CPP从入门到入土之类和对象 I

Vect. 已于 2025-03-19 10:23:19 修改

目录

一、 类的定义

• 格式

class是CPP类的**关键字**

```
class{
// 成员函数
// 成员变量 为了区分成员变量的名字 我们一般会在名字前或后加_ 又或者m开头
// eg: int _a; float mval;
};
```

和结构体struct的定义方式相同

注意：CPP中struct也可以定义类，CPP兼容C中struct的用法，同时struct升级成了类，明显的变化是 struct中可以定义函数，一般情况下我们还是推荐用class定义类

看段代码：


```
#include "Show.h"

// class为类 是CPP的自定义类型
```

```cpp
// 定义一个栈
// 函数定义和申明在同一个类域中
class Stack {
// 接口函数一般公开使用
public:
    // 缺省参数，如果不传参，默认_capacity为4
    void Init(int n = 4) {
        arr = (int*)malloc(sizeof(int) * n);
        if (arr == nullptr)
        {
            perror("申请空间失败");
            return;
        }
        _capacity = n;
        _size = 0;
    }

    void Push(int input = 1) {
        if (_size == _capacity) {
            size_t _newcapacity = 2 * sizeof(int);
            int* tmp = (int*)realloc(arr, _newcapacity * sizeof(int));
            if (tmp == nullptr) {
                perror("realloc() err!");
                return;
            }
            arr = tmp;
            _capacity = _newcapacity;
        }
        arr[_size++] = input;
    }

    int Top() {
        assert(_size > 0);
        return arr[_size - 1];
    }

    void Destroy() {
        free(arr);
        arr = nullptr;
```

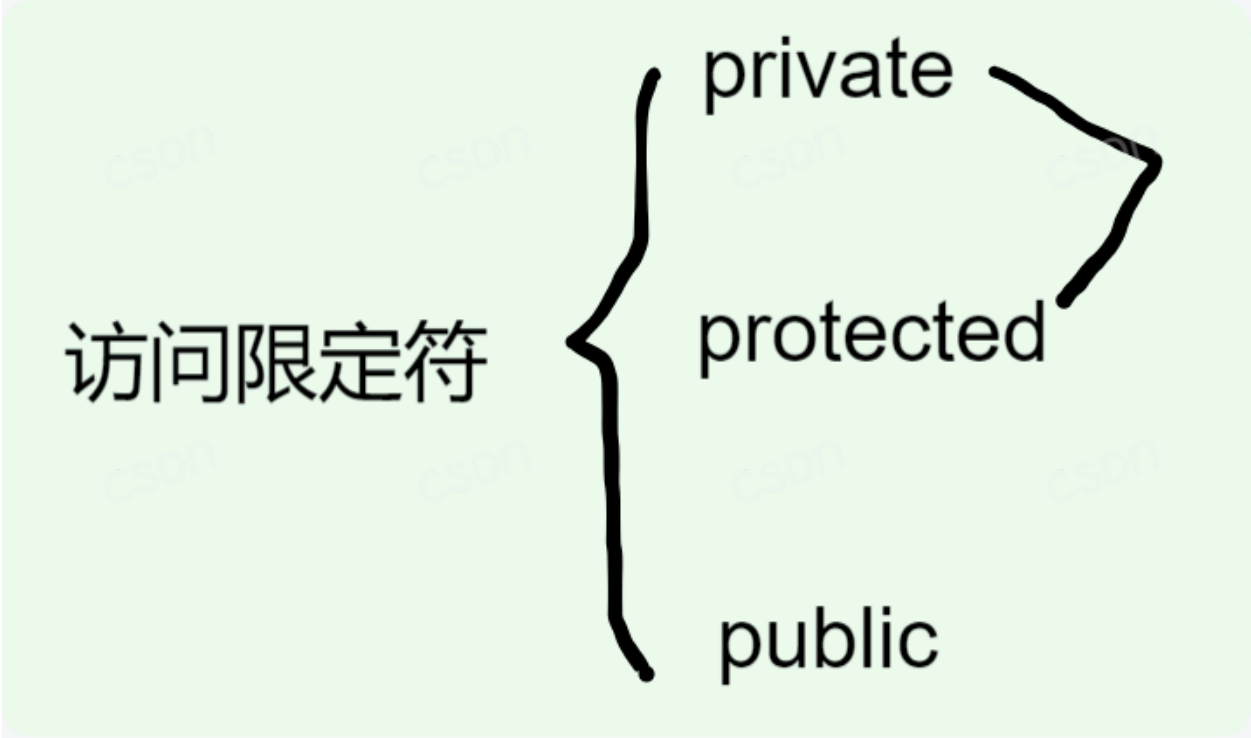
```
        _size = _capacity = 0;
    }

// 数据和实现方法一般被保护起来
private:

    // 为了区分成员变量，一般习惯上成员变量
    // 会加一个特殊标识，如_或者m开头
    size_t _size;
    size_t _capacity;
    int* arr;
};


int main() {
     //不用typedef Stack就是一个类 而st是Stack这个类的实例化对象
    Stack st;
    st.Stack::Init(12);
    st.Stack::Push(1);
    st.Stack::Push(2);

    cout << st.Stack::Top();

    st.Stack::Destroy();

    return 0;

}
```

