

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

Дискретная математика  
ОТЧЁТ ПО КУРСОВОЙ РАБОТЕ

Калькулятор «большой» конечной арифметики  
Вариант 11

Студент,  
группы 5130201/30002

\_\_\_\_\_ Михайлова А. А.

Преподаватель

\_\_\_\_\_ Востров А. В.

«\_\_\_\_\_» \_\_\_\_\_ 2024 г.

Санкт-Петербург, 2024

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Математическое описание</b>	<b>4</b>
1.1 «Малая» и «большая» конечные арифметики . . . . .	4
1.2 Свойства операций в алгебре . . . . .	4
1.3 Таблицы операций «малой» конечной арифметики . . . . .	5
1.4 Примеры вычислений . . . . .	6
<b>2 Особенности реализации программы</b>	<b>7</b>
2.1 Функция print_menu . . . . .	7
2.2 Функция Menu . . . . .	7
2.3 Функция Compare . . . . .	8
2.4 Функция full . . . . .	9
2.5 Функция Print . . . . .	9
2.6 Функция Print_div . . . . .	10
2.7 Функция Input_char . . . . .	11
2.8 Функция Summa_with_negative . . . . .	12
2.9 Функция Summa . . . . .	14
2.10 Функция difference . . . . .	16
2.11 Функция sup . . . . .	17
2.12 Функция Division . . . . .	18
<b>3 Результаты работы программы</b>	<b>21</b>
<b>Заключение</b>	<b>25</b>
<b>Список использованной литературы</b>	<b>26</b>

# Введение

Курсовая работа заключается в реализации калькулятора «большой» конечной арифметики  $\langle Z_i; +; * \rangle$  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 любого  $x$ :  $x*a = a$ .

Курсовая работа выполнена на языке C++ в среде разработки XCode.

Правило «+1» указано в таблице 1.

Таблица 1: Правило +1

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

# 1 Математическое описание

## 1.1 «Малая» и «большая» конечные арифметики

Алгебраическая структура — множество значений, над которыми определены операции и отношения.

Коммутативное кольцо с единицей — это алгебраическая структура  $\langle M; +, * \rangle$ , в которой:

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

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

«Малая» конечная арифметика — конечное коммутативное ассоциативное кольцо с единицей  $\langle M_i; +, * \rangle$ , на котором определены действия вычитания « $-$ » и деления « $\div$ », причём действие деления определено частично.

В нашем случае  $i = 8$  и «малая» конечная арифметика принимает вид  $\langle M_8; +, * \rangle$ .

«Большая» конечная арифметика — конечное коммутативное ассоциативное кольцо с единицей  $\langle M_i^n; +, * \rangle$ , на котором определены действия вычитания « $-$ » и деления « $\div$ », причём деление определено с остатком.

В нашем случае  $n = 8, i = 8$ , и «большая» конечная арифметика принимает вид  $\langle M_8^8; +, * \rangle$ .

## 1.2 Свойства операций в алгебре

1. Ассоциативность сложения:

$$\forall a, b, c \in M : a + (b + c) = (a + b) + c; \quad a * (b * c) = (a * b) * c;$$

2. Коммутативность сложения:

$$\forall a, b \in M : a + b = b + a; \quad a * b = b * a;$$

3. Дистрибутивность сложения относительно умножения:

$$\forall a, b, c \in M : a * (b + c) = (a * b) + (a * c);$$

4. Нейтральный элемент по сложению:

$$\exists a \in M : \forall x \in M : x + a = a + x = x;$$


5. Нейтральный элемент по умножению:

$$\exists a \in M : \forall x \in M : x * a = a * x = x.$$

### 1.3 Таблицы операций «малой» конечной арифметики

Введём отношение порядка в «малой» конечной арифметике, которое определяется правилом «+1»:  $a < b < h < f < c < e < g < d$

На Рис. 1 представлены таблицы операции и переноса разряда для «+». На Рис. 2 представлены таблицы операции и переноса разряда для «\*».

