Project02: A Better Calculator

- 作者：罗嘉诚 (Jiacheng Luo)
- 学号：12112910

1 需求分析

结合 `Project2` 给出的说明文档 `Project2.pdf` ，我们分析本次项目需要实现计算器的功能。

`例子01`

```
2+3
5
5+2*3
11
```

`例子02`

```
(5+2)*3
21
```

从上述 `例子01` 和 `例子02` 可以看出，本项目所实现的计算器的基本功能是：对含有 `+` `*` `(` `)` 的表达式，按照约定俗称的运算规则（先乘除、后加减、有括号的先算括号），计算出它的值， 并在终端中输出。

这启示我们从如下几个方面来丰富计算器的内容：

- 一些更人性化的设计，如 `自动过滤多余空格`、`错误信息提示`。
- 支持更多的运算符，如支持 `+`（加） `-`（减） `*` （乘） `/`（除） `%`（取余） `^` （乘方）。
- 支持多种的括号类型，如支持 `[]` 、 `{}` 、 `()` 三种常见括号。
- 支持丰富的数的种类与范围，如支持 `高精度小数` 的上述所有运算。
- 支持更多的数的表示方法，如支持 `常规小数法` 或者 `科学记数法` 表示输入的数据。

`例子03`

```
x=3
y=6
x+2*y
15
```

从上述 `例子03` 可以看出，本项目所实现的计算器应该支持：定义变量、使用变量的值。

这启示我们从如下几个方面来丰富计算器的内容：

- 变量的值应当可以使用表达式进行计算，即按照 `变量名 = 表达式` 的格式进行变量定义。并且较后出现的变量定义时，可以使用先定义变量的值进行计算。
- 变量可以重复定义，变量的值应当为最新定义变量的值。
- 默认定义变量 `last` 保存计算器上次计算获得的数值。（计算器使用时，我们通常需要一个默认变量存储上次计算的值，以进行连续计算）

例子04

```
sqrt(3.0)
1.7
```

从上述 例子04 可以看出，本项目实现的计算器应该支持：一些数学函数的计算。

这启示我们从如下几个方面来丰富计算器的内容：

- 更丰富的默认数学函数支持，如实现 `sqrt()` （开根号） `trunc()` （截取整数） `floor()` （向下取整） `ceil()` （向上取整） `exp()` （自然指数） `ln()` （自然对数） `fac()` （阶乘） `sin()/cos()/tan()` （三角函数） `arcsin()/arccos()/arctan()` （反三角函数）
- 更多目数的默认数学函数支持，如实现 `pow(x, n)` （乘方） `gcd(a, b)` （最大公约数） `lcm(a, b)` （最小公倍数）
- 自定义数学函数支持，采用 `函数名(变量1, 变量2, 变量3, ...) = 表达式` 自定义数学函数。
- 数学函数应该作为一般的运算符，参与计算器处理的其他运算。

例子05

```
99999999999999999999999999999.2222222222 + 1.0
100000000000000000000000000000.2222222222
```

从上述 例子05 可以看出，本项目实现的计算器应该支持：更高精度与范围的科学计算。

这启示我们从如下几个方面来丰富计算器的内容：

- 一些更人性化的设计，如 自动过滤多余空格。
- 支持丰富的数的种类与范围，如支持 高精度小数 的上述所有运算。
- 支持更多的数的表示方法，如支持 常规小数法 或者 科学记数法 表示输入的数据。

关于其他需求，说明文档中给出 `Unix` 系统中的 `BC` 计算器作为例子。关于这一点，我是这么理解的：并不是让我们以此作为标准实现一个一模一样的计算器（事实上，至少以目前我的水平也无法做到），而是关于一些说明文档中没有明确提到的机制，从这个范例计算得出结果并从中借鉴（事实上，我的项目也是这么做的）。

另外，关于项目管理的组织形式，说明文档中也给出了明确的定义：

- 多 `c/cpp` 文件，使用 `CMake` 进行管理。
- 鼓励使用 `GitHub` 和 `Git` 进行项目存储及版本管理。

接下来就是一些常规的需求：

- `Deadline 23:59 Oct. 16`，超时提交 0 分。
- 只完成 需求1 和 需求2 最高得分为 80 分。
- 评分依据为 码风 （推荐使用 `Google C++ Style Guide`）和 项目报告

2 高精度数值型数据相关实现

2.1 存储

在本项目中不区分 `整数` 和 `小数`，而是采用大数类（`BigNumber Class`）来存储高精度数值型数据。

其中重要的参数是 `negative_`、`digits_`、`dotpos_`。

- `negative_` 是一个 `bool` 类型的变量，当高精度数据 `< 0` 的时候，值为 `true` 否则为 `false`。
- `digits_` 是一个 `std::vector<int>` 类型的变量，从低到高位存储高精度数据的每个有效位数值。
- `dotpos_` 是一个 `int` 类型的变量，表示小数点位于 `digits_` 中第几位（从 0 开始）数值的左侧。

为了方便理解，这里给出一个例子：存储高精度数值型数据 `-3.1415926`。

- `negative_ = true`
- `digits_ = [6, 2, 9, 5, 1, 4, 1, 3]`
- `dotpos_ = 7`

特别的，非负整数 0 在本项目中的存储为：

- `negative_ = false`
- `digits_ = [0]`
- `dotpos_ = 0`

项目在大数类（`BigNumber Class`）中实现了三个构造器，分别是：

- `BigNumber();` 空构造，仅用于某些数值运算内部的构造，不对外进行构造。
- `BigNumber(std::string number);` 将 `std::string` 类型的字符串进行构造。
- `BigNumber(int number);` 将 `int` 类型的常规整数值进行构造。

