一

*通过string类模拟实现剖析构造函数*

*①string类模拟实现*

```
//string类模拟实现
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>
using namespace std;
class String
{
public:
	String()
	{
		//m_data = (char*)malloc(1);    //string类型变量的默认初始值是为null,所以不开辟空间也无可厚非
	}
	String(const char* str)
	{
		m_data = (char*)malloc(sizeof(char) * (strlen(str) + 1));
		strcpy(m_data, str);
	}

	String(const String& s)
	{
		m_data = (char*)malloc(sizeof(char) * (strlen(s.m_data) + 1));
		if (m_data != nullptr)
			strcpy(m_data, s.m_data);
	}
	String& operator=(const String& s)
	{
		if (this == &s)
			return *this;

		free(m_data);

		m_data = (char*)malloc(sizeof(char) * (strlen(s.m_data) + 1));
		strcpy(m_data, s.m_data);

		return *this;
	}
```

```
    ~String()
    {
        free(m_data);
        m_data = NULL;
    }
private:
    char* m_data;
};

int main()
{
    String s1;
    String s2("Hello");
    String s3;
    s3 = s2;
    return 0;
}
```

在C++中有string类（string类型变量的默认初始值是为null），对其模拟实现如上

```
String()
    {
    }
String(const char *str)
    {
                m_data = (char *)malloc(sizeof(char)*(strlen(str)+1));
                strcpy(m_data, str);
    }
```

## ②string类模拟实现--构造函数改进1

```
//string类模拟实现--构造函数改进1
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>
using namespace std;

class String
{
public:
	String(const char *str = nullptr)
	{
		if(str == nullptr)
		{
			m_data = (char*)malloc(sizeof(char));
			m_data[0] = '\0';
		}
		else
		{
			m_data = (char *)malloc(sizeof(char)*(strlen(str)+1));
			strcpy(m_data, str);
		}
	}

	String(const String &s)
	{
		m_data = (char*)malloc(sizeof(char) * (strlen(s.m_data)+1));
		if (m_data != nullptr)
			strcpy(m_data, s.m_data);
	}
	String& operator=(const String &s)
	{
		if(this == &s)
			return *this;

		free(m_data);

		m_data = (char*)malloc(sizeof(char)*(strlen(s.m_data)+1));
		strcpy(m_data, s.m_data);

		return *this;
	}
	~String()
	{
		free(m_data);
```

```
            m_data = NULL;
        }
private:
        char *m_data;
};

int main()
{
        String s1;
        String s2("Hello");
        String s3;
        s3 = s2;
        return 0;
}
```

亮点：将缺省构造函数和设默认值的构造函数结合在一块

String(const char *str = nullptr)这个构造函数是用指针接收字符串，并设置了默认值

当String s1;

实例化对象时构造函数会为对象的私有数据成员开辟一个字节的内存空间

String s2("Hello");

构造函数会为对象的私有数据成员开辟和"Hello"字符串字节数相同的内存空间

## ③*string类模拟实现--构造函数改进2*

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>
using namespace std;

class String
{
public:
        String(const char *str = "")    //设默认值
        {
                m_data = (char *)malloc(sizeof(char)*(strlen(str)+1));
                if (m_data != nullptr)
```

```
            strcpy(m_data, str);
    }
    String(const String &s)
    {
        m_data = (char*)malloc(sizeof(char) * (strlen(s.m_data)+1));
        if (m_data != nullptr)
            strcpy(m_data, s.m_data);
    }
    String& operator=(const String &s)
    {
        if(this != &s)
        {
            free(m_data);
            m_data = (char*)malloc(sizeof(char)*(strlen(s.m_data) + 1));
            if (m_data != nullptr)
                strcpy(m_data, s.m_data);
        }
        return *this;
    }
    ~String()
    {
        free(m_data);
        m_data = NULL;
    }
private:
    char *m_data;
};

