ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ И ТЕЛЕКОМУНИКАЦИОННЫХ ТЕХНОЛОГИИ РЕСПУБЛИКИ УЗБЕКИСТАН
НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ

КУРСОВАЯ РАБОТА

ПО ПРЕДМЕТУ “ПРОГРАММИРОВАНИЕ НА С++”

НА ТЕМУ: “*Реализовать средствами языка С++ класс больших целых чисел* ”

ВЫПОЛНИЛ: ***СТУДЕНТ 2^{В} КУРСА ПО НАПРАВЛЕНИЮ КОМПЬЮТЕРНОГО ИНЖИНИРИНГА САЙТОВ АЗИЗБЕК***

ПРИНЯЛ: ***ЯДГАРОВ Ш.***

НУКУС 2015

Содержание

План:

Введение

Постановка задачи

Реализовать средствами языка C++ класс больших целых чисел. Для написания класса были выделены следующие задачи:

- *Организовать чтение из консоли и печать в консоль целых чисел, длина которых превышает 232 разрядов (стандартный тип long)*
- *Реализовать выполнение арифметических операции с данными числами:*

o Сложение

o Вычитание

o Произведение

o Нахождение целой части от деления

o Нахождение остатка от деления

o Возведение в степень

Факториал

Замечания о типах данных

C++ имеет гибкие типы данных (см. приложение B по основным типам). Различным программам, например, могут требоваться целочисленные данные различного размера. В C++ есть несколько типов данных для представления целых чисел. Диапазон целочисленных значений для каждого типа зависит от аппаратной части конкретного компьютера. В дополнение к типам int и char в C++ имеются типы short (сокращение от short int) и long (сокращение от long int).Минимальным диапазоном значений для целых типа short является диапазон от -32768 до 32767. Для подавляющего большинства вычислений с целыми числами достаточно типа long, минимальный диапазон значений которого простирается от -2147483648 до 2147483647. На большинстве компьютеров int эквивалентен или short, или long. Диапазон значений типа int по крайней мере такой же, как у short, и не больше, чем у long. Тип данных char пригоден для представления любых символов из набора символов компьютера. Тип char можно также использовать для представления небольших целых чисел.

Переносимость программ.

Поскольку размер типа int варьируется от системы к системе, используйте тип long, если вы предполагаете обрабатывать числа, значения которых могут лежать вне диапазона от -32768 до 32767, и хотите запускать свою программу на нескольких различных компъютерных системах.

Типы данных

long double

double

float

unsigned long int

(синонимичен с unsigned long)

long int

(синонимичен с long)

unsigned int

(синонимичен с unsigned)

int

unsigned short int

(синонимичен с unsigned short)

short int

(синонимичен с short)

unsigned char

char

bool.

Типичная ошибка программирования.

Преобразование от более высокого типа в иерархии возведения к более

низкому может исказить значение данных, что приведет к потере информации.Если аргументы в вызове функции не согласуются с числом и типом параметров, объявленных в соответствующем прототипе, происходит ошибка компиляции. Ошибкой является также случай, когда число аргументов в вызове правильное, но они не могут быть неявно преобразованы к ожидаемым типам.

Заголовочные файлы стандартной библиотеки C++.

Стандартная библиотека C++ разделяется на несколько частей, каждая из которых имеет свой собственный заголовочный файл. Эти заголовочные файлы содержат прототипы родственных функций, входящих в состав каждой из частей библиотеки. Они содержат также определения различных классовых типов и констант, которые необходимы этим функциям. Заголовочные файлы «инструктируют» компилятор, каким образом следует взаимодействовать с библиотечными компонентами и компонентами, которые пишет пользователь.

ПЕРЕМЕННАЯ ПРИМЕР ХАРАКТЕРИСТИКА

int

float

double

char

string

long

OF

"this is

string"

10L

Простые положительные или отрицательные числа, используемые

для перечисления

Действительные числа

Расширенная версия f l o a t : использует больше памяти, допускает работу с большим диапазоном и обеспечивает более высокую точность вычислений

Символьный тип; значением переменных может быть символ алфавита, цифра, знак препинания или знак арифметической операции.

