Министерство науки и Высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Курский государственный университет»

Факультет физики, математики, информатики

Кафедра программного обеспечения и администрирования информационных систем

КурсовАЯ РАБОТА

по дисциплине

Теория формальных языков и трансляций

на тему: РАЗРАБОТКА КОМПИЛЯТОРА МОДЕЛЬНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Вариант 16

Обучающегося 3 курса

очной формы обучения

направления подготовки

02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Проектирование информационных систем и баз данных

Козявина Максима Сергеевича

Руководитель:

доцент кафедры АГиТОМ

Селиванова Ирина Васильевна

Допустить к защите:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_г.

Курск, 2023

Содержание

[Введение 3](#_Toc154436562)

[1 Разработка грамматики модельного языка программирования 7](#_Toc154436563)

[2 Лексический анализ 11](#_Toc154436564)

[3 Синтаксический анализ 15](#_Toc154436565)

[4 Семантический анализ 22](#_Toc154436566)

[5 Перевод в польскую инверсную запись 25](#_Toc154436567)

[6 Генерация ассемблерного кода из ПОЛИЗа 28](#_Toc154436568)

[Заключение 41](#_Toc154436569)

[Список источников 42](#_Toc154436570)

[Приложение А 43](#_Toc154436571)

[Текст программы 43](#_Toc154436572)

# Введение

В наши дни, несмотря на огромное количество разработанных языков программирования и соответствующих им компиляторов, процесс создания новых приложений в этой области не замедляется. Это обусловлено как прогрессом в технологии производства компьютерных систем, так и увеличивающимися требованиями к решению сложных задач. Существуют различные причины для разработки нового компилятора, такие как функциональные ограничения, отсутствие локализации или низкая эффективность существующих решений. Именно поэтому основы теории языков и формальных грамматик, а также практические методы разработки компиляторов играют особую роль в инженерном образовании в области информатики и вычислительной техники.

Создание собственного компилятора способствует не только пониманию основ языков программирования, но и дает возможность разработать специфичный язык программирования. Это поможет лучше понять основы информатики и алгоритмизации, а также применить формальную логику, теорию языков программирования и оптимизацию кода. Разработка компилятора также позволит изучить особенности работы с низкоуровневым кодом, углубить знания по работе с памятью и регистрами процессора.

Цель данной курсовой работы включает: получение навыков разработки программы компилятора на основе полученных теоретических знаний в области формальных языков и трансляций, позволяющего получить ассемблерный код и имеющего понятный пользовательский интерфейс, позволяющий визуализировать результаты работы всех этапов компиляции.

В рамках этой цели можно выделить следующие задачи:

* изучение особенностей разработки грамматик модельных языков программирования;
* разработка алгоритмов и методов программной реализации лексического, синтаксического, семантического анализов и перевода в польскую инверсную запись;
* визуализация этапов выполнения лексического, синтаксического и семантического анализов при практическом применении предложенного языка программирования;
* разработка алгоритмов перевода исходного кода в польскую инверсную запись и трансляции на ассемблерный язык;
* компиляция ассемблерного кода в исполняемый файл.

Объектом исследования выступают языки программирования.

Предметом исследования в курсовой работе выступают алгоритмы и методы разработки компиляторов языков программирования и их трансляции.

Работа выполнена на основе следующих структурных особенностей, указанных в варианте №16 задания к курсовой работе:

Выполняется задание согласно варианту 16:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№**  **Варианта** | Задания | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 16 | 3 | 2 | 4 | 1 | 2 | 1 | 4 | 1 | 3 | 1 | 3 | 2 |

1. Структура программы

<программа> ::= {/ (<описание> | <оператор>) ; /}

2. Синтаксис команд описания данных

<описание> ::= dim <идентификатор> {, <идентификатор> } <тип>

3. Синтаксис идентификаторов

<идентификатор> ::= <буква> <буква> <непустая последовательность цифр>

4. Описание типов

(в порядке следования: целый, действительный, логический)

<тип> ::= % | ! | $

<оператор> ::= <составной> | <присваивания> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

5. Синтаксис составного оператора

<составной> ::= «{» <оператор> { ; <оператор> } «}»

6. Оператор присваивания

<присваивание> ::= <идентификатор> ass <выражение>

7. Оператор условного перехода

<условный> ::= if (<выражение>) «{» <оператор> «}»{ elseif (<выражение>) «{» <оператор> «}»} { else «{» <оператор> «}»}

8. Синтаксис оператора

<фиксированного\_цикла> ::= for <присваивания> to <выражение> «{» <оператор> “}”

9. Синтаксис оператора

<условного\_цикла> ::= do while <выражение> «{» <оператор> “}”

10. Синтаксис оператора

<ввода> ::= read (<идентификатор> {, <идентификатор> })

11. Синтаксис оператора

<вывода>::= output (<выражение> { пробел <выражение> })

12. Признак начала комментария Признак конца комментария

/\* \*/

Аннотация: в данной пояснительной записке содержится информация о разработке программы компилятора модельного языка программирования. Приложение включает в себя множество функций, таких как: лексический, синтаксический, семантический анализ и трансляция в польскую инверсную запись. Результатом работы приложения является поиск ошибок в написанном коде и перевод модельного языка в низкоуровневый язык с последующим преобразованием в исполняемый файл.

Ключевые слова: компилятор, язык программирования, лексический анализ, синтаксический анализ, семантический анализ, польская инверсная запись.

Annotation: this explanatory note contains information about the development of a model programming language compiler program. The application includes many functions, such as: lexical, syntactic, semantic analysis and translation into Polish inverse notation. The result of the application is to search for errors in the written code and translate the model language into a low-level language with subsequent conversion into an executable file.

Key words: compiler, programming language, lexical, syntactic, semantic analysis, reverse polish notation.

# **1 Разработка грамматики модельного языка программирования**

**1.1 Форма Бэкуса-Наура**

<программа> ::= {/ (<описание> | <оператор>) ; /}

<описание> ::= dim <идентификатор> {, <идентификатор> } <тип>

<идентификатор> ::= <буква> <буква> <непустая последовательность цифр>

(в порядке следования: целый, действительный, логический)

<тип> ::= % | ! | $

<оператор> ::= <составной> | <присваивания> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

<составной> ::= «{» <оператор> { ; <оператор> } «}»

<присваивание> ::= <идентификатор> ass <выражение>

<условный> ::= if (<выражение>) «{» <оператор> «}»{ elseif (<выражение>) «{» <оператор> «}»} { else «{» <оператор> «}»}

<фиксированного\_цикла> ::= for <присваивания> to <выражение> «{» <оператор> “}”

<условного\_цикла> ::= do while <выражение> «{» <оператор> «}»

<ввода> ::= read (<идентификатор> {, <идентификатор> })

<вывода>::= output (<выражение> { пробел <выражение> })

<выражение> ::= <число> | <идентификатор> | not (<идентификатор>, <выражение>, <булево значение>) | - (<идентификатор>, <выражение>, <число>) | (<идентификатор>, <число>) <знак> (<идентификатор>, <число>)

<число> ::= <непустая последовательность цифр> | <непустая последовательность цифр>.<непустая последовательность цифр>

<булево значение> ::= true | false

<последовательность цифр> ::= {<цифра>}

<непустая последовательность цифр> ::= {/<цифра>/}

<знак> ::= + | - | \* | / | > | < | <= | >= | = | and | or

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<начало\_комментария> ::= </\*>

<конец\_комментария> ::= <\*/>

**1.2 Формальная грамматика**

Грамматика представляет собой набор {T, N, P, S}, где T – множество терминальных символов.

T = { ;, +, -, \*, /, >, >=, <, <=, =, !=, , «,», (, ), {, }, if, else, elseif, for, to, do, while, ass, dim, read, output, and, or, not, ID, TYPE, BOOL, NUM }

N – множество нетерминальных символов.

N = { PROG, CMD, IDS, I, EXPR, ASSIGN, FORCOND, COND1, COND, CALC, ANY }

S – начальный символ из набора нетерминалов.

S = PROG

P – множество правил.

Правила формальной грамматики:

Программа

PROG : CMD

Команды

CMD : EXPR ; CMD | EXPR ; | ASSIGN ; CMD | ASSIGN ;

Список идентификаторов

IDS : IDS , ID | I | I , ID

I : ID

Выражения

EXPR : dim IDS TYPE | read ( IDS ) | output ( IDS ) | if ( COND ) { CMD } | if ( COND ) { CMD } else { CMD } | if ( COND ) { CMD } COND1 | do while ( COND ) { CMD } | for ( FORCOND ) { CMD }

Присвоение

ASSIGN : ID ass CALC | ID ass ANY | ID ass I | ID ass COND

Условие фиксированного цикла

FORCOND : ASSIGN to ANY | ASSIGN to I

Условия конструкций elseif, else

COND1 : elseif ( COND ) { CMD } else { CMD } | elseif ( COND ) { CMD } COND1 | elseif ( COND ) { CMD }

Условие

COND : ANY = ANY | ANY != ANY | ANY < ANY | ANY <= ANY | ANY > ANY | ANY >= ANY | ANY and ANY | ANY or ANY | not ANY | not ID | ANY = ID | ANY != ID | ANY < ID | ANY <= ID | ANY > ID | ANY >= ID | ANY and ID | ANY or ID | ID = ANY | ID != ANY | ID < ANY | ID <= ANY | ID > ANY | ID >= ANY | ID and ANY | ID or ANY | ID = ID | ID != ID | ID < ID | ID <= ID | ID > ID | ID >= ID | ID and ID | ID or ID

Вычисление значения  
CALC : ANY + ANY | ANY - ANY | ANY \* ANY | ANY / ANY | ANY + ID | ANY - ID | ANY \* ID | ANY / ID | ID + ANY | ID - ANY | ID \* ANY | ID / ANY | ID + ID | ID - ID | ID \* ID | ID / ID

Константа числового или булевого типа

ANY : NUM | BOOL

# 2 Лексический анализ

**2.1 Алгоритмы**

Алгоритмы лексического анализа могут основываться на разделении на лексемы с помощью символов разделителей, чтением исходного кода посимвольно или с использованием регулярных выражений. Комментарии могут быть удалены либо же проигнорированы непосредственно при лексическом анализе.

В данном алгоритме используется посимвольное чтение исходного текста и последующая проверка символа или их комбинации на наличие соответствующей лексемы языка. При несоответствии выдаётся ошибка. Во время лексического анализа комментарии игнорируются.

**2.2 Ошибки вывода**

При лексическом анализе может быть выявлено 2 типа ошибок:

Первый тип – неверный идентификатор. Ошибка выдаётся если встречено слово, не являющееся ключевым словом, описанным в словаре WORDS класса Lexer и не соответствующее правилам наименования идентификаторов языка. В этом случае будет выведена ошибка Lexer error: Unknown identifier "[имя идентификатора]" in line [номер строки].

Второй тип – непредвиденный символ. Ошибка выдаётся если встречен символ, не являющийся одним из символов-лексем языка, описанных в словаре SYMBOLS класса Lexer. В этом случае будет выведена ошибка Lexer error: Unexpected symbol "[символ]" in line [номер строки].

**2.3 Хэш-таблица**

Для разбиения исходного кода на лексемы используются встроенные в язык Python хэш-таблицы, именуемые словарями. В множествах и словарях языка Python используется метод открытой адресации. Он заключается в том, что в ячейки таблицы помещаются не указатели на списки, а сами пары ключ-значение. Значение зашифровывается хэш-функцией и при возникновении коллизии пара ключ-значение записываются в следующую пустую ячейку после той, которая получилась в результате работы хэш-функции. При извлечении данных из хэш-таблицы данные дополнительно сверяются с ключом, который так же хранится в хэш-таблице.

