拷贝构造；拷贝赋值，析构

- 只要Class里面带了指针，就一定要自己写拷贝构造和拷贝赋值
- 拷贝赋值：“=”的操作符重载
- 析构函数
-
- 如果不写拷贝赋值或者拷贝构造函数，使用编译器默认的拷贝机制，就会造成浅拷贝

```
浅拷贝：
a b 都是一个指针；使用编译器默认的拷贝机制 a = b；就是将a的指针指向b的地址空间，然后a原来的地址空
深拷贝：
将 b指向的内存空间的值拷贝到 a所指向的内存空间
```

- 拷贝构造函数：

```
形如 String(const String &str);的函数就叫做拷贝构造
需要做的工作： 申请出足够的内存空间
```

拷贝赋值

- 先delete掉自己，然后创建一个跟另一块一样大小的地址空间，再进行赋值操作（将a的值赋给b）

拷贝构造；拷贝赋值，析构 Demo

```
#include <iostream>
#include <cstring>
#include <stdio.h>
class String
{
public:
    String(const char *str = 0);
    String(const String &str);//拷贝构造
    String & operator = (const String &str);
    char * real() const {return m_data;}
    ~String();
private:
    char *m_data;
};
inline String :: String(const char *str )
{

    m_data = new char [strlen(str) + 1];
```

```
    strcpy(m_data,str);

}
inline String& String :: operator =  (const String &str)
{
    if(this == &str) //这个必须有
            return *this;
    delete[] m_data;
    m_data = new char [strlen(str.m_data) + 1];
    strcpy(m_data,str.m_data);
}
inline String :: ~String()
{
delete[] m_data;
}
int main(int argc,char ** argv)
{
    String s1();
    String s2("hello");
    String s3("hellooo");
    s2 = s3;
    printf("s2 is %s s3 is %s",s2.real(),s3.real());
    return 0;

}
```

- 操作符<<的重载 我也不知道哪里错了

```
ostream& operator << (ostream &os ,const String & str)
{
        os << str.real();
        return os;
}

```