МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет о выполнении курсовой работы
«Калькулятор «большой» конечной арифметики»
Дисциплина «Дискретная математика»
Вариант 33

Выполнил студент группы
№5130201/30001 _____ Мелещенко С.И.

Проверил _____ Востров А. В.

Санкт-Петербург, 2024

Содержание

Введение

Данный отчет представляет собой описание выполнения курсовой работы на тему «Калькулятор «большой» конечной арифметики». Работа выполнена в рамках изучения дисциплины «Дискретная математика».

Целью работы являлась реализация калькулятора, выполняющего операции «большой» конечной арифметики <Zi;+,*> с 8-разрядными числами. Основными действиями, которые требовалось реализовать, стали сложение, вычитание, умножение и деление, основанные на правилах «малой» конечной арифметики. Для правила «+1» представленного ниже (см. [Рис. 1]).

a	b	c	d	e	f	g	h
b	e	d	g	h	a	f	c

Рис. 1: Правило «+1»

При разработке калькулятора учитывались следующие свойства алгебраической структуры.

1. Коммутативность операций сложения и умножения.
2. Ассоциативность операций сложения и умножения.
3. Дистрибутивность умножения относительно сложения.
4. Наличие аддитивной единицы a.
5. Наличие мультипликативной единицы b.
6. Для любого элемента x верно x*a=a.

Работа направлена на изучение теоретических основ конечной арифметики и их практическое применение для создания программных решений. Полученные результаты имеют значение как в образовательном контексте, так и в более широких областях, связанных с алгоритмами обработки данных и проектированием вычислительных систем.

Программа была реализована на языке программирования C++ в среде программирования Microsoft Visual Studio.

1 Математическое описание

1.1 Алгебраические структуры

Словом «алгебра» называют раздел математики, посвященный изучению систем операций над элементами некоторого множества.

Множество M вместе с набором операций $\sum = \{\varphi_1, ..., \varphi_m\}, \varphi_i : M^{n_i} \longrightarrow M$, где n_i - арность операций φ_i , называется алгебраической структурой, универсальной алгеброй или просто алгеброй. При этом множество M называется основным (несущим) множеством или основной (носителем); вектор арностей $(n_1, ..., n_m)$ называется типом; множество операций $\sum$ называется сигнатурой.

Малая конечная арифметика - это конечное коммутативное ассоциативное кольцо с единицей $< Z_8; +, * >$, где определены действия вычитания и деления, причем деление определено частично. Это система вычислений, основанная на заданном конечном множестве чисел, где операции выполняются с учетом специальных правил.

Большая конечная арифметика — это конечное коммутативное ассоциативное кольцо с единицей $< Z_n^i ; +, * >$, где определены действия вычитания и деления, причем деление определено с остатком. В нашей реализации n = 8 - количество раз- рядов числа, i = 8 - количество символов носителя (алфавита): $< Z_8^8 ; +, * >$. Это обобщение малой конечной арифметики, предназначенное для выполнения вычислений с числами, имеющими больший разрядный диапазон.

1.2 Свойства операций

Некоторые часто встречающиеся свойства операций имеют специальные названия. Пусть задана алгебра

$$\langle M; \Sigma \rangle \text{ и } a, b, c \in M; \circ, \diamond \in \Sigma; \circ, \diamond : M \times M \to M$$

1) ассоциативность

$$\forall a, b, c \in M \ ((a \circ b) \circ c = a \circ (b \circ c))$$

2) коммутативность

$$\forall a, b \in M \ (a \circ b = b \circ a)$$

3) дистрибутивность

слева: $\forall\, a, b, c \in M \;\; (a \diamond (b \circ c) = (a \diamond b) \circ (a \diamond c))$;
справа: $\forall\, a, b, c \in M \;\; ((a \circ b) \diamond c = (a \diamond c) \circ (b \diamond c))$;

4) поглощение

справа: $\forall\, a, b \in M \;\; ((a \circ b) \diamond a = a)$;
слева: $\forall\, a, b \in M \;\; (a \diamond (a \circ b) = a)$;

5) идемпотентность

$$\forall\, a \in M \;\; (a \circ a = a)$$

6) нейтральный элемент

левый: $\exists\, e \in M \;\; (\forall\, a \in M \;\; (e \circ a = a))$
правый: $\exists\, e \in M \;\; (\forall\, a \in M \;\; (a \circ e = a))$

7) сократимость

слева : $\forall\, a, x, y \in M \;\; (a \circ x = a \circ y \Longrightarrow x = y)$
справа: $\forall\, a, x, y \in M \;\; (x \circ a = y \circ a \Longrightarrow x = y)$

8) обратный элемент

слева: $a^{-1} \circ a = e$,
справа: $a \circ a^{-1} = e$.

9) разрешимость

$$\forall\, a, b \in M \;\; (\exists !\, x, y \in M \;\; (a \circ x = y \circ a = b))$$

1.3 Построение таблиц сложения и умножения

Диаграмма Хассе:
$a - b - e - h - c - d - g - f$
Составим таблицу сложения (см. [Рис. 2]).

+	a	b	c	d	e	f	g	h
a	a	b	c	d	e	f	g	h
b	b	e	d	g	h	a	f	c
c	c	d	a	b	g	h	e	f
d	d	g	b	e	f	c	h	a
e	e	h	g	f	c	b	a	d
f	f	a	h	c	b	g	d	e
g	g	f	e	h	a	d	c	b
h	h	c	f	a	d	e	b	g

Рис. 2: Таблица сложения

Пример рассчетов:
$a + e = e$
$d + e = d + (b + b) = (d + b) + b = g + b = f$
$c + g = c + (b + d) = (c + d) + b = b + b = e$
$e + e = e + (b + b) = (e + b) + b = h + b = c$
Ниже приведена таблица для смещения по сложению (см. [Рис. 3]).

+ S	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a
b	a	a	a	a	a	b	a	a
c	a	a	b	b	a	b	b	a
d	a	a	b	b	a	b	b	b
e	a	a	a	a	a	b	b	a
f	a	b	b	b	b	b	b	b
g	a	a	b	b	b	b	b	b
h	a	a	a	b	a	b	b	a

Рис. 3: Таблица смещения по сложению

Смещения можно определить по диаграмме Хассе. Например, (f + e) проходит из конца в начало один раз, поэтому смещение и b.

Также была составлена таблица для умножения.

*	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a
b	a	b	c	d	e	f	g	h
c	a	c	a	c	a	c	a	c
d	a	d	c	b	e	h	g	f
e	a	e	a	e	c	g	c	g
f	a	f	c	h	g	b	e	d
g	a	g	a	g	c	e	c	e
h	a	h	c	f	g	d	e	b

Рис. 4: Таблица умножения

Пример рассчетов:
$b * a = a$
$d * b = d$
$d * f = d * (b + g) = d * b + d * g = d + g = h$
$c * g = c * (c + e) = c * c + c * e = a + a = a$
$h * h = h * (c + f) = h * c + h * f = c + d = b$
Ниже приведена таблица для смещения по умножению (см. [Рис. 5]).

