

МИНОБРНАУКИ РОССИИ
«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
Дискретная математика
«Калькулятор «большой» конечной арифметики»
Вариант №13

Студент,

группа 5130201/30002

_____ Невечерин А. А.

Преподаватель

_____ Востров А. В.

«_____» _____ 2024г.

Санкт-Петербург
2024

Содержание

Введение	3
1 Математическое описание	4
1.1 Алгебраические структуры	4
1.2 Построение таблиц операций	5
1.3 Некоторые вычисления для построения таблиц	7
2 Особенности реализации программы	8
2.1 Класс Screen	8
2.1.1 ProgramStart	8
2.1.2 MainMenu	9
2.1.3 CheckElement	12
2.1.4 CheckAction	12
2.2 Класс Calculator	13
2.2.1 Конструктор	13
2.2.2 GenerateSummTable	13
2.2.3 GenerateDiffTable	14
2.2.4 GenerateComposTable	15
2.2.5 Summ	15
2.2.6 Diff	17
2.2.7 Compos	19
2.2.8 Div	20
2.2.9 LessComparison	22
2.2.10 MoreComparison	22
3 Результаты работы программы	24
Заключение	27
Список литературы	28

Введение

По итогам данной работы будет разработан калькулятор «большой» конечной арифметики $\langle Z_8^8; +, * \rangle$ (8 разрядов) для четырех действий $(+, -, *, \div)$ на основе «малой» конечной арифметики, где задано правило «+1» и выполняются свойства коммутативности $(+, *)$, ассоциативности $(+, *)$, дистрибутивности $*$ относительно $+$, заданы аддитивная единица «a» и мультипликативная единица «b», а также выполняется свойство: для $(\forall x) x * a = a$.

Правило «+1» задано в следующей таблице:

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

1 Математическое описание

1.1 Алгебраические структуры

Всюду определённая (тотальная) функция $\varphi_i : M^n \rightarrow M$ называется n -арной (n -местной) операцией на M .

Множество M вместе с набором операций $\Sigma = \{\varphi_1, \dots, \varphi_m\}$, $\varphi_i : M^{n_i} \rightarrow M$ где n_i — арность операции φ_i , называется алгебраической структурой, универсальной алгеброй или просто алгеброй.

Кольцо — это алгебраическая структура $\langle M; +, * \rangle$, в которой:

1. $(a + b) + c = a + (b + c)$
2. $\exists 0 \in M (\forall a (a + 0 = 0 + a = a))$
3. $\forall a \in M (a + (-a) = 0)$
4. $a + b = b + a$
5. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
6. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

Кольцо называется коммутативным, если:

7. $a \cdot b = b \cdot a$

Кольцо называется кольцом с единицей, если:

8. $\exists 1 \in M (a \cdot 1 = 1 \cdot a = a)$

«Малая» конечная арифметика — конечное коммутативное кольцо с единицей $\langle M_i; +, * \rangle$, на котором определены действия: вычитание «-» и деление без остатка « \div ». В данной работе имеем «малую» конечную арифметику $\langle M_8; +, * \rangle$, причем $M = \{a, b, c, d, e, f, g, h\}$.

«Большая» конечная арифметика — конечное коммутативное кольцо с единицей $\langle M_i^n; +, * \rangle$, на котором определены перенос, и действия: вычитание «-» и деление с остатком. « \div ». В данной работе имеем «большую» конечную арифметику вида $\langle M_8^8; +, * \rangle$.

1.2 Построение таблиц операций

Для правильной работы калькулятора были построены следующие таблицы операций:

Таблица 1. Операция «+»

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

Таблица 2. Перенос «+»

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

Таблица 3. Действие «-»

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

Таблица 4. Перенос «-»

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

Таблица 5. Операция «*»

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

Таблица 6. Перенос «*»

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

Также введем отношение строгого линейного порядка в малой конечной арифметике, которое определяется правилом «+1»: $a \prec b \prec g \prec h \prec c \prec e \prec f \prec d$

1.3 Некоторые вычисления для построения таблиц

1. $h + g = h + (b + b) = e$
2. $d + c = d + (h + b) = d + b + (g + b) = d + b + b + (b + b) = h$
3. $b + a = b$
4. $h * g = h * (b + b) = h + h = h + (g + b) = h + b + (b + b) = f$
5. $b * a = a$
6. $e * b = e$

2 Особенности реализации программы

В этом разделе будет подробно рассмотрена структура разработанного приложения. Отдельное внимание будет уделено его классовой архитектуре, т.е. будет показано какие классы были созданы, для чего и т.д. Также будет рассмотрена иерархия классов, их атрибуты и методы, что позволит лучше понять, как они взаимодействуют друг с другом для выполнения поставленных задач.

2.1 Класс Screen

Класс Screen отвечает за взаимодействие с пользователем через консольный интерфейс. Он управляет выводом данных на экран, обеспечивает ввод данных от пользователя и организует получение информации из других частей программы. Основными функциями класса являются отображение меню и обработка пользовательского ввода. Screen хранит в себе только указатель на объект класса Calculator.

2.1.1 ProgramStart

Вход: отношение порядка записанное в вектор символов.