Хеш-функцией выступает метод \_\_hash\_\_ определённый в каждом хешируемом объекте языка Python. Эта функция может быть описана для любого типа данных, а для стандартных описана по стандарту. Для простых типов данных, например, int – результатом может быть само число, а для сложных может находиться комбинация хешей для составных частей и генератора случайных чисел, который гарантирует одинаковые значения для одного типа данных в рамках одного запуска программы.

**2.4 Реализация лексического анализа**

Лексический анализ реализует модуль lexer и, в частности, класс Lexer, содержащий хэш-таблицы символов и ключевых слов – SYMBOLS и WORDS. Входными данными для класса является текстовый поток input\_stream из файла текста программы. Метод getc() получает следующий символ из потока input\_stream. Метод set\_error() выставляет сообщение об ошибке. Метод next\_token() считывает следующую лексему посимвольно, определяет тип лексемы, игнорирует комментарии и вызывает метод set\_error() при возникновении ошибки. В результате в поле symbol выводится тип лексемы и в поле value значение для целочисленных, дробных и булевых литералов или имя идентификатора.

**2.5 Тестирование**

Найти значение функции y = kx+b

Тест 1. Программа написана без ошибок

Исходный код программы на модульном языке программирования:

dim xx1, yy1, kk1, bb1, rr1 !;

rr1 ass xx1 \* kk1;

rr1 ass rr1 + bb1;

yy1 ass rr1;

Результат работы лексического анализа представлен на рисунке 1.

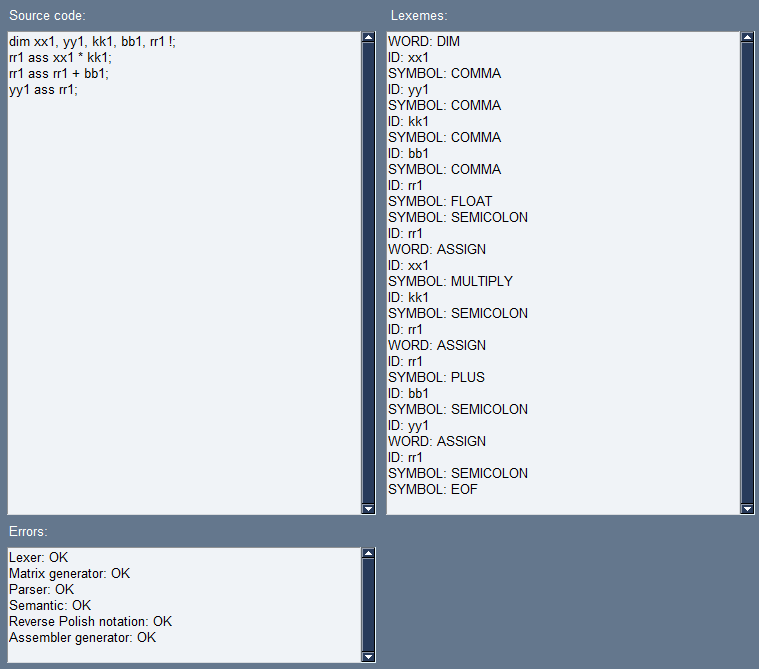


Рисунок 1 – Тест 1 алгоритма лексического анализа

Тест 1. Программа написана с лексической ошибкой, не существует оператора или идентификатора as.

Исходный код программы на модульном языке программирования:

dim xx1, yy1, kk1, bb1, rr1 !;

rr1 as xx1 \* kk1;

rr1 ass rr1 + bb1;

yy1 ass rr1;

Результат работы лексического анализа представлен на рисунке 2.

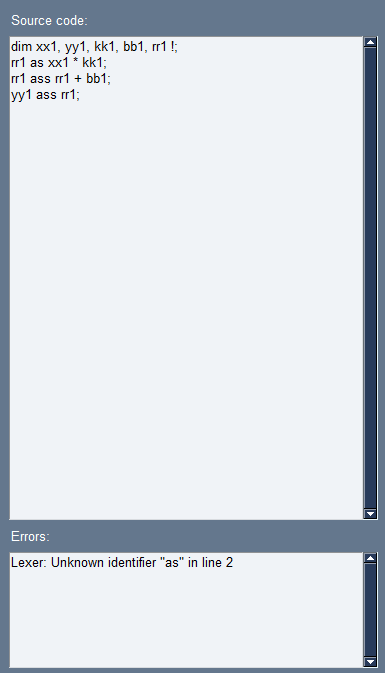


Рисунок 2 – Тест 2 алгоритма лексического анализа

# 3 Синтаксический анализ

**3.1 Алгоритмы**

Существует несколько алгоритмов синтаксического анализа:

Алгоритм рекурсивного спуска – алгоритм нисходящего синтаксического анализа, реализуемый путём взаимного вызова процедур парсинга, где каждая процедура соответствует одному из правил контекстно-свободной грамматики или БНФ. Применения правил последовательно, слева-направо поглощают токены, полученные от лексического анализатора. Это один из самых простых алгоритмов парсинга, который является также малоэффективным т.к. может обрабатывать большое количество текста недостаточно быстро и может обрабатывать не все грамматики.

Алгоритм сдвиг-свёртки используется в данном алгоритме. Он используется для грамматики операторного предшествования. Для моделирования его работы необходима входная цепочка символов и стек символов, в котором автомат может обращаться не только к самому верхнему символу, но и к некоторой цепочке символов на вершине стека. Также необходимо построить матрицу операторного предшествования.

Этот алгоритм для заданной КС-грамматики можно описать следующим образом:

1) Поместить в верхушку стека символ «начало строки», считывающую головку МП-автомата поместить в начало входной цепочки. В конец входной цепочки надо дописать символ «конец строки».

2) В стеке ищется самый верхний терминальный символ sj при этом сам символ sj остается в стеке. Из входной цепочки берется текущий символ ai (справа от считывающей головки МП-автомата).

3) Если символ sj – это символ начала строки, а символ ai – символ конца строки, то алгоритм завершен, входная цепочка символов разобрана.

4) В матрице предшествования ищется клетка на пересечении строки, помеченной символом sj, и столбца, помеченного символом ai.

5) Если клетка, пустая, то значит, входная строка символов не принимается, алгоритм прерывается и выдает сообщение об ошибке.

6) Если клетка, содержит символ “=.” или “<.” то необходимо выполнить перенос. При выполнении переноса текущий входной символ ai помещается на верхушку стека, считывающая головка сдвигается на одну позицию вправо. После этого надо вернуться к шагу 2.

7) Если клетка, содержит символ “.>”, то необходимо произвести свертку. Для выполнения свертки из стека выбираются все терминальные символы, связанные отношением “=.”, начиная от вершины стека, а также все нетерминальные символы, лежащие в стеке рядом с ними. Эти символы вынимаются из стека и собираются в цепочку.

8) Во всем множестве правил грамматики ищется правило, у которого правая часть совпадает с цепочкой. Если правило найдено, то в стек помещается нетерминальный символ из левой части правила, иначе, если правило не найдено, это значит, что входная строка символов не принимается, алгоритм прерывается и выдает сообщение об ошибке. После выполнения свертки необходимо вернуться к шагу 2.

В данном алгоритме матрица операторного предшествования строится автоматически из правил грамматики перед выполнением алгоритма сдвиг-свёртки.

**3.2 Ошибки вывода**

При построении матрицы операторного предшествования может быть выведены ошибки:

1. при встрече неизвестного символа - unknown symbol "[символ]";
2. при обнаружении неверно построенного файла правил вывода - wrong file formatting;
3. при конфликте в правилах вывода с указанием ошибки;

При выполнении алгоритма сдвиг-свёртки:

1. при отсутствии правила в исходных правилах вывода - Unable to locate rule [правило];
2. при отсутствии связи в таблице операторного предшествования - unknown construction [символ 1] [символ 2] in [фрагмент кода].

**3.3 Реализация синтаксического анализа**

Для построения матрицы операторного предшествования в класс Matrix передаётся текстовый поток input\_stream из текстового файла, в котором описаны все терминальные символы, все имена правил и перечислены все правила вывода причём имя и само правила разделяется символом «:». Все терминальные символы должны быть разделены пробелом. Метод print\_matrix() позволяет вывести результат на экран а Метод generate() генерирует матрицу операторного предшествования. Строятся множества крайних левых и крайних правых символов для всех символов и только для терминальных символов. После по этим множествам составляется матрица операторного предшествования.

На рисунке 3 представлена полученная матрица операторного предшествования.

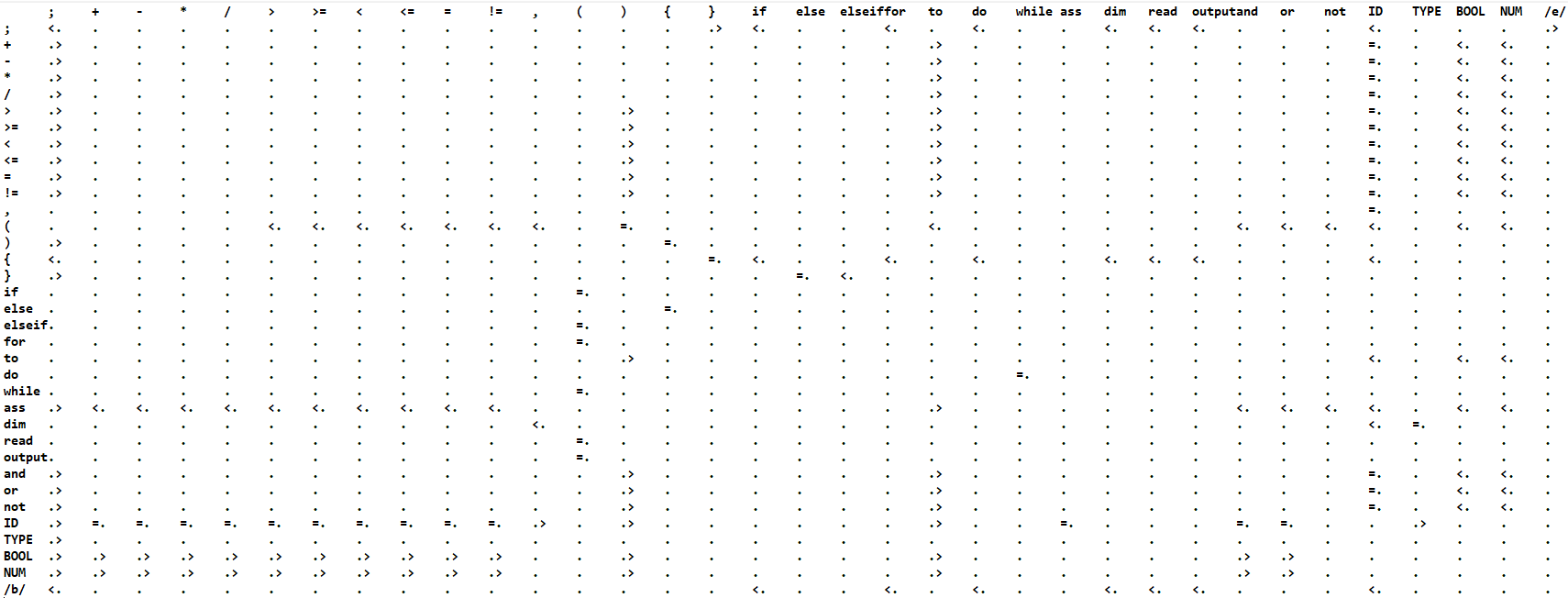


Рисунок 3 – Матрица операторного предшествования

Эта матрица передаётся для выполнения алгоритма сдвиг-свёртки в класс Parser. Перед выполнением сдвиг-свёртки все нетерминалы в правилах вывода заменяются на один нетерминал «Е». При вызове метода check выполняется проверка лексем на корректность по матрице операторного предшествования. Метод get\_t возвращает крайний терминал стека, а при передаче аргумента get\_t(n) может вернуть n-ый символ с конца стека. Для вывода ошибок используется метод set\_error.