Ход решения

Первым делом была выбрана структура класса больших целых чисел. Так как число может быть как положительным, так и отрицательным было введено символьное поле, отвечающее за знак числа «+» или «–». Само число решено было записывать с помощью очереди с двусторонним доступом (deque) – контейнер из стандартной библиотеки шаблонов (STL). Очередь представляет собой динамический массив с множеством стандартных методов для его обработки. «Цифру» каждого разряда большого числа мы будем помещать в соответствующую ячейку массива. Например, число 12345, записанное с помощью deque<int> mas; будет выгледеть как набор элементов этого массива: mas[0] = 1, mas[1] = 2, mas[2] = 3, mas[3] = 4, mas[4] = 5. Также класс будет содержать некоторое количество методов, для решения поставленных задач. Класс будет иметь название BigInteger. Структура класса изображена на рисунке 1.

Рис. 1. Схема класса BigInteger

Далее перешли к разработке методов класса. Сначала для непосредственной работы с большими числами были реализованы методы для чтения числа из консоли и печати в консоль – chtenie() и vector_print(BigInteger) соответственно.

Метод chtenie() считывает в виде строки данные введенные в консоль. Затем проверяет первый символ строки на наличие знака «–». Потом посимвольно, используя вспомогательную строку, содержащую цифры («0123456789»), заносит каждую цифру в конец очереди-массива. На выходе мы получаем большое число, содержащее знак «», либо «–». Также в методе используется вспомогательный метод dell_null(BigInteger), который возвращает число, удаляя впереди стоящие ничего не значащие нули (т.е. 00123 –> 123).

Метод vector_print(BigInteger) выполняет печать в консоль числа, предварительно задействовав, описанный выше вспомогательный метод dell_null(BigInteger). Сначала происходит печать знака числа, затем, используя цикл, выполняется печать каждого элемента очереди-массива. Результаты работы методов представлены на рисунке 2.

Рис. 2. Ввод-вывод большого положительного и отрицательного числа в консоль

Затем был реализован метод сложения двух больших чисел sumta (BigIneger, BigInteger). Сначала метод проверяет знаки переданных ему чисел, если знаки разные, он, немного модифицируя знаки, передает числа методу, вычисляющему разность двух больших чисел, речь о котором пойдет дальше. Если же знаки чисел одинаковые, непосредственно происходит операция их сложения. Принцип основан на сложении «столбиком». Находим «длину» минимального по разрядам числа, затем складываем поразрядно числа в пределах этой «длины», добавляя 1, если сумма на предыдущем шаге была > 9, находим остаток от деления полученного числа на 10 (int% 10) и записываем его в разряд, над которым выполнялась операция. После того, как разряды у одного из чисел закончатся, «добиваем» результирующее число цифрами из оставшихся разрядов большего числа. Естественно, чтобы произвести операцию с первым разрядом, нужно обратиться к последнему элементу нашей очереди массива. Пример сложения представлен на рисунке 3.

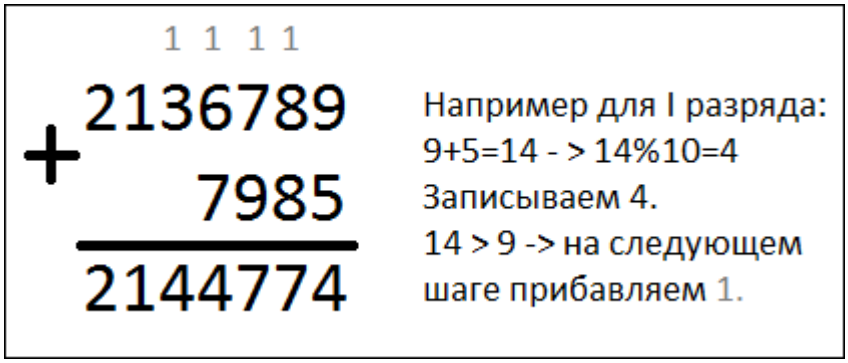

Рис. 3. Операция сложения

Следующей операцией над большими числами стала операция вычитания, представленная методом rasnost (BigInteger, BigInteger). Вначале метод проверяет знаки переданных ему чисел, если знаки чисел равные (с учетом знака разности), то метод передает числа методу summa (BigInteger, BigInteger). Например, 12 – (-7). Метод передаст 12 и 7 для обработки методу summa (BigInteger, BigInteger). Если знаки чисел разные, то происходит операция вычитания. Принцип основан на вычитании «столбиком». Находим число с меньшим количеством разрядов (меньшей «длиной»), ставим его вторым. И начинаем поразрядно вычитать. При этом, если в разряде первого числа значение больше, чем в разряде второго числа, прибавляем к разности этих чисел 10. Но на следующем шаге вычитаем 1 из следующего полученного результата поразрядного вычитания. Выполняем эти действия, пока не закончатся разряды наименьшего числа, затем «добиваем» результирующее число цифрами из оставшихся нетронутыми разрядов большего числа. Пример вычитания представлен на рисунке 4.

