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 компьютерных технологий и информационных систем

Направление: 09.03.01 Информатика и вычислительная техника

Компьютерная алгебра
Отчет по тематическому заданию
Отношение порядка

Студенты групп
5130901/30102 _____ Первова У. С.
5130901/30201 _____ Ким М. В.
_____ Дубина Д. А.
_____ Косушкин Т. В.
_____ Панасюк С. А.

Старший преподаватель _____ Малышев И. А.

«____»_____ 2025г.

Санкт-Петербург, 2025

Оглавление

Постановка задачи

Разработать библиотеку функций для обработки символьных представлений математического объекта, имеющего заданную алгебраическую структуру.

В качестве математического объекта было выбрано отношение порядка.

Разработанная библиотека должна обеспечивать следующий функционал:

- Создание / удаление объекта;
- (Пере-)определение свойств объекта;
- Копирование объекта;
- Построение выражений, содержащих объекты и их свойства;
- Редукцию (упрощение) выражений, содержащих объекты и их свойства;
- Ввод-вывод данных для символьной обработки

1 Математическое описание

В данной части отчета даны определения используемых понятий и теоретическое описание выполняемых операций.

1.1 Кольцо

Кольцо — это множество *M* с двумя бинарными операциями + и * (они называются сложением и умножением соответственно).

Кольцо называется коммутативным, если *a*b = b*a* умножение коммутативно.

Кольцо называется кольцом *с* единицей, если $\exists 1 \in M$ *(a*1 = 1*a = a)* существует единица, то есть кольцо с единицей — моноид по умножению.

1.2 Свойства операций

1. *(a + b) + c = a + (b + c)* – сложение ассоциативно.
2. *∃0 ∈ M : (∀a (a + 0 = 0 + a = a))* – существует ноль.
3. *∀a (∃(-a) (a + (-a) = 0))* – существует обратный элемент.
4. *a + b = b + a* – сложение коммутативно
5. $(a * b) * c = a * (b * c)$ – умножение ассоциативно, то есть кольцо — полугруппа по умножению.
6. $a * (b + c) = (a * b) + (a * c), (a + b) * c = (a * c) + (b * c)$ – умножение дистрибутивно относительно сложения слева и справа.

1.3 «Малая» конечная арифметика

«Малая» конечная арифметика – конечное коммутативное кольцо с единицей <Zi;+,*>, на котором определены действия вычитания и деления, деление определено частично (так как не каждый элемент может иметь обратное число).

1.4 «Большая» конечная арифметика

«Большая» конечная арифметика – конечное коммутативное кольцо с единицей $<Z_i^n;+,*>$ (где i – количество элементов в арифметике, n – разрядность арифметики), на котором определены действия вычитания и деления (деление определено с остатком).

1.5 Алгебраические структуры

Коммутативное кольцо с единицей – кольцо <Zi;+,*> такое, что умножение коммутативно и существует единица по умножению.

В данной работе «малая» конечная арифметика: рассматриваем конечное кольцо с единицей <Z_8;+,*>.

«Большая» конечная арифметика: разрядность равна 8 (n = 8) и количество элементов в арифметике равно 8 (i = 8):

<Z_8^8;+,*>.

1.6 Отношение порядка

Антисимметричное транзитивное отношение называется *отношением порядка.* Отношение порядка позволяет сравнивать между собой различные элементы одного множества.

Используя правило «+1» построим отношение порядка в «малой» конечной арифметике для данного варианта. Отношение порядка в виде диаграммы Хассе представлено на рисунке 1.

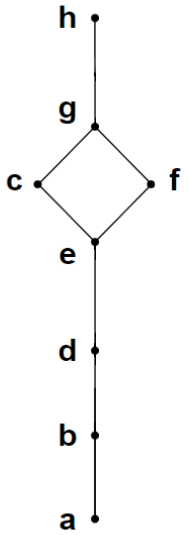

Рис.1. Диаграмма Хассе

a ≺ b ≺ d ≺ e ≺ {c,f} ≺ g ≺ h.

2 Особенности реализации

Данная работа реализует отношение порядка, обеспечивает возможность создавать и редактировать отношение.

Был сформирован калькулятор «большой» конечной арифметики <Zi;+,*> (8 разрядов) для четырех действий (+, –, *, ÷) 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 *(∀x) x*a=a*.

