





## 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 кругом пользователей, а именно 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.

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. С++ 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).

Таблица 1 – Грамматика

<pre>expr = var   var '=' value   var '=' var op var op = '*' var = 'R' number   'V' number   'M' number value = ratioValue   vectorValue   matrixValue ratioValue = number '/' number vectorValue = '[' vectorItem ']' vectorItem = number   number ' ' vectorItem matrixValue = '[' matrixList ']' matrixList = matrixRow   matrixRow ';' matrixList matrixRow = number   number ' ' MatrixRow number = digit {digit} digit = '0'   ...   '9'</pre>
---

Как правило, интерпретатор состоит из следующих компонентов:

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

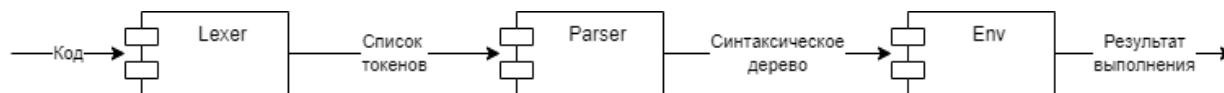
Непосредственно в программе были созданы следующие классы:

- 1) Lexer. Лексический анализатор;
- 2) Parser. Синтаксический анализатор;
- 3) Env. Окружение интерпретатора и обработчик команд;
- 4) Token. Токен, содержащий описание лексем;
- 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 интерпретатора приведена на рисунке \_.