在具体实现中，这三者都基于一个 `void set_BigNumber(std::string);` 类内部函数，使用传入的 `std::string` 类型的字符串，更改当前大数各个重要参数 `negative_`、`digits_`、`dotpos_` 的值。这个类被设置为 `private`，在一定程度上保证了其安全性。

2.2 输出

在本项目中的大数类（`BigNumber Class`）中，实现了一个 `to_string()` 函数。

该函数将高精度数值型数据以 `常规小数法` 转化成一个可供输出的 `std::string` 类型的字符串。

另外，为了便于调试输出，和后续版本的更新，在编写项目的过程中，可以使用 `show()` 方法输出上述提到的 `negative_`、`digits_`、`dotpos_` 三个重要的参数。

为了方便理解，对于上面给出的存储 `-3.1415926` 这个例子，使用：

- `std::string to_string()` 函数，会返回 `std::string` 类型的字符串 `"-3.1415926"`
- `show()` 方法，会在 `终端` 中输出：

```
negative_ = 1
digits_ = [6,2,9,5,1,4,1,3,]
dotpos_ = 7
```

2.3 精度控制

在本项目中的很多数学运算会要求在一定精度下进行计算。

我们定义一个全局变量`scale`，表示进行需要保留精度的数学运算中，保留到小数点后的位数。

仿照 `Unix-BC` 的做法，项目默认 `scale = 0` ，但在计算中可以通过外部输入进行指定。

大数类（`BigNumber Class`）在实现的过程中，主要采用 `checkCarry()` 和 `checkScale()` 两个函数进行精度控制。

- `BigNumber& checkCarry()` 函数用来检查有效位数的 **进位** 并 **去除前导零**，同时检测当前数是否是 `0` ，如果是 `0` 则将当前各项数据调整到合适的值，并返回当前高精度数值型数据的地址。
- `BigNumber& checkScale(int setted_scale)` 函数用来截取当前高精度数值型数据小数点后 `setted_scale` 位之前的所有数据，舍弃这之后的所有数据，保证当前数据的精度。

为了方便理解，对于 `-003.1415926` 这个例子，使用：

- `checkCarry()` 会返回 `-3.1415926`。
- `BcheckScale(3)` 会返回 `-3.141`。

值得注意的是，这只是对于项目精度控制非常粗略的介绍，后续介绍高精度数值的计算部分时，会进一步说明在具体实现的过程中，项目实际上的精度控制是怎样进行的。

2.4 大小比较及其精度控制

2.4.1 大小比较

在本项目中需要大量使用两个高精度数值型数据的大小比较。

为了代码实现的方便，和后期维护的方便，我们对支持大数类（`BigNumber Class`）重载了所有需要使用到的大小比较符号：`<` `<=` `==` `!=` `>=` `>`，下述符号的重载返回 `bool` 类型的值，当等号或者不等号成立时，返回值为`true`，否则为`false`。

- `bool operator < (BigNumber num1, BigNumber num2);`
- `bool operator <= (BigNumber num1, BigNumber num2);`
- `bool operator == (BigNumber num1, BigNumber num2);`
- `bool operator != (BigNumber num1, BigNumber num2);`
- `bool operator >= (BigNumber num1, BigNumber num2);`
- `bool operator > (BigNumber num1, BigNumber num2);`

上述函数的实现，基于 `int BigNumberCmp(BigNumber num1, BigNumber num2);` 函数来实现，该函数实现的功能是：当 `num1 < num2` 时返回 `-1`，当 `num1 = num2` 时返回 `0`，当 `num1 < num2` 时返回`1`。

利用高精度数值型数据的符号，可以将两个数据分成同号和异号两种情况，就只需要考虑异号的情况即可。

此时数值的大小比较只和数值的绝对值相关，此时我们通过`int BigNumberAbsCmp(BigNumber num1, BigNumber num2);` 这个函数去比较它们，该函数实现的功能时：当 `|num1| < |num2|` 时返回`-1` ，当 `|num1| = |num2|` 时返回`0`,当 `|num1| > |num2|` 时返回`1`。

在具体实现该函数时，首先需要将两个数据的小数点的位置对齐，然后比较其所有的有效数字。

由于高精度数值的大小比较和直观理解并没有什么区别，此处就不举例帮助理解了。

2.4.2 精度控制

在上述关于`高精度数值型数据的大小比较`的讨论时，我们考虑的保留的精度是`计算时精度`，即一旦在某精度的计算条件下，高精度数值型数据一旦生成，其精度`不会`随着后续精度`scale`的手动更改而更改。

2.5 四则运算运算及其精度控制

2.5.1 加法和减法运算

在本项目中，对有效位数为 n 的高精度数值型数据进行 `+`（加） `-`（减） `*` （乘） `/`（除） `%`（取模）运算，都采用时间复杂度为 $O(n\log_2 n)$ 及以下的时间复杂度进行实现，并重载了相关运算符 `+` `+=` `-` `-=` `*` `*=` `/` `/=` `%` `%=`。

对于实现高精度数值型数据的加法，直接采用按位相加、模拟进位的朴素方法（当然还需要注意精度对齐的问题），时间复杂度为 $O(n)$。对此，我们重载了 `+` 和 `+=` 运算符：`BigNumber operator + (BigNumber num1, BigNumber num2);`，`BigNumber operator += (BigNumber &num1, BigNumber num2);`

对于实现高精度数值型数据的减法，直接采用按位相减、借位补余数的朴素方法（当然还需要注意精度对齐的问题），时间复杂度为 $O(n)$。对此，我们重载了 `-`和 `-=` 运算符：`BigNumber operator - (BigNumber num1, BigNumber num2);`，`BigNumber operator -= (BigNumber &num1, BigNumber num2);`