**3.4 Тестирование**

Проверить, что треугольник со сторонами a, b, c существует.

Тест 1. Программа написана без ошибок

Исходный код программы на модульном языке программирования:

dim aa1, bb1, cc1, ss1 !;

dim rr1 $;

rr1 ass true;

read(aa1, bb1, cc1);

ss1 ass aa1 + bb1;

if (ss1 < cc1) {

rr1 ass false;

};

ss1 ass aa1 + cc1;

if (ss1 < bb1) {

rr1 ass false;

};

ss1 ass bb1 + cc1;

if (ss1 < aa1) {

rr1 ass false;

};

output(rr1);

Результат работы синтаксического анализа представлен на рисунке 4.

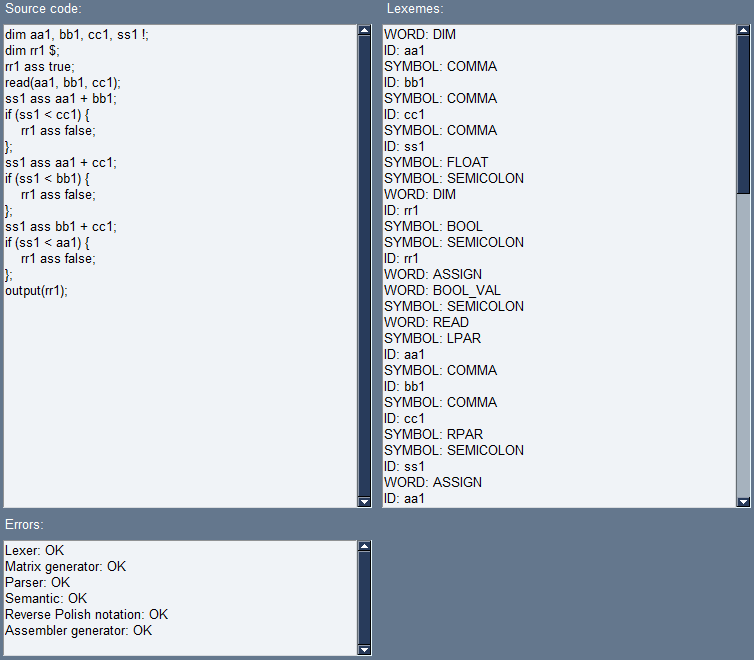
****

Рисунок 4 – Тест 1 алгоритма синтаксического анализа

Тест 2. Программа написана с синтаксической ошибкой, elseif не может идти после «;».

dim aa1, bb1, cc1, ss1 !;

dim rr1 $;

rr1 ass true;

ss1 ass aa1 + bb1;

elseif (ss1 < cc1) {

rr1 ass false;

};

...

Результат работы синтаксического анализа представлен на рисунке 5.

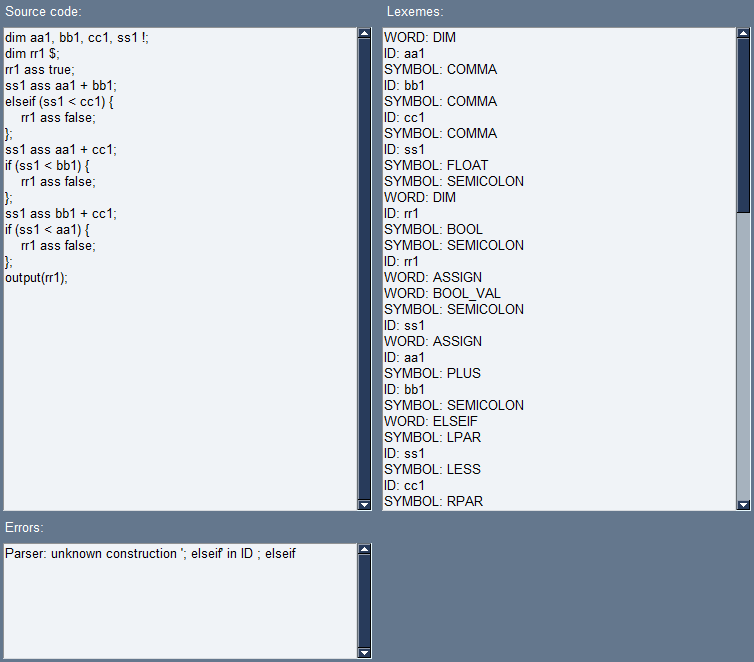


Рисунок 5 – Тест 2 алгоритма синтаксического анализа

Тест 3. Программа написана с синтаксической ошибкой, не существует правила грамматики для фрагмента кода ss1 < cc1 = 1

dim aa1, bb1, cc1, ss1 !;

dim rr1 $;

rr1 ass true;

ss1 ass aa1 + bb1;

if (ss1 < cc1 = 1) {

rr1 ass false;

};

...

Результат работы синтаксического анализа представлен на рисунке 6.

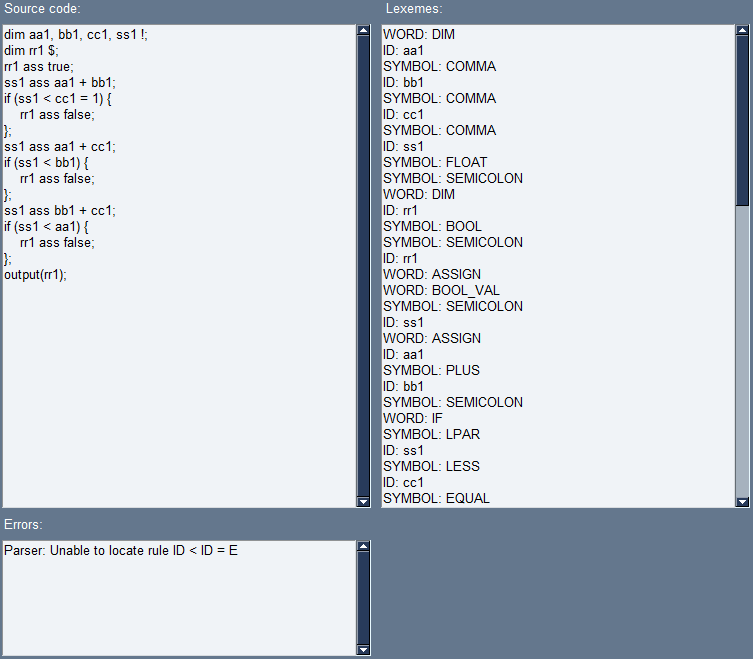


Рисунок 6 – Тест 3 алгоритма синтаксического анализа

# 4 Семантический анализ

**4.1 Алгоритмы**

Для семантического анализа используется простой алгоритм поиска в лексемах определённых закономерностей и проверка правильности типов и определения переменных в этих закономерностях.

**4.2 Ошибки вывода**

При попытке обратится к переменной, которая не была определена до этого будет выведена ошибка Undeclared variable [имя переменной].

При попытке присвоить переменной литерал несоответствующего типа или значение переменной несоответствующего типа будет выведена ошибка Wrong type: [имя переменной] is [тип переменной] unable to assign [тип, который пытался присвоить пользователь].

**4.3 Реализация семантического анализа**

Метод check проверяет входную цепочку лексем на соответствие типов и ведёт учёт переменных, определённых в исходном коде. Считывание происходит поэлементно. Алгоритм ищет цепочку вида dim [имена переменных] [тип] для занесения переменной в список определённых, цепочки вида [переменная] ass [переменная], [переменная] ass [литерал] или [переменная] ass [переменная или литерал] [операция] [переменная или литерал] для контроля соответствия типов. Любые другие вхождения переменных будут сопоставляться со списком определённых в программе переменных. В классе определены поля-массивы операций RETURN\_BOOL и RETURN\_NUM, которые определяют какой тип возвращают соответствующие операции. Для вывода ошибок используется метод set\_error.

**4.4 Тестирование**

Рассчитать сумму чётных чисел от 2 до n.

Тест 1. Программа написана без ошибок

Исходный код программы на модульном языке программирования:

dim ii1, ss1, tm1, nn1 !;

read(nn1);

nn1 ass nn1/2;

nn1 ass nn1 + 1;

for (ii1 ass 1 to nn1) {

tm1 ass ii1 \* 2;

ss1 ass ss1 + tm1;

};

Результат работы алгоритма семантического анализа представлен на рисунке 7.

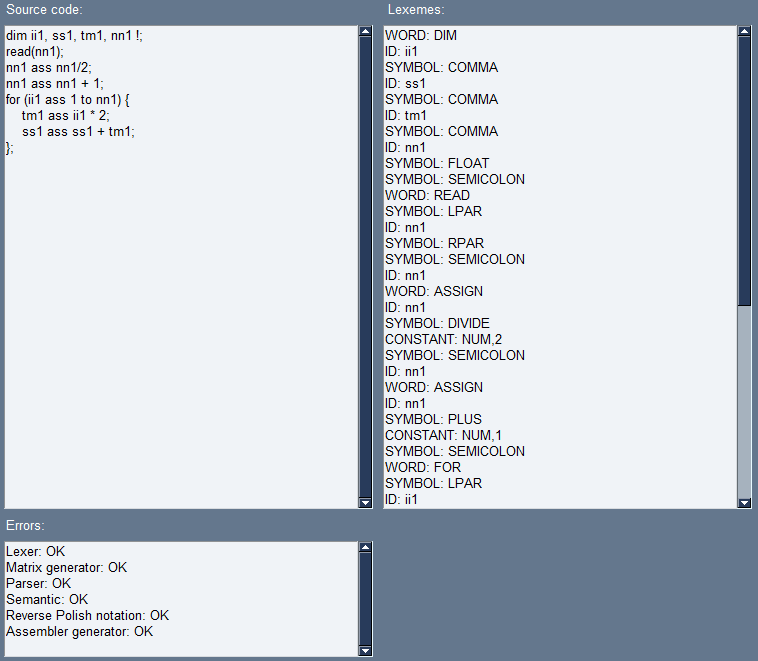
****

Рисунок 7 – Тест 1 алгоритма семантического анализа

Тест 1. Программа написана с семантической ошибкой, невозможно присвоить переменной типа FLOAT константу типа BOOL

Исходный код программы на модульном языке программирования:

dim ii1, ss1, tm1, nn1 !;

read(nn1);

nn1 ass nn1/2;

nn1 ass nn1 + 1;

for (ii1 ass 1 to nn1) {

tm1 ass true;

ss1 ass ss1 + tm1;

};

Результат работы алгоритма семантического анализа представлен на рисунке 8.

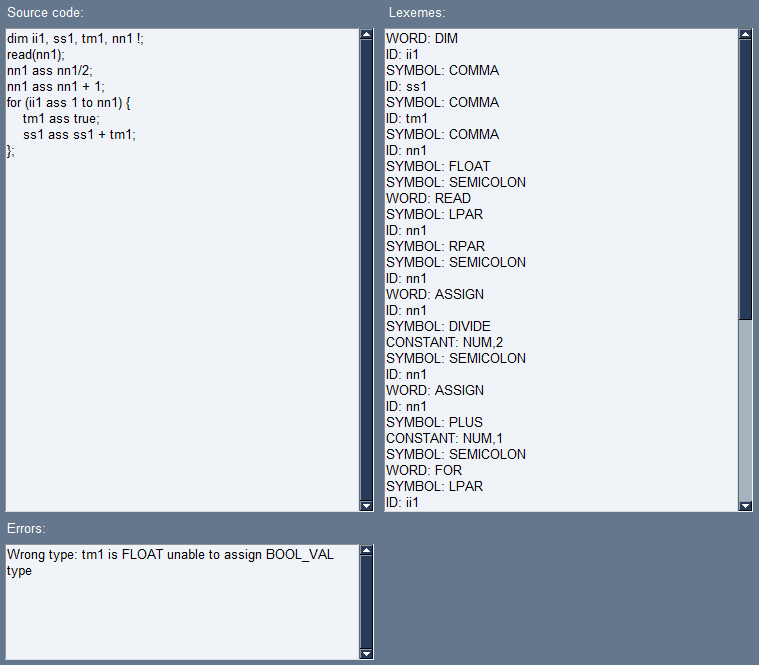
****

Рисунок 8 – Тест 2 алгоритма семантического анализа

# 5 Перевод в польскую инверсную запись

**5.1 Алгоритм**

Для перевода в польскую инверсную запись используется алгоритм Замельсона и Бауэра который заключается в предварительном составлении таблицы приоритетов для каждого оператора языка. В такой таблице хранятся стековый и магазинный приоритеты. Код читается слева направо и каждый элемент добавляется либо в результат, либо в магазин. Идентификаторы и константы переписываются в выходную строку ПОЛИЗа. При обнаружении разделителя его сравнительный приоритет PC сравнивается с магазинным приоритетом PM разделителя из вершины магазина операций. Если PC > PM, то разделитель входной строки помещается в магазин (разделитель из исходной строки поступает в магазин и в том случае, когда магазин пуст). Если PC ≤ PM, то символ извлекается из магазина и записывается в выходную строку ПОЛИЗа.

