计算机学院 高级程序语言设计 课程实验报告

实验题目：联合体、数据共享与保护		学号：202300130150
日期：2024.3.26	班级： 4	姓名： 王成意

实验目的：

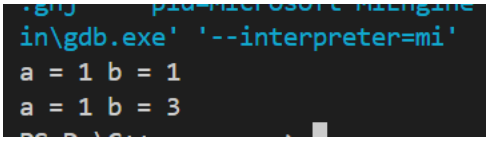
1. 练习标识符的不同作用域和生命周期，对象的静态、动态生存期在编程中的应用
2. 练习类静态（static）成员的使用
3. 练习友元：友元类
4. 练习 const：常对象、常成员、常引用、常指针

实验步骤与内容：

1. 作用域与生命周期：

（1）运行以下程序段，分析打印结果。

结果：

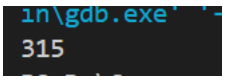


分析：先正常输出，b=3 在第二个方括号内生效，输出时 b 变成了 3.

（2）运行以下程序段。该程序段通过 pt 获取 a 的地址。根据你对作用域与生命周期的理解，描述一下运行程序前后你对打印结果的预期，并尝试回答为什么输出结果是这样的。

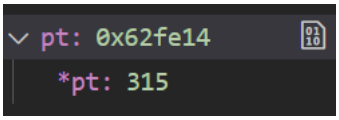
预期：315

结果：



原因：

a 的作用域在其所在的最小方括号里，a=315 是 a 对应的地址有个 315 值，这时候 pt 指向了此地址，所以*pt=315.



设计一个学生类，要求：

1. 需要有以下私有成员数据：学号（int stid），学生总数（numOfStudents int）。其中学生总数是静态数据。
2. 还需要有以下函数：print()打印学生信息。包括学号以及当前学生总数。printNumOfStudents()返回当前学生总数。
3. 有唯一的构造函数 Student(int stid)实现对学号的初始化。
4. 学生总数需要被初始化为 0
5. 每实例化一个学生对象需要将学生总数加一，每销毁（析构）一个学生对象需要将学生总数减一。

设计代码：

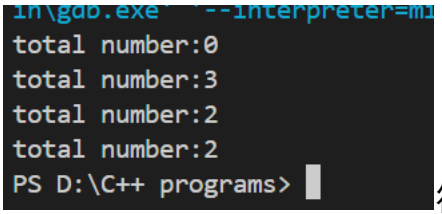
```
class student{
    private:
        static int tot;
        int id;
    public:
        static void gettot();
        ~student();
        student(int x=0)
        {
            this->id = x;
            tot++;
        }
        void print();
};
student::~student()
{
    tot--;
}
void student::gettot()
{
    printf("total number:%d\n", tot);
}
```

```
void student::print()
{
    printf("id:%d\ntot number:%d\n", this->id, tot);
}
int student::tot = 0;
```

原主函数在这里应该是：

```
student::gettot();
    student s1(0);
    student s2(1);
    {
        student s3(2);
        student::gettot();
    }
    student::gettot();
    s1.gettot();
    return 0;
```

分析结果：



符合预期，小括号中的 s3 在括号完成之后执行了析构函数，使得 num-1.

（2）在类中添加以下函数，在 main 函数中让 s1 和 s2 分别调用这个函数，截图并分析结果

结果：

```
in\gdb.exe' '--interpreter=mi'
total number:0
total number:3
total number:2
total number:2
0x62fe0c 0x40c030
0x62fe08 0x40c030
PS D:\C++ programs>
```

分析可知，static 类型的变量是 class 共享的，id（private）是对象独有的。

代码：

```cpp
#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

#define ll long long

using namespace std;

class a;

class b;

class c;

class a{

    private:

        int aa;

    public:

        a() = default;

        a(a &x);

        void change(c &y);
```

```
};
a::a(a &x) : aa(x.aa) {}
class b{
    friend class a;

private:
    int bb;

public:
    b() = default;
    b(b &x);
};
b::b(b &x) : bb(x.bb) {}
class c{
    friend class b;

private:
    int cc;

public:
    c() = default;
    c(c &x);
```

```
};

c::c(c &x) : cc(x.cc) {}

void a::change(c &y)

{

    y.c = 0;

}

signed main()

{



    return 0;

}
```