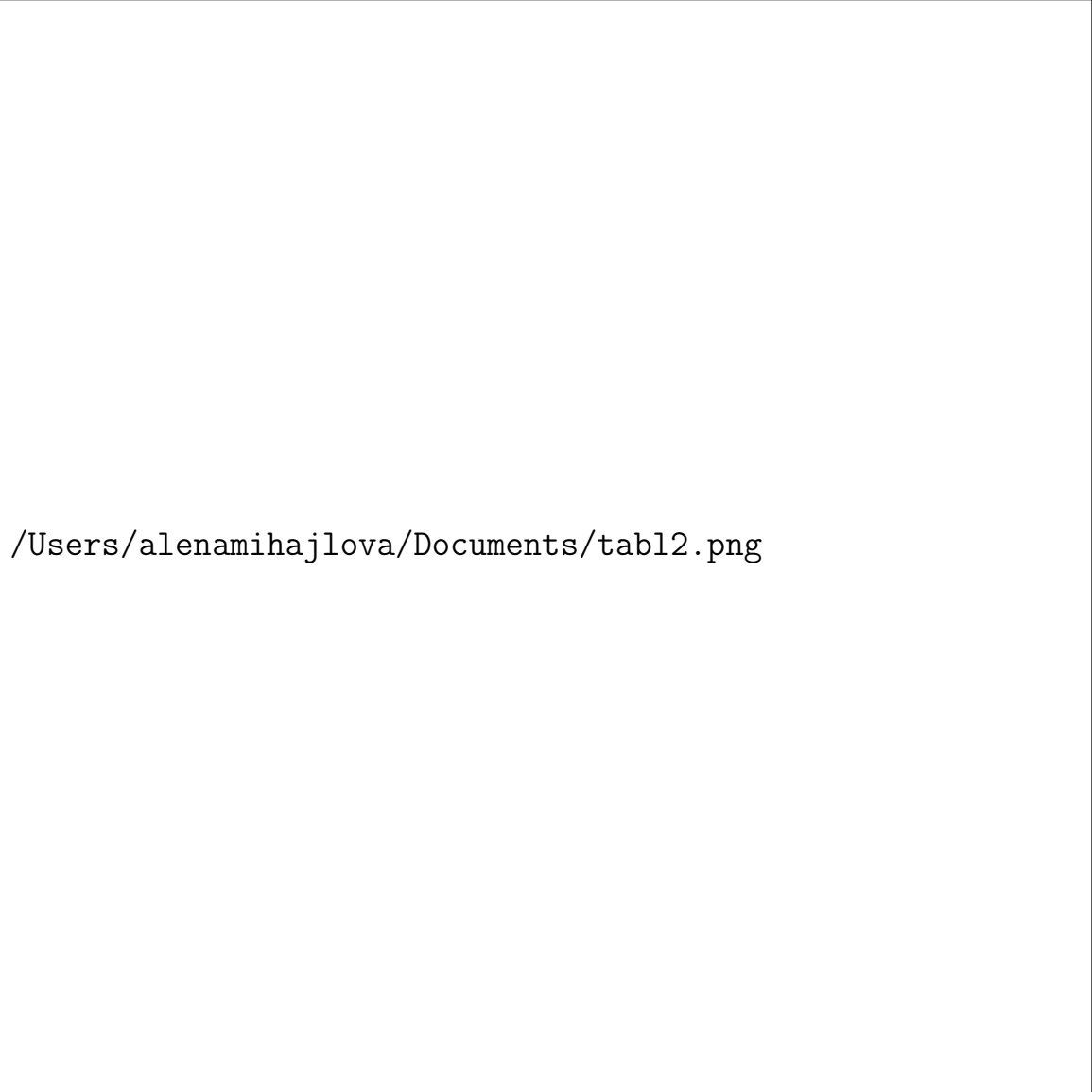


/Users/alenamihajlova/Documents/taaabl.png

Рис. 1: Таблицы операции и переноса разряда «+»

## 1.4 Примеры вычислений

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



/Users/alenamihajlova/Documents/tab12.png

Рис. 2: Таблицы операции и переноса разряда «\*»

## 2 Особенности реализации программы

### 2.1 Функция print\_menu

Вход: введённые числа `vector<int> first` и `vector<int> second`.

Выход: меню выбора действия.

Функция `print_menu` выводит на экран выбор следующего действия после ввода чисел для пользователя.

Ознакомиться с кодом можно в Листинг 1.

Листинг 1: Функция `print_menu`

```
void print_menu()
{
    printf("0_ _выход\n");
    printf("1_ _сложение\n");
```

```

printf("2_ _вычитание\n");
printf("3_ _умножение\n");
printf("4_ _деление\n");
}

```

## 2.2 Функция Menu

Вход: введённые числа `vector<int> first` и `vector<int> second`.

Выход: результат выбранной операции.

Функция Menu служит основным меню программы. Реализует выбор операции и вызов функции вывода результата на экран.

Ознакомиться с кодом можно в Листинг 2.

Листинг 2: Функция Menu

```

void Menu()
{
    int action;
    string word;
    print_menu();
    cin.clear();
    cin.ignore(cin.rdbuf()->in_avail());
    getline(cin, word);
    while (!(word.find_first_not_of("01234") == string::npos))
    {
        cout << "вы_ввели_цифру_не_из_диапазона.__
попробуйте_снова." << endl;
        cin.clear();
        cin.ignore(cin.rdbuf()->in_avail());
        getline(cin, word);
    }
    action=stoi(word);
    switch (action)
    {
        case 0:
            exit(0);
        case 1:
            Summa(first, second);
            Print();
            break;
        case 2:
            difference(first, second);
            Print();
    }
}

```



```

        break;
        case 3:
        Sup(first, second);
        Print();
        break;
        case 4:
        Division(first, second);
        break;
        default:
        break;
    }
}

```

## 2.3 Функция Compare

Вход: введенные числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: наибольшее из чисел.

Функция `Compare` служит поиском наибольшего из введенных чисел.

Ознакомиться с кодом можно в Листинг 3.

Листинг 3: Функция `Compare`

```

vector<int> Compare(vector<int>& vector1, vector<int>& vector2)
{
    if (vector1.size() > vector2.size())
        return vector1;
    else return vector2;
}

```

## 2.4 Функция full

Вход: введенные числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: заполненное нулями до нужного размера минимальное число.

Функция `full` предназначена для выравнивания двух векторов целых чисел `vector1` и `vector2` по длине, добавляя нули в меньший вектор.

Ознакомиться с кодом можно в Листинг 4.