* S	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a
b	a	a	a	a	a	a	a	a
c	a	a	e	e	b	h	h	b
d	a	a	e	h	b	c	h	b
e	a	a	b	b	a	b	b	a
f	a	a	h	c	b	g	d	e
g	a	a	h	h	b	d	c	e
h	a	a	b	b	a	e	e	b

Рис. 5: Таблица смещения по умножению

2 Особенности реализации

2.1 Библиотечные классы

iostream - редоставляет функции для работы с вводом и выводом, такие как std::cin и std::cout.

vector - для использования динамических массивов (std::vector) и операций с ними, таких как добавление и удаление элементов.

string - предоставляет класс std::string для работы со строками, включая их манипуляцию и преобразование.

2.2 Класс Arithmetic

2.2.1 Поля класса

char alph[8] - массив символов, представляющий алфавит, используемый для операций. В данном случае содержит символы 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'.

char poryadok[8] - массив символов, задающий порядок следования символов для работы с арифметическими операциями.

vector<char> plusone - вектор, задающий смещение (следующий символ) для операций сложения.

string a - строка, хранящая первое вводимое число для выполнения арифметических операций.

string b - строка, хранящая второе вводимое число для выполнения арифметических операций.

char Sum[8][8] - таблица сложения, содержащая результаты сложения символов алфавита.

char SumSdvig[8][8] - таблица сдвига при сложении, используемая для учета переноса между разрядами.

char Mult[8][8] - таблица умножения, содержащая результаты умножения символов алфавита.

char MultSdvig[8][8] - таблица сдвига при умножении, используемая для учета переноса между разрядами.

2.2.2 Конструтор класса

Это конструктор класса Arithmetic, который инициализирует объект. Поступающий на вход пустой вектор plusone (правило "+1") заполняется значениями, также поступающий на вход пустой массив poryadok (отношения порядка) заполняется значениями. Вызываются методы создания таблиц. В качестве результата создается объект класса Arithmetic с заранее определенными значениями и таблицами.

Листинг 1: Arithmetic::Arithmetic()

```
plusone.push_back('b');
plusone.push_back('e');
plusone.push_back('d');
plusone.push_back('g');
plusone.push_back('h');
plusone.push_back('a');
plusone.push_back('f');
plusone.push_back('c');

poryadok[0] = 'a';
poryadok[1] = 'b';
poryadok[2] = 'e';
poryadok[3] = 'h';
poryadok[4] = 'c';
poryadok[5] = 'd';
poryadok[6] = 'g';
poryadok[7] = 'f';

fillTableSum();
fillTableSdvigSum();
fillTableMult();
fillTableSdvigMult();
```

2.2.3 fillTableSum()

Вход: пустой массив Sum (таблица сложения), alph (алфавит), plusone (правило "+1").

Выход: заполненная таблица Sum (таблица сложения).

Первый цикл заполняет строки и столбцы таблицы для индексов 0 и 1. Sum[i][0] и Sum[0][i] устанавливают, что сложение с "нулевым элементом"дает сам элемент из алфавита alph[i]. Sum[i][1] и Sum[1][i] заполняются значениями из массива plusone, что соответствует добавлению "единицы"к каждому элементу.

Второй цикл. Для индексов i и j начиная с 2, функция вычисляет результат сложения символов alph[i] и alph[j] с помощью функции twoSum. Sum[i][j] — результат сложения символов alph[i] и alph[j]. Sum[j][i] устанавливается равным Sum[i][j], что подразумевает коммутативность операции сложения.

Листинг 2: void Arithmetic::fillTableSum()

```
for (int i = 0; i < 8; i++) {
Sum[i][0] = alph[i];
Sum[0][i] = alph[i];
```

```
        Sum[i][1] = plusone[i];
        Sum[1][i] = plusone[i];
}

for (int i = 2; i < 8; i++) {
        for (int j = i; j < 8; j++) {
                char buf = twoSum(alph[i], alph[j]);
                Sum[i][j] = buf;
                Sum[j][i] = buf;

        }
}
```

2.2.4 Метод twoSum(char a, char b)

Вход: char a (цифра 1) и char b (цифра 2), plusone и alph.

Выход: char result — результат сложения символов a и b.

Пока check не станет равным 'b' buf обновляется на следующий элемент алфавита (на основе индекса, который возвращает searchIndex). check обновляется на следующий символ в последовательности plusone для текущего buf. Увеличивается счетчик k, который отслеживает количество шагов, необходимых для преобразования b в 'b'. Подсчитываем сколько шагов необходимо, чтобы "обратиться"от символа b к 'b'.

Пока счетчик k больше нуля result обновляется на следующий символ в последовательности plusone, начиная с a. Счетчик k уменьшается.

Функция возвращает результат сложения, вычисленный как result.

Листинг 3: char Arithmetic::twoSum(char a char b)

```
        int k = 0;
        char result = a;
        char buf = b;
        char check = b;
        while (check != 'b') {
                buf = alph[searchIndex(buf)];
                check = plusone[searchAlph(buf)];
                k++;
        }
        while (k > 0) {
                result = plusone[searchAlph(result)];
                k--;
        }
        return result;
```

2.2.5 Метод fillTableSdvigSum()

Функция fillTableSdvigSum заполняет двумерную таблицу SumSdvig, содержащую результаты специальных операций над символами, которые зависят от их порядковых значений в массиве poryadok.

Входные параметры: alph, poryadok, незаполненная SumSdvig (таблица сдвигов по сложению).

Выходные данные: заполненная таблица SumSdvig.

Внешний цикл проходит по всем индексам массива alph, внутренний цикл также проходит по всем индексам массива alph. Функция обрабатывает все возможные пары символов alph[i] и alph[j]. searchPoryadok(alph[i]) возвращает индекс символа alph[i] в массиве poryadok. Значения индексов символов alph[i] и alph[j] суммируются. Результат делится на 8 с использованием целочисленного деления (/ 8), чтобы вычислить "группу к которой относится результат. Индекс tmp, вычисленный на предыдущем шаге, используется для извлечения символа из массива poryadok. Этот символ записывается в таблицу SumSdvig на позицию [i][j].

Листинг 4: void Arithmetic::fillTableSdvigSum()

```
                    for (int i = 0; i < 8; i++) {
                    for (int j = 0; j < 8; j++) {
int tmp = searchPoryadok(alph[i]) + searchPoryadok(alph[j]);
tmp = tmp / 8;
SumSdvig[i][j] = poryadok[tmp];
                    }
        }
```

2.2.6 Метод fillTableMult()

Вход: alph, незаполненная Mult (таблица умножения).

Выход: заполненная таблица Mult.

Инициализация базовых значений:

Умножение любого символа на 'a' (нулевой элемент): результат всегда 'a'. Умножение любого символа на 'b' (единичный элемент): результат равен самому символу.

