实验： 研究C++的对象模型

李雨轩

计算机2205

2204112913

一、实验要求

1. 定义一个类，其中有静态数据成员、各种类型非静态数据成员（含字符指针），甚至包括引用（可选），静态和非静态成员函数（含分配空间的构造函数、析构函数）。
2. 定义全局对象、main函数中局部对象、另一个被main调用的外部函数func中定义局部对象（可以是形参）、main函数中动态创建对象，每种对象至少2个。观察、分析各种对象地址。
3. 输出对象中各个静态与非静态数据成员的值、地址、对象的存储空间大小等信息。由此理解对象的本质、静态数据成员是本类对象共享一份拷贝等问题。此外，应观察对齐现象。
4. （可选）输出对象的每个字节，以揭示引用的实现方法。
5. 对于上述各种对象，输出静态、非静态成员函数地址，以及main、func等外部函数的地址，并分析。要求采用合理方法，避免编译器提出警告。

   注意：本题作为实验报告内容，要求有代码、注释、结果截图及分析。以班为单位统一收，电子版，发我的邮箱 libaohong32@163.com

二、问题分析与实验过程

为了研究C++的对象模型，我设计了下面用于研究C++对象模型的程序，它展示了类的构造、拷贝构造、析构函数、静态成员函数、非静态成员函数以及对象的动态分配和释放等方面的特性。**（源代码见附页）**

1. 定义一个类：

定义了一个名为 `MyClass` 的类，该类包含以下内容：

- 静态数据成员 `staticData`
- 非静态数据成员：

    - 整型数据 `integerData`
    - 双精度浮点型数据 `doubleData`
    - 字符指针 `charPtrData`
    - 字符串类型数据 `stringData`
- 引用数据成员 `refData`
- 静态成员函数 `staticFunction()`
- 非静态成员函数 `nonStaticFunction()`
- 构造函数和析构函数，构造函数需要动态分配内存来初始化字符指针 `charPtrData`。

```
class MyClass {
public:
    static int staticData;
    int integerData;
    double doubleData;
    char* charPtrData;
    std::string stringData;
    int& refData;

public:
    MyClass(int intValue, double doubleValue, const char* charValue, const std::string&
stringValue, int& ref);

    MyClass(const MyClass& obj)
        : integerData(obj.integerData), doubleData(obj.doubleData), stringData(obj.stringData),
refData(obj.refData);

    ~MyClass();

    static void staticFunction();

    void nonStaticFunction();
};
```

2. 创建各种类型的对象：

- 全局对象 `globalObj1` 和 `globalObj2`
- `main` 函数中的局部对象 `localObj1` 和 `localObj2`
- 外部函数 `func` 中的局部对象 `funcObj`
- `main` 函数中动态创建的对象 `dynamicObj1` 和 `dynamicObj2`

3. 输出对象信息：

- 输出各个对象中静态和非静态数据成员的值、地址以及对象的存储空间大小。
- 观察静态数据成员的共享情况，即不同对象的静态数据成员是否共享同一份拷贝。
- 观察对象的存储空间布局，并分析可能出现的对齐现象。

下面是相应的代码和输出：

```
// 在类外初始化静态数据成员
int MyClass::staticData = -1010;

void func(MyClass obj) {
    // 外部函数中的局部对象
    obj.nonStaticFunction();
}

// 全局对象
MyClass globalObj1(1, 2.3, "Global 1", "Object", MyClass::staticData);
MyClass globalObj2(2, 3.7, "Global 2", "Object", MyClass::staticData);

int main() {

    // main 函数中的局部对象
    MyClass localObj1(3, 4.9, "Local  1", "Object", MyClass::staticData);
    MyClass localObj2(4, 5.1, "Local  2", "Object", MyClass::staticData);

    // 调用外部函数 func()，传递局部对象
    funcObj(5, 6.3, "Func Object", "Object", MyClass::staticData);

    // 动态创建对象
    MyClass* dynamicObj1 = new MyClass(6, 7.2, "Dynamic1", "Object", MyClass::staticData);
    MyClass* dynamicObj2 = new MyClass(7, 8.5, "Dynamic2", "Object", MyClass::staticData);

    std::cout << "\nDetails of localObj1:" << std::endl;
    // 输出 localObj1 的信息...

    std::cout << "\nDetails of funcObj:" << std::endl;
    // 输出 funcObj 的信息...

    std::cout << "\nDetails of dynamicObj1:" << std::endl;
    // 输出 dynamicObj1 的信息...

    // 释放动态分配的对象
    delete dynamicObj1;
    delete dynamicObj2;

    return 0;
}
```

输出结果会显示每个对象的静态和非静态数据成员的值、地址以及对象的存储空间大小。通过观察这些输出，我们可以理解C++对象模型中对象的本质、静态数据成员的共享情况，以及对齐现象。

三、输出结果说明与分析

（输出结果见附页）

1. 全局对象的构造和析构：

```
++++++ Construct Global 1 is created ++++++
++++++ Construct Global 2 is created ++++++
```

