类和对象复习(上)

注：这些都是很基础的知识，我也掌握的还不错，所以会比较简略，提及的内容大多是想保证章节完整性的.

1. 类的定义

1.1 类定义格式

a. 类中成员变量在名称前或后加_ ,或者是m开头加以区分.

b. C中的struct在C++中升级成为了类

c. 定义在类中的成员函数默认为inline

1.2 访问限定符

a. public修饰的类成员可以直接被外部访问,而protected和private修饰的成员不能被外部直接访问.

b. 限定符的作用域到下一个限定符为止,或是到类的作用域 " } "结束.

c. class类内成员默认被private修饰,struct类内成员默认被public修饰.

1.3 类域

a. 类定义了一个新的作用域,当外部不对该类实例化时,可以通过类名加::访问类内公有成员.

b. 类域影响的是编译的查找原则,如果一个函数不指定为某类域,那么编译器就把它当做全局函数,在编译时,找不到函数内所用的成员函数就会报错.

2. 实例化

2.1 实例化的概念

a. 用类类型在物理内存中创建对象的过程，称为类实例化出对象。

b. 成员变量在类内只是是声明，只有类实例化出对象，才会分配空间。

c. 一个类可以实例化出多个对象，实例化出的对象 占用实际的物理空间，存储类成员变量。

2.2 对象大小

a. 类实例化出的每个对象，都有独立的数据空间，对象中肯定包含成员变量。但是成员函数是如何包含在其中的呢？对象是无法储存函数编译后形成的汇编指令的，这些指令被存放在代码段中，所以说如果对象相对成员函数进行存储的话，只能存储函数指针。但是如果实例化100个对象，同时也就意味着同样的函数指针我们要存储100个，这简直就是对内存的暴殄天物。所以，对于C++这样一款追求极致效率的语言来说，这是不能容忍的，也就是说函数指针是不需要存储的，函数指针是一个地址，代用函数被编译成调用指令[call 地址]，其实在编译器编译链接时，就要找到函数的地址，并非是在运行时，只有动态多态是在运行时查找地址，就需要存储函数地址。

    综上所述，类内成员函数的存储方式其实和类的this指针有关，函数编译后的指令只会在代码段中存储一份，在调用函数时，编译器会自动将对象的地址（`this` 指针）作为隐含参数传递给函数。

    所有对象共享同一份函数代码，但通过不同的 `this` 指针访问各自的成员变量。

    静态成员函数：直接通过类名调用，无需 `this` 指针，与全局函数类似。

b. **内存对齐规则**

   i. 第一格成员在与结构体偏移量为0的地址处

   ii. 其他成员变量要对齐到某个数字（对齐数）的整数倍的地址处。

   iii. VS中默认的对齐数为8

   iv. 如果嵌套了结构体的情况，嵌套的结构体对齐到自己的最大对齐数的整数倍处，结构体的整体大小就是所有最大对齐数（含嵌套结构体的对齐数）的整数倍。

代码块

```
#include<iostream>
using namespace std;
// 计算一下A/B/C实例化的对象是多大?
class A
{
public:
    void Print()
    {
        cout << _ch << endl;
    }
private:
    char _ch;
    int _i;
};
class B
{
public:
    void Print()
    {
```

```
20          //...
21      }
22  };
23
24  class C
25  {
26  };
27
28  int main()
29  {
30      A a;
31      B b;
32      C c;
33      cout << sizeof(a) << endl;
34      cout << sizeof(b) << endl;
35      cout << sizeof(c) << endl;
36      return 0;
37  }
```

上面的程序运行后，我们看到没有成员变量的B和C类对象的大小是1，为什么没有成员变量还要给1个字节呢？因为如果一个字节都不给，怎么表示对象存在过呢！所以这里给1字节，纯粹是为了占位标识对象存在。

3. this指针

· Date类中有 Init 与 Print 两个成员函数，函数体中没有关于不同对象的分，那当d1调用Init和Print函数时，该函数是如何知道应该访问的是d1对象还是d2对象呢？那么这里就要看到C++给了一个隐含的this指针解决这里的问题

· 编译器编译后，类的成员函数默认都会在形参第一个位置，增加一个当前类类型的指针，叫做this 指针。比如Date类的Init的真实原型为，

void Init(Date* const this, int year, int month, int day)

· 类的成员函数中访问成员变量，本质都是通过this指针访问的，如Init函数中给_year赋值，this->_year = year;

· C++规定不能在实参和形参的位置显示的写this指针(编译时编译器会处理)，但是可以在函数体内显 示使用this指针。

· **this指针存放在栈中**

代码块

```
1  #include<iostream>
2  using namespace std;
3  class Date
```

```cpp
{
public:
    // void Init(Date* const this, int year, int month, int day)
    void Init(int year, int month, int day)
    {
            // 编译报错：error C2106: “=”: 左操作数必须为左值
            // this = nullptr;
            // this->_year = year;
            _year = year;
            this->_month = month;
            this->_day = day;
    }

    void Print()
    {
            cout << _year << "/" << _month << "/" << _day << endl;
    }

private:
    // 这里只是声明，没有开空间
    int _year;
    int _month;
    int _day;
};

int main()
{
    // Date类实例化出对象d1和d2
    Date d1;
    Date d2;
    // d1.Init(&d1, 2024, 3, 31);
    d1.Init(2024, 3, 31);
    d1.Print();
    d2.Init(2024, 7, 5);
    d2.Print();
    return 0;
}

```