

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт Компьютерных наук и кибербезопасности

Направление 02.03.01 Математика и компьютерные науки

Высшая школа технологий искусственного интеллекта

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ ПО ДИСКРЕТНОЙ МАТЕМАТИКЕ:

**Калькулятор «большой» конечной
арифметики
Вариант 18**

Обучающийся: _____

Санько В. В.

Руководитель: _____

Востров А. В.

«_____» _____ 20__ г.

Санкт-Петербург, 2024

Содержание

Введение	3
1 Математическое описание	4
1.1 Описание алгебраических структур	4
1.2 Свойства операций	4
1.3 Таблицы операций малой конечной арифметики	5
1.4 Примеры вычисления выражений	6
2 Особенности реализации	7
2.1 class Calculator	7
2.2 Функция int main()	8
2.3 Функция выбора операции (evaluate)	10
2.4 Поиск индекса буквы в алфавите (findIndex)	11
2.5 Получение буквы по индексу (getLetter)	11
2.6 Сравнение двух строк (compareStrings)	11
2.7 Сложение двух букв (addLetters)	12
2.8 Сложение двух строк (addStrings)	13
2.9 Вычитание двух букв (subtractLetters)	14
2.10 Вычитание двух строк (subtractStrings)	15
2.11 Умножение строки и буквы (multiplyStrings)	17
2.12 Умножение двух строк (multiplyStrings)	17
2.13 Деление строк со знаком(divideWithSings)	19
2.14 Деление строк (divideStrings)	20
2.15 Проверка строки на валидность (validateString)	22
2.16 Определение операции в зависимости от знака(calculateWithSigns)	23
3 Результаты работы	25
Заключение	27
Список литературы	28

Введение

Задачей проекта является создание калькулятора для “большой” конечной арифметики в системе $(Z_8; +, *)$ с 8-разрядными числами, который будет использовать принципы “малой” конечной арифметики. Калькулятор должен обеспечивать выполнение операций сложения, вычитания, умножения и деления. Правило «+1» представлено в таблице 1.

Z_8	a	b	c	d	e	f	g	h
+1	b	d	e	h	g	a	f	c

Таблица 1: правило «+1»

1 Математическое описание

1.1 Описание алгебраических структур

Малая конечная арифметика — конечное коммутативное кольцо с единицей $\langle M; +, \cdot \rangle$, на котором определены операции сложения и умножения, а также определены вычитание и деление.

В данной работе $M = \{a, b, c, d, e, f, g, h\}$ — носитель малой конечной арифметики $\langle M; +, \cdot \rangle$, а операции $+$, \cdot порождены правилом «+1» (таблица 1).

Пусть имеется малая конечная арифметика $\langle M; +, \cdot \rangle$. Большая конечная арифметика — конечное коммутативное кольцо с единицей $\langle M^n; +^n, \cdot^n \rangle$, на котором также определены действия вычитание и деление с остатком.

В этой работе $n = 8$ и большая конечная арифметика принимает вид $\langle M^8; +^8, \cdot^8 \rangle$. Операции $+$ и \cdot определены посредством операций малой конечной арифметики $+$, \cdot и операций переноса $+_c, \cdot_c : M \times M \rightarrow M$. Элементы большой конечной арифметики обозначены $(a_7, a_6, \dots, a_2, a_1, a_0)$, где $a_i \in M$.

1.2 Свойства операций

Из определения малой конечной арифметики следует:

1. Ассоциативность $+$ и \cdot :

$$\forall x, y, z \in M : x + (y + z) = (x + y) + z; \quad x(yz) = (xy)z$$

2. Коммутативность $+$ и \cdot :

$$\forall x, y \in M : x + y = y + x; \quad xy = yx$$

3. Дистрибутивность $+$ относительно \cdot :

$$\forall x, y, z \in M : x(y + z) = xy + xz$$

4. Существование нейтрального элемента e по $+$:

$$\exists e \in M : \forall x \in M : x + e = e + x = x$$

5. Существование нейтрального элемента a по \cdot :

$$\exists a \in M : \forall x \in M : x \cdot a = a \cdot x = x$$

Как следствие:

$$\forall x \in M : x \cdot a = a \cdot x = a$$

1.3 Таблицы операций малой конечной арифметики

На рисунке 1 представлены таблицы операций, порожденные правилом «+1», с заданными нейтральными элементами по $+$ и \cdot .