**5.2 Ошибки вывода**

Если в результате работы алгоритма магазин не окажется пустым, то будет выведена ошибка Reverse Polish notation: Stack is not empty.

**5.3 Реализация перевода в ПОЛИЗ**

Метод convert() класса RPN реализует перевод заданной последовательности операторов и операндов в польскую инверсную запись. Метод возвращает два массива с элементами ПОЛИЗа. declare\_rpn – отдельный ПОЛИЗ для декларирования переменных, main\_rpn - ПОЛИЗ для основной части программы.

При выполении перевода используется if\_stack – стек меток необходимых для перемещения по конструкциям if-elseif-else, end\_stack – стек меток необходимых для прыжка на конец конструкций, заканчивающихся на закрывающую фигурную скобку, cycle\_stack – стек позволяющий проще определять какой тип цикла в данный момент преобразуется. Это необходимо т.к. каждый цикл использует свой способ управления метками.

Все приоритеты операторов хранятся в ассоциативной таблице PRIORITY. Каждому оператору соответствует свой массив из двух элементов в котором хранится сравнительный и магазинный приоритет.

Для обозначения метки в ПОЛИЗе используется выражение (имя\_метки).

Для обозначения перехода по метке используется выражение [имя\_метки].

**5.4 Тестирование**

Рассчитать сумму чётных чисел от 1 до 10 с использованием конструкции while.

Тест 1. Программа выводит корректную польскую инверсную запись.

Исходный код программы на модульном языке программирования:

dim ii1, ss1, tm1 !;

for (ii1 ass 1 to 6) {

tm1 ass ii1 \* 2;

ss1 ass ss1 + tm1;

};

Результат работы алгоритма преобразования в ПОЛИЗ представлен на рисунке 9.

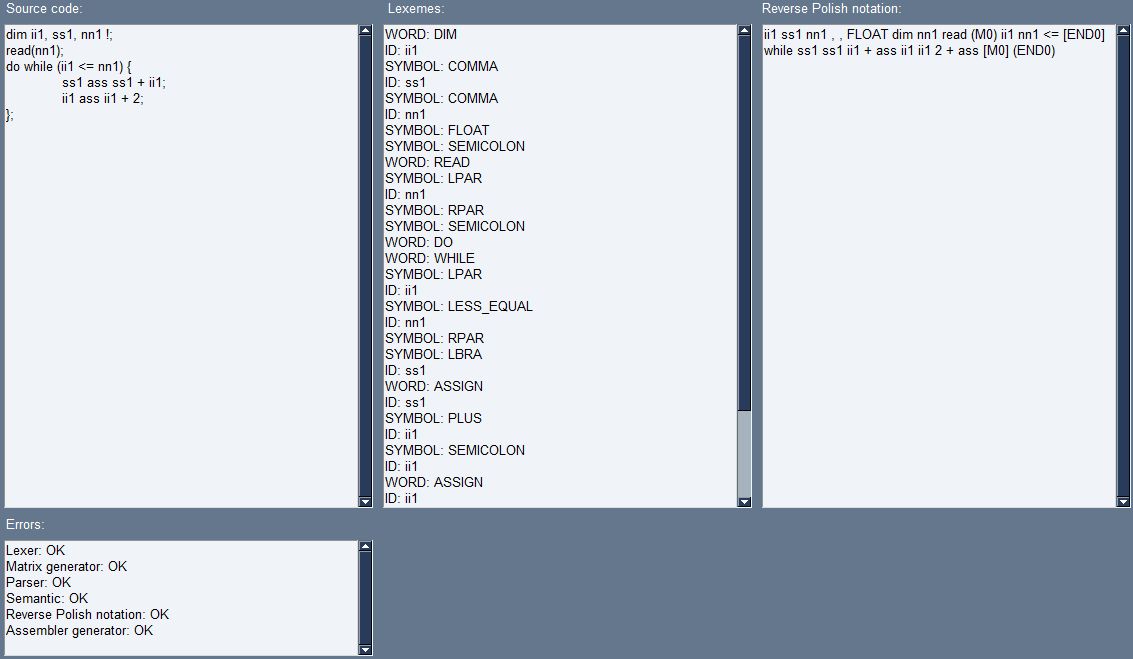


Рисунок 9 – Тест 1 алгоритма преобразования в ПОЛИЗ

# 6 Генерация ассемблерного кода из ПОЛИЗа

**6.1 Алгоритм**

Для перевода в ассемблер NASM используется алгоритм, считывающий поэлементно ПОЛИЗ и генерирующий ассемблерный код в соответствии со встретившимся элементом.

**6.2 Реализация перевода в ассемблерный код языка NASM**

Метод generate() класса ASM реализует перевод заданной польской инверсной записи в ассемблерный язык NASM x86. Он принимает на вход 2 части ПОЛИЗа – declare\_rpn и main\_rpn. Первая часть используется для декларирования переменных в сегменте данных, а вторая для реализации алгоритма программы в сегменте кода. Все числа представлены в формате single IEEE-754. Ввод и вывод происходит в том же формате с использованием шестнадцатеричного кодирования. Для конвертации чисел из десятичных чисел модульного языка программирования в нужный формат ассемблерного языка используется библиотека ieee754 языка Python. Ассемблерный код полученный в результате данного алгоритма можно перевести в объектный файл с помощью программы NASM и скомпилировать данный файл в исполняемый посредствам компилятора GCC или же запустить и удостоверится в его работе с помощью программы SASM. Создание исполняемого файла возможно нажатием кнопки «Build EXE» графического интерфейса программы.

**6.3 Тестирование**

Тест 1. Вычислить n-ое число Фибоначчи.

Исходный код программы на модульном языке программирования

dim aa1, aa2, ss1, nn1, ii1 !;

aa1 ass 1;

aa2 ass 1;

nn1 ass 10;

ii1 ass 3;

do while (ii1 <= nn1) {

ss1 ass aa1 + aa2;

aa1 ass aa2;

aa2 ass ss1;

ii1 ass ii1 + 1;

};

output(ss1);

Результат работы алгоритма генерации ассемблерного кода представлен на рисунке 10.

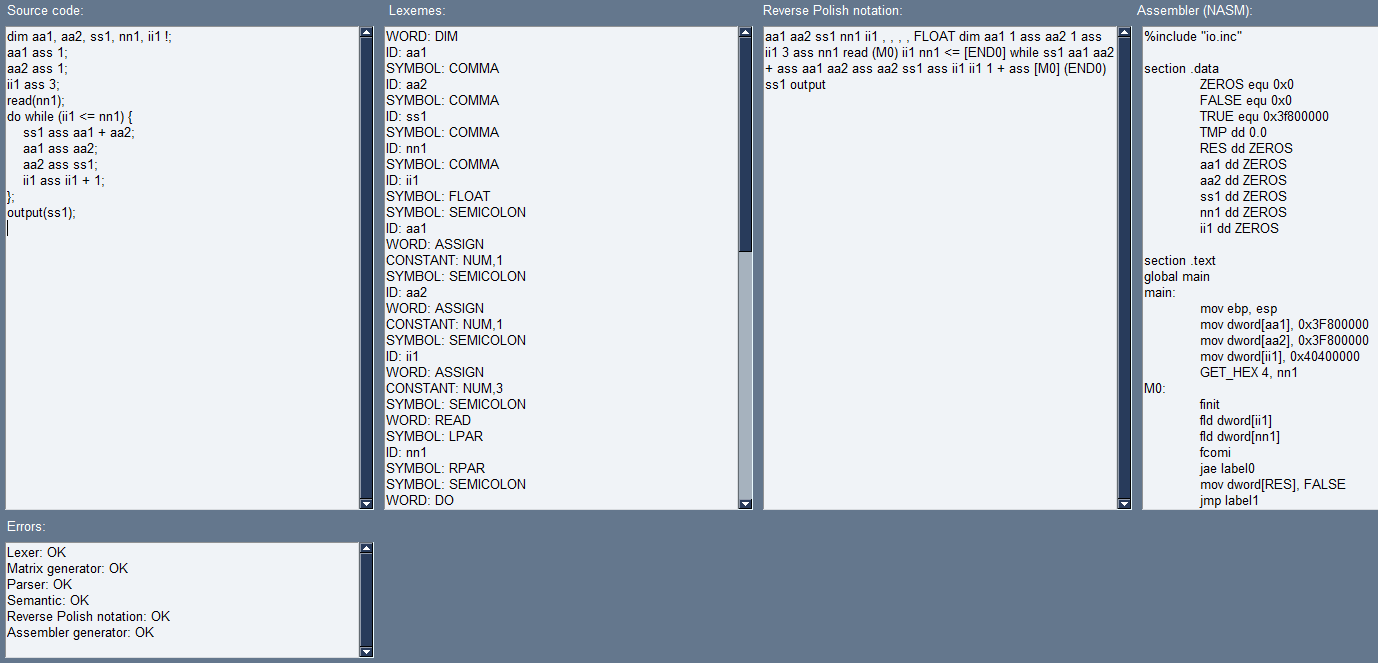
****

Рисунок 10 – Тест 1 алгоритма генерации ассемблерного кода

Полученный код на языке ассемблера:

%include "io.inc"

section .data

ZEROS equ 0x0

FALSE equ 0x0

TRUE equ 0x3f800000

TMP dd 0.0

RES dd ZEROS

aa1 dd ZEROS

aa2 dd ZEROS

ss1 dd ZEROS

nn1 dd ZEROS

ii1 dd ZEROS

section .text

global main

main:

mov ebp, esp

mov dword[aa1], 0x3F800000

mov dword[aa2], 0x3F800000

mov dword[ii1], 0x40400000

GET\_HEX 4, nn1

M0:

finit

fld dword[ii1]

fld dword[nn1]

fcomi

jae label0

mov dword[RES], FALSE

jmp label1

label0:

mov dword[RES], TRUE

label1:

finit

fld dword[RES]

mov dword[TMP], 0x00000000

fld dword[TMP]

fcomi

jz END0

finit

fld dword[aa1]

fld dword[aa2]

fadd

fstp dword[RES]

mov eax, dword[RES]

mov dword[ss1], eax

mov eax, dword[aa2]

mov dword[aa1], eax

mov eax, dword[ss1]

mov dword[aa2], eax

finit

fld dword[ii1]

mov dword[TMP], 0x3F800000

fld dword[TMP]

fadd

fstp dword[RES]

mov eax, dword[RES]

mov dword[ii1], eax

jmp M0

END0:

PRINT\_HEX 4, ss1

NEWLINE

Ret

При выполнении ассемблерного кода в среде SASM можно удостовериться, что ассемблирование прошло успешно. В ячейке памяти с именем ss1 хранится результат вычисления n-ого числа Фибоначчи. Значение n вводится пользователем а значение n-ого числа Фибоначчи выводится на экран. Входные данные 0x41200000 соответствует числу 10, а 0x425C0000 соотвествует 55 в кодировке IEEE-754.

