# 计算机学院 高级程序语言设计 课程实验报告

| | | |
|---|---|---|
| 实验题目：对象数组，指针，动态内存 | | 学号：202300130150 |
| 日期：2024.4.2 | 班级： 4 | 姓名： 王成意 |

实验目的：
1. 练习对象数组
2. 训练使用指针的能力：深刻认识灵活性与潜在的危险，理解指针与引用的不同。
3. 练习动态内存的使用
4. 练习 vector 与 string 的使用
5. 联系感受类的继承与派生

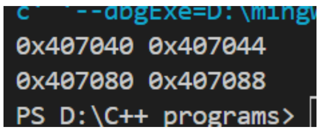
实验步骤与内容：
1. 数组

（1）定义两个数组，int a[10],double b[10] 分别用 cout 打印 a，(a+1)以及 b，(b+1)的值，他们有什么不同，为什么？注意打印结果是 16 进制

代码：

```
#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

#define ll long long

using namespace std;

int a[10];

double b[10];

signed main()

{

    cout << a << " " << (a + 1) << endl;

    cout << b << " " << (b + 1) << endl;
```

```
    return 0;

}
```

结果：

```
c  --dbgExe=D:\mingw
0x407040 0x407044
0x407080 0x407088
PS D:\C++ programs>
```

解释：他们的起始地址和每个数组元素的大小不同。
B 和 b+1 差了 8 位，因为 double 型占 8 个字节；
A 和 a+1 差了 4 位，因为 int 型占 4 字节。
再次调整程序发现，(a+16) 和 b 的地址相同。

（2）修改实验 7 中 5_10.cpp 的 main 函数，分别包含以下语句：

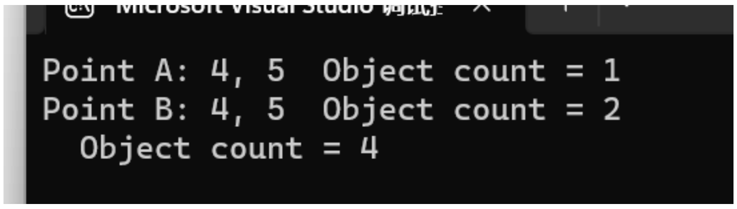
```
Point ps[2];
Point::showCount();
```

以及

```
Point *ps[2];
Point::showCount();
```

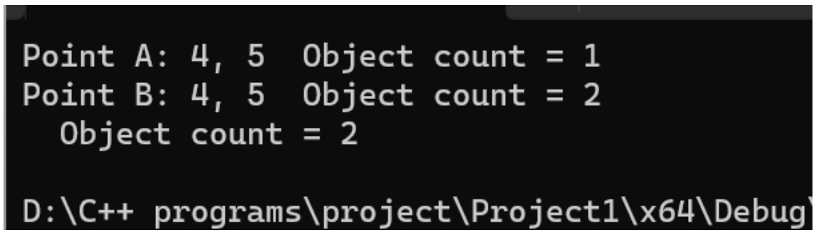
运行结果并截屏，解释两者的区别。

结果 1：



```
Microsoft Visual Studio 调试控
Point A: 4, 5  Object count = 1
Point B: 4, 5  Object count = 2
  Object count = 4
```

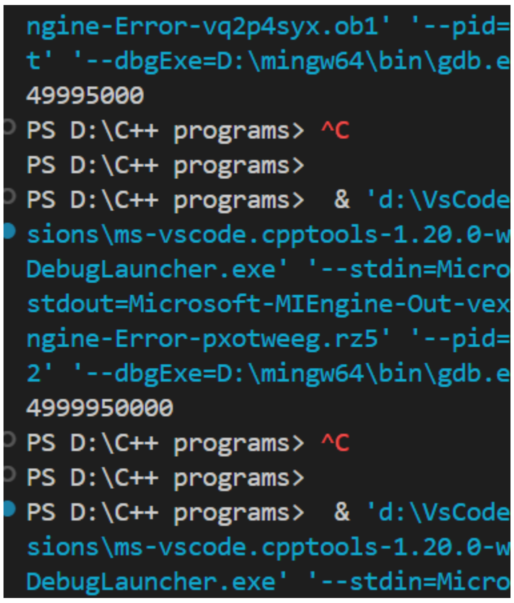
结果 2：



```
Point A: 4, 5  Object count = 1
Point B: 4, 5  Object count = 2
  Object count = 2

D:\C++ programs\project\Project1\x64\Debug
```

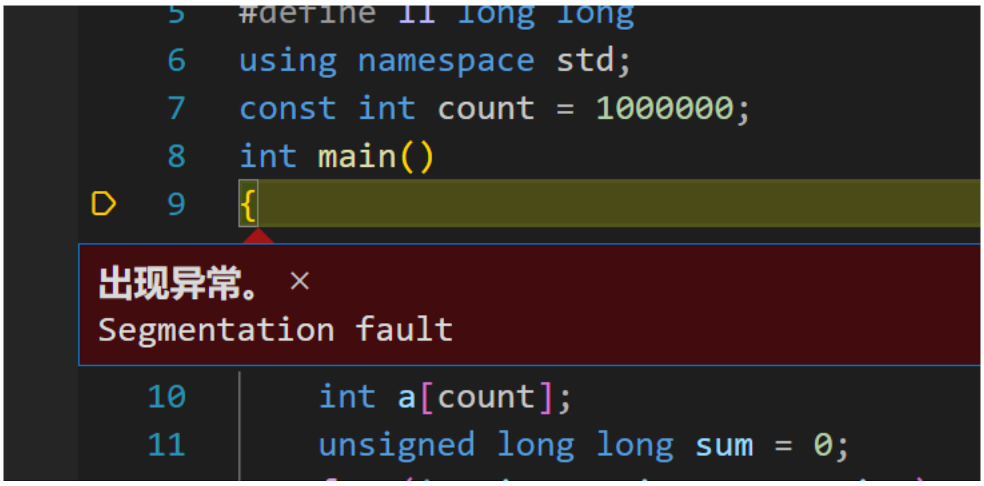
解释：第二个修改里*ps 是指针，并新建 point 型对象，所以在 point 类里面没有新的 object。

（3）运行以下代码，并修改 count 的值，每次增大十倍，例如测试 100000, 1000000 等 看看增大到多少的时候程序运行出现错误？
10000 和 100000：



```
ngine-Error-vq2p4syx.ob1' '--pid=
t' '--dbgExe=D:\mingw64\bin\gdb.e
49995000
PS D:\C++ programs> ^C
PS D:\C++ programs>
PS D:\C++ programs>  & 'd:\VsCode
sions\ms-vscode.cpptools-1.20.0-w
DebugLauncher.exe' '--stdin=Micro
stdout=Microsoft-MIEngine-Out-vex
ngine-Error-pxotweeg.rz5' '--pid=
2' '--dbgExe=D:\mingw64\bin\gdb.e
4999950000
PS D:\C++ programs> ^C
PS D:\C++ programs>
PS D:\C++ programs>  & 'd:\VsCode
sions\ms-vscode.cpptools-1.20.0-w
DebugLauncher.exe' '--stdin=Micro
```

1000000：



```
#define ll long long
using namespace std;
const int count = 1000000;
int main()
{
```

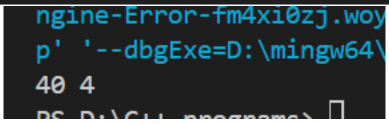
出现异常。 ×
Segmentation fault

```
    int a[count];
    unsigned long long sum = 0;
```

（4）定义一个数组 int a[10];分别打印 sizeof(a)以及 sizeof(a[0])的值，分析他们分别对应着什么值？

```
#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

#define ll long long

using namespace std;

int a[10];
```

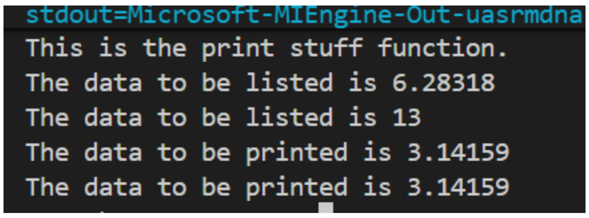
```
signed main()

{

    cout << sizeof(a) << " " << sizeof(a[0]);

    return 0;

}
```

结果：

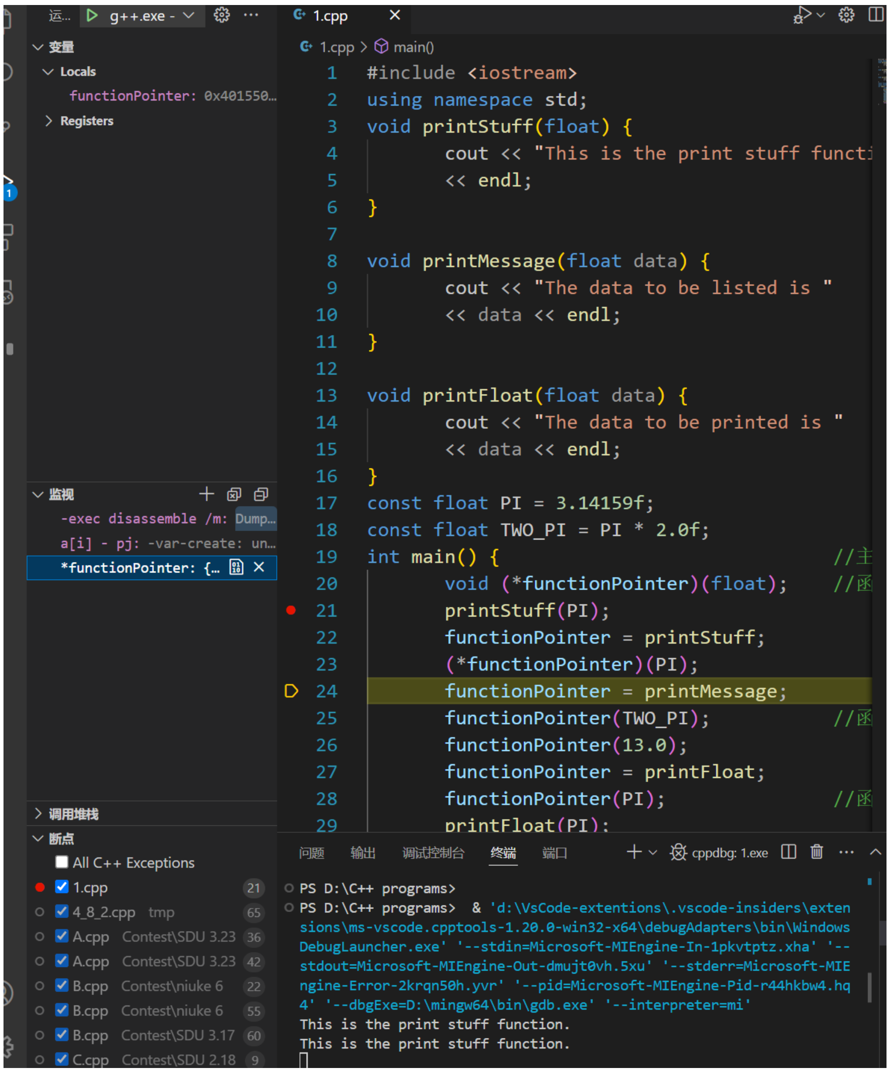
```
ngine-Error-fm4xi0zj.woy
p' '--dbgExe=D:\mingw64\
40 4
PS D:\C++ programs> 
```

