# CPP第一次实验报告

计算机2101 陈实 2215015058

## 实验平台

1. IDE：CLion：2024.1
2. C++标准：C++17

## 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，不要求），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。
> 4. 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的地址，并分析。
> 5. 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>

## 实验代码

```cpp
#include <cstring>
#include <iostream>
#include <iomanip> // 包含设置输出字段宽度的头文件
using namespace std;

class Student
{
    public:
        static int count;
        char *name;
        int age;
        double score;
        bool & pass;
        //构造函数
        Student(char *n, int a, double s, bool &pass) : pass(pass) {
            name = new char[strlen(n)+1];
            strcpy(name, n);
            age = a;
            score = s;
            count++;
        }
        //析构函数
        ~Student(){
            delete []name;
            count--;
```

```
        }
        //静态成员函数
        static void printCount(){
            cout << "count = " << count << endl;
        }

        //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
        void print(){
            cout << "This is " << setw(5) << name << "  address = " << this << " size: "<<sizeof(this) << endl;
            cout << "name = " << setw(6) << name << "  address = " << &name << " size: "<<sizeof(name) << " value = " << (void *)name  << endl;
            cout << "age = " << setw(7) << age << "  address = " << &age << " size: "<<sizeof(age) << " value = " << age << endl;
            cout << "score = " << setw(5) << score << "  address = " << &score << " size: "<<sizeof(score) << " value = " << score << endl;
            cout << "pass = " << setw(6) << pass << "  address = " << &pass << " size: "<<sizeof(pass) << " value = " << pass << endl;
            cout << "count = " << setw(5) << count << "  address = " << &count << " size: "<<sizeof(count) << " value = " << count << endl;
            cout << "size = " << sizeof(*this) << endl;
            cout << "*name = " << (void *)name << endl;
            cout << "-----------------" << endl;
        }
};

//全局对象

int Student::count = 0;                 //静态数据成员初始化
bool pass = true;                       //引用初始化
bool fail = false;                      //引用初始化
Student global_stu1("stu1", 20, 90, pass);      //全局对象1
Student global_stu2("stu2", 21, 80, fail);      //全局对象2

void func(){
    Student func_stu("stu7", 26, 30, fail);
    func_stu.print();
    Student func_stu1("stu8", 26, 30, fail);
    func_stu1.print();
}

int main(){
    //局部对象
    Student local_stu1("stu3", 22, 70, pass);    //局部对象1
    Student local_stu2("stu4", 23, 60, fail);    //局部对象2

    //动态创建对象
    Student *p1 = new Student("stu5", 24, 50, pass);    //动态创建对象1
    Student *p2 = new Student("stu6", 25, 40, fail);    //动态创建对象2

    //创建数组存储全部对象
    Student *p[6] = {&global_stu1, &global_stu2, &local_stu1, &local_stu2, p1, p2};

```

```
73
74      //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
75      global_stu1.print();
76      global_stu2.print();
77      local_stu1.print();
78      local_stu2.print();
79      p1->print();
80      p2->print();
81      func();
82      //输出静态非静态成员函数地址，以及main、func的地址
83      for (int i = 0; i < 6; i++) {
84          cout <<"----------------" << endl;
85          cout << p[i]->name << endl;
86
87          // 输出静态成员函数地址
88          cout << "printCount address = " << reinterpret_cast<void*>(&p[i]-
    >printCount) << endl;
89          // 输出非静态成员函数地址
90          cout << "print address = " << reinterpret_cast<void*>(&p[i]-
    >print) << endl;
91      }
92      cout <<"----------------" << endl;
93      cout << "main address = " << reinterpret_cast<void*>(&main) << endl;
94      cout << "func address = " << reinterpret_cast<void*>(&func) << endl;
95      cout << "----------------" << endl;
96      //输出pass和fail的地址
97      cout << "pass address = " << &pass << endl;
98      cout << "fail address = " << &fail << endl;
99  }
```

