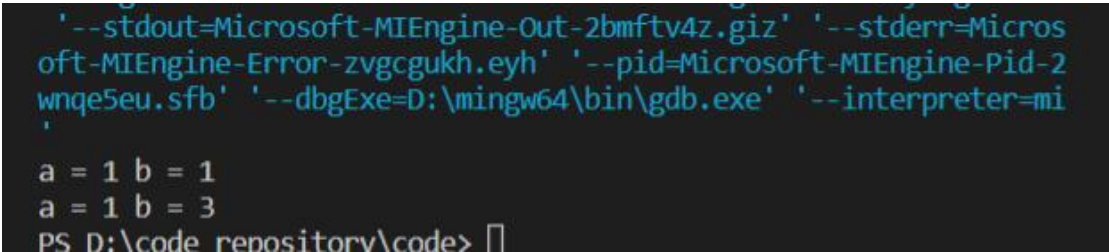
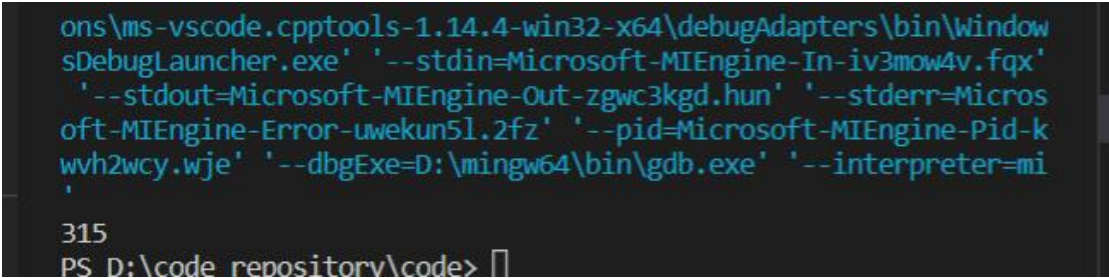


计算机学院 高级语言程序设计 课程实验报告

实验题目：联合体、数据共享与保护		学号：202200130048
日期：2023. 3. 17	班级： 6	姓名： 陈静雯
Email：1205037094@qq. com		
<p>实验步骤与内容：</p> <ol style="list-style-type: none">1. 作用域与生命周期2. 类的静态成员3. 友元4. const 保护<ol style="list-style-type: none">(1) 常对象(2) 常成员(3) 常引用(4) 常指针		
<p>结论分析与体会：</p> <ol style="list-style-type: none">1. (1)<pre>'--stdout=Microsoft-MIEngine-Out-2bmftv4z.giz' '--stderr=Microsoft-MIEngine-Error-zvgcgukh.eyh' '--pid=Microsoft-MIEngine-Pid-2wnqe5eu.sfb' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi' a = 1 b = 1 a = 1 b = 3 PS D:\code_repository\code> █</pre> <p>分析：a=1, b=1 是在 main 函数，b=3 作用域是在 {} 里，此时 {} 外的 b 对 {} 里的不可见，所以 {} 中输出 b 是 3</p> <ol style="list-style-type: none">1. (2)<p>预测：pt 指向的地址未知，一开始定义时没有初始化 pt 指针，{} 里 pt 指向 a 的地址，a 的作用域和生命周期都只有 {}，{} 运行结束后系统会回收分配给 a 的地址，所以 pt 指向的不存在</p><p>结果：</p><pre>ons\ms-vscode.cpptools-1.14.4-win32-x64\debugAdapters\bin\Window sDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-iv3mow4v.fqx' '--stdout=Microsoft-MIEngine-Out-zgwc3kgd.hun' '--stderr=Micros oft-MIEngine-Error-uwekun5l.2fz' '--pid=Microsoft-MIEngine-Pid-k wvh2wcy.wje' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi' 315 PS D:\code_repository\code> █</pre>		

