

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	4
1.1 Назначение и область применения.....	4
1.2 Описание программы.....	4
1.3 Выбор инструментальных средств.....	4
2 РАЗРАБОТКА ПРОГРАММЫ «Математический интерпретатор»	6
2.1 Реализация программы.....	6
2.2 Тестирование	9
ЗАКЛЮЧЕНИЕ	12
СПИСОК ИСТОЧНИКОВ.....	13
ПРИЛОЖЕНИЕ 1	14

ВВЕДЕНИЕ

На рынке инструментального программного обеспечения представлено множество программных пакетов для работы с математикой. Такими примерами являются matlab, wolfram, octave. Однако для более компактных расчётов, использовать их может быть довольно затруднительно. Для компактных расчётов может подойти калькулятор, однако он не обладает многими возможностями, например, в калькуляторе нельзя задавать переменные. Именно поэтому существует необходимость в компактном математическом интерпретаторе, который был бы прост в освоении и не занимал много места на жёстком диске, в отличие от вышеупомянутых matlab, wolfram и octave.

Целью данной курсовой работы является разработка математического интерпретатора, в парадигме объектно-ориентированного программирования. В ходе выполнения работы следует решить следующие задачи:

- 1) Декомпозиция задачи;
- 2) Построение диаграмм, описывающих приложение;
- 3) Создание приложения, обладающего необходимыми функциями и понятным пользовательским интерфейсом, с применением объектно-ориентированного подхода;
- 4) Тестирование и отладка приложения.

В результате выполнения поставленных целей и задач должно быть создано приложение «Математический интерпретатор».

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Назначение и область применения

Программа «Математический интерпретатор» предназначена для работы с рациональными дробями, векторами и матрицами.

Область применения ограничивается следующим кругом пользователей: программистами, математиками и научными сотрудниками.

1.2 Описание программы

Интерпретатор должен представлять собой консольное приложение, с которым можно взаимодействовать либо непосредственным вводом команд с клавиатуры, либо указать файл с записанными командами.

Функции интерпретатора:

- 1) задание переменных рациональных дробей;
- 2) задание переменных векторов;
- 3) задание переменных матриц;
- 4) вывод значения указанной переменной;
- 5) умножение вектора на рациональную дробь;
- 6) умножение матрицы на вектор;
- 7) транспонирование матрицы;
- 8) считывание команд из файла.

При неправильном вводе команды, интерпретатор должен выводить сообщение об ошибке.

Имя файла с командами должно указываться в качестве аргумента командной строки при запуске интерпретатора.

При считывании команд из файла, исполняемая команда должна отображаться в консоли.

Выход из командного интерпретатора должен осуществляться посредством ввода команды "exit".

1.3 Выбор инструментальных средств

В качестве языка программирования для решения данной задачи был выбран язык C++. Данный язык является языком высокого уровня, который

сохраняет лидерские позиции среди других языков на протяжении уже достаточно долгого промежутка времени. C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений. Также данный язык постоянно обновляется и совершенствуется, что позволяет ему сохранять актуальность.

В качестве среды разработки была выбрана Codeblocks, так как она является бесплатной и включает в себя «из коробки» следующие инструменты:

- 1) компилятор gcc;
- 2) отладчик gdb;
- 3) редактор с подсветкой синтаксиса.

2 РАЗРАБОТКА ПРОГРАММЫ «Математический интерпретатор»

2.1 Реализация программы

Для реализации интерпретатора для начала необходимо описать грамматику. Опишем её с помощью РБНФ (Расширенных форм Бэкуса-Наура), грамматика приведена в таблице 1. Для распознавания правил 3, 5, 6 и 8-10 будут использоваться регулярные выражения.

Таблица 1 – Грамматика

1	<code>expr = var var '=' value var '=' var op var</code>
2	<code>op = '*'</code>
3	<code>var = 'R' number 'V' number 'M' number</code>
4	<code>value = ratioValue vectorValue matrixValue</code>
5	<code>ratioValue = number '/' number</code>
6	<code>vectorValue = '[' vectorItem ']</code>
7	<code>vectorItem = number number ' ' vectorItem</code>
8	<code>matrixValue = '[' matrixList ']</code>
9	<code>matrixList = matrixRow matrixRow ';' matrixList</code>
10	<code>matrixRow = number number ' ' MatrixRow</code>
11	<code>number = digit {digit}</code>
12	<code>digit = '0' ... '9'</code>

Как правило, интерпретатор состоит из следующих компонентов:

- 1) лексический анализатор;
- 2) синтаксический анализатор;
- 3) обработчик команд.

