应用实例：多项式运算器

养天地正氣 法古今完人

多项式运算器

❖ 逆波兰计算器的简单回

从普通操作数变成
多项式如何变化？

- 输入：
  - ？ —— 表示后续将读入一个操作数
  - +,-,*,/ —— 表示读入了一个运算符
  - = —— 表示表达式输入结束，需要将求解的结果输出
- 算法思想：
  - 如输入的命令为“?”：继续读入一个操作数，用栈存储；
  - 如输入的是“+-*/”等运算符：将最近保存过的数拿出两个作算术运算，并将运算的中间结果存储起来
  - 如输入的命令为“=”：将运算结果即栈顶取出显示

养天地正氣 法古今完人

多项式运算器

- 主程序：唯一的变化是栈中存储的数据元素的类型从普通数值变成了多项式类型

```
 int main( )
/* Post: The program has executed simple polynomial arithmetic
commands entered by the user.
Uses: The classes Stack and Polynomial and the functions
introduction, instructions,do_command, and get_command. */
{
    Stack stored_polynomials;
    introduction( );
    instructions( );
    while (do_command(get_command( ), stored_polynomials));
}
```

多项式运算器

- 基本思想
  - 与普通数值型逆波兰计算器类似，用户输入的命令仍然是字符? , = , + , -, * , /, 含义完全一致，处理对象变成了多项式
  - 假设已经有一个多项式类存在，则所有命令均可调用多项式类中的方法进行——多项式类中有哪些方法呢？

    equals_sum，equals_difference，equals_product，和equals_quotient 分别表示多项式的+,-,*和/运算；
    print和read 表示多项式的打印输出和输入。

多项式运算器

❑ 命令的执行

```
 bool do_command(char command, Stack &stored_polynomials)
/* Pre: The first parameter specifies a valid calculator command.
Post: The command specified by the first parameter has been
applied to the Stack of Polynomial objects given by the second
parameter. A result of true is returned unless command == ‘q’.
Uses: The classes Stack and Polynomial. */
{
     Polynomial p, q, r;
```

多项式运算器

```
switch (command) {
    case ‘?’:
        p.read( );
        if (stored_polynomials.push(p) == overflow)
            cout << "Warning: Stack full, lost polynomial" << endl;
        break;
    case ‘=‘:
        if (stored_polynomials.empty( ))
            cout << "Stack empty" << endl;
        else {
            stored_polynomials.top(p);
            p.print( );
        }
        break;
```

多项式运算器

```
case ‘+’:
        if (stored_polynomials.empty( ))
                cout << "Stack empty" << endl;
        else {
                stored_polynomials.top(p);//get the first operand  to be added
                stored_polynomials.pop( );
                if (stored_polynomials.empty( )) {
                        cout << "Stack has just one polynomial" << endl;
                        stored_polynomials.push(p);
                }
                else {
                stored_polynomials.top(q); //get the second operand  to be added
                stored_polynomials.pop( );
                r.equals_sum(q, p);//do command r=q+p;
                if (stored_polynomials.push(r) == overflow) //push r back to the stack
                        cout << "Warning: Stack full, lost polynomial" << endl;
                        }
        }
        break;
```

多项式运算器

```
    // Add options for further user commands.
    case 'q':
     cout << "Calculation finished." << endl;
    return false;
    }  //end switch
return true;
}
```

多项式运算器

- 借助占位程序编写多项式类进行程序测试

```
class Polynomial {
public:
        void read( );
        void print( );
        void equals_sum(Polynomial p, Polynomial q);
        void equals_difference(Polynomial p, Polynomial q);
        void equals_product(Polynomial p, Polynomial q);
        Error_code equals_quotient(Polynomial p, Polynomial q);
private:
        double value;      //先用简单的数值表示
};
```

```
void Polynomial :: equals_sum(Polynomial p, Polynomial q)
{
        value = p.value + q.value;
}
```

多项式运算器

❖ 遗留问题：

- ➢ 多项式在计算机中如何表示？
- ➢ 多项式的存储使用什么逻辑结构和物理结构？
- ➢ 多项式的基本运算如何实现？

封装性——多项式类的定义及实现