Листинг 4: Функция `full`

```

void full(vector<int>& vector1, vector<int>& vector2)
{
    int diff = vector1.size() - vector2.size();
    diff = abs(diff);
}

```

```

    if (diff!=0)
    {
        if (Compare(vector1, vector2) == vector1)
        {
            vector2.insert(vector2.end(), diff, 0);
        }
        else
        {
            vector1.insert(vector1.end(), diff, 0);
        }
    }
    else return;
}

```

## 2.5 Функция Print

Вход: результат операции result.

Выход: печать результата.

Функция Print предназначена для печати результата операции в консоль.

Ознакомиться с кодом можно в Листинг 5.

Листинг 5: Функция Print

```

void Print()
{
    if ((result.size() >= 10) && (result[0] == '-'))
    {
        cout << "переполнение. доступный диапазон: [-\n";
        cout << "dddddd;dddddd]. попробуйте снова." << endl;
    }
    else if (result.size() >= 9 && (result[0] != '-'))
    {
        cout << "переполнение. доступный диапазон: [-\n";
        cout << "dddddd;dddddd]. попробуйте снова." << endl;
    }
    else
    {
        Cut(result);
        cout << "ответ:";
        for (int i = 0; i < result.size(); i++)
        {
            cout << result[i];
        }
    }
}

```

```

        cout << endl;
    }
}

```

## 2.6 Функция Print\_div

Вход: знак - bool znak, целая часть - vector<int> cel и остаток - vector<int> ostat.

Выход: печать результата при делении.

Функция Print\_div предназначена для печати результата операции деления в консоль.

Ознакомиться с кодом можно в Листинг 6.

Листинг 6: Функция Print\_div

```

void Print_div(bool znak, vector<int> cel, vector<int> ostat)
{
    vector<char> celaya_chast;
    vector<char> ostatoc;
    cout << "ответ:";
    for (int i = 0; i < cel.size(); i++)
    {
        celaya_chast.push_back(Arifmetic[cel[i]]);
    }
    if ((znak1 == true && znak2 == false || znak2 == true &&
znak1 == false)&&znak==false||
    ((znak1 == true && znak2 == false || znak2 == true &&
znak1 == false) && first == second))
    {
        celaya_chast.insert(celaya_chast.begin(), '-');
    }
    for (int i = 0; i < ostat.size(); i++)
    {
        ostatoc.push_back(Arifmetic[ostat[i]]);
    }
    for (int i = 0; i < celaya_chast.size(); i++)
    {
        cout << celaya_chast[i];
    }
    Cut(ostatoc);
    cout << '(';
    for (int i = 0; i < ostatoc.size(); i++)

```

```

    {
        cout << ostatoc[i];
    }
    cout << ') ' << endl;
}

```

## 2.7 Функция Input\_char

Вход: вектор для хранения числа - `vector<int>& vector` и номер числа - `int num`.

Выход: заполненный числом вектор.

Функция `Input_char` предназначена для ввода пользователем числа.

Ознакомиться с кодом можно в Листинг 7.

Листинг 7: Функция `Input_char`

```

void Input_char(vector<int> &vector, int num)
{
    vector.clear();
    string word;
    cin.clear();
    cin.ignore(cin.rdbuf()->in_avail());
    getline(cin, word);
    while(word.size()==0)
    {
        cout << "Error" << endl;
        vector.clear();
        word.clear();
        cin >> word;
    }
    while ((word.size() >=9 && word[0]!='-') || (word.size() >=
10 && word[0] == '-'))
    {
        cout << "Overflow! Acceptable range: [-ddddddd;
ddddddd]. Try again." << endl;
        vector.clear();
        word.clear();
        cin >> word;
    }
    int i=0;
    bool znak_of=false;
    sign(word, num);
    num == 1 ? znak_of = znak1 : znak_of = znak2;
}

```

```

znak_of ? i = 1 : i = 0;
for (i; i <= word.size()-1; i++)
{
    char x = word[i];
    if (x >= 'a' && x <= 'h')
    {
        vector.push_back(Index(x));
    }
    else
    {
        cout << "Error" << endl;
        vector.clear();
        word.clear();
        znak_of ? i = 0 : i = -1;
        cin >> word;
    }
}
reverse(vector.begin(), vector.end());
}

```

## 2.8 Функция Summa\_with\_negative

Вход: введенные числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: результат операции.

Функция `Summa_with_negative` выполняет сложение двух векторов целых чисел `vector1` и `vector2`, где один из множителей имеет отрицательное значение. Определяется максимальный `maxim` и минимальный `minim` векторы. Устанавливается флаг `perenos`, который отслеживает необходимость переноса при вычитании. Цикл проходит по каждому элементу векторов. Внутри используется указатель `it`, который определяется как индекс в массиве `Arifmetic`, смещенный на значение текущего элемента из `maxim` с учетом возможного переноса. Если текущий элемент равен нулю, указатель `it` перемещается в конец массива `Arifmetic`, устанавливая флаг переноса. Для каждого элемента из `minim`, выполняется вычитание: указатель `it` уменьшается, при этом также учитывается возможность выхода за пределы, что снова приводит к перемещению в конец массива и установке флага переноса при необходимости. После завершения цикла вектор `result` переворачивается. Если `maxim` равен значению, представляющему отрицательное число, то в начало `result` добавляется символ минус.

Ознакомиться с кодом можно в Листинг 8.