Результат работы ассемблерного кода на языке NASM представлен на рисунке 11.

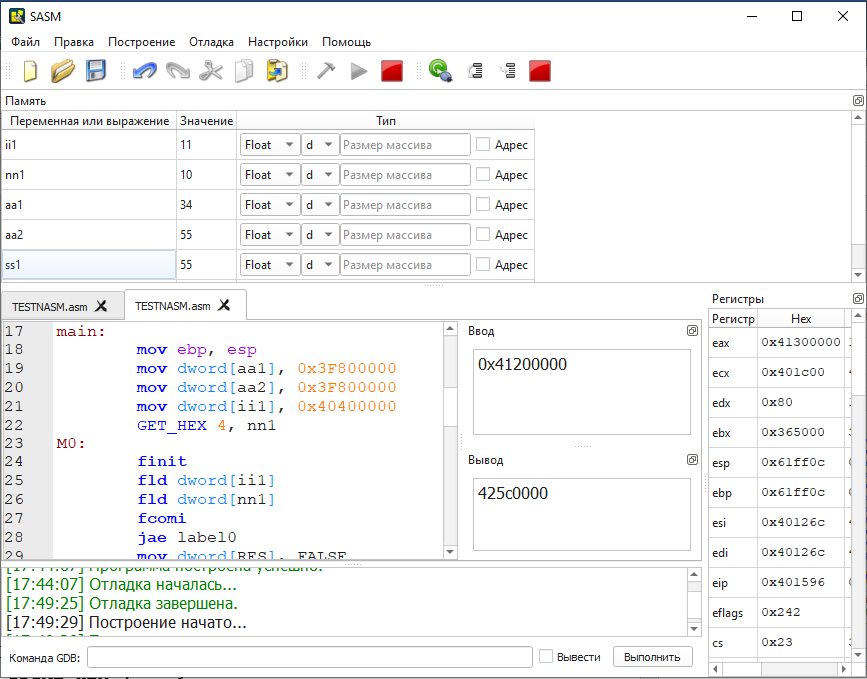


Рисунок 11 – Тест 1 выполнения ассемблерного кода в среде SASM

Тест 2. Выяснить взаимное расположение прямых и найти точку пересечения если она есть.

Исходный код программы на модульном языке программирования

dim kk1, kk2, bb1, bb2, xx1, yy1, tm1 !;

dim sm1, pr1, eq1, eq2 $;

read(kk1, bb1, kk2, bb2);

eq1 ass kk1 = kk2;

eq2 ass bb1 = bb2;

if (eq1 and eq2) {

sm1 ass true;

} elseif (eq1 = true) {

pr1 ass true;

} else {

xx1 ass bb2 - bb1;

tm1 ass kk1 - kk2;

xx1 ass xx1 / tm1;

yy1 ass kk1 \* xx1;

yy1 ass yy1 + bb1;

};

output(xx1, yy1);

Результат работы алгоритма генерации ассемблерного кода представлен на рисунке 12.

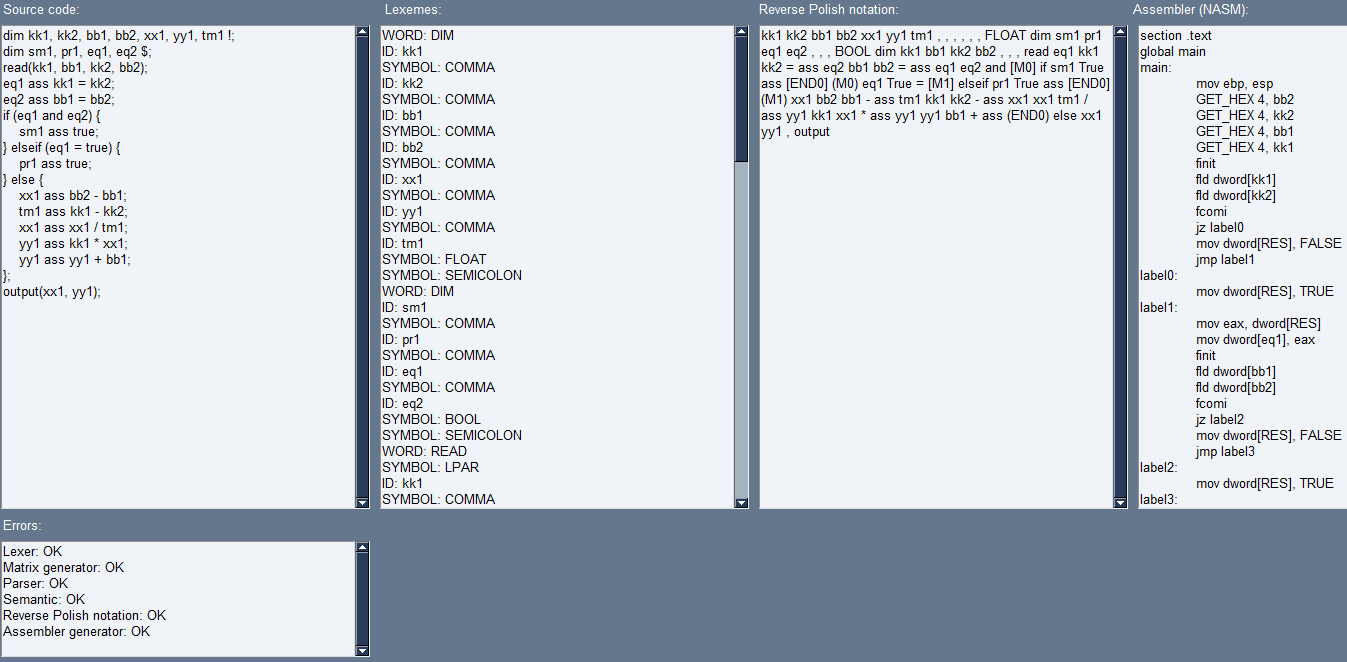


Рисунок 12 – Тест 2 алгоритма генерации ассемблерного кода

Полученный код на языке ассемблера:

%include "io.inc"

section .data

ZEROS equ 0x0

FALSE equ 0x0

TRUE equ 0x3f800000

TMP dd 0.0

RES dd ZEROS

kk1 dd ZEROS

kk2 dd ZEROS

bb1 dd ZEROS

bb2 dd ZEROS

xx1 dd ZEROS

yy1 dd ZEROS

tm1 dd ZEROS

sm1 dd ZEROS

pr1 dd ZEROS

eq1 dd ZEROS

eq2 dd ZEROS

section .text

global main

main:

mov ebp, esp

GET\_HEX 4, bb2

GET\_HEX 4, kk2

GET\_HEX 4, bb1

GET\_HEX 4, kk1

finit

fld dword[kk1]

fld dword[kk2]

fcomi

jz label0

mov dword[RES], FALSE

jmp label1

label0:

mov dword[RES], TRUE

label1:

mov eax, dword[RES]

mov dword[eq1], eax

finit

fld dword[bb1]

fld dword[bb2]

fcomi

jz label2

mov dword[RES], FALSE

jmp label3

label2:

mov dword[RES], TRUE

label3:

mov eax, dword[RES]

mov dword[eq2], eax

mov eax, dword[eq1]

and eax, dword[eq2]

mov dword[RES], eax

finit

fld dword[RES]

mov dword[TMP], 0x00000000

fld dword[TMP]

fcomi

jnz label4

jmp M0

label4:

mov dword[sm1], TRUE

jmp END0

M0:

finit

fld dword[eq1]

mov dword[TMP], TRUE

fld dword[TMP]

fcomi

jz label5

mov dword[RES], FALSE

jmp label6

label5:

mov dword[RES], TRUE

label6:

finit

fld dword[RES]

mov dword[TMP], 0x00000000

fld dword[TMP]

fcomi

jnz label7

jmp M1

label7:

mov dword[pr1], TRUE

jmp END0

M1:

finit

fld dword[bb2]

fld dword[bb1]

fsub

fstp dword[RES]

mov eax, dword[RES]

mov dword[xx1], eax

finit

fld dword[kk1]

fld dword[kk2]

fsub

fstp dword[RES]

mov eax, dword[RES]

mov dword[tm1], eax

finit

fld dword[xx1]

fld dword[tm1]

fdiv

fstp dword[RES]

mov eax, dword[RES]

mov dword[xx1], eax

finit

fld dword[kk1]

fld dword[xx1]

fmul

fstp dword[RES]

mov eax, dword[RES]

mov dword[yy1], eax

finit

fld dword[yy1]

fld dword[bb1]

fadd

fstp dword[RES]

mov eax, dword[RES]

mov dword[yy1], eax

END0:

PRINT\_HEX 4, yy1

NEWLINE

PRINT\_HEX 4, xx1

NEWLINE

ret

Результат работы ассемблерного кода на языке NASM для входных данных k1 = 2, b1 = -1, k2 = -3, b2 = 1 представлен на рисунках 13-14. 0x3ECCCCCD и 0xBE4CCCCC в кодировке IEEE-754 соответственно равны x = 0.4 и y = -0.2.

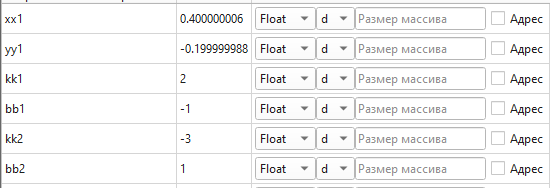


Рисунок 13 – Тест 2 выполнения ассемблерного кода в среде SASM

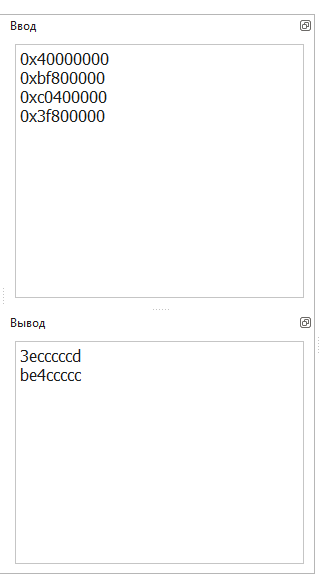


Рисунок 14 – Тест 2 выполнения ассемблерного кода в среде SASM

Тест 3.

Результат работы ассемблерного кода из теста 2 на языке NASM для входных данных k1 = 2, b1 = -1, k2 = 2, b2 = 1 представлен на рисунке 15. По результату pr1 = 1 видно, что две прямые параллельны.



Рисунок 15 – Тест 3 выполнения ассемблерного кода в среде SASM

Тест 4.

Результат работы ассемблерного кода из теста 2 на языке NASM для входных данных k1 = 2, b1 = 1, k2 = 2, b2 = 1 представлен на рисунке 16. По результату sm1 = 1 видно, что прямые являются одинаковыми и у них бесконечное количество общих точек.

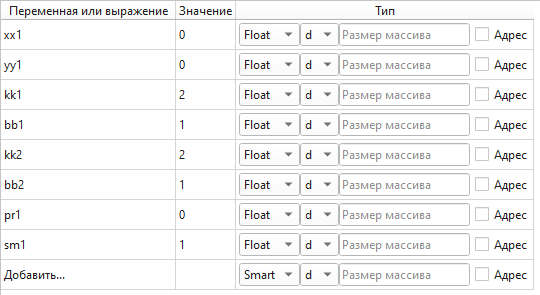


Рисунок 16 – Тест 4 выполнения ассемблерного кода в среде SASM

Тест 5.

Ассемблерный код на языке NASM и входные данные взяты из теста 2. Созданные объектный и исполняемый файлы с помощью функции «Build EXE» и демонстрация запуска исполняемого файла представлены на рисунках 17-18.

