实验报告：简单计算器

涂奕腾 20201018

一、需求分析

1. 用顺序表来完成任意同维度向量的计算，包括加法、减法、夹角余弦值等。
2. 完成任意一元多项式的计算，包括加法、减法、乘法、导数(包括任意阶)等。
3. 四则运算表达式求值。
4. 含单变量的表达式求值。变量可以是 C/C++的标识符。
5. 定义并运行简单函数。
6. 保留函数定义历史，并可以运行历史函数。
7. 函数的调用。比如已经定义了函数 f(X)，新定义函数 g(x)中调用了 f。
8. 支持矩阵的运算。比如矩阵的加、减、乘、转制、特征值、行列式的值等等。

二、概要设计

将任务拆分为 4 种计算类型，逐个分析

1. 向量和多项式

主要使用了线性表的数据结构，以顺序表为例说明。

```
typedef struct{
    ElemType *elem;
    int length,listsize;

    void clear();//清空线性表
    Status init();//初始化线性表
    void destory(); //销毁线性表
    ElemType get(int pos){ }//取线性表上某一位置的元素
    Status insert(int i, ElemType e);//在线性表的第 i 位置插入元素 e
    Status del(int i, ElemType &e);//删除线性表上 i 位置的元素 e
}

Status TransList_Sq(SqList &A, SqList &B);//将线性表 B 转移到线性表 A
上并清空 B
```

其中，应包含如下计算操作：

```
namespace OPT{//用 OPT 命名空间封装运算操作部分
void Add(SqList A, SqList B, SqList &C, int on);//加法和减法操作，
on=1 为加法，on=-1 为减法
void Mul(SqList A, SqList B, SqList &C);//乘法操作
void dev(SqList A, SqList &B);//求导操作
void idev(SqList A, SqList &B);//求不定积分操作
double Cos(SqList A,SqList B);//求夹角余弦操作
}
```

2. 表达式求值

主要使用了串和栈的数据结构，定义如下。

```
template<typename T>
class Stack{//封装成 STL 风格的栈
public:
    Stack();
    ~Stack();
    bool empty();//判断栈是否为空
    T top();//取栈顶元素
    void push(T e);//压栈
    void pop();//出栈

private:
    int stacksize;
    T *Base;
    T *Top;
};
```

```
class String{//封装成 STL 风格的字符串
public:
    String();
    String(char *t);
    Status assign(char *t);//生成一个值等于串常量 t 的串
    Status assign(String t);//复制已有的串 t
    void clear();//清空串
    void insert(int pos, char chr);//在 S 的第 pos 个字符前插入字符 chr
    void insert(int pos, String T);//在 S 的第 pos 个字符前插入串 T
    void del(int pos, int len);//删除自 pos 位置起的 len 个字符
    int compare(String T);//比较两字符串的值
    String substring(int pos ,int len);//第 pos 个字符起长为 len 的子串
    int index(String t, int pos = 1);//字符串匹配
    void replace(String s1, String s2);//将原串中的所有 s1 用 s2 替代
    void print();//输出串
    char *ch;
    int length;
};
```

3. 函数

主要是基于 2. 表达式求值 上进行字符串处理，在数据结构上仅引入了一个一个结构体和计数变量存储函数信息。

```
struct{String name,var,exp;} func[MaxSize];
int cnt = 0;
```

4. 矩阵

以二维数组存储矩阵信息，并封装了矩阵的运算，定义如下：

```
class Matrix{//封装矩阵运算
public:
```

```
    int n, m;//矩阵的行，列

    Matrix(int _n,int _m);
    void clear();//清空矩阵
    ElemType getval(int x,int y);//取第 x 行，第 y 列的值
    void modify(int x, int y, ElemType e);//将第 x 行第 y 列的值增加 e
    void change(Matrix B);//用矩阵 B 覆盖该矩阵
    void print();//矩阵输出
    void transpose();//矩阵转置
    void nummul(ElemType k);//矩阵数乘
    Status add(Matrix B, int on = 1);//矩阵加减法,由 on 控制
    Status mul(Matrix B);//矩阵乘法
    Status qpow(int k);//矩阵快速幂
    Status determinant(double &res);//辗转相除+高斯消元实现行列式求值

private:
    ElemType a[MARSIZE][MARSIZE];
};
```

三、设计细节

1. 向量和多项式

计算部分直接按照计算过程模拟即可，特别注意了对求导后得到的零多项式输出上的处理。

2. 表达式求值

根据课本上的算符优先级，通过栈的数据结构实现表达式求值，其中在表达式的处理上使用了串的结构。而对于单变量的表达式求值，则是提取出单变量名后用赋值对原算式进行字符串替换，转化为普通的表达式求值。

此外加入了一些特判:

(1)小数的判断：除了正常形式的小数外，判断了形如.123 和 123.的小数

(2)负数的判断：特别判断了以“-”开头的表达式，例如 -(3+3)

(3)表达式是否正确的判断：一是判断括号是否匹配，二是判断每次出栈时是否栈空，三是判断结束时算符栈和数栈的情况。

(4)单变量的赋值是否合法的判断，检查变量的赋值是否为数

3. 函数运算

主要是通过串的操作提取出函数名，变量名和表达式部分。特别地，对于函数的嵌套，则是匹配到相应的函数名后使用串的替换来实现。

主要进行了以下特判：

(1)输入的正确性，如赋值的正确性,末尾的“;”等

(2)函数名和变量名的正确性，特别地进行了函数重名错误的判断

(3)表达式的正确性，通过检验表达式求值的合法性实现判断

4. 矩阵运算

在输入部分由于没有使用稀疏矩阵，所以我选择了将整个矩阵完全输入的方式而没有按位置输入。

在除了矩阵转置以外的操作中都检测了操作的合法性。

对于矩阵求幂，使用了快速幂的算法，复杂度优化为$O(n^3 logk)$

对于行列式求值，使用高斯消元法+辗转相除法，均摊复杂度为$O(n^3 + n^2 logn)$