Принцип работы интерпретатора заключается в распознавании введённой команды (лексический анализ), построение синтаксического дерева (синтаксический анализ) и обработки команды в соответствии с семантикой языка.

Укажем семантические правила (Таблица 2). Данные правила уже заложены в работе обработчика команд.

Таблица 2 – Семантические правила

№	Правило
1	Значение переменной рациональной дроби указывается в виде «целое число / целое число»
2	Значение переменной вектора указывается в виде «[число_1 число_2 ... число_n]»
3	Умножение вектора на рациональную дробь указывается в виде «V[номер] * R[номер]»
4	Если указано имя переменной, то вывести значение переменной

Непосредственно в программе были созданы следующие классы:

- 1) Lexer. Лексический анализатор;
- 2) Parser. Синтаксический анализатор;
- 3) Env. Окружение интерпретатора и обработчик команд;
- 4) Token. Токен, содержащий описание лексемы;
- 5) Node. Узел синтаксического дерева;
- 6) VarType. Общий класс типов переменных;
- 7) RationalType. Класс переменной рациональной дроби;
- 8) VectorType. Класс переменной вектора.

Общая схема работы разработанного интерпретатора приведена на рисунке 1.

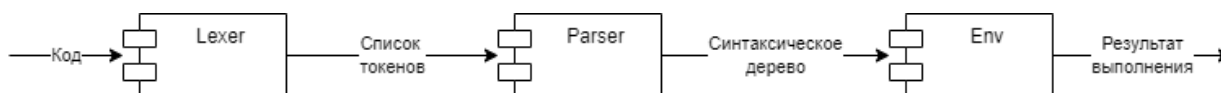


Рисунок 1 – Схема работы интерпретатора

Интерпретатор также принимает аргументы командной строки, а именно если указать флаг “-f” и далее имя файла, то команды из файла будут загружены в память интерпретатора и поочерёдно исполнены, выводя результат каждой из команд.

Построим UML-диаграмму для каждого из компонентов (Рисунок 2).

