**目 录**

**[Ch2作业 1](#_Toc11177)**

[第一题 1](#_Toc5202)

[各个指针的解释 1](#_Toc6715)

[赋值语句辨析 1](#_Toc31428)

[第二题 4](#_Toc11403)

[(1) 请解释下面语句含义 4](#_Toc24193)

[(2) 请解释下面语句含义 4](#_Toc28270)

[(3) 请解释下面语句含义 4](#_Toc6699)

[第三题 5](#_Toc23958)

[第四题 6](#_Toc29662)

[第五题 6](#_Toc12826)

[第六题 7](#_Toc19479)

[第七题 7](#_Toc14401)

**[Ch3作业](#_Toc23766)** [10](#_Toc23766)

[第1题 10](#_Toc11663)

[第2题 11](#_Toc21227)

[第3题 11](#_Toc25454)

[第4题 11](#_Toc6797)

[第5题 12](#_Toc15336)

[第6题 13](#_Toc16802)

**[Ch4作业](#_Toc31043)** [13](#_Toc31043)

[第1题 13](#_Toc9578)

[第2题 14](#_Toc24454)

[第3题 14](#_Toc3693)

[第4题 15](#_Toc24373)

[第5题 15](#_Toc13525)

[第6题 16](#_Toc31421)

[附录A MyArray.h 17](#_Toc14493)

[附录B MyArray.cpp 19](#_Toc25701)

[附录C MyArrayTest.cpp 22](#_Toc13375)

**[Ch5作业](#_Toc204)** [26](#_Toc204)

[5.1 26](#_Toc1476)

[5.2 27](#_Toc6704)

**[Ch6-8作业](#_Toc2403)** [27](#_Toc2403)

[第1题 27](#_Toc13075)

[第2题 29](#_Toc28695)

[第3题 30](#_Toc13162)

[第4题 31](#_Toc28056)

[第5题 32](#_Toc19787)

[第6题 34](#_Toc29553)

[第7题 35](#_Toc8564)

**[Ch10作业](#_Toc9595)** [35](#_Toc9595)

**[Ch11作业](#_Toc24803)** [37](#_Toc24803)

[11.1 37](#_Toc17186)

[11.2 38](#_Toc24115)

# Ch2作业

## 第一题

1. char c = '1', \*pc = 0;
2. const char cc = 'a';
3. const char\* pcc = 0;
4. char\* const cpc = &c;
5. const char\* const cpcc = &cc;
6. char\* const \*pcpc;
7. char\* const\* const pccp = &cpc;

### 各个指针的解释

1.char c = '1', \* pc = 0;

// c 是一个char类型的变量，初始化为 '1'。

// pc 是一个指向字符类型的指针，初始化为0。

2.const char cc = 'a';

// cc 是一个 const char类型的变量，初始化为 'a'。

3.const char\* pcc = 0;

// pcc 是一个指向 const char类型的指针，初始化为0。pcc 所指向的**变量不能修改**，但指针可以改变。

4.char\* const cpc = &c;

// cpc 是一个 const字符指针，指向字符变量 c。**指针不可修改**，但它指向的内容可以改变。

5.const char\* const cpcc = &cc;

// cpcc 是一个指向 const 字符的 const 指针。它指向字符 cc，既不能修改指针本身，也不能修改它所指向的内容。

6.char\* const\* pcpc;

// pcpc 是一个**指向** **char\* const** **指针的指针**。这个指针可以指向其他指针，但该指针指向的指着指针不能修改，该指针指向的指针指向的内容可以修改

7.char\* const\* const pccp = &cpc;

// pccp 是一个 const（不可修改的）指向 const 字符指针的指针，指向指针 cpc。既不能修改 pccp 本身，也不能修改它指向的指针 cpc。

### 赋值语句辨析

**c = cc;** //1

**正确**，cc是const char常量字符，可以赋值给普通字符类型c

cc = c; //2编译报错

编译报错cc是const char类型，不可修改

**pcc = &c;** //3

**正确**,pcc是const char\*指针，可以指向c，但不能通过\*pcc进行修改

**pcc = &cc;** //4

**正确**，指针和变量类型对应

**pc = &c;** //5

**正确**，指针和变量类型对应

pc = &cc; //6编译报错

编译报错,**只读单元的指针(地址)不能赋给指向可写单元的指针变量**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误(活动) | E0513 | 不能将 "const char \*" 类型的值分配到 "char \*" 类型的实体 | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 23 |  |  |

pc = pcc; //7编译报错

编译报错,只读单元的指针(地址)不能赋给指向可写单元的指针变量

pc = cpcc; //8编译报错

编译报错,只读单元的指针(地址)不能赋给指向可写单元的指针变量

cpcc = pc; //9编译报错

编译报错,cpcc是const指针，不可修改指向内容

**\*pc = "ABCD"[2];** //10

正确，pc是普通指针，通过\*pc将字符串"ABCD"的第三个字符赋值给c

cc = a; //11编译报错

编译报错,cc是const char类型，不可修改

\*cpcc = \*pc; //12编译报错

编译报错,cpcc 是指向 const char 的指针，不能修改。

**pc = \*pcpc;** //13运行报错

**运行报错**,pcpc未初始化

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误 | C4700 | 使用了未初始化的局部变量“pcpc” | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 30 |  |  |

**\*\*pcpc = \*pc;** //14运行报错

**运行报错**,pcpc未初始化

**\*pc = \*\*pcpc;** //15运行报错

**运行报错**,pcpc未初始化

\*pcc = 'b'; //16编译报错

编译报错,pcc 是指向 const char 的指针，不能修改。

\*pcpc = 'c'; //17编译报错

编译报错,pcpc是指向 char\*const 的指针，char\*const指针自身不能修改。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误(活动) | E0137 | 表达式必须是可修改的左值 | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 34 |  |  |