Заполнение основной части таблицы:

Для пар символов с индексами от 2 до 7 функция вызывает вспомогательную функцию twoMult, которая выполняет специфическое умножение. Результат умножения симметрично записывается в ячейки [i][j] и [j][i].

Листинг 5: void Arithmetic::fillTableMult()

```
        for (int i = 0; i < 8; i++) {
        Mult[i][0] = 'a';
```

```
            Mult[0][i] = 'a';
            Mult[i][1] = alph[i];
            Mult[1][i] = alph[i];
    }
    for (int i = 2; i < 8; i++) {
            for (int j = i; j < 8; j++) {
                    char sd = twoMult(alph[i], alph[j]);
                    Mult[i][j] = sd;
                    Mult[j][i] = sd;

            }
    }
```

2.2.7 Метод twoMult(char a, char b)

Вход: char a (цифра 1) и char b (цифра 2).

Выход: result результат умножения двух символов.

result и buf устанавливаются равными первому аргументу a, вычисляется значение k как searchPoryadok(b) - 1, где searchPoryadok(b) возвращает порядковый индекс символа b в массиве poryadok.

Цикл выполняется, пока $k > 0$. result обновляется результатом сложения текущего result и buf, полученным через вызов функции searchSum.

После завершения цикла result содержит результат умножения символов a и b.

Листинг 6: char Arithmetic::twoMult(char a char b)

```
char result = a;
char buf = a;
int k = searchPoryadok(b) - 1;
while (k > 0) {
        result = searchSum(result, buf);
        k--;
}
return result;
```

2.2.8 Метод fillTableSdvigMult()

Вход: alph, poryadok и незаполненная таблица MultSdvig (таблица сдвига по умножению).

Выход: заполненная таблица MultSdvig.

Два вложенных цикла перебирают все пары индексов i и j (от 0 до 7), что позволяет вычислить значение для каждой ячейки таблицы MultSdvig. Для

текущих символов alph[i] и alph[j] вызывается функция searchPoryadok, чтобы получить их порядковые индексы. Индексы перемножаются, и результат делится на 8 (с использованием целочисленного деления), чтобы определить "сдвиг". tmp используется как индекс в массиве poryadok для выбора соответствующего символа. Этот символ записывается в ячейку MultSdvig[i][j].

Листинг 7: void Arithmetic::fillTableSdvigMult()

```
for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
int tmp =  searchPoryadok(alph[i]) * searchPoryadok(alph[j]);
tmp = tmp / 8;
MultSdvig[i][j] = poryadok[tmp];
        }
}
```

2.2.9 Метод ChooseOperationsMenu(int op)

Вход: int op пользовательский выбор операции.

Выход: выбранный метод.

Функция использует конструкцию switch, чтобы выполнить различные действия в зависимости от значения op:

case 0 — Не выполняет никаких действий, просто выходит из функции (пустое тело).

case 1 — Вызывает функцию sumInput(), которая, вероятно, обрабатывает ввод для операции сложения.

case 2 — Вызывает функцию subInput(), которая обрабатывает ввод для операции вычитания.

case 3 — Вызывает функцию multInput(), которая обрабатывает ввод для операции умножения.

case 4 — Вызывает функцию divInput(), которая обрабатывает ввод для операции деления.

Листинг 8: void Arithmetic::ChooseOperationsMenu(int op)

```
switch (op) {
        case 0:
                break;
        case 1:
                sumInput();
                break;
        case 2:
                subInput();
                break;
        case 3:
```

```
            multInput();
            break;
        case 4:
            divInput();
            break;
        }
```

2.2.10 Методы search

Вход: char a (цифра 1) и char b (цифра 2).

Выход: символ, который является результатом сложения символов a и b по таблице сложения.

Листинг 9: char Arithmetic::searchSum(char a, char b)

```
return Sum[searchAlph(a)][searchAlph(b)];
```

Вход: char a (цифра 1) и char b (цифра 2).

Выход: символ, который является результатом сдвигового сложения.

Листинг 10: char Arithmetic::searchSumSd(char a, char b)

```
return SumSdvig[searchAlph(a)][searchAlph(b)];
```

Вход: char a и char b.

Выход: символ, который является результатом умножения символов a и b по таблице сложения.

Листинг 11: char Arithmetic::searchMult(char a, char b)

```
return Mult[searchAlph(a)][searchAlph(b)];
```

Вход: char a и char b.

Выход: символ, который является результатом сдвигового умножения.

Листинг 12: char Arithmetic::searchMultSd(char a, char b)

```
return MultSdvig[searchAlph(a)][searchAlph(b)];
```

Вход: a — символ, который нужно найти в векторе plusone.

Выход: возвращает индекс символа a в векторе plusone.

Листинг 13: int Arithmetic::searchIndex(char a)

```
for (int i = 0; i < 8; i++)
if (plusone[i] == a) return i;
```

Вход: a — символ, который нужно найти в массиве alph.

Выход: возвращает индекс символа a в массиве alph.

Листинг 14: int Arithmetic::searchAlph(char a)

```
for (int i = 0; i < 8; i++)
if (a == alph[i]) return i;
```

Вход: a — символ, который нужно найти в массиве poryadok.
Выход: возвращает индекс символа a в массиве poryadok.

Листинг 15: int Arithmetic::searchPoryadok(char a)

```
for (int i = 0; i < 8; i++)
if (a == poryadok[i]) return i;
```

Вход: a и b — символы, для которых нужно найти разность.
Выход: возвращает символ, который является результатом вычитания символа b из символа a с использованием таблицы сложения.

Листинг 16: char Arithmetic::searchSub(char a char b)

```
for (int i = 0; i < 8; i++)
if (Sum[i][searchAlph(b)] == a) return alph[i];
```

2.2.11 Методы Input

Вход: строки a и b для пользовательского ввода.
Выход: вывод результата операции сложения в консоль.

Листинг 17: void Arithmetic::sumInput()

```
a = Input1();
b = Input2();
string answer = sum(a, b, 1);
cout << "Ответ: " << a << " + " << b << " = " << answer << \
"\n";
```

Вход: a и b.
Выход: вывод результата операции вычитания в консоль.

Листинг 18: void Arithmetic::subInput()

```
a = Input1();
b = Input2();
string answer = sum(a, b, 0);
cout << "Ответ: " << a << " - " << b << " = " << answer <<\
"\n\n";
```

Вход: a и b.
Выход: вывод результата операции умножения в консоль.

Листинг 19: void Arithmetic::multInput()

```
a = Input1();
```

```
b = Input2();
string answer = mult(a, b);
cout << "Ответ: " << a << " * " << b << " = " << answer << \
"\n\n";
```

Вход: a и b.

Выход: вывод результата операции деления в консоль.

Число строка 1"символизирует о том, что выводить остаток не нужно.