1. 定义了一个Student类，其中有静态数据`int count`、各种类型非静态数据成员（含字符指针）`char *name`、`int age`、`double score`、`bool & pass`，
2. 静态成员函数`static void printCount()`，用于输出`count`的值。
3. 成员函数`void print()`，用于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。
4. 构造函数`Student(char *n, int a, double s, bool &pass)`，析构函数`~Student()`。
5. 定义了全局对象`global_stu1`、`global_stu2`，main函数中局部对象`local_stu1`、`local_stu2`，另一个被main调用的外部函数`func`中定义局部对象`stu`。

# 实验结果

## 各个对象地址

```
1  # 全局对象
2  This is  stu1  address = 0x7ff76fa68060
3  This is  stu2  address = 0x7ff76fa68080
4  # 局部对象
5  This is  stu3  address = 0x10dd5ffbc0
6  This is  stu4  address = 0x10dd5ffba0
7  # 动态创建对象
8  This is  stu5  address = 0x1e16ce56190
9  This is  stu6  address = 0x1e16ce561e0
```

```
# func函数中的局部对象
This is  stu7  address = 0x10dd5ffb00
This is  stu8  address = 0x10dd5ffae0
```

1. 可以发现对于全局对象、局部对象，地址之间的间隔为 `0x20`，即 32 个字节，与 `sizeof(Student) = 32` 相符。
2. 全局对象存储在静态存储区域中，所以地址较高，后分配的对象地址递增
3. 局部对象存储在栈中，且栈是向下生长的，所以后分配的对象地址递减，stu3的地址比stu4的地址高。
4. 动态创建对象存储在堆中，地址递增。
5. func函数中的局部对象同样存储在栈中
6. 每次运行中，对于全局对象，地址是固定的，是因为全局对象存储在静态存储区域中，在链接时就已经确定了
7. 对于局部对象，每次运行中，地址是不固定的，是因为局部对象存储在栈中，每次运行时栈的基地址不同，这是为了防止栈溢出被恶意利用，对于堆同理

## 各个对象成员的值、地址、对象的存储空间大小

```
This is  stu1  address = 0x7ff682c78060
age =      20  address = 0x7ff682c78068
score =    90  address = 0x7ff682c78070
pass =      1  address = 0x7ff682c74000
name =   stu1  address = 0x7ff682c78060
count =     6  address = 0x7ff682c78040
size = 32
----------------
This is  stu2  address = 0x7ff682c78080
age =      21  address = 0x7ff682c78088
score =    80  address = 0x7ff682c78090
pass =      0  address = 0x7ff682c78044
name =   stu2  address = 0x7ff682c78080
count =     6  address = 0x7ff682c78040
size = 32
----------------
This is  stu3  address = 0x7492dff690
age =      22  address = 0x7492dff698
score =    70  address = 0x7492dff6a0
pass =      1  address = 0x7ff682c74000
name =   stu3  address = 0x7492dff690
count =     6  address = 0x7ff682c78040
size = 32
----------------
This is  stu4  address = 0x7492dff670
age =      23  address = 0x7492dff678
score =    60  address = 0x7492dff680
pass =      0  address = 0x7ff682c78044
name =   stu4  address = 0x7492dff670
count =     6  address = 0x7ff682c78040
size = 32
----------------
This is  stu5  address = 0x26baca96190
age =      24  address = 0x26baca96198
score =    50  address = 0x26baca961a0
pass =      1  address = 0x7ff682c74000
```

```
name =   stu5  address = 0x26baca96190
count =     6  address = 0x7ff682c78040
size = 32
----------------
This is  stu6  address = 0x26baca961e0
age =      25  address = 0x26baca961e8
score =    40  address = 0x26baca961f0
pass =      0  address = 0x7ff682c78044
name =   stu6  address = 0x26baca961e0
count =     6  address = 0x7ff682c78040
size = 32
----------------
This is  stu7  address = 0x7492dff5d0
age =      26  address = 0x7492dff5d8
score =    30  address = 0x7492dff5e0
pass =      0  address = 0x7ff682c78044
name =   stu7  address = 0x7492dff5d0
count =     7  address = 0x7ff682c78040
size = 32
----------------
This is  stu8  address = 0x7492dff5b0
age =      26  address = 0x7492dff5b8
score =    30  address = 0x7492dff5c0
pass =      0  address = 0x7ff682c78044
name =   stu8  address = 0x7492dff5b0
count =     8  address = 0x7ff682c78040
size = 32
```

```
pass address = 0x7ff682c74000
fail address = 0x7ff682c78044
```