结果：

```
D:\C++ programs\1.cpp: In member function 'void a::change(c&)':
D:\C++ programs\1.cpp:43:4: error: 'int c::cc' is private within this c
ntext
  y.cc = 0;
    ^~
D:\C++ programs\1.cpp:34:6: note: declared private here
  int cc;
      ^~
```

答案是不行。

这样改就行：

```
class c{
    friend class a;

private:
```

4.1 常对象：

（1）测试上面这段代码，两次调用的 print 分别是调用的哪个函数？

```
5:4
20;52
```

第一次调用的是普通函数，第二次调用的是常函数。

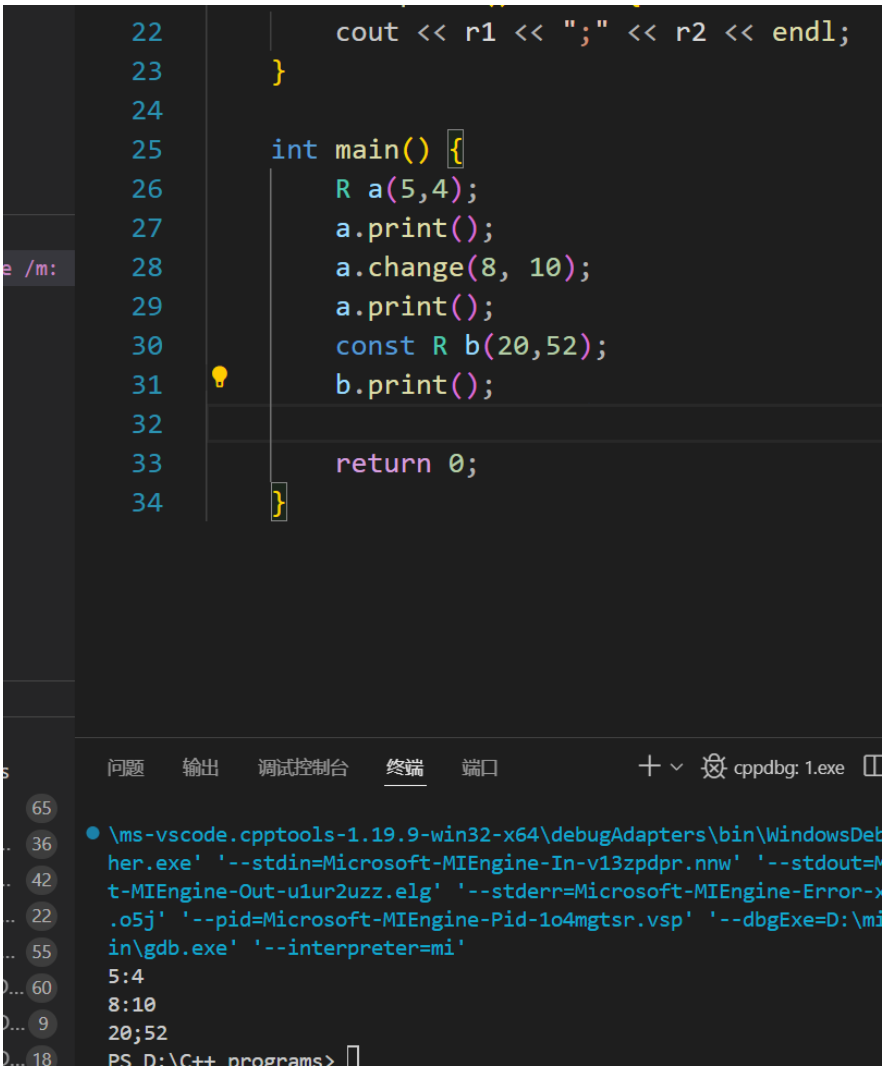
（2）在 R 类中增加一个成员函数 change()如下，尝试用 a 对象和 b 对象分别调用它可以吗？为什么？

不可以，会出现报错：

```
D:\C++ programs\1.cpp:32:16: error: passing 'const R' as 'this' argument
 discards qualifiers [-fpermissive]
   b.change(1, 1);
               ^
D:\C++ programs\1.cpp:9:8: note:   in call to 'void R::change(int, int)'
   void change(int x1,int x2){
        ^~~~~~
```