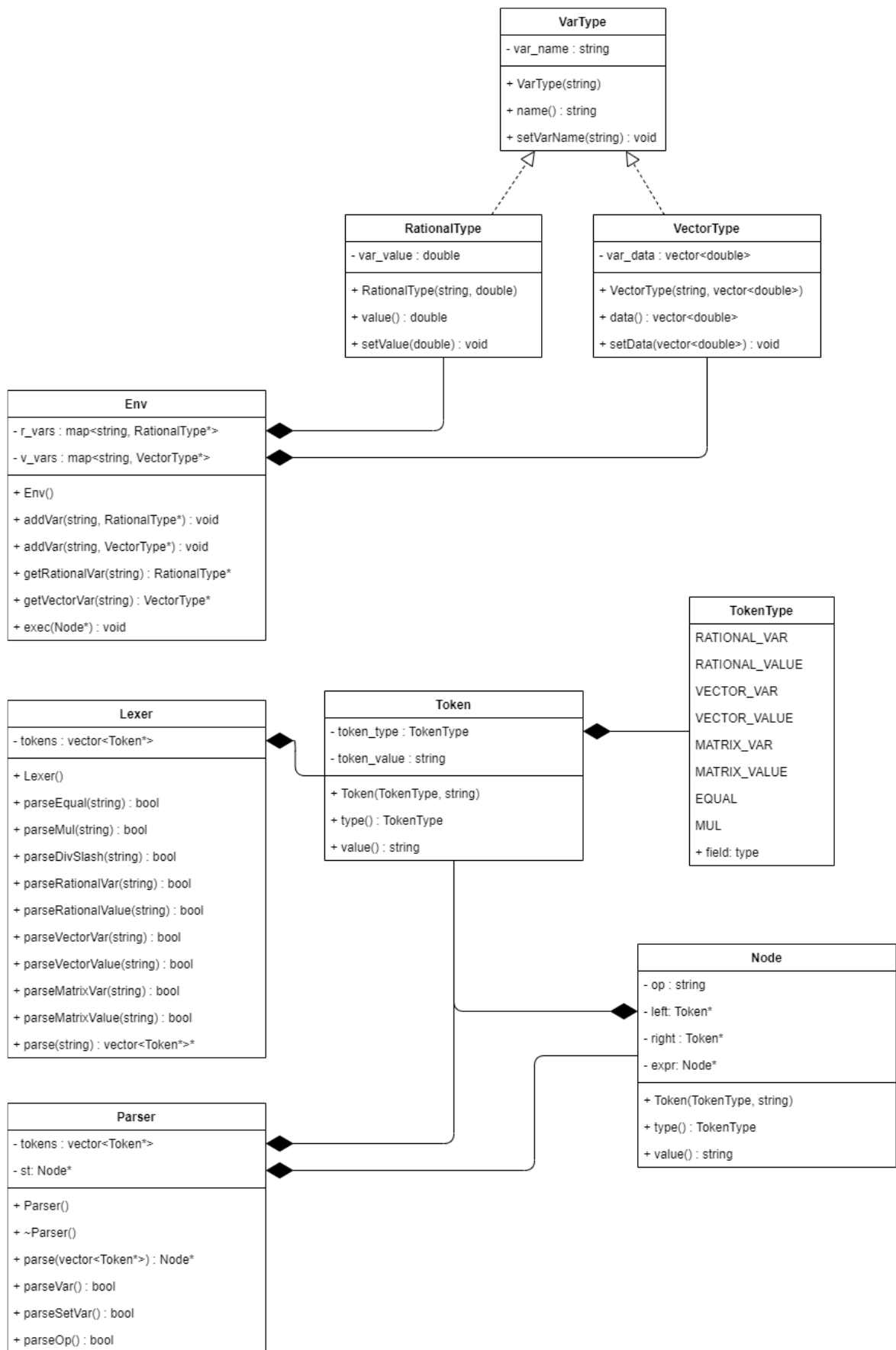


Рисунок 2 – Диаграмма классов

2.2 Тестирование

Продemonстрируем работу интерпретатора. После открытия программы появляется строка ввода команд (Рисунок 3).

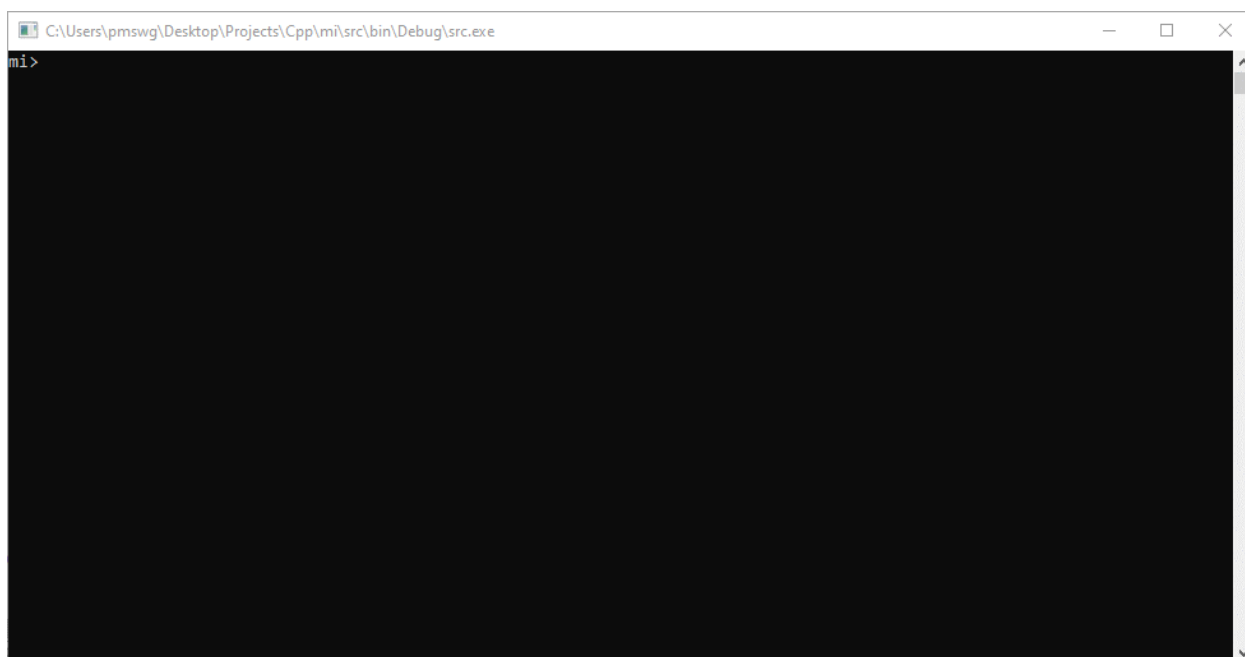


Рисунок 3 – Окно интерпретатора

Зададим переменную R1 и V1, а затем выведем их значения (Рисунок 4).