Выход: создание экземпляра класса Calculator.

Метод запрашивает у пользователя ввод отношения порядка, проверяет корректность введённых данных и создает экземпляр калькулятора, передавая ему текущую арифметику.

```
1 void Screen::ProgramStart() {
2     settings::programShouldEnd = false;
3
4     cout << "Необходимо задать отношение порядка в формате
5         (a-b-c-d-e-f-g...): ";
6     string error_message = "\Вы задали некорректное отношение
7         \Попробуйте еще раз : ";
8
9     vector<string> current_arithmetic;
10
11     bool input_flag = true;
12     while (input_flag) {
13         current_arithmetic.clear();
14
15         int buffer_size = 64;
16         string entered_arithmetic = StrInput(buffer_size);
17
18         for (int symb_ind = 0; symb_ind < entered_arithmetic.size();
19             symb_ind++) {
20             if (symb_ind % 2 == 0 && entered_arithmetic[symb_ind] !=
21                 '-') {
22                 current_arithmetic.push_back({
23                     entered_arithmetic[symb_ind] });
24                 input_flag = false;
25             }
26             else if (symb_ind % 2 == 0 && entered_arithmetic[symb_ind]
27                 == '-') {
28                 cout << error_message;
```



```

23         input_flag = true;
24         break;
25     }
26     if (symb_ind % 2 != 0 && entered_arithmetic[symb_ind] !=
        ' ') {
27         input_flag = true;
28         cout << error_message;
29         break;
30     }
31 }
32 }
33
34 this->calculator = new Calculator(current_arithmetic);
35 MainMenu();
36 }

```

2.1.2 MainMenu

Вход: Экземпляр класса Calculator.

Выход: Вывод главного меню программы в консоль и обеспечение выполнения арифметических действий.

Метод выводит, полученные во время инициализации калькулятора, таблицы действий, меню для ввода арифметических выражений и обрабатывает пользовательский ввод. Он пытается выполнить арифметическое действие, основываясь на введенных данных, и в случае, если это возможно, выводит ее результат в консоль. Так же в этом методе происходит проверка на корректность введенных данных и переполнение результата действий.

```

1  void Screen::MainMenu() {
2      if (settings::programShouldEnd) return;
3      Clear();
4
5      cout << *calculator;
6
7      bool global_flag = true;
8
9      while (global_flag) {
10         cout << endl << "Введите выражениевозможные ( действия:
            +,-,*,/): ";
11
12         int buffer_size = 128;
13         string error_message = "\Вы ввелинекорректноевыражение
            \Попробуйте еще раз : ";
14
15         bool input_flag = true;
16         while (input_flag) {
17             string entered_arithmetic = StrInput(buffer_size);
18             vector<string> elements = splitstr(entered_arithmetic);
19
20             if (elements.size() == 1 && elements[0] == "x") {
21                 global_flag = false;
22                 settings::programShouldEnd = true;

```

```

23         break;
24     }
25
26     if (elements.size() != 3) {
27         cout << error_message;
28         continue;
29     }
30
31     elements[0].erase(0,
32         elements[0].find_first_not_of(this->calculator>
33         getCurrentArithmetic()[0]));
34     elements[2].erase(0,
35         elements[2].find_first_not_of(this->calculator>
36         getCurrentArithmetic()[0]));
37
38     if (elements[0].size() == 0) elements[0] =
39         this->calculator->getCurrentArithmetic()[0];
40     if (elements[2].size() == 0) elements[2] =
41         this->calculator->getCurrentArithmetic()[0];
42
43     if (elements[0][0] != '-' && elements[0].size() > 8) {
44         cout << "\Вы ввели слишком большое число \Попробуйте
45         еще раз : ";
46         continue;
47     }
48     if (elements[0][0] == '-' && elements[0].size() > 9) {
49         cout << "\Вы ввели слишком маленькое число \Попробуйте
50         еще раз : ";
51         continue;
52     }
53     if (elements[2][0] != '-' && elements[2].size() > 8) {
54         cout << "\Вы ввели слишком большое число \Попробуйте
55         еще раз : ";
56         continue;
57     }
58     if (elements[2][0] == '-' && elements[2].size() > 9) {
59         cout << "\Вы ввели слишком маленькое число \Попробуйте
60         еще раз : ";
61         continue;
62     }
63
64     if (!CheckElement(elements[0]) ||
65         !CheckElement(elements[2]) ||
66         !CheckAction(elements[1])) {
67         cout << error_message;
68         continue;
69     }
70
71     string answer;