Полученное правило «+1» приведено в таблице 1:

Таблица 1. Правило «+1»

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
+1	*b*	*d*	*g*	*e*	*{c,f}*	*g*	*h*	*a*

Также в данной работе был реализован ряд функций и некоторый консольный интерфейс.

Ниже приведено математическое представление работы символьного калькулятора.

2.1 Таблицы операций «малой» конечной арифметики

Таблицы 2 – 5 представляют собой таблицы сложения, умножения и переносов.

Все операции в данном отношении порядка рассчитываются путем обхода таблиц.

Таблица 2. Сложение

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

Таблица 3. Умножение

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

Таблица 4. Перенос по сложению

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

Таблица 5. Перенос по умножению

$*_p$	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	d	b	b	d	d	e
d	a	a	b	a	a	b	b	b
e	a	a	b	a	b	b	d	d
f	a	a	d	b	b	d	d	e
g	a	a	d	b	d	d	e	c/f
h	a	a	e	b	d	e	c/f	g

Примеры вычислений в «малой» и «большой» конечной арифметиках:

В «малой» конечной арифметике:

1. g + f = c+b+f = c+b+b+e = b+e+b+e = b+b+d+b+b+d = b+b+b+b+b+b+d = d
2. e * e = e+e+e = h+e = d

В «большой» конечной арифметике:

1. f + f = ba + b = bb
2. ed * c = ea*c + d*c = bb + dga = bhb

3 Программная реализация

В качестве инструментального языка программирования был выбран язык C++.

3.1 Описание переменных

- `std::string string`: строка, содержащая последовательность символов "abcdefgh".
- `std::string task = "b d g e c/f g h a"`: вариант задания.
- `std::unordered_map<char, std::string> base`: хранит сформированное отношение порядка по правилу «+1».
- `std::unordered_map<char, std::string> I_base`: инвертированная база для деления.
- `std::vector<std::string> road`: последовательность значений, представляющая диаграмму Хассе.

3.2 Описание основных функций

3.2.1 Функции построения таблиц

1. void generatePlusTable()

   Данный статический метод используется для построения таблицы сложения. За основу берется диаграмма Хассе, таблица заполняется сдвинутой диаграммой в соответствии с выбранным символом. На вход функции подается отношение порядка, хранящееся в переменной `std::vector<std::string> road`. На выходе получаем таблицу сложения `std::vector<std::vector<std::string>> plusTable;` Реализация метода приведена в листинге 1.

Листинг 1. Генерация таблицы сложения

```
void generatePlusTable() {
    for (size_t indx = 0; indx < string.size(); ++indx) {
        char ch = string[indx];
        int shift1 = dist(ch);
        std::vector<std::string> new_road = shift(shift1);

        for (int i = 0; i < size; ++i) {
            plusTable[indx][i] = new_road[roadFind(string[i])];
        }
    }

    for (const auto& el : road) {
        if (el.size() == 3) {
            plusTable[string.find(el[0])][0] = el[0];
            plusTable[string.find(el[2])][0] = el[2];
            plusTable[0][string.find(el[0])] = el[0];
            plusTable[0][string.find(el[2])] = el[2];
        }
    }
}
```

2. void generateMulTable()

Данный статический метод генерирует таблицу умножения по диаграмме Хассе. В таблице учитываются возможные переполнения через метод splitMul(), который разбивает умножение на сложение. На вход функции подаются параметры типа `int i, int j, int available`. На выходе получаем готовую таблицу умножения `std::vector<std::vector<std::string>> mulTable`; Реализация метода приведена в листинге 2.

Листинг 2. Генерация таблицы умножения

```
void generateMulTable() {
    mulTable[0] = std::vector<std::string>(size, string.substr(0, 1));
    mulTable[1] = std::vector<std::string>(string.size());
    for (size_t i = 0; i < string.size(); ++i) {
        mulTable[1][i] = std::string(1, string[i]);
    }

    for (size_t i = 0; i < size; ++i) {
        mulTable[i][0] = string.substr(0, 1);
        mulTable[i][1] = string.substr(i, 1);
    }

    int available = 1;

    for (size_t i = 2; i < size; ++i) {
        for (size_t j = 2; j < size; ++j) {
            answer.clear();
            splitMul(i, j, available);

            std::vector<std::string> arrayAdds;
            for (int u = 0; u < answer.size(); u++) {
                int x = answer[u][0];
                int y = answer[u][1];
                arrayAdds.push_back(mul(x, y));
            }

            mulTable[i][j] = addList(arrayAdds);
        }
    }
}
```