AI写代码

- **访问限定符和类域**

观察上方代码，我们发现有个**public和private，**他们的作用是什么?

访问限定符有**public、private、protected**三个



- CPP有**封装**的特性，用**类**将**对象**的**特性**和**实现方法**有机结合起来，而利用访问限定符**设置访问权限**将**接口**提供给外部用户使用
- **private**和**protected（现阶段并未使用）**是将对象藏起来，在类外部不能直接访问，而**public**则是将对象公开，在类外可直接访问
- 访问权限作用域从该访问限定符出现的位置开始直到下一个访问限定符出现时为止，如果后面没有访问限定符，作用域就到类结束
- class定义成员没有被访问限定符修饰时默认为**private**，struct默认为**public**，所以建议用class定义类
- 有什么意义？细想一下，如果没有权限约束，类中的成员可以随意被外界修改，那么遇到恶意修改，程序分分钟就崩溃了

所以，一般来说，成员函数公开，成员变量隐私

所以类就有自己的作用域了，称为**类域**

在类体外定义成员时，需要用**域操作符::**指明成员属于哪个类域，防止冲突

类域影响的是编译的查找规则，相当于将自己的成员包起来不让编译器容易找到

代码示例：

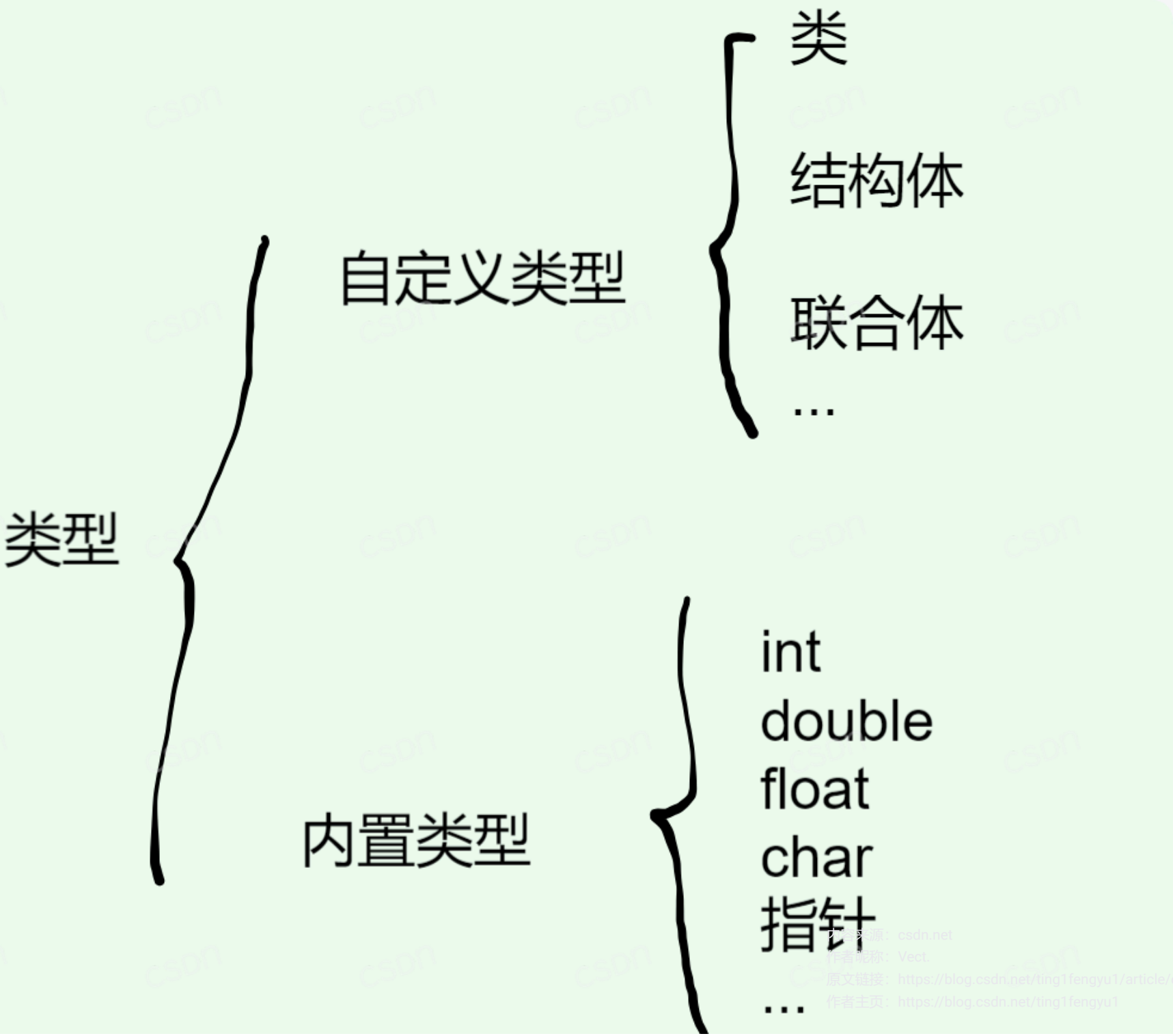
```cpp
// show.h
#include <iostream>
#include <assert.h>
using namespace std;

// 函数声明
class Stack {
public:
    // 成员函数
    void Init(int n = 4);
    void Push(int input);
    int Top();
    void Destroy();
private:
    // 成员变量
    int* _arr;
    size_t _top;
    size_t _capacity;
};

// show.cpp
#include "Show.h"
// 函数定义
void Stack::Init(int n = 4) {
    _arr = (int*)malloc(sizeof(int) * n);
    if (_arr == nullptr){
        perror("申请空间失败");
        return;
    }
    _capacity = n;
    _top = 0;
}

void Stack::Push(int input) {
    if (_top == _capacity) {
        size_t _newcapacity = 2 * sizeof(int);
        int* tmp = (int*)realloc(_arr, _newcapacity * sizeof(int));
        if (tmp == nullptr) {
```