Листинг 20: void Arithmetic::divInput()

```
vector<string> answer;
a = Input1();
b = Input2();
answer = div(a, b, answer);
if (answer[1] == "-1")
{
        cout << "\nОтвет: " << a << " / " << b << " = "\
    << answer[0] << "\n";
}
else
{
        cout << "\nОтвет: " << a << " / " << b << " = "\
    << answer[0] << "\n";
        cout << "        остаток = " << answer[1] << "\n\n";
}
```

2.2.12 Метод sum(string a string b bool fl)

string Arithmetic::sum(string a, string b, bool fl) выполняет операцию сложения или вычитания двух строковых чисел.

Вход: string a (число 1), string b (число 2), bool fl определяющий выполняем мы операцию сложения или вычитания.

Выход: возвращает answer - ответ.

Переменные minuses_a и minuses_b отслеживают количество знаков минус для чисел a и b соответственно. Если первое число начинается с минуса ('-'), то оно удаляется, а переменная счетчик минусов увеличивается на 1. Таким образом, мы получаем абсолютные значения чисел a и b для дальнейшей работы.

```
int minuses_a = 0;
int minuses_b = 0;
if (a[0] == '-') {
    minuses_a++;
    a.erase(0, 1);
```

```
}
if (b[0] == '-') {
    minuses_b++;
    b.erase(0, 1);
}
```

Дальше происходит выравнивание длин чисел, дополняя слева символом 'a' и их сравнение.

```
while (a.size() != b.size())
a.size() < b.size() ? a = 'a' + a : b = 'a' + b;
flag = compare(a, b);
```

Один из операндов отрицательный и флаг fl == 1 (операция сложения), или Операнды одинакового знака и флаг fl == 0 (операция вычитания). Сначала вызывается метод minus(a, b) для вычитания чисел. Если результат выходит за пределы восьмизначного числа, то возвращается строка "overflow". Далее проверяется, должен ли результат быть отрицательным. Например, если операция сложения, а один из операндов отрицательный и flag == -1, то результат будет отрицательным.

```
if ((minuses_a + minuses_b) == 1 && fl == 1 \
(minuses_a + minuses_b) != 1 && fl == 0) {
    answer = minus(a, b);
    if (answer.size() > 8)
        answer = "overflow";
if (answer != "overflow" && ((minuses_b == 1 && flag == -1 \
&& fl == 1)
        (minuses_a == 1 && flag == 1 && fl == 1)
    ((minuses_a + minuses_b) == 0 && flag == -1 && fl == 0)
    ((minuses_a + minuses_b) == 2 && flag == 1 && fl == 0))) {
        answer = '-' + answer;
    }
}
```

Если условие для вычитания не выполняется, выполняется операция сложения с помощью метода plus(a, b). Опять же проверяется, не превышает ли результат восьмизначное число (в этом случае возвращается "overflow"). Затем проверяется, не должен ли результат быть отрицательным в зависимости от знаков операндов и флага операции. В конце происходит возращение результата.

```
else {
    answer = plus(a, b);
    if (answer.size() > 8)
        answer = "overflow";
```

```
    if (answer != "overflow" &&
        ((minuses_a == 1 && flag == 1 && fl == 0)
         (minuses_a == 1 && flag == -1 && fl == 0)
    ((minuses_a + minuses_b) == 2 && flag == -1 && fl == 1)
    ((minuses_a + minuses_b) == 2 && flag == 1 && fl == 1)))
        answer = '-' + answer;
}
return answer;
```

2.2.13 plus(string a, string b)

Метод plus выполняет сложение двух строковых чисел.

Вход: string a, string b.

Выход: answer - ответ.

Сначала a и b выравниваются по длине, затем инициализируются shift1, shift2, shift3 - временные переменные для хранения промежуточных результатов операции сложения и переноса.

Цикл идет по цифрам чисел a и b начиная с младшего разряда (с конца строк). В каждом шаге выполняется несколько операций сложение текущих цифр чисел a и b (используя таблицу сложения), вычисляется временный сдвиг для текущей цифры, вычисляется временный сдвиг с учетом возможного переноса, возвращается итоговое значение сложения с учетом переноса,находится сдвиг для следующего разряда, текущий результат добавляется к итоговому числу (слева). Если это последний разряд (когда i == 0) и есть остаточный сдвиг, то он добавляется к началу результата.

После, все ведущие символы 'a' удаляются. Возвращается итоговый результат в виде строки.

Листинг 21: string Arithmetic::plus(string a string b)

```
char shift1 = 'a', shift2 = 'a', shift3 = 'a', solution;
string answer = "";

while (a.size() != b.size()) a.size() < b.size() ?\
a = 'a' + a : b = 'a' + b;

for (int i = a.size() - 1; i >= 0; i--) {
        solution = searchSum(a[i], b[i]);
        shift2 = searchSumSd(a[i], b[i]);
        shift3 = searchSumSd(solution, shift1);
        solution = searchSum(solution, shift1);
        shift1 = searchSum(shift2, shift3);
        answer = solution + answer;
        shift2 = shift3 = 'a';
```

```
        if (i == 0 && shift1 != 'a') answer = shift1 + answer;
}
while (answer.size() > 1 && answer[0] == 'a')\
answer.erase(0, 1);
return answer;
```

2.2.14 minus(string a, string b)

Вход: string a, string b.

Выход: answer - ответ.

Метод string Arithmetic::minus(string a, string b) выполняет вычитание двух строковых чисел.

Сначала происходит сравнение чисел. Метод сравнивает два числа a и b с помощью функции compare, а результат сравнения сохраняется в переменную flag. Если числа равны (flag == 0), возвращается строка "a которая представляет 0 в этой системе счисления. Если b больше a, то меняем их местами, чтобы всегда вычитать меньшее из большего, и помечаем результат как отрицательный.

```
flag = compare(a, b);
if (flag == 0) {
    return "a";
}
else {
    bool isNegative = false;
    if (flag == -1) {
        a.swap(b);
        isNegative = true;
    }
}
```

После выравнивания строк идет основной цикл вычитания. Основной цикл идет по цифрам чисел с конца (младших разрядов). Если требуется заимствование (первое число меньше второго в текущем разряде), то заимствуем из предыдущего разряда. Если заимствование не нужно, выполняем стандартное вычитание текущих цифр. Результат вычитания добавляется в строку answer, которая формирует итоговый результат.

```
while (b.size() < a.size()) {
    b = 'a' + b;
}
for (int i = a.size() - 1; i >= 0; i--) {
    if (Comparasion(a[i], b[i])) {
        int j = i - 1;
```

```
            while (j >= 0) {
                if (a[j] != 'a') {
                    a[j] = searchSub(a[j], 'b');
                    break;
                }
                else {
                    a[j] = 'z';
                }
                j--;
            }
            solution = searchSub('f', b[i]);
            solution = searchSum('b', solution);
            solution = searchSum(a[i], solution);
        }
        else {
            solution = searchSub(a[i], b[i]);
        }
        answer = solution + answer;
    }
```

Удаляются ведущие нули, добавляется минус при надобности и возвращается результат.

```
while (answer.size() > 1 && answer[0] == 'a') {
    answer.erase(0, 1);
}
if (flag == -1) {
    answer = '-' + answer;
}
return answer;
```

2.2.15 compare(string a, string b)

Вход: string a, string b.