*Рисунок 1 – Схема работы интерпретатора*

Построим UML-диаграмму для каждого из компонентов

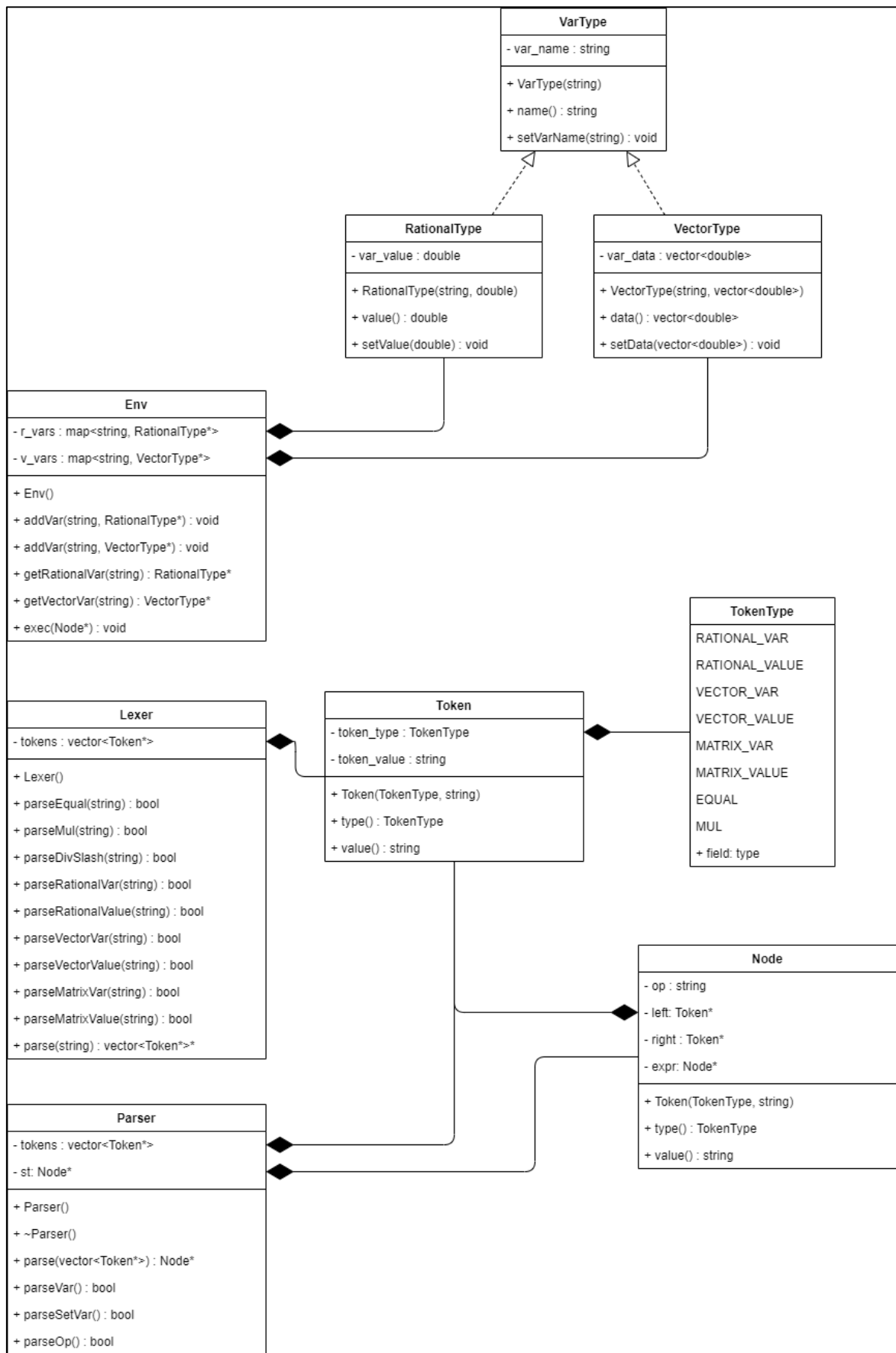
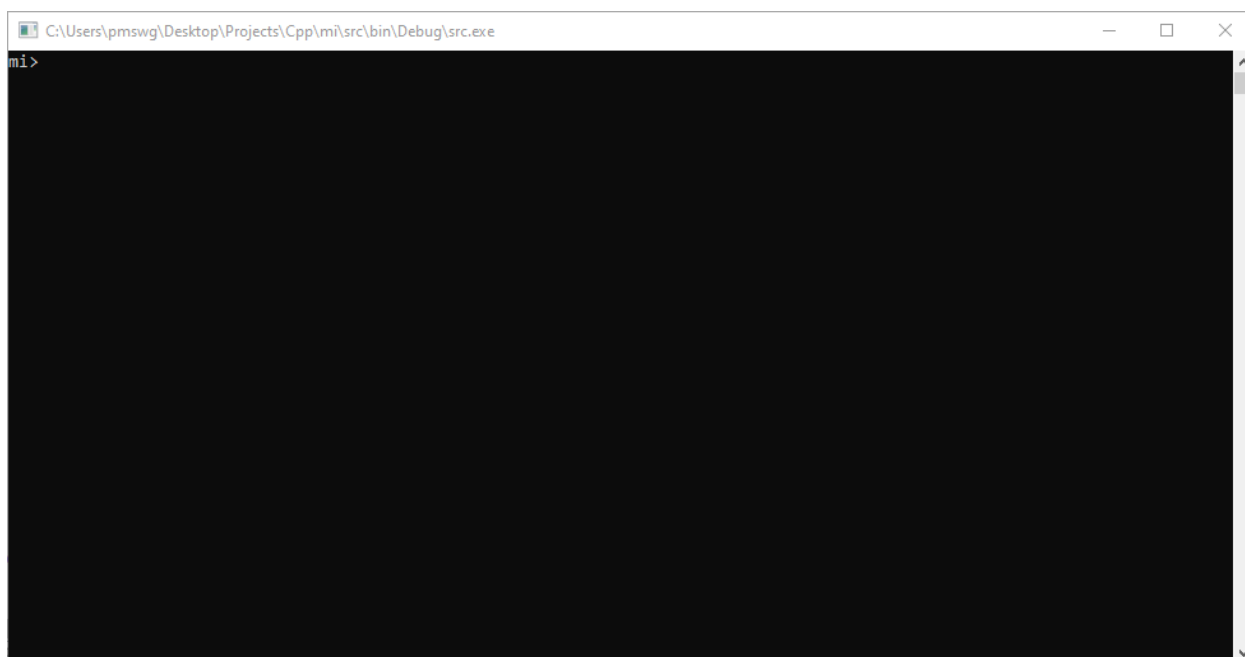