(a) По сложению									(b) По умножению								
$+$	a	b	c	d	e	f	g	h	\cdot	a	b	c	d	e	f	g	h
a	a	b	c	d	e	f	g	h	a	a	a	a	a	a	a	a	a
b	b	d	e	h	g	a	f	c	b	a	b	c	d	e	f	g	h
c	c	e	a	g	b	h	d	f	c	a	c	a	a	c	c	a	c
d	d	h	g	c	f	b	a	e	d	a	d	a	c	d	g	c	g
e	e	g	b	f	d	c	h	a	e	a	e	c	d	a	g	b	d
f	f	a	h	b	c	g	e	d	f	a	f	c	g	g	b	d	e
g	g	f	d	a	h	e	c	b	g	a	g	a	c	b	d	c	d
h	h	c	f	e	a	d	b	g	h	a	h	c	g	d	e	d	b

(c) Переносы по сложению									(d) Переносы по умножению								
$+_c$	a	b	c	d	e	f	g	h	\cdot_c	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
b	a	a	a	a	a	b	a	a	b	a	a	a	a	a	a	a	a
c	a	a	b	a	b	b	b	a	c	a	a	d	b	d	h	h	b
d	a	a	a	a	a	b	b	a	d	a	a	b	a	b	b	b	a
e	a	a	b	a	b	b	b	b	e	a	a	b	b	h	c	h	b
f	a	b	b	b	b	b	b	b	f	a	a	b	b	c	g	e	d
g	a	a	b	b	b	b	b	b	g	a	a	b	b	h	e	c	d
h	a	a	a	a	b	b	b	a	h	a	a	b	a	b	d	d	b

Таблица 1: Таблицы операций в малой конечной арифметике.

Примечание: В таблицах использованы следующие правила:

- Символ $+$ соответствует сложению;
- Символ \cdot соответствует умножению;
- Символы $+_c$ и \cdot_c представляют переносы при сложении и умножении.

1.4 Примеры вычисления выражений

Введём отношение строгого линейного порядка в малой конечной арифметике по правилу $+1$:

$$a \prec a + b \prec a + b + b \prec \dots \prec a + \underbrace{b + \dots + b}_{n-1}$$

В данной работе $n = 8$, и отношение строгого линейного порядка принимает вид:

$$a \prec b \prec d \prec h \prec c \prec e \prec g \prec f.$$

Примеры вычисления выражений в малой конечной арифметике:

1. $d + h = d + (d + b) = d + ((b + b) + b) = (((d + b) + b) + b) = (h + b) + b = c + b = e$
2. $f + d = f + (b + b) = (f + b) + b = a + b = b$
3. $d \cdot c = c(b + b) = c + c = a$
4. $h \cdot h = h(d + b) = h \cdot d + h = h(b + b) + h = (h + h) + h = g + h = b$

2 Особенности реализации

2.1 class Calculator

Класс содержащий все внешние и вложенные функции программы, а также все константы. Первая константа *alphabet* = "abdhcegf" отвечает за правильную индексацию символов, порядок букв соответствует порядку чисел от 0 до 7 включительно.

BASE число всех букв.

max_value = "ffffffff" соответствует границе перехода в замкнутой системе.

metka = *false* флаг помогающий определить знак строки.

```
class Calculator {
public:
    string evaluate(const string str1, const string str2, const char operation);
    string calculateWithSigns(string str1, string str2, char operation);
    string divideWithSigns(string str1, string str2);
    string divideStrings(const string& dividend, const string& divisor);
    string multiplyStrings(const string& str1, const string& str2);
    string addStrings(const string& str1, const string& str2);
    string subtractStrings(const string& str1, const string& str2);

private:
    char addLetters(char letter1, char letter2, int& carry);
    char NsubtractLetters(char letter1, char letter2, int& borrow, bool& flag);
    char subtractLetters(char letter1, char letter2, int& borrow);
    bool Compare(const string& str1, const string& str2);
    char getLetter(int index);
    int findIndex(char letter);
    string multiplyStrings(string str, char letter);

    const string alphabet = "abdhcegf";
    const int BASE = alphabet.size();
    const string max_value = "ffffffff";
    bool metka = false;
};
```

2.2 Функция `int main()`

Функция отвечает за меню пользователя. Калькулятор осуществляется за счет цикла `do while`. Пользователь вводит 2 строки, которые проходят проверку, затем выводится меню операций и пользователь вводит номер операции. При помощи конструкции `switch case`, номер операции определяет каким символом будет заполнена переменная `operation`. После инициализации вызывается функция `calc.evaluate()` и выводится результат. После нажатия любой кнопки консоль очищается и цикл повторяется.

На вход: строки S_1 , S_2 и символ *operation*.