Выход: -1 или 0 (результат сравнения чисел).

Если строки a или b представляют отрицательные числа (начинаются с символа '-'), то этот символ удаляется из строки. Чтобы гарантировать, что обе строки имеют одинаковую длину, добавляются ведущие символы 'a'. Цикл проходит по каждому символу строк, начиная с младших разрядов. Если символ строки a меньше символа строки b, то возвращается -1 (что означает, что a меньше b). Если символ строки b меньше символа строки a, то возвращается 1 (что означает, что a больше b). Если цикл дошел до конца строк и не нашел различий, значит строки равны, и возвращается 0 (что означает, что числа равны).

Листинг 22: int Arithmetic::compare(string a string b)

```
if (a[0] == '-') a.erase(0, 1);
if (b[0] == '-') b.erase(0, 1);

while (a.size() != b.size()) a.size() < b.size() ?\
a = 'a' + a : b = 'a' + b;

for (int i = 0; i < a.size(); i++) {
        if (Comparasion(a[i], b[i])) return -1;
        else if (Comparasion(b[i], a[i])) return 1;
        if (i == a.size() - 1) return 0;
}
```

2.2.16 Comparasion(char a, char b)

Вход: char a, char b.

Выход: true или false (результат сравнения символов).

Внешний цикл проходит по всем индексам массива poryadok для символа a, а внутренний цикл — по всем индексам массива poryadok для символа b. Когда находится пара символов a и b в массиве poryadok, производится сравнение их индексов: Если индекс i (позиция символа a в poryadok) меньше индекса j (позиция символа b в poryadok), то символ a считается "меньше"символа b и возвращается true. Если индекс i больше или равен j, то символ a считается "больше"или равен символу b, и возвращается false.

Листинг 23: bool Arithmetic::Comparasion(char a char b)

```
for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
                if (poryadok[i] == a && poryadok[j] == b)
                        if (i < j)
                                return true;
                        else
                                return false;
        }
}
```

2.2.17 mult(string a, string b)

Вход: string a, string b.

Выход: answer - ответ.

Сначала определяется количество минусов в числах a и b (если они есть). Если минус присутствует, он удаляется из строки для удобства дальнейших вычислений. Затем выравнивается длина.

```
int minuses_a = 0;
int minuses_b = 0;
if (a[0] == '-') {
    minuses_a++;
    a.erase(0, 1);
}
if (b[0] == '-') {
    minuses_b++;
    b.erase(0, 1);
}
while (a.size() != b.size())
a.size() < b.size() ? a = 'a' + a : b = 'a' + b;
```

Затем начинаем умножение цифр чисел a и b с их младших разрядов. Для каждой пары цифр используется метод searchMult для вычисления произведения, а также сдвиги для корректного результата. answer_buf формирует промежуточный результат умножения для каждой пары цифр, а затем прибавляется к текущему результату с использованием метода plus.

```
for (int i = b.size() - 1; i >= 0; i--) {
    for (int j = a.size() - 1; j >= 0; j--) {
        solution = searchMult(a[j], b[i]);
        shift2 = searchMultSd(a[j], b[i]);
        shift3 = searchSumSd(solution, shift1);
        solution = searchSum(solution, shift1);
        shift1 = searchSum(shift2, shift3);
        answer_buf = solution + answer_buf;
        shift3 = 'a';
        shift2 = 'a';
if (j == 0 && shift1 != 'a') answer_buf = shift1 + answer_buf;
    }
    shift1 = 'a';
    int s = k;
    while (s > 0) {
        answer_buf += 'a';
        s--;
    }
    k++;
    answer = plus(answer, answer_buf);
    answer_buf = "";
}
```

В конце удаляются ведущие символы 'a', если длина результата больше 8, то "overflow если результат не является "overflow"и не равен нулю (не начина-

ется с 'a'), то добавляется знак минус, если в исходных числах был нечетный (один) минус.

```
while (answer.size() > 1 && answer[0] == 'a')
    answer.erase(0, 1);
if (answer.size() > 8)
    answer = "overflow";
if (answer[0] != 'a' && answer != "overflow" \
&& (minuses_a + minuses_b) == 1)
    answer = '-' + answer;
```

2.2.18 div(string a, string b, vector<string> result)

Ввод: string a, string b, vector<string> result.

Вывод: result - результат.

Определяется количество минусов в числах a и b. Если числа отрицательные, минус удаляется из строки, а счетчик минусов увеличивается. Также убираем лидирующие 'a', если их больше 1.

```
int minuses_a = 0;
int minuses_b = 0;

if (a[0] == '-') {
    minuses_a++;
    a.erase(0, 1);
}
if (b[0] == '-') {
    minuses_b++;
    b.erase(0, 1);
}

while (a.size() > 1 && a[0] == 'a') a.erase(0, 1);
while (b.size() > 1 && b[0] == 'a') b.erase(0, 1);
```

Если число b равно 'a' и a = 'a', то ответ - весь диапазон чисел. Если только b равно 'a', то деление невозможно. 1"служит флагом и помогает в выводе ответа не выводить остаток. Строки a и b выравниваются по длине, добавляя ведущие символы 'a'.

```
if (compare(b, "a") == 0) {
    if (a == "a") {
                result.push_back("[-ffffffff;ffffffff]");
                result.push_back("-1");
                result.push_back("1");
                return result;
```

```
            }
    result.push_back("пустое множество");
    result.push_back("-1");
    result.push_back("1");
    return result;
        }
```

Ищем коэффициент (цифры от 'b' до 'h'), который будет использоваться для деления. Для каждой цифры подбирается значение, при котором произведение с числом b меньше или равно числу a. Пробуем умножать число b на возможный коэффициент и сравниваем результат с числом a. Как только находим подходящий коэффициент, добавляем его к частному. Если не удается найти подходящий коэффициент, цикл завершается.

```
int flag = 0;
while (true) {
    flag = 0;
    string temp_answer = answer;
    for (char i = 'b'; i <= 'f'; i++) {
        string trial = mult(b, plus(temp_answer, string(1, i)));
        if (trial[0] == '-') trial.erase(0, 1);
        if (compare(trial, a) != 1) { // Если trial <= a
            answer = plus(temp_answer, string(1, i));
            flag = 1;
            break;
        }
    }

    if (flag == 0) break;
}
```

Убираем ведущие нули (символы 'a'), остаток вычисляется как разница между числом a и произведением числа b на найденное частное, убираем ведущие нули.

```
while (answer.size() > 1 && answer[0] == 'a')\
answer.erase(0, 1);
remainder = minus(a, mult(b, answer));
while (remainder.size() > 1 && remainder[0] == 'a')\
remainder.erase(0, 1);
```

Если первое из чисел (входных) отрицательное и остаток деления без учета знака не равен 'a', то пересчитываем целую часть и остаток. Увеличиваем частное на "b"и пересчитываем остаток. Затем добавляем знак минус к частному. Результат деления (частное и остаток) добавляется в вектор result.