前者：a[10]总共占用的字节数；
后者：a[0]自己占用的字节数。

2. 练习以下课程内容，做好结果截图与分析
   练习第六章 PPT，P88 例 6-11，函数指针



```
stdout=Microsoft-MIEngine-Out-uasrmdna
This is the print stuff function.
The data to be listed is 6.28318
The data to be listed is 13
The data to be printed is 3.14159
The data to be printed is 3.14159
```



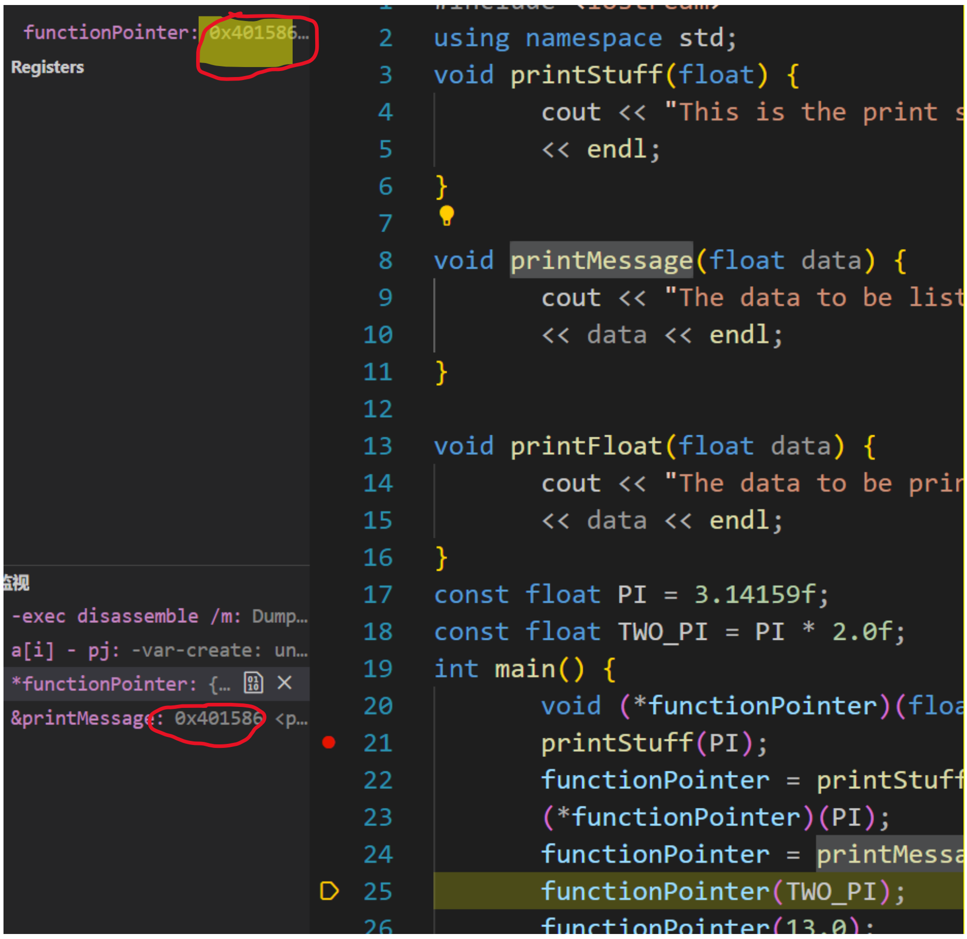
```
functionPointer: 0x401550…
```

```
-exec disassemble /m: Dump…
a[i] - pj: -var-create: un…
*functionPointer: {…
```

```
All C++ Exceptions
1.cpp                          21
4_8_2.cpp    tmp               65
A.cpp        Contest\SDU 3.23  36
A.cpp        Contest\SDU 3.23  42
B.cpp        Contest\niuke 6   22
B.cpp        Contest\niuke 6   55
B.cpp        Contest\SDU 3.17  60
C.cpp        Contest\SDU 2.18  9
```

```cpp
#include <iostream>
using namespace std;
void printStuff(float) {
        cout << "This is the print stuff functi
        << endl;
}

void printMessage(float data) {
        cout << "The data to be listed is "
        << data << endl;
}

void printFloat(float data) {
        cout << "The data to be printed is "
        << data << endl;
}
const float PI = 3.14159f;
const float TWO_PI = PI * 2.0f;
int main() {                                    //主
        void (*functionPointer)(float);         //函
        printStuff(PI);
        functionPointer = printStuff;
        (*functionPointer)(PI);
        functionPointer = printMessage;
        functionPointer(TWO_PI);                //函
        functionPointer(13.0);
        functionPointer = printFloat;
        functionPointer(PI);                    //函
        printFloat(PI);
```

```
PS D:\C++ programs>
PS D:\C++ programs>  & 'd:\VsCode-extentions\.vscode-insiders\exten
sions\ms-vscode.cpptools-1.20.0-win32-x64\debugAdapters\bin\Windows
DebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-1pkvtptz.xha' '--
stdout=Microsoft-MIEngine-Out-dmujt0vh.5xu' '--stderr=Microsoft-MIE
ngine-Error-2krqn50h.yvr' '--pid=Microsoft-MIEngine-Pid-r44hkbw4.hq
4' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
This is the print stuff function.
This is the print stuff function.
```

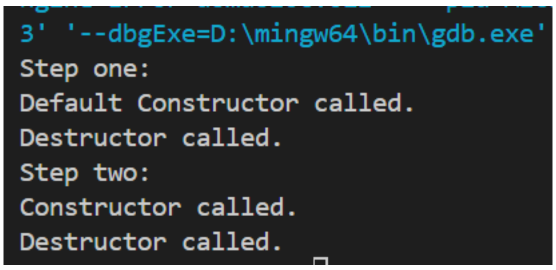


```
functionPointer: 0x401586…
Registers

监视
-exec disassemble /m: Dump…
a[i] - pj: -var-create: un…
*functionPointer: {…
&printMessage: 0x401586 <p…
```

```
using namespace std;
void printStuff(float) {
        cout << "This is the print s
        << endl;
}

void printMessage(float data) {
        cout << "The data to be list
        << data << endl;
}

void printFloat(float data) {
        cout << "The data to be prin
        << data << endl;
}
const float PI = 3.14159f;
const float TWO_PI = PI * 2.0f;
int main() {
        void (*functionPointer)(floa
        printStuff(PI);
        functionPointer = printStuff
        (*functionPointer)(PI);
        functionPointer = printMessa
        functionPointer(TWO_PI);
        functionPointer(13.0);
```

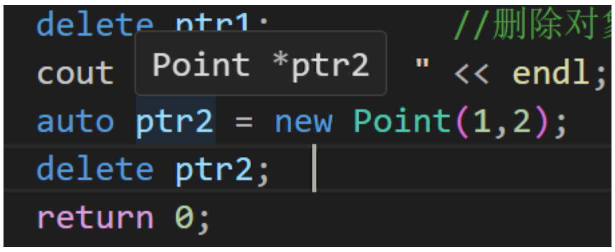
此后的传递与前两个类似，故省略。

练习第六章 PPT，P100 例 6-16， 用 new 创建动态对象。



```
3' '--dbgExe=D:\mingw64\bin\gdb.exe'
Step one:
Default Constructor called.
Destructor called.
Step two:
Constructor called.
Destructor called.
```

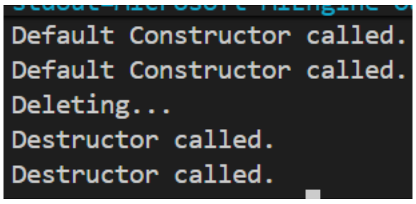
两次调用的构造函数并不相同。
其中如果忘记了相应的类型可以用 auto：



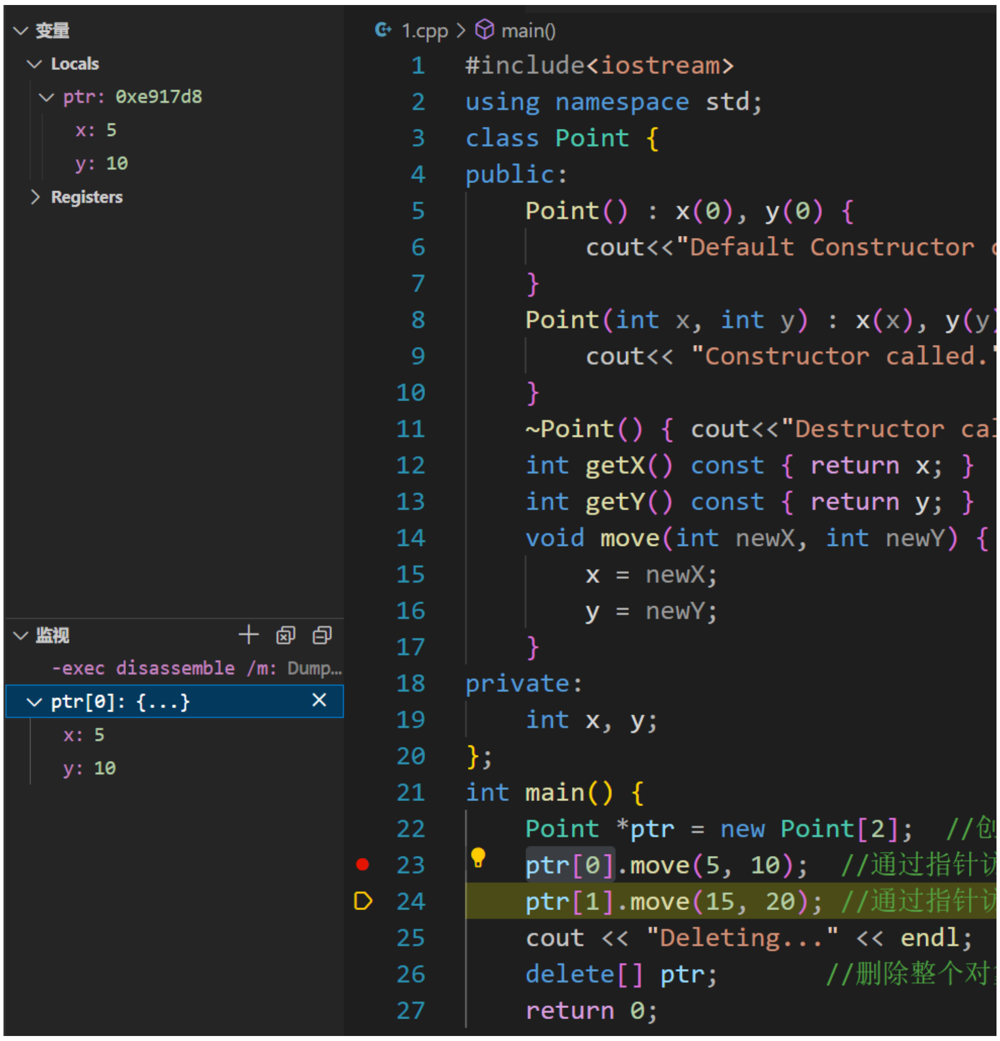
```
delete ptr1;              //删除对
cout  Point *ptr2   " << endl;
auto ptr2 = new Point(1,2);
delete ptr2;
return 0;
```