```

```

63     if (elements[1] == "+") answer =
        calculator->Summ(elements[0], elements[2]);
64     if (elements[1] == "-") answer =
        calculator->Diff(elements[0], elements[2]);
65     if (elements[1] == "*") answer =
        calculator->Compos(elements[0], elements[2]);
66     if (elements[1] == "/") {
67         if (elements[2] ==
            this->calculator->getCurrentArithmetic()[0] &&
            elements[0] ==
            this->calculator->getCurrentArithmetic()[0]) {
68             string max_element =
                this->calculator->getCurrentArithmetic()[this->
69                 calculator->getCurrentArithmetic().size() -
                    1];
70             string ans;
71             for (int i = 0; i < 8; i++) {
72                 ans += max_element;
73             }
74             cout << "Результат: [" << ans << "; " << ans << "]" <<
                endl;
75             break;
76         }
77     else if (elements[2] ==
        this->calculator->getCurrentArithmetic()[0]) {
78         cout << "Результат: Пустоемножество !" << endl;
79         break;
80     }
81     else {
82         vector<string> div_answer =
            calculator->Div(elements[0], elements[2]);
83         if (div_answer[0][0] != '-' && div_answer[0].size() >
            8) {
84             cout << "Результат: Результатслишкомбольшой .
                Переполнение!";
85             break;
86         }
87         if (div_answer[0][0] == '-' && div_answer[0].size() >
            9) {
88             "Результат: Результатслишкоммаленький . Переполнение!";
89             break;
90         }
91         if (div_answer[1][0] != '-' && div_answer[1].size() >
            8) {
92             "Результат: Остатокслишкомбольшой . Переполнение!";
93             break;
94         }
95         if (div_answer[1][0] == '-' && div_answer[1].size() >
            9) {
96             "Результат: Остатокслишкоммаленький . Переполнение!";

```

```

97         break;
98     }
99
100     cout << "Результат: " << div_answer[0] << " Остаток: "
        << div_answer[1] << endl;
101     break;
102 }
103 }
104
105     if (answer[0] != '-' && answer.size() > 8) answer =
        "Результат слишкомбольшой . Переполнение!";
106     if (answer[0] == '-' && answer.size() > 9) answer =
        "Результат слишкоммаленький . Переполнение!";
107     cout << "Результат: " << answer << endl;
108     input_flag = false;
109 }
110 }
111 }

```

2.1.3 CheckElement

Вход: `element` — строка, представляющая число для проверки.

Выход: `true`, если введенное число может существовать в текущей арифметике, `false` в противном случае.

Метод проверяет, соответствует ли каждый символ в строке допустимым символам, используя список допустимых символов из текущей арифметики.

```

1  bool Screen::CheckElement(string element) {
2      vector<string> valid_characters =
        this->calculator->getCurrentArithmetic();
3      valid_characters.push_back("-");
4      bool all_valid = true;
5
6      for (char c : element) {
7          string char_as_string(1, c);
8          if (std::find(valid_characters.begin(),
                valid_characters.end(), char_as_string) ==
                valid_characters.end()) {
9              all_valid = false;
10             break;
11         }
12     }
13     return all_valid;
14 }

```

2.1.4 CheckAction

Вход: `action` — строка, представляющая арифметическое действие для проверки.

Выход: `true`, если действие является допустимым, и `false` в противном случае.

Метод проверяет, является ли переданная строка одним из допустимых арифметических действий.

```

1  bool Screen::CheckAction(string action) {
2      vector<string> valid_characters({"+", "-", "*", "/" });
3      bool all_valid = true;
4      if (std::find(valid_characters.begin(),
5                    valid_characters.end(), action) == valid_characters.end()) {
6          all_valid = false;
7      }
8      return all_valid;
9  }

```

2.2 Класс Calculator

Класс Calculator реализует логику арифметических действий, таких как сложение, вычитание, умножение и деление, на основе заданной арифметики. Он генерирует и управляет таблицами для каждого действия, обрабатывает входные данные и выдает результаты вычислений. Класс также обрабатывает условия, связанные с арифметическими действиями, учитывая знаки чисел и другие исключения.

Класс хранит в себе указатели на объекты класса Table, а так же вектор с текущей арифметикой калькулятора.

Класс Table по своей сути является оберткой структуры `map<string, map<string, string>`, в котором прописаны дополнительные методы для красивого и удобного вывода таблиц действий в консоль.

2.2.1 Конструктор

Вход: `arithmetic_vector` — вектор_символов (отношение порядка), по которым будет строится текущая арифметика.

Выход: Конструктор инициализирует объект калькулятора и создает таблицы действий.

Конструктор класса Calculator инициализирует переменную `current_arithmetic` и вызывает методы, генерирующие таблицы действий.

```

1  Calculator::Calculator(vector<string> arithmetic_vector) {
2      this->current_arithmetic = arithmetic_vector;
3
4      GenerateSummTable();
5      GenerateDiffTable();
6      GenerateComposTable();
7  }

```

2.2.2 GenerateSummTable

Вход: Два объекта класса Table: `summ_table` и `summ_trans_table`.

Выход: Генерируются таблицы суммирования и переноса суммирования.