\*cpcc = 'd'; //18编译报错

编译报错,cpcc 指向 const char内容，不能修改其指向的内容。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误(活动) | E0137 | 表达式必须是可修改的左值 | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 36 |  |  |

\*pcpc = pc; //19编译报错

编译报错,\*pcpc是char\*const指针，指针本身不能修改

**pcpc = &cpc;** //20

**正确**,cpc是char\* const指针，其地址可以赋值给pcpc

pccp = &cpc; //21编译报错

编译报错,pccp是const指针，不是可修改的左值

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误(活动) | E0137 | 表达式必须是可修改的左值 | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 38 |  |  |

故运行正确语句为1、3、4、5、10、20

编译报错为2、6、7、8、9、11、12、16、17、18、19、21

运行报错为13、14、15

## 第二题

请解释语句含义，下面是一个示例：

char (\*pf)(int);

解释：该语句声明一个函数指针pf，指向这样一个函数：参数为int，返回char

可以用typeid操作符打印一个标识符或表达式求值结果的类型，使用该操作符时，必须#include <typeinfo>

std::cout << typeid(pf).name() << std:endl;

### 请解释下面语句含义

1. int\* (\*p)[4];

该语句声明一个指针p，指向一个大小为4的数组，数组的每个元素是一个指向int的指针

### 请解释下面语句含义

1. int\* (\*t)(int) = 0;
2. typedef int\* (\*F)(int);
3. F f = t;

第一条语句定义一个指针t，指向这样一个函数:参数为int，返回int\*

第二条语句定义了一个函数指针类型F，函数类型同上

第三条语句声明一个F类型（函数指针）的变量f，并初始化为t（类型对应）

### 请解释下面语句含义

1. int (\*(\*g)[10])(int, int) = 0;
2. typedef int (\*G[10])(int, int);
3. G\* pg = g;

第一条语句定义一个指针g，指向一个大小为10的数组，数组的每个元素是一个函数指针，指向这样一个函数：参数是两个int，返回值为int

第二条语句定义了一个数组G，G的元素是一个函数指针，函数类型同上

第三天语句定义一个G\*类型的变量pg（即指向G类型的指针），并初始化为g（类型对应）

## 第三题

如果给出下面的定义

1. int i = 1024;
2. const int ci = 1024;
3. double d = 3.14;
4. int f(int x) { return x; }
5. int& g(int& x) { return x; }

则下面的语句哪些编译成功，哪些会编译报错，请说明原因。

1. int& ri\_1 = i; *//1*

正确，ri\_1是int类型的引用，i是int类型的变量

1. int& ri\_2 = ci; *//2*

**错误**，普通引用不能绑定const对象

1. int& ri\_3 = d; *//3*

**错误**，除了二种例外情况，引用类型都要与绑定的对象严格匹配

1. int& ri\_4 = i \* 2; *//4*

**错误**，普通引用只能绑定到左值

1. int& ri\_5; *//5*

**错误**，引用必须在定义时初始化

1. int& ri\_6 = i++; *//6*

**错误**，普通引用只能绑定到左值

1. int& ri\_7 = --i; *//7*

正确，--i是左值，额可以被ri\_7绑定

1. int& ri\_8 = f(i); *//8*

**错误**，普通引用只能绑定到左值

1. int& ri\_9 = g(i); *//9*

正确，g返回的是i的引用，ri\_9可以绑定到左值

1. const int& cri\_1 = i; *//10*

正确，const引用可以绑定非const左值

1. const int& cri\_2 = 1; *//11*

正确，const引用可以绑定右值

1. const int& cri\_3 = d; *//12*

正确，const引用可以绑定不同类型变量

1. const int& cri\_4 = 3.14; *//13*

正确，const引用可以绑定不同类型右值

1. const int& cri\_5 = f(); *//14*

**错误**，函数缺少参数

1. int& & rri\_1 = 1; *//15*

错误，不能声明引用的引用

1. int&\* pri; *//16*

**错误**，不允许使用指向引用的指针

1. int&& rri\_2 = 1; *//17*
2. int&& rri\_3 = i; *//18*

**错误**，右值引用不能绑定左值

1. int&& rri\_4 = f(i); *//19*

正确，右值引用绑定到右值，函数f返回值是右值

1. int&& rii\_5 = rri\_4; *//20*

**错误**，rri\_4 是一个右值引用，但它本身是一个左值,右值引用不能绑定左值

1. int& ri\_10 = rri\_4; *//21*

正确，普通引用绑定左值rri\_4

故报错语句为2、3、4、5、6、8、14、16、18、20

## 第四题

如果给出下面的定义

1. int a[3] = { 1,2,3 };
2. void f1(int \*p){}
3. void f2(int\*& pr);
4. void f3(int \* const &cpr){}
5. void f4(int(&ra)[3]) {}

请问下面test函数里四个调用语句是否成立，请说明原因

1. void test() {
2. f1(a); *//1*
3. f2(a); *//2*
4. f3(a); *//3*
5. f4(a); *//4*
6. }
7. 成立，数组名a等价为首元素地址即int\*类型，可以被f1接受
8. **不成立**，数组名a被视作常量指针，普通引用不能绑定const类型
9. 成立，与f2相比，改为const引用便能绑定a了
10. 成立，函数接受的参数为int[3]类型的引用，与a的类型匹配

## 第五题

如果给出下面的定义

1. int& f(int &x){
2. for (int y = 1; y != 1 && x < 50; x += 13, y++) if (x > 49) { x -= 31;
3. y -= 5; }
4. return x -= 20;
5. }
6. void test() {
7. int i = 40;
8. f(i) = 1;
9. }

则当执行完test()函数后，i的值是多少？请说明原因。

由于函数返回的是引用，f(i)=1等价于i=1,所以最后i为1

## 第六题

如果给出下面的定义

1. void swap(int x, int y) {
2. *//交互xy的值*
3. int temp = x;
4. x = y;
5. y = temp;
6. }