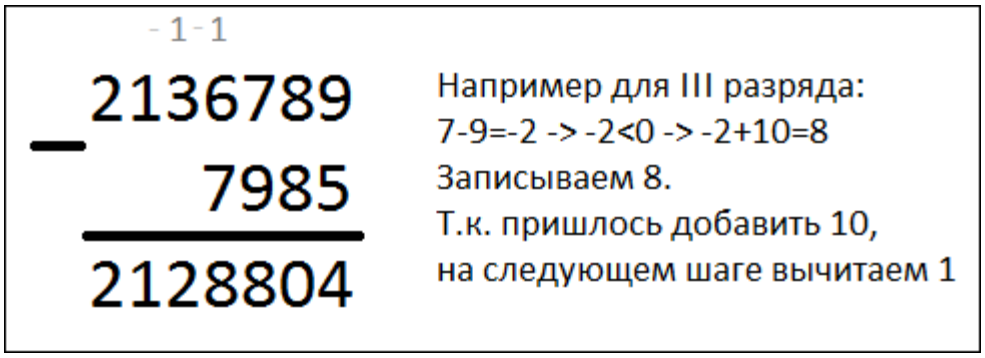

Рис. 4. Операция вычитания

После того, как были реализованы операции сложения и вычитания, перешли к написанию операции умножения, которую выполняет метод proisvedenie (BigInteger, BigInteger). Также в его основе лежит принцип умножения «столбиком». Выбираем одно из больших чисел, и поэтапно умножаем числа из каждого разряда этого большого числа, на другое большое число. В результате на каждом шаге у нас получаются «промежуточные» большие числа, суммируя которые с помощью описанного выше метода summa (BigInteger, BigInteger) мы получаем необходимый нам результат. При этом не забываем в зависимости от разряда множителя «добивать» начальные разряды «промежуточных» больших чисел нулями. Как и в предыдущих операциях при поразрядном перемножении в результат мы записываем остаток от деления на 10 и,

если результат поразрядного перемножения больше либо равен 10, то на следующем шаге перемножения мы прибавляем число равное целой части от деления предыдущего результата на 10. Также для ускорения процесса перемножения были выделены особые случаи – умножение на 0 и на 1. Пример перемножения двух больших чисел представлен на рисунке 5.

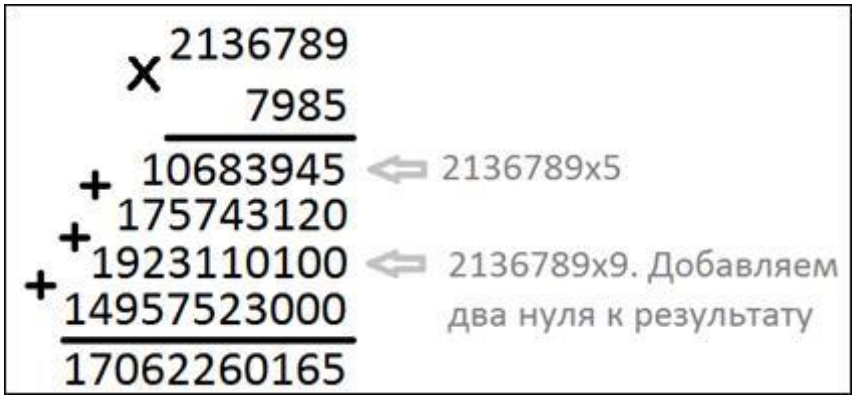

Рис. 5. Операция умножения

*После реализации метода перемножения двух больших чисел операции возведения числа в степень и операция взятия факториала числа не представляют большой трудности, так как могут быть выражены через умножение. Метод stepen (BigInteget, int), представляющий операцию возведения в степень большого числа, принимает в качестве аргументов само большое число и целое число, задающее степень, в которую необходимо возвести большое число. Метод вызывает операцию перемножения числа на само себя в цикле, количество шагов которого равно заданной степени. После выполнения данного цикла получаем нужную нам степень большого числа. Метод factorial(BigInteger), представляющий операцию получения факториала большого числа, мог быть выполнен двумя способами: используя рекурсию или используя итерацию, т.е. цикл. Был выбран второй вариант, так как он более производительный и не требует повторного вызова метода factorial(BigInteger), что замедляло бы работу программы. Для перемножения здесь использовалось вспомогательное большое число, которое изначально приравнивалось к числу, факториал которого нужно было найти. Затем на каждом шаге итерации оно уменьшалось на 1. На каждом шаге это число умножалось на ранее полученный результат, т.е. получалась выражение вида N!=((N*N-1)*N-2*)…*1. После выполнения данного цикла получаем факториал заданного числа.*