Листинг 8: Функция Summa\_with\_negative

```
vector<int> Summa_with_negative(vector<int>& vector1, vector<int>&
    vector2)
{
    val_result.clear();
    vector<int> maxim = Max_numb(vector1, vector2);
    vector<int> minim;

    maxim == vector1 ? minim = vector2 : minim = vector1;

    bool perenos = false;

    for (int i = 0; i < vector1.size(); i++)
    {
        auto it = begin(Arifmetic) + maxim[i] - (perenos ?
1 : 0);

        if (*it == '\0')
        {
            it = end(Arifmetic) - 1;
            perenos = true;
        }
        else
        {
            perenos = false;
        }

        for (int j = 0; j < minim[i]; j++)
        {
            if (it == begin(Arifmetic))
            {
                it = end(Arifmetic) - 1;
                perenos = true;
            }
            else
            {
                it--;
            }
        }

        result.push_back(*it);
        val_result.push_back(Index(*it));
    }
}
```

```

reverse(result.begin(), result.end());

if (maxim == is_negative())
{
    result.insert(result.begin(), '-');
}

return val_result;
}

```

## 2.9 Функция Summa

Вход: введённые числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: результат операции.

Функция `Summa` выполняет сложение двух векторов целых чисел `vector1` и `vector2` с учетом возможного переноса и знака результата. Если одно из условий отрицательного числа выполнено, вызывается вспомогательная функция `Summa_with_negative`. В противном случае происходит поэлементное сложение: для каждого элемента из `vector1` добавляется значение из `Arifmetic`, учитывая возможный перенос от предыдущего сложения.

Ознакомиться с кодом можно в Листинг 9.

Листинг 9: Функция Summa

```

vector<int>& Summa(vector<int>& vector1, vector<int>& vector2,
    bool sc) {
    ostat_result.clear();
    result.clear();

    if (Negativ() && sc) {
        ostat_result = Summa_with_negative(vector1,
vector2);
        return ostat_result;
    } else {
        bool perenos = false;

        for (int i = 0; i < vector1.size(); i++) {
            auto it = begin(Arifmetic) + vector1[i] +
(perenos ? 1 : 0);

            if (*it == '\0') {
                it = begin(Arifmetic);

```

```

        perenos = true;
    } else {
        perenos = false;
    }

    for (int j = 0; j < vector2[i]; j++) {
        it++;
        if (it == end(Arifmetic)) {
            it = begin(Arifmetic);
            perenos = true;
        }
    }
    result.push_back(*it);
    ostat_result.push_back(Index(*it));
}
reverse(result.begin(), result.end());
if (perenos) {
    result.insert(result.begin(), Arifmetic
[1]);
    ostat_result.insert(ostat_result.end(), 1)
;

}
if (znak1 == true && znak2 == true) {
    result.insert(result.begin(), '-');
}

return ostat_result;
}
}

```

## 2.10 Функция difference

Вход: введённые числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: результат операции.

Функция `difference` рассчитывает разность между двумя векторами целых чисел `vector1` и `vector2`, учитывая знаки операндов. Сначала создается пустой вектор `res` для хранения результата. Если второй знак отрицательный, а первый положительный, то `znak2` устанавливается в положительное, и функция вызывает `Summa`, чтобы выполнить сложение, так как вычитание в этом случае эквивалентно сложению. Если первый знак отрицательный, а второй положительный, `znak1` также устанавливается в положительное, производится сложение с помощью `Summa`, после чего в результат добавляется



символ «-», чтобы указать, что итоговое значение является отрицательным. В противном случае функции рассматривается вариант, когда оба числа отрицательные, и тогда вызывается `Summa_with_negative` для вычисления разницы с учетом отрицательных значений. В каждом из случаев функция возвращает итоговый вектор с результатом.

Ознакомиться с кодом можно в Листинг 10.

Листинг 10: Функция `difference`

```
vector<int> difference(vector<int>& vector1, vector<int>&
vector2)
{
    vector<int> res;
    if (znak2 == true && znak1==false )
    {
        znak2 = false;
        return Summa(vector1, vector2);
    }
    else if (znak1 == true && znak2==false)
    {
        znak1 = false;
        res=Summa(vector1, vector2);
        result.insert(result.begin(), '-');
        return res;
    }
    else
    {
        return Summa_with_negative(vector1,
vector2);
    }
}
```

## 2.11 Функция `sup`

Вход: введенные числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: результат операции.

Функция `Sup` реализует операцию умножения двух векторов целых чисел `vector1` и `vector2`. В начале функции определяется максимальный и минимальный множитель. Если один из знаков (`znak1` или `znak2`) отрицательный, устанавливается флаг `minus`, и оба знака сбрасываются. Затем начинается цикл, который продолжается, пока значение `mnozhitel_min` больше единицы. В каждой итерации вызывается функция `full`, чтобы обновить значения множителей, производя сложение `mnozhitel_max` с `mnozhitel`. При этом `mnozhitel_max` обновляется с помощью `Summa`, а `mnozhitel_min` уменьшает-

ся на единицу через difference. После завершения цикла очищается вектор result, и проверяется, не равен ли один из исходных векторов нулю. Если это так, в result добавляется символ 'а', указывающий на ошибку, и функция завершается. В противном случае в result добавляются символы из Arithmetic, соответствующие значениям вектора mnozhitel\_max. В результате вектор result переворачивается, и, если установлен флаг minus, добавляется символ минус в начало.