На выход: сигнал закончена программа успешно или нет.

```
int main() {
    setlocale(LC_ALL, "rus");

    Calculator calc;
    string str1, str2;
    char operation;
    int choice;

    do {
        cout << "Калькулятор <большой> конечной арифметики" << endl;
        cout << "Введите первую строку: ";
        while (true) {
            cin >> str1;
            if (isValidInput(str1)) {
                break;
            }
            else {
                cout << "Некорректный ввод. Пожалуйста, введите строку, состоящую
                только из букв a-g, длиной до 8 символов (или до 9, если начинается с
                '-' ). Попробуйте снова: ";
            }
        }

        cout << "Введите вторую строку: ";
        while (true) {
            cin >> str2;
```



```

        if (isValidInput(str2)) {
            break;
        }
        else {
            cout << "Некорректный ввод. Пожалуйста, введите строку, состоящую
            только из букв a-g, длиной до 8 символов (или до 9, если
            начинается с '-'). Попробуйте снова: ";
        }
    }

    cout << "\nВыберите операцию:\n";
    cout << "1. Сложение \n";
    cout << "2. Вычитание \n";
    cout << "3. Умножение \n";
    cout << "4. Деление \n";
    cout << "0. Выход\n";
    cout << "Ваш выбор: ";
    cin >> choice;

    cin.ignore();

    if (choice == 0) {
        cout << "Выход из программы...\n";
        break;
    }

    switch (choice) {
    case 1:
        operation = '+';
        break;
    case 2:
        operation = '-';
        break;
    case 3:
        operation = '*';
        break;

```

```

        case 4:
            operation = '/';
            break;
        default:
            cout << "Некорректный выбор!" << endl;
            continue;
    }

    cout << calc.evaluate(str1, str2, operation) << endl;

    cout << "Нажмите Enter для продолжения...";
    cin.get();

    system("cls");

} while (choice != 0);

return 0;
}

```

2.3 Функция выбора операции (evaluate)

Функция в зависимости от выбора пользователя вызывает соответствующую операцию.

На вход: строки S_1 , S_2 и символ *operation*.

На выход: строка ответа S .

```

string Calculator::evaluate(const string str1, const string str2, const char operation) {
    string result;
    if (operation == '+' || operation == '-') {
        result = calculateWithSigns(str1, str2, operation);
    }
    else if (operation == '*') {
        result = multiplyStrings(str1, str2);
    }
    else if (operation == '/') {
        result = divideWithSigns(str1, str2);
    }
    return result;
}

```

```
}
```

2.4 Поиск индекса буквы в алфавите (findIndex)

Функция ищет индекс буквы x в алфавите A . Она проходит по массиву символов и возвращает позицию, если буква найдена. Если буква отсутствует, возвращается -1 , что указывает на ошибку. Реализация использует линейный поиск, что делает функцию эффективной для небольших алфавитов, но менее производительной для больших.

На вход: буква x .

На выход: индекс i , где $i \in \{0, 1, \dots, n - 1\}$, либо -1 .

```
int Calculator::findIndex(char letter) {  
    return alphabet.find(letter);  
}
```

2.5 Получение буквы по индексу (getLetter)

Функция возвращает букву из алфавита A по её индексу i . Индекс вычисляется по модулю размера алфавита n , что позволяет выполнять циклический доступ. Если i превышает длину алфавита, используется остаток от деления на n .

На вход: целое число i .

На выход: буква $A[i \bmod n]$.

```
char Calculator::getLetter(int index) {  
    return alphabet[index % BASE];  
}
```

2.6 Сравнение двух строк (compareStrings)

Функция сравнивает две строки S_1 и S_2 , определяя, меньше ли первая строка по значению, чем вторая. Если строки равны по длине, производится побуквенное сравнение. Используется удаление ведущих нулевых символов (нулевой эквивалент — a), что упрощает обработку значений.

На вход: строки S_1 и S_2 .

На выход: булево значение: `true`, если $S_1 < S_2$, иначе `false`.

```
bool Calculator::Compare(const string& str1, const string& str2) {  
    string s1 = str1;  
    string s2 = str2;
```

```

s1.erase(0, s1.find_first_not_of('a'));
s2.erase(0, s2.find_first_not_of('a'));

if (s1.size() != s2.size()) {
    return s1.size() < s2.size();
}

for (size_t i = 0; i < s1.size(); ++i) {
    if (findIndex(s1[i]) != findIndex(s2[i])) {
        return findIndex(s1[i]) < findIndex(s2[i]);
    }
}

return false;
}

```

2.7 Сложение двух букв (addLetters)

Функция выполняет сложение двух букв x_1 и x_2 с учетом флага переноса `carry`. Первый элемент алфавита перемещается из начала в конец, перебирая строку таким образом. Количество перестановок соответствует индексу первой буквы. Результатом сложения будет буква находящаяся под индексом второй буквы в новой строке. Результат — новая буква x_r и обновленное значение переноса. Перенос `carry` обновляется только в случае, если полученная буква в первоначальной строке меньше минимальной из двух букв n .

На вход: две буквы x_1 , x_2 и флаг переноса `carry`.