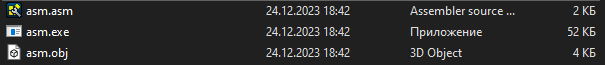


Рисунок 17 – Тест 5 файлы, полученные при построении исполняемого файла

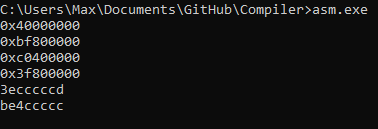


Рисунок 18 – Тест 5 демонстрация работы исполняемого файла

# Заключение

В процессе разработки программного обеспечения для решения конкретной задачи была изучена специфическая литература по теме работы: литература, посвящённая созданию компиляторов, языкам программирования Python и NASM; разработаны алгоритмы лексического, синтаксического и семантического анализов, реализованы алгоритмы перевода в ПОЛИЗ и трансляции на ассемблерный язык программирования NASM с возможностью пользовательского ввода-вывода и последующей компиляции в исполняемый файл. Так же были изучены библиотеки языка Python, работающие с графическим интерфейсом и форматом чисел IEEE-754.

Программа может быть использована пользователем для компиляции модульного языка программирования в ассемблерный код.

Усовершенствовать данное приложение можно улучшением интерфейса и добавлением новых языковых конструкций в модульный язык программирования.

# Список источников

1. Молчанов А. Ю. М76 Системное программной обеспечение. Лабораторный практикум — СПб: Питер, 2005 — 284 с.: ил (дата обращения 24.10.2023)
2. Справка по NASM – Общая информация о языке NASM [Сайт]. URL: https://www.nasm.us/xdoc/2.16.01/html/nasmdoc3.html (дата обращения 14.12.2023)
3. Справка по Python – Механизм реализации хеш-таблиц [Сайт]. URL: https://docs.python.org/3/reference/datamodel.html#object.\_\_hash\_\_ (дата обращения 16.09.2023)
4. Малявко, А. А. Формальные языки и компиляторы : учебное пособие для вузов / А. А. Малявко. — Москва : Издательство Юрайт, 2022. — 429 с. — (Высшее образование). (дата обращения 26.11.2023)

# Приложение А

# Текст программы

**# lexer.py**

decrypt = ['NUM', 'REAL', 'ID', 'if', 'else', 'elseif', 'for', 'to', 'do', 'while', '(', ')', '+', '-', '>', '>=', '<', '<=', '=', '!=', '\*', '/', ';', '.', ',', '{', '}', 'ass', 'dim', 'and', 'or', 'not', 'BOOL\_VAL', 'read', 'output', 'INT', 'FLOAT', 'BOOL', 'EOF']  
  
decrypt\_to\_operators = ['NUM', 'NUM', 'ID', 'if', 'else', 'elseif', 'for', 'to', 'do', 'while', '(', ')', '+', '-', '>', '>=', '<', '<=', '=', '!=', '\*', '/', ';', '.', ',', '{', '}', 'ass', 'dim', 'and', 'or', 'not', 'BOOL', 'read', 'output', 'TYPE', 'TYPE', 'TYPE', '/e/']  
  
decrypt\_to\_lexemes = ["NUM", "REAL", "ID", "IF", "ELSE", "ELSEIF", "FOR", "TO", "DO", "WHILE", "LPAR", "RPAR", "PLUS", "MINUS", "GREAT", "GREAT\_EQUAL", "LESS", "LESS\_EQUAL", "EQUAL", "NOTEQUAL", "MULTIPLY", "DIVIDE", "SEMICOLON", "DOT", "COMMA", "LBRA", "RBRA", "ASSIGN", "DIM", "AND", "OR", "NOT", "BOOL\_VAL", "READ", "OUTPUT", "INT", "FLOAT", "BOOL", "EOF"]  
  
class Lexer:  
 (NUM, REAL, ID, IF, ELSE, ELSEIF, FOR, TO, DO, WHILE, LPAR, RPAR, PLUS, MINUS, GREAT, GREAT\_EQUAL, LESS, LESS\_EQUAL, EQUAL, NOTEQUAL, MULTIPLY, DIVIDE, SEMICOLON, DOT, COMMA, LBRA, RBRA, ASSIGN, DIM, AND, OR, NOT, BOOL\_VAL, READ, OUTPUT, INT, FLOAT, BOOL, EOF) = range(39)  
 SYMBOLS = {';': SEMICOLON, '+': PLUS, '-': MINUS, '\*': MULTIPLY, '/': DIVIDE, '>': GREAT, '>=': GREAT\_EQUAL, '<': LESS, '<=': LESS\_EQUAL, '=': EQUAL, '!=': NOTEQUAL, '.': DOT, ',': COMMA, '(': LPAR, ')': RPAR, '{': LBRA, '}': RBRA, '%': INT, "!": FLOAT, '$': BOOL}  
 WORDS = {'if': IF, 'else': ELSE, 'elseif': ELSEIF, 'for': FOR, 'to': TO, 'do': DO, 'while': WHILE, 'ass': ASSIGN, 'dim': DIM, 'read': READ, 'output': OUTPUT, 'and': AND, 'or': OR, 'not': NOT, 'true': BOOL\_VAL, 'false': BOOL\_VAL}  
 char = ' '  
 lines\_count = 1  
  
 def \_\_init\_\_(self, input\_stream):  
 self.symbol = None  
 self.value = None  
 self.input\_stream = input\_stream  
 self.error = False  
 self.error\_msg = "Lexer: OK"  
 def set\_error(self, msg):  
 self.error = True  
 self.error\_msg = "Lexer: " + msg  
 def getc(self):  
 self.char = self.input\_stream.read(1)  
 if self.char == "\n":  
 self.lines\_count += 1  
 def next\_token(self):  
 self.value = None  
 self.symbol = None  
 while self.symbol is None:  
 if len(self.char) == 0:  
 self.symbol = Lexer.EOF  
 elif self.char.isspace():  
 self.getc()  
 elif self.char in Lexer.SYMBOLS:  
 new\_symbol = self.char  
 self.getc()  
 double\_symbol = new\_symbol + self.char  
 if new\_symbol == '-':  
 if self.char.isdigit():  
 intval = int(self.char)  
 self.getc()  
 while self.char.isdigit():  
 intval = intval \* 10 + int(self.char)  
 self.getc()  
 if self.char == '.':  
 floatval = 0  
 count = 1  
 self.getc()  
 while self.char.isdigit():  
 floatval = floatval + int(self.char) / (10 \*\* count)  
 count += 1  
 self.getc()  
 self.value = -(intval + floatval)  
 self.symbol = Lexer.REAL  
 else:  
 self.value = -intval  
 self.symbol = Lexer.NUM  
 else:  
 self.symbol = Lexer.SYMBOLS[new\_symbol]  
 elif double\_symbol in Lexer.SYMBOLS:  
 self.symbol = Lexer.SYMBOLS[double\_symbol]  
 self.getc()  
 elif double\_symbol == '/\*':  
 out\_char = ''  
 while out\_char != '\*/':  
 self.getc()  
 if len(self.char) == 0:  
 self.symbol = Lexer.EOF  
 break  
 out\_char += self.char  
 if out\_char == '\*' or out\_char == '\*/':  
 continue  
 out\_char = ''  
 self.getc()  
 else:  
 self.symbol = Lexer.SYMBOLS[new\_symbol[0]]  
 elif self.char.isdigit():  
 intval = 0  
 while self.char.isdigit():  
 intval = intval \* 10 + int(self.char)  
 self.getc()  
 if self.char == '.':  
 floatval = 0  
 count = 1  
 self.getc()  
 while self.char.isdigit():  
 floatval = floatval + int(self.char) / (10 \*\* count)  
 count += 1  
 self.getc()  
 self.value = intval + floatval  
 self.symbol = Lexer.REAL  
 else:  
 self.value = intval  
 self.symbol = Lexer.NUM  
 elif 'a' <= self.char <= 'z':  
 identifier = ''  
 while 'a' <= self.char <= 'z':  
 identifier += self.char.lower()  
 self.getc()  
 if identifier in Lexer.WORDS:  
 self.symbol = Lexer.WORDS[identifier]  
 if identifier == "false":  
 self.value = False  
 elif identifier == "true":  
 self.value = True  
 elif len(identifier) == 2 and self.char.isdigit():  
 identifier += self.char  
 self.getc()  
 while self.char.isdigit():  
 identifier += self.char  
 self.getc()  
 self.symbol = Lexer.ID  
 self.value = identifier  
 else:  
 self.set\_error('Unknown identifier "' + identifier + '" in line ' + str(self.lines\_count))  
 break  
 else:  
 self.set\_error('Unexpected symbol "' + self.char + '" in line ' + str(self.lines\_count))  
 break  
  
**# matrix.py**

from enum import Enum

class ORDER(Enum):

PRECEDED = 1

FOLLOWS = 2

EQUALS = 3

class LastSymbols:

def \_\_init\_\_(self):

self.left = {}

self.right = {}

def \_\_eq\_\_(self, other):

if isinstance(other, LastSymbols):

return self.left == other.left and self.right == other.right

return NotImplemented

class MatrixGenerator:

SYMBOLS = []

WORDS = []

T = []

def \_\_init\_\_(self, input\_stream):

self.input\_stream = input\_stream

self.operator\_matrix = {}

self.error = False

self.error\_msg = "Matrix generator: OK"

self.rules = {}

self.symbols\_lr = LastSymbols()

self.t\_lr = LastSymbols()

def set\_error(self, msg):

self.error = True

self.error\_msg = "Matrix generator: " + msg

def print\_matrix(self):

print(" " \* 6, end="")

for x in [\*MatrixGenerator.T, "/e/"]:

print("{0:<6}".format(x), end="")

print()

for i in [\*MatrixGenerator.T, "/b/"]:

print("{0:<6}".format(i), end="")

for j in [\*MatrixGenerator.T, "/e/"]:

if self.operator\_matrix[i].get(j) is None:

print("." + " " \* 5, end="")

elif self.operator\_matrix[i][j] == ORDER.PRECEDED:

print("<." + " " \* 4, end="")

elif self.operator\_matrix[i][j] == ORDER.EQUALS:

print("=." + " " \* 4, end="")

elif self.operator\_matrix[i][j] == ORDER.FOLLOWS:

print(".>" + " " \* 4, end="")

print()

def generate(self):

rule\_names = self.input\_stream.readline().split()