那么下面的函数test()执行完后，实参i和j的值是否互换？请解释原因。

1. void test() {
2. int i = 0, j = 1;
3. swap(i, j);
4. }

不会，函数调用是值传递的，形参的变化不会影响实参的值。正确做法如下

1. void swap(int &x, int &y) {
2. int temp = x;
3. x = y;
4. y = temp;
5. }

## 第七题

请实现如下函数创建如下的不规则数组，并打印在屏幕上



采用自定义的简单vector类实现动态数组

源码

1. #include"vector.hpp"
2. void matrix(int size) {
4. vector<vector<int>> matrix(size);
5. for (int i = 0; i < size; i++) {
6. matrix[i] = vector<int>(size - i, 0);
7. }
8. *// 打印数组*
9. for (int i = 0; i < size; ++i) {
10. for (int j = 0; j < matrix[i].size(); ++j) {
11. cout << matrix[i][j] << " ";
12. }
13. cout << endl;
14. }
15. }
16. int main() {
17. matrix(7);
18. return 0;
19. }
20. vector.hpp
21. #include<iostream>
22. template <class T>
23. class vector {
24. public:
25. *// 数据*
26. T\* data;
27. *// 大小*
28. int Size;
29. *// 容量*
30. int capacity;
31. *// 构造函数*
32. vector() : Size(0), capacity(1), data(new T[1]) {}
33. *// 析构函数*
34. ~vector() {
35. delete[] data;
36. }
37. *//含参构造函数(初始化大小,元素)*
38. vector(int n, T t) : Size(n), capacity(n), data(new T[n]) {
39. for (int i = 0; i < n; i++) {
40. data[i] = t;
41. }
42. }
43. *// 拷贝构造函数*
44. vector(const vector& v) : Size(v.Size), capacity(v.capacity), data(new T[v.capacity]) {
45. std::copy(v.data, v.data + v.Size, data);
46. }
47. *// 赋值运算符*
48. vector& operator=(const vector& v) {
49. if (this != &v) {
50. T\* new\_data = new T[v.capacity];
51. std::copy(v.data, v.data + v.Size, new\_data);
52. delete[] data;
53. data = new\_data;
54. Size = v.Size;
55. capacity = v.capacity;
56. }
57. return \*this;
58. }
59. *// 添加元素*
60. void push\_back(T t) {
61. if (Size == capacity) {
62. resize(capacity \* 2);
63. }
64. data[Size++] = t; *// 添加元素*
65. }
66. *// 删除元素*
67. void pop\_back() *// 删除最后一个元素*
68. {
69. if (Size > 0) Size--;
70. }
71. *// 返回大小*
72. int size() const {
73. return Size;
74. }
75. *// 返回元素*
76. T& operator[](int index) {
77. if (index < 0 || index >= Size) {
78. throw std::out\_of\_range("Index out of range");*//抛出异常*
79. }
80. return data[index];
81. }
82. *//判断是否为空*
83. bool empty() const
84. {
85. return Size == 0;
86. }
87. void clear() {
88. Size = 0;
89. }
90. *// 调整容量*
91. void resize(int new\_capacity) {
92. T\* new\_data = new T[new\_capacity];*//新建一个数组*
93. std::copy(data, data + Size, new\_data);*//复制数据*
94. delete[] data;*//删除原来的数组*
95. data = new\_data;
96. capacity = new\_capacity;
97. }
98. };

# Ch3作业

## 第1题

如下函代码是否有编译错误？如果有请说明原因

float g(int i ) {}

int g(int j) {}

答：编译错误，无法仅按返回类型区分不同函数，两者参数列表相同

## 第2题

如下函代码是否有编译错误？如果有请说明原因

int f(int x) {}

int f(int x, int y = 0){}

int f(int x, ...) { }

int test2(){ f(1,2,3); }

答：由于函数体为空，没有返回值，所以会编译报错。

如果假定函数体存在合适内容，那么不会报错，程序可以正确调用第三个函数f，因为传递了三个参数。如果只传递一个参数，才会编译错误（无法区分第一个f和第二个f）  
假定函数体内容如下

## 第3题

如下函代码是否有编译错误？如果有请说明原因

int f(int x) {}

int f(int x, int y = 0) {}

int f(int x, ...) { }

int test3() { f(1); }

答：报错，无法区分第一个f和第二个f，对重载函数的调用不明确

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误 | C2668 | “f”: 对重载函数的调用不明确 | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 9 |  |  |

## 第4题

函数的get的定义能否让函数test4编译通过并正常运行？如果不行请说明原因，并修改get函数使得 test4能编译通过并正常运行

int get(int\* p, int index)

{

return p[index];

}

void test4()

{

int a[10];

for (int i = 0; i < 10; i++)

{

get(a, i) = i;

}

}

答：不能正常运行。Get函数的返回的是右值，不可进行赋值操作。修改返回为引用即可  
int& get(int\* p, int index)

{

return p[index];

}

## 第5题

给定下面二个重载的函数定义

void f(int\* p) { std::cout << "void f(int \* p)" << std::endl; }

void f(const int \*p){ std::cout << "void f(const int \* p)" << std::endl; }

下面的语句1和语句2分别调用了哪个f

void test5()

{

int a[2] = { 1,2 };

const int b[2] = { 1,2 };

f(a); f(b);

}

答：由于数组名等价于指向第一个元素的指针

所以语句1对应int\*，语句2对应const int \*

· 语句 1：f(a) 调用的是 void f(int\* p)。

· 语句 2：f(b) 调用的是 void f(const int\* p)。

## **第6题**

下面的代码是否可以编译通过？如果不行请说明原因

void print(const char\* p) {}

void print(int i) {}

void test6()