3. void generateMinusTable()

Этот статический метод генерирует таблицу вычитания (минус). В отличие от таблицы сложения, в данном случае значения вычисляются через абсолютные различия между индексами элементов в road. На вход функции подается отношение порядка, хранящееся в переменной `std::vector<std::string> road`. На выходе получаем таблицу вычитания `std::vector<std::vector<std::string>> mulTable;` Реализация метода приведена в листинге 3.

Листинг 3. Генерация таблицы вычитания

```
void generateMinusTable() {
    for (size_t indx = 0; indx < string.size(); ++indx) {
        for (size_t jndx = 0; jndx < string.size(); ++jndx) {
            int dist = abs(roadFind(string[jndx]) - roadFind(string[indx]));
            minusTable[indx][jndx] = road[dist];
        }
    }
    for (const auto& el : road) {
        if (el.size() == 3) {
            minusTable[string.find(el[0])][0] = el[0];
            minusTable[string.find(el[2])][0] = el[2];
            minusTable[0][string.find(el[0])] = el[0];
            minusTable[0][string.find(el[2])] = el[2];
        }
    }
}
```

3.3 Функции для произведения арифметических действий

1. std::string pomogi(const std::string& a, const std::string& b, char sign)

   Это основная функция для обработки и форматирования результатов арифметических действий. На вход которой подаются значения операндов и знак операции `const std::string& a, const std::string& b, char sign`. Функция возвращает переменную типа string, в которой сохраняется результат операции. Реализация функции представлена в листинге 4.

Листинг 4. Функция вызова вычислений из main.cpp.

```
std::string pomogi(const std::string& a, const std::string& b, char sign) {
    std::string answer = calc(a, b, sign);
    answer = replaceF(answer, "c", "c/f");
    if (std::count(answer.begin(), answer.end(), ',') > size-1) {
        return "[ OVERFLOW ]";
    }
    return answer;
}
```

2. std::string calc(const std::string& a, const std::string& b, char sign)

Вычисляет результат в зависимости от переданного знака операции. На вход функции подаются значения операндов и знак операции const std::string& a, const std::string& b, char sign. Возвращаемое значение - переменная типа string, которая хранит результат операции. Реализация функции представлена в листинге 5.

Листинг 5. Калькулирующая функция для вычислений

```
std::string calc(const std::string& a, const std::string& b, char sign) {
        std::string result = "";

        if (sign == '/') {
            auto pair = div(a, b);
            std::string integer = pair.first;
            std::string res = pair.second;

            // NUMBER DIV ZERO

            if (std::count(a.begin(), a.end(), 'a') != a.size() &&
                std::count(b.begin(), b.end(), 'a') == b.size()) {
                return "empty";
            }

            ;
            // ZERO DIV ZERO
            /*
            if (std::count(b.begin(), b.end(), 'a') == 8 &&
                std::count(a.begin(), a.end(), 'a') == 8) {
                return "[ Неопределенность ]";
            }
            */

            if (std::count(integer.begin(), integer.end(), 'h') == 8 &&
                std::count(res.begin(), res.end(), 'h') == 8) {
                return "[-hhhhhhhh; hhhhhhhh]";
            }
            return "(" + formatRes(integer) + ", " + formatRes(res) + ")";
        }

        if (a.empty() || b.empty()) {
            return "";
        }

        std::tuple<std::string, std::string, int, std::string, char> quintet;
        quintet = format(a, b, sign);
        std::string formattedA = std::get<0>(quintet);
        std::string formattedB = std::get<1>(quintet);
        int n = std::get<2>(quintet);
        std::string PorM = std::get<3>(quintet);
        char formattedSign = std::get<4>(quintet);

        bool justMul = false;

        if (formattedSign == '*') {
            std::string tempB = formattedB;
            while (!tempB.empty() && tempB[0] == 'a') {
                tempB = tempB.substr(1);
            }
            if (tempB.size() <= 1) {
                formattedB = std::string(n, formattedB.back());
                justMul = true;
            }
        }

        if (formattedB.find_first_not_of('a') == std::string::npos && (formattedSign
== '+' || formattedSign == '-')) {
            return PorM + replaceF(formattedA, "f", "c");
        }
```

