# 武汉大学计算机学院

# 2021-2022学年度第1学期2020级

# 《高级语言程序设计》期末考试试卷A卷参考答案

# 姓名： 学号： 班级：

说明：答案请全部写在答题纸上，写在试卷上无效。

未经主考教师同意，考试试卷、答题纸、草稿纸均不得带离考场，否则视为违规。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 题号 | 一 | 二 | 三 | 四 | 五 | 总分 |
| 总分 | 20 | 10 | 20 | 30 | 20 | 100 |

## 简答题：（共4小题，每小题5分，共20分）

1. 请说明用关键字const 分别修饰引用、数组、指针、对象的声明时有什么作用？使用时有何特殊要求？

**【参考答案】：（目标1），得分点 1,1,1,2**

用const修饰引用声明了常引用，常引用的被引用的目标不能被更新；

用const修饰数组声明了常数组，常数组中的元素不能被更新；

用const修饰指针声明了常指针，常指针是指向常量的指针；

用const修饰对象声明了常对象，常对象声明时必须被初始化，不能修改对象的数据成员。

1. 请画出字符串 char str[ ] = “Computer Science” 的数据存储示意图

**【参考答案】：（目标1），得分点 5**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | o | m | p | u | t | e | r |  | S | c | i | e | n | c | e | \0 |

1. 什么是函数重载？名称相同而功能不同的函数被调用时，是如何确定正确的函数体入口？

**【参考答案】：（目标2），得分点 3,2**

函数重载是指具有相似功能的不同函数使用同一函数名，但这些同名函数的参数类型、参数个数、返回值、函数功能可以不同

编译系统将根据函数参数的类型和个数来匹配使用哪一个函数的代码

1. 使用继承机制定义派生类时，基类的哪些函数不能自动被派生类继承？请说明派生类定义的对象，其构造函数和析构函数执行的顺序是怎样的？

**【参考答案】：（目标2），得分点 2,3**

(1)基类构造函数、析构函数不能被派生类自动继承，基类的私有成员函数也不能被派生类继承；

(2)派生类定义对象时，构造函数的调用顺序为：首先调用基类的构造函数；然后再调用对象成员类的构造函数；最后调用派生类的构造函数。

析构函数的调用顺序是：先调用派生类的析构函数；再调用对象成员类的析构函数（如果有对象成员）；最后调用基类的析构函数。

派生类对象的析构函数调用顺序与调用构造函数的顺序相反。

## 分析改错题：（共1小题，共10分）

1. （10分）下列程序存在错误，请分析找出它们，说明错误原因并改正：

|  |  |
| --- | --- |
|  | #include <iostream> |
|  | #include <cassert> |
|  | using namespace std; |
|  |  |
|  | class Point { |
|  | public: |
|  | Point() : x(0), y(0) { |
|  | cout<<"Point无参数构造函数."<<endl; |
|  | } |
|  | Point(int x, int y) : x(x), y(y) { |
|  | cout<< "Point有参数构造."<<endl; |
|  | } |
|  | ~Point() { cout<<"Point析构函数."<<endl; } |
|  | int getX() const { return x; } |
|  | int getY() const { return y; } |
|  | void move(int newX, int newY) { |
|  | x = newX; |
|  | y = newY; |
|  | } |
|  | private: |
|  | int x, y; |
|  | }; |
|  |  |
|  | class ArrayPoints { //动态数组类 |
|  | public: |
|  | ArrayPoints(int size) : size(size) { |
|  | points = new Point[size]; |
|  | } |
|  | ~ArrayPoints() { |
|  | cout << "删除数组..." << endl; |
|  | delete[] points; |
|  | } |
|  | Point &element(int index) { //获得下标为index的数组元素 |
|  | assert(index >= 0 && index < size); //检查数组下标不会越界 |
|  | return points[index]; |
|  | } |
|  | private: |
|  | Point \*points; //指向动态数组首地址 |
|  | int size; //数组大小 |
|  | }; |
|  |  |
|  | int main() { |
|  | int count; |
|  | cout << "请输入指针数量: "; |
|  | cin >> count; |
|  | ArrayPoints pointsArr1(count); //创建对象数组 |
|  | pointsArr1.element(0).move(5,10); |
|  | pointsArr1.element(1).move(15,20); |
|  | ArrayPoints pointsArr2 = pointsArr1; //创建对象数组副本 |
|  |  |
|  | cout << "数组Point\_0: " << pointsArr2.element(0).getX() << "," |
|  | << pointsArr2.element(0).getY() << endl; |
|  | cout << "数组Point\_1: " << pointsArr2.element(1).x << ", " |
|  | << pointsArr2.element(1).y << endl; |
|  | return 0; |
|  | } |