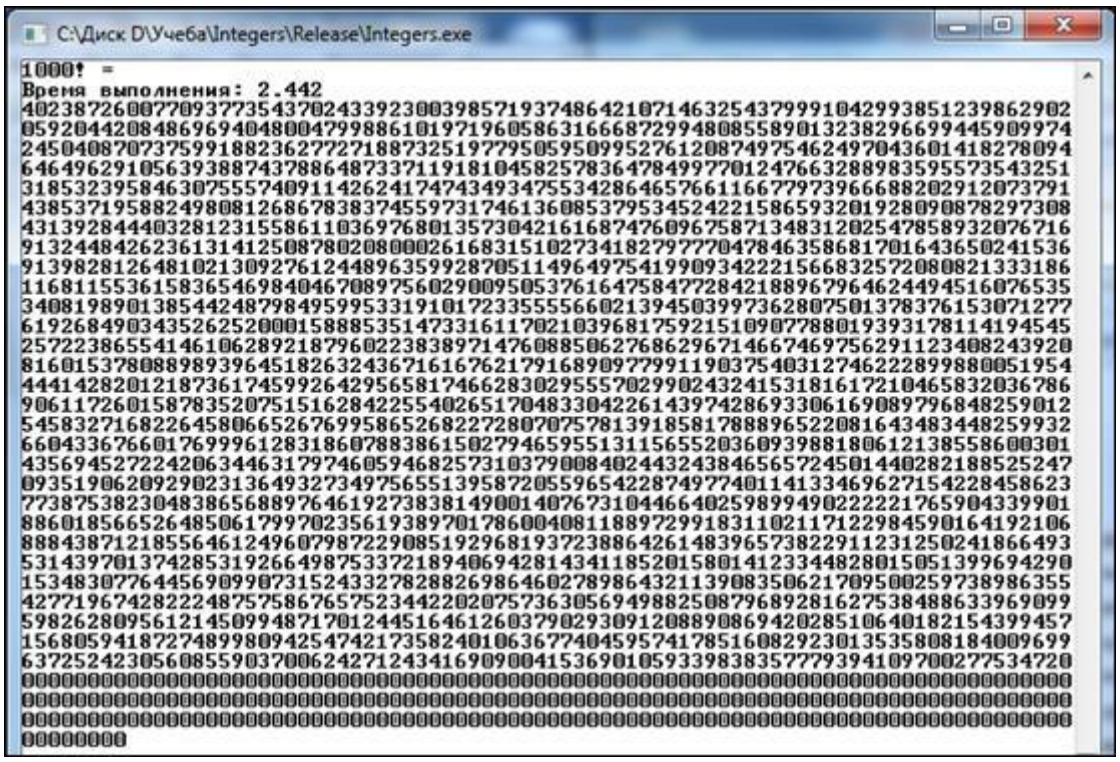

Рисунок 6. Вычисление факториала 1000

Построение проекта осуществлялось в режиме Release. Технические характеристики компьютера, на котором выполнялись расчеты, представлены на рисунке 7.

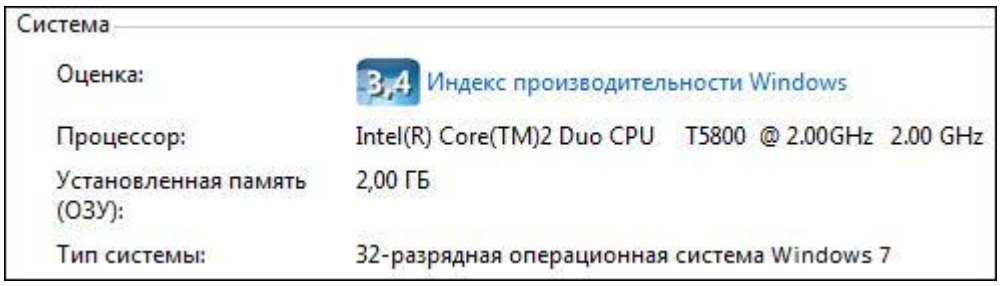

Рисунок 7. Технические характеристики

Для того, чтобы исключить возможное влияние операционной системы на время выполнения вычисления, был проведен численный эксперимент, результаты которого представлены в таблице 1.