Метод генерирует таблицу суммирования и таблицу переноса суммирования на основе текущей арифметики, выполняя операции по всем парам элементов.

```

1  void Calculator::GenerateSummTable() {
2      map<string, map<string, string>> temp_map;
3      map<string, map<string, string>> temp_trans_map;

```

```

4
5     for (size_t i = 0; i != this→current_arithmetic.size(); ++i) {
6         for (size_t j = i; j != this→current_arithmetic.size();
7             ++j) {
8             string result = this→current_arithmetic[(i + j) %
9                 this→current_arithmetic.size()];
10            string element1 = this→current_arithmetic[i];
11            string element2 = this→current_arithmetic[j];
12
13            temp_map[element1][element2] = result;
14            temp_map[element2][element1] = result;
15
16            bool is_trans = i + j >= this→current_arithmetic.size();
17            temp_trans_map[element1][element2] = is_trans ?
18                current_arithmetic[1] : current_arithmetic[0];
19            temp_trans_map[element2][element1] = is_trans ?
20                current_arithmetic[1] : current_arithmetic[0];
21        }
22    }
23
24    this→summ_table = new Table(temp_map, " + ");
25    this→summ_trans_table = new Table(temp_trans_map, " +1 ");
26 }

```

2.2.3 GenerateDiffTable

Вход: Два объекта класса Table: diff_table и diff_trans_table.

Выход: Генерируются таблицы вычитания и переноса вычитания.

Метод создает таблицы вычитания и переноса вычитания, используя данные из таблицы сложения. Имея «map_summ[key1, key2] = element» из таблицы суммирования, key1 и element меняются местами и таким образом заполняют таблицу вычитания («map_diff[element, key2] = key1»).

```

1 void Calculator::GenerateDiffTable() {
2     map<string, map<string, string>> temp_map;
3     map<string, map<string, string>> temp_trans_map;
4
5     for (const auto& [key1, row] : this→summ_table→getTable()) {
6         for (const auto& [key2, column_elemnet] : row) {
7             temp_map[key1][column_elemnet] = key2;
8             temp_trans_map[key1][column_elemnet] =
9                 this→summ_trans_table→getElementFromTable(key2, key1);
10        }
11    }
12
13    this→diff_table = new Table(temp_map, " - ");
14    this→diff_trans_table = new Table(temp_trans_map, " -1 ");
15 }

```

2.2.4 GenerateComposTable

Вход: Два объекта класса Table: compos_table и compos_trans_table.

Выход: Генерируются таблицы умножения и переноса умножения. Метод генерирует таблицу умножения и таблицу переноса умножения на основе текущей арифметики, выполняя операции по всем парам элементов.

```
1 void Calculator::GenerateComposTable() {
2     map<string, map<string, string>> temp_map;
3     map<string, map<string, string>> temp_trans_map;
4
5     for (size_t i = 0; i != this->current_arithmetic.size(); ++i) {
6         for (size_t j = i; j != this->current_arithmetic.size();
7             ++j) {
8             string result = this->current_arithmetic[(i * j) %
9                 this->current_arithmetic.size()];
10            string element1 = this->current_arithmetic[i];
11            string element2 = this->current_arithmetic[j];
12
13            temp_map[element1][element2] = result;
14            temp_map[element2][element1] = result;
15
16            bool is_trans = i * j >= this->current_arithmetic.size();
17            temp_trans_map[element1][element2] = is_trans ?
18                current_arithmetic[i * j /
19                    this->current_arithmetic.size()] :
20                current_arithmetic[0];
21            temp_trans_map[element2][element1] = is_trans ?
22                current_arithmetic[i * j /
23                    this->current_arithmetic.size()] :
24                current_arithmetic[0];
25        }
26    }
27
28    this->compos_table = new Table(temp_map, " * ");
29    this->compos_trans_table = new Table(temp_trans_map, " *1 ");
30 }
```

2.2.5 Summ

Вход: два объекта класса string - числа, для которых будет выполняться операция сложения.

Выход: строка - результат сложения данных объектов.

Метод Summ выполняет поразрядное суммирование двух символьных чисел путем прохода строк с числами от первого до последнего разряда, высчитывая на каждом шаге сумму двух символов на соответствующих позициях с помощью, ранее сгенерированных таблиц, также сохраняя значение переноса разряда.

Также в данном методе происходит обработка знаков введенных чисел.

```
1 string Calculator::Diff(string first_num, string second_num) {
2     int max_num_size;
3     bool minus_flag = false;
```

```

4
5 bool is_first_num_minus = first_num[0] == '-';
6 bool is_second_num_minus = second_num[0] == '-';
7
8 if (is_first_num_minus && is_second_num_minus) {
9     first_num.erase(0, 1);
10    second_num.erase(0, 1);
11    string tmp = first_num;
12    first_num = second_num;
13    second_num = tmp;
14 }
15 else if (is_first_num_minus) {
16     first_num.erase(0, 1);
17     minus_flag = true;
18     return "-" + this->Summ(second_num, first_num);
19 }
20 else if (is_second_num_minus) {
21     second_num.erase(0, 1);
22     return this->Summ(first_num, second_num);
23 }
24
25 if (LessComparison(first_num, second_num)) {
26     max_num_size = second_num.size();
27     int first_num_size = first_num.size();
28     for (int i = 0; i < max_num_size - first_num_size; i++) {
29         first_num = this->current_arithmetic[0] + first_num;
30     }
31     string tmp = first_num;
32     first_num = second_num;
33     second_num = tmp;
34     minus_flag = true;
35 }
36
37 else if (first_num.size() != second_num.size()) {
38     max_num_size = first_num.size();
39     int second_num_size = second_num.size();
40     for (int i = 0; i < max_num_size - second_num_size; i++) {
41         second_num = this->current_arithmetic[0] + second_num;
42     }
43 }
44
45 else max_num_size = first_num.size();
46
47 string result;
48 string trans = this->current_arithmetic[0]; string trans1;
49
50 for (int symb_ind = max_num_size - 1; symb_ind >= 0; symb_ind--) {
51     string diff_without_trans =
        this->diff_table->getElementFromTable(first_num[symb_ind],
        second_num[symb_ind]);