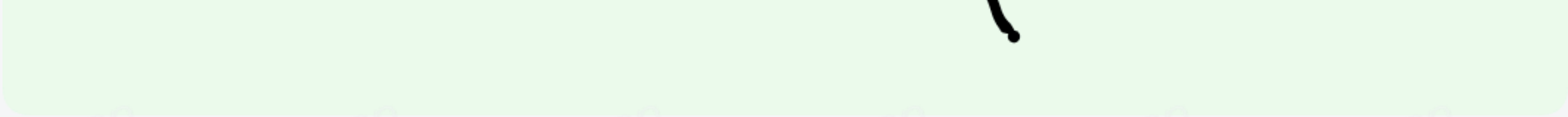
```
            perror("realloc() err!");
        }
        _arr = tmp;
        _capacity = _newcapacity;
    }
    _arr[_top++] = input;
}

int Stack::Top() {
    assert(_top > 0);
    return _arr[_top - 1];}

void Stack::Destroy() {
    free(_arr);
    _arr = nullptr;
    _top = _capacity = 0;
}

// test.cpp
int main() {
    //不用typedef Stack就是一个类 而st是Stack这个类的实例化对象
    Stack st;
    st.Stack::Init(12);
    st.Stack::Push(1);
    st.Stack::Push(2);

    cout << st.Stack::Top();

    st.Stack::Destroy();

    return 0;

}
```

AI写代码

二、实例化

- **用类这个类型在内存中创建对象的过程，称为类实例化出对象**





- 类是对象进行的一种抽象描述，是模型，限定了类有哪些成员变量，这些成员变量只是声明，没有分配空间，用类实例化出对象时，才会分配空间
- 一个类可以实例化出多个对象，实例化出的对象才占用实际的物理空间，存储类成员变量，就像我上面说的类是一个模型，一个大纲，指导具体的怎么做

代码示例：

```
class Date {
public:
    void Today(int year, int month, int day) {
        int _year = year;
```

