()

{

Point p(9,0);

Point p1(1,1);

Point p2 = p1;

cout << p2.GetX() << endl;

cout << (p2 == p1) << endl;

cout << p1.Diatance(p) << endl;

return 0;

}

2.在不修改Test类的情况下，通过编写程序直接或间接调用Test类的所有成员函数。

#include <iostream>

using namespace std;

class Test

{

public:

void Display() const

{

}

protected:

Test(int aVal){

val = aVal;

}

void Function()

{

val = 0;

}

private:

int val;

};

class MyClass:public Test

{

public:

MyClass(int a):Test(a){}

void display() const{

Display();

}

void function()

{

Function();

}

};

3.认真阅读下面要求，完成本次任务代码的编写。

#include <iostream>

#include <string>

using namespace std;

class Info

{

public:

Info(string name = "",string phone = "",string address = "");

void setName(string name);

void setPhone(string phone);

void setAddress(string address);

string getName();

string getPhone();

string getAddress();

private:

string m\_Name;

string m\_Phone;

string m\_Address;

};

Info::Info(string name, string phone , string address)

{

m\_Name = name; m\_Phone = phone; m\_Address = address;

}

void Info::setName(string name)

{

m\_Name = name;

}

void Info::setPhone(string phone)

{

m\_Phone = phone;

}

void Info::setAddress(string address)

{

m\_Address = address;

}

string Info::getName()

{

return m\_Name;

}

string Info::getPhone()

{

return m\_Phone;

}

string Info::getAddress()

{

return m\_Address;

}

class AddressBook

{

public:

//需在构造函数中根据size动态创建存放信息的数组内存

AddressBook(int size = 10);

//拷贝构造函数

AddressBook(const AddressBook & aRef);

//赋值运算符的重载

AddressBook &operator=(const AddressBook & aRef);

//需在析构函数中队构造函数中动态申请的内存进行释放

~AddressBook();

//通讯录中添加信息，添加成功后返回true,失败返回false

//成功失败的依据为是否查过通讯录的最大长度

bool addInfo(Info & Info);

//根据姓名的查找信息，返回匹配信息的下标，没有匹配返回-1

int findInfoByName(string name);

//下标运算符的重载

Info & operator[](int index);

//根据姓名打印信息

void printInfoByName(string name);

private:

int m\_CurCount; //当前通讯录信息个数

const int m\_MaxSize; //通讯录最大长度

Info \* m\_pArr; //指向兑取存放信息的数组内存

};

//需在构造函数中根据size动态创建存放信息的数组内存

AddressBook::AddressBook(int size)

:m\_MaxSize(size){

m\_CurCount = 0;

m\_pArr = new Info[size];

}

//拷贝构造函数

AddressBook::AddressBook(const AddressBook & aRef)

: m\_CurCount(aRef.m\_CurCount), m\_MaxSize(aRef.m\_MaxSize)

{

m\_pArr = new Info[m\_MaxSize];

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

{

m\_pArr[i].setName(aRef.m\_pArr[i].getName());

m\_pArr[i].setPhone(aRef.m\_pArr[i].getPhone());

m\_pArr[i].setAddress(aRef.m\_pArr[i].getAddress());

}

}

//赋值运算符的重载

AddressBook &AddressBook::operator=(const AddressBook & aRef)

{

m\_CurCount = aRef.m\_CurCount;

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

{

m\_pArr[i].setName(aRef.m\_pArr[i].getName());

m\_pArr[i].setPhone(aRef.m\_pArr[i].getPhone());

m\_pArr[i].setAddress(aRef.m\_pArr[i].getAddress());

}

return \*this;

}

//需在析构函数中队构造函数中动态申请的内存进行释放

AddressBook::~AddressBook()

{

delete []m\_pArr;

}

//通讯录中添加信息，添加成功后返回true,失败返回false

//成功失败的依据为是否查过通讯录的最大长度

bool AddressBook::addInfo(Info & Info)

{

if ( m\_CurCount >= m\_MaxSize)

{

cout << "通讯录已满，无法添加！" << endl;

return false;

}

m\_pArr[m\_CurCount].setName(Info.getName());

m\_CurCount++;

return true;

}

//根据姓名的查找信息，返回匹配信息的下标，没有匹配返回-1

int AddressBook::findInfoByName(string name)

{

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

{

if(m\_pArr[i].getName() == name)

return i;

}

return -1;

}

//下标运算符的重载

Info & AddressBook::operator[](int index)

{

if (index >= m\_CurCount || index < 0)

{

cout << "通讯录下标越界！" << endl;

exit(1);

}

return m\_pArr[index];

}

//根据姓名打印信息

void AddressBook::printInfoByName(string name)

{

int pos = findInfoByName(name);

if (pos >= 0)

{

cout << "姓 名:" << m\_pArr[pos].getName() << endl;

cout << "手机号:" << m\_pArr[pos].getPhone() << endl;

cout << "地 址:" << m\_pArr[pos].getAddress() << endl;

}

}