На выход: буква x_r и обновленный флаг переноса `carry`.

```

char addLetters(char letter1, char letter2, int& carry) {
    size_t pos1 = alphabet.find(letter1);
    size_t pos2 = alphabet.find(letter2);

    string result = alphabet;

    if (carry > 0) {
        pos2++;
        carry = 0;
    }
}

```

```

}

for (size_t i = 0; i < pos2; ++i) {
    result = result.substr(1) + result[0];
}

char m = minimum(letter1, letter2);

if (minimum(result[pos1], m) == result[pos1]) {
    carry++;
}

return result[pos1];
}

```

2.8 Сложение двух строк (addStrings)

Функция дополняет строки символами a до одной длины, а затем складывает две строки S_1 и S_2 . Для этого строки выравниваются по длине, а затем выполняется посимвольное сложение с младших разрядов. Если по завершении сложения $\text{carry} = 1$, к результату добавляется ведущий символ b (единица). После символы a в начале строки удаляются и идет проверка на переполнение. Если длина строки больше максимального допустимого, первый символ удаляется и вызывается функция `subtractStrings()` для границы счисления и получившегося результата. В итоге если было переполнение, ответ дополняется минусом.

На вход: строки S_1 и S_2 .

На выход: строка R , представляющая сумму $S_1 + S_2$.

```

string Calculator::addStrings(const string& str1, const string& str2) {
    string result = "";
    int carry = 0;

    int len1 = str1.size(), len2 = str2.size();
    string s1 = string(max(len1, len2) - len1, 'a') + str1;
    string s2 = string(max(len1, len2) - len2, 'a') + str2;

    for (int i = s1.size() - 1; i >= 0; --i) {
        result += addLetters(s1[i], s2[i], carry);
    }
}

```

```

}

if (carry > 0) {
    result += getLetter(carry);
}

reverse(result.begin(), result.end());
result.erase(0, result.find_first_not_of('a'));
if (result.empty()) result = "a";

if (result.size() > BASE) {
    result = result.substr(1);

    result = subtractStrings(max_value, result);
    metka = true;
}

if (metka) {
    result = "-" + result;
}

metka = false;
return result;
}

```

2.9 Вычитание двух букв (subtractLetters)

Функция выполняет вычитание двух букв x_1 и x_2 с учетом флага заимствования `borrow`. Находится меньшая из двух букв, для того чтобы сделать заимствование разряда из большей, отдельно рассмотрен случай двух 'a'. Логика схожа со сложением однако, перебор строки происходит в обратную сторону. Если результат вычитания меньше нуля, флаг `borrow` обновляется до 1, и к результату прибавляется размер алфавита n .

На вход: две буквы x_1 , x_2 и флаг заимствования `borrow`.

На выход: буква x_r и обновленный флаг заимствования `borrow`.

```

char subtractLetters(char letter1, char letter2, int& borrow) {
    size_t pos1 = alphabet.find(letter1);
    size_t pos2 = alphabet.find(letter2);

```

```

string result = alphabet;
char m = minimum(letter1, letter2);

if (borrow > 0 && m == letter1 && letter1 == 'a') {
    pos1 = pos1 + BASE - 1;
}
else if (borrow > 0 && m == letter1) {
    pos2--;
    borrow = 0;
}
else if (borrow > 0 && m == letter2) {
    pos1--;
    borrow = 0;
}

if (m == letter1 && pos1 != pos2) borrow++;

for (size_t i = 0; i < pos2; ++i) {
    result = result.substr(BASE - 1) + result.substr(0, BASE - 1);
}

return result[pos1];
}

```

2.10 Вычитание двух строк (subtractStrings)

Функция дополняет строки символами *a* до одной длины, а затем сравнивает две строки для того чтобы определиться с тем какая строка будет уменьшаемым, а какая вычитаемым и определить подходящую логику. Вычитание строк S_1 и S_2 , использует посимвольное вычитание с младших разрядов. Если результат отрицательный, используется флаг заимствования **borrow**. После символы *a* в начале строки удаляются и идет проверка на переполнение. Если длина строки больше максимального допустимого, первый символ удаляется и вызывается функция `subtractStrings()` для границы счисления и получившегося результата. В итоге если было переполнение, ответ дополняется минусом.

На вход: строки S_1 и S_2 .

На выход: строка R , представляющая разность $S_1 - S_2$.

```
string Calculator::subtractStrings(const string& str1, const string& str2) {
    string larger = str1;
    string smaller = str2;
    bool resultIsNegative = false;

    string result = "";
    int borrow = 0;

    int len1 = larger.size(), len2 = smaller.size();
    string s1 = string(max(len1, len2) - len1, 'a') + larger;
    string s2 = string(max(len1, len2) - len2, 'a') + smaller;

    bool com = Compare(s1, s2);

    if (com == false && !metka) {
        for (int i = s1.size() - 1; i >= 0; --i) {
            result += subtractLetters(s1[i], s2[i], borrow);
        }
    }
    else {
        for (int i = s1.size() - 1; i >= 0; --i) {
            result += NsubtractLetters(s1[i], s2[i], borrow, resultIsNegative);
        }
    }

    reverse(result.begin(), result.end());
    result.erase(0, result.find_first_not_of('a'));

    if (result.empty()) result = "a";

    if (result.size() > BASE) {
        result = result.substr(1);
        result = subtractStrings(result, max_value);
        metka = true;
    }
}
```



```

}

if (metka || resultIsNegative) {
    result = "-" + result;
}

metka = false;
return result;
}

```

2.11 Умножение строки и буквы (multiplyStrings)

Умножение реализовано при помощи сложения строки n раз, где n — соответствует индексу буквы.