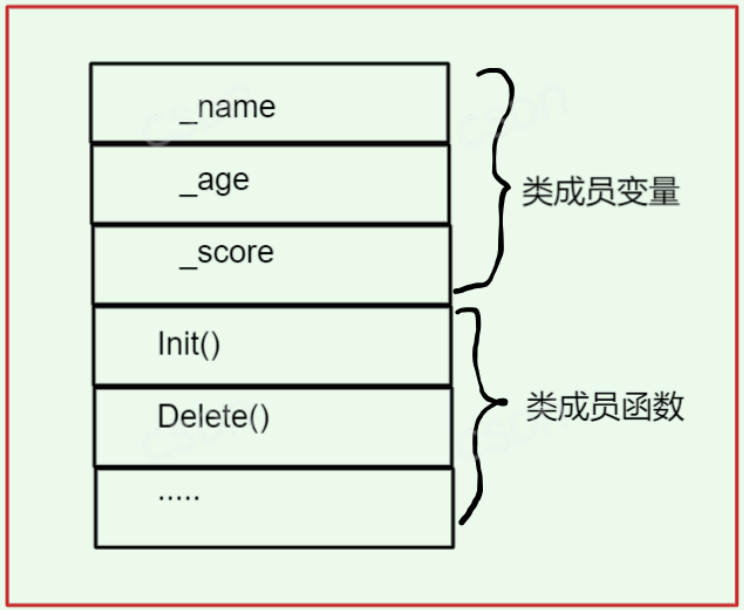
```
        int _month = month;
        int _day = day;
    }
    void Print() {
        cout << _year << "-" << _month << "-" << day;
    }
 //仅声明，并未开辟空间，实例化对象时才开辟空间
private:
    int _year;
    int _month;
    int _day;
};

int main() {

    Date d;
    d.Today(2024, 3, 17);

    return 0;

}
```