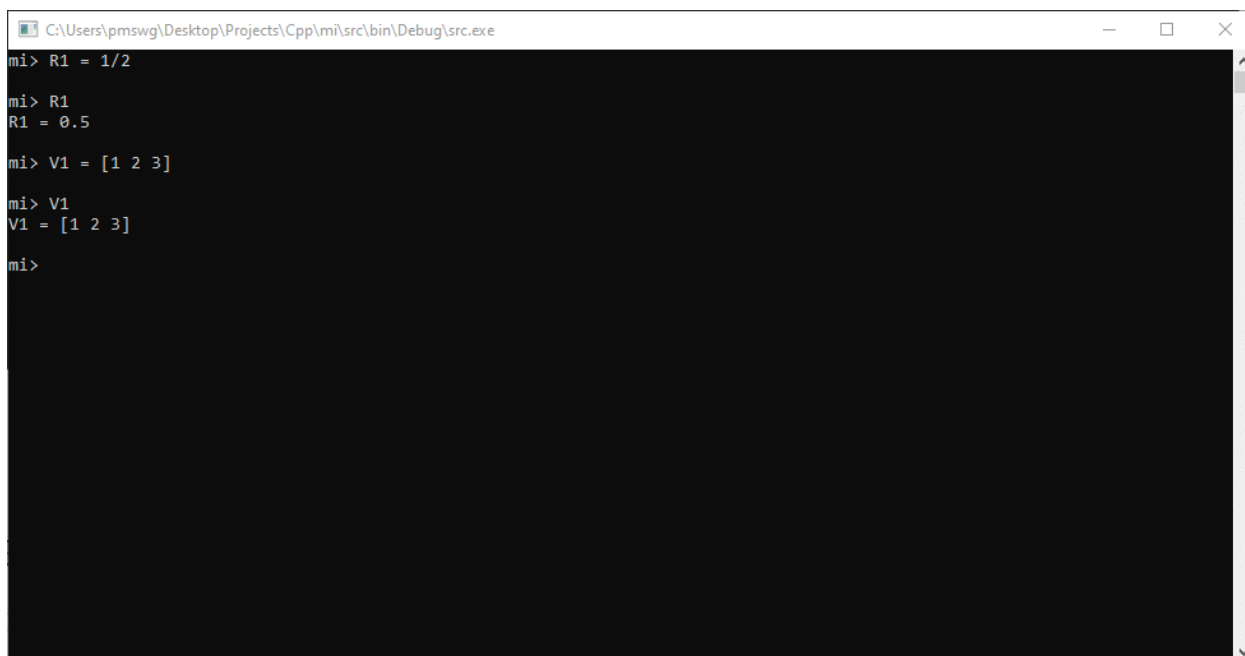


Рисунок 4 – Задание переменных и их вывод

При попытке вывести несуществующую переменную интерпретатор выведет сообщение (Рисунок 5).

```
C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug\src.exe
mi> V1
Variable 'V1' doesn't exist
mi> R1
Variable 'R1' doesn't exist
mi>
```

Рисунок 5 – Попытка вывод значений несуществующих переменных

Приведём ещё один пример работы интерпретатора (Рисунок 6).

```
C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug\src.exe
mi> R1
Variable 'R1' doesn't exist
mi> R1 = 1/5
mi> R1
R1 = 0.2
mi> V1 = [2 3 1]
mi> V1
V1 = [2 3 1]
mi> V1 = V1 * R1
mi> V1
V1 = [0.4 0.6 0.2]
mi> exit

Process returned 0 (0x0)   execution time : 25.453 s
Press any key to continue.
```