由于两个高精度数值型数据进行加减运算，可以分成其绝对值相加和相减两种情况，因此项目中具体实现时，依赖两个函数：

- `BigNumber BigNumberAbsAdd(BigNumber num1, BigNumber num2, bool negative);` 参数`num1` 和 `num2` 分别表示两个加数，`negative` 的值表示计算完毕`|num1| + |num2|` 的值返回数值的符号。
- `BigNumber BigNumberAbsSub(BigNumber num1, BigNumber num2, bool negative);` 参数`num1` 和 `num2` 表示减数和被减数，`negative` 的值表示计算完毕`||num1| - |num2||` 的值返回数值的符号。

2.5.2 乘法运算

对于实现高精度数值型数据的乘法，如果采用朴素的按位模拟，时间复杂度将为达到$O(n^2)$，此时我们采用快速傅里叶变换`FFT`来优化它。

两个高精度数值型数据 `num1` `num2` 进行乘法，其结果为 `num3`，那么：

- 考虑符号：`num3.negative_ = num1.negative_ xor num2.negative_`
- 考虑有效数字：`num3.digits_ = num1.digits_ * num2.digits_`
- 考虑小数点位置：`num3.dotpos_ = num1.dotpos_ + num2.dotpos_`

其中考虑有效数字的乘法，即两个高精度整数相乘，朴素的模拟做法是 $O(n^2)$，我们知道两个有效数字个数级别为 n 的两个大整数相乘，其结果长度依然还是 n 级别的，此处出现了时间复杂度瓶颈。

我们将`num1` 和 `num2` 的有效数字看作两个多项式 $f(x)$ 和 $g(x)$ 的系数，并且用点值表示法表示这两个多项式，如果我们要求答案`num3 = num1 * num2` 其有效数字的点值表示，只需要将乘数的点值表示对应相乘即可，这可以做到$O(n)$完成。

将用系数表示的多项式用点值表示，可采用快速傅里叶变换 `FFT` 优化，时间复杂度为$O(n\log_2 n)$ 。

同样，将点值表示多项式转换成系数表示，也可以采用快速傅里叶变换 `FFT` 优化，时间复杂度为$O(n\log_2 n)$ 。

通过上述算法，我们可以使用$O(n\log_2 n)$ 的复杂度计算两个高精度数值型数据的乘法。

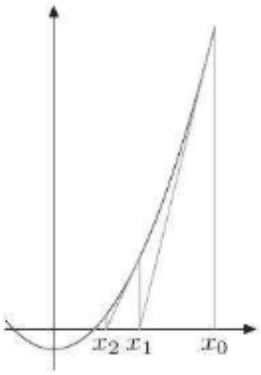
对此，我们重载了 `*` 和 `*=` 运算符：`BigNumber operator * (BigNumber num1, BigNumber num2);`，`BigNumber operator *= (BigNumber &num1, BigNumber num2);`

这两个运算依赖于函数`BigNumber BigNumberAbsMul(BigNumber num1, BigNumber num2, bool negative);`参数`num1`和`num2`分别表示两个乘数，`negative`的值表示计算完毕`||num1| * |num2||`的值返回数值的符号。

2.5.3 除法运算

对于实现高精度数值型数据的乘法，如果采用朴素的按位模拟，时间复杂度将为达到$O(n^2)$，此时我们采用牛顿迭代法来优化它。

牛顿迭代法，是在高等数学课程中重点学习的内容，对于函数$f(x)=0$的一个根x^*，可以选择一个临近这个根的点作为初始点x_0，然后计算该点处函数的切线，交 x 轴为点x'，该点更接近于实际的根x^*，如下图所示：



根据倒数的知识，$x=x_0-\frac{f(x_0)}{f'(x_0)}$，每次以 x 代 x_0 后继续迭代，最终的x_0 的值更接近于所求的根。并且进一步的分析说明，在一些特殊条件满足的情况下，迭代是二阶收敛的（即每迭代一次，有效数字几乎增加一倍）。

为了计算`num1 / num2`我们转化为计算`num1 * (1 / num2)`，由于我们已经实现了$O(n\log_2 n)$ 的乘法运算，接下去的问题就转化为计算`1 / num2`的值。

考虑方程 $ax-1=0$ 在一定精度下计算 $x=\frac{1}{a}$，且迭代式为 $x=x_0-\frac{\frac{1}{x_0}-a}{-\frac{1}{x_0^2}}=2x_0-Ax_0^2$。

可以看见，上述只有减法和乘法运算，都可以在 $O(n\log_2 n)$ 时间复杂度内解决。

进一步的分析可以知道，这样的迭代是二阶收敛的，每迭代一次，有效数字几乎增加一倍。

利用主定理可以计算这个算法的时间复杂度：$T(n)=T(\frac{n}{2})+O(n\log_2 n)$ 可以推出：$T(n)=O(n\log_2 n)$

这个做法在后续实现计算高精度数值型数据的`开平方sqrt()`也会被使用，其可扩展性还是较强的。

通过上述算法，我们可以使用$O(n\log_2 n)$ 的复杂度计算两个高精度数值型数据在一定精度范围内的除法。

对此，我们重载了`/`和`/=`运算符：`BigNumber operator / (BigNumber num1, BigNumber num2);`，`BigNumber operator /= (BigNumber &num1, BigNumber num2);`

这两个运算依赖于函数`BigNumber BigNumberAbsDiv(BigNumber num1, BigNumber num2, bool negative, bool isDiv);`参数`num1`和`num2`分别表示被除数和除数，`negative`的值表示计算完毕`||num1| / |num2||`的值返回数值的符号，`isDiv`表示此次运算是否截取整数。