Ознакомиться с кодом можно в Листинг 11.

Листинг 11: Функция sup

```
void Sup(vector<int>& vector1, vector<int>& vector2){
    vector<int> mnozhitel_max = Max_numb(vector1, vector2);
    vector<int> mnozhitel = Max_numb(vector1, vector2);
    vector<int> mnozhitel_min;
    vector<int> edinica(1, 1);
    mnozhitel_max == vector1 ? mnozhitel_min = vector2 :
mnozhitel_min = vector1;
    bool minus = false;
    if (znak1 == true && znak2 == false || znak1 == false &&
znak2 == true)
    {
        minus = true;
        znak1 = false;
        znak2 = false;
    }
    if (znak1 == true && znak2 == true)
    {
        znak1 = false;
        znak2 = false;
    }
    while (Vector_to_value(mnozhitel_min) > 1)
    {
        full(mnozhitel_min, edinica);
        full(mnozhitel_max, mnozhitel);
        mnozhitel_max = Summa(mnozhitel_max, mnozhitel);
        int res = Vector_to_value(mnozhitel_max);
        mnozhitel_min = difference(mnozhitel_min, edinica)
;
        int res2 = Vector_to_value(mnozhitel_min);
    }
    result.clear();
    if (Null_div(vector1) || Null_div(vector2))
    {
```

```

        result.push_back('a');
        return;
    }
    for (int i = 0; i < mnozhitel_max.size(); i++)
    {
        result.push_back(Arifmetic[mnozhitel_max[i]]);
    }
    reverse(result.begin(), result.end());
    if (minus) { result.insert(result.begin(), '-'); }
}

```

## 2.12 Функция Division

Вход: введенные числа `vector<int>& vector1` и `vector<int>& vector2`.

Выход: результат операции.

Функция `Division` реализует операцию деления двух векторов целых чисел `vector1` и `vector2`. Сначала она проверяет, не равны ли оба вектора нулю. Если это так, выводится сообщение об ошибке. Если нулевой только `vector2`, выводится сообщение о "пустом множестве". Функция использует цикл, который продолжается, пока значение `res` больше или равно `delimoe`. Внутри цикла вызывается `full` для увеличения счетчика `schet` на единицу, и происходит сложение с использованием `Summa_with_negative`, обновляющее значение `chast`. Затем результат обновляется, и счетчик `schet` увеличивается. Если оба знака `znak1` и `znak2` отрицательны и результат не равен нулю, происходит дополнительное увеличение счетчика и повторное уменьшение `chast`. Векторы `chast` и `schet` переворачиваются, и результат выводится с помощью функции `Print_div`.

Ознакомиться с кодом можно в Листинг 12.

Листинг 12: Функция `Division`

```

void Division(vector<int>& vector1, vector<int>& vector2)
{
    if (Null_div(vector1) && Null_div(vector2))
    {
        cout << "[-ddddddddd;ddddddddd]" << endl;
        return;
    }
    if (Null_div(vector2))
    {
        cout << "пустое_множество" << endl;
        return;
    }
}

```

```

    bool men = false;
    int otv=0;
    int delimoe = Vector_to_value(vector2);
    vector<int> chast = vector1;
    int res = Vector_to_value(chast);
    vector<int> schet(1,0);
    vector<int> edinica(1, 1);
    if (Max_numb(vector1, vector2) == vector2&&znak1==false&&
znak2==true){men = true;}
    while (res >=delimoe)
    {
        full(schet, edinica);
        if (otv == 1 &&res!= Vector_to_value(vector1)){
reverse(schet.begin(), schet.end()); }
        chast = Summa_with_negative(chast, vector2);
        res = Vector_to_value(chast);
        full(schet, edinica);
        schet = Summa(schet, edinica, false);
        otv= Vector_to_value(schet);
    }
    if ((znak1 == true||(znak1&&znak2==true))&&res!=0)
    {
        full(schet, edinica);
        chast = Summa_with_negative(chast, vector2);
        res = Vector_to_value(chast);
        full(schet, edinica);
        schet = Summa(schet, edinica, false);
        otv = Vector_to_value(schet);
    }
    reverse(chast.begin(), chast.end());
    reverse(schet.begin(), schet.end());
    Print_div(men,schet, chast);
}

```

### 3 Результаты работы программы

После запуска программы пользователя просят ввести два числа и выбрать действие – рис. 3.



Рис. 3: Начальное меню

После выбора действия высвечивается результат выбранной операции и можно снова ввести числа – рис. 4.

