## Условие

Необходимо разработать программную библиотеку на языке C или C++, реализующую простейшие арифметические действия и проверку условий над целыми неотрицательными числами. На основании этой библиотеки, нужно составить программу, выполняющую вычисления над парами десятичных чисел и выводящую результат на стандартный файл вывода.

Список арифметических операций:

Сложение (+).
Вычитание (-).
Умножение (*).
Возведение в степень ( ^ ).
Деление (/).

Список условий:
Больше (>).
Меньше (<).
Равно (=).

В случае выполнения условия, программа должна вывести на экран строку true, в противном случае — false.

## Описание реализации:

Реализован класс $TBigInt$ каждый элемент которого содержит длинну числа - $Lenght$ основание системы счисления - $BucketSize = 10000$ (основание системы счисления следует брать большим для сокращения времени работы операций. Также удобно взять степнь 10 для упрощения вывода числа на экран) и массив "цифр" реализованный на векторе - $Data$.

**Чтение числа**
В вектор с элементами типа $char$ считывается строка из входного потока и вызывается конструктор. В конструкторе начинается чтение строки с последнего символа и по схеме Горнера символы преобразуются в число. Как только число становится больше чем основание системы счисления или считано больше 4 цифр записываем его, деленное по модулю на основание, в массив цифр и увеличиваем длинну числа на 1. Продолжаем так до тех пор пока не будет обработана вся строка.

**Вывод числа**
Читаем массив цифр с конца. Выводим "цифру" на последнем месте как обычно, а остальные выводим дополняя 0 до 4 цифр в десятичной системе счисления.

**Сложение**

Проверяем длины чисел и вызываем вспомогательную функцию сложения где первым аргументом является число с большей длинной, а вторым - с меньшей. Внутри вспомогательной функции заводим переменную $remain$ для обработки переполнения цифр и начинаем складывать массивы цифр прибавляя к ним значение $remain$, а результат записываем в массив цифр результата. Если сумма цифр больше основания системы счисления, то делим его по модулю и в переменную $remain$ записываем 1, в противном случае записываем туда 0. После того как цифры в меньшем числе закончились начинаем цикл до тех пор пока переполнение не будет равно 0. В цикле мы добавляем к текущей цифре результата переполнение и проверяем не произошло ли снова переполнение если да, то в текущую цифру записываем 0 и переходим к следующей, иначе выходим из цикла. Если же закончились цифры и в большем числе то добавляем 1 в конец массива цифр результата.
Сложность $O(n)$, где $n$ - длина наибольшего числа.

**Вычитание**

В целом аналогично сложению но перед выполнением операции проверяем чтобы вычитаемое было меньше уменьшаемого. Иначе ответ получится отрицательным, что по условию является недопустимым, и вырабатывается исключение. Иначе происходит вычитание реализованное так же как сложение, но с соответствующей заменой сложения цифр на их вычитание.После завершения вычитания вызывается функция затирающая ведущие нули.
Сложность $O(n)$.

**Умножение**

Перед началом операции сравниваются длины чисел и вызывается вспомогательная функция умножения первым аргументом которой является большее число. Внутри вспомогательной функции завдоим переменную для ответа размер которой будет равен произведению длин перемножаемых чисел плюс 1. Начинаем цикл по цифрам второго числа: если эта цифра не 0, то проходим по первому числу и в массив цифр результата по индексу $i+j$ записываем сумму произведения цифр на позициях $i$ и $j$, цифры результата по индексу $i + j$ и переполнения. Целую часть деления этого элемента записываем в переполнение, а сам элемент по модулю делим на основание. Продолжаем цикл до тех пор пока не кончатся цифры во втором числе. После выполнения умножения вызываем функцию затирания ведущих нулей на случай если последняя цифра в массиве цифр осталась не задействованной или произошло умножение на 0.
Сложность $O(nm)$, где $n$ и $m$ длины чисел.

**Деление на короткое число**

Если число равно 0 то генерируем исключение так как деление на 0 запрещено. Заводим переменную $remain$ где будет храниться остаток от деления цифры на короткое число.

Проходим по большому числу в обратном порядке и в соответсвующую цифру результата записываем результат деления суммы остатка умноженного на основание системы счисления и цифры на короткое, а остаток от этого деления записываем в $remain$. Продолжаем до тех пор пока не кончится число. В конце вызываем функцию затирания ведущих нулей на случай если в результате деления длина числа уменьшилась. Сложность $O(n)$.

**Возведение в степень**

Если показатель степени равен 0, то в случае равенства нулю основания генерируем исключение так как это недопустимая операция, иначе возвращаем 1. Если же показатель отличен от нуля то если основание равно 1 или 0 возращаем это число иначе вызываем вспомогательную функцию возедения в степень. Вспомогательная функция устроена довольно просто: если показатель равен 0 - возвращаем 1, иначе если показатель нечетный вызываем рекурсивно эту функцию для показателя на 1 меньше и умножаем результат на основание, в противном случае делим показатель пополам и возводим в квадрат результат возведения в степень деленную пополам.
Сложность $O(\Sigma_{k=1}^{\lceil \log_2 n \rceil} n^{2k})$.

**Деление на длинное число**

Если делитель меньше основания системы счисления то вызывается деление на короткое число. Иначе начинается алгоритм длинного деления. Алгоритм тривиальный - реализовано деление столбиком а внутри цифра ответа подбирается бинарным поиском. В текущее число последовательно добавляются цифры делимого и подбирается наиболее близкое к искомой цифре число. После этого из текущего числа вычитается делитель умноженный на найденную цифру. А в массив цифр ответа записывается найденная цифра. После того как будут обработаны все цифры числа вызываем функцию затирания ведущих нулей.
Сложность $O(nm \log_2(BucketSize))$.

## Выводы

Реализация длинной арифметики не является трудной задачей. Единственное требование это аккуратная реализация и знание алгоритмов умножения и деления из начальной школы. Однако описаное решение не является оптимальным. Так например можно реализовать умножение по алгоритму Карацубы что даст сложность умножения приблизительно $O(n^{1.5})$ также можно оптимизировать деление убрав бинарный поиск.