2.5.4 取模运算

在一定精度下的取模运算`num1 % num2`，其值等于表达式`num1 - (num1 / num2) * num2`的值。

通过上述对于`-` `/` `*`运算的分析，我们可以在 $O(n\log_2 n)$ 的时间复杂度内计算取模运算的结果。

对此，我们重载了`%`和`%=`运算符：`BigNumber operator % (BigNumber num1, BigNumber num2);`，`BigNumber operator %= (BigNumber &num1, BigNumber num2);`

2.5.5 精度控制

- 对于加`+`减`-`乘`*`运算，我们不对结果做精度限制，即保留其真实值的所有精度。

  如对于`scale = 2`的情况下，做`1.123 + 1.232`的结果为`2.355`而不是`2.35`；做`1.123 - 1.232`的结果为`-0.109`而不是`-0.10`；做`1.123 * 1.232`的结果为`1.383536`而不是`1.38`。
- 对于除`/`运算，我们对结果做限制到设定值`scale`的精度限制，舍弃精度以外的所有值。

  如对于`scale = 2`的情况下，做`1.123 / 1.232`的结果为`0.91`。
- 对于取模`%`运算，其精度为`num1 - (num1 / num2) * num2`精度。

  如对于`scale = 2`的情况下，做`1.123 % 1.232`的结果为`0.00188`，而不是`0.00`。

2.6 乘方运算及其精度控制

在高精度数值型数据的乘方运算中，我们认为：$0^0 = 1$。

对于实现高精度数值型数据的乘方运算`num1 ^ num2`，项目分成了不同的情况分别进行处理，每一种处理方法各有优势也各有劣势。

2.6.1 快速幂

当`num2`是整数的情况下（即`num2`和其整数部分`trunc(num2)`相等），采用此方法。

当`num2 = 0`的情况，显然答案为`1`。

当`num2 < 0`的情况，显然`num1 ^ num2 = 1 / (num1 ^ (-num2))`，可以转化到`num2 > 0`的情况。

接下来考虑`num2 > 0`且`num2`是整数的情况。

快速幂考虑将`num2`拆分成二进制的形式，对于`num2`对应是`1`的那些位`k`。计算`num1^(2 ^ k)`的乘积即可，我们只需要将底数`num1`不断平方就可以算出它们。

快速幂做法的时间复杂度为$O(n\log_2 n\log_2 \text{num}_2)$

2.6.2 泰勒展开

当`num2`是不是整数的情况，采用此方法。

考虑泰勒展开 $a^x = e^{x\ln a} = 1 + \frac{x\ln a}{1!} + \frac{x(\ln a)^2}{2!} + \frac{x(\ln a)^3}{3!} + \dots$ 进行计算`num1 ^ num2`，迭代到精度足够时停止。

对于确定的`num1`和`num2`，值`ln(num2)`是确定的值，可以使用后面介绍的高精度数值型数自然对数运算的相关函数`ln()`进行计算。

根据测试数据，我们预测迭代的次数大致与 n 的规模相近，因此这个算法的时间复杂度为 $O(n^2\log_2 n)$。

依据上面两个算法，我们对`num2`是否为整数进行判断，采用不同的算法进行计算。对此，我们重载了`^`和`^=`运算符：`BigNumber operator ^ (BigNumber num1, BigNumber num2);`，`BigNumber operator ^= (BigNumber &num1, BigNumber num2);`

当然，上述算法的实现离不开后面要介绍的两个函数 `BigNumber ln(BigNumber num);`（自然对数函数）及 `BigNumber trunc(BigNumber num);`（取整函数），前者用来计算 `ln(num2)` 的值，后者被 `trunc(num2) == num2` 使用，以检查 `num2` 是否为整数。

2.6.3 精度控制

我们对使用乘方运算进行计算的高精度数值型数据做限制到设定值 `scale` 的精度限制，舍弃精度以外的所有值。

2.7 取整函数及其精度控制

2.7.1 取整函数

本项目在高精度数值型数据取整方法中，主要实现了三种取整方法：

- `BigNumber trunc(BigNumber num);` 直接取整，舍去小数点后的所有数值。
- `BigNumber floor(BigNumber num);` 向下取整，取小于等于 `num` 的最大整数。
- `BigNumber ceil(BigNumber num);` 向上取整，取大于等于 `num` 的最大整数。

不难发现，如果对于 `num` 的正负进行讨论，那么 `floor()` 和 `ceil()` 都是和 `trunc()` 直接相关（最多只相差 `1` 最多只需要进行一次加法或者减法操作即可），而 `trunc()` 的实现非常简单，只需要将小数点后的所有有效位数全部删除即可。

因此，使用 $O(n)$ 的时间复杂度就能实现取整操作。

2.7.2 精度控制

对高精度数值型数据取操作后，其保留小数点后的位数被设置为 `0`。

2.8 开平方运算及其精度控制

2.8.1 开平方运算

在 2.5.3 高精度除法运算中，曾提到利用牛顿迭代法进行开平方运算。

如果考虑直接计算 `sqrt(num)` 需要考虑方程：$x^2 - \text{num} = 0$ 。

利用 牛顿迭代法 得出的迭代式为：$x = x_0 - \frac{x_0^2 - \text{num}}{2x_0} = \frac{x_0}{2} + \frac{\text{num}}{2x_0}$

如果我们考虑方程：$\frac{1}{x^2} - \frac{1}{\text{num}} = 0$ 。

利用 牛顿迭代法 得出的迭代式为：$x = x_0 - \frac{\frac{1}{x_0^2} - \frac{1}{\text{num}}}{-\frac{2}{x_0^3}} = \frac{3x_0}{2} - \frac{x_0^3}{2\text{num}}$