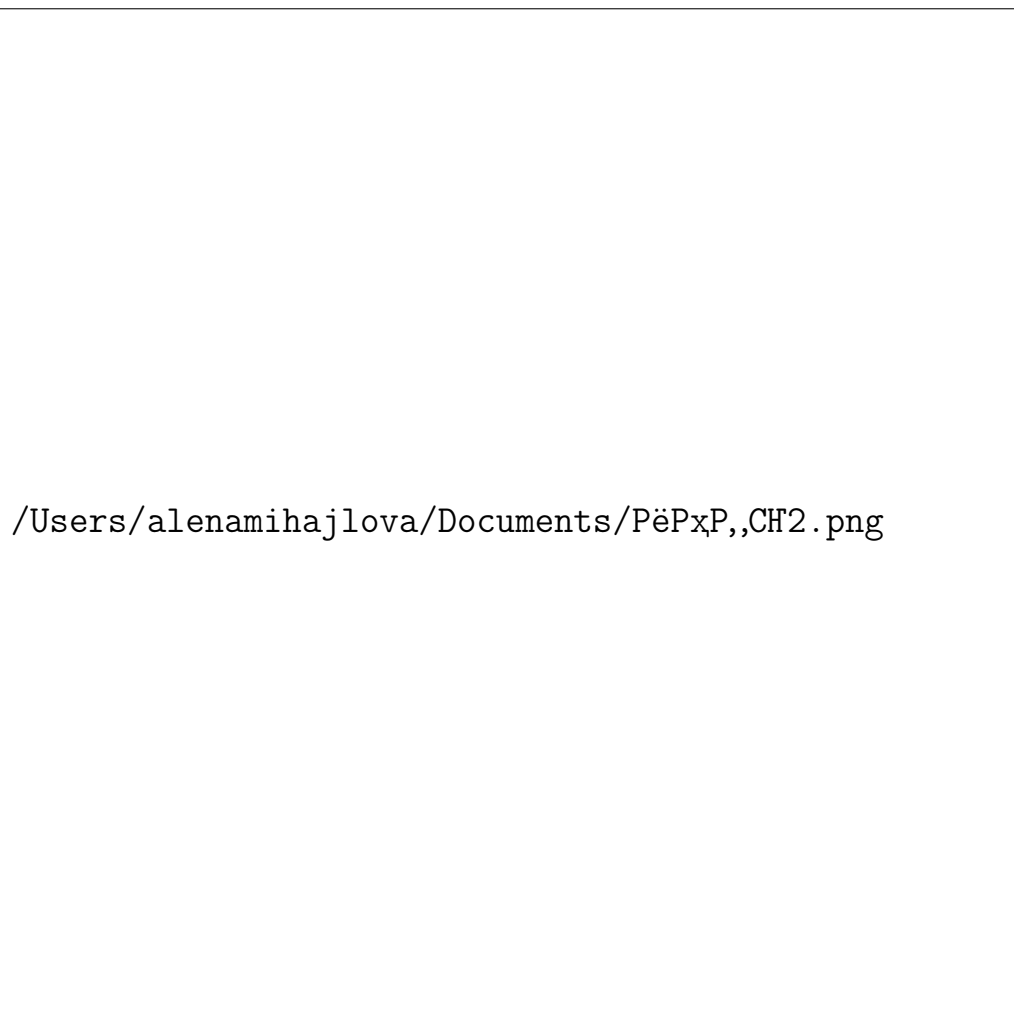
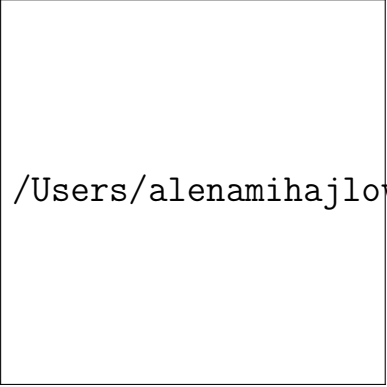


Рис. 4: Результат выбранной операции

Результаты операций сложения, вычитания, умножения, деления – рис.


5.



/Users/alenamihajlova/Documents/PëPxP,,CH8.png

Рис. 5: Результаты операций сложения, вычитания, умножения, деления

В случае деления «а» на «а» ответом является диапазон – рис. 6.



/Users/alenamihajlova/Documents/PëPxP,,CH3.png

Рис. 6: Деление «а» на «а»

В случае деления числа на «а» ответом является пустое множество – рис. 7.

