一、类和对象的初步认识

$ 什么是类？通过一个例子来说

```
struct student
{
 int number;
 char sex;
 int age;
 char grender;
};
```

$ 首先 C 语言与 C++ 当中的类到底有什么区别：通过一个例子来说：

```
struct Student
{
 void SetStudentInfo(const char* name, const char* gender, int age)
 {
 strcpy(_name, name);
 strcpy(_gender, gender);
 _age = age;
 }
 void PrintStudentInfo()
  {
  cout<<_name<<" "<<_gender<<" "<<_age<<endl;
  }
  char _name[20];
  char _gender[3];
  int _age;
};
```

即 C++ 中的结构体当中不仅可以定义变量还可以定义函数

二、类的定义及两种声明方式：

```
class student{
 //即 class 或者 struct 加类名
 //struct 默认成员是公有的
};
```

$ 下面我们写一个加法程序分别在类内定义和类外定义：

```
class Student{
 //即 class 或者 struct 加类名
 //struct 默认成员是公有的
 int ADD(int left, int right) //类内定义函数
 {
```

```
 return left + right;
 }
 int Sub(int left, int right); //类内声明减法程序
};
```

$ 类外定义减法程序 Sub

```
//其中 "::" 是作用域限定符说明减法程序隶属于 Struct 类
int Student::Sub(int left, int right)
{
 return left + right;
}
```

三、类的访问限定符及封装

注意：类的访问限定符只在编译时有用，当数据映射到内存后，没有任何限定符上的区别；

$ C++实现封装的方式：用类将对象的属性与方法结合在一块，让对象更加完善，通过访问权限选择性的将其 接口提供给外部的用户使用

```
class Student
{
public:
 //方法
 Student() //构造函数
 {}
 ~Student() //析构函数
 {}
 Get_val()
  {
  return _val;
  }
private: //public 保护的成员可以在类外直接访问
 //private 保护的成员受限制只能在类内访问
 //或者对外提供一个方法。比如对外获取 _val;
 //成员
 int _val;
};
```

$ C++ 中 class 和 struct 的区别?

C++需要兼容C语言，所以C++中struct可以当成结构体去使用。另外C++中struct还可以用来定义类。 和class是定义类是一样的，区别是struct的成员默认访问方式是public，class是struct的成员默认访问方式 是private

四、类的作用域?

类定义了一个新的作用域，类的所有成员都在类的作用域中。在类体外定义成员，需要使用 :: 作用域解析符 指明成员属于哪个类域。

五、类的实例化即用一个抽象的类构造出一个或者多个具体的对象出来

六、结构体大小的计算

编译器给定空类的大小为 1 字节

$ 结构体内存对齐规则：

a、第一个成员在与结构体偏移量为0的地址处。

b、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。 注意：对齐数 = 编译器默认的一个对齐数 与 该成员大小的较小值。 VS中默认的对齐数为8，gcc中的对齐数为4

c、结构体总大小为：最大对齐数（所有变量类型最大者与默认对齐参数取最小）的整数倍。

d、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。

e、Linux 中如何设置默认对齐数：

```
#pragma pack()
```

七、this 指针

```
class Date
{
 Date()
 {}
 ~Date()
 {}
 void Display()
 {
 cout << _year << _month << _day;
  }
  void SetDate(int year, int month, int day)
  {
  //this 指针指向当前对象
  this->_year = year;
  this->_month = month;
  this->_day = day;
  }
private:
  int _year;
  int _month;
  int _day;
};
```

$ this 指针的特性：

a、this 指针的类型：类类型*const

b、只能用于成员函数内部

c、this 指针本质上是一个成员函数的形参

d、this指针是成员函数第一个隐含的指针形参，一般情况由编译器通过ecx寄存器自动传递，不需要用户 传递

八、类的 6 个默认成员函数

```
class Date
{
public:
 Date()//无参构造函数
 {}
 Date(int year, int month, int day)//带参构造函数
 {
 _year = year;
 _month = month;
  _day = day;
  }
  Date(int year = 1990, int month = 1, int day = 1)//带默认参数的构造函数
  {
  _year = year;
  _month = month;
  _day = day;
  }
 private:
  int _year;
  int _month;
  int _day;
 };
 void TestDate()
 {
  Date date1;//调用无参默认构造函数
  Date date2(2015, 1, 1);//调用带参的构造函数
 }
```