{

void print(int);

print("Hello");

}

答：不通过。由于在函数体内声明print(int)，根据作用域的优先级，print语句将会调用内部参数为int的print，而"Hello" 是const char\*类型，不匹配

# Ch4作业

## 第1题

下面代码中对象a1和a2的数据成员i值分别是多少

class A {

public:

int i;

} a1;

void test1() {

static A a2;

std::cout << "a1.i = " << a1.i << std::endl;

std::cout << "a2.i = " << a2.i << std::endl;

}

如果在test1里定义A a3，是否可以打印出a3.i ？

答：a1和a2的i均为0，由于a1是全局变量，a2是静态变量，其成员变量会初始化为0；

如果定义普通对象a3，由于未初始化其i的值，将会报错（VS2022)

在VsCode环境下，a3.i也被初始化为0

在其他环境下，也有可能输出未定义的值（随机数据）

## **第2题**

下面的代码中语句1和语句2编译是否报错？请说明原因

class A

{

int x;

public:

A(int x) :x(x) {}

};

A a;

A b(1);

答：语句1会报错，因为没有对应的无参构造函数。

语句2不会报错，因为有对应的带参数的构造函数。

## **第3题**

给定A类的定义

class A {

public :int x;

public:

A(int x) :x(x) {}

};

请为class B添加合适的构造函数，使得下面的语句能够编译通过

class B {

const A a;

const A& ra;

public:

//构造函数

} b;

答：B():a(0),ra(a){}

由于在定义 class B 时，后面直接实例化了一个对象 b，并且没有提供参数，因此 B 需要一个无参的默认构造函数。同时，由于 B 包含了 const A a 和 const A& ra，我们必须确保在默认构造函数中初始化这两个成员。因为 const 成员和引用成员一旦定义，就必须立即被初始化，不能留空。

## **第4题**

给定下面类的定义

class A {

int x;

public:

A(int x) :x(x) {}

};

class B {

const int i = 0;

};

class E包含如下数据成员

class E {

A a;

B b;

//则下面class E的构造函数哪些是正确的,哪些是错误的？请说明原因

public:

E() :a(1) {} //构造函数1

E() :b() {} //构造函数2

E() :a(1), b() {} //构造函数3

};

答：13正确，2错误。

1. **正确**。因为 a 被初始化为 1，b 的 i 被默认初始化为 0，符合要求。
2. **错误**，因为 a 没有被初始化，而a又没有无参构造函数
3. **正确**。ab都正确初始化

## **第5题**

下面代码存在哪些编译错误，请说明原因

class A {

A() {}

} a;

class B {

int i;

int j;

public:

explicit B(int x, int y = 0) :i(x), j(y) {}

};

B b1;

b2 = 1;

答：

1. **A的构造函数为私有**，在外部创建对象时无法调用构造函数

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误 | C2248 | “A::A”: 无法访问 private 成员(在“A”类中声明) | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 8 |  |  |

1. B**没有提供无参构造函数**，B b1将会报错

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误 | C2512 | “B”: 没有合适的默认构造函数可用 | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 15 |  |  |

1. B 类的构造函数被声明为 **explicit**，这意味着它禁止隐式转换,1无法转换为B类型

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 严重性 | 代码 | 说明 | 项目 | 文件 | 行 | 禁止显示状态 | 详细信息 |
| 错误 | C2440 | “初始化”: 无法从“int”转换为“B” | Project1 | C:\Users\muke\source\repos\Project1\源.cpp | 18 |  |  |

## **第6题**

编程题

一维整型数组MyArray的定义如下，请实现相应的函数成员

源代码见附录

## 附录A MyArray.h

1. #pragma once
2. #include <string>
3. class MyArray
4. {
5. private:
6. int size = 0;           *// 数组大小*
7. int \*const p = nullptr; *// 指向动态分配的内存，保存数组的内容*
8. public:
9. MyArray(int size = 10); *// 构造函数，参数size指定数组大小*
10. MyArray(const MyArray &old); *// 拷贝构造函数，要求实现深拷贝*
11. MyArray &operator=(const MyArray &rhs); *// 重载=，要求实现深拷贝*
12. MyArray(MyArray &&old) noexcept; *// 移动拷贝*
13. MyArray &operator=(MyArray &&rhs) noexcept; *// 移动=*
14. ~MyArray(); *// 析构函数，要求能防止反复释放资源*
15. int length(); *// 返回数组大小*
16. int &get(int index); *// 返回下标为index的元素，不考虑越界情况*
17. *// 以下方法测试用*
18. *// 一个对象是否为空。如果size或p有一个为0，则返回true*
19. bool isEmpty() const;
20. *// 比较二个MyArray对象是否相等。当二个MyArray对象都不是Empty，size一样，数组的内容完全一样时，这二个MyArray对象才相等*
21. bool equals(const MyArray &other) const;
22. *// 将MyArray数组内容变成字符串。要求数组元素之间用空格分开，形如这样的格式"0 1 2 3 4 5"*
23. *// 如果MyArray对象是Empty，则返回string对象的内容为""*
24. std::string toString();
25. *// 返回内部p指针，仅仅用于测试，不能用于任何其他地方*
26. const int \*const getP();