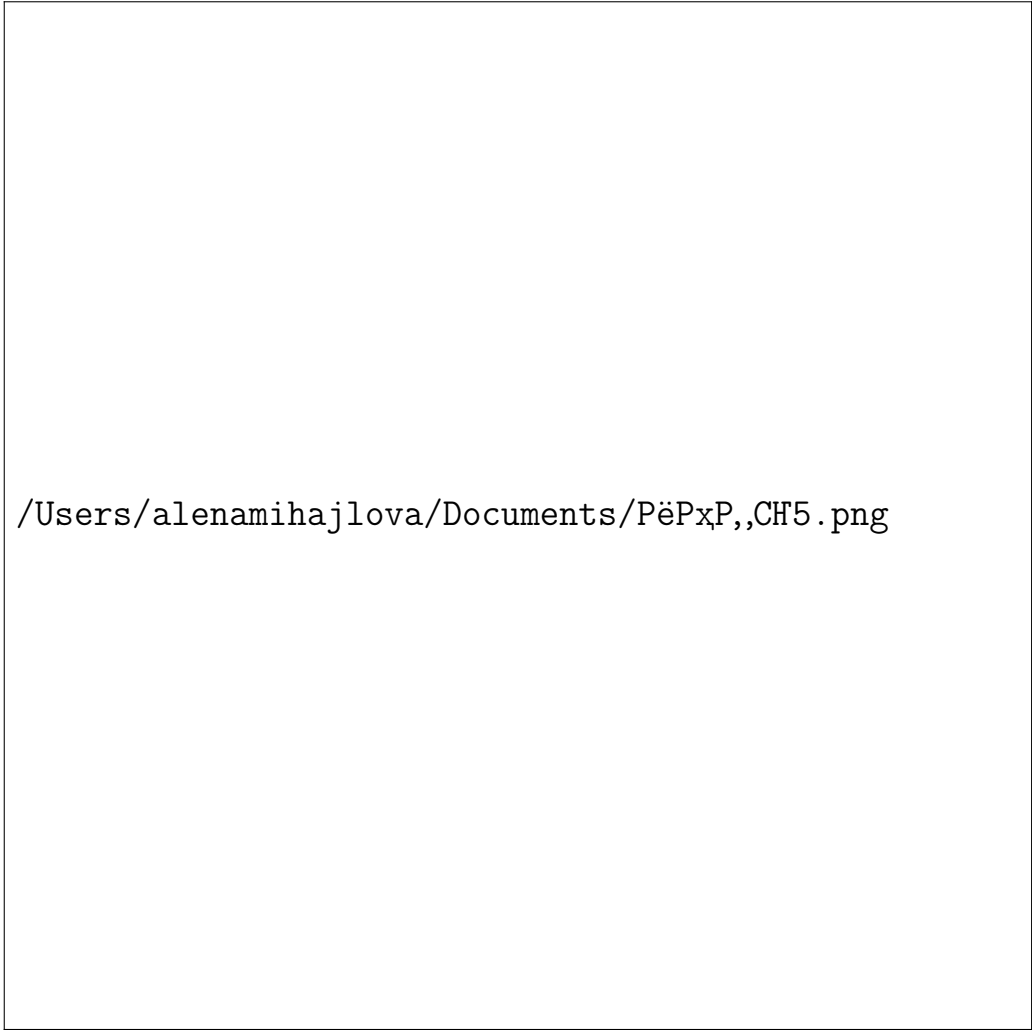


Рис. 7: Деление на «а»

В случае переполнения пользователь получает предупреждение – рис. 8.

При некорректном вводе выводится ошибка – рис. 9, рис 10.





/Users/alenamihajlova/Documents/PëPxP,,CH5.png

Рис. 8: Переполнение

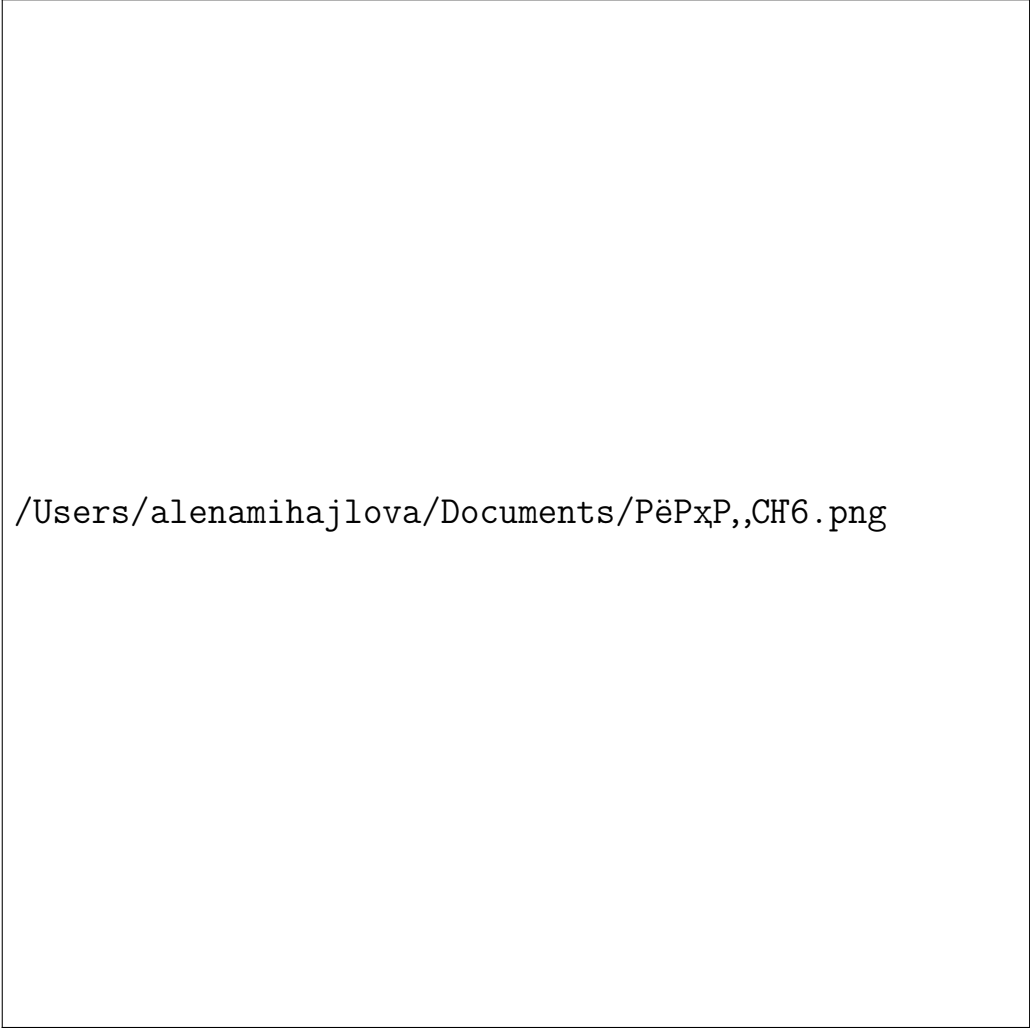
## Заключение

В результате выполнения курсовой работы был реализован калькулятор «большой» конечной арифметики  $\langle Z_8^8; +; * \rangle$  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 любого  $x$ :  $x * a = a$ . Спроектированный калькулятор может выполнять следующие действия: сложение, вычитание, умножение, деление. Программа контролирует некорректный ввод, выходы за границы допустимых значений.