На вход: строка str и буква $letter2$.

На выход: строка R , представляющая произведение $str \cdot letter$.

```

string multiplyStrings(string str, char letter2) {
    string result = "a";

    for (size_t i = 0; i < alphabet.find(letter2); ++i) {
        result = addStrings(result, str);
    }

    return result;
}

```

2.12 Умножение двух строк (multiplyStrings)

Первым делом идет проверка на знак. Функция перемножает строки S_1 и S_2 . Затем проверка на нулевой элемент. Реализация умножения происходит путем умножения строки S_2 на каждую букву строки S_1 , затем происходит сложение текущего результата и суммы. Перед каждой итерацией осуществляется смещение на разряд, путем добавления 'а' в конце результирующей строки. В отличии от других функций результирующая строку не приходится разворачивать так как умножение реализуется от старшего разряда к младшему. При переполнении будет происходить вычитание максимальной границы до тех пор, пока результат не будет входит в диапазон значений.

На вход: строки S_1 и S_2 .

На выход: строка R , представляющая произведение $S_1 \cdot S_2$.

```
string multiplyStrings(const string& str1, const string& str2) {

    string sa1 = str1, sa2 = str2;
    if (sa1[0] == '-') {
        sa1 = sa1.substr(1);
    }
    if (sa2[0] == '-') {
        sa2 = sa2.substr(1);
    }
    sa1.erase(0, sa1.find_first_not_of('a'));
    sa2.erase(0, sa2.find_first_not_of('a'));
    if (sa1.empty()) sa1 = "a";
    if (sa2.empty()) sa2 = "a";

    if (sa1 == "a" || sa2 == "a") return "a";

    string result = "";
    bool resultIsNegative = false;

    string s1 = str1, s2 = str2;

    if (s1[0] == '-') {
        resultIsNegative = !resultIsNegative;
        s1 = s1.substr(1);
    }
    if (s2[0] == '-') {
        resultIsNegative = !resultIsNegative;
        s2 = s2.substr(1);
    }

    int carry = 0;
    string temp = "a";

    for (size_t i = 0; i < s1.size(); ++i) {
```

```

        result += "a";
        temp = multiplyStrings(s2, s1[i]);
        result = addStrings(result, temp);
    }

    result.erase(0, result.find_first_not_of('a'));

    if (result.empty()) result = "a";

    if (result.size() > BASE) {
        result = result.substr(1);
        result = subtractStrings(result, max_value);
        resultIsNegative = true;
    }

    result.erase(0, result.find_first_not_of('-'));
    if (resultIsNegative) {
        result = "-" + result;
    }
    if (result == "-a") {
        result = result.substr(1);
    }

    return result;
}

```

2.13 Деление строк со знаком(divideWithSings)

Функция осуществляет проверку на знак. Затем вызывается функция divideStrings() для подсчета частного и функция вычитания изначально делимой строки и умножения частного на делитель. Рассматривается случай отрицательного делимого для того чтобы остаток всегда был положительным. в таком случае частное увеличивается на единицу а остаток считается как разность умножения нового частного на делитель и делимого.

На вход: строки S_1 (делимое) и S_2 (делитель).

На выход: строка Q , представляющая частное и строка K , представляющая остаток.

```

string Calculator::divideWithSings(string str1, string str2) {
    bool isNegative1 = str1[0] == '-';

```

```

bool isNegative2 = str2[0] == '-';

if (isNegative1) str1 = str1.substr(1);
if (isNegative2) str2 = str2.substr(1);

string s = str1, s2 = str2;
s.erase(0, s.find_first_not_of('a'));
s2.erase(0, s2.find_first_not_of('a'));
if (s.empty() && s2.empty()) return "пустое множество";
if (s.empty()) return "[-ffffff; fffffff]";

string result = divideStrings(str1, str2);
string remainder = subtractStrings(str1, multiplyStrings(result, str2));

if (isNegative1) {
    result = addStrings(result, "b");
    remainder = subtractStrings(multiplyStrings(result, str2), str1);
}

if (remainder[0] == '-' || remainder[1] == '-') {
    remainder = "b";
}

if (isNegative1 ^ isNegative2) {
    result = "-" + result;
}

return result + " + " + remainder;
}

```

2.14 Деление строк (divideStrings)

Функция делит строку S_1 на строку S_2 , возвращая частное Q . Первоочередно осуществляется проверка деления на a . В этом случае будет происходить повторный ввод делителя до тех пор, пока он станет ненулевым и будет удовлетворять условиям. Далее происходит деление слева направо, текущий разряд прибавляем к остатку, если текущий остаток меньше делителя добавляем a в частное, иначе находим сколько раз делитель помещается в остатке. Повторяем до тех пор,

пока элементы не закончатся. Также учитывается деление на а.