## 附录B MyArray.cpp

1. #include "..\..\include\ch4\_homework\MyArray.h"
2. #include <iostream>
3. *// 构造函数*
4. MyArray::MyArray(int size) : size(size), p(new int[size])
5. {
6. for (int i = 0; i < size; ++i)
7. {
8. p[i] = i + 1;
9. }
10. }
11. *// 拷贝构造函数（深拷贝）*
12. MyArray::MyArray(const MyArray &old) : size(old.size), p(new int[old.size])
13. {
14. for (int i = 0; i < size; ++i)
15. {
16. p[i] = old.p[i];
17. }
18. }
19. *// 移动构造函数*
20. MyArray::MyArray(MyArray &&old) noexcept : size(old.size), p(old.p)
21. {
22. \*(const\_cast<int \*\*>(&old.p)) = nullptr; *// 防止旧对象释放资源*
23. old.size = 0;
24. }
25. *// 拷贝赋值运算符（深拷贝）*
26. MyArray &MyArray::operator=(const MyArray &rhs)
27. {
28. if (this == &rhs)
29. {
30. return \*this; *// 防止自赋值*
31. }
32. if (size != rhs.size)
33. {
34. delete[] p;                                    *// 释放旧资源*
35. \*(const\_cast<int \*\*>(&p)) = new int[rhs.size]; *// 分配新内存*
36. size = rhs.size;
37. }
38. for (int i = 0; i < size; ++i)
39. {
40. p[i] = rhs.p[i];
41. }
42. return \*this;
43. }
44. *// 移动赋值运算符*
45. MyArray &MyArray::operator=(MyArray &&rhs) noexcept
46. {
47. if (this != &rhs)
48. {
49. delete[] p;
50. \*(const\_cast<int \*\*>(&p)) = rhs.p;
51. size = rhs.size;
52. \*(const\_cast<int \*\*>(&rhs.p)) = nullptr;
53. rhs.size = 0;
54. }
55. return \*this;
56. }
57. *// 析构函数*
58. MyArray::~MyArray()
59. {
60. if(p == nullptr)
61. {
62. size=0;
63. return;
64. }
65. else
66. {
67. delete[] p;
68. \*(const\_cast<int \*\*>(&p)) = nullptr;
69. size = 0;
70. }
71. }
72. *// 返回数组大小*
73. int MyArray::length()
74. {
75. return size;
76. }
77. *// 返回下标为index的元素*
78. int &MyArray::get(int index)
79. {
80. return p[index];
81. }
82. *// 判断数组是否为空*
83. bool MyArray::isEmpty() const
84. {
85. return size == 0 || p == nullptr;
86. }
87. *// 比较两个MyArray对象是否相等*
88. bool MyArray::equals(const MyArray &other) const
89. {
90. if (size != other.size || isEmpty() || other.isEmpty())
91. {
92. return false;
93. }
94. for (int i = 0; i < size; ++i)
95. {
96. if (p[i] != other.p[i])
97. {
98. return false;
99. }
100. }
101. return true;
102. }
103. *// 将MyArray数组内容转换为字符串*
104. std::string MyArray::toString()
105. {
106. if (isEmpty())
107. {
108. return "";
109. }
110. std::string result;
111. for (int i = 0; i < size; ++i)
112. {
113. result += std::to\_string(p[i]);
114. if (i < size - 1)
115. result += " ";
116. }
117. return result;
118. }
119. *// 返回内部指针p，仅用于测试*
120. const int \*const MyArray::getP()
121. {
122. return p;
123. }

## 附录C MyArrayTest.cpp

1. #include <gtest/gtest.h>
2. #include "../include/ch4\_homework/MyArray.h"
3. #include <iostream>
4. using namespace std;
5. */\**
6. 在实现好MyArray类基础上，编写下面的代码来测试验证MyArray的功能
7. \*/
8. */\**
9. 请构建构建大小为size的MyArray对象a，要求元素的值分别为1 2 3 4 ... size
10. 并返回该对象 ，用于测试
11. \*/
12. MyArray buildArrayInstace(int size)
13. {
14. MyArray a(size);
15. return a;
16. }
17. */\**
18. 测试MyArray的实例化以及toString函数
19. \*/
20. TEST(TEST\_MYARRY, MyArray\_Test\_instantiate\_toString)
21. {
22. *// given*
23. MyArray a = buildArrayInstace(5);
24. *// when*
25. *// 得到对象a的字符串表示*
26. std::string content = a.toString();
27. *// then*
28. std::string target("1 2 3 4 5");
29. ASSERT\_EQ(target, content);
30. }
31. */\**
32. 测试MyArray的移动构造函数
33. \*/
34. TEST(TEST\_MYARRY, MyArray\_Test\_MoveConstructor)
35. {
36. *// given*
37. MyArray a = buildArrayInstace(10);
38. const int \*const pa = a.getP(); *// 保留a的内部指针*
39. *// when*
40. MyArray b(std::move(a));
41. *// 检测内容是否相等*
42. std::string target("1 2 3 4 5 6 7 8 9 10");
43. *// then*
44. ASSERT\_EQ(target, b.toString());
45. *// 检查是否为浅拷贝*
46. *// then*
47. ASSERT\_EQ(pa, b.getP());
48. *// 检查a对象是不是为可以安全析构*
49. *// then*
50. ASSERT\_EQ(0, a.length());
51. ASSERT\_EQ(nullptr, a.getP());
52. }
53. */\**
54. 测试MyArray的move assignment（移动=）
55. \*/
56. TEST(TEST\_MYARRY, MyArray\_Test\_MoveAssignment)
57. {
58. *// given*
59. MyArray a = buildArrayInstace(5);
60. MyArray b = buildArrayInstace(10);
61. const int \*const pa = a.getP(); *// 保留a的内部指针*
62. *// when*
63. b = std::move(a);
64. *// 检测内容是否相等*
65. std::string target("1 2 3 4 5");
66. *// then*
67. ASSERT\_EQ(target, b.toString());
68. *// 检查是否为浅拷贝*
69. *// then*
70. ASSERT\_EQ(pa, b.getP());
71. *// 检查a对象是不是为可以安全析构*
72. *// then*
73. ASSERT\_EQ(0, a.length());
74. ASSERT\_EQ(nullptr, a.getP());
75. }
76. */\**
77. 测试MyArray的析构函数
78. \*/
79. TEST(TEST\_MYARRY, MyArray\_Test\_Deconstructor)
80. {
81. *// given*
82. MyArray a = buildArrayInstace(5);
83. int rtn = 0;
84. *// when*
85. a.~MyArray();
86. a.~MyArray();
87. rtn = 1; *// 如果析构函数实现正确，会执行到rtn = 1;*
88. *// then*
89. ASSERT\_EQ(0, a.length());
90. ASSERT\_EQ(nullptr, a.getP());
91. ASSERT\_EQ(1, rtn);
92. }
93. */\**
94. 测试MyArray的拷贝构造函数
95. \*/
96. TEST(TEST\_MYARRY, MyArray\_Test\_CopyConstructor)
97. {
98. *// given*
99. MyArray a = buildArrayInstace(10);
100. *// 检测内容是否相等*
101. *// when*
102. MyArray b(a);
103. *// then*
104. ASSERT\_EQ(true, a.equals(b));
105. *// 检查是否为深拷贝*
106. *// when*
107. a.~MyArray(); *// 析构对象a*
108. *// then*
109. std::string target("1 2 3 4 5 6 7 8 9 10");
110. ASSERT\_EQ(target, b.toString()); *// 这时对象b的内容应该还在，深拷贝*
111. }
112. */\**
113. 测试MyArray的copy assignment（拷贝=）
114. \*/
115. TEST(TEST\_MYARRY, MyArray\_Test\_CopyAssignment)
116. {
117. *// given*
118. MyArray a = buildArrayInstace(10);
119. MyArray b = buildArrayInstace(5);
120. *// 检测内容是否相等*
121. *// when*
122. a = b;
123. *// then*
124. ASSERT\_EQ(true, a.equals(b));
125. *// 检查是否为深拷贝*
126. *// when*
127. b.~MyArray();
128. *// then*
129. std::string target("1 2 3 4 5");
130. ASSERT\_EQ(target, a.toString());
131. }