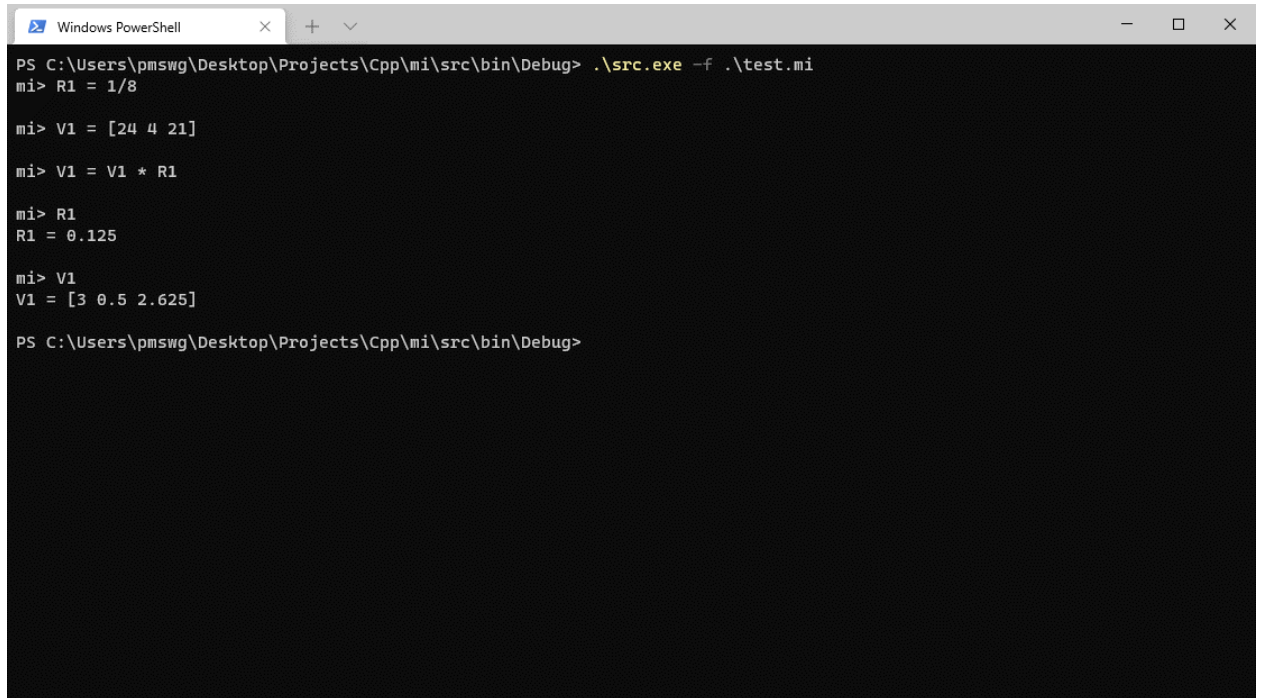
Рисунок 6 – Пример работы

Теперь создадим файл «test.mi» и запишем туда следующие команды (Таблица 3).

Таблица 3 – Команды в файле «test.mi»

```
R1 = 1/8
V1 = [24 4 21]
V1 = V1 * R1
R1
V1
```

После чего перейдём в директорию, в которой находится командный интерпретатор и укажем имя файл в качестве аргумента при запуске интерпретатора (Рисунок 7).



```
Windows PowerShell
PS C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug> .\src.exe -f .\test.mi
mi> R1 = 1/8

mi> V1 = [24 4 21]

mi> V1 = V1 * R1

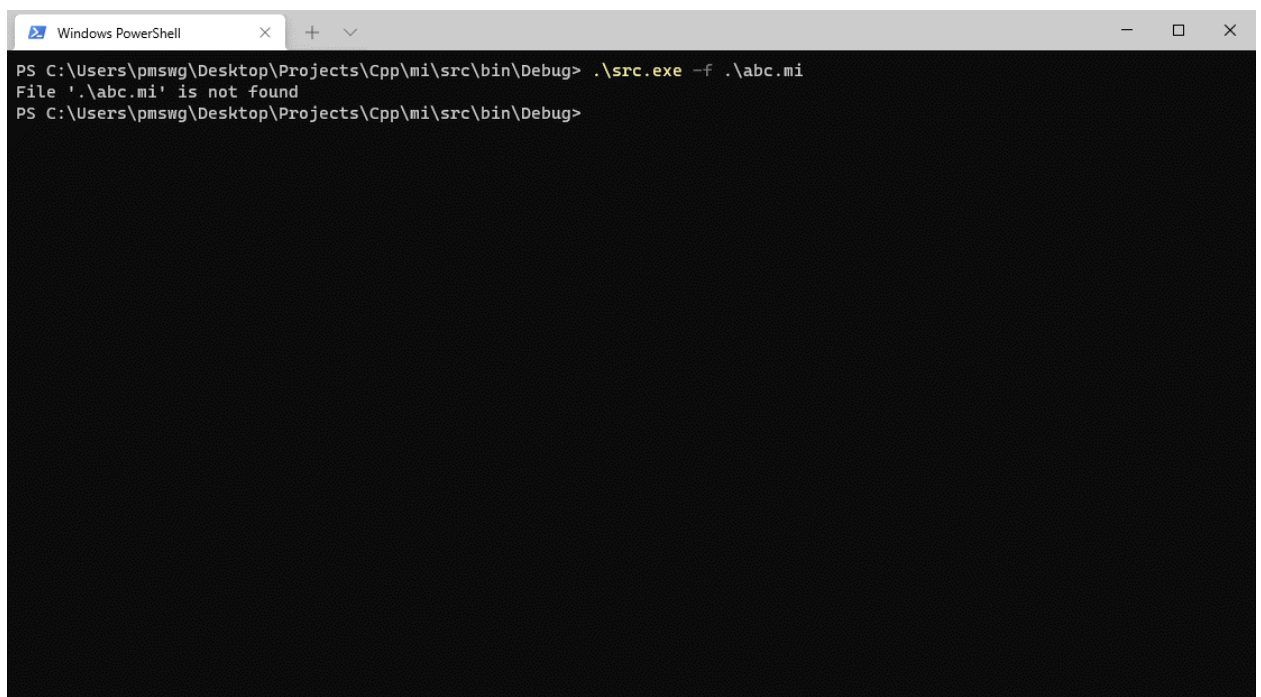
mi> R1
R1 = 0.125

mi> V1
V1 = [3 0.5 2.625]

PS C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug>
```