На вход: строки S_1 (делимое) и S_2 (делитель).

На выход: строка Q , представляющая частное $S_1 \div S_2$.

```
string Calculator::divideStrings(const string& dividend, const string& divisor) {
    string div = divisor;
    while (div == "a") {
        cout << "Значение не может равняться 'а', введите корректное значение."
        << endl;
        cin >> div;
        for (size_t i = 0; i < div.length(); ++i) {
            if (div[i] < 'a' || div[i] > 'g') {
                div = "a";
            }
        }
        cout << dividend << endl;
        cout << div << endl;
        cin.ignore();
    }

    string result = "";
    string current = "";

    for (char digit : dividend) {
        current += digit;

        current.erase(0, current.find_first_not_of('a'));
        if (current.empty()) current = "a";

        if (Compare(current, div)) {
            result += 'a';
        }
        else {
            int count = 0;
            while (!Compare(current, div)) {
                current = subtractStrings(current, div);
                ++count;
            }
        }
    }
}
```

```

        }
        result += getLetter(count);
    }
}

result.erase(0, result.find_first_not_of('a'));
if (result.empty()) result = "a";

return result;
}

```

2.15 Проверка строки на валидность (validateString)

Функция проверяет, содержит ли строка S только символы из алфавита A .

Особенности реализации: Используется посимвольная проверка, что делает функцию линейной по сложности относительно длины строки.

На вход: строка S .

На выход: булево значение: `true`, если строка валидна, иначе `false`.

```

bool isValidInput(const string& input) {
    int count = input.find_first_not_of(' ');
    string s = input.substr(count);
    if (s.length() > 9 || (s.length() > 8 && s[0] != '-')) {
        return false;
    }

    if (input[0] == '-') {
        for (size_t i = 1; i < input.length(); ++i) {
            if (input[i] < 'a' || input[i] > 'g') {
                return false;
            }
        }
    }
    else {
        for (size_t i = 0; i < input.length(); ++i) {
            if (input[i] < 'a' || input[i] > 'g') {
                return false;
            }
        }
    }
}

```

```

    }
}

return true;
}

```

2.16 Определение операции в зависимости от знака(`calculateWithSigns`)

Функция определяет какую операцию выполнить, сложение или вычитание в зависимости от знака.

На вход: строки S_1 и S_2 .

На выход: вызов функции `subtractStrings()` или `addStrings()`.

```

string Calculator::calculateWithSigns(string str1, string str2, char operation) {
    bool isNegative1 = str1[0] == '-';
    bool isNegative2 = str2[0] == '-';

    if (isNegative1) str1 = str1.substr(1);
    if (isNegative2) str2 = str2.substr(1);

    string s1 = str1;
    string s2 = str2;
    s1.erase(0, s1.find_first_not_of('a'));
    s2.erase(0, s2.find_first_not_of('a'));

    string result = "";

    if (!isNegative1 && !isNegative2) {
        if (s1.size() >= s2.size()) {
            result = (operation == '+' ? addStrings(str1, str2) :
                subtractStrings(str1, str2);
        }
        else {
            result = (operation == '+' ? addStrings(str1, str2) : "-" +
                subtractStrings(str2, str1);
        }
    }
    else if (isNegative1 && isNegative2) {

```

```

    if (s1.size() >= s2.size()) {
        result = (operation == '+') ? "-" + addStrings(str1, str2) : "-" +
            subtractStrings(str1, str2);
    }
    else {
        result = (operation == '+') ? "-" + addStrings(str1, str2) :
            subtractStrings(str2, str1);
    }
}

else if (isNegative1 && !isNegative2) {
    result = (operation == '+') ? subtractStrings(str2, str1) : "-" +
        addStrings(str1, str2);
}

else if (!isNegative1 && isNegative2) {
    result = (operation == '+') ? subtractStrings(str1, str2) :
        addStrings(str1, str2);
}

if (result[0] == '-' && result[1] == '-') {
    result = result.substr(2);
}

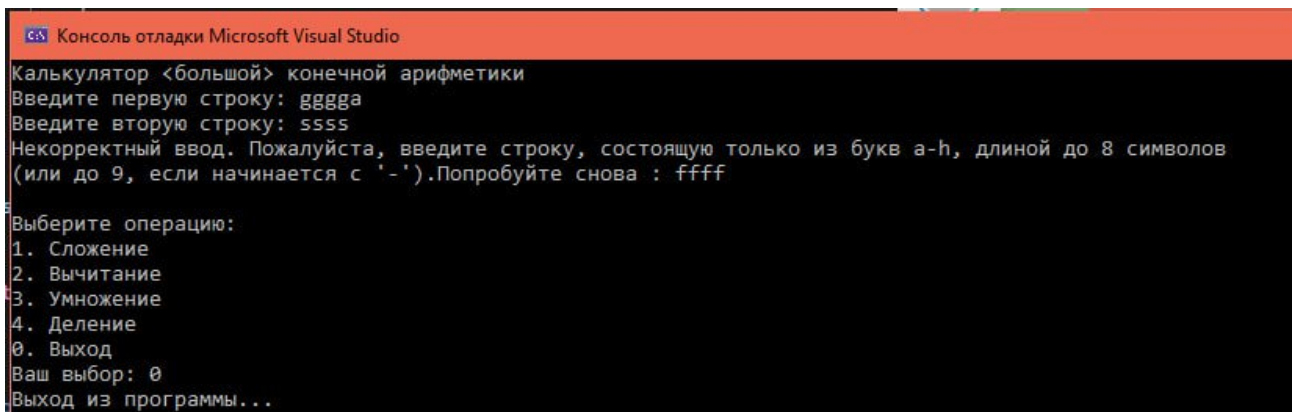
else if (result == "-a") {
    result = result.substr(1);
}

return result;
}