Рисунок 2 – Диаграмма классов



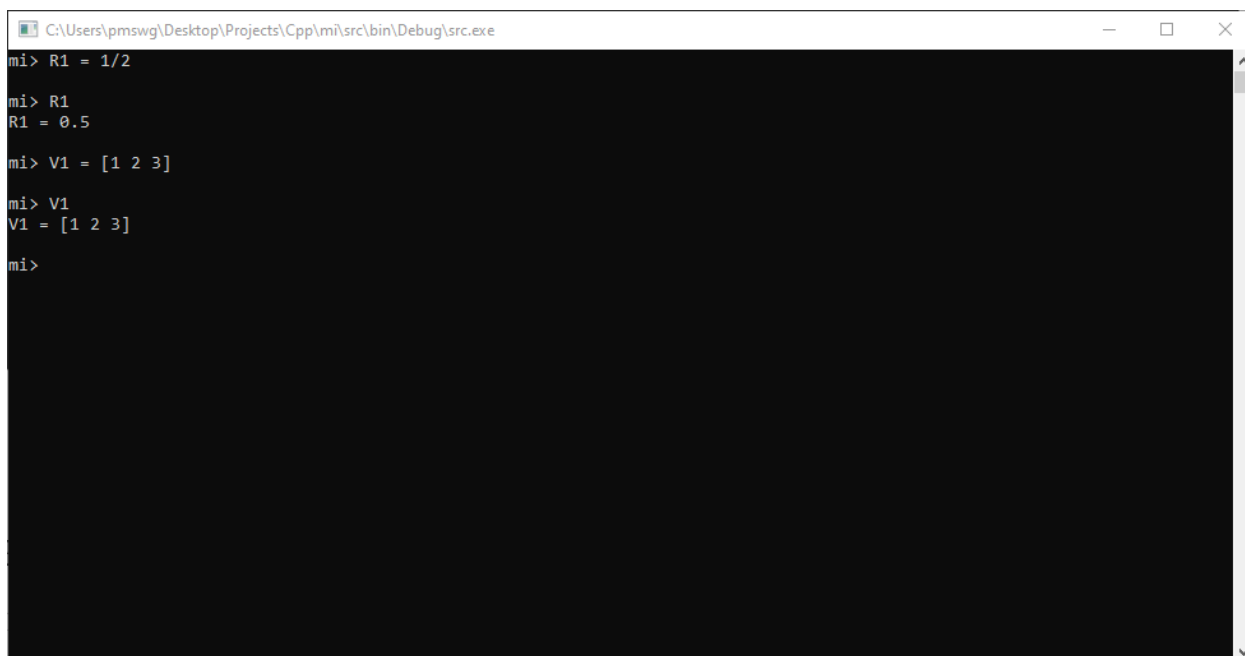
## 2.2 Тестирование

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 ).

Таблица 2 – Команды

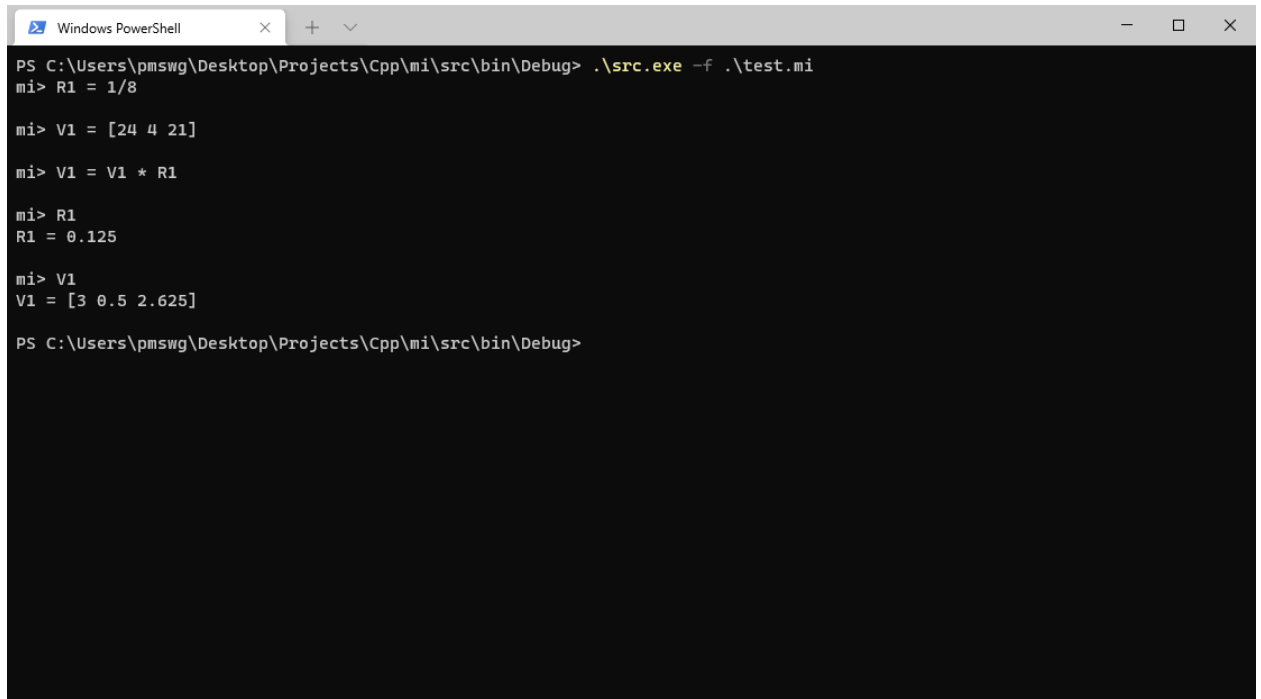
<pre>R1 = 1/8 V1 = [24 4 21]</pre>
------------------------------------