Таблица 1. Время вычисления факториала 1000

Порядковый номер вычисления	Время вычисления, сек
1	2,446
2	2,448
3	2,426
4	2,451
5	2,441
6	2,442
7	2,442
8	2,443
Среднее время вычисления	2,442

Порядковый номер вычисления Время вычисления, сек

1 2,446

2 2,448

3 2,426

4 2,451

5 2,441

6 2,442

7 2,442

8 2,443

Среднее время вычисления 2,442

Таким образом, программа вычисляет факториал 1000 в среднем за 2,442 секунды.

Пожалуй, самой сложной для реализации, является операция деления и нахождение остатка от деления двух больших чисел. Методы соответствующие данным операциям были названы delenie (BigInteger, BigInteger) и ostasok_delenie (BigInteger, BigInteger) соответсвенно. В основе лежит принцип деления «столбиком». Пример работы алгоритма приведен на рисунке 8.

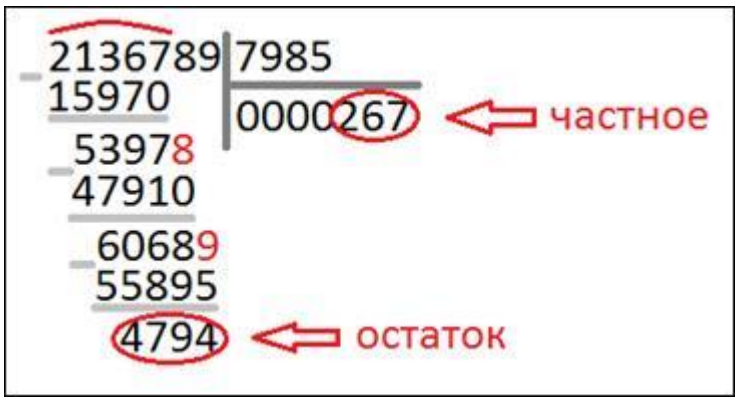

Рис. 8. Операция деления и нахождение остатка от деления

Ход алгоритма следующий: сравниваем делитель с делимым, прибавляя поразрядно по одной цифре к делителю в случае, если получившийся делитель меньше делимого, при этом в частное записываем 0. На рисунке 8 видно этот этап: 2<7985, в частное записываем 0, затем 21<7985, в частное записываем 0, и так далее пока не поменяется знак неравенства 21367>7985. После этого запускается цикл по нахождению следующей цифры частного. На каждом шаге делитель прибавляется на величину равную самому делителю, пока он не станет больше либо равен нашему промежуточному делимому, т.е. 21367. Шаг цикла, на котором выполнится данное условие, и будет искомой цифрой для частного. Затем вычитаем из промежуточного делимого полученное в ходе цикла число и получаем промежуточный остаток. Так как он точно меньше делителя (в связи с предыдущими условиями), добавляем к нему следующую не задействованную цифру делимого и переходим к первому шагу алгоритма. Алгоритм считается выполненным, если получается остаток, меньший делителя и не осталось ни одной незадействованной цифры делимого. В зависимости от задачи, метод возвращает либо частное, либо остаток от деления.

Для удобства пользователей был введен еще один метод vishislenie(), который предоставляет возможность выполнять вышеперечисленные арифметические операции с двумя или одним большим числом путем простого ввода необходимого для вычисления выражения. Пример работы данного метода приведен на рисунке 9.