如果我们考虑计算 `frac(1 / A)` ，那么上述的迭代式就变成了：$x = \frac{3x_0}{2} - \frac{x_0^3}{2}\text{num}$

然后使用一次乘法计算 `frac(1 / A) * A` 即可得到精确值了。

根据测试数据，我们预测迭代的次数大致与 n 的规模相近，因此这个算法的时间复杂度为 $O(n^2 \log_2 n)$。

依据上述算法，实现了 `BigNumber sqrt(BigNumber num);` 函数，用于计算高精度数值型数据开平方运算。

2.8.2 精度控制

我们对使用开平方运算的高精度数值型数据做限制到设定值 `scale` 的精度限制，舍弃精度以外的所有值。

2.9 阶乘运算及其精度控制

项目实现了 `BigNumber fac(BigNumber num);` 进行阶乘运算，函数返回 `trunc(num)!`。

由于阶乘运算结果为整数，其保留小数点后的位数被设置为 `0`。

2.10 自然指数运算及其精度控制

2.10.1 自然指数运算

对于高精度数值型数据，计算其自然指数的幂在科学计算中具有重要意义。

项目实现了一个函数 `BigNumber exp(BigNumber num);` 在一定精度范围内用来计算 `e ^ num` 的值。

若 `num = 0` 那么有 `e ^ num = 1` ；若 `num < 0` 那么有 `e ^ num = 1 / (e ^ (-num))`

接下来只需要考虑 `num > 0` 的情况即可。

当 `num > 1` 的时候，`num` 可以采用 `num = num0 * 10 ^ power` 来表示，让 `0 < power <= 1`。

这样，`e ^ num = e ^ (num0 * 10 ^ power) = (e ^ num0) ^ (10 ^ power)` 。

先 使用 $e^x = 1 + x + \frac{x^2}{2} + \ldots + \frac{x^n}{n!} + \ldots$ 泰勒展开式来估计 `(e ^ num0)` 的值，再使用快速幂计算 `(e ^ num0) ^ (10 ^ power)` 得到最终的答案。

时间复杂度为 $O(n^2 \log_2 n)$

2.10.2 精度控制

我们对使用自然指数运算进行计算的高精度数值型数据做限制到设定值 `scale` 的精度限制，舍弃精度以外的所有值。

2.11 自然对数运算及其精度控制

2.11.1 自然对数运算

对于高精度数值型数据，计算其自然对数科学计算中具有重要意义。

项目实现了一个函数 `BigNumber ln(BigNumber num);` 在一定精度范围内用来计算 `ln(num)` 的值。

若 `num = 1` 那么有 `ln(num) = 0` ；若 `num < 0` 那么有 `ln(num) = -ln(1 / num)`

接下来只需要考虑 `num > 0` 的情况即可。

考虑泰勒展开：$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + \ldots$ 当 x 非常接近与 1 的时候下降较快。

我们考虑用$\ln(x) = 2\ln\sqrt{x}$ 进行放缩，先使用泰勒展开计算放缩 k 次之后的$\ln x^{\frac{1}{2^k}}$ 的值

，然后再用这个值乘以2^k 得到原来数字的 ln 值。

时间复杂度为 $O(n^2 \log_2 n)$

2.11.2 精度控制

我们对使用自然对数运算进行计算的高精度数值型数据做限制到设定值 `scale` 的精度限制，舍弃精度以外的所有值。

2.12 三角函数运算及其精度控制

2.12.1 三角函数运算

本项目中，我们还实现了三角函数和反三角函数运算的若干个函数：

- `BigNumber sin(BigNumber num);` 计算正弦值。
- `BigNumber cos(BigNumber num);` 计算余弦值。
- `BigNumber tan(BigNumber num);` 计算正切值。
- `BigNumber arctan(BigNumber num);` 计算反正切值。
- `BigNumber arccot(BigNumber num);` 计算反余切值。
- `BigNumber arcsin(BigNumber num);` 计算反正弦值。
- `BigNumber arccos(BigNumber num);` 计算反余弦值。

利用恒等式：

- $\cos(x) = \sin(x + \frac{\pi}{2})$
- $\tan(x) = \frac{\sin(x)}{\cos(x)}$
- $\operatorname{arccot}(x) = \frac{\pi}{2} - \arctan(x)$
- $\arcsin(x) = \arctan(\frac{x}{\sqrt{1-x^2}})$
- $\arccos(x) = \frac{\pi}{2} - \arcsin(x)$

我们发现只需要求出：`sin()` `arctan(x)` `pi` 的值即可求出上面式子的值。

利用 $\frac{\pi}{2} = \sum_{n=0}^{+\infty} \frac{n!}{(2n+1)!!}$ 可以递推计算出 `pi` 的近似值。

利用泰勒展开，$\sin(x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \dots$. 和规约到$[0, 2\pi)$ 可以计算出 $\sin(x)$ 的值。

利用泰勒展开，$\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \dots$. 和规约到$[0, 2\pi)$ 可以计算出 $\arctan(x)$ 的值。

由于后者在 $|x|$ 接近于 1 或者大于 1 收敛速度非常慢，我们采用如下方法计算：

- 若 `x < 0` 则 `arctan(x) = - arctan(-x)` 把问题约归正实数。
- 若 `x > 1` 利用 `arctan(x)= pi/2 - arctan(1/x)` 把问题约归到$[0, 1]$。
- 若 `x > 0.5` 根据 `arctan(x) = arctan(y) + arctan((x - y) / (1 + xy))`，这里我们取`y = 0.5` 得到 `arctan(x) = arctan(0.5) + arctan((x - 0.5) / (1 + 0.5 * x))`