```



```

52     result =
        this→diff_table→getElementFromTable(diff_without_trans ,
        trans) + result;
53
54     trans1 =
        this→diff_trans_table→getElementFromTable(diff_without_trans ,
        trans);
55     trans =
        this→diff_trans_table→getElementFromTable(first_num[symb_ind] ,
        second_num[symb_ind]);
56
57     if (trans1 == current_arithmetic[1]) trans =
        current_arithmetic[1];
58 }
59
60 if (trans != this→current_arithmetic[0]) result = trans + result;
61 result.erase(0, result.find_first_not_of(current_arithmetic[0]));
62
63 if (result.size() == 0) result = current_arithmetic[0];
64 else if (minus_flag) result = "-" + result;
65
66 return result;
67 }

```

2.2.6 Diff

Вход: два объекта класса string - числа, для которых будет выполняться вычитание.

Выход: строка - результат вычитания данных объектов.

Метод Diff выполняет поразрядное вычитание двух символьных чисел путем прохода строк с числами от первого до последнего разряда, высчитывая на каждом шаге разность двух символов на соответствующих позициях, с помощью ранее сгенерированных таблиц, также сохраняя значение переноса разряда.

Также в данном методе происходит обработка знаков введенных чисел.

```

1  string Calculator::Diff(string first_num, string second_num) {
2      int max_num_size;
3      bool minus_flag = false;
4
5      bool is_first_num_minus = first_num[0] == '-';
6      bool is_second_num_minus = second_num[0] == '-';
7
8      if (is_first_num_minus && is_second_num_minus) {
9          first_num.erase(0, 1);
10         second_num.erase(0, 1);
11         string tmp = first_num;
12         first_num = second_num;
13         second_num = tmp;
14     }
15     else if (is_first_num_minus) {
16         first_num.erase(0, 1);
17         minus_flag = true;

```

```

18     return "-" + this->Summ(second_num, first_num);
19 }
20 else if (is_second_num_minus) {
21     second_num.erase(0, 1);
22     return this->Summ(first_num, second_num);
23 }
24
25 if (LessComparison(first_num, second_num)) {
26     max_num_size = second_num.size();
27     int first_num_size = first_num.size();
28     for (int i = 0; i < max_num_size - first_num_size; i++) {
29         first_num = this->current_arithmetic[0] + first_num;
30     }
31     string tmp = first_num;
32     first_num = second_num;
33     second_num = tmp;
34     minus_flag = true;
35 }
36
37 else if (first_num.size() != second_num.size()) {
38     max_num_size = first_num.size();
39     int second_num_size = second_num.size();
40     for (int i = 0; i < max_num_size - second_num_size; i++) {
41         second_num = this->current_arithmetic[0] + second_num;
42     }
43 }
44
45 else max_num_size = first_num.size();
46
47 string result;
48 string trans = this->current_arithmetic[0]; string trans1;
49
50 for (int symb_ind = max_num_size - 1; symb_ind >= 0;
51     symb_ind--) {
52     string diff_without_trans = this->diff_table->
53         getElementFromTable(first_num[symb_ind],
54             second_num[symb_ind]);
55     result =
56         this->diff_table->getElementFromTable(diff_without_trans,
57             trans) + result;
58
59     trans1 = this->diff_trans_table->
60         getElementFromTable(diff_without_trans, trans);
61     trans = this->diff_trans_table->
62         getElementFromTable(first_num[symb_ind],
63             second_num[symb_ind]);
64
65     if (trans1 == current_arithmetic[1]) trans =
66         current_arithmetic[1];
67 }

```

```

62
63     if (trans != this→current_arithmetic[0]) result = trans +
        result;
64     result.erase(0,
        result.find_first_not_of(current_arithmetic[0]));
65
66     if (result.size() == 0) result = current_arithmetic[0];
67     else if (minus_flag) result = "-" + result;
68
69     return result;
70 }

```

2.2.7 Compos

Вход: два объекта класса string - числа, для которых будет выполняться операция умножения.

Выход: строка - результат умножения данных объектов.

Метод Compos выполняет поразрядное умножение двух символьных чисел путем прохода строк с числами от первого до последнего разряда, высчитывая на каждом шаге произведение двух символов на соответствующих позициях, с помощью ранее сгенерированных таблиц, также сохраняя значение переноса разряда.