成功识别。

练习第六章 PPT，P103 例 6-17， 动态创建对象数组。

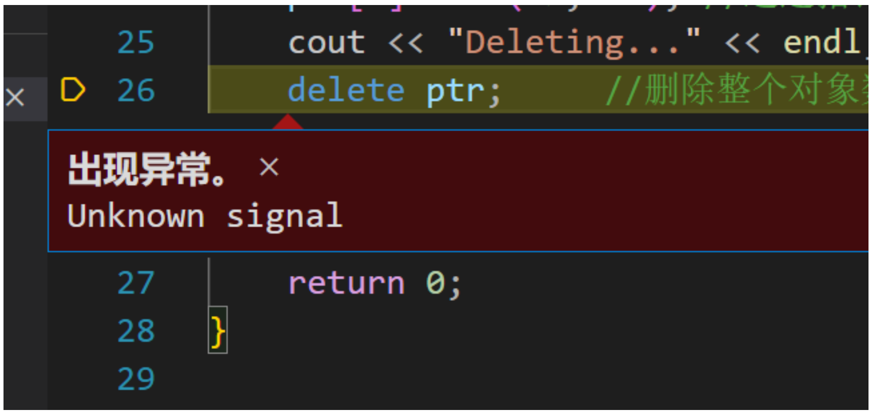


```
Default Constructor called.
Default Constructor called.
Deleting...
Destructor called.
Destructor called.
```



```
#include<iostream>
using namespace std;
class Point {
public:
    Point() : x(0), y(0) {
        cout<<"Default Constructor c
    }
    Point(int x, int y) : x(x), y(y
        cout<< "Constructor called."
    }
    ~Point() { cout<<"Destructor cal
    int getX() const { return x; }
    int getY() const { return y; }
    void move(int newX, int newY) {
        x = newX;
        y = newY;
    }
private:
    int x, y;
};
int main() {
    Point *ptr = new Point[2];  //创
    ptr[0].move(5, 10);  //通过指针访
    ptr[1].move(15, 20); //通过指针访
    cout << "Deleting..." << endl;
    delete[] ptr;       //删除整个对
    return 0;
```

由此可见，访问成功。
如果 delete 不加[]：

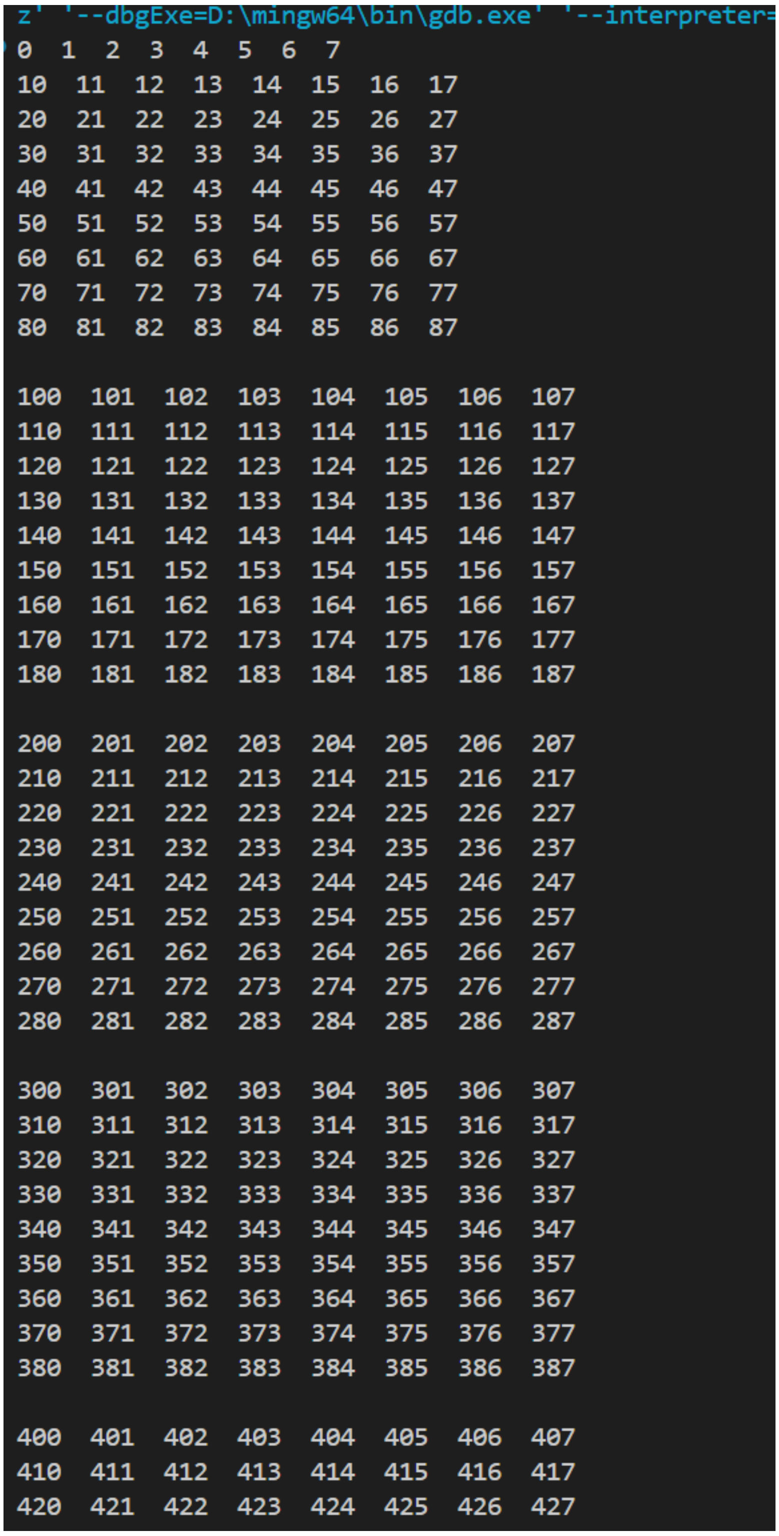


```
    cout << "Deleting..." << endl
    delete ptr;       //删除整个对象
```

**出现异常。** ×
Unknown signal

```
    return 0;
}
```

练习第六章 PPT，P107 例 6-18， 动态创建多维数组。



```
z' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=
0  1  2  3  4  5  6  7
10  11  12  13  14  15  16  17
20  21  22  23  24  25  26  27
30  31  32  33  34  35  36  37
40  41  42  43  44  45  46  47
50  51  52  53  54  55  56  57
60  61  62  63  64  65  66  67
70  71  72  73  74  75  76  77
80  81  82  83  84  85  86  87

100  101  102  103  104  105  106  107
110  111  112  113  114  115  116  117
120  121  122  123  124  125  126  127
130  131  132  133  134  135  136  137
140  141  142  143  144  145  146  147
150  151  152  153  154  155  156  157
160  161  162  163  164  165  166  167
170  171  172  173  174  175  176  177
180  181  182  183  184  185  186  187

200  201  202  203  204  205  206  207
210  211  212  213  214  215  216  217
220  221  222  223  224  225  226  227
230  231  232  233  234  235  236  237
240  241  242  243  244  245  246  247
250  251  252  253  254  255  256  257
260  261  262  263  264  265  266  267
270  271  272  273  274  275  276  277
280  281  282  283  284  285  286  287

300  301  302  303  304  305  306  307
310  311  312  313  314  315  316  317
320  321  322  323  324  325  326  327
330  331  332  333  334  335  336  337
340  341  342  343  344  345  346  347
350  351  352  353  354  355  356  357
360  361  362  363  364  365  366  367
370  371  372  373  374  375  376  377
380  381  382  383  384  385  386  387

400  401  402  403  404  405  406  407
410  411  412  413  414  415  416  417
420  421  422  423  424  425  426  427
```