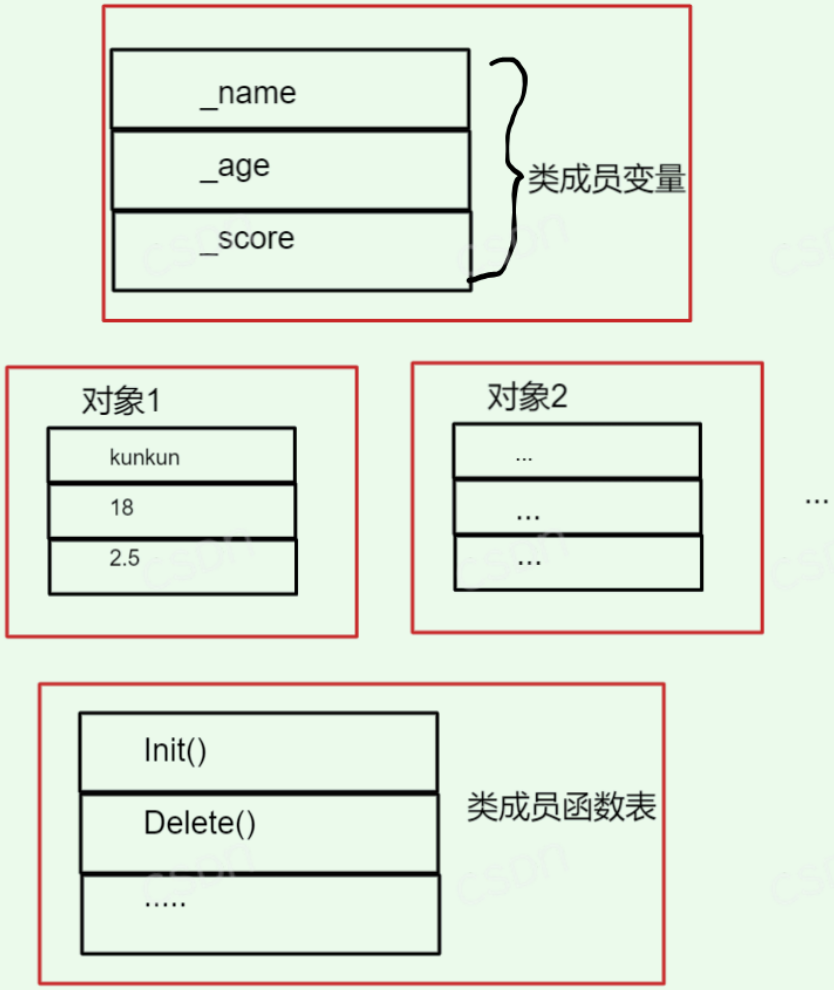
AI写代码

- **对象大小**

分析一下：**对象会实例化存储在内存中，每个实例化对象都有自己独立的空间，所以成员变量一定有自己的空间**，而成员函数呢？
函数被编译后是一段指令，对象中没办法存储，这些指令存储在一个单独的区域(代码段)，如果非要存储，则是存储函数指针，那有必要存储函数指针吗?
Date实例化d1和d2两个对象，d1和d2都有各自独立的成员变量 _year/_month/_day存储各自的数据，但是d1和d2的成员函数Init/Print指针却是一样的，存储在对象中就浪费了，一两个对象还好，如果实例化100个、1000个对象呢，每次存储相同的地址实在是太浪费了
实函数指针不需要存储，函数指针是一个地址，调用函数被编译成汇编指令[call 地址]，编译器在编译链接时，就找到了函数的地址，不是在运行时找

类对象存储设计方案：





CPP规定类实例化的对象也要符合内存对齐的规则，因为类就是特殊的结构体

复习一下内存对其原则：

> **1.结构体的第一个成员对齐到和结构体变量起始位置偏移量为0的地址处**

2.其他成员变量要对齐到对齐数的整数倍的地址处

对齐数=编译器默认的一个对齐数与该成员变量大小的较小值

VS默认为8

Linux中没有默认对齐数 gcc默认对齐数就是成员自身大小

3. 结构体总大小为最大对齐数（结构体中每个成员变量都有一个对齐数，所有对齐数中最大的）的整数倍

4. 如果嵌套了结构体的情况，嵌套的结构体成员对齐到自己的成员中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（含嵌套结构体中成员的对齐数）的整数倍。

详细看我这篇文章：拒绝废话，四万字帮你快速打通C语言的任督二脉（已完结）_如何快速看懂c语言代码-CSDN博客

```cpp
// 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()
    {
        //...
    }
};

class C
{
    //...
};
```

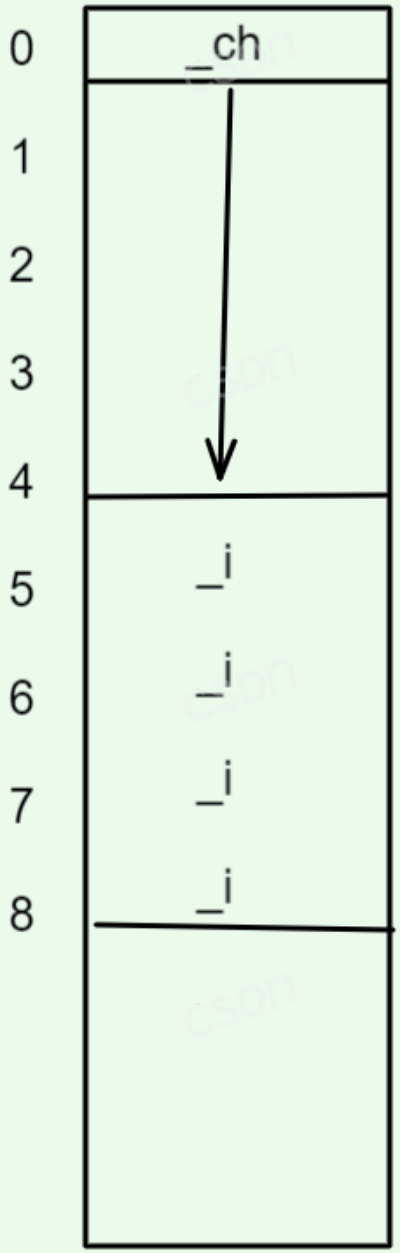
```
28
29 int main() {
30
31     cout << sizeof(A) << "\n" << sizeof(B) << "\n" << sizeof(C);
32     return 0;
33 }
```