Также в данном методе происходит обработка знаков введенных чисел.

```

1  string Calculator::Compos(string first_num, string second_num) {
2      if (first_num == this→current_arithmetic[0] || second_num ==
        this→current_arithmetic[0]) return
        this→current_arithmetic[0];
3
4      bool is_first_num_minus = first_num[0] == '-';
5      bool is_second_num_minus = second_num[0] == '-';
6      bool minus_flag = false;
7
8      if (is_first_num_minus && is_second_num_minus) {
9          first_num.erase(0, 1);
10         second_num.erase(0, 1);
11     }
12     else if (is_first_num_minus) {
13         first_num.erase(0, 1);
14         minus_flag = true;
15     }
16     else if (is_second_num_minus) {
17         second_num.erase(0, 1);
18         minus_flag = true;
19     }
20
21     string result = this→current_arithmetic[0];
22     string trans; string trans1;
23
24     for (int symb_ind2 = second_num.size() - 1; symb_ind2 >= 0;
        symb_ind2--) {
25         trans = this→current_arithmetic[0];

```

```

26
27     string pod_result;
28     for (int i = 0; i < second_num.size() - 1 - symb_ind2; i++) {
29         pod_result += current_arithmetic[0];
30     }
31
32     for (int symb_ind1 = first_num.size() - 1; symb_ind1 >= 0;
33         symb_ind1--) {
34         string compos_without_trans = this->compos_table->
35             getElementFromTable(first_num[symb_ind1],
36                 second_num[symb_ind2]);
37         pod_result = this->summ_table->
38             getElementFromTable(compos_without_trans, trans) +
39                 pod_result;
40
41         trans1 = this->summ_trans_table->
42             getElementFromTable(compos_without_trans, trans);
43         trans = this->compos_trans_table->
44             getElementFromTable(first_num[symb_ind1],
45                 second_num[symb_ind2]);
46
47         trans = this->Summ(trans, trans1);
48     }
49     if (trans != this->current_arithmetic[0]) pod_result = trans
50         + pod_result;
51
52     result = this->Summ(result, pod_result);
53 }

```

2.2.8 Div

Вход: два объекта класса string - числа, для которых будет выполняться деление. Первое число - делимое, второе - делитель.

Выход: Вектор, состоящий из двух элементов. Первый элемент - целая часть результата деления объектов. Второй элемент - остаток результата деления двух объектов.

Отрицательные знаки, если они были у делителя или делимого, отбрасываются.

Метод Div циклически выполняет поиск подходящего множителя, на который нужно умножить делитель, чтобы получить делимое. Процесс останавливается, когда при увеличении множителя на единицу результат умножения станет больше делителя.

Если произведение полученного частного на делитель не равняется делимому, то метод вернет еще и остаток равный разности делимого и произведения частного на делитель.

Если делитель отрицательный, а делимое положительное, тогда результатом деления будет значение частного с отрицательным знаком. Остаток останется такой же.

Если делимое отрицательное, а делитель положительный, тогда результатом деления

будет частное увеличенное на мультипликативную единицу с отрицательным знаком. Остаток же будет равен разности произведения увеличенного частного на делитель и делимого.

Если делимое отрицательное и делитель отрицательный, тогда результатом деления будет частное увеличенное на мультипликативную единицу. Остаток же будет равен разности произведения увеличенного частного на делитель и делимого.

Если делитель равен аддитивной единице, тогда результатом действия будет пустое множество. Если и делимое, и делитель равны аддитивной единице, то мы получим отрезок значений от минимального возможного числа, до максимального возможного числа.

```

1    vector<string> Calculator::Div(string first_num, string
      second_num) {
2      bool is_first_num_minus = first_num[0] == '-';
3      bool is_second_num_minus = second_num[0] == '-';
4      bool minus_flag = false;
5      bool dop_minus_flag = false;
6
7      if (is_first_num_minus && is_second_num_minus) {
8        dop_minus_flag = true;
9        first_num.erase(0, 1);
10       second_num.erase(0, 1);
11     }
12     else if (is_first_num_minus) {
13       first_num.erase(0, 1);
14       dop_minus_flag = true;
15       minus_flag = true;
16     }
17     else if (is_second_num_minus) {
18       second_num.erase(0, 1);
19       minus_flag = true;
20     }
21
22     string temp_second_num = second_num;
23     string multiplier = this->current_arithmetic[1];
24     while (LessComparison(temp_second_num, first_num)) {
25       temp_second_num = Summ(temp_second_num, second_num);
26       multiplier = Summ(multiplier, this->current_arithmetic[1]);
27     }
28
29     if (multiplier == this->current_arithmetic[1] &&
        !dop_minus_flag) minus_flag;
30
31     if (MoreComparison(temp_second_num, first_num)) {
32       string ans;
33       string trans;
34       if (dop_minus_flag) {
35         ans = multiplier;
36         trans = Diff(Compos(multiplier, second_num), first_num);
37       }
38       else {
39         ans = Diff(multiplier, this->current_arithmetic[1]);

```

```

40     trans = Diff(first_num, Diff(temp_second_num, second_num));
41 }
42 return minus_flag ? vector<string>({ "-" + ans, trans }) :
    vector<string>({ ans, trans });
43 //return Diff(multiplier, this->current_arithmetic[1]);
44 }
45 else return minus_flag ? vector<string>({ "-" + multiplier,
    this->current_arithmetic[0] }) : vector<string>({
    multiplier, this->current_arithmetic[0] });
46 }

```

2.2.9 LessComparison

Вход: два объекта класса string - числа, которые будут сравниваться.