1. 对于所有的对象，`name`、`age`、`score`、`pass`大小都是已知的

| 成员 | 大小 |
| --- | --- |
| `char *name` | 8 |
| `int age` | 4 |
| `double score` | 8 |
| `bool & pass` | 1 |

   在内存中，`name`、`age`、`score`、`pass`是按照声明的顺序存储的，所以`name`的地址是最低的，`pass`的地址是最高的。

2. 内存对齐：
   1. `char *name`：8字节对齐
   2. `int age`：4字节对齐
   3. `double score`：8字节对齐
   4. `bool & pass`：引用类型在编译的时候通常会被转换为指针，所以`pass`的大小是8字节
      所以`sizeof(Student) = 32`
3. 值
   - this指针：指向当前对象的指针，值为对象的起始地址

- name：字符数组指针，指向字符串的首地址
- age：int类型，值为年龄
- score：double类型，值为分数
- pass：bool类型，值为是否通过
- count：静态数据成员，值为对象的个数

4. 注意到所有的对象的`count`的地址都是一样的，说明静态数据成员是本类对象共享一份拷贝
5. 对于`pass`和`fail`，是全局变量，存储在静态存储区域中，对象中`bool & pass`是引用，所以值和引用的地址是一样的
6. 所以对象的本质是：对象是一块连续的内存空间，存储了对象的成员变量，成员函数是共享的，静态数据成员是本类对象共享一份拷贝

## 静态非静态成员函数地址，以及main、func的地址

```
stu1
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu2
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu3
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu4
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu5
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu6
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
main address = 0x7ff76fa61522
func address = 0x7ff76fa61450
```

1. c的虚拟存储空间的组成包含(由低到高)：
    1. 代码段：存放程序的代码
    2. 数据段：存放全局变量、静态变量、常量
    3. 堆：动态分配的内存
    4. 栈：局部变量、函数参数、返回地址
    5. 代码段和数据段是只读的，堆和栈是可读写的
2. 对于所有的对象，静态成员函数的地址是一样的，是因为静态成员函数是不属于任何对象的，是属于类的，所以地址是唯一的

3. 对于非静态成员函数，默认情况下会有一个隐含的指向调用对象的指针，即`this`指针，通过`obj→func()`调用，`this`指针指向`obj`，以达成使用同一个非静态成员函数的目的来操作不同的对象
4. 对于`main`和`func`函数，这俩个都是全局函数，获取他们的地址的结果是一个代码段的地址。存在一个隐式的函数指针，指向函数的入口地址。

## 实验总结

1. 对于对象的地址，全局对象存储在静态存储区域中，局部对象存储在栈中，动态创建对象存储在堆中
2. 对于对象的成员的值、地址、对象的存储空间大小，对象是一块连续的内存空间，存储了对象的成员变量，成员函数是共享的，静态数据成员是本类对象共享一份拷贝