# Ch5作业

## 5.1

基于以下定义，在类外初始化五个静态成员

1. struct A
2. {
3. static int \*j;
4. static int A::\*a;
5. static int i[5];
6. int x;
7. static int &k;
8. static int \*n;
9. };

答：

int A::i[5] = { 0 };//初始化数组

int A::\* A::a = &A::x;//指向类的成员变量

int\* A::j = &A::i[0];//指向类的静态成员变量

int& A::k = A::i[0]; //引用类的静态成员变量

int\* A::n = &A::i[0]; //指向类的静态成员变量

测试：

int main()

{

A a;

//j的值(地址)是i[0]的地址

cout << a.j << endl;

//偏移量

cout << a.a << endl;

//x的地址

cout << &a.x << endl;

A b;

//j的值(地址)是i[0]的地址

cout << b.j << endl;

// 偏移量

cout << b.a << endl;

cout << &b.x << endl;//与a.x的地址不同

return 0;

}

## 5.2

指出错误说明原因

1. struct A {
2. int i;
3. static const int j = 0;
4. static const double d = 0.0;
5. static void f(A a) {
6. int x = a.i;
7. int y = i + a.i;
8. }
9. };

答：

static const double d = 0.0;

错误，**静态常量整数成员**才可以直接初始化

int y = i + a.i;

错误，静态成员函数不依赖于对象示例，不能访问与实例相关的成员i

# Ch6-8作业

## 第1题

1. */\**
2. 1:对于如下类型声明，分别给出类A、B、C可访问的成员及其访问权限\*/
3. class A {
4. int a1, a2;
5. protected:
6. int a3, a4;
7. public:
8. int a5, a6;
9. };
10. class B : A {
11. int b1, b2;
12. protected:
13. using A::a3;
14. int b3, b4;
15. public:
16. using A::a6;
17. int b5, b6;
18. ;
19. };
20. struct C : public B {
21. int c1, c2;
22. using B::a3;
23. protected:
24. int c3, c4;
25. public:
26. int c5, c6;
27. int b5, b6, a6;
28. };

## 答：

### 一、类 A

1.私有成员 ：a1, a2//仅类 A 的成员函数可以访问。

2.受保护成员 ：a3, a4//类 A 及其派生类可以访问。

3.公有成员 ：a5, a6//类 A、派生类以及外部代码都可以访问

### 二、类 B

类 B 继承自 A，默认是**私有继承**

#### 2.1 继承后 A 的成员在 B 中的访问权限：

1. A 的**私有成员** (a1, a2)：

不可见

2.A 的**保护成员** (a3, a4)：

在私有继承的情况下，本来应该变为私有成员，但在类 B 中**显式地 using A::a3** 将 a3 的访问权限提升为protect，所以：

a3 在类 B 中为protect。

a4 在类 B 中仍为private。

3.A 的 **公有成员** (a5, a6)：

在私有继承的情况下，变为 私有成员，但在类 B 中**显式地 using A::a6** 将 a6 的访问权限提升为public，所以：

a5 在类 B 中为private。

a6 在类 B 中为public。

#### 2.2 类 B 自己的成员：

1.私有成员 ：b1, b2

2.保护成员：b3, b4

3.公有成员 (public)：b5, b6

访问权限描述同类A，不再重述

### 三、类 C

类 C 使用 **public 继承** B，因此 B 的公有和受保护成员在 C 中保持原有访问权限。

#### 3.1 继承后 B 的成员在 C 中的访问权限：

1.B 的 私有成员 (b1, b2)：不可见

2.B 的 受保护成员 (a3, b3, b4)：在 C 中仍为保护成员。

3.B 的 公有成员 (a6, b5, b6)：在 C 中仍为 公有成员**(C中重新声明了它们，默认访问C的成员，如果需要访问继承来的需要加B::限定**

#### 3.2 类 C 自己的成员：

1.私有成员 (private)：无（**结构体的默认是 public**）。

2.受保护成员 (protected)：c3, c4

3.公有成员 (public)：c1, c2, c5, c6,b5,b6,a6

## 第2题

1. */\**
2. 2: 根据以下代码，回答问题或完成类的定义
3. \*/
4. class A {
5. int i;
6. public:
7. A(int i);
8. void print();
9. };
10. */\**
11. 2.1 请在class A的类体外实现构造函数A(int i)，要求必须在成员初始化列表初始化i成员
12. \*/
13. class B : public A {
14. int j;
15. public:
16. B(int i, int j);
17. };
18. */\**
19. 2.2 请在class B的类体外实现构造函数B(int i,int j)
20. 要求必须在成员初始化列表初始化j成员
21. \*/