Рисунок 7 – Выполнение команд из файла

В случае если указан не существующий файл, то интерпретатор выведет сообщение (Рисунок 8).



```
Windows PowerShell
PS C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug> .\src.exe -f .\abc.mi
File '.\abc.mi' is not found
PS C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug>
```

Рисунок 8 – Проверка загрузки файла

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы было создано приложение «Математический интерпретатор», позволяющее производить вычисления с рациональными дробями, векторами и массивами.

В ходе выполнения курсовой работы были решены следующие задачи:

- 1) Была проведена декомпозиция задачи, были выделены основные этапы разработки приложения;
- 2) Были созданы диаграммы, поясняющие внутреннее устройство приложения;
- 3) С применением объектно-ориентированного подхода был написан код программы на языке C++, удовлетворяющей всем поставленным требованиям и обладающей всеми необходимыми функциями;
- 4) Было проведено тестирование и отладка программы, в ходе которой были выявлены и устранены ошибки в приложении. Удалось добиться стабильной и корректной работы приложения.

Функции, которые были реализованы:

- 1) задание переменных рациональных дробей;
- 2) задание переменных векторов;
- 3) вывод значения указанной переменной;
- 4) умножение вектора на рациональную дробь;
- 5) считывание команд из файла.

СПИСОК ИСТОЧНИКОВ

1. Шилдт, Г. С++: Базовый курс, 3-е издание. : Пер. с англ. – М.: Издательский дом «Вильямс», 2011. – 624 с.;
2. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. – 2-е изд., перераб. и доп. – М.: Финансы и статистика, 2006. – 544 с.;
3. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированный анализ и проектирование с примерами приложений. — 3-е издание. — «Вильямс», 2010. – 718 с.
4. How to Approach Writing an Interpreter From Scratch [Электронный ресурс]. – URL: <https://www.toptal.com/scala/writing-an-interpreter>.

ПРИЛОЖЕНИЕ 1

lexer.h

```
#ifndef LEXER_H
#define LEXER_H

#include <string>
#include <iostream>
#include <ctype.h>
#include <vector>
#include <regex>

#include "types/rational_type.h"
#include "token.h"

using namespace std;

class Lexer
{
    TokenList tokens;

public:
    Lexer();

    bool parseEqual(string input);
    bool parseMul(string input);
    bool parseDivSlash(string input);

    bool parseRationalVar(string input);
    bool parseRationalValue(string input);

    bool parseVectorVar(string input);
    bool parseVectorValue(string input);

    bool parseMatrixVar(string input);

    TokenList* parse(string input);
};

#endif // LEXER_H
```