Листинг 5. Калькулирующая функция для вычислений

```
        int overflow = 0;
        std::string overflow_ = "a";

        for (int i = n-1; i >= 0; i--) {
            if (formattedSign == '*' && !justMul) {
                std::string tempResult = calc(formattedA, std::string(1,
formattedB[i]), '*');
                if (tempResult.empty()) continue;

                tempResult = normalFormat(tempResult) + repeat("a", n-i - 1);
                overflow_ = calc(tempResult, normalFormat(overflow_), '+');
                continue;
            }
            int x = string.find(formattedA[i]);
            int y = string.find(formattedB[i]);

            std::string first_number = getTable(formattedSign)[x][y];
            int first_overflow = overflowF(formattedA[i], formattedB[i],
formattedSign);

            char z = road[std::abs(overflow)][0];

            std::string number;
            if (first_overflow == -1) {
                number = road[lenRoad - roadFind(first_number[0])];
            }
            else if (first_overflow >= 0) {
                number = first_number;
            }

            if (overflow >= 0) {
                std::string number_x =
                    getTable('+')[string.find(number[0])][string.find(z)];
                overflow = first_overflow + overflowF(number[0], z, '+');
                number = number_x;
            }
            else {
                if (number == "a" && first_overflow == 0 && overflow < 0) {
                    number = "h";
                    overflow = -1;
                }
                else {
                    number = getTable('-')[string.find(number[0])][string.find(z)];
                    overflow = first_overflow;
                }
            }

            result = number + "," + result;
        }

        if (overflow > 0) {
            result = road[overflow] + "," + result;
        }

        if (overflow_ != "a") {
            return PorM + overflow_;
        }

        while (!result.empty() && result[0] == ',') {
            result = result.substr(1);
        }

        while (result.substr(0, 2) == "a,") {
            result = result.substr(2);
        }

        return PorM + result.substr(0, result.size() - 1);
    }
```

3. std::pair<std::string, std::string> div(const std::string& a, const std::string& b)

Для деления используется специальная логика, обрабатывающая остатки и целую часть через вызов функции div.

Для других арифметических действий происходит рекурсивное вычисление через таблицы значений и комбинирование результатов. форматирует результат, используя функцию replaceF, а также проверяет на переполнение через символы. Реализация функции представлена в листинге 6.

Листинг 6. Функция для деления

```
    std::pair<std::string, std::string> div(const std::string& a, const std::string&
b) {
        bool altA = (a[0] == '-');
        bool altB = (b[0] == '-');
        std::string aCopy = (altA ? a.substr(1) : a);
        std::string bCopy = (altB ? b.substr(1) : b);

        std::tuple<std::string, std::string, std::string> trio;
        trio = comparison(aCopy, bCopy);
        std::string aNorm = std::get<0>(trio);
        std::string bNorm = std::get<1>(trio);
        std::string comp = std::get<2>(trio);

        std::string nAnB = ((altA && altB) ? "-" : "");
        std::string AorB = ((!altA && altB) ? "-" : "");

        std::string res = aNorm;
        std::string count = "a";

        if (std::count(bCopy.begin(), bCopy.end(), 'a') == bCopy.size()) {
            return { std::string(8, road.back()[0]), std::string(8, road.back()[0]) };
        }
        if (std::count(aCopy.begin(), aCopy.end(), 'a') == aCopy.size()) {
            return { "a", "a" };
        }

        if (!altA == altB && comp == ">") {
            while (std::get<2>(comparison(res, bNorm)) != "<") {
                res = minusFunc(res, bNorm);
                count = increment(count);
            }
            if (res.empty()) {
                return { count, "a" };
            }
            return { "-" + count, res};
        }


        if (altA == altB) {
            if (comp == "<") {
                return { "a", nAnB + aNorm };
            }
            else {
                while (std::get<2>(comparison(res, bNorm)) != "<") {
                    res = minusFunc(res, bNorm);
                    count = increment(count);
                }
                if (res.empty()) {
                    return { count, "a" };
                }
                return { count, nAnB + res };
            }
        }
```