经过上述优化后，代码时间复杂度为 $O(n \log_2 n)$。

2.12.2 精度控制

我们对使用自然对数运算进行计算的高精度数值型数据做限制到设定值 `scale` 的精度限制，舍弃精度以外的所有值。

3 计算器相关实现

本项目中，通过实现 `calculator` 这个结构体来实现计算器的所有功能。

计算器的相关实现过程中，主要涉及到：变量的处理和表达式处理。

3.1 变量处理

3.1.1 存储

项目中的`calculator`结构体中采用`variableNum` `variableId` `variables`三个重要变量来存储计算器中的所有变量（包括预先定义的`last`特殊变量）。

- `variableNum`是一个`int`类型的变量，用来表示当前已经定义变量的个数。
- `variableId`是一个`std::map<std::string, int>`类型的变量，用来将变量名称映射到其在`variables`中存储的位置。
- `variables`是一个`std::vector<BigNumber>`类型的变量，用来存储各个变量的值。

在使用`calculator()`构造`calculator`结构体时，`variableNum = 1`，`variableId`中只有一个从`std::string "last"`映射到`0`的信息，`variables = [0]`。

3.1.2 调用

项目中采用`BigNumber getval(std::string val_str);`来计算变量名称为`val_str`变量的值，当这个变量没有被找到时，会将`val_str`视为常数，返回它的值。

3.1.3 特殊变量机制

在计算器的实现中有两个较为特殊的变量，即`scale`和`last`。

- `scale`变量是`int`类型的变量，必须通过常规表示法表示的非负整数来表示，表明计算的精度。
- `last`变量是`BigNumber`类型的变量，初始值被设置为`0`，用来表示计算器上一次计算语句的值。

3.2 表达式处理

项目中的语句只包括形如`varName = expression`的赋值语句，以及`expression`的计算语句。

可以发现，赋值语句可以采用上述的变量处理以及计算语句的处理来实现，我们采用`calculator`类中的`void call(std::string statement);`过程来处理这两个语句。

这个过程基于`BigNumber calculate (std::string expression);`函数和`void modify_variables(std::string varName, std::string expression);`过程。

3.2.1 calculate 函数

`void call(std::string statement);`过程基于`BigNumber calculate (std::string expression);`函数的实现，这个函数实现了将中缀表达式`expression`转换为后缀表达式，并计算它的值。

这可以用栈模拟实现，读取并分割数据的时间复杂度是 $O(n)$ 的。

首先定义计算优先级别：

- 先乘方`^`再乘`*` 除`/` 取模`%` 再加`+` 减`-`，同级别从左到右计算。
- 有括号的`()`先计算括号中的结果。
- 函数`Function()`先计算`()`的结果再计算函数的值。

下边是优先级别表（其中`0`表示不会出现这种情况即语法错误，`=`表示匹配，`>`表示左边符号比上面符号优先计算，`<`表示左边符号在上面符号之后计算）：

	+ -	* / %	(	)	#	^	func
+ -	>	<	<	>	>	<	>
* / %	>	>	<	>	>	<	>
(	<	<	<	=	0	<	>
)	>	>	=	0	<	<	>
#	<	<	<	<	=	<	>
^	>	>	<	>	0	>	0
func	<	<	<	0	0	<	<

函数`Procede()`是用来建立符号和数组的映射关系，方便实现。

`#`是表达式的结束符，为了算法简洁，在表达式最左边也虚设一个`#`构成表达式的一对括号。表中`( = )`表示左右括号相遇时，括号内的运算已经完成。同理，`# = #`表示整个表达式求值完毕。

为实现算符优先算法，使用两个栈。一个`std::stack<char> stack_ch`栈，用于寄存运算符；一个`std::stack<BigNumber> stack_num;`栈，用于寄存操作数或运算结果。

算法思想核心思想如下：依此读入表达式中的每个字符，若是操作数则入`stack_ch`栈，若是运算符则和`stack_num`栈的栈顶运算符进行比较优先权后进行相应操作，直至整个表达式求值完毕（即`stack_ch`的栈顶元素和当前读入的字符均为`#`）。

3.2.2 modify_variables 过程

`void call(std::string statement);`过程基于`void modify_variables(std::string varName, std::string expression);`函数的实现，这个函数实现了将表达式`expression`的值进行计算并赋值到`varName`中。

首先依据 calculate 函数计算表达式的值，再使用变量的存储的值进行更改即可。

4 错误处理

由于本项目的核心是实现一个计算器，对错误指令需要区分的精确程度要求较低，因此为了简单起见，本项目的错误处理采用了简单的`exception`异常处理，即哪里出现错了，就`throw`异常，在执行语句时统一处理即可。

项目中有三个异常类，分别是：

- `class number_parse_error : public std::exception`数字格式错误类，用来输出错误的数字格式信息。

1. `Entering a number in the WRONG FORMAT.`错误，输入不是数字格式。

- `class number_calculate_error : public std::exception`数值计算错误类，用来输出进行计算时的错误信息。

1. `The number of SIGNIFICANT DIGITS is too LARGE.`错误，表示有效数字位数超过项目设定的`最长有效数字位数的个数`（初始定义为`1e7`）。
2. `Divisor can NOT be ZERO!`错误，运算过程中出现了除以 0 的错误。
3. `NEGATIVE NUMBER encountered during sqrt().`错误，开平方运算时出现了负数。
4. `NEGATIVE NUMBER encountered during fac().`错误，阶乘运算时出现了负数。
5. `FLOAT NUMBER encountered during fac().`错误，阶乘运算时出现了小数。
6. `NON-POSITIVE NUMBER encountered during ln().`错误，取自然对数时出现了非正数。