lexer.cpp

```
#include "lexer.h"

Lexer::Lexer()
{
}

bool Lexer::parseEqual(string input)
{
    return input == "=";
}

bool Lexer::parseMul(string input)
{
    return input == "*";
}

bool Lexer::parseDivSlash(string input)
{
    return input == "/";
}

bool Lexer::parseRationalVar(string input)
{
    std::regex _regex("R[0-9]+",
                      std::regex_constants::ECMAScript);

    return std::regex_search(input, _regex);
}

bool Lexer::parseRationalValue(string input)
{
}
```

```

        std::regex _regex("[0-9]+/[0-9]+",
                           std::regex_constants::ECMAScript);

        return std::regex_search(input, _regex);
    }

bool Lexer::parseVectorVar(string input)
{
    std::regex _regex("V[0-9]+",
                      std::regex_constants::ECMAScript);

    return std::regex_search(input, _regex);
}

bool Lexer::parseVectorValue(string input)
{
    std::regex _regex("[ [0-9]+]",
                      std::regex_constants::ECMAScript);

    return std::regex_search(input, _regex);
}

bool Lexer::parseMatrixVar(string input)
{
    std::regex _regex("M[0-9]+",
                      std::regex_constants::ECMAScript);

    return std::regex_search(input, _regex);
}

TokenList* Lexer::parse(string input)
{
    if (!input.empty()) {
        string token = "";

        for (auto c : input) {
            token += c;

            token = std::regex_replace(token, std::regex( "\\s+" ), "");

            if (parseRationalVar(token)) {
                tokens.push_back(new Token(TokenType::RATIONAL_VAR, token));
                token.clear();
            }

            if (parseVectorVar(token)) {
                tokens.push_back(new Token(TokenType::VECTOR_VAR, token));
                token.clear();
            }

            if (parseMatrixVar(token)) {
                tokens.push_back(new Token(TokenType::MATRIX_VAR, token));
                token.clear();
            }

            if (parseEqual(token)) {
                tokens.push_back(new Token(TokenType::EQUAL, token));
                token.clear();
            }

            if (parseRationalValue(token)) {
                tokens.push_back(new Token(TokenType::RATIONAL_VALUE, token));
                token.clear();
            }

            if (parseVectorValue(token)) {
                tokens.push_back(new Token(TokenType::VECTOR_VALUE, token));
                token.clear();
            }

            if (parseMul(token)) {
                tokens.push_back(new Token(TokenType::MUL, token));
                token.clear();
            }
        }

        return &tokens;
    }
}

```

<pre> } return nullptr; } </pre>	
<pre> #ifndef TOKEN_H #define TOKEN_H #include <string> #include <vector> using namespace std; enum TokenType { RATIONAL_VAR, RATIONAL_VALUE, VECTOR_VAR, VECTOR_VALUE, MATRIX_VAR, MATRIX_VALUE, EQUAL, MUL, DIV_SLASH }; class Token { TokenType token_type; string token_value; public: Token(TokenType token_type, string token_value); TokenType type(); string value(); }; using TokenList = vector<Token*>; #endif // TOKEN_H </pre>	token.h
<pre> #include "token.h" Token::Token(TokenType token_type, string token_value) : token_type(token_type), token_value(token_value) { } TokenType Token::type() { return this->token_type; } string Token::value() { return this->token_value; } </pre>	token.cpp
<pre> #ifndef PARSER_H #define PARSER_H #include <iostream> #include "node.h" using namespace std; class Parser { TokenList *tokens; Node *st; } </pre>	parser.h


```

public:
    Parser();
    ~Parser();

    Node* parse(TokenList *tokens);

    bool parseVar();
    bool parseSetValue();
    bool parseOp();

};

#endif // PARSER_H

```

parser.cpp

```

#include "parser.h"

Parser::Parser()
{
    st = new Node();
}

Parser::~~Parser()
{
}

Node* Parser::parse(TokenList *tokens)
{
    if (tokens && !tokens->empty()) {
        this->tokens = tokens;

        parseVar();

        if (tokens->empty()) {
            st->op = "out";
            st->right = nullptr;
        } else {
            if (parseSetValue()) {
                st->op = "assignment";
            } else if (parseOp()) {
                st->op = "assignment";
            }
        }

        return st;
    }
}

bool Parser::parseVar()
{
    Token *t = tokens->back();

    if (t->type() != RATIONAL_VAR && t->type() != VECTOR_VAR) {
        return false;
    }

    st->left = t;

    tokens->pop_back();

    return true;
}

bool Parser::parseSetValue()
{
    Token *t = tokens->back();

    if (t->type() != EQUAL) {
        return false;
    }

    tokens->pop_back();

    t = tokens->back();
}

```

```

        if (t->type() != RATIONAL_VALUE && t->type() != VECTOR_VALUE) {
            return false;
        }

        st->right = t;

        tokens->pop_back();

        return true;
    }

    bool Parser::parseOp()
    {
        Token *t = tokens->back();

        if (t->type() != VECTOR_VAR) {
            return false;
        }

        st->expr = new Node();
        st->expr->left = t;

        tokens->pop_back();

        t = tokens->back();

        if (t->type() != MUL) {
            return false;
        }

        st->expr->op = "multiplication";

        tokens->pop_back();

        t = tokens->back();

        if (t->type() != RATIONAL_VAR) {
            return false;
        }

        st->expr->right = t;

        tokens->pop_back();

        return true;
    }
}

```

node.h

```

#ifndef NODE_H
#define NODE_H

#include <string>

#include "token.h"

using namespace std;

class Node
{
public:
    string op;
    Token *left;
    Token *right;
    Node *expr;

    Node();

    ~Node();
};

#endif // NODE_H

```

node.cpp

```

#include "node.h"

Node::Node() : left(nullptr), right(nullptr), expr(nullptr)
{