int main()
{
    String s1;
    String s2("Hello");
    String s3;
    s3 = s2;
    return 0;
}
```

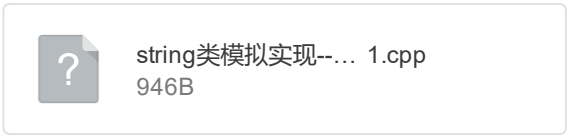
相比于上面1的改进，这个构造函数设默认值为空字符串（属于常量，空字符串中只有一个字符'/0'）

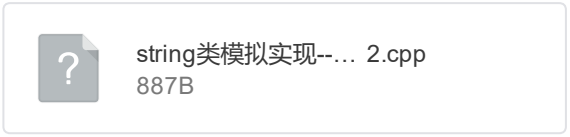
```
String(const char *str = "")    //设默认值
    {
        m_data = (char *)malloc(sizeof(char)*(strlen(str)+1));
        if (m_data != nullptr)
            strcpy(m_data, str);
```

```
}
```

实例化对象时要是缺省则构造函数会为对象的私有数据成员开辟一个字节的内存空间
如果是用字符串"Hello"初始化对象则构造函数会为对象的私有数据成员开辟和"Hello"字符串字节数相同的内存空间
和改造1取得的效果是一样的
但代码更简洁好看

以上源代码：


# 二

*动态内存开辟函数new*

在C语言中我们常用
malloc calloc realloc free这四中动态内存开辟函数，他们都是在堆上开辟内存空间的
关于这四个函数的用法请大家去看：

在C++中有动态内存开辟函数new----->在堆上开辟
下面做简单用法总结

**①开辟一个整形空间，以及释放**

```
int *pi1 = new int;      开辟
*pi1 = 1;                    赋值
delete pi1;      释放
```

②开辟数组，以及释放

```
int *pa1 = new int[10];  //开辟数组
for(int i=0; i<10; ++i)
{
        pa1[i] = i+1;           //赋值
}
for(int i=0; i<10; ++i)
{
        cout<<pa1[i]<<" ";
}
cout<<endl;
delete []pa1;           //释放
```

注意数组释放要在指向所开辟空间的指针前加　　[]

顺便提一下在堆上开辟的匿名数组即使被pa1指针接收到了但，pa1指针拿到的是首元素地址（将数组名给指针，指针永远拿到的都是首元素地址，只有在strlen中时拿到的才是整个数组的地址）

所以错误的用法如下：（auto关键字的用法介绍请看 ：　　C++知识体系的建立　）

```
int *pa1 = new int[10]; //数组
for (auto& e : (int*)pal)  //error
{
    *e = e + 1;
}
for (auto& e : (int*)pal)
{
    cout << e << " ";
}
```

错误: 未定义标识符 "pal"

```
int *pa1 = new int[10]; //数组
    for (auto& e : (int*)pal)
    {
        *e = e + 1;
    }
    for (auto& e : (int*)pal)
    {
        cout << e << " ";
    }
cout<<endl;
    delete []pa1;
```

③开辟并初始化，以及释放

```
int *pa2 = new int(10);        //开辟并初始化
    delete pa2;
```

④开辟对象，以及释放

Test是类名

```
Test *pt = new Test; //1 开辟空间 2 调动构造函数
    delete pt;          //1 调动析构函数 2 释放空间
```

掌握了二中的知识后我们便可以对一中类的模拟实现改造成new开辟delete释放版本如下：

```
class String
{
public:
	String(char *str = "")
	{
		m_data = new char[strlen(str)+1];
		strcpy(m_data, str);
	}
	String(const String &s)
	{
		m_data = new char[strlen(s.m_data)+1];
		strcpy(m_data, s.m_data);
	}
	String& operator=(const String &s)
	{
		if(this != &s)
		{
			free(m_data);
			m_data = new char[strlen(s.m_data)+1];
			strcpy(m_data, s.m_data);
		}
		return *this;
	}
	~String()
	{
		delete []m_data;
		m_data = NULL;
	}
private:
	char *m_data;
};

int main()
{
	String s1;
	String s2("Hello");
	String s3;
	s3 = s2;
	return 0;
}
```

string类模拟实现--… ).cpp
773B

# 总结：

动态内存管理中最容易犯的错误是内存泄漏越界访问

和C中malloc,realloc,calloc必须与free成对使用一样，C++中new和delete也必须成对使用

还有要注意数组名给指针，指针永远拿到的都是首元素地址