因为 change 函数改变了私有成员，而 b 是常量不能改变。

删除了就正确：



4.2 常成员

（1） 结合以上程序，将 A::A(int i) : a(i) { } 语句 改为 A::A(int i) :{ a=i; }可以吗？为什么？

不可以，因为 a 作为常量没有在一开始赋初值。（this->a 没有初始值。）

其次多了“:”

```
D:\C++ programs\1.cpp:16:16: error: expected identifier before '{' token
  A::A(int i) : { a=i;}
                ^
D:\C++ programs\1.cpp:16:2: error: uninitialized const member in 'const
int' [-fpermissive]
  A::A(int i) : { a=i;}
  ^
D:\C++ programs\1.cpp:9:13: note: 'const int A::a' should be initialized
   const int a;
             ^
D:\C++ programs\1.cpp:16:20: error: assignment of read-only member 'A::a
'
  A::A(int i) : { a=i;}
                    ^
```

删除“:”

```
D:\C++ programs\1.cpp:16:2: error: uninitialized const member in 'const
int' [-fpermissive]
  A::A(int i)  { a=i;}
  ^
D:\C++ programs\1.cpp:9:13: note: 'const int A::a' should be initialized
   const int a;
             ^
D:\C++ programs\1.cpp:16:19: error: assignment of read-only member 'A::a
'
  A::A(int i)  { a=i;}
                   ^
```

4.3 常引用

```
in\gdb.exe' '--interpre
The distance is: 5
```

（1） 在以上程序基础上，尝试将友元函数 dist(const Point &p1, const Point &p2) 的两处声明和类外定义的形参都取消 const 可以吗？为什么？

不可以，因为在 24 行传参时传入的是常量。

```
s -g "D:\C++ programs\1.cpp" -o "D:\C++ programs\1.exe" -std=c++17
:\C++ programs\1.cpp: In function 'int main()':
:\C++ programs\1.cpp:24:16: error: binding reference of type 'Point&' t
 'const Point' discards qualifiers
   cout << dist(myp1, myp2) << endl; //计算两点间的距离
                ^~~~
:\C++ programs\1.cpp:15:8: note:   initializing argument 1 of 'float di
st(Point&, Point&)'
 float dist( Point &p1,  Point &p2) { //常引用作形参
       ^~~~
```

（2） 在以上程序基础上，尝试将 main()函数中 const Point myp1(1, 1), myp2(4, 5)；语句中的 const 取消，程序能正常运行吗？为什么？

```
20
21      int main() {    //主函数
22          Point myp1(1, 1), myp2(4, 5);   /
23          cout << "The distance is: ";
24          cout << dist(myp1, myp2) << endl;
25          return 0;
26      }
```

```
题    输出    调试控制台    终端    端口        cppdbg: 1.exe

S D:\C++ programs>
S D:\C++ programs>  & 'd:\VsCode-extentions\.vscode-insiders\ex
ns-vscode.cpptools-1.19.9-win32-x64\debugAdapters\bin\WindowsDe
er.exe' '--stdin=Microsoft-MIEngine-In-324zjq1w.t4y' '--stdout=
-MIEngine-Out-v4g3dfsa.cfp' '--stderr=Microsoft-MIEngine-Error-
ffc' '--pid=Microsoft-MIEngine-Pid-ofdd5nvi.jv5' '--dbgExe=D:\m
n\gdb.exe' '--interpreter=mi'
he distance is: 5
S D:\C++ programs>
```

可以正常运行。

函数中 const 只保证传入的值不会在函数内改变，而传入的值确实没有改变，若这样就会报错：

```
float dist(const Point &p1, const Point &p2) {  //常引
    double x = p1.x - p2.x;
    p1.x--;
    double y = p1.y - p2.y;
    return static_cast<float>(sqrt(x * x + y * y));
}
```

```
cmd /c chcp 65001>nul && D:\mingw64\bin\g++.exe -fdiagnostics-color=always -g "D:\C++ pro
D:\C++ programs\1.cpp: In function 'float dist(const Point&, const Point&)':
D:\C++ programs\1.cpp:17:7: error: decrement of member 'Point::x' in read-only object
   p1.x--;
       ^~
```

4.4 常指针：

请测试哪行代码会出现代码错误？

10，13

```
    const int *ap1 =
    int * const ap2
    *ap1 = 10;
    ap1 = &b;
    *ap2 = 10;
    ap2 = &b;
    return 0;
}
```

const int* a:
const 与 int 结合，因此变量 a 是一个指向常量整型的指针。
不能通过修改*ap1 更改 a。
int* const a
const 与*结合，因此变量 a 是一个指向整型的常量指针
因此 ap2 不可更改，报错。

结论分析与体会：
此次试验意义深刻，让我更好的了解了 const，friend，指针，作用域。