```

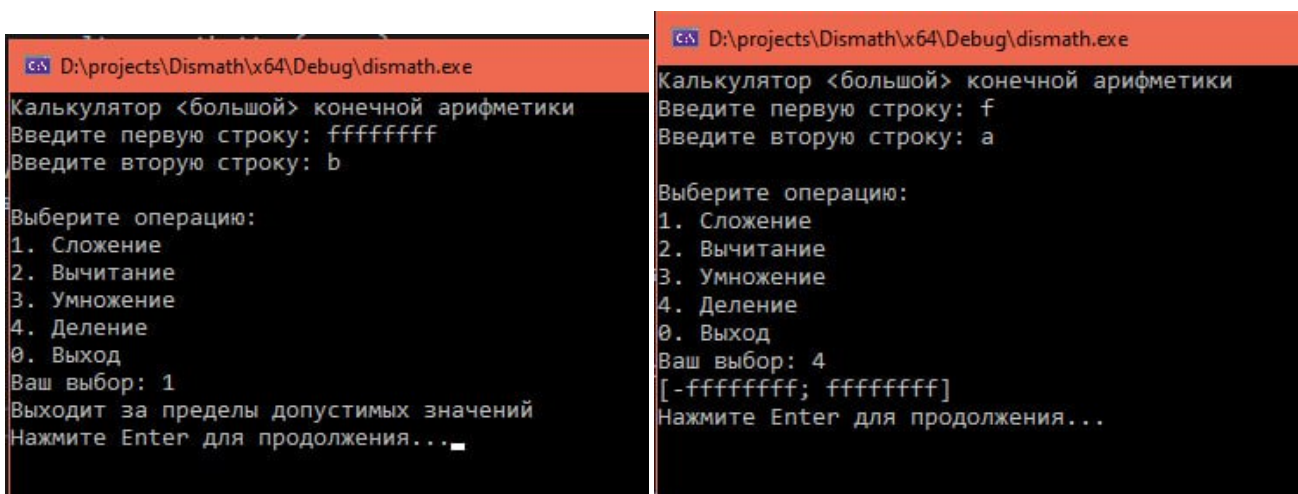

3 Результаты работы

В результате работы было реализовано 4 различных операций. Каждая операция визуализируется на консоли. Пользователь имеет возможность проводить операции поочередно, до момента пока сам не захочет закончить. Для надежности и корректной работы программы была реализована защита на ввод некорректных значений.



```
Консоль отладки Microsoft Visual Studio
Калькулятор <большой> конечной арифметики
Введите первую строку: gggga
Введите вторую строку: ssss
Некорректный ввод. Пожалуйста, введите строку, состоящую только из букв a-h, длиной до 8 символов
(или до 9, если начинается с \'-\'). Попробуйте снова : ffff
Выберите операцию:
1. Сложение
2. Вычитание
3. Умножение
4. Деление
0. Выход
Ваш выбор: 0
Выход из программы...
```