Листинг 6. Функция вызова вычислений из main.cpp.

```
        else {
            if (comp == "<") {
                return { "-b", AorB + minusFunc(res, bNorm) };
            }
            else {
                res = aNorm;
                count = "a";
                while (std::get<2>(comparison(res, bNorm)) != "<") {
                    res = minusFunc(res, bNorm);
                    count = increment(count);
                }
                if (res.empty()) {
                    return { "-" + count, "a" };
                }
                res = minusFunc(res, bNorm);
                count = increment(count);
                return { "-" + count, AorB + res };
            }
        }
    }
```

3.4 Функции для работы с отношением порядка

3.4.1 Вставка нового объекта

Данная функция обеспечивает добавление нового объекта в отношение порядка. Новая вершина в графе отношения. Реализация представлена в листинге 7.

Листинг 7. Вставка объекта.

```
void insert_base_clone(std::vector<insert_data> all_data) {
    for (auto data : all_data) {
        insert_vertex_base_clone(data);
    }
}
```

3.4.2 Удаление объекта

Данная функция обеспечивает удаление объекта из отношения порядка. Реализация представлена в листинге 8.

Листинг 8. Удаление объекта

```
size_t deleting_object(std::string obj, bool layer_clear) {
    size_t code = 0;
    std::vector<char> objects_to_del = branch_list(obj);

    if (layer_clear) {
        clearing_all_layer(objects_to_del);
    } else {
        code = clearing_part_of_layer(objects_to_del);
    }
    return code;
}
```

3.4.3 Изменение объекта

Данная функция обеспечивает изменение объекта в отношении порядка. Реализация представлена в листинге 9.

Листинг 9. Изменение объекта

```
    void replace_vertex_base_clone(char object, char parent, std::vector<char>& childs_list,
std::string childs_in_string) {
        if (base_clone.find(parent) != base_clone.end()) {
            std::string childs = base_clone[parent];
            childs = erase_childs(childs, childs_list);
            if (childs == "")
                base_clone[parent] = object;
            else
                base_clone[parent] += "/" + object;
        }
        if (base_clone.find(object) != base_clone.end()) {
            base_clone[object] = base_clone[object].empty() ? childs_in_string :
base_clone[object] + "/" + childs_in_string;
        } else {
            base_clone[object] = childs_in_string;
        }
    }
```

4 Визуальное представление

4.1 Пользовательское меню

Реализация пользовательского меню приведена на рисунке 2.

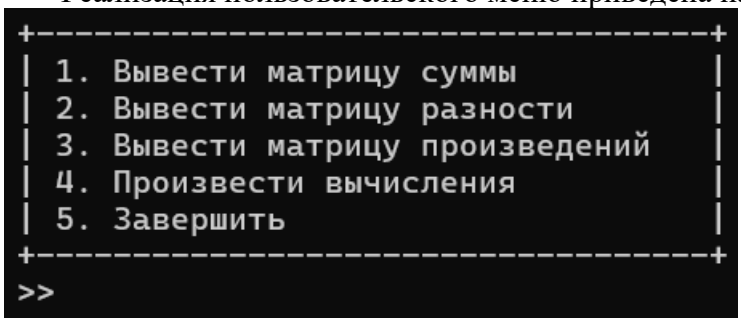

Рис. 2. Пользовательское меню

4.2 Арифметические действия

3.2.1 Сложение

Для выполнения сложения поочередно вводятся оба операнда и знак операции. Далее печатается результат операции. Пример сложения приведен на рисунке 3.

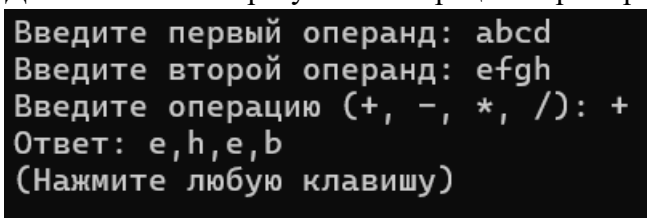

Рис. 3. Пример сложения

3.2.2 Вычитание

Для выполнения вычитания поочередно вводятся оба операнда и знак операции. Далее печатается результат операции. Пример вычитания приведен на рисунке 4.