7. `Number is OUT of DOMAIN [-1, 1] encountered during arcsin().` 错误，反三角正弦函数出现了定义域外的数。
8. `Number is OUT of DOMAIN [-1, 1] encountered during arccos().` 错误，反三角余弦函数出现了定义域外的数。

- `class expression_parse_error : public std::exception` 表达式格式错误类，用来输出错误的表达式错误信息。

1. `undefined OPERATOR used.` 错误的运算符使用。
2. `undefined FUNCTION called.` 未定义的函数调用。
3. `undefined FUNCTION or VARIALE called.` 未定义的函数或者变量调用。
4. `Bracket can NOT MATCH.` 括号不匹配。
5. `Numbers can NOT be assigned as variables.` 数字不能被赋值。
6. `The value of "scale" can NOT be NEGATIVE.` 精度`scale` 不能被设置成负数。
7. `The value of "scale" MUST be a POSITIVE INTEGER.`精度`scale` 必须被设置成非负整数。
8. `The value of "scale" is too LARGE.` 精度 `scale` 的值被设置成过大值，超过项目设定的`最长有效数字位数的个数` （初始定义为 `1e7`）。

- `std::exception` 输出 `Expression Parsing Error.` 处理过程中的其他错误。

5 项目特色

5.1 高精度支持

项目的实现过程中，所有的高精度数值型数据都可以在保留一定精度的条件下进行数值计算，其中使用了很多数学中的技巧。

5.2 运算效率高

通过各个算法的时间复杂度分析，项目能做到在 $O(n\log_2 n)$ 及以下的时间复杂度进行实现四则运算，在 $O(n^2\log_2 n)$ 及以下的时间复杂度进行各数值计算，可以轻松处理 10^3 精度内的数据。

5.3 多运算符和数学函数支持

项目实现了更多的运算符，如支持 `+`（加） `-`（减） `*` （乘） `/`（除） `%`（取余） `^` （乘方）。

项目实现了更丰富的默认数学函数支持，如实现 `sqrt()` （开根号）`trunc()`（截取整数）`floor()`（向下取整）`ceil()`（向上取整）`exp()` （自然指数）`ln()` （自然对数）`fac()` （阶乘）`sin()/cos()/tan()` （三角函数）`arcsin()/arccos()/arctan()` （反三角函数）等，都在精度范围内得到不错的结果。

5.4 一些更人性化的设计

5.4.1 自动忽略空格

在表达式输入时，项目程序会自动忽略其中的所有空格，方便用户的输入。

5.4.2 错误信息提示

若表达式的处理不符合预期，程序不会崩溃，而是产生错误信息提示用户，并进行接下去的处理。

5.4.3 支持多括号输入

支持多种的括号类型的表达式输入，如支持 `[]` 、`{}` 、`()` 三种常见括号。

5.4.4 支持科学记数法输入

支持更多的数的表示方法，如支持`常规小数法`或者`科学记数法`表示输入的高精度数值型数据。

6 项目及代码

本项目已经上传到了`GitHub`上，推荐您进入该项目阅读我的代码。

项目链接：https://github.com/Maystern/SUSTech_cpp_Project02_a-better-calculator。

您可以在`src`中的`cpp`和`headfile`两个子目录中查阅我的代码。

关于`GitHub`上的项目介绍，请参阅`README.md`文档，关于项目如何运行，介绍如下：

项目推荐在`Ubuntu`环境下运行。

1. 使用命令`git clone https://github.com/Maystern/SUSTech_cpp_Project02_a-better-calculator.git`将项目下载到当前目录。
2. 在当前目录下执行`cd SUSTech_cpp_Project02_a-better-calculator`进入项目根目录。
3. 在项目根目录执行`sh Run_Calculator.sh`命令，该命令执行后，将在`./build`中自动使用`cmake`编译原代码文件，并打开位于`./build/bin`目录下的二进制可执行程序`calculator`。
4. 您只需要在接下去的`终端`中输入命令即可使用计算器。
5. 退出计算器，您只需要输入命令`quit`或者`Crtl + C`即可。

关于代码量，我使用了`cloc`工具进行了统计，结果如下：

```
root@MaysternLaptop:/home/cpp_fall2022/project02# cloc .
      51 text files.
      49 unique files.
      16 files ignored.

github.com/AlDanial/cloc v 1.90  T=0.03 s (1050.4 files/s, 175005.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                              7            145            314           2082
D                                6              0              0            745
C                                1            131             62            610
Markdown                         2            343              0            446
CMake                           11             76             35            369
make                             2            114            123            278
C/C++ Header                     5              0              9            108
Bourne Shell                     1              0              0              6
TypeScript                       1              0              0              2
-------------------------------------------------------------------------------
SUM:                            36            809            543           4646
-------------------------------------------------------------------------------
```

7 程序实现效果

接下来的例子是在项目说明文档`project2.pdf`所列出的 5 个例子，我们都得到了比较合理的输出。

`例子1` 不带括号的整数的加法和乘法

```
问题  输出  调试控制台  终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
2+3
5
5+2*3
11
```

例子2 带括号的整数的加法和乘法

```
问题  输出  调试控制台  终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
(5+2)*3
21
```

例子3 使用数值定义变量并使变量参与加法和乘法运算

```
问题  输出  调试控制台  终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
x=3
y=6
x+2*y
15
```

例子4 保留 1 位小数进行简单的函数运算

```
问题  输出  调试控制台  终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
scale=1
sqrt(3.0)
1.7
```