**【参考答案】: （目标3）**

**得分点 3**

**错误1） ArrayPoints 没有定义拷贝复制函数，进行深拷贝处理；**

**得分点 2**

**错误2）53、54行中Point 的成员x，y是私有成员，不能使用pointsArr2.element(1).x 和pointsArr2.element(1).y 直接访问。**

**得分点 3**

**改正 ：1）增加拷贝构造函数**

|  |  |
| --- | --- |
|  | ArrayPoints::ArrayPoints(const ArrayPoints& v) { |
|  | size = v.size; |
|  | points = new Point[size]; |
|  | for (int i = 0; i < size; i++) |
|  | points[i] = v.points[i]; |
|  | } |

**得分点2**

**2）使用getX（）和getY（）访问Point私有成员x，y**

|  |  |
| --- | --- |
|  | cout << "数组Point\_1: " << pointsArr2.element(1).***getX()*** <<"," |
|  | << pointsArr2.element(1).***getY()*** << endl; |

## 程序阅读与分析题：（共2小题，每小题各10分，共20分）

1. （10分）请仔细阅读以下程序，完成下列两个任务：
2. 函数functsum计算的值，如果 functsum( )调用时的实参数是sum = functsum(1.0，1.0，3)，sum的结果值将是什么值？
3. 请用非递归方式重新改写函数functsum( )。

|  |  |
| --- | --- |
|  | #include <iostream> |
|  | using namespace std; |
|  |  |
|  | float functsum(float a, float q, int n){ |
|  | static float parts = 0; |
|  | if (n == 1) { |
|  | parts = a + q; |
|  | } |
|  | else { |
|  | parts = parts + functsum(a, q, n - 1)\* functsum(a, q, 1); |
|  | } |
|  | return parts; |
|  | } |
|  | int main( ){ |
|  | float sum = 0; |
|  | float input\_a = 0.0; |
|  | float input\_q = 0.0; |
|  | int input\_n = 0; |
|  |  |
|  | cout << "请输入 a q n的值 "; |
|  | cin >> input\_a >> input\_q >> input\_n; |
|  |  |
|  | sum = functsum(input\_a, input\_q, input\_n); |
|  | cout << "sum:" << sum; |
|  | return 0; |
|  | } |

**【参考答案】:（目标1）得分点 4, 6**

**1）：sum结果是14**

**2）：使用循环结构，代码略**

1. （10分）请仔细阅读以下程序，写出程序的输出结果

|  |  |
| --- | --- |
|  | #include <iostream> |
|  | using namespace std; |
|  | class A |
|  | { public: |
|  | virtual void print(void) { cout<<"A::print()"<<endl; } |
|  | }; |
|  | class B:public A |
|  | {public: |
|  | virtual void print(void) { cout<<"B::print()"<<endl; }; |
|  | }; |
|  | class C:public B |
|  | {public: |
|  | virtual void print(void) { cout<<"C::print()"<<endl; } |
|  | }; |
|  | void print(A a) |
|  | { a.print(); |
|  | } |
|  | int main( ) |
|  | { A a, \*pa,\*pb,\*pc; |
|  | B b; |
|  | C c; |
|  | pa=&a; pb=&b; pc=&c; |
|  | pa->print(); |
|  | pb->print(); |
|  | pc->print(); |
|  | cin.ignore (); |
|  | return 0; |
|  | } |

**【参考答案】(目标3) 得分点3,3,4**

**A::print()**

**B::print()**

**C::print()**

## 编程实现题（共2小题，每小题各15分，共30分）

1. （15分）请编程完成函数xorIntArrayMerge( )的代码，该函数的功能是将两个整数数组合并，合并后的结果中将会去掉原来两个数组中都出现的整数。

例如：有如下的两个数组：

int arrA[6 = {54,35，86, 77, 29, 101};

int arrB[5] = {35, 65, 86, 99,29};