факториал большой число перемножение

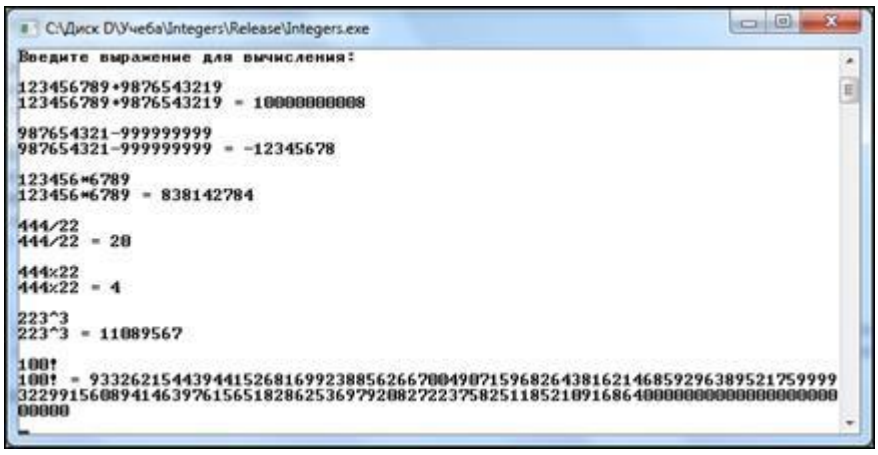

Рис. 9. Режим вычисления выражений

Приложение

Листинг 1. Файл BigInteger.h класс BigInteger.

```
#include <iostream>
#include <deque> // очередь (избиблиотеки STL)
#include <string>
using namespace std;
class BigInteger {
deque<int> vect; // «содержит» число
char znak; // знак числа
public: BigInteger()
{
vect = deque<int>();
znak = ' ';
}
 Сравнение модулей больших чисел
int sravnenie (BigInteger big1, BigInteger big2)
{
if (big1.vect.size() > big2.vect.size()) return 1; // 1, еслипервоечисло > второго
if (big1.vect.size() < big2.vect.size()) return -1; // -1, еслипервоечисло < второго
if (big1.vect.size() == big2.vect.size())
{
for (int i = 0; i < (int) big1.vect.size(); i++)
{
if (big1.vect.at(i) > big2.vect.at(i)) return 1;
if (big1.vect.at(i) < big2.vect.at(i)) return -1;
```

```
}
return 0; // 0, если числа равны
}
}
 Чтение числа из консоли
BigInteger chtenie()
{
BigInteger big;
string temp = «0123456789»; // вспомогательнаястрока
string minus = «–»;
string str;
cin >> str;
if (str.at(0) == minus.at(0)) big.znak = '-'; // определение знака числа
for (int i = 0; i < (int) str.length(); i++) // циклсчитывающийцифрыизстроки
for (int j = 0; j < 10; j++)
if (str.at(i) == temp.at(j)) big.vect.push_back(j);
return dell_null(big);
}
 Функция удаления нулей из начала числа
BigInteger dell_null (BigInteger big)
{
while (big.vect.size() > 1)
{
if (big.vect.at(0)!= 0) break;
else {big.vect.pop_front();}
```

```
}
return big;
}
 Печать числа в консоль
void vector_print (BigInteger big)
{
big = dell_null(big); // убираем нули из начала числа
if (big.vect.size() == 1 && big.vect.at(0) == 0) big.znak = ' '; // если число равно 0, то не ставим знак
if (big.znak == '-') // если число отрицательное, сначала печатаем знак –
cout << big.znak;
for (int i = 0; i < (int) big.vect.size(); i++)
cout << big.vect.at(i);
}
 Сумма больших чисел
BigInteger summa (BigInteger big1, BigInteger big2)
{
if (big1.znak!= big2.znak) // если разные знаки, то отправляем на метод разность
{
if (big1.znak == '-') // заменяем– x+y на y-x
{
big1.znak = ' ';
return rasnost (big2, big1);
}
else // заменяем x+-y на x-y
```

```
{
big2.znak = ' ';
return rasnost (big1, big2);
}
}
deque<int> summa = deque<int>(); // сюда записывается результат
int temp = 0; // 1 для добавления к старшему разряду
int metka = 0; // для вычисления позиции, с которой остаются разряды только одного числа
if (big1.vect.size() >= big2.vect.size()) // ставим большее число на первое место
{
for (int i = big1.vect.size() – 1, j = big2.vect.size() – 1; j >=0; i–, j–) // начиная с первых разрядов складываем числа
{
summa.push_front((big1.vect.at(i) + big2.vect.at(j) + temp)%10);
if ((big1.vect.at(i) + big2.vect.at(j) + temp) >= 10) temp = 1; else temp = 0; // прибавляем 1 наследующемшаге, еслисуммабольше 10
metka = i;
}
for (int i = metka-1; i >= 0; i–) // начиная с позиции метки добиваем цифрами из большего числа, учитывая возможное прибавление 1
{
summa.push_front((big1.vect.at(i)+temp)%10);
if ((big1.vect.at(i) + temp) == 10) temp = 1; else temp = 0;
}
if (temp == 1) summa.push_front(1); // срабатывает в случае когда увеличивается разряд, например 99+1=100
```

```
}
else
{
for (int i = big2.vect.size() – 1, j = big1.vect.size() – 1; j >=0; i–, j–)
{
summa.push_front((big2.vect.at(i) + big1.vect.at(j) + temp)%10);
if ((big2.vect.at(i) + big1.vect.at(j) + temp) >= 10) temp = 1; else temp = 0;
metka = i;
}
for (int i = metka-1; i >= 0; i–)
{
summa.push_front((big2.vect.at(i)+temp)%10);
if ((big2.vect.at(i) + temp) == 10) temp = 1; else temp = 0;
}
if (temp == 1) summa.push_front(1);
}
big1.vect = summa;
return big1;
}
 Разностьбольшихчисел
BigInteger rasnost (BigInteger big1, BigInteger big2)
{
if (big2.znak == '-') big2.znak = ' '; // x–y преобразуем в x+y и передаем в метод суммы
else big2.znak = '-';
```