结果太多无法截完，取了其中一部分。

可见

```
*(*(*(cp + i) + j) + k) =(i * 100 + j * 10 + k);
```

这一句话等效于

```
cp[i][j][k] =(i * 100 + j * 10 + k);
```

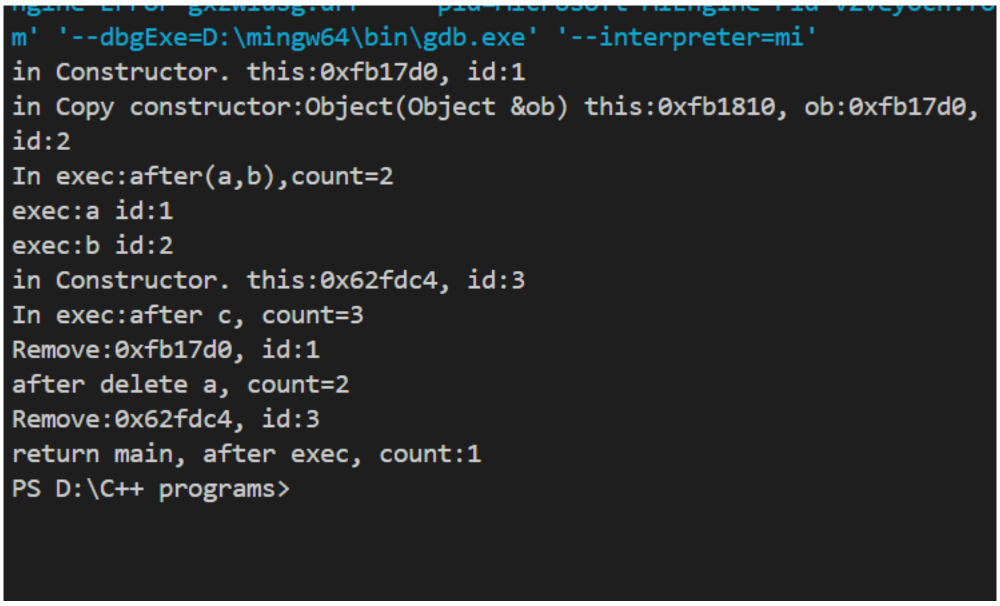
再次验证了动态创建多维数组的可行性。

3. 分析实验 8 附件中的 c6test.cpp。

1）请解释 exec 函数中以下语句（第 44 行）的含义：

    Object *a=new Object, &b(*new Object(*a));//请解释该语句

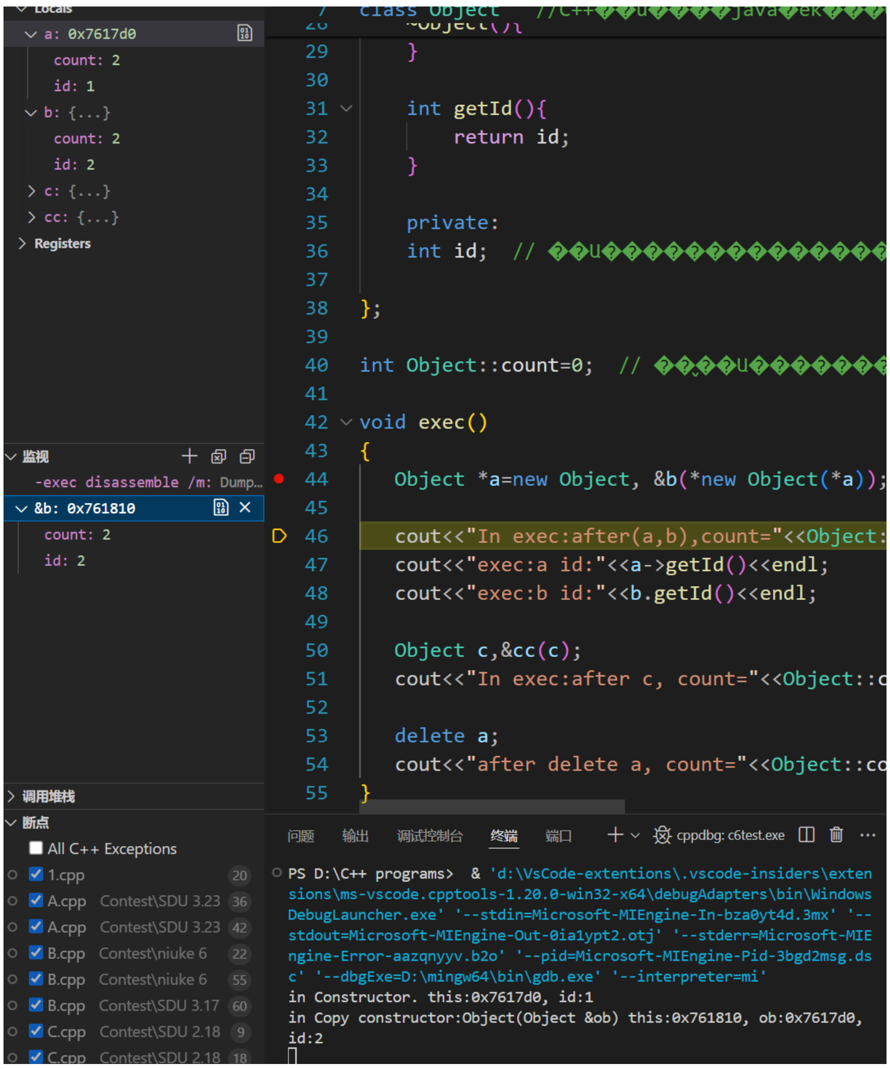
结果：



```
m' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
in Constructor. this:0xfb17d0, id:1
in Copy constructor:Object(Object &ob) this:0xfb1810, ob:0xfb17d0,
id:2
In exec:after(a,b),count=2
exec:a id:1
exec:b id:2
in Constructor. this:0x62fdc4, id:3
In exec:after c, count=3
Remove:0xfb17d0, id:1
after delete a, count=2
Remove:0x62fdc4, id:3
return main, after exec, count:1
PS D:\C++ programs>
```

解释：

查看 debug：



```
Locals
  a: 0x7617d0
    count: 2
    id: 1
  b: {...}
    count: 2
    id: 2
  c: {...}
  cc: {...}
Registers

监视
  -exec disassemble /m: Dump...
  &b: 0x761810
    count: 2
    id: 2
```

```cpp
class Object   //C++��u����java�ek����
    ~Object(){
    }

    int getId(){
        return id;
    }

    private:
    int id;  // ��u��������������
};

int Object::count=0;  // ����u�������

void exec()
{
   Object *a=new Object, &b(*new Object(*a));

   cout<<"In exec:after(a,b),count="<<Object:
   cout<<"exec:a id:"<<a->getId()<<endl;
   cout<<"exec:b id:"<<b.getId()<<endl;

   Object c,&cc(c);
   cout<<"In exec:after c, count="<<Object::c

   delete a;
   cout<<"after delete a, count="<<Object::co
}
```

```
PS D:\C++ programs>  & 'd:\VsCode-extentions\.vscode-insiders\exten
sions\ms-vscode.cpptools-1.20.0-win32-x64\debugAdapters\bin\Windows
DebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-bza0yt4d.3mx' '--
stdout=Microsoft-MIEngine-Out-0ia1ypt2.otj' '--stderr=Microsoft-MIE
ngine-Error-aazqnyyv.b2o' '--pid=Microsoft-MIEngine-Pid-3bgd2msg.ds
c' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
in Constructor. this:0x7617d0, id:1
in Copy constructor:Object(Object &ob) this:0x761810, ob:0x7617d0,
id:2
```

All C++ Exceptions
1.cpp 20
A.cpp Contest\SDU 3.23 36
A.cpp Contest\SDU 3.23 42
B.cpp Contest\niuke 6 22
B.cpp Contest\niuke 6 55
B.cpp Contest\SDU 3.17 60
C.cpp Contest\SDU 2.18 9
C.cpp Contest\SDU 2.18 18
调用堆栈
断点
问题 输出 调试控制台 终端 端口
cppdbg: c6test.exe

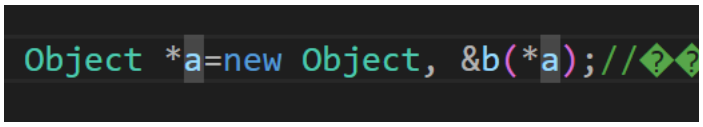
可见 a 是一个 object 类型的指针，指向了动态开辟的一块内存，地址是 0x7617d0；

B 是调用了复制构造函数：

```cpp
Object(const Object

&ob){   //���ε��갃��������L�д���ε��갃�

������Y�U��������Ç�L���g�Ĭ���ϵT�Ω�

����

        count++;

        id=count;       //

id����Ç����ҵ�������ἐU��
```

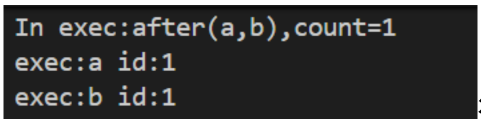
```
        cout<<"in Copy constructor:Object(Object &ob)
this:"<<this<<", ob:"<<&ob<<", id:"<<id<<endl;
//��������.��ͱ����z����.�覬����id


    }
```

如果括号里不是 new object，而是*a，像这样：



```
Object *a=new Object, &b(*a);//��
```

那么 b 就是 a 的别名，验证：

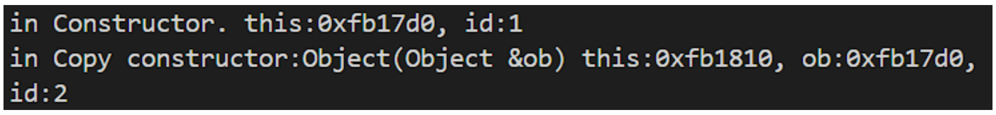


```
In exec:after(a,b),count=1
exec:a id:1
exec:b id:1
```

符合条件。

但是 b 是（*new Object(*a)）的别名，这一串是复制 a 构造的一个新的对象，别名是 b（或者说本名是 b，因为它没有其他名字）

所以 b 与 a 的地址不同（见上 debug 图里的 ab 地址），id 也不同（因为 b 所代表的是一个新的对象，复制 a 来的）：



```
in Constructor. this:0xfb17d0, id:1
in Copy constructor:Object(Object &ob) this:0xfb1810, ob:0xfb17d0,
id:2
```

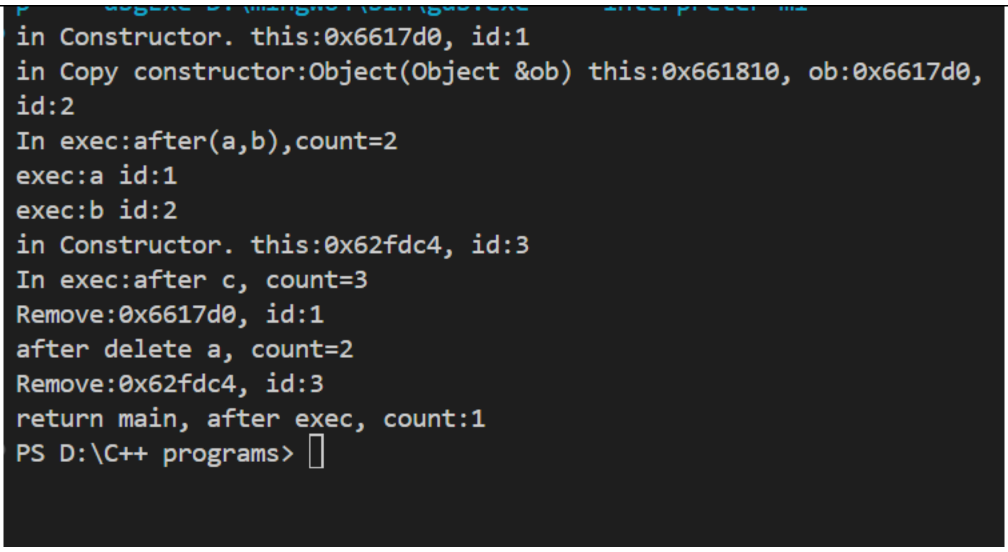
其中 this 是&b，ob 是*a 的别名，所以&ob 就是 a。

```
ob:"<<&ob<<"
```

2）如果此句中的&b(*new Object(*a))改为如下两种形式，其含义及对运行结果的影响有什么不同？

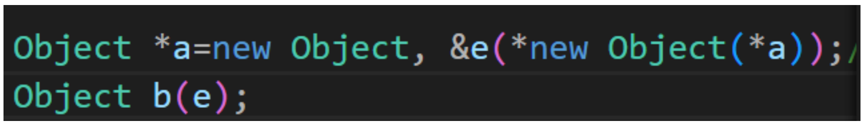
第一种：

a）*b(new Object(*a))；//注意此时 b 是指针，后序对类成员的调用需要用 b->而不是 b.