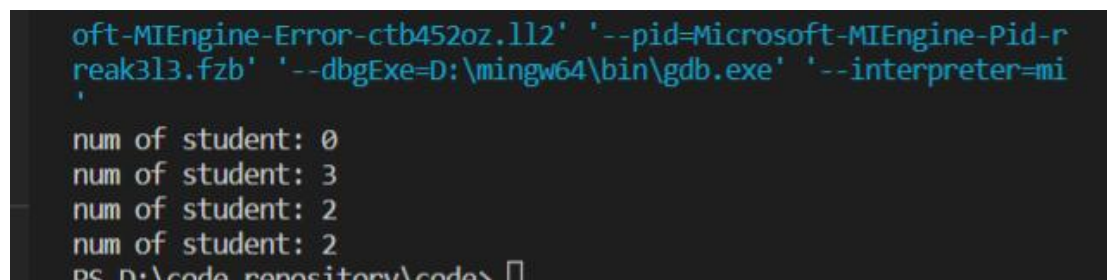
原因：可能是这个地址不分配给 a，但是它还是存在，就是 int a 给 a 分配地址，初始化为 315，则这个空间存储的就是 315，虽然 {} 运行结束，a 的生存周期结束，但是这个存储空间里的值没有被回收，所以 pt 指向这个地址还是 315

2. (1)

代码：

```
#include <iostream>
using namespace std;
class Student{
public:
    Student(int x){
        stid=x;
        numofstudent++;
    };
    void print();
    static void printNumOfStudents();
    ~Student(){
        numofstudent--;
    };
private:
    int stid;
    static int numofstudent;
};
int Student::numofstudent=0;
void Student::print(){
    cout<<"stid is: "<<stid<<endl;
    cout<<"num of student: "<<numofstudent<<endl;
}
void Student::printNumOfStudents(){
    cout<<"num of student: "<<numofstudent<<endl;
}
}
```

结果：



```
oft-MIEngine-Error-ctb452oz.ll2' '--pid=Microsoft-MIEngine-Pid-r
reak3l3.fzb' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi
,
num of student: 0
num of student: 3
num of student: 2
num of student: 2
PS D:\code repository\code>
```

分析：{} 中构造了一个 s3，{} 结束后 s3 生存周期结束，会调用析构函数使总数减 1，静态函数 printNumOfStudents 有两种调用方法，一个通过类名，一个通过类中的对象

2. (2)

```
num of student: 2
num of student: 2
0x61fe0c 0x407030
0x61fe08 0x407030
PS D:\code_repository\codes>
```

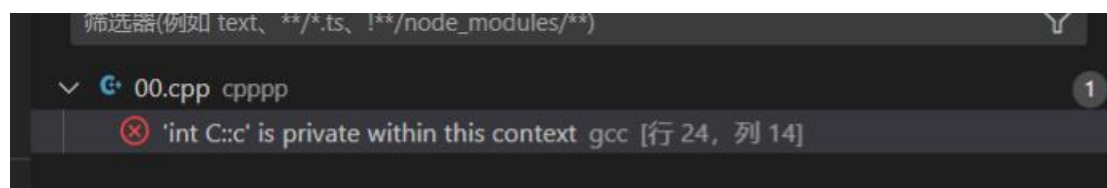
分析: stid 的地址 s1 和 s2 不同, 而 numOfStudents 是对于整个类来说的, 是静态成员, 地址不变

3.

代码:

```
#include <iostream>
using namespace std;
class B{
public:
    friend class A;
private:
    int b=1;
};
class C{
public:
    friend class B;
private:
    int c=2;
};
class A{
public:
    A(){};
    void show();
private:
    int a=0;
    C cc;
};
void A::show() {
    cout<<cc.c<<endl;
}
int main() {
    A aa;
    aa.show();
}
```