合并后的结果为 {54, 77,101,65, ,99}

若A、B为指向合并前的两个数组的指针，lenA、lenB为两个数组的长度，C为指向合并后数组的指针。

函数定义为：

void xorIntArrayMerge (int \*A, int lenA, int \*B, int lenB, int\*C

{

//请在此处补充完成函数处理代码

};

**【参考答案】:（目标 1）得分点 5,10**

#include <iostream>

using namespace std;

void xorIntArrayMerge(int A[], int lenA, int B[], int lenB, int C[])

{

int currentElement = 0;

int indexC = 0;

bool equalFlag = false;

int i = 0, j = 0;

//从数组A中剔除和数组B中重复的元素

for (i = 0; i < lenA; i++) {

currentElement = \*(A + i);

for (j = 0; j < lenB; j++) {

if (currentElement == \*(B + j)) {

equalFlag = true;

break;

}

}

if (!equalFlag) {

\*(C + indexC) = currentElement;

indexC++;

}

equalFlag = false;

}

equalFlag = false;

//从数组B中剔除和数组A中重复的元素

for (j = 0; j < lenB; j++) {

currentElement = \*(B + j);

for (i = 0; i < lenA; i++) {

if (currentElement == \*(A + i)) {

equalFlag = true;

break;

}

}

if (!equalFlag) {

\*(C + indexC) = currentElement;

indexC++;

}

equalFlag = false;

}

return;

}

int main()

{

int arrA[6] = { 54, 35, 86, 77, 29, 101 };

int arrB[5] = { 35, 65, 86, 99, 29 };

int arrSizeA = 6;

int arrSizeB = 5;

int arrC[11] = { 0 };

xorIntArrayMerge(arrA, arrSizeA, arrB, arrSizeB, arrC);

cin.ignore();

return 0;

}

1. （15分）请编程实现以下功能：  
   输入为：一个不超多5位的整数；  
   输出为：1）这个整数有多少位，2）逐位逆序打印每一位数字

**【参考答案】:（目标2）得分点6,9**

//code\_9.cpp

#include <iostream>

using namespace std;

int main() {

int inputN, right\_digit, newnum = 0, length = 0;

cout << "输入不超过5位的整数: ";

cin >> inputN;

cout << "逆序输出结果为： ";

//处理负整数

if (inputN < 0) {

inputN = -inputN;

cout << "-";

}

do {

right\_digit = inputN % 10;

cout << right\_digit;

inputN /= 10;

length++;

} while (inputN != 0);

cout << endl;

cout << "数据位数为：" << length << endl;

cin.ignore();

return 0;

}

## 程序分析与设计题（共20分）

1. （20分）请设计一组科学计算所需的类，用于处理标量（Scalar）、向量（Vector）和矩阵（Matrix）数据。

1）由抽象类张量(Tensor)派生出标量（Scalar）、向量（Vector）和矩阵（Matrix）三个子类；

2）标量（Scalar）、向量（Vector）和矩阵（Matrix）的构造函数负责动态申请所需的存储空间保存各自数据，析构函数负责释放所申请的存储空间；

3）重载运算符 “+”和“=”能够对标量（Scalar）、向量（Vector）和矩阵（Matrix）数据执行相应的加法和赋值运算 。

请按要求完成以下任务；

1）请用规范的C++语言形式写出上述各类的class定义，要求写出所需的成员变量和成员函数，但成员函数的函数体代码均不必在class定义时实现；

2）请单独写出向量（Vector）的构造函数Vector：：Vector（）和析构函数Vector：：~ Vector（）的完整实现代码，构造函数所需的参数请自行定义；