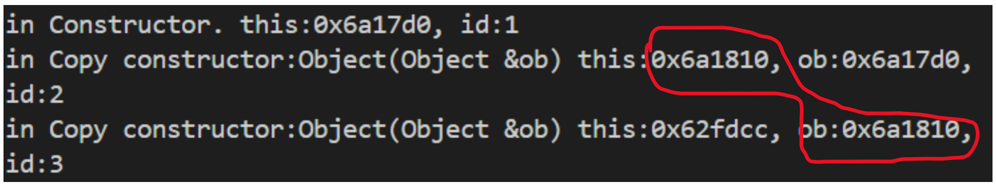
```
in Constructor. this:0x6617d0, id:1
in Copy constructor:Object(Object &ob) this:0x661810, ob:0x6617d0,
id:2
In exec:after(a,b),count=2
exec:a id:1
exec:b id:2
in Constructor. this:0x62fdc4, id:3
In exec:after c, count=3
Remove:0x6617d0, id:1
after delete a, count=2
Remove:0x62fdc4, id:3
return main, after exec, count:1
PS D:\C++ programs> 
```

无影响。
第二种:
b) b(*new Object(*a))
调用了两次复制构造函数:
第一次复制构造函数是地址为 0x1001810 的对象复制构造*a,第二次复制构造是 b 复制构地址为 0x1001810 的对象:
等价于:



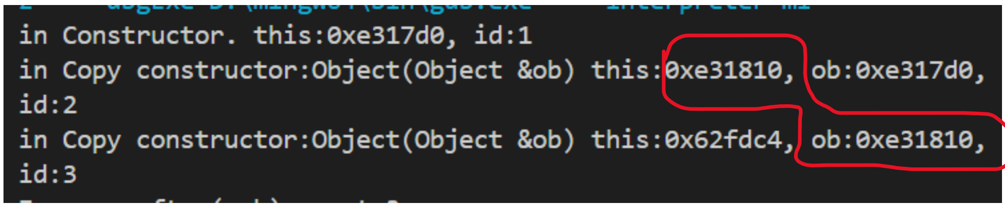
```
Object *a=new Object, &e(*new Object(*a));
Object b(e);
```

观察结果:
原始:



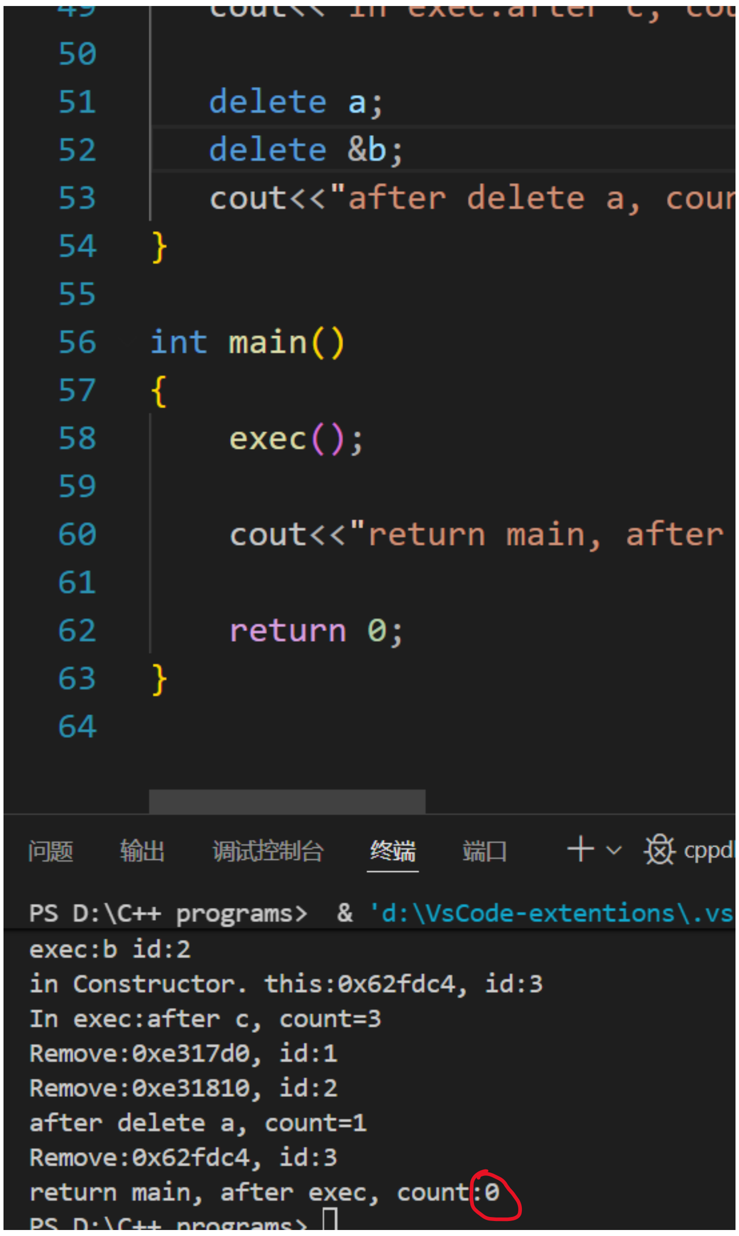
```
in Constructor. this:0x6a17d0, id:1
in Copy constructor:Object(Object &ob) this:0x6a1810, ob:0x6a17d0,
id:2
in Copy constructor:Object(Object &ob) this:0x62fdcc, ob:0x6a1810,
id:3
```

等价:



```
in Constructor. this:0xe317d0, id:1
in Copy constructor:Object(Object &ob) this:0xe31810, ob:0xe317d0,
id:2
in Copy constructor:Object(Object &ob) this:0x62fdc4, ob:0xe31810,
id:3
```

二者是等价的。
3) 此题如何避免内存泄漏? 除了 delete a 之外, 还有谁需要被 delete?
还需要 delete b。
观察 debug 发现并没有对 b 执行析构函数。
加上 delete &b:



```
    cout<<"In exec:after c, cou

    delete a;
    delete &b;
    cout<<"after delete a, cour
}

int main()
{
    exec();

    cout<<"return main, after

    return 0;
}

```

```
PS D:\C++ programs>  & 'd:\VsCode-extentions\.vs
exec:b id:2
in Constructor. this:0x62fdc4, id:3
In exec:after c, count=3
Remove:0xe317d0, id:1
Remove:0xe31810, id:2
after delete a, count=1
Remove:0x62fdc4, id:3
return main, after exec, count:0
PS D:\C++ programs> 
```

可见应该 delete &b。

4. vector 练习

（1）请下一个程序，实例化一个 int 类型的 vector 对象 arr。push_back 函数可以向 vector 数组中插入数值，例如 arr.push_back(10)插入值 10。请写程序让用户持续的输入整数值插入 arr 数组中，当用户输入 0 时停止输入。然后通过以下代码遍历 vector 数组将用户输入的值相加并输出。

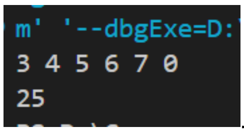
```
int sum = 0;
    for(int i=0;i<arr.size();i++){
     sum += arr[i];
    }
    cout<<sum<<endl;
```

代码：

```
#include<cstdio>
```

```cpp
#include<cstring>

#include<iostream>

#include<algorithm>

#include<vector>

#define ll long long

using namespace std;

vector<int> arr;

int a;

signed main()

{

    while(1)

    {

        cin >> a;

        if(a==0)

            break;

        arr.push_back(a);

    }

    int sum = 0;

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

        sum += arr[i];

    }

    cout<<sum<<endl;
    return 0;

}
```

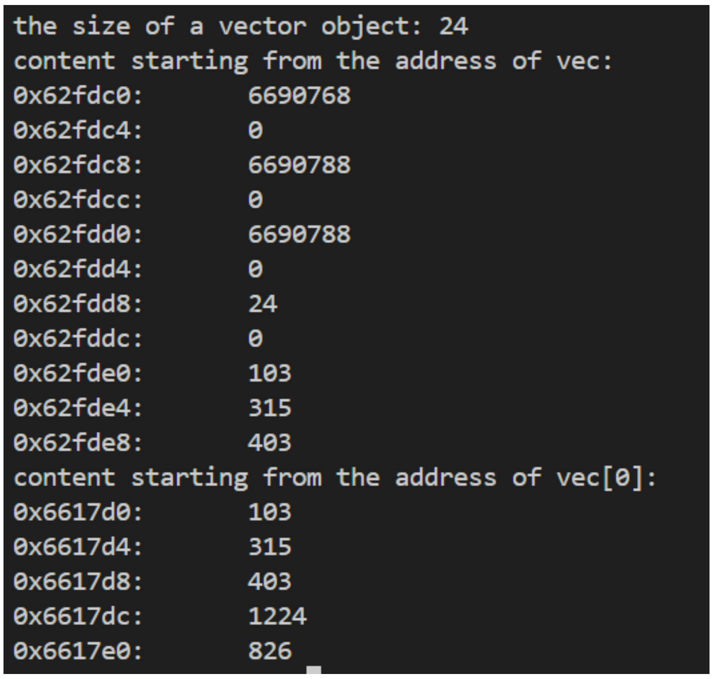
结果：



```
m' '--dbgExe=D:
3 4 5 6 7 0
25
```

正确。

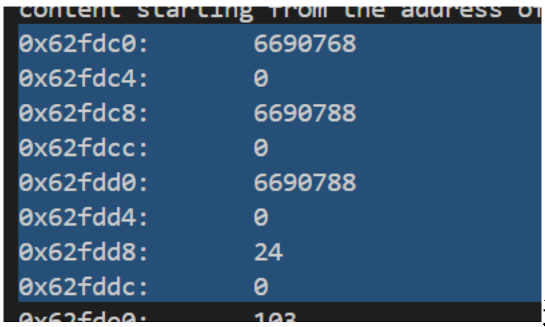
（2）运行并解释如下代码段的内容。



```
the size of a vector object: 24
content starting from the address of vec:
0x62fdc0:       6690768
0x62fdc4:       0
0x62fdc8:       6690788
0x62fdcc:       0
0x62fdd0:       6690788
0x62fdd4:       0
0x62fdd8:       24
0x62fddc:       0
0x62fde0:       103
0x62fde4:       315
0x62fde8:       403
content starting from the address of vec[0]:
0x6617d0:       103
0x6617d4:       315
0x6617d8:       403
0x6617dc:       1224
0x6617e0:       826
```