Если первое или второе число отрицательное, приписываем знак '-' к целой части.

```
if (minuses_a == 1 && minuses_b == 0) {
        if (remainder != "a") {
                answer = plus(answer, "b");
                string old_remainder = remainder;
                remainder = minus(b, old_remainder);
        }
        answer = '-' + answer;
}

if (minuses_a == 0 && minuses_b == 1) {
        answer = '-' + answer;
}

// Записываем результат
result.push_back(answer);
result.push_back(remainder);
result.push_back("0");
return result;
```

2.3 Функции

2.3.1 void Start()

Выводит приветственное окно.
Вход: пустой.
Выход: вывод текста.

Листинг 24: void Start()

```
cout << endl << "*******************************\
*********************************"<<endl
            <<    "* ДОБРО ПОЖАЛОВАТЬ В КАЛЬКУЛЯТОР\
            БОЛЬШОЙ КОНЕЧНОЙ АРИФМЕТИКИ *" << endl <<
                "*******************************\
                **********************************"\
                << endl << endl  <<
    "Вариант 33:  a->b->e->h->c->d->g->f" << endl <<
    "               a - нейтральный элемент по сложению"\
    << endl <<
    "               b - нейтральный элемент по умножению";
```

2.3.2 string Input1()

Функция ввода первого числа и обработки корректности ввода. Если первый знак - это минус то допустимы строки меньше или равные размером 9, иначе меньше или равные 8. Кроме того принимаются слова состоящие только из букв алфавита и знака минус (только первым символом).

Вход: input - ввод первого числа.

Выход: вывод текста, input.

Листинг 25: string Input1()

```
    cout << endl << "************************" << endl
      << "* ВВЕДИТЕ ПЕРВОЕ ЧИСЛО *" << endl
      << "************************" << endl << endl;

 string input;
 while (true) {
      cin >> input;
      if ((input.size() > 8 && input[0] != '-') \
      (input.size() > 9 && input[0] == '-')) {
          cout << "Ошибка ввода! Число должно состоять
          из 8 разрядов и меньше! "<<endl;
          continue;
      }
      bool valid = true;
      for (size_t i = 0; i < input.size(); ++i) {
          char c = input[i];
          if (string("abehcdgf").find(c) == string::npos) {
              if (c == '-' && i == 0 && input.size() > 1)
              continue;
              valid = false;
              break;
          }
      }
      if (!valid) {
          cout << "Ошибка ввода! Допустимы только символы\
          -, a, b, e, h, c, d, g, f и - может быть только первым!
          Повторите ввод! " << endl;
          continue;
      }
      break;
 }

 return input;
```

2.3.3 string Input2()

Функция ввода второго числа.
Вход: input - ввод второго числа.
Выход: вывод текста, input.

Листинг 26: string Input2()

```
 cout << endl << "************************" << endl
    << "* ВВЕДИТЕ ВТОРОЕ ЧИСЛО *" << endl
    << "************************" << endl << endl;

string input;
while (true) {
    cin >> input;
    if ((input.size() > 8 && input[0] != '-') \
    (input.size() > 9 && input[0] == '-')) {
        cout << "Ошибка ввода! Число должно состоять
        из 8 разрядов и меньше! " << endl;
        continue;
    }
    bool valid = true;
    for (size_t i = 0; i < input.size(); ++i) {
        char c = input[i];
        if (string("abehcdgf").find(c) == string::npos) {
            if (c == '-' && i == 0 && input.size() > 1)
            continue;
            valid = false;
            break;
        }
    }
    if (!valid) {
        cout << "Ошибка ввода! Допустимы только символы\
        -, a, b, e, h, c, d, g, f и - может быть только первым!
        Повторите ввод! " << endl;
        continue;
    }
    break;
}

return input;
```

2.3.4 void Menu()

Выводит заголовок меню.
Вход: пустой.
Выход: вывод текста.

Листинг 27: void Menu()

```
 cout << endl << endl << endl << "**************************"
<< endl
     << "* МЕНЮ ДОСТУПНЫХ ОПЕРАЦИЙ *" << endl <<
        "**************************";
```

2.3.5 int OperationsMenu()

Выводит меню. Инициализирует 2 переменные int op - для ввода номера операции, bool validInput = false - для обработки некорректного ввода (меньше 0 больше 4). Возвращает op и передает в метод класса.
Вход: op - выбор номера действия.
Выход: вывод текста, op.

Листинг 28: int OperationsMenu()

```
  cout << endl << endl<< "* МЕНЮ *" << endl << endl;
cout << "1) Сложение" << endl
    << "2) Вычитание" << endl
    << "3) Умножение" << endl
    << "4) Деление" << endl
    << "                          ДЛЯ ВЫХОДА ИЗ ПРИЛОЖЕНИЯ
    ВВЕДИТЕ 0"<<endl<<endl;
cout << "Введите номер нужной операции
(от 1 до 4 включительно)" << endl << endl;

int op;
bool validInput = false;
while (!validInput) {
    cout << "Ввод: ";
    cin >> op;
    if (cin.fail()  op < 0  op > 4) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Ошибка ввода! Введите число
        от 0 до 4 включительно!" << endl;
    }
    else return op;
```

```
}
```

2.3.6 void Bye()

Выводит сообщение "До скорых встреч!"
Вход: пустой.
Выход: вывод текста.

Листинг 29: void Bye()

```
cout << endl << "********************" << endl
    << "* ДО НОВЫХ ВСТРЕЧ! *" << endl <<
    "********************" << endl << endl;
```

2.4 Порядок вызова в Main.cpp

Для начала выводим приветственное окно, затем создается объект класса Arithmetic с именем a1. Этот объект будет использоваться для выполнения различных арифметических операций. Вызывается функция Menu(), которая выводит основное меню. Запускаем цикл while до тех пор, пока пользователь не выберет опцию для завершения работы программы. Каждый раз в цикле меню отображается снова, предоставляя пользователю возможность выбрать операцию. Вызов функции OperationsMenu(), которая, вероятно, выводит меню с операциями, которые пользователь может выполнить и возвращает выбранную операцию в переменную operation. Если пользователь выбрал операцию (не 0), вызывается метод ChooseOperationsMenu объекта a1 (класс Arithmetic), который выполняет соответствующую операцию. При выборе нуля - выводим окно с прощанием.

Листинг 30: int Continue()

```
setlocale(LC_ALL, "rus");
Start();
Arithmetic a1;
Menu();
while (true) {
    Menu(); // Вывод меню
    int operation = OperationsMenu();

    if (operation == 0) {
        Bye();
        break;
    }

    a1.ChooseOperationsMenu(operation);
}
```

3 Результаты работы программы

При запуске приложения высвечивается приветственное окно и меню. Пользователю сразу дается возможность выбрать номер операции (см. [Рис. 6]).