MatrixGenerator.T = self.input\_stream.readline().split()

for name in rule\_names:

self.rules[name] = []

self.symbols\_lr.left[name] = []

self.symbols\_lr.right[name] = []

self.t\_lr.left[name] = []

self.t\_lr.right[name] = []

for symbol in ["/b/", \*MatrixGenerator.T, "/e/"]:

self.operator\_matrix[symbol] = {}

lines = self.input\_stream.read().split('\n')

for line in lines:

last\_token\_index = 0

tokens = line.split()

if tokens[1] == ":" and tokens[0] in rule\_names and len(tokens) > 2:

rule\_name = tokens[0]

tokens = tokens[2:]

for i in range(len(tokens)):

if tokens[i] == "|":

self.rules[rule\_name].append(tokens[last\_token\_index:i])

last\_token\_index = i+1

elif tokens[i] not in MatrixGenerator.T and tokens[i] not in rule\_names:

self.set\_error('unknown symbol "' + tokens[i] + '"')

return self.error

self.rules[rule\_name].append(tokens[last\_token\_index:])

else:

self.set\_error("wrong file formatting")

return self.error

for rule\_name in self.rules.keys(): # начальное заполнение крайних левых, правых

for arr in self.rules[rule\_name]:

first\_t = None

last\_t = None

first\_symbol = None

last\_symbol = None

for item in arr:

if item in MatrixGenerator.T:

if first\_t is None:

first\_t = item

last\_t = item

else:

last\_t = item

if first\_symbol is None:

first\_symbol = item

last\_symbol = item

else:

last\_symbol = item

if last\_t is not None:

if first\_t not in self.t\_lr.left[rule\_name]:

self.t\_lr.left[rule\_name].append(first\_t)

if last\_t not in self.t\_lr.right[rule\_name]:

self.t\_lr.right[rule\_name].append(last\_t)

if last\_symbol is not None:

if first\_symbol not in self.symbols\_lr.left[rule\_name]: self.symbols\_lr.left[rule\_name].append(first\_symbol)

if last\_symbol not in self.symbols\_lr.right[rule\_name]: self.symbols\_lr.right[rule\_name].append(last\_symbol)

changed = True

while changed: # алгоритм для всех симв

changed = False

for rule\_name in self.rules.keys():

for item in self.symbols\_lr.left[rule\_name]:

if item in self.rules.keys():

for i in self.symbols\_lr.left[item]:

if i not in self.symbols\_lr.left[rule\_name]: self.symbols\_lr.left[rule\_name].append(i)

changed = True

for item in self.symbols\_lr.right[rule\_name]:

if item in self.rules.keys():

for i in self.symbols\_lr.right[item]:

if i not in self.symbols\_lr.right[rule\_name]: self.symbols\_lr.right[rule\_name].append(i)

changed = True

for rule\_name in self.rules.keys(): # алгоритм для терм симв

for item in self.symbols\_lr.left[rule\_name]:

if item not in MatrixGenerator.T:

for i in self.t\_lr.left[item]:

if i not in self.t\_lr.left[rule\_name] and i in MatrixGenerator.T:

self.t\_lr.left[rule\_name].append(i)

for item in self.symbols\_lr.right[rule\_name]:

if item not in MatrixGenerator.T:

for i in self.t\_lr.right[item]:

if i not in self.t\_lr.right[rule\_name] and i in MatrixGenerator.T:

self.t\_lr.right[rule\_name].append(i)

for symbol in MatrixGenerator.T:

for rule in self.rules.values():

for item in rule:

for i in range(len(item)):

if symbol == item[i]:

if i < len(item)-1 and item[i+1] in MatrixGenerator.T: # x ai b y

if self.operator\_matrix[symbol].get(item[i+1]) is not None and self.operator\_matrix[symbol][item[i+1]] != ORDER.EQUALS:

print("ОШИБКА", item, item[i:i+2], "x ai b y OLD:", self.operator\_matrix[symbol][item[i+1]])

# print(item[i:i + 2], "x ai b y")

self.operator\_matrix[symbol][item[i+1]] = ORDER.EQUALS

elif i < len(item)-2 and item[i+1] not in MatrixGenerator.T and item[i+2] in MatrixGenerator.T: # x ai U b y

if self.operator\_matrix[symbol].get(item[i+2]) is not None and self.operator\_matrix[symbol][item[i+2]] != ORDER.EQUALS:

print("ОШИБКА", item, item[i:i+3], "x ai U b y")

# print(item[i:i + 3], "x ai U b y")

self.operator\_matrix[symbol][item[i+2]] = ORDER.EQUALS

if i < len(item)-1 and item[i+1] not in MatrixGenerator.T: # x ai U y

for L in self.t\_lr.left[item[i+1]]:

if L in MatrixGenerator.T:

if self.operator\_matrix[symbol].get(L) is not None and self.operator\_matrix[symbol][L] != ORDER.PRECEDED:

print("ОШИБКА", item, L, item[i:i + 2], "x ai U y")

# print(L, item[i:i + 2], "x ai U y")

self.operator\_matrix[symbol][L] = ORDER.PRECEDED

continue

if i > 0 and item[i-1] not in MatrixGenerator.T: # x U ai y

for R in self.t\_lr.right[item[i-1]]:

if R in MatrixGenerator.T:

if self.operator\_matrix[R].get(symbol) is not None and self.operator\_matrix[R][symbol] != ORDER.FOLLOWS:

print("ОШИБКА", item, R, item[i - 1:i + 1], "x U ai y")

self.operator\_matrix[R][symbol] = ORDER.FOLLOWS

continue

for s in self.t\_lr.left[rule\_names[0]]:

self.operator\_matrix["/b/"][s] = ORDER.PRECEDED

for s in self.t\_lr.right[rule\_names[0]]:

self.operator\_matrix[s]["/e/"] = ORDER.FOLLOWS

self.print\_matrix()

return self.operator\_matrix

**# parser.py**

import matrix

class Parser:

def \_\_init\_\_(self, op\_matrix, operators, rules):

self.operators = operators

self.t = ['/b/', \*operators]

self.operator\_matrix = op\_matrix

self.error = False

self.error\_msg = "Parser: OK"

self.stack = []

self.rules: dict = rules

def set\_error(self, msg):

self.error = True

self.error\_msg = "Parser: " + msg

def print\_matrix(self):

pass

def get\_t(self, n=1):

if n < 1:

n = 1

for item in self.stack[::-1]:

if item in self.t:

n -= 1

if n == 0:

return item

return "ERROR"

def check(self):

self.stack.append('/b/')

i = 0

while True:

if self.get\_t() == '/b/' and self.operators[i] == '/e/':

break

if self.get\_t() not in self.operator\_matrix.keys() or self.operators[i] not in self.operator\_matrix[self.get\_t()].keys():

self.set\_error("unknown construction '" + self.get\_t() + " " + self.operators[i] + "' in " + " ".join(self.operators[i-2:i+1]))

break

if self.operator\_matrix[self.get\_t()][self.operators[i]] == matrix.ORDER.EQUALS or self.operator\_matrix[self.get\_t()][self.operators[i]] == matrix.ORDER.PRECEDED:

self.stack.append(self.operators[i])

i += 1

elif self.operator\_matrix[self.get\_t()][self.operators[i]] == matrix.ORDER.FOLLOWS:

rule = []

if self.operator\_matrix[self.get\_t(2)][self.get\_t()] == matrix.ORDER.PRECEDED:

count = 1

while True:

if self.stack[-1] in self.t:

if count == 0:

break

count -= 1

rule.append(self.stack.pop())

find = False

rule.reverse()

for name in self.rules.keys():

for r in self.rules[name]:

if rule == r:

find = True

self.stack.append("E")

break

if find:

break

if not find:

self.set\_error("Unable to locate rule " + " ".join(rule))

break

elif self.operator\_matrix[self.get\_t(2)][self.get\_t()] == matrix.ORDER.EQUALS:

count = 2

while self.operator\_matrix[self.get\_t(count+1)][self.get\_t(count)] == matrix.ORDER.EQUALS:

count += 1

while True:

if self.stack[-1] in self.t:

if count == 0:

break

count -= 1

rule.append(self.stack.pop())

find = False

rule.reverse()

# print("RULE:", rule)

for name in self.rules.keys():

for r in self.rules[name]:

if rule == r:

find = True

self.stack.append("E")

break

if find:

break

if not find:

self.set\_error("Unable to locate rule " + " ".join(rule))

break

**# semantic.py**

import lexer

class Variable:

def \_\_init\_\_(self, type):

self.type = type

class Operator:

def \_\_init\_\_(self, type, name, value=None):

self.type = type

self.name = name

self.value = value

class Semantic:

RETURN\_BOOL = [">", ">=", "<", "<=", "=", "!=", "and", "or", "not"]

RETURN\_NUM = ["+", "-", "\*", "/"]

def \_\_init\_\_(self, operators):

self.operators: list[Operator] = operators

self.error = False

self.error\_msg = "Semantic: OK"

self.variables: dict[Variable] = {}

def set\_error(self, msg):

self.error = True

self.error\_msg = "Semantic: " + msg

def check(self):

i = 0

while i < len(self.operators):

if self.operators[i].type == lexer.Lexer.DIM:

j = i + 1

while self.operators[j].name != "TYPE":

j += 1

vars\_type = lexer.decrypt[self.operators[j].type]

for x in range(i + 1, j + 1, 2):

self.variables[self.operators[x].value] = Variable(vars\_type)

elif self.operators[i].type == lexer.Lexer.ID and self.operators[i + 1].type == lexer.Lexer.ASSIGN:

if self.variables.get(self.operators[i].value) is None:

self.error = True

self.error\_msg = "Undeclared variable " + self.operators[i].value

id\_type = self.variables[self.operators[i].value].type

const\_type = lexer.decrypt[self.operators[i + 2].type]

operation\_return = lexer.decrypt[self.operators[i + 3].type]

if operation\_return == ";" or operation\_return == "to":

if const\_type == "ID" and self.variables[self.operators[i].value].type != id\_type:

print("ID SEMANTIC")

self.error = True

if self.variables.get(self.operators[i + 2].value) is None:

self.error\_msg = "Undeclared variable " + self.operators[i+2].value

break

self.error\_msg = ("Wrong type: " + self.operators[i].value + " is " +

id\_type + " unable to assign " +

self.variables[self.operators[i + 2].value].type + " type")

break

elif const\_type != "ID" and not ((id\_type == "INT" or id\_type == "FLOAT") and (const\_type == "NUM"

or const\_type == "REAL") or (id\_type == "BOOL" and const\_type == "BOOL\_VAL")):

print("CONST SEMANTIC")

self.error = True

self.error\_msg = ("Wrong type: " + self.operators[i].value + " is " +

id\_type + " unable to assign " +

const\_type + " type")

break

elif ((id\_type == "INT" or id\_type == "FLOAT") and operation\_return not in

self.RETURN\_NUM) or id\_type == "BOOL" and operation\_return not in self.RETURN\_BOOL:

print(operation\_return)

print("OPERATOR SEMANTIC")

self.error = True

if operation\_return in self.RETURN\_NUM:

self.error\_msg = ("Wrong type: " + self.operators[i].value + " is " + id\_type

+ " unable to assign number type")

else:

self.error\_msg = ("Wrong type: " + self.operators[i].value + " is " + id\_type

+ " unable to assign BOOL type")

break

elif self.operators[i].type == lexer.Lexer.ID:

if self.variables.get(self.operators[i].value) is None:

self.error = True

self.error\_msg = "Undeclared variable " + self.operators[i].value

i += 1

**# RPN.py**

class PRN:

OPERANDS = ["ID", "NUM", "REAL", "BOOL\_VAL"]

UNARY = ["NOT"]

IGNORED = ["EOF", "do"]

UNPRINTABLE = [";", "{", "(", "}", ")", "to"]