解释：

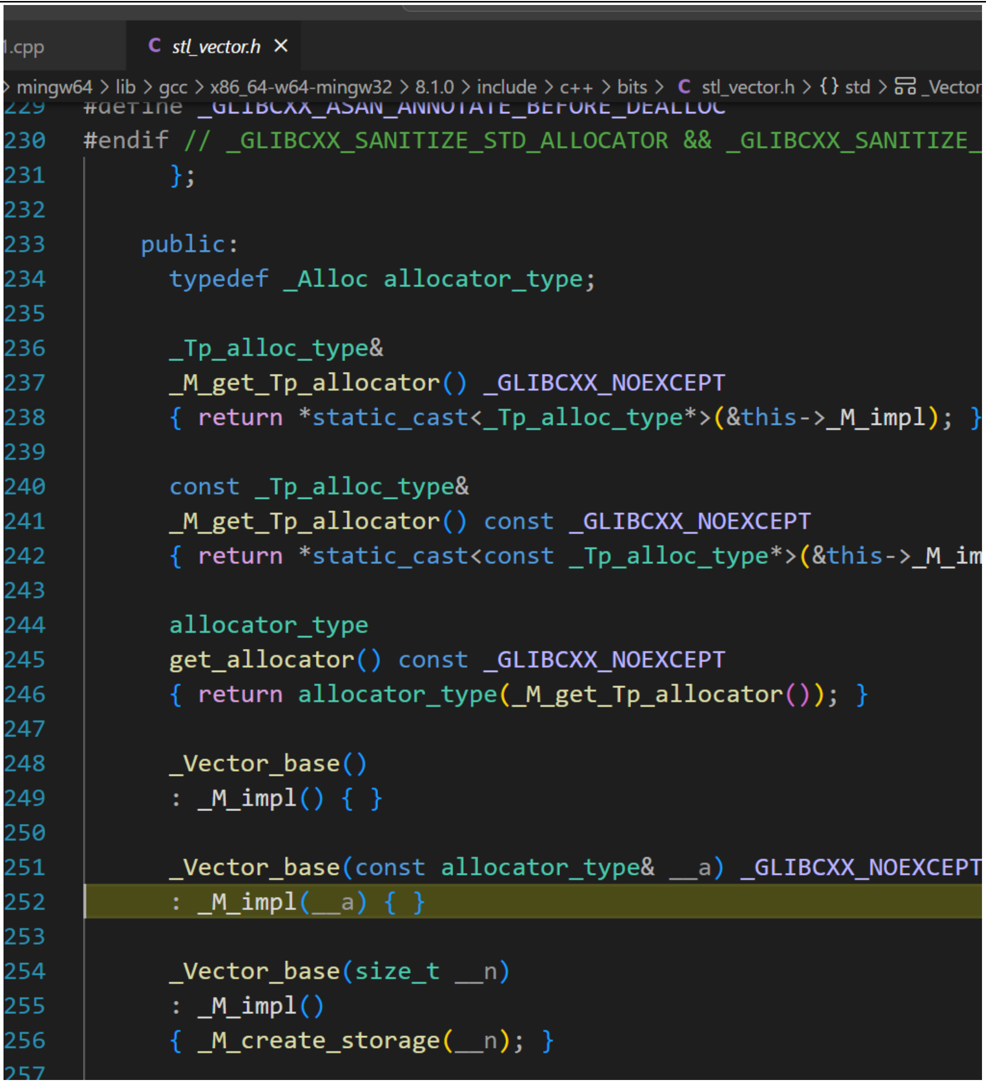
Vector 并不是从一开始就储存数据内容的，



```
content starting from the address of
0x62fdc0:       6690768
0x62fdc4:       0
0x62fdc8:       6690788
0x62fdcc:       0
0x62fdd0:       6690788
0x62fdd4:       0
0x62fdd8:       24
0x62fddc:       0
0x62fde0:       103
```

这些都是内部的准备。

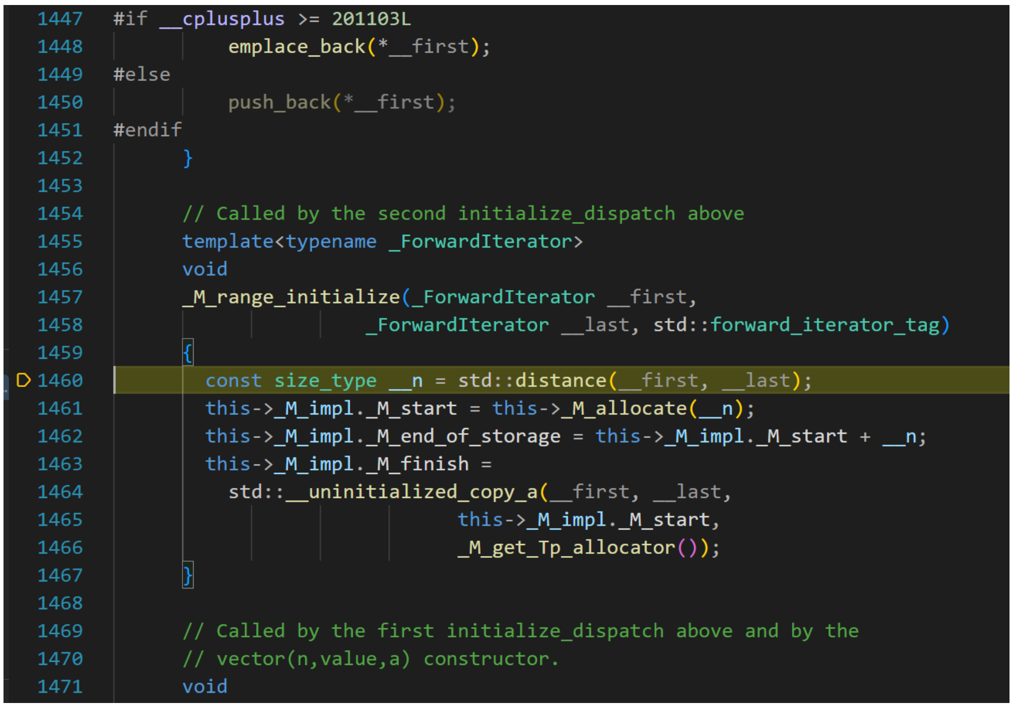
事实上，vector 是一个类模板：



.cpp   stl_vector.h ×

mingw64 > lib > gcc > x86_64-w64-mingw32 > 8.1.0 > include > c++ > bits > stl_vector.h > {} std > _Vector

```
#define _GLIBCXX_ASAN_ANNOTATE_BEFORE_DEALLOC
#endif // _GLIBCXX_SANITIZE_STD_ALLOCATOR && _GLIBCXX_SANITIZE_
    };

  public:
    typedef _Alloc allocator_type;

    _Tp_alloc_type&
    _M_get_Tp_allocator() _GLIBCXX_NOEXCEPT
    { return *static_cast<_Tp_alloc_type*>(&this->_M_impl); }

    const _Tp_alloc_type&
    _M_get_Tp_allocator() const _GLIBCXX_NOEXCEPT
    { return *static_cast<const _Tp_alloc_type*>(&this->_M_im

    allocator_type
    get_allocator() const _GLIBCXX_NOEXCEPT
    { return allocator_type(_M_get_Tp_allocator()); }

    _Vector_base()
    : _M_impl() { }

    _Vector_base(const allocator_type& __a) _GLIBCXX_NOEXCEPT
    : _M_impl(__a) { }

    _Vector_base(size_t __n)
    : _M_impl()
    { _M_create_storage(__n); }
```



```
#if __cplusplus >= 201103L
	    emplace_back(*__first);
#else
	    push_back(*__first);
#endif
	}

      // Called by the second initialize_dispatch above
      template<typename _ForwardIterator>
	void
	_M_range_initialize(_ForwardIterator __first,
			    _ForwardIterator __last, std::forward_iterator_tag)
	{
	  const size_type __n = std::distance(__first, __last);
	  this->_M_impl._M_start = this->_M_allocate(__n);
	  this->_M_impl._M_end_of_storage = this->_M_impl._M_start + __n;
	  this->_M_impl._M_finish =
	    std::__uninitialized_copy_a(__first, __last,
					this->_M_impl._M_start,
					_M_get_Tp_allocator());
	}

      // Called by the first initialize_dispatch above and by the
      // vector(n,value,a) constructor.
      void
```

其中有很多复杂的东西。

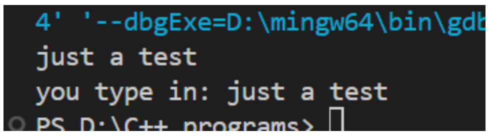
5. string 练习

请写一个程序，实例化一个 string 类型的变量，变量名为 str，之后利用 getline 函数读取一条字符串“just a test”到 str 中。之后再实例化一个 string 变量 str1，内容是"you type in: "+str。之后打印 str1 的内容。别忘了#include <string>

代码：

```cpp
#include<cstdio>

#include<cstring>

#include<string>

#include<iostream>

#include<algorithm>

#define ll long long

using namespace std;

signed main()

{

    string str;

    getline(cin, str);

    string str1 = "you type in: " + str;

    cout << str1;

    return 0;

}
```