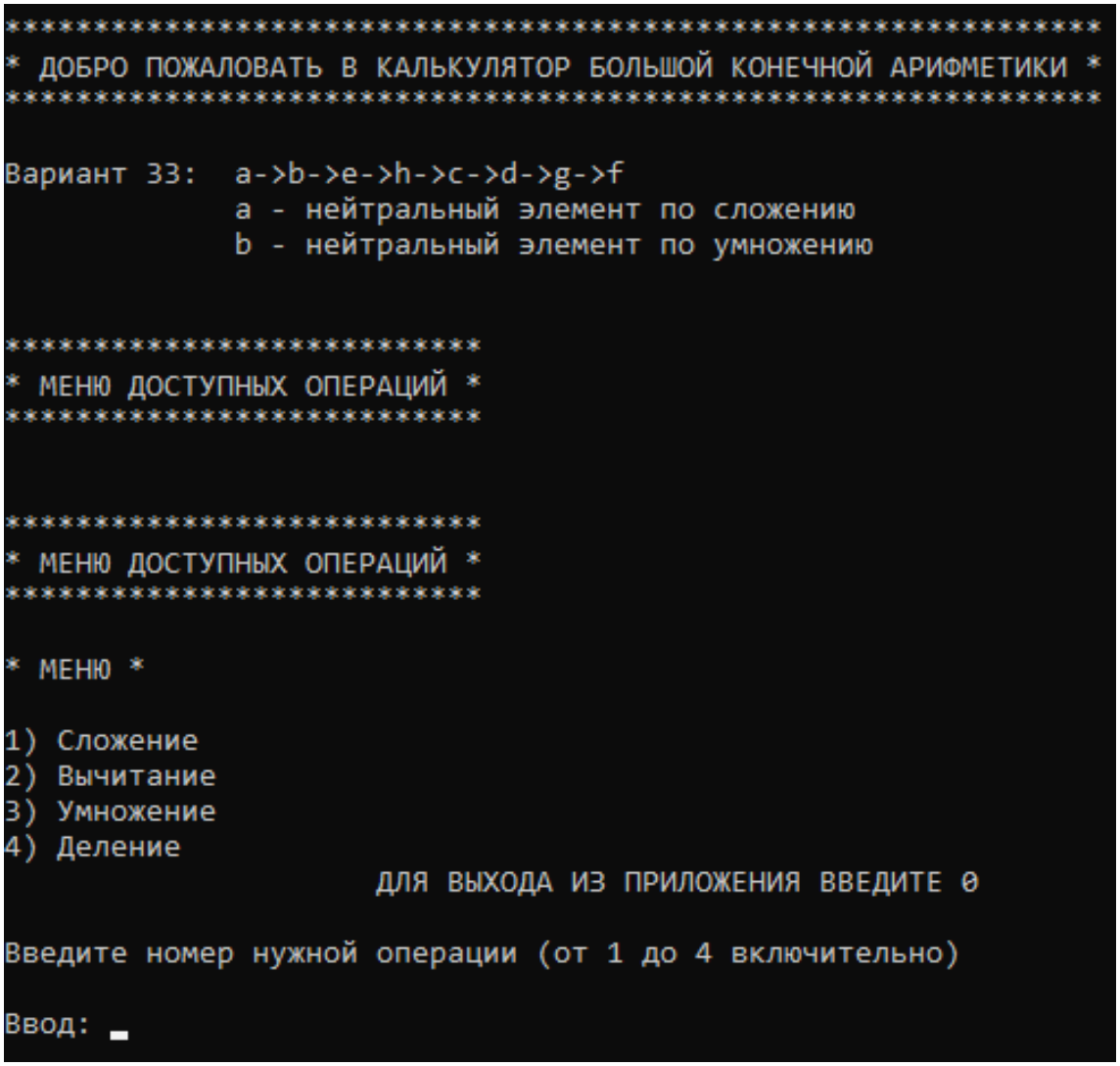

Рис. 6

После корретного ввода доступен ввод первого числа (см. [Рис. 7]).

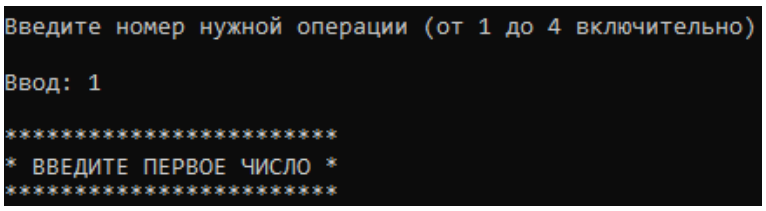

Рис. 7

При некорректном вводе - ошибка (см. [Рис. 8]).

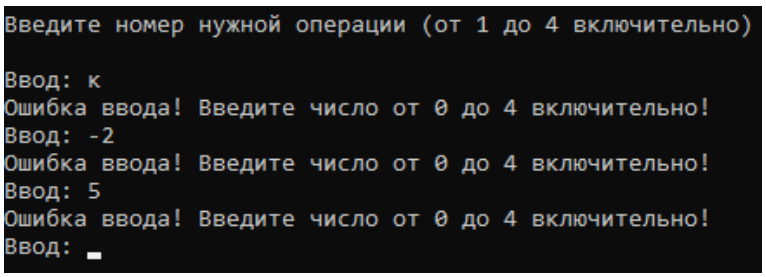

Рис. 8

После ввода обоих чисел - результат операции (см. [Рис. 9]).

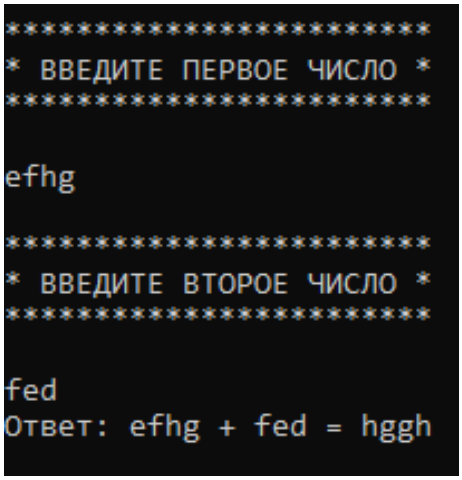

Рис. 9

При некорректном вводе чисел - ошибка (см. [Рис. 10]).

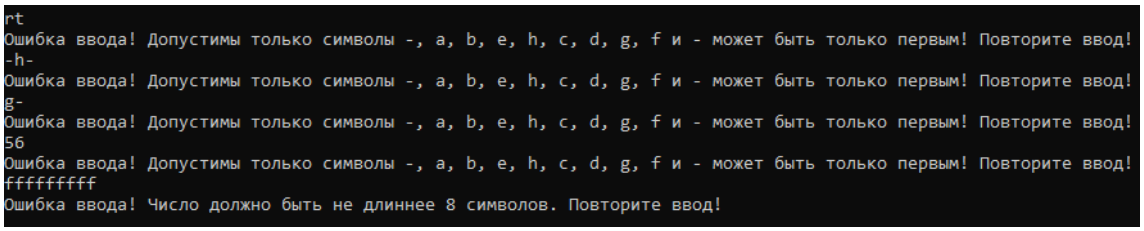

Рис. 10

Когда результат состоит более, чем из 8 разрядов, то ответ - это переполнение (см. [Рис. 11]).