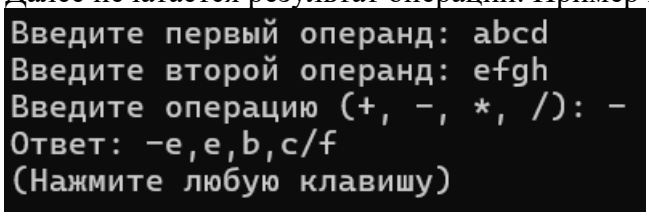

Рис. 4. Пример вычитания

3.2.3 Умножение

Для выполнения умножения поочередно вводятся оба операнда и знак операции. Далее печатается результат операции. Пример умножения приведен на рисунке 5.

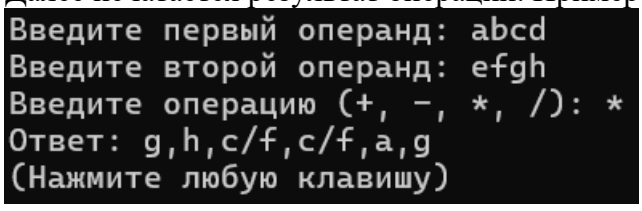

Рис. 5. Пример умножения

3.2.4 Деление

Для выполнения деления поочередно вводятся оба операнда и знак операции. Далее печатается результат операции. Пример деления приведен на рисунке 6.

```
Введите первый операнд: abcd
Введите второй операнд: efgh
Введите операцию (+, -, *, /): /
Ответ: (a, bc/fd)
(Нажмите любую клавишу)
```

Рис. 6. Пример деления

4.3 Обработка переполнений

При сложении и умножении возможно возникновение переполнения и выход за возможный диапазон значений. Пример обработки переполнений приведен на рисунке 7.

```
Введите первый операнд: hhhhhhhh
Введите второй операнд: ff
Введите операцию (+, -, *, /): +
Ответ: [ OVERFLOW ]
(Нажмите любую клавишу)
```

```
Введите первый операнд: hhhhhhhh
Введите второй операнд: df
Введите операцию (+, -, *, /): *
Ответ: [ OVERFLOW ]
(Нажмите любую клавишу)
```

Рис. 7. Обработка переполнений

4.4 Защита от некорректного ввода

Была реализована защита от некорректного ввода, которая выдает предупреждение об ошибочном вводе и ждет правильного ввода. Пример защиты от некорректного ввода приведен на рисунке 8.

```
Введите первый операнд: sss
Неверный операнд: 46986
Неверный операнд: hhhhhhhhh
Неверный операнд:
```

Рис. 8. Некорректный ввод

4.5 Обработка деления на ноль

При делении на ноль возможны несколько исходов. Пример их обработки приведен на рисунке 9.

```
Введите первый операнд: aa
Введите второй операнд: a
Введите операцию (+, -, *, /): /
Ответ: [-hhhhhhhh; hhhhhhhh]
(Нажмите любую клавишу)
```

```
Введите первый операнд: dfg
Введите второй операнд: a
Введите операцию (+, -, *, /): /
Ответ: empty
(Нажмите любую клавишу)
```

```
Введите первый операнд: a
Введите второй операнд: fff
Введите операцию (+, -, *, /): /
Ответ: (a, a)
```

Рис. 9. Обработка деления на ноль

Заключение

Был реализован калькулятор «большой» конечной арифметики $<Z_8^8;+,*>$ для нелинейной диаграммы Хассе (вариант с ветвлением) для четырех действий (+, –, *, ÷) 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.

Реализовано пользовательское меню для удобства взаимодействия с пользователем. Программа выполняет 4 арифметических действия: сложение, умножение, вычитание, деление с числами не выходящими за допустимый диапазон разрядной сетки. Предусмотрена обработка переполнений, деления на ноль и защита от некорректного ввода.

В качестве достоинств программы можно отметить использование объектно-ориентированного подхода, что позволяет сделать код более структурированным и понятным для чтения.

Также реализация всех операций в отдельных методах и функциях позволяет добавлять новый функционал в программу путем малого изменения исходного кода.

Минусами реализации можно отметить использование использование «приватных» переменных и методов класса, требует специальных методов, которые предоставляют к ним доступ, что нагромождает код программы.

В качестве расширения можно реализовать новые операции такие как возведение в степень, поиск НОД и НОК для двух чисел.