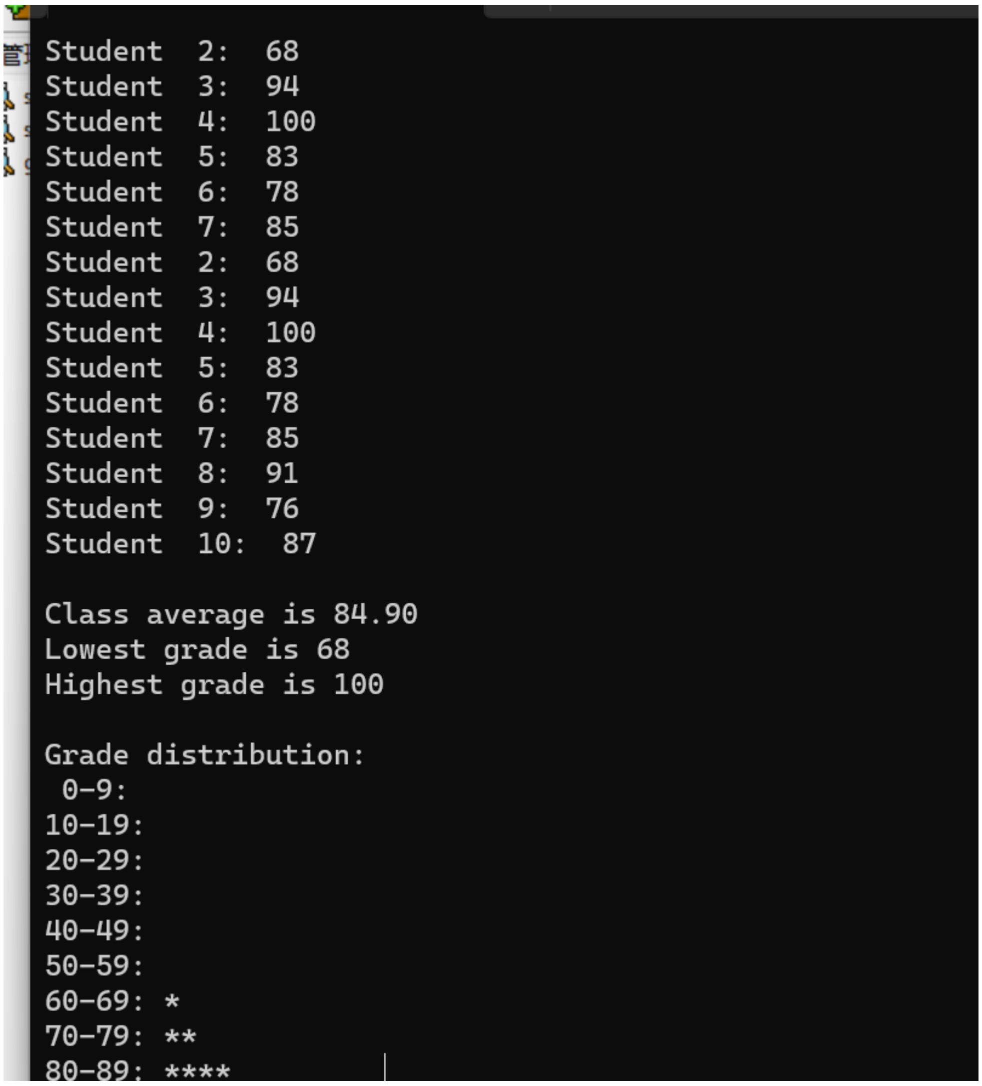
结果：



```
4' '--dbgExe=D:\mingw64\bin\gdb
just a test
you type in: just a test
PS D:\C++ programs>
```

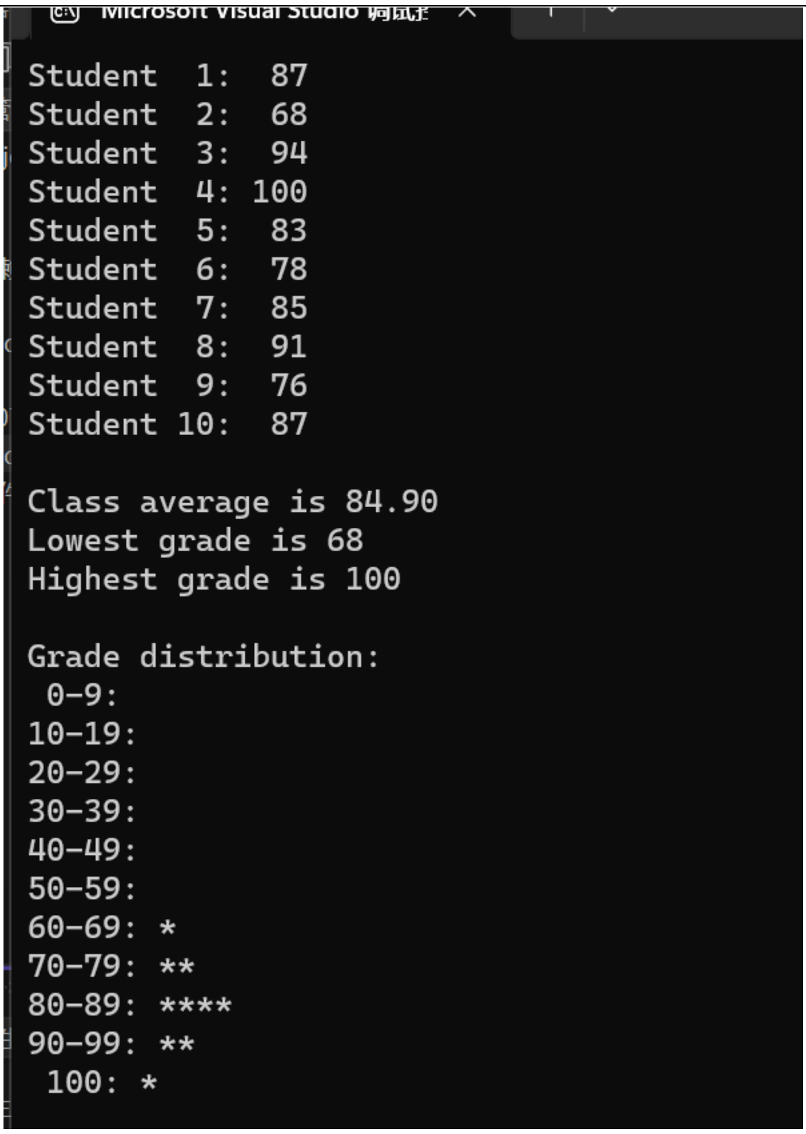
6. 分析对象数组项目 fig07_15to17 以及 fig07_22to24。

对于 fig07_15to17：
困惑：Dev-C++有问题？



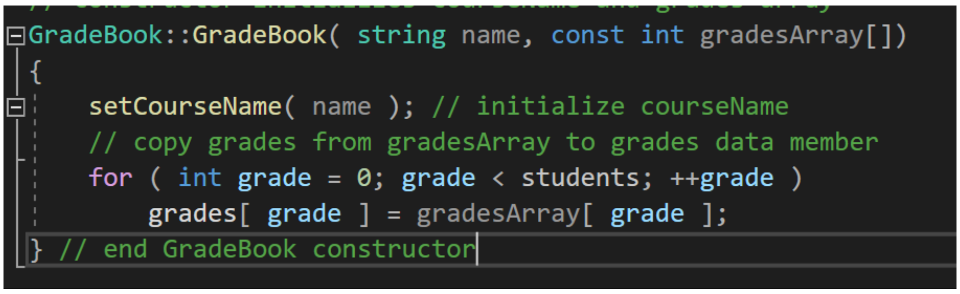
```
Student  2:  68
Student  3:  94
Student  4:  100
Student  5:  83
Student  6:  78
Student  7:  85
Student  2:  68
Student  3:  94
Student  4:  100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student  10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100

Grade distribution:
 0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
```

用 VS：



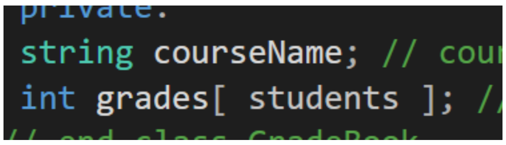
```
Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100

Grade distribution:
  0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

正常显示。
（? ? ? ? ? ? ? ）
解释：从 fig07_17.cpp 的 main 函数开始：
GradeBook myGradeBook("CS101 Introduction to C++ Programming", gradesArray );
　　这一行调用了复制构造函数：



```
GradeBook::GradeBook( string name, const int gradesArray[])
{
    setCourseName( name ); // initialize courseName
    // copy grades from gradesArray to grades data member
    for ( int grade = 0; grade < students; ++grade )
        grades[ grade ] = gradesArray[ grade ];
} // end GradeBook constructor
```

使 private 变量赋值：



```
private:
string courseName; // cou
int grades[ students ]; /
// end class GradeBook
```

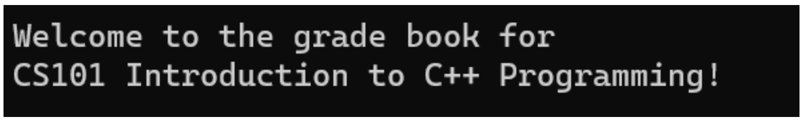
随后调用 display message：

```
void GradeBook::displayMessage()
{
    // this statement calls getCourseName to get the
    // name of the course this GradeBook represents
    cout << "Welcome to the grade book for\n" << getCourseName() << "!"
        << endl;
  } // end function displayMessage
```

中间夹杂调用 getcoursename：

```
string GradeBook::getCourseName()
{
     return courseName;
  } // end function getCourseName
```

显示出这一行：



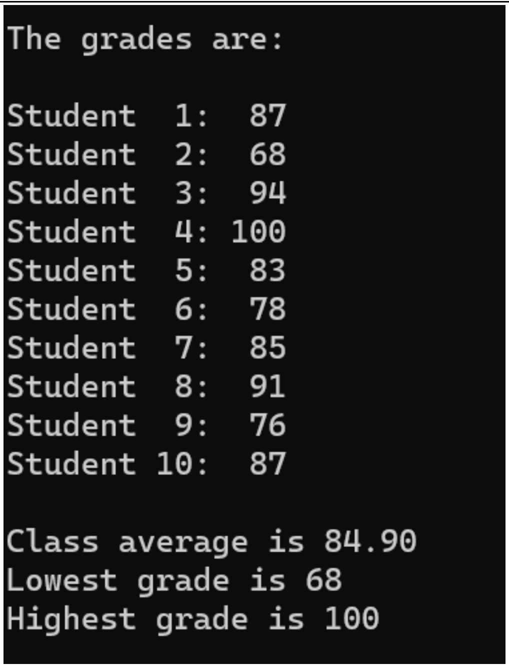
```
Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

随后调用：

```
void GradeBook::processGrades()
{
    outputGrades(); // output grades array

// display average of all grades and minimum and maximum grades
    cout << "\nClass average is " << setprecision( 2 ) << fixed <<
    getAverage() << "\nLowest grade is " << getMinimum() <<
        "\nHighest grade is " << getMaximum() << endl;

    outputBarChart(); // print grade distribution chart
  } // end function processGrades
```