if (big1.znak == big2.znak) return summa (big1, big2); // – x-y преобразуемв– (x+y) передаемметодусуммы

deque<int> rasn = deque<int>(); // сюда записывается разность

int temp = 0; // 1 для вычитания из старшего разряда

int metka = 0; // для вычисления позиции, с которой остаются разряды только одного числа

big1 = dell_null(big1); // предварительно удаляем незначащие нули из начала числа

big2 = dell_null(big2);

if (sravnenie (big1, big2)!= -1) // ставим большее число сверху в столбике

{

for (int i = big1.vect.size() – 1, j = big2.vect.size() – 1; j >=0; i–, j–)

{

if ((big1.vect.at(i) – big2.vect.at(j) + temp) >= 0) // поразрядновычитаем

{

rasn.push_front (big1.vect.at(i) – big2.vect.at(j) + temp);

temp = 0;

}

else

{

rasn.push_front (big1.vect.at(i) – big2.vect.at(j) + 10 + temp); // заимствуем 1 изстаршегоразряда

temp = -1;

}

metka = i;

}

```
for (int i = metka-1; i >= 0; i–) // добиваем числами оставшихся разрядов, учитывая -1
{
rasn.push_front (abs((big1.vect.at(i)+temp+10)%10));
if ((temp == -1) && (big1.vect.at(i) + temp) < 0) temp = -1; else temp = 0;
}
big1.vect = rasn;
return big1;
}
else
{
for (int i = big2.vect.size() – 1, j = big1.vect.size() – 1; j >=0; i–, j–)
{
if ((big2.vect.at(i) – big1.vect.at(j) + temp) >= 0)
{
rasn.push_front (big2.vect.at(i) – big1.vect.at(j) + temp);
temp = 0;
}
else
{
rasn.push_front (big2.vect.at(i) – big1.vect.at(j) + 10 + temp);
temp = -1;
}
metka = i;
}
for (int i = metka-1; i >= 0; i–)
```

```
{
rasn.push_front (abs((big2.vect.at(i)+temp+10)%10));
if ((temp == -1) && (big2.vect.at(i) + temp) < 0) temp = -1; else temp = 0;
}
big2.vect = rasn;
return big2;
}
}
```