答：

A::A(int i) : i(i) {}

B::B(int i, int j) : A(i), j(j) {}

## 第3题

1. */\**
2. 3: 根据以下代码，回答问题或完成类的定义
3. \*/
4. class A {
5. protected:
6. int i{ 0 };
7. public:
8. A(int i = 0) {}
9. };
10. class B : public A {
11. int i;
12. public:
13. */\**
14. 3.1 B类构造函数没有显式构造基类A对象，编译是否可以通过？请说明原因
15. \*/
16. B(int i) { this->i = i; }
17. int sum(int i) {
18. */\**
19. 3.2 请完成sum函数的具体实现，要求对局部变量i、A类继承的i、B类的i求和并返回
20. \*/
21. }
22. };

答：(1)可以通过，A中i有默认值0,即使没有定义构造函数也可以编译通过。

(2) int sum(int i) {

return i + A::i + this->i;

}

其中i为传给sum的i，this->i为B类的i

## 第4题

1. */\**
2. 4: 根据以下代码，回答问题
3. \*/
4. class A {
5. int i;
6. public:
7. A(int x) :i(x) { }
8. };
9. class B {
10. int j;
11. public:
12. B() :j(0) {}
13. B(int x) :j(x) { }
14. };
15. class C : public A {
16. const int k;
17. A a;
18. B b;
19. A& ra;
20. public:
21. */\**
22. 下面类C的构造函数中正确的有哪些？错误的有哪些？错误的请说明原因
23. \*/
24. C(int v) :A(v), k(v), a(v), ra(a) {}          *//1*
25. C(int v):A(v),a(v),ra(a){ k = v;}             *//2*
26. C(int v):A(v),k(v),a(v),b(v),ra(a){}         *//3*
27. C(int v):k(v),a(v),b(v),ra(a){}               *//4*
28. };

答：1，3正确。2，4错误

2错在const 成员必须通过初始化列表中初始化，不能在构造函数体内部进行赋值。

4错在C继承A，但A没有默认构造函数 ，因此必须在初始化列表中显式调用A的构造函数

## 第5题

1. */\**
2. 5:下面程序，写出指定语句的输出结果，并解释原因。
3. \*/
4. class A {
5. public:
6. virtual void f() { std::cout << "A::f()" << std::endl; }
7. virtual void f(double x) { std::cout << "A::f(double)" << std::endl; }
8. static void g() { std::cout << "A::g()" << std::endl; }
9. A() = default;
10. virtual ~A() = default;
11. };
12. class B : public A {
13. public:
14. virtual void f() { std::cout << "B::f()" << std::endl; }
15. virtual void f(double x) { std::cout << "B::f(double)" << std::endl; }
16. static void g() { std::cout << "B::g()" << std::endl; }
17. static void g(int) { std::cout << "B::g(int)" << std::endl; }
18. B() = default;
19. virtual ~B() = default;
20. };
21. */\**
22. 请说明当执行下面的test函数后，
23. 每一条语句的情况，如果可以运行请给出运行结果；如果编译出错请说明原因。
24. \*/
25. void test1(A& o) {
26. o.f();        *//语句1*
27. o.f(1.0);     *//语句2*
28. o.g();        *//语句3*
29. *// o.g(1);       //语句4*
30. *// ((B)o).f();   //语句5*
31. *// ((B)o).g();   //语句6*
32. }
33. void test2(A& o) {
34. o.f(1.0f);    *//语句7*
35. o.f(1.0);     *//语句8*
36. *//  o.g(1);       //语句9*
37. o.g();        *//语句10*
38. ((A)o).g();   *//语句11*
39. }
40. void test() {
41. B b;
42. test1(b);
43. test2(b);
44. }

答：

1. 输出B::f()

f是虚函数，根据传入的对象类型，调用了B中的f

1. 输出B::f(double)

原因同上，根据函数重载，传入double参数，调用另一个f

1. 输出A::g()

由于 g() 是静态成员函数，它的调用不依赖于对象的动态类型（运行时类型），而是依赖于调用点处的静态类型.在 test1 函数中，参数 o 的静态类型是 A&，即使它实际上引用的是一个 B 类型的对象。因此，当调用 o.g(); 时，编译器会查找类型 A 中的静态成员函数 g()

1. **报错**，原因同(3)，编译时g为A::g()，而A中的g()是无参数的。
2. **报错**，没有定义A到B的转换 no known conversion for argument 1 from 'A' to 'const B&'
3. **报错**，没有定义A到B的转换 no known conversion for argument 1 from 'A' to 'const B&'
4. B::f(double)

动态绑定，具体原因如上。1.0f自动转换为double类型

1. B::f(double)

动态绑定，具体原因如上

1. **报错**，具体原因如(4)
2. 输出A::g()

静态绑定，具体原因如(3)

1. 输出A::g()