Выход: true, если первое число меньше второго, иначе false.

Метод сравнивает две строки на основе их размера и порядка элементов арифметики.

```

1 bool Calculator::LessComparison(string first_num, string
    second_num) {
2     if (first_num.size() < second_num.size()) return true;
3     if (first_num.size() > second_num.size()) return false;
4
5     for (int symb_ind = 0; symb_ind < first_num.size();
        symb_ind++) {
6         string ch = { first_num[symb_ind] };
7         string ch2 = { second_num[symb_ind] };
8         auto ind1 = distance(this->current_arithmetic.begin(),
            find(this->current_arithmetic.begin(),
                this->current_arithmetic.end(), ch));
9         auto ind2 = distance(this->current_arithmetic.begin(),
            find(this->current_arithmetic.begin(),
                this->current_arithmetic.end(), ch2));
10
11         if (ind1 < ind2) return true;
12
13         if (ind1 > ind2) return false;
14     }
15
16     return false;
17 }

```

2.2.10 MoreComparison

Вход: два объекта класса string - числа, которые будут сравниваться.

Выход: true, если первое число больше второго, иначе false.

Метод сравнивает две строки на основе их размера и порядка элементов арифметики.

```

1 bool Calculator::MoreComparison(string first_num, string
    second_num) {
2     if (first_num.size() > second_num.size()) return true;
3     if (first_num.size() < second_num.size()) return false;

```

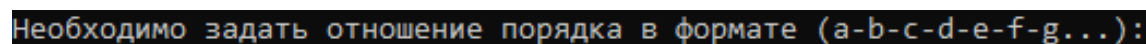
```

4
5     for (int symb_ind = 0; symb_ind < first_num.size();
        symb_ind++) {
6         string ch = { first_num[symb_ind] };
7         string ch2 = { second_num[symb_ind] };
8         auto ind1 = distance(this→current_arithmetic.begin(),
                            find(this→current_arithmetic.begin(),
                                this→current_arithmetic.end(), ch));
9         auto ind2 = distance(this→current_arithmetic.begin(),
                            find(this→current_arithmetic.begin(),
                                this→current_arithmetic.end(), ch2));
10
11         if (ind1 > ind2) return true;
12
13         if (ind1 < ind2) return false;
14     }
15
16     return false;
17 }

```

3 Результаты работы программы

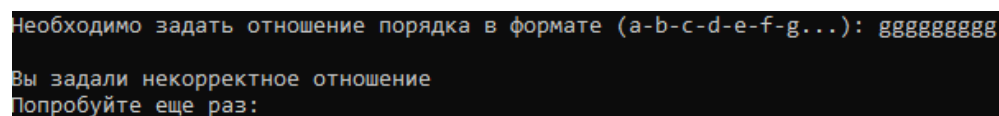
1) После запуска программы пользователь просят ввести отношение порядка, которое будет использоваться в калькуляторе.



```
Необходимо задать отношение порядка в формате (a-b-c-d-e-f-g...):
```

Рис. 1. Старт программы

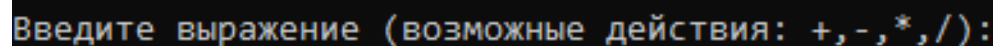
2) При некорректном вводе, пользователь получает сообщение об ошибке.



```
Необходимо задать отношение порядка в формате (a-b-c-d-e-f-g...): gggggggggg  
Вы задали некорректное отношение  
Попробуйте еще раз:
```

Рис. 2. Ошибка ввода отношения порядка

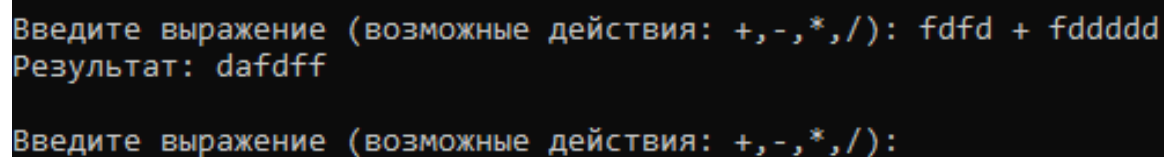
3) После ввода корректного отношения порядка, пользователь получает доступ к главному меню.



```
Введите выражение (возможные действия: +, -, *, /):
```

Рис. 3. Главное меню

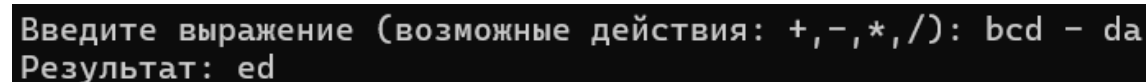
4) Результат выполнения сложения.



```
Введите выражение (возможные действия: +, -, *, /): fdfd + fddddd  
Результат: dafdff  
Введите выражение (возможные действия: +, -, *, /):
```

Рис. 4. Выполнение сложения

5) Результат выполнения вычитания.



```
Введите выражение (возможные действия: +, -, *, /): bcd - da  
Результат: ed
```

Рис. 5. Выполнение вычитания

6) Результат выполнения умножения.


```
Введите выражение (возможные действия: +,-,*,/): fdfd * gd
Результат: gscsfb
```