$V1 = V1 * R1$

R1

V1

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



```
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

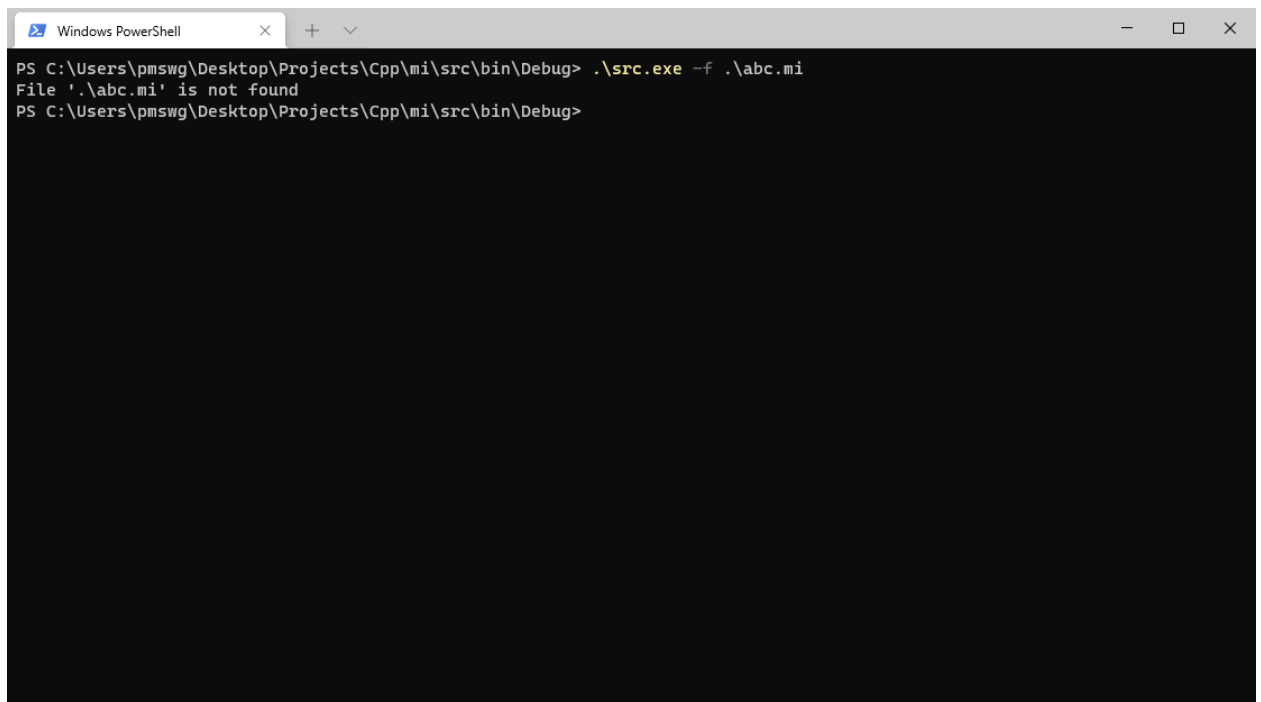
mi> R1
R1 = 0.125

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).



A screenshot of a Windows PowerShell window. The title bar at the top reads "Windows PowerShell" and includes standard window controls (minimize, maximize, close). The command prompt shows the following sequence of text: a prompt "PS C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug>" followed by the command ".\src.exe -f .\abc.mi", then the error message "File '.\abc.mi' is not found", and finally another prompt "PS C:\Users\pmswg\Desktop\Projects\Cpp\mi\src\bin\Debug>". The background of the terminal is black, and the text is white.

```
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.

Итак, все задачи, возникшие при выполнении курсовой работы, были решены, а поставленные цели достигнуты.

## 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.

## ПРИЛОЖЕНИЕ 1

<b>lexer.h</b>
<b>lexer.cpp</b>
<b>token.h</b>
<b>token.cpp</b>
<b>parser.h</b>
<b>parser.cpp</b>
<b>node.h</b>
<b>node.cpp</b>
<b>env.h</b>
<b>env.cpp</b>
<b>main.cpp</b>
<pre>#include &lt;iostream&gt; #include &lt;string&gt; #include "env.h" #include "lexer.h" #include "parser.h" #include &lt;stdlib.h&gt; #include &lt;exception&gt; #include &lt;fstream&gt; using namespace std;  int main(int argc, char **argv) {     Env env;     Lexer l;     Parser p;      TokenList *lst;</pre>

```

Node *st;

vector<string> commands;

if (argc > 0) {
    if (argc == 3) {
        string fileFlag = argv[1];
        string filename = argv[2];

        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 {
                cerr << "Incorrect syntax" << endl;
            }

            cout << endl;
        }
    }
}

```



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