1、构造函数：成员变量私有，对其进行初始化

$ 构造函数是一种随着对象创建自动被调用的公有成员函数，有且仅在对象定义时自动执行一次，它的主要作用是对其进行初始化

$ 特征

a、函数名与类名相同

b、无返回值

c、对象实例化是系统自动调用构造函数对其进行初始化

d、构造函数可以重载

e、构造函数可以在类内定义也可以在类外定义

$ 对象被定义必定有构造函数被调用

$ 全缺省和无参的都被认为是缺省构造函数但是二者不可以共存

2 、析构函数

```
Date::~Date()//动态开辟的内存一般需要进行析构
{}
```

$ 当一个对象的生命周期结束时，C++ 编译系统会自动调用一个成员函数，即析构函数；

注意：析构函数不是删除对象，而是做一些清理工作

```
class MyVector
{
public:
 MyVector(int size)
 {
 _ptr = (int *)malloc(size * sizeof(int));
 }
 ~MyVector()
 {
  free(_ptr);//释放堆上的空间
  _ptr = 0;//指针置空
 }
}
```

$ 特征

a、析构函数实在名字前面加上~

b、无参数无返回值不可重载

c、一个类有且只有一个析构函数

d、对象生命周期结束时，C++编译系统自动调用析构函数进行清理工作

3、拷贝构造函数

```
Date::Date(const Date& d)
{
 _year = year;
 _month = month;
 _day = day;
}
```

```
void TestDate()
{
 Date date1;
 Date date2(date1);
 Date date3(date2);
}//拷贝构造函数、析构等函数若存在动态开辟均需要自己手写代码
```

$ 创建对象时，使用同类对象来进行初始化，即拷贝构造函数

$ 在类的成员函数中可以直接访问同类对象的私有成员、C++的访问限定符是以类为单位的也就是说在这个单位内的成员可以互相访问

$ 特征

a、拷贝构造函数时构造函数的一个重载

b、拷贝构造函数的参数只有一个必须使用引用传参

4、运算符重载

```
operator + 合法的运算符//重载运算符以后不能改变运算符的优先级，操作数个数
```

5、赋值操作符重载：时对一个已经存在的对象进行拷贝赋值

Date 类的实现

```
class Date
{
public:
 Date()
 {}
 ~Date()
 {}
 Date(const Date& d)
 {
 _year = year;
 _month = month;
 _day = day;
 }
 Date& operator = (const Date& d)//赋值运算符重载
 {
 if(this != &d)//不是自己给自己赋值
 {
 this->_year = year;
 this->_month = month;
 this->_day = day;
 }
```

```
 return *this;
 }
private:
 int _year;
 int _month;
 int _day;
};
```

6、const 成员函数

$ 将const修饰的类成员函数称之为const成员函数，const修饰类成员函数，实际修饰该成员函数隐含的this 指针，表明在该成员函数中不能对类的任何成员进行修改

```
const对象可以调用非const成员函数吗？
非const对象可以调用const成员函数吗？
const成员函数内可以调用其它的非const成员函数吗？
非const成员函数内可以调用其它的const成员函数吗？
```