静态绑定，具体原因如(3),强制转换无意义

## 第6题

1. */\**
2. 6: 下面是类A、B、C的定义
3. \*/
4. class A {
5. public:
6. virtual void fa() {}
7. virtual void fb() = 0;
8. virtual void fc() = 0;
9. A() = default;
10. virtual ~A() = default;
11. };
12. class B : public A {
13. public:
14. virtual void fb() {}
15. };
16. class C : public B {
17. public:
18. */\**
19. 6-1: 如果C想成为一个具体类，则C必须要实现的方法是什么
20. \*/
21. };
22. */\**
23. 6-2: 请指出下面代码有错误的地方，并说明原因:
24. \*/
25. void f(A& o) {}
26. A& f(A\* p) { return \*p; }
27. void f(A o) {}

答：

(1)**必须实现纯虚函数fc**

因为A中有两个纯虚函数fb,fc,子类B已经实现了一个，C继承B，必须实现剩下的

(2)

**第三行错误**，抽象类A不能创造实例，只能通过引用或指针的形式传参

## 第7题

1. */\**
2. 7: 请给出下面程序的输出结果
3. \*/
4. class A {
5. public:
6. virtual void draw() { std::cout << "Draw A" << std::endl; }
7. virtual void display() { draw(); std::cout << "Display A" << std::endl; }
8. A() = default;
9. virtual ~A() = default;
10. };
11. class B : public A {
12. public:
13. virtual void draw() { std::cout << "Draw B" << std::endl; }
14. virtual void display() { A::display(); std::cout << "Display B" << std::endl; }
15. B() = default;
16. virtual ~B() = default;
17. };
18. void test() {
19. std::unique\_ptr<A> ptr = std::make\_unique<B>();
20. ptr->display();
21. }

答：输出

Draw B

Display A

Display B

首先调用 A::display()

其中又调用了draw()，根据实际类型进行了动态绑定

最后displayB

# Ch10作业

指出每行输出结果

1. #include <iostream>
2. using namespace std;
3. struct A
4. {
5. A() { cout << "A"; }
6. };
7. struct B : A
8. {
9. A a;
10. B() : A() { cout << "B"; }
11. };
12. struct C : virtual A
13. {
14. C() : A() { cout << "C"; }
15. };
16. struct D : virtual A, C
17. {
18. C c;
19. D() : A(), C() { cout << "D"; }
20. };
21. struct E : virtual B, virtual C
22. {
23. E() : B(), C() { cout << "E"; }
24. };
25. struct F : virtual B, D, virtual E
26. {
27. D d;
28. E e;
29. F() : B(), D(), E() { cout << "F"; }
30. };
31. int main()
32. {
33. A a;
34. cout << "\n"; *// 输出：*
35. B b;
36. cout << "\n"; *// 输出：*
37. C c;
38. cout << "\n"; *// 输出：*
39. D d;
40. cout << "\n"; *// 输出：*
41. E e;
42. cout << "\n"; *// 输出：*
43. F f;
44. cout << "\n"; *// 输出：*
45. }

答：输出为(空格是为了阅读方便，实际输出不带空格)

A

AA B

A C

AC AC D

AAB AC E

AAB ACE CACD ACACD AABACE F

解释：

1. A没有父类，直接输出A，完成A的构造
2. 首先构造B的父类A，然后构造B的成员A a，最后输出B，完成B的构造
3. C虚继承A，先构造A，然后输出C，完成C的构造
4. D虚继承A，先构造A，D同时继承C，再构造C(此时不必再重新构造A了),接着构造数据成员c(输出AC),最后完成D的构造函数，输出D，完成D的构造
5. E虚继承B,C，先构造B，输出AAB，然后构造C，输出AC，最后输出E，完成E的构造
6. F虚继承B，E，**先构造B**，输出AAB,后构造E，构造E要**构造C**(输出AC),然后输出E完成**E的构造**，然后**构造D**，D要继承c，输出C（虚基类A已经构造），然后构造D的数据成员c输出AC（另开一棵树），最后输出D完成D的构造。

接着，完成E的两个数据成员的构造，结果同前。

一个用于理解的**并不规范**的示意图



# Ch11作业

## 11.1

*/\**

*阅读下面程序并回答问题*

*\*/*struct A {

int i;

int j;

operator int() { return i + j; }

A(int x, int y):i(x),j(y){}

} a = { 1,2 };

struct B : A {

int m;

int n;

operator A() { return A(i, j); }

operator int() { return A(\*this) + m + n; }

B(int x, int y, int m, int n):A(x, y), m(m), n(n){}

} b = { a,a,3,4 };

void test() {

*/\**

*下面语句执行以后，i的值是多少？请说明详细计算过程*

*\*/*

int i = a + b;

std::cout << "i = " << i << std::endl;

}

答：i为16

A a转化为int后为3

B中构造的A(3,3),转化为int后为6，m,n,为3，4

四项相加后3+6+3+4=16

## 11.2

阅读下面程序并回答问题

1. struct A {
2. int x;
3. static int y;
4. public:
5. operator int() { return x + y; }
6. A& operator+=(const A& a);
7. A operator++(int);
8. A(int x = 1, int y = 1) :x(x) { A::y = y; }
9. };
10. int A::y = 20;
11. A& A::operator+=(const A& a) {
12. x += a.x;
13. y += a.y;
14. return \*this;
15. }
16. A A::operator++(int) {
17. return A(x++, y++);
18. }
19. void test() {
20. A a(2, 5), b(6), c;
21. */\**
22. 下面每条语句执行完后，i的值是多少，对象a，b，c的内容分别是多少
23. \*/
24. int i = b.y;
25. i = a++;
26. i = a + c;
27. i = (b += c);
28. }

答：注意**y++后创造临时对象A将静态变量y重置为1**

i=1

a:x=2,y=1,b:x=6,y=1,c:x=1,y=1

i=3

a:x=3,y=1,b:x=6,y=1,c:x=1,y=1

i=6

a:x=3,y=1,b:x=6,y=1,c:x=1,y=1

i=9

a:x=3,y=2,b:x=7,y=2,c:x=1,y=2