3）请单独写出向量（Vector）的运算符重载函数Vector：：operator+( )的完整实现代码。

**说明：**

(1)代码中要求有注释说明

(2)对于成员函数定义，需要注释说明其功能、形式参数的含义、返回值等；

(3)对于成员变量，需要注释说明其含义用途；

**【参考答案】:**

**1）（目标4）得分点8**

**2）（目标3）得分点 6**

**3）（目标3）得分点 6**

**此设计题目为半开放性问题，类的成员定义可以不同，只要所给出的的设计理由具有合理性，即可得相应分；类定义的语法应准确规范，面向对象的概念应用要求正确；函数定义的语法应准确规范。**

**参考代码如下，以便理解设计实现要点，学生答案的代码可以不同。**

#include <iostream>

using namespace std;

//定义 Tensors 抽象类类

class Tensors

{

public:

virtual ~Tensors() {};

virtual int getSize() = 0; //纯虚函数，取数据个数

virtual int getDimension() = 0; //纯虚函数，取数据维数

double\* data = nullptr; //数据

protected:

int size = 0; //数据个数

int dimension = 0; //维数

string name = "Tensor";

};

//定义 Vector 类

class Vector :public Tensors

{

public:

Vector(const double\* initData, int size);

Vector(const Vector& v); //拷贝构造

~Vector();

int getSize();

int getDimension();

Vector& operator+(const Vector& v1);

Vector& operator=(const Vector& from);

};

//构造Vector

Vector::Vector(const double\* initData, int size)

{

double\* ptr;

this->size = size;

this->dimension = 2;

this->name = "Vector";

this->data = new double[size];

ptr = this->data;

cout << "构造Vector ：" << endl;

//初始化数据

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

\*(ptr + i) = \*(initData + i);

cout << "\*(ptr + i) =" << \*(ptr + i) << ", \*(initData + i)=" << \*(initData + i) << endl;

}

}

//拷贝构造

Vector::Vector(const Vector& v) {

size = v.size;

dimension = v.dimension;

data = new double[v.size];

name = "copy构造";

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

\*(data + i) = \*(v.data + i);

}

}

//析构Vector

Vector::~Vector()

{

cout << endl << "析构Vector ：" << this->name << ", " << this->data << endl;

delete[] this->data;

}

//+运算符重载

Vector& Vector::operator+(const Vector& v1) {

if (this->size == v1.size) {

cout << endl << " Vector()+ 函数,name:" << this->name << " | this->size: "

<< this->size << " | v1.size: " << v1.size << endl;

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

\*(this->data + i) += \*(v1.data + i);

cout << endl << " \*(this->data + i):" << \*(this->data + i) << " | \*(v1.data + i) :"

<< \*(v1.data + i) << endl;

}

}

return \*(this);

}

//=运算符重载

Vector& Vector::operator=(const Vector& from) {

Vector reVector(from.data, from.size);

return reVector;

}

int Vector::getSize() {

return size;

}

int Vector::getDimension() {

return dimension;

}

//定义 Scalar 类

class Scalar :public Tensors

{

public:

Scalar(double\* data);

~Scalar();

int getSize();

int getDimension();

Scalar& operator+(const Scalar& v1);

Scalar& operator=(const Scalar& from);

};

Scalar::Scalar(double\* initData)

{

this->data = initData;

this->size = 1;

this->dimension = 1;

}

Scalar::~Scalar() {}

int Scalar::getSize() {

return size;

}

int Scalar::getDimension() {

return dimension;

}

//定义 Matrix 类

class Matrix : public Tensors

{

public:

Matrix(double\* data, int size, int cols, int rows);

~Matrix();

int getSize();

int getDimension();

Matrix& operator+(const Matrix& m1);

Matrix& operator=(const Matrix& from);

};

Matrix::Matrix(double\* data, int size, int cols, int rows) {

//初始化Matrix ....

}

Matrix::~Matrix() {}

int Matrix::getSize() {

return size;

}

int Matrix::getDimension() {

return dimension;

}

int main() {

double dataVector1[] = { 2.0,3.0 };

int sizeVector1 = 2;

double dataVector2[] = { 4.0,5.0 };

int sizeVector2 = 2;

Vector v1(dataVector1, sizeVector1);

Vector v2(dataVector2, sizeVector2);

v1 = v1 + v2;

cout << "v1中的数据为：" << endl;

for (int i = 0; i < sizeVector1; i++) {

cout << \*(v1.data + i) << " ";

}

cin.ignore();

return 0;

}