结果: 编译报错, A 不能访问 C 的私有数据

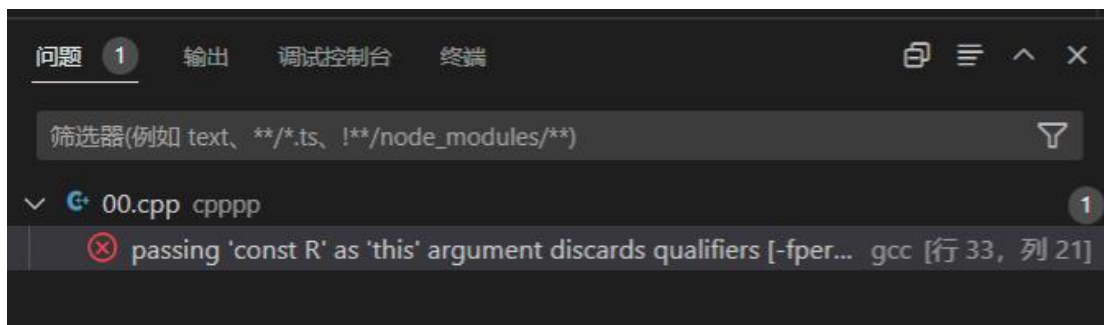


4. 1

```
jsymdar.tvq --dbgExe=D:\mingw64\bin\gdb.exe --interpreter=mi
5:4
20:52
PS D:\code_repository\code>
```

分析：a. print 调用第一个，b 调用第二个即常函数，b 定义的是常对象

4. 2



分析：a 可以调用，b 调用编译器会报错，b 定义的是常对象，不能修改对象值，这个错误大概意思就是会丢失 const 这个限定词，即调用函数可能会使常对象值改变，所以报错

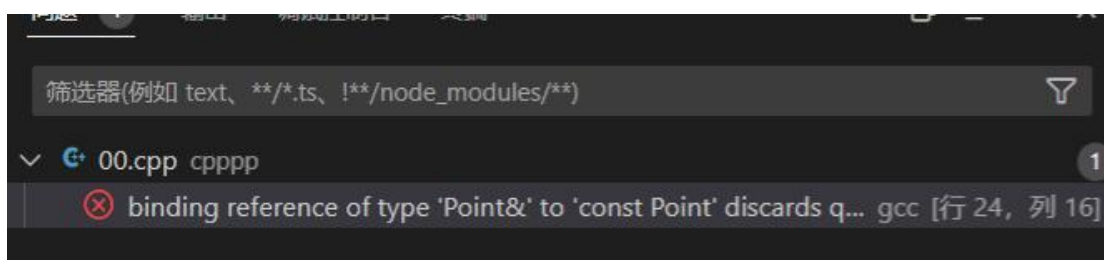
4. 2



分析：a 是常数据成员，A::A(int i) { a=i; } 不能初始化 a，A::A(int i) : a(i) { } 常数据成员只能通过初始化列表来初始化

4. 3

(1)



分析：报错，因为 myp1, mpy2 是常对象，而函数形参不是常引用

(2)

```
'--stdout=Microsoft-MIEngine-Out-m5klek0y.00u' '--stderr=Microsoft-MIEngine-Error-smry1xpt.ico' '--pid=Microsoft-MIEngine-Pid-gn1tpewz.gom' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
The distance is: 5
PS D:\code_repository\codes>
```

分析：正常运行，应该是 myp1 和 myp2 没有 const 保护，调用函数对 myp 的值修改或不修改都可以，所以编译器没有报错，将 myp 当常对象形参来用

4.4

```
int main() {
    int a = 10;
    int b = 20;
    const int *ap1 = &a;
    int * const ap2 = &a;
    *ap1 = 10;
    ap1 = &b;
    *ap2 = 10;
    ap2 = &b;
    return 0;
}
```



分析：const int *ap1 指 ap1 这个地址里的值不能修改，int * const ap2 指针指向的地址不能改变。所以对 *ap1 = 10 错误信息是说 location '*ap1' 即 ap1 地址是只读的，而 ap2 = &b 是说 ap2 这个指针是只读的