AI写代码

分析：

A：8 成员函数不占内存

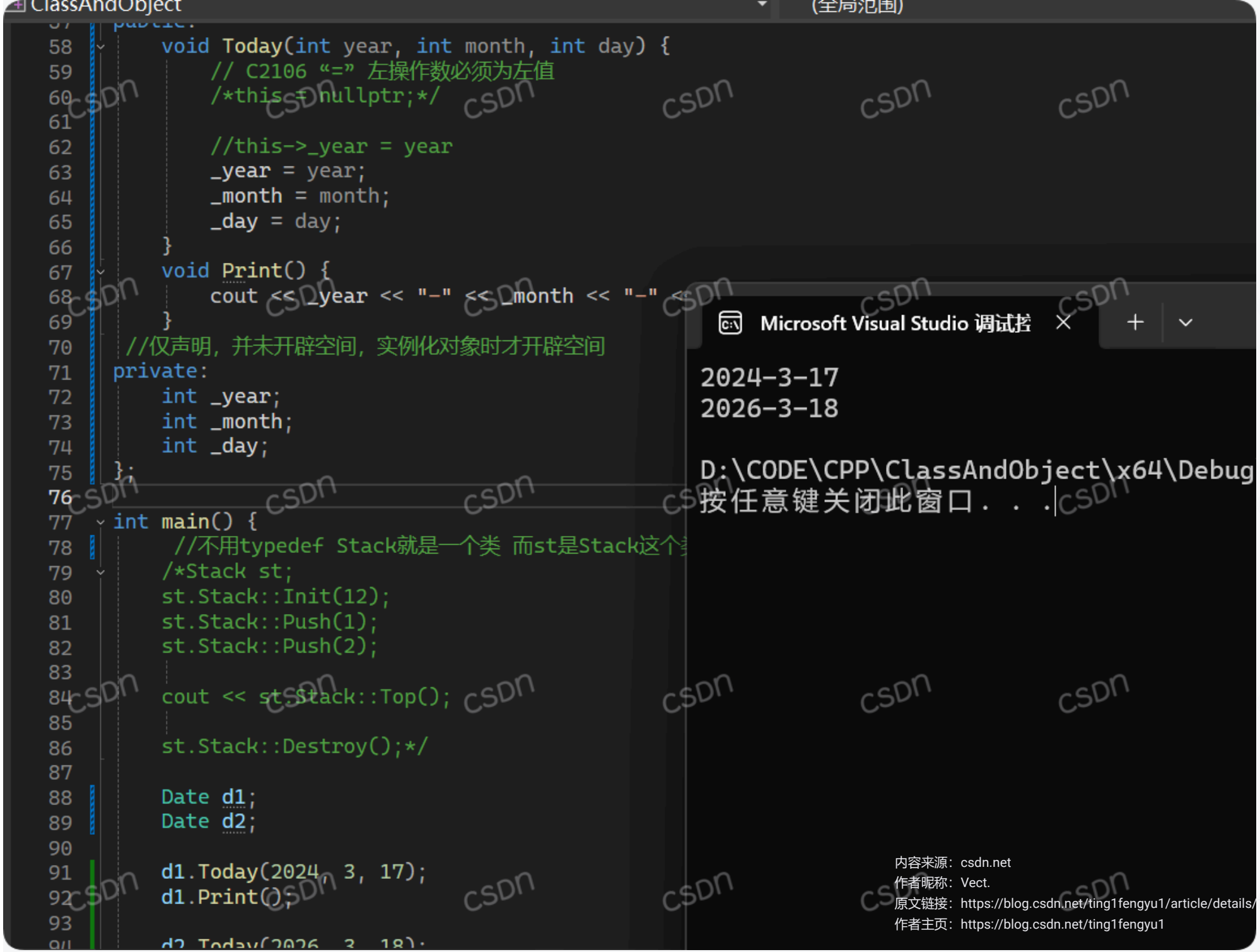
```
char _ch;
int _i;
```



B、C：1

空类大小默认为1，这个1代表着占位标识对象存在

三、 this指针



有个问题，我们上面**Date**类中有**Today()**和**Print()**两个成员函数，函数体中没有关于不同对象的区分，那当d1调**Today()**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?

CPP提出隐含的**this指针**来解决这个问题

- 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**。如**Date类**的**Init()**的真实原型为**Today(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**Today()**函数中给**_year**赋值: **this->_year = year;**
- CPP规定不能在实参和形参的位置显式地写**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指针是存在栈区的，VS会存到**寄存器ecx**中

```
        //this->_year = year
        _year = year;
00007FF7280F25FE  mov         rax,qword ptr [this]
00007FF7280F2605  mov         ecx,dword ptr [year]
00007FF7280F260B  mov         dword ptr [rax],ecx
        _month = month;
00007FF7280F260D  mov         rax,qword ptr [this]
00007FF7280F2614  mov         ecx,dword ptr [month]
00007FF7280F261A  mov         dword ptr [rax+4],ecx
        _day = day;
00007FF7280F261D  mov         rax,qword ptr [this]
00007FF7280F2624  mov         ecx,dword ptr [day]
00007FF7280F262A  mov         dword ptr [rax+8],ecx
    }
```

看到这里了，麻烦点个关注和赞再走呗~