这两行显示了全局对象 `globalObj1` 和 `globalObj2` 的构造。同时显示在`main`函数之前，说明全局对象在`main`函数执行前就已经创建。

2. main 函数的开始和结束标记：

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ MAIN FUNCTION BEGIN:
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ MAIN FUNCTION END
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

这两行表示了 `main` 函数的开始和结束。

3. 对象的构造和析构：

```
++++++ Construct Local  1 is created ++++++
++++++ Construct Local  2 is created ++++++
```

这两行显示了局部对象 `localObj1` 和 `localObj2` 的构造。

```
++++++ Destruct Local  2 is deleted ++++++
++++++ Destruct Local  1 is deleted ++++++
```

这两行显示了局部对象 `localObj2` 和 `localObj1` 的析构。构造和析构顺序符合对象的创建和销毁顺序。

```
++++++ Construct Func Object is created ++++++
```

这行显示了外部函数 `func` 中的局部对象 `funcObj` 的构造。

```
++++++ Destruct Func Object is deleted ++++++
```

这行显示了外部函数 `func` 中的局部对象 `funcObj` 的析构。

```
++++++ Construct Dynamic1 is created ++++++
++++++ Construct Dynamic2 is created ++++++
```

这两行显示了动态创建的对象 `dynamicObj1` 和 `dynamicObj2` 的构造。

```
++++++ Destruct Dynamic1 is deleted ++++++
++++++ Destruct Dynamic2 is deleted ++++++
```

这两行显示了动态创建的对象 `dynamicObj1` 和 `dynamicObj2` 的析构。

4. 对象的静态和非静态数据成员的值、地址和存储空间：

1. **静态数据成员地址**：
   - 静态数据成员 `staticData` 在内存中只有一份拷贝，无论有多少个 `MyClass` 对象被创建，它们都共享同一份静态数据成员。
   - 静态数据成员被定义为 `static int staticData`。因此，无论是通过类名还是对象名来访问该静态成员，都指向同一个地址。
2. **非静态数据成员地址**：
   - 每个对象的非静态数据成员都有自己的地址，即使两个对象的数据成员值相同，它们的地址也不同。
   - 输出中显示了全局对象 `globalObj1` 和 `globalObj2` 的地址不同，即使它们具有相同的数据成员值，但它们的地址不同。对于局部对象和动态创建的对象，同样也可以观察到它们的地址不同，这表明它们的非静态数据成员在内存中占据不同的位置。
3. **全局对象 globalObj1 和 globalObj2**：
   - 这些对象的静态数据成员 `staticData` 在内存中只有一份拷贝，由于是全局对象，它们的地址位于静态存储区。
   - 其非静态数据成员（如 `integerData`、`doubleData`、`charPtrData`、`stringData` 等）分别存储在这些对象所在的内存空间中。
4. **局部对象 localObj1 和 localObj2**：
   - 这些对象的静态数据成员 `staticData` 在内存中仍然只有一份拷贝，但由于是局部对象，它们的地址位于栈上。
   - 同样，其非静态数据成员也存储在对象所在的内存空间中，这些对象的地址位于栈上。（`0x7FFFFFF...`）
5. **动态创建的对象 dynamicObj1 和 dynamicObj2**：
   - 与局部对象类似，这些对象的静态数据成员 `staticData` 仍然共享一份拷贝，但由于是动态创建的，它们的地址位于堆上。
   - 非静态数据成员也存储在对象所在的堆上的内存空间中。

四、结论总述

- `globalObj1` 和 `globalObj2` 是在全局作用域下创建的对象，它们在程序启动时被创建，构造函数被调用，输出了相关的构造信息。在程序结束时，析构函数被调用，释放了相应的资源。

- `localObj1` 和 `localObj2` 是在 `main()` 函数中定义的局部对象，它们在 `main()` 函数执行时创建，在 `main()` 函数结束时销毁。在创建时，输出了构造函数的信息，在销毁时，输出了析构函数的信息。

- `dynamicObj1` 和 `dynamicObj2` 是通过 `new` 运算符在堆上分配内存创建的对象。它们在堆上分配了内存，需要手动释放。在创建时，输出了构造函数的信息，在释放内存时，输出了析构函数的信息。
- 输出了非静态成员函数和静态成员函数的地址。可以观察到这些函数在内存中的位置。非静态成员函数的地址和静态成员函数的地址是固定的，不依赖于对象的实例。
- 通过输出对象的大小，观察到了对象在内存中的布局情况。可以发现对象的大小并不等于所有数据成员的大小之和，这是因为编译器对对象进行了对齐操作。这种对齐操作是为了提高访问速度和内存访问的效率。

- 通过观察对象的构造和析构函数的调用，可以了解对象的生命周期。全局对象在程序启动时被创建，在程序结束时被销毁；局部对象在函数执行时创建，在函数结束时销毁；动态对象则由程序员手动创建和销毁。