```

<pre> } Node::~~Node() { delete expr; } </pre>	
<pre> #ifndef ENV_H #define ENV_H #include <iostream> #include <vector> #include <string> #include <map> #include "types/rational_type.h" #include "types/vector_type.h" #include "parser.h" using namespace std; class Env { map<string, RationalType*> r_vars; map<string, VectorType*> v_vars; bool isRationalVarExists(string var_name); bool isVectorVarExists(string var_name); public: Env(); void addVar(string var_name, RationalType* var_type); void addVar(string var_name, VectorType* var_type); RationalType* getRationalVar(string var_name); VectorType* getVectorVar(string var_name); void exec(Node *st); }; #endif // ENV_H </pre>	env.h
<pre> #include "env.h" Env::Env() { } bool Env::isRationalVarExists(string var_name) { r_vars.find(var_name) != r_vars.end(); } bool Env::isVectorVarExists(string var_name) { v_vars.find(var_name) != v_vars.end(); } void Env::addVar(string var_name, RationalType* var_type) { auto v = r_vars.find(var_name); if (v == r_vars.end()) { r_vars.insert(std::pair<string, RationalType*>(var_name, var_type)); } } void Env::addVar(string var_name, VectorType* var_type) { auto v = v_vars.find(var_name); } </pre>	env.cpp

```

        if (v == v_vars.end()) {
            v_vars.insert(std::pair<string, VectorType*>(var_name, var_type));
        }
    }

RationalType* Env::getRationalVar(string var_name)
{
    if (!var_name.empty()) {
        if (isRationalVarExists(var_name)) {
            return r_vars.at(var_name);
        }
    }

    return nullptr;
}

VectorType* Env::getVectorVar(string var_name)
{
    if (!var_name.empty()) {
        if (isVectorVarExists(var_name)) {
            return v_vars.at(var_name);
        }
    }

    return nullptr;
}

void Env::exec(Node *st)
{
    if (st) {
        if (st->op == "out") {
            if (st->left->type() == RATIONAL_VAR) {
                RationalType *rt = getRationalVar(st->left->value());

                if (rt) {
                    cout << rt->name() << " = " << rt->value() << endl;
                } else {
                    cerr << "Variable '" << st->left->value() << "' doesn't exist" << endl;
                }
            }

            if (st->left->type() == VECTOR_VAR) {
                VectorType *vt = getVectorVar(st->left->value());

                if (vt) {
                    cout << vt->name() << " = " << "[";

                    int i = 0;
                    for (auto v : vt->data()) {
                        cout << v;

                        if (i+1 < vt->data().size()) {
                            cout << " ";
                        }

                        i++;
                    }

                    cout << "]" << endl;
                } else {
                    cerr << "Variable '" << st->left->value() << "' doesn't exist" << endl;
                }
            }
        }

        if (st->op == "assignment") {
            if (st->left->type() == RATIONAL_VAR && !st->expr) {
                double value = 0., a = 0., b = 0.;
                string r_value = st->right->value();
                int n = r_value.find("/");

                a = atof(r_value.substr(0, n).c_str());
                b = atof(r_value.substr(n+1, r_value.size()-1).c_str());

                value = a / b;

                if (!isRationalVarExists(st->left->value())) {
                    RationalType *rt = new RationalType(st->left->value(), value);

```



```

        if (fileFlag == "-f" && !filename.empty()) {
            ifstream file(filename);
            string line;

            if (file.is_open()) {
                while (getline(file, line)) {
                    commands.push_back(line);
                }
            } else {
                cerr << "File '" << filename << "' is not found" << endl;
            }

            file.close();
        }

        if (!commands.empty()) {
            for (auto line : commands) {
                cout << "mi> " << line << endl;

                lst = l.parse(line);

                if (lst) {
                    std::reverse(lst->begin(), lst->end());

                    try
                    {
                        st = p.parse(lst);
                    } catch (string error) {
                        cerr << error;
                    }

                    env.exec(st);
                }

                cout << endl;
            }
        }
    } else {
        string input = "";
        string output = "";

        while (input != "exit") {
            cout << "mi> ";

            getline(cin, input);

            lst = l.parse(input);

            if (lst && !lst->empty()) {
                std::reverse(lst->begin(), lst->end());

                try
                {
                    st = p.parse(lst);
                } catch (string error) {
                    cerr << error;
                }

                env.exec(st);
            } else if (input != "exit") {
                cerr << "Incorrect syntax" << endl;
            }

            cout << endl;
        }
    }

    for (Token *t : *lst) {
        delete t;
    }
}

```