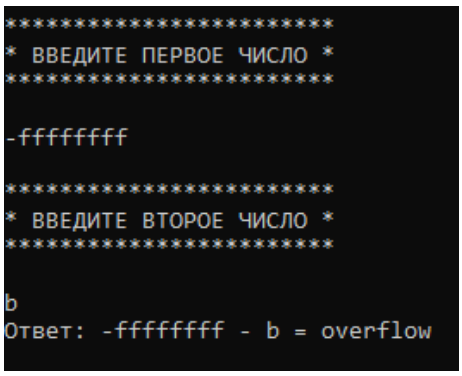

Рис. 11

При умножении на ’a’ ответ ’a’ (см. [Рис. 12]).

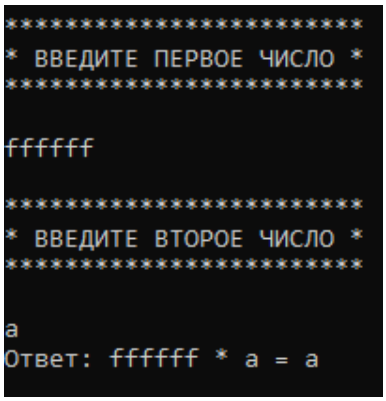

Рис. 12

При делении на ’a’ ответ - пустое множество (см. [Рис. 13]).

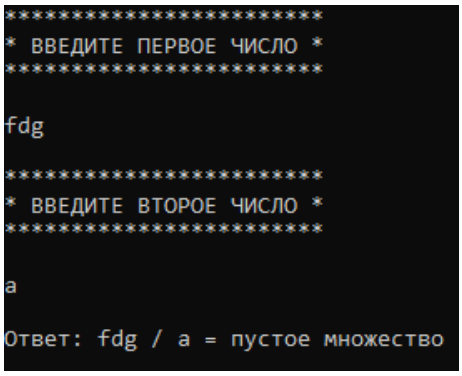

Рис. 13

При делении ’a’ на ’a’ ответ - весь диапазон [-fffffff;fffffff] (см. [Рис. 14]).

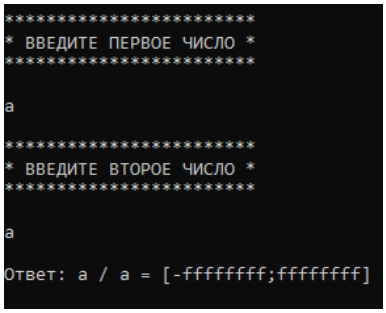

Рис. 14

При делении, когда первое число отрицательное (см. [Рис. 15]).

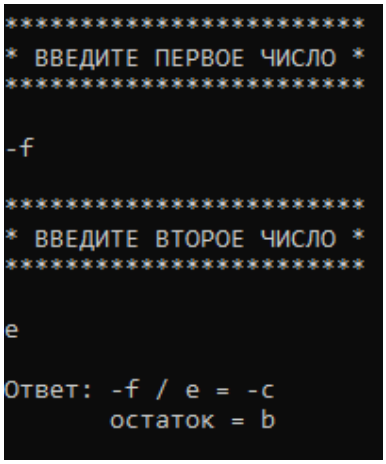

Рис. 15

При делении, когда второе число отрицательное (см. [Рис. 16]).

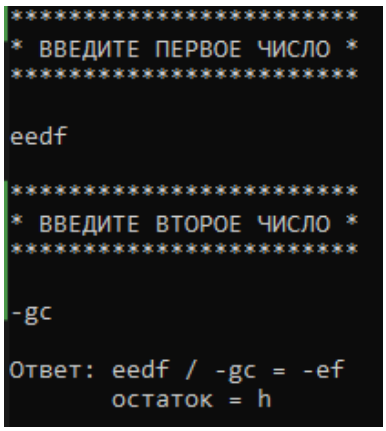

Рис. 16

При вводе нуля - выход из приложения (см. [Рис. 17]).

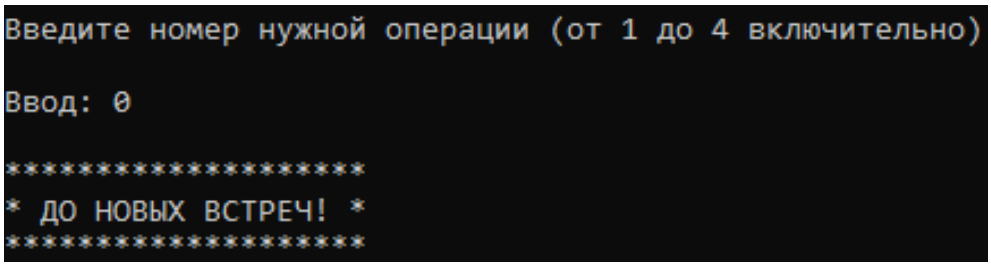

Рис. 17

Заключение

В ходе выполнения работы был реализован калькулятор для "большой"конечной арифметики <Zi;+,*> с 8-разрядными числами с такими действиями как сложение, вычитание, умножение и деление, основанных на правилах «малой» конечной арифметики.

Присутствует огранчение на вводимые числа - они должны содержать меньше 8 разрядов. При получении в ходе действия ответа более 8 разрядов, выводится сообщение о переполнении.

Недостатком данной программы кроме вышеприведенных ограничений является сложность реализации методов mult и div, данные блоки кода можно было больше разбить на дополнительные методы, чтобы улучшить читаемость. Некоторые функции, например, minus и div, можно было бы оптимизировать для повышения производительности.

Преимуществом программы является то, что работа идет сразу с буквами по программно заполненным таблицам, реализована работа с отрицательными числами. Для работы с действиями сложения и вычитания используется общий метод, что улучшает читаемость кода и отражает суть действия вычитания. Так же плюсом является разбиение части кода, где мы работаем с таблицами, на большое количество простых методов, что упрощает его понимание и модификацию.

Дополнительно может быть реализовано больше других действий например возведение в степень, НОК, НОД. Так же было бы хорошей доработкой добавление ввода, который позволяет вернуться из режима ввода чисел к меню.

Может быть добавлен пользовательский ввод отношения порядка и разрядности чисел.

Список литературы

[1] Алгебраические структуры. Ф. А. Новиков. *Дискретная математика для программистов.*. 2009.– 3-е издание. – Питер : Питер Пресс, 2009.

[2] Основные понятия арифметики.

// URL: https://studfile.net/preview/1399243/page:15/

(дата обращения: 02.12.2024)

[3] Определение кольца и поля.

// URL: http://ru.discrete-mathematics.org/fall2014/3/fields_theory/seminar_1_ring.pdf

(дата обращения: 01.12.2024)