例子5 较少小数位数的高精度加运算并忽略输入表达式的所有空格

```
问题  输出  调试控制台  终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
99999999999999999999999999.2222222222 + 1.0
100000000000000000000000000.2222222222
```

事实上，项目说明文档 `project2.pdf` 所列出的 5 个例子，并不足以说明项目所做的所有工作。

接下来我将会列出几个更加复杂的例子，来更好的说明项目所做的所有工作。

例子6 项目支持使用 `scale = 正整数` 更改运算的精度

问题 输出 调试控制台 终端

```
root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
scale = 20
sqrt(3)
1.73205080756887729352
1 / 3
0.33333333333333333333
scale = 5
sqrt(3)
1.73205
1 / 3
0.33333
```

例子7 项目支持使用 变量名 = 表达式 的方式定义变量

问题 输出 调试控制台 终端

```
root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
x = 100 / 4 * 3 + sqrt(4) + 2 ^ 2
y = sqrt(x)
x + y
90
```

例子8 项目支持整数数据的+（加）-（减）*（乘）/（除）%（取余）^（乘方）（`scale = 0`）

问题 输出 调试控制台 终端

```
root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
19260817 + 114514
19375331
19260817 - 114514
19146303
19260817 / 114514
168
19260817 % 114514
22465
2 ^ 10
1024
```

例子8 项目支持小数数据的+（加）-（减）*（乘）/（除）%（取余）^（乘方）（精度为`scale`）

问题 输出 调试控制台 终端

```
root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
scale = 3
3.123 + 1.456
4.579
3.123 - 1.456
1.667
3.123 * 1.456
4.547088
3.123 / 1.456
2.144
3.123 % 1.456
0.001336
3.123 ^ 1.456
5.221
```

例子9 项目支持科学记数法整数e整数 或者常规小数法 输入数据和处理数据

问题 输出 调试控制台 终端

```
root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
x = 12.3e-1
x
1.23
scale = 5
sqrt(x)
1.10905
sqrt(1.23)
1.10905
12.3e-1 + 2.3e0
3.53
12.3e-1 * 2.3e-2
0.02829
12.3e-1 - 2.3e-1
1.00
12.3e-1 / 2.3e-2
53.47826
12.3e-1 % 2.3e-2
0.00000002
```

例子10 项目实现多种错误信息提示

问题 输出 调试控制台 终端

```
root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
1e1000000 * 1e10000000
An ERROR occurred while calculating: The number of SIGNIFICANT DIGITS is too LARGE.
1 / 0
An ERROR occurred while calculating: Divisor can NOT be ZERO!
sqrt(-1)
An ERROR occurred while calculating: NEGATIVE NUMBER encountered during sqrt().
fac(-1)
An ERROR occurred while calculating: NEGATIVE NUMBER encountered during fac().
fac(3.1)
An ERROR occurred while calculating: FLOAT NUMBER encountered during fac().
ln(0)
An ERROR occurred while calculating: NON-POSITIVE NUMBER encountered during ln().
arcsin(1.0001)
An ERROR occurred while calculating: Number is OUT of DOMAIN [-1, 1] encountered during arcsin().
arccos(1.0001)
An ERROR occurred while calculating: Number is OUT of DOMAIN [-1, 1] encountered during arccos().
((3.14)))
An ERROR occurred while processing expression: Bracket can NOT MATCH.
1.134 = 1.123
An ERROR occurred while processing expression: Numbers can NOT be assigned as variables.
scale = -1
An ERROR occurred while processing expression: The value of "scale" can NOT be NEGATIVE.
scale = 3.14
An ERROR occurred while processing expression: The value of "scale" MUST be a POSITIVE INTEGER.
scale = 100000000
An ERROR occurred while processing expression: The value of "scale" is too LARGE.
```

例子11 项目支持丰富的默认数学函数

```
问题    输出    调试控制台    终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
scale = 10
trunc(3.14)
3
ceil(3.14)
4
floor(-3.14)
-4
sqrt(3)
1.7320508075
exp(3)
20.0855368954
ln(3)
1.0986122496
fac(3)
6
sin(3)
0.1411200081
cos(3)
-0.9899924962
tan(3)
-0.1425465431
arcsin(0.5)
0.5235987756
arccos(0.5)
1.0471975485
arctan(0.5)
0.4636476090
```

例子12 项目支持多括号输入

```
问题    输出    调试控制台    终端

root@MaysternLaptop:/home/cpp_fall2022/project02/build/bin# ./calculator
scale = 10
(2 + 3) * [1 + 4] * {1 / 4}
6.2500000000
sqrt(sqrt{sqrt[16]})
1.4142135623
```

8 一些问题

在功能上，项目虽然实现了很多功能，但在如下方面可供探讨和完善：

- 更快速计算科学函数：在本项目中部分数学函数是在$O(n^2 \log_2 n)$ 的时间复杂度内解决的，应该还有更快速的解决办法。
- 实现更多的计算器功能：如实现自定义函数、实现多变量数学函数、实现方程求解等。

关于码风方面，虽然与`project01`的码风相比，有很大改善，但还是需要继续学习`Google C++ Style`的相关资料，学习如何进行语法检查。

关于项目管理方面，已经初步使用`CMake`工具让项目看起来更像一个项目，后续需要探索如何使用`GitHub`进行项目的多人协作。

关于指针和内存管理方面，本项目还是采用 `STL` 容器，后续应该加强指针和内存管理的学习和实践，锻炼自己对于指针和内存管理的能力。

9 后记

因为项目编程和报告撰写所花时间有限，本次项目可能存在许多瑕疵和不足，

非常欢迎您对此提出宝贵的批评与意见。

非常感谢您能看到这里!