Рис. 1: Результат

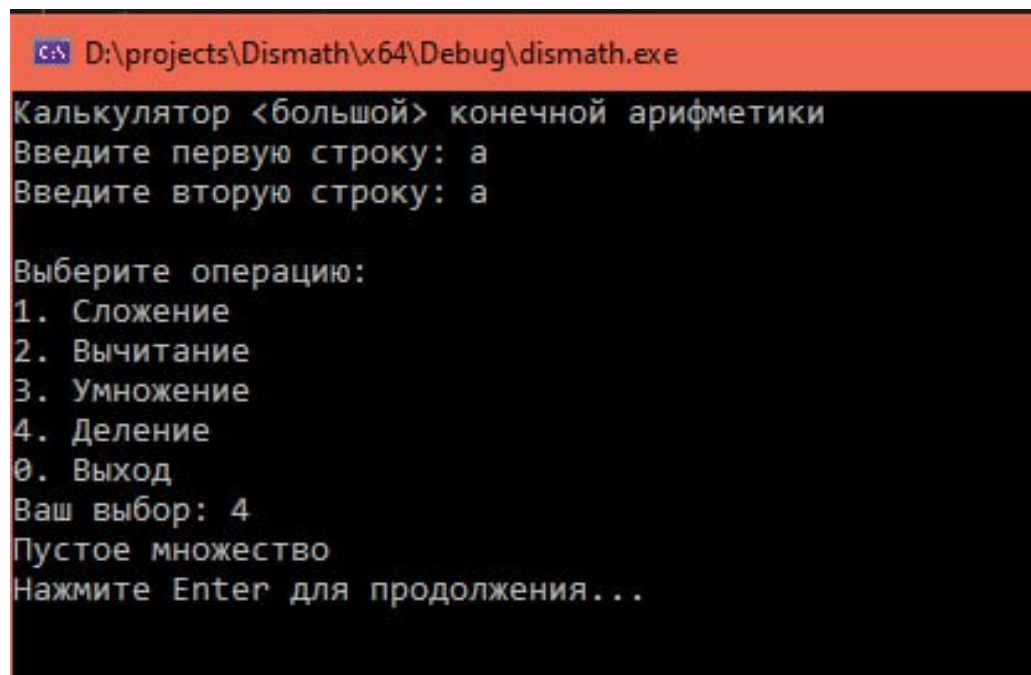


```
D:\projects\Dismath\x64\Debug\dismath.exe
Калькулятор <большой> конечной арифметики
Введите первую строку: fffffff
Введите вторую строку: b
Выберите операцию:
1. Сложение
2. Вычитание
3. Умножение
4. Деление
0. Выход
Ваш выбор: 1
Выходит за пределы допустимых значений
Нажмите Enter для продолжения...

D:\projects\Dismath\x64\Debug\dismath.exe
Калькулятор <большой> конечной арифметики
Введите первую строку: f
Введите вторую строку: a
Выберите операцию:
1. Сложение
2. Вычитание
3. Умножение
4. Деление
0. Выход
Ваш выбор: 4
[-ffffff; fffffff]
Нажмите Enter для продолжения...
```

Рис. 2: Переполнение

Рис. 3: Деление на 'a'



```
D:\projects\Dismath\x64\Debug\dismath.exe
Калькулятор <большой> конечной арифметики
Введите первую строку: a
Введите вторую строку: a

Выберите операцию:
1. Сложение
2. Вычитание
3. Умножение
4. Деление
0. Выход
Ваш выбор: 4
Пустое множество
Нажмите Enter для продолжения...
```

Рис. 3: Неопределенность

Заключение

В рамках выполнения курсовой работы была разработана программа для работы с конечной арифметикой, которая поддерживает операции сложения, вычитания, умножения и деления для чисел в системе $\langle M^8; +, * \rangle$, $M = \{a, b, c, d, e, f, g, h\}$. Программа позволяет выполнять все основные действия с числами, используя правила и свойства конечной арифметики, включая коммутативность и ассоциативность операций, а также наличие единичных элементов для сложения и умножения.

Главными достоинствами программы являются поддержка всех заданных операций, возможность работы с числами в конечной системе счисления и ясный, легко понимаемый код. Один из главных преимуществ кода это реализация при помощи индексации, так как это позволяет легко масштабировать программу не меняя структуру кода.

Недостатком можно назвать неудобный вызов функций сложения и вычитания, так как рассматривается 4 отдельных случая с большим количеством использования `if else`, а также частичное дальнейшее дублирование кода в функциях связанных со сложением и вычитанием. Также стоит отметить сложность тестирования и сложность отладки при больших значениях.

Масштабирование функции можно легко реализовать путем изменения исходной строки `alphabet`. Строку можно заменить на любую другую и даже если изменить размер строки программу не придется изменять, за исключением проверки на ввод. Логика позволяет масштабировать программу, все ограничения будут зависеть от вычислительной мощности.

Список материалов

- [1] Новиков, Ф.А. Дискретная математика для программистов. – СПб.: Питер, 2008. – С. 384.
<https://stugum.files.wordpress.com/2014/03/novikov.pdf>
- [2] wiki.fenix - сайт справочник по дисциплине математика(последнее открытие 12.12.2024)
<https://wiki.fenix.help/informatika/sdnf?ysclid=lr6bu660ud240614897>
- [3] Горбатов В. А. Фундаментальные основы дискретной математики. Информационная математика. — М.: Наука. Физматлит, 2000. — с. 544.
<https://studizba.com/show/1019108-1-gorbatov-va-fundamentalnye-osnovy.html>