## 完整输出结果

```
E:\Users\lenovo\CLionProjects\homework\cmake-build-debug\3-2.exe
This is  stu1  address = 0x7ff76fa68060 size: 8
name =   stu1  address = 0x7ff76fa68060 size: 8 value = 0x1e16ce56110
age =      20  address = 0x7ff76fa68068 size: 4 value = 20
score =    90  address = 0x7ff76fa68070 size: 8 value = 90
pass =      1  address = 0x7ff76fa64000 size: 1 value = 1
count =     6  address = 0x7ff76fa68040 size: 4 value = 6
size = 32
*name = 0x1e16ce56110
----------------
This is  stu2  address = 0x7ff76fa68080 size: 8
name =   stu2  address = 0x7ff76fa68080 size: 8 value = 0x1e16ce56130
age =      21  address = 0x7ff76fa68088 size: 4 value = 21
score =    80  address = 0x7ff76fa68090 size: 8 value = 80
pass =      0  address = 0x7ff76fa68044 size: 1 value = 0
count =     6  address = 0x7ff76fa68040 size: 4 value = 6
size = 32
*name = 0x1e16ce56130
----------------
This is  stu3  address = 0x10dd5ffbc0 size: 8
name =   stu3  address = 0x10dd5ffbc0 size: 8 value = 0x1e16ce56150
age =      22  address = 0x10dd5ffbc8 size: 4 value = 22
score =    70  address = 0x10dd5ffbd0 size: 8 value = 70
pass =      1  address = 0x7ff76fa64000 size: 1 value = 1
count =     6  address = 0x7ff76fa68040 size: 4 value = 6
size = 32
*name = 0x1e16ce56150
----------------
This is  stu4  address = 0x10dd5ffba0 size: 8
name =   stu4  address = 0x10dd5ffba0 size: 8 value = 0x1e16ce56170
age =      23  address = 0x10dd5ffba8 size: 4 value = 23
score =    60  address = 0x10dd5ffbb0 size: 8 value = 60
pass =      0  address = 0x7ff76fa68044 size: 1 value = 0
count =     6  address = 0x7ff76fa68040 size: 4 value = 6
size = 32
*name = 0x1e16ce56170
----------------
This is  stu5  address = 0x1e16ce56190 size: 8
name =   stu5  address = 0x1e16ce56190 size: 8 value = 0x1e16ce561c0
```

```
age =      24  address = 0x1e16ce56198 size: 4 value = 24
score =    50  address = 0x1e16ce561a0 size: 8 value = 50
pass =      1  address = 0x7ff76fa64000 size: 1 value = 1
count =     6  address = 0x7ff76fa68040 size: 4 value = 6
size = 32
*name = 0x1e16ce561c0
----------------
This is  stu6  address = 0x1e16ce561e0 size: 8
name =   stu6  address = 0x1e16ce561e0 size: 8 value = 0x1e16ce56210
age =      25  address = 0x1e16ce561e8 size: 4 value = 25
score =    40  address = 0x1e16ce561f0 size: 8 value = 40
pass =      0  address = 0x7ff76fa68044 size: 1 value = 0
count =     6  address = 0x7ff76fa68040 size: 4 value = 6
size = 32
*name = 0x1e16ce56210
----------------
This is  stu7  address = 0x10dd5ffb00 size: 8
name =   stu7  address = 0x10dd5ffb00 size: 8 value = 0x1e16ce56230
age =      26  address = 0x10dd5ffb08 size: 4 value = 26
score =    30  address = 0x10dd5ffb10 size: 8 value = 30
pass =      0  address = 0x7ff76fa68044 size: 1 value = 0
count =     7  address = 0x7ff76fa68040 size: 4 value = 7
size = 32
*name = 0x1e16ce56230
----------------
This is  stu8  address = 0x10dd5ffae0 size: 8
name =   stu8  address = 0x10dd5ffae0 size: 8 value = 0x1e16ce563c0
age =      26  address = 0x10dd5ffae8 size: 4 value = 26
score =    30  address = 0x10dd5ffaf0 size: 8 value = 30
pass =      0  address = 0x7ff76fa68044 size: 1 value = 0
count =     8  address = 0x7ff76fa68040 size: 4 value = 8
size = 32
*name = 0x1e16ce563c0
----------------
----------------
stu1
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu2
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu3
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu4
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
stu5
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
```

```
----------------
stu6
printCount address = 0x7ff76fa62d10
print address = 0x7ff76fa62d60
----------------
main address = 0x7ff76fa61522
func address = 0x7ff76fa61450
----------------
pass address = 0x7ff76fa64000
fail address = 0x7ff76fa68044

进程已结束，退出代码为 0
```