计算机学院 高级程序语言设计 课程实验报告

实验题目：综合实验 1		学号：202300130150
日期：2024.5.23	班级： 23.4	姓名： 王成意
实验目的：		

实验步骤与内容：

1. 手动实现一个 MyOStream 类，该类可以实现如下功能。

1）该类支持使用<<运算符打印 char、int、float、double、string 等基本类型。重载函数中可以使用 printf 函数对相应数据进行打印。
2）该类支持自定义打印前缀的功能，默认打印前缀为空字符串，之后每次打印都会先打印前缀后打印给定的数据。
3）设置打印前缀只对下一次的打印有效，之后恢复成空字符串。
4）通过 prefix(string np)成员函数可以将前缀设置为 np。
5）也可以通过 setPrefix(string np)函数在输出流中设置（可以用一个非成员 setPrefix 函数返回包含配置信息的一个特殊对象，让 MyOStream 类重载<<调用对应该配置类时完成设置工作）。

功能函数如下：

```
#ifndef MYO_H

#define MYO_H

#include<cstdio>

#include<string>

using namespace std;

class myostream

{

public:

    string np;

    myostream(string x = "")
```

```
{
        this->np = x;
}
myostream operator << (const int a)
{
    for(auto x:np)
    {
        printf("%c", x);
    }
    printf("%d", a);
    np = "";
    return *this;
}
myostream operator << (const string a)
{
    for(auto x:np)
    {
        printf("%c", x);
    }
    for(auto x:a)
    {
        printf("%c", x);
```

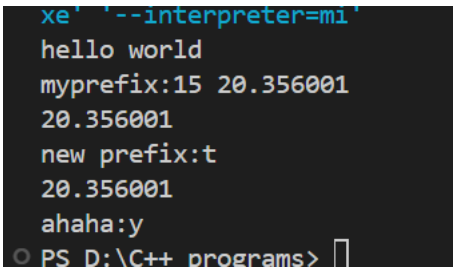
```
        }
        np = "";
        return *this;
    }
    myostream operator << (const char a)
    {
        for(auto x:np)
        {
            printf("%c", x);
        }
        printf("%c", a);
        np = "";
        return *this;
    }
    myostream operator << (const double a)
    {
        for(auto x:np)
        {
            printf("%c", x);
        }
        printf("%lf", a);
        np = "";
```

```
        return *this;
    }
    void prefix(const string a)
    {
        this->np = a;
    }
    myostream operator << (const myostream &a)
    {
        this->np = a.np;
        for(auto x:np)
        {
            printf("%c", x);
        }
        np = "";
        return *this;
    }
};
myostream setprefix(const string a)
{
    myostream x;
    x.np = a;
    return x;
```

```
}

#endif
```

Main 函数如下（加强版）：

```
int a = 15;

    float f = 20.356;

    char ch = 't';

    myostream mout;

    mout << "hello world" << '\n';

    mout << setprefix("myprefix:") << a << " " << f <<
'\n';

    mout << f << '\n';

    mout.prefix("new prefix:");

    mout << ch << "\n"

         << f << "\n"

    << setprefix("ahaha:") << 'y' << "\n";
```

测试结果： 

，符合预期。

2. 手动实现一个 my_auto_ptr 类。实现 12 章 PPT 介绍的 auto_ptr 给出的 get() release() reset()功能。重载*操作符。

代码：

```
#ifndef AUTO_PTR_H

#define AUTO_PTR_H

template <typename T>

class auto_ptr

{

private:

    T *p;


public:

    explicit auto_ptr(T *q=0) throw () : p(q) {}

    T *get() const throw()

    {

        return this->p;

    }

    T * release() throw()

    {

        T *tmp = this->p;

        this->p = 0;

        return tmp;

    }
```

```
    T *reset(T *q=0) throw()
    {
        if(p!=q)
        {
            delete this->p;
            this->p = q;
        }
        return p;
    }
    T& operator*() const throw()
    {
        return *(this->p);
    }
    ~auto_ptr()
    {
        delete this->p;
    }
};


#endif
```

测试代码：

```
#include<cstdio>
```

```cpp
#include<iostream>
#include<string>
#include<vector>
#include"auto_ptr.h"
using namespace std;
class Test{
public:
    Test(){
        cout<<"Test constructed"<<endl;
    }
    ~Test(){
        cout<<"Test deconstructed"<<endl;
    }
};

int main()
{
    auto_ptr<Test> pp(new Test());
    cout << pp.get() << endl;
    cout << pp.release() << endl;
    cout << pp.get() << endl;
    cout << pp.reset(new Test()) << endl;
```

```
    cout << pp.get() << endl;

    return 0;

}
```

测试结果：

```
xe' '--interpreter=mi'
Test constructed
0xe817d0
0xe817d0
0
Test constructed
0xe81810
0xe81810
Test deconstructed
PS D:\C++ programs>
```

分析：pp 首先关联了 new test，get 函数体现了这一点，随后 pp 与这个指针解绑（但是 new 的这个指针还没有 delete，所以整个输出里只有一个析构），随后 pp 与另外一个新 new 的指针建立关系，get 函数也证明了建立成功，随后 main 函数结束，pp 关联的 new 出来的指针也随之自动析构。

验证重载*：

```
int *p = new int;
*p = 2;
auto_ptr<int> pp(p);
cout << *pp << endl;
return 0;
```

结果：

```
xe' '--interpr
2
PS D:\C++ prog
```

，可见重载成功。

结论分析与体会：

非常好的综合实验，使我对 c++的认知更进一步。