Рис. 6. Выполнение умножения

7) Результат выполнения деления.

```
Введите выражение (возможные действия: +,-,*,/): dffaa / fdh
Результат: bba Остаток: hea

Введите выражение (возможные действия: +,-,*,/):
```

Рис. 7. Выполнение деления

8) Ошибка при некорректном вводе выражения.

```
Введите выражение (возможные действия: +,-,*,/): fdbnsfbs++++43fs
Вы ввели некорректное выражение
Попробуйте еще раз: █
```

Рис. 8. Ошибка некорректного ввода

9) При вводе числа, выходящего за границы допустимых значений.

```
Вы ввели некорректное выражение
Попробуйте еще раз: ffffffffffffffffffff + f

Вы ввели слишком большое число
Попробуйте еще раз : █
```

Рис. 9. Отмена переполнения

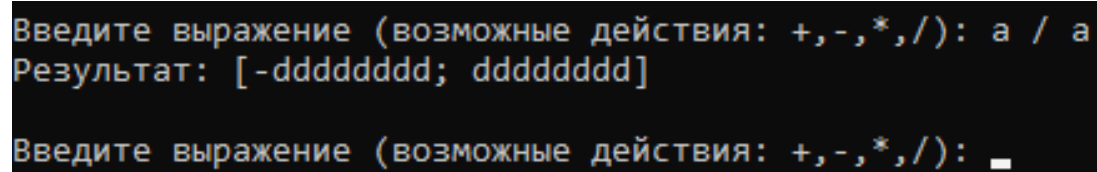
10) При получении результата, выходящего за границы допустимых значений.

```
Введите выражение (возможные действия: +,-,*,/): fffff * fffff
Результат: Переполнение!

Введите выражение (возможные действия: +,-,*,/): |
```

Рис. 10. Ошибка переполнения результата

11) Деление a на a .

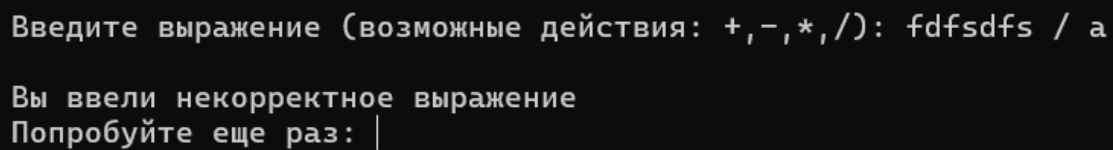


```
Введите выражение (возможные действия: +,-,*,/): a / a
Результат: [-dddddddd; dddddddd]

Введите выражение (возможные действия: +,-,*,/): _
```

Рис. 11. Результат деления a на a

12) Деление любого символа на a .



```
Введите выражение (возможные действия: +,-,*,/): fdfsdfs / a
Вы ввели некорректное выражение
Попробуйте еще раз: |
```

Рис. 12. Результат деления любого числа на a

Заключение

В результате выполнения данной курсовой работы был разработан калькулятор «большой» конечной арифметики $\langle Z_8^8; +, * \rangle$ для четырех действий $(+, -, *, \div)$ на основе «малой» конечной арифметики, где задано правило «+1» и выполняются свойства коммутативности $(+, *)$, ассоциативности $(+, *)$, дистрибутивности $*$ относительно $+$, заданы аддитивная единица «a» и мультипликативная единица «b», а также выполняется свойство: для $(\forall x) x * a = a$. Программа умеет выполнять такие действия как: сложение, вычитание, умножение, деление. Также программа контролирует пользовательский ввод и не дает пользователю ввести некорректные данные.

Достоинства программы:

- Пользователь может ввести большое количество различных отношений порядка и работать со своей собственной арифметикой. Ограничение на вводимые отношения порядка - алфавит арифметики, получаемый из введенного отношения порядка, должен иметь минимум 3 различных символа, максимум 26 символов).
- Программа предоставляет пользователю вывод всех таблиц действий.
- Разделение вычислений и контроля результатов, а также реализация взаимодействия с введенной арифметикой дают возможность быстро масштабировать калькулятор до больших разрядностей
- Отсутствие привязанности к конкретным символам.

Недостатки программы:

- Для деления используется неоптимизированный алгоритм, в котором при увеличении делимого, время выполнения программы значительно увеличивается.
- Строгий формат ввода выражений.

Масштабирование: в программу можно добавить следующие функции:

- Сохранение результатов вычисления для возможности работы с более сложными выражениями.
- Управление порядком действий.
- Работа с такими действиями как: НОД, НОК и возведение в степень.

Список литературы

1. Секция "Телематика"/ текст : электронный / — URL: <https://tema.spbstu.ru/dismath/> (Дата обращения 27.12.2024).
2. Новиков Ф. А. Дискретная математика для программистов. 3-е изд. — Санкт-Петербург: Питер Пресс, 2009. - 384стр.
3. Microsoft C++ Standart Library Documentation / текст : электронный / - URL: <https://learn.microsoft.com/ru-ru/cpp/standard-library/> (Дата обращения 27.12.2024).