其中，依次调用 getAverage()，getMinimum()，getMaximum()函数接口显示相应的数值：

```
int GradeBook::getMinimum()
{
    int lowGrade = 100; // assume lowest grade is 100

    // loop through grades array
    for ( int grade = 0; grade < students; ++grade )
    {
        // if current grade lower than lowGrade, assign it to lowGrade
        if ( grades[ grade ] < lowGrade )
            lowGrade = grades[ grade ]; // new lowest grade
    } // end for

    return lowGrade; // return lowest grade
} // end function getMinimum

// find maximum grade
int GradeBook::getMaximum()
{
    int highGrade = 0; // assume highest grade is 0

    // loop through grades array
    for ( int grade = 0; grade < students; ++grade )
    {
        // if current grade higher than highGrade, assign it to highGrade
        if ( grades[ grade ] > highGrade )
            highGrade = grades[ grade ]; // new highest grade
    } // end for

    return highGrade; // return highest grade
} // end function getMaximum

// determine average grade for test
double GradeBook::getAverage()
{
    int total = 0; // initialize total

    // sum grades in array
    for ( int grade = 0; grade < students; ++grade )
        total += grades[ grade ];

    // return average of grades
    return static_cast< double >( total ) / students;
  } // end function getAverage
```
**依次对应：**



```
The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100
```

最后调用 outputBarChart()：

根据其内容显示：

```
cout << "\nGrade distribution:" << endl;
```

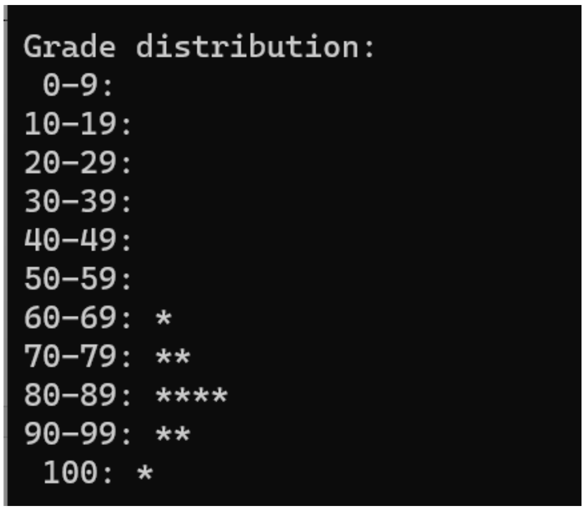
对应：

```
Grade distribution:
```

随后一段

```
for ( int count = 0; count < frequencySize; ++count )
{
    // output bar labels ("0-9:", ..., "90-99:", "100:" )
    if ( count == 0 )
        cout << " 0-9: ";
    else if ( count == 10 )
        cout << " 100: ";
    else
        cout << count * 10 << "-" << ( count * 10 ) + 9 << ": ";

    // print bar of asterisks
    for ( int stars = 0; stars < frequency[ count ]; ++stars )
        cout << '*';

    cout << endl; // start a new line of output
  }
```

则对应：

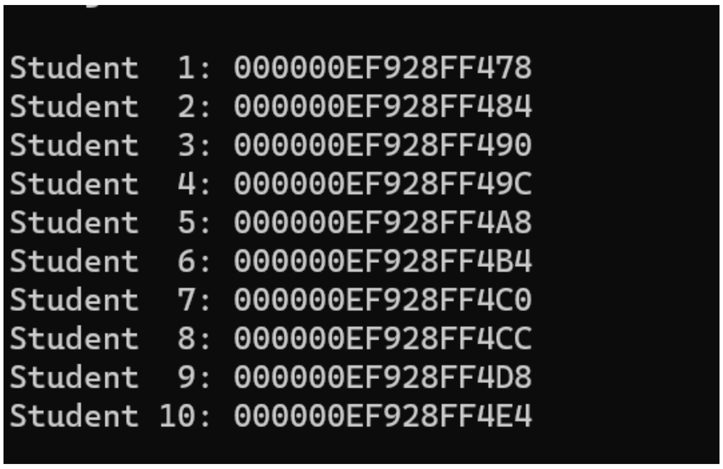


```
Grade distribution:
  0-9:
 10-19:
 20-29:
 30-39:
 40-49:
 50-59:
 60-69: *
 70-79: **
 80-89: ****
 90-99: **
  100: *
```

至此程序结束。

对于 fig07_22to24:

原结果:



```
Student  1: 000000EF928FF478
Student  2: 000000EF928FF484
Student  3: 000000EF928FF490
Student  4: 000000EF928FF49C
Student  5: 000000EF928FF4A8
Student  6: 000000EF928FF4B4
Student  7: 000000EF928FF4C0
Student  8: 000000EF928FF4CC
Student  9: 000000EF928FF4D8
Student 10: 000000EF928FF4E4
```

显然这里是有问题的。分析发现:

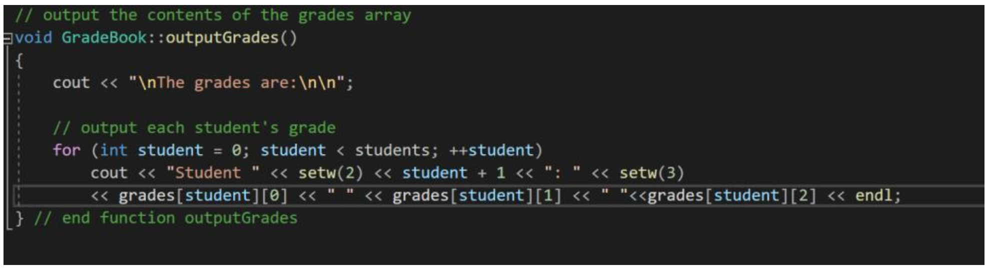
在 outputGrades 中:

```
void GradeBook::outputGrades()
{
    cout << "\nThe grades are:\n\n";

    // output each student's grade
    for ( int student = 0; student < students; ++student )
        cout << "Student " << setw( 2 ) << student + 1 << ": " << setw( 3 )
            << grades[ student ] << endl;
  } // end function outputGrades
```

输出的是 grades[ student ], 是地址, 不是数值。

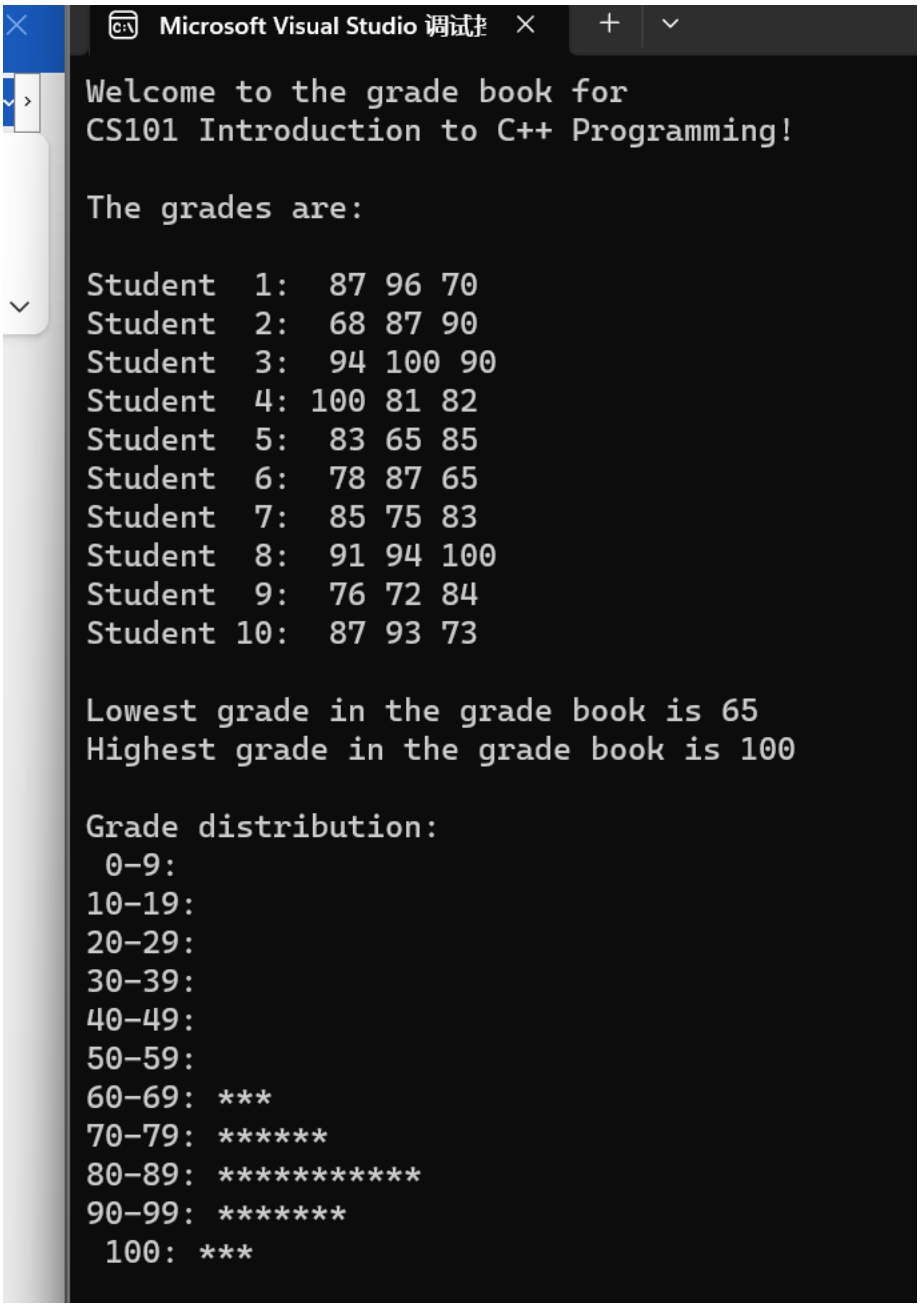
更改后的程序:

其他处的分析与第一个项目类似，省略。



```cpp
// output the contents of the grades array
void GradeBook::outputGrades()
{
    cout << "\nThe grades are:\n\n";

    // output each student's grade
    for (int student = 0; student < students; ++student)
        cout << "Student " << setw(2) << student + 1 << ": " << setw(3)
        << grades[student][0] << " " << grades[student][1] << " "<<grades[student][2] << endl;
} // end function outputGrades
```

结果：



```
Welcome to the grade book for
CS101 Introduction to C++ Programming!

The grades are:

Student  1:  87 96 70
Student  2:  68 87 90
Student  3:  94 100 90
Student  4: 100 81 82
Student  5:  83 65 85
Student  6:  78 87 65
Student  7:  85 75 83
Student  8:  91 94 100
Student  9:  76 72 84
Student 10:  87 93 73

Lowest grade in the grade book is 65
Highest grade in the grade book is 100

Grade distribution:
 0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: ******
80-89: ***********
90-99: *******
 100: ***
```

其他处的分析与第一个项目类似，省略。

7. 略。

结论分析与体会：
非常好的实验，使我感受到指针的魅力和 new 的伟大，以及面向对象的程序的井井有条。