7、类的取地址操作符重载及 const 修饰的取地址操作符重载

Date.h

```
#pragma once
#include<iostream>
using namespace std;

// 自定义对象尽量传引用

class Date
{
 /*
 友元函数：
 1、不是类的成员函数
 2、可以访问类的私有成员变量
 3、友元函数不能用 const 修饰
 4、破坏了类的封装性
 5、友元函数可以在类的任何地方声明，不受访问限定符限制
 6、友元函数没有隐藏的 this
 友元函数不需要通过对象来调用
 */
 friend ostream& operator<<(ostream& _cout, const Date& d); //友元函数
public:
 /*
 构造函数：是一种随着对象创建而自动被调用的公有成员函数，有且仅在定义对象时自动执行一次，对对象进行初始化
 特征：
```

```
1、函数名与类名相同
2、无返回值
3、对象构造（对象实例化）时系统自动调用对应的构造函数
4、构造函数可以重载
5、构造函数可在类内定义也可在类外定义类内声明
6、如果类定义没有给出构造函数，则 C++ 编译器自动产生一个默认的构造的函数，但只要我们定义了一个构造函数，系统就不会自动生成默认的构造函数
7、无参的构造函数和全默认值的构造函数都认为时默认构造函数，并且认为默认构造函数只能有一个
*/
Date(int year = 1900, int month = 1, int day = 1) //构造函数
: _year(year)
, _month(month)
, _day(day)
{}
void Display() //打印函数
{
cout << _year << "-" << _month << "-" << _day << endl;
}

/*
拷贝构造函数：创建对象时能使同类对象来进行初始化，即拷贝构造函数
特征：
1、其实就是构造函数的一个重载
2、拷贝构造函数参数只有一个并且只能使用引用传参，使用传值传参会引发无穷递归

无穷递归的产生：
1、在类的成员函数中可以直接访问同类对象的私有成员
2、C++ 的访问限定符，是以类为单位，也就是说在这个单位内的成员可以互相访问

*/
Date(const Date& d) //拷贝构造函数
{
_year = d._year;
_month = d._month;
_day = d._day;
}

/*
类的赋值操作符重载
1、对一个已经存在的对象进行拷贝赋值
```

```
*/
Date& operator=(const Date& d) //赋值函数（运算符重载）
{
if (this != &d)
{
this->_year = d._year;
this->_month = d._month;
this->_day = d._day;
}
return *this;
}
Date operator+(int days)
{
_day += days;
while (_day > 30)
{
_day -= 30;
_month += 1;
}
while (_month > 12)
{
_month -= 12;
_year += 1;
}
return *this;
}
Date operator+=(int days);
Date operator-(int days);
Date operator-=(int days);
int operator-(const Date& d);
Date& operator++() //年份自增函数
{
_day++;
while (_day > 30)
{
_day -= 30;
_month += 1;
 }
 while (_month > 12)
 {
```

```
_month -= 12;
_year += 1;
}
return *this;
}
Date operator++(int);
Date& operator--() //年份自减函数
{
_day --;
while (_day < 1)
{
_day = 30;
_month -= 1;
}
while (_month < 1)
{
_month = 12;
_year -= 1;
}
return *this;
}
Date operator--(int);
bool operator>(const Date& d) //2018-1-1 > 1990-1-1
{
return _year > d._year
|| (_year == d._year && _month > d._month)
|| (_year == d._year && _month == d._month && _day > d._day);
}
bool operator>=(const Date& d) //2018-10-10 >= 2018-9-10
{
if (_year >= d._year
|| (_year == d._year && _month >= d._month)
|| (_year == d._year && _month == d._month && _day >= d._day))
{
cout << "true" << endl; //条件成立打印 true
return true;
}
else
{
cout << "false" << endl;
```

```
143 return false;
144 }
145 }
146 bool operator<(const Date& d)
147 {
148 if (_year < d._year
149 || (_year == d._year && _month < d._month)
150 || (_year == d._year && _month == d._month && _day < d._day))
151 {
152 cout << "true" << endl;
153 return true;
154 }
155 else
156 {
157 cout << "false" << endl;
158 return false;
159 }
160 }
161 bool operator<=(const Date& d)
162 {
163 if (_year <= d._year
164 || (_year == d._year && _month <= d._month)
165 || (_year == d._year && _month == d._month && _day <= d._day))
166 {
167 cout << "true" << endl;
168 return true;
169 }
170 else
171 {
172 cout << "false" << endl;
173 return false;
174 }
175
176 }
177 bool operator==(const Date& d)
178 {
179 return _year == d._year
180 && _month == d._month
181 && _day == d._day;
182 }
```

```
 bool operator!=(const Date & d)
 {
 return _year != d._year
 || _month != d._month
 || _day != d._day;
 }
 /*
 析构函数：当一个对象的生命周期结束时，C++ 编译器会自动调用一个成员函数，即析构函数
 特征：
 1、析构函数构成 ~类名
 2、无参数无返回值
 3、一个类有且只有一个析构函数
 4、对象生命周期结束时，自动调用析构函数
 析构函数不是删除对象，而是对对象删除前的一些删除工作
 */
 ~Date()
 {}
private:
 int _year;
 int _month;
 int _day;
};

ostream& operator<<(ostream& _cout, const Date& d)
{
 _cout << d._year << "-" << d._month << "-" << d._day << endl;
}
void TestDate()
{
 Date date1;
 cout << date1 << endl;
 Date date2(2000, 1, 1);
 Date date3;
 Date date4 = date2;
 Date date5(1997, 1, 10);
 Date date6(1999, 1, 1);
 date4.Display();
 date5.Display();
 date5.operator++();
```

```
 date5.Display();
 date5.operator--();
 date5.Display();
 date5.operator<(date6);
 date2.operator>=(date1);
 date6.Display();
 date1.operator<(date2);
 date1.operator==(date2);
 date1.operator>=(date2);
 date5.operator<=(date6);
 date5.operator>=(date6);
 date1.Display();
 date2.Display();
 date2.operator+(80);
 date2.Display();
 date3.Display();
 date3.operator--();
 date3.Display();
 date3.operator<(date1);
 date5.operator<(date1);
 date4.Display();
 date4.operator++();
 date4.Display();
}

```

Date.cpp

```
#include "Date.h"

int main()
{
 TestDate();
 return 0;
}
```

九、面试题：

```
结构体为什么要进行内存对齐？怎么对齐？
如何让结构体按照制定的对齐参数进行结构体内存对齐？
如何直到结构体当中某个成员相对结构体起始地址的偏移量？
什么是大小端？如何测试一台机器是大端还是小端？有没有遇到要考虑大端小端的场景？
C++ 中 struct 和 class 的区别？
this 指针存在于哪里？this 指针可以为空吗？
```

如何在类外访问一个类中 `private` 的成员变量？

面向对象的三大特性：封装、继承、多态；什么是封装？

拷贝构造函数传值的无穷递归理解？

`C++` 中不能重载的运算符有哪些？