Произведение больших чисел

```
BigInteger proisvedenie (BigInteger big1, BigInteger big2)
{
BigInteger proisv;
proisv.vect.push_back(0);
BigInteger reserv;
BigInteger reserv2;
for (int i = big1.vect.size() – 1, count = 0; i >= 0; i –, count++)
{
if (big1.vect.at(i) == 0) {} // умножениена 0
else
if (big1.vect.at(i) == 1) // умножение на 1, просто прибавляем число с «добитыми» нулями
{
reserv2.vect = big2.vect;
for (int k = 0; k < count; k++) // добиваем нулями в зависимости от разряда умножения
reserv2.vect.push_back(0);
```

```
proisv = summa (reserv2, proisv);
}
else
{
int temp = 0;
for (int k = 0; k < count; k++) // добиваемнулями
reserv.vect.push_front(0);
for (int j = big2.vect.size() – 1; j >=0; j–) // умножаемпервоечислона«цифру» изразрядаучитывая temp
{
reserv.vect.push_front((big1.vect.at(i)*big2.vect.at(j) + temp)%10);
if ((big1.vect.at(i)*big2.vect.at(j) + temp) >=10) temp = (big1.vect.at(i)*big2.vect.at(j) + temp)/10; else temp = 0;
}
if (temp!=0) reserv.vect.push_front(temp); // приувеличениразрядовчисла
proisv = summa (reserv, proisv); // складываем предыдущие результаты
reserv.vect.clear();
}
}
if (big1.znak!= big2.znak)
proisv.znak = '-';
return proisv;
}
 Возведение в степень большого числа
BigInteger stepen (BigInteger big, int steps)
{
```

```
BigInteger step;
//deque<int> step = deque<int>();
step.vect = big.vect; // постоянный множитель
for (int i = 1; i < steps; i++) // числошаговравноестепени
big = proisvedenie (big, step);
if (steps% 2 == 0)
big.znak = ' ';
return big;
}
Факториал большого числа
BigInteger faktorial (BigInteger big)
{
big.znak = ' ';
BigInteger fak;
fak.vect.push_back(1);
BigInteger edinica;
edinica.vect.push_back(1); // дляуменьшенияна 1
{
while (big.vect.size()!= 0 && big.vect.at(0)!= 0) // пока число не стало равным 0
{
fak = proisvedenie (big, fak);
big = rasnost (big, edinica);
big = dell_null(big);
fak = dell_null(fak);
}
```

```
}
return fak;
}
 Деление больших чисел
BigInteger delenie (BigInteger delimoe, BigInteger delitel)
{
BigInteger chastnoe;
BigInteger ostatok;
BigInteger reserv2;
BigInteger reserv3;
reserv2.vect = delitel.vect;
for (int i = 0; i < (int) delimoe.vect.size(); i++)
{
ostatok = dell_null(ostatok);
ostatok.vect.push_back (delimoe.vect.at(i)); // промежуточныйостаток
if (sravnenie (ostatok, delitel) == -1) {chastnoe.vect.push_back(0);} // покапромежуточныйостатокбольшеделителяпишемвчастное 0
else
{
for (int j = 0; j < 10; j++) // цикл, формирующий цифры частного
{
if (sravnenie (ostatok, reserv2) == -1) // промежуточный остаток меньше делителя*j
{
chastnoe.vect.push_back(j);
ostatok = rasnost (ostatok, reserv3);
```

```
reserv2.vect = delitel.vect;

break;

}

if (sravnenie (ostatok, reserv2) == 0) // промежуточныйостатоккратныйделителю

{

chastnoe.vect.push_back (j+1);

ostatok.vect.clear();

reserv2.vect = delitel.vect;

break;

}

reserv3 = reserv2;

reserv2 = summa (reserv2, delitel); // прибавляем сам делитель, пока не станет больше остатка

}

}

} // цифры делимого заканчиваются и остаток меньше делимого, цикл завершается

if (delimoe.znak!= delitel.znak) chastnoe.znak = '-';

return chastnoe;

 Остаток от деления больших чисел.

BigInteger ostatok_delenie (BigInteger delimoe, BigInteger delitel)

{ // все как в методе delenie(), только возвращаем не частное, а остаток

BigInteger chastnoe;

BigInteger ostatok;

BigInteger reserv2;

BigInteger reserv3;
```

```
reserv2.vect = delitel.vect;
for (int i = 0; i < (int) delimoe.vect.size(); i++)
{
ostatok = dell_null(ostatok);
ostatok.vect.push_back (delimoe.vect.at(i));
if (sravnenie (ostatok, delitel) == -1) {chastnoe.vect.push_back(0);}
else
{
for (int j = 0; j < 10; j++)
{
```

Заключения

Таким образом, в ходе написания программы, поставленные задачи были полностью выполнены: сделаны механизмы чтения и печати в консоль больших чисел, реализованы арифметические операции. Производительность программы, на мой взгляд, достаточно высокая, что было достигнуто за счет некоторой оптимизации кода (в основном в результате переработки операции произведения). Небольшое замедление программы возможно вследствии того, что использован контейнер стандартной библиотеки шаблонов. Этот недостаток компенсируется простотой использования стандартного контейнера и множеством поставляемых с ним методов отладки, проверкой ошибок на этапе компиляции программы и механизмом исключений. Немаловажным является реализация метода предоставляющего пользователям в удобной форме производить вычисления с большими целыми числами.

Список используемой литературы

1. Лаптев В.В., Морозов А.В. «Объектно-ориентированное программирование. Задачи и упражнения». Издательство: «Питер» 2007 г.

2. Лафоре Р. «Объектно-ориентированное программирование в C++». Издательство: «Питер», 2004 г.