Достоинства программы:

1. Операции вычитания, умножения и деления реализованы при помощи операции сложения;
2. Отсутствие утечек памяти из-за использования контейнера STL vector.

Недостатки программы:



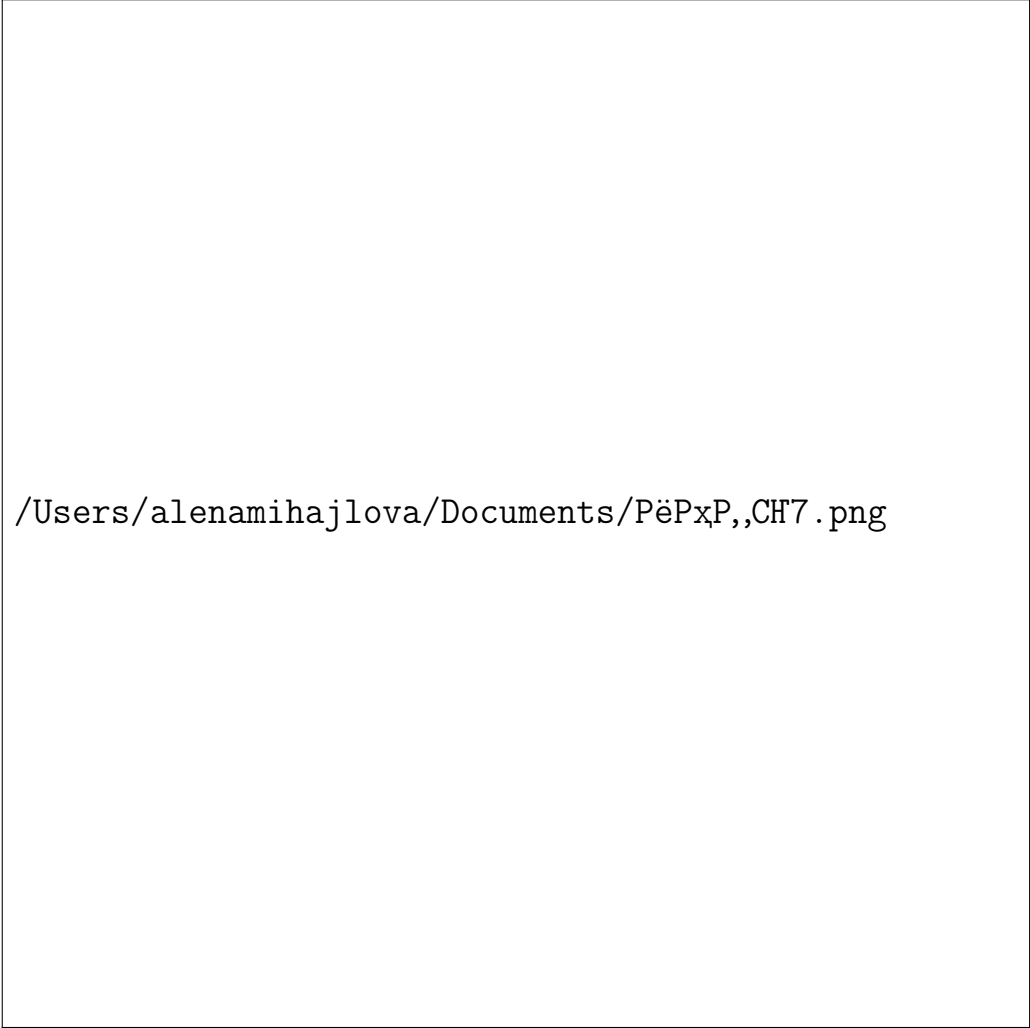
`/Users/alenamihajlova/Documents/PëPxP,,CH6.png`

Рис. 9: Некорректный ввод при вводе чисел

1. Хотя операции вычитания, умножения и деления и реализованы при помощи операции сложения, из-за использования циклов время выполнения операции значительно увеличивается.

Масштабирование:

1. Добавление графического интерфейса;
2. Дополнительно можно реализовать возведение в степень, поиск НОД и НОК для двух чисел;
3. Реализовать сохранение результатов вычисления.



`/Users/alenamihajlova/Documents/PëPxP,,CH7.png`

Рис. 10: Некорректный ввод при выборе операции

## Список литературы

- [1] Секция "Телематика" / текст : электронный / — URL: <https://tema.spbstu.ru/dismath/> (Дата обращения 26.12.2024).
- [2] Кук Д., Бейз Г. КОМПЬЮТЕРНАЯ МАТЕМАТИКА / М.: Наука, 1990 - 384 с.
- [3] Новиков Ф. А. Дискретная математика для программистов. 3-е изд. — Санкт-Петербург: Питер Пресс, 2009. — 384 с.