PRIORITY = {

"INT": [2, 2],

"FLOAT": [2, 2],

"BOOL": [2, 2],

"(": [3, 1],

")": [0, 0],

"+": [5, 5],

"-": [5, 5],

">": [5, 5],

"<": [5, 5],

">=": [5, 5],

"<=": [5, 5],

"=": [5, 5],

"!=": [5, 5],

"\*": [5, 5],

"/": [5, 5],

";": [0, 0],

",": [3, 2],

"{": [1, 0],

"}": [0, 0],

"if": [1, 2],

"else": [1, 0],

"elseif": [1, 2],

"for": [1, 2],

"while": [1, 2],

"ass": [2, 1],

"dim": [2, 2],

"and": [5, 5],

"or": [5, 5],

"not": [5, 5],

"read": [2, 2],

"output": [2, 2],

"to": [0, 0],

}

def \_\_init\_\_(self, operators):

self.operators = operators

self.error = False

self.error\_msg = "Reverse Polish notation: OK"

self.stack = []

self.end\_stack = []

self.if\_stack = []

self.if\_count = 0

self.end\_count = 0

self.target\_counter = ""

self.cycle\_stack = []

def set\_error(self, msg):

self.error = True

self.error\_msg = "Reverse Polish notation: " + msg

def convert(self):

res = []

declare\_res = []

is\_declare = False

for i in range(len(self.operators)):

print("OPERATOR:", self.operators[i][0])

if self.operators[i][0] in self.OPERANDS:

if is\_declare:

declare\_res.append(self.operators[i][1])

else:

res.append(self.operators[i][1])

print("OPERAND")

else:

if self.operators[i][0] == "dim":

is\_declare = True

if self.operators[i][0] == "for":

self.target\_counter = self.operators[i+2][1]

if self.operators[i][0] == "if" or self.operators[i][0] == "for" or self.operators[i][0] == "while":

self.end\_stack.append("END" + str(self.end\_count))

self.end\_count += 1

self.cycle\_stack.append(self.operators[i][0])

if self.operators[i][0] == "do":

self.if\_stack.append("M" + str(self.if\_count))

self.if\_count += 1

res.append("(" + self.if\_stack[-1] + ")")

if self.operators[i][0] in self.IGNORED:

print("CONTINUE WITH: " + self.operators[i][0])

continue

if len(self.stack) == 0 or self.PRIORITY[self.stack[-1]][1] < self.PRIORITY[self.operators[i][0]][0]:

print("STACK:", len(self.stack))

self.stack.append(self.operators[i][0])

else:

print("CYCLE:", self.cycle\_stack)

stack\_symbol = ""

if self.operators[i][0] == ")":

while stack\_symbol != "(":

stack\_symbol = self.stack.pop()

if stack\_symbol not in self.UNPRINTABLE:

if stack\_symbol == "elseif":

stack\_symbol = "if"

res.append(stack\_symbol)

stack\_symbol = self.stack.pop()

if stack\_symbol == "if" or stack\_symbol == "else" or stack\_symbol == "elseif": # else????

self.if\_stack.append("M" + str(self.if\_count))

self.if\_count += 1

res.append("[" + self.if\_stack[-1] + "]")

if stack\_symbol == "for" or stack\_symbol == "while":

res.append("[" + self.end\_stack[-1] + "]")

if stack\_symbol not in self.UNPRINTABLE:

res.append(stack\_symbol)

if self.operators[i][0] == "}":

while stack\_symbol != "{":

stack\_symbol = self.stack.pop()

if stack\_symbol not in self.UNPRINTABLE:

if stack\_symbol == "elseif":

stack\_symbol = "if"

res.append(stack\_symbol)

if self.operators[i+1][0] == ";":

cycle\_type = self.cycle\_stack.pop()

if cycle\_type == "for":

res.append(self.target\_counter)

res.append(self.target\_counter)

res.append("1")

res.append("+")

res.append("ass")

res.append("[" + self.if\_stack.pop() + "]")

res.append("(" + self.end\_stack.pop() + ")")

if cycle\_type == "while":

res.append("[" + self.if\_stack.pop() + "]")

res.append("(" + self.end\_stack.pop() + ")")

if cycle\_type == "if":

res.append("(" + self.end\_stack.pop() + ")")

if len(self.stack) == 0 or self.stack[-1] != "else":

res.append("(" + self.if\_stack.pop() + ")")

else:

# stack\_symbol = self.stack.pop()

# res.append(stack\_symbol)

res.append("[" + self.end\_stack[-1] + "]")

res.append("(" + self.if\_stack.pop() + ")")

if self.operators[i][0] == ";":

while stack\_symbol != "{" and len(self.stack) != 0:

stack\_symbol = self.stack.pop()

if stack\_symbol not in self.UNPRINTABLE:

if stack\_symbol == "elseif":

stack\_symbol = "if"

if is\_declare:

declare\_res.append(stack\_symbol)

else:

res.append(stack\_symbol)

is\_declare = False

if stack\_symbol == "{":

self.stack.append(stack\_symbol)

if self.operators[i][0] == "INT" or self.operators[i][0] == "FLOAT" or self.operators[i][0] == "BOOL":

stack\_symbol = self.stack.pop()

while stack\_symbol == ",":

declare\_res.append(stack\_symbol)

stack\_symbol = self.stack.pop()

self.stack.append(stack\_symbol)

declare\_res.append(self.operators[i][0])

if self.operators[i][0] == "to":

while stack\_symbol != "(":

stack\_symbol = self.stack.pop()

if stack\_symbol not in self.UNPRINTABLE:

if stack\_symbol == "elseif":

stack\_symbol = "if"

res.append(stack\_symbol)

self.stack.append(stack\_symbol)

self.if\_stack.append("M" + str(self.if\_count))

self.if\_count += 1

res.append("(" + self.if\_stack[-1] + ")")

res.append(self.target\_counter)

print(res)

print(declare\_res)

print("---")

if (len(self.stack) == 1 and self.stack[-1] != ";") and len(self.stack) > 0:

print("STACK ERROR:", self.stack)

self.set\_error("Stack is not empty")

return declare\_res, res

**# ASM.py**

from ieee754 import single

class ASM:

TYPES = ["INT", "BOOL", "FLOAT"]

OPERATORS = ["+", "-", "\*", "/", ">", ">=", "<", "<=", "=", "!=", ",", "if", "else", "elseif", "for", "while", "ass", "read", "output", "and", "or", "not"]

def \_\_init\_\_(self, declare\_rpn, main\_rpn):

self.error = False

self.error\_msg = "Assembler generator: OK"

self.declare\_rpn = declare\_rpn

self.main\_rpn = main\_rpn

self.variables = []

self.var\_names = []

self.stack = []

self.calced = False

self.labels\_count = 0

def set\_error(self, msg):

self.error = True

self.error\_msg = "Assembler generator: " + msg

def two\_operands\_calc(self):

res = ""

second = self.stack.pop()

first = self.stack.pop()

res += "\tfinit\n"

if first in self.var\_names:

res += "\tfld dword[" + first + "]\n"

else:

if first == "True":

res += "\tmov dword[TMP], TRUE\n\tfld dword[TMP]\n"

elif first == "False":

res += "\tmov dword[TMP], FALSE\n\tfld dword[TMP]\n"

else:

res += "\tmov dword[TMP], 0x" + single(first).hex()[0] + "\n\tfld dword[TMP]\n"

if second in self.var\_names:

res += "\tfld dword[" + second + "]\n"

else:

if second == "True":

res += "\tmov dword[TMP], TRUE\n\tfld dword[TMP]\n"

elif second == "False":

res += "\tmov dword[TMP], FALSE\n\tfld dword[TMP]\n"

else:

res += "\tmov dword[TMP], 0x" + single(second).hex()[0] + "\n\tfld dword[TMP]\n"

return res

def generate(self):

res = ('%include "io.inc"\n\n'

'section .data\n'

'\tZEROS equ 0x0\n'

'\tFALSE equ 0x0\n'

'\tTRUE equ 0x3f800000\n'

'\tTMP dd 0.0\n'

'\tRES dd ZEROS\n')

for el in self.declare\_rpn:

if el == "," or el == "dim":

continue

elif el in self.TYPES:

for i in range(len(self.variables)):

if self.variables[i][1] is None:

self.variables[i][1] = el

res += "\t" + self.variables[i][0] + " dd ZEROS\n"

else:

self.variables.append([el, None])

self.var\_names.append(el)

res += "\nsection .text\nglobal main\nmain:\n\tmov ebp, esp\n"

for i in range(len(self.main\_rpn)):

el = str(self.main\_rpn[i])

if el == "+":

res += self.two\_operands\_calc()

res += "\tfadd\n"

self.calced += 1

res += "\tfstp dword[RES]\n"

elif el == "-":

res += self.two\_operands\_calc()

res += "\tfsub\n"

self.calced += 1

res += "\tfstp dword[RES]\n"

elif el == "\*":

res += self.two\_operands\_calc()

res += "\tfmul\n"

self.calced += 1

res += "\tfstp dword[RES]\n"

elif el == "/":

res += self.two\_operands\_calc()

res += "\tfdiv\n"

self.calced += 1

res += "\tfstp dword[RES]\n"

elif el == "ass":

if self.calced:

res += "\tmov eax, dword[RES]\n"

res += "\tmov dword["+self.stack.pop()+"], eax\n"

self.calced = False

else:

second = self.stack.pop()

first = self.stack.pop()

if second in self.var\_names:

res += "\tmov eax, dword[" + second + "]\n"

res += "\tmov dword["+first+"], eax\n"

elif second == "False":

res += "\tmov dword["+first+"], FALSE\n"

elif second == "True":

res += "\tmov dword["+first+"], TRUE\n"

else:

res += "\tmov dword["+first+"], 0x"+single(second).hex()[0]+"\n"

elif el == ">":

res += self.two\_operands\_calc()

res += "\tfcomi\n"

res += "\tjb label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tmov dword[RES], FALSE\n"

res += "\tjmp label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "label"+str(self.labels\_count-2)+":\n"

res += "\tmov dword[RES], TRUE\n"

res += "label"+str(self.labels\_count-1)+":\n"

self.calced = True

elif el == ">=":

res += self.two\_operands\_calc()

res += "\tfcomi\n"

res += "\tjbe label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tmov dword[RES], FALSE\n"

res += "\tjmp label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "label"+str(self.labels\_count-2)+":\n"

res += "\tmov dword[RES], TRUE\n"

res += "label"+str(self.labels\_count-1)+":\n"

self.calced = True

elif el == "=":

res += self.two\_operands\_calc()

res += "\tfcomi\n"

res += "\tjz label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tmov dword[RES], FALSE\n"

res += "\tjmp label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "label"+str(self.labels\_count-2)+":\n"

res += "\tmov dword[RES], TRUE\n"

res += "label"+str(self.labels\_count-1)+":\n"

self.calced = True

elif el == "<":

res += self.two\_operands\_calc()

res += "\tfcomi\n"

res += "\tja label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tmov dword[RES], FALSE\n"

res += "\tjmp label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "label"+str(self.labels\_count-2)+":\n"

res += "\tmov dword[RES], TRUE\n"

res += "label"+str(self.labels\_count-1)+":\n"

self.calced = True

elif el == "<=":

res += self.two\_operands\_calc()

res += "\tfcomi\n"

res += "\tjae label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tmov dword[RES], FALSE\n"

res += "\tjmp label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "label"+str(self.labels\_count-2)+":\n"

res += "\tmov dword[RES], TRUE\n"

res += "label"+str(self.labels\_count-1)+":\n"

self.calced = True

elif el == "!=":

res += self.two\_operands\_calc()

res += "\tfcomi\n"

res += "\tjnz label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tmov dword[RES], FALSE\n"

res += "\tjmp label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "label"+str(self.labels\_count-2)+":\n"

res += "\tmov dword[RES], TRUE\n"

res += "label"+str(self.labels\_count-1)+":\n"

self.calced = True

elif el == "and":

second = self.stack.pop()

first = self.stack.pop()

if first in self.var\_names:

res += "\tmov eax, dword[" + first + "]\n"

else:

res += "\tmov eax, 0x" + single(first).hex()[0] + "\n"

if second in self.var\_names:

res += "\tand eax, dword[" + second + "]\n"

else:

res += "\tand eax, 0x" + single(second).hex()[0] + "\n"

res += "\tmov dword[RES], eax\n"

self.calced = True

elif el == "or":

second = self.stack.pop()

first = self.stack.pop()

if first in self.var\_names:

res += "\tmov eax, dword[" + first + "]\n"

else:

res += "\tmov eax, 0x" + single(first).hex()[0] + "\n"

if second in self.var\_names:

res += "\tor eax, dword[" + second + "]\n"

else:

res += "\tor eax, 0x" + single(second).hex()[0] + "\n"

res += "\tmov dword[RES], eax\n"

self.calced = True

elif el == "output":

out\_vars = []

count = 0

prev = self.stack.pop()

while prev == ",":

count += 1

prev = self.stack.pop()

out\_vars.append(prev)

for \_ in range(count):

out\_vars.append(self.stack.pop())

for var in out\_vars[::-1]:

res += "\tPRINT\_HEX 4, "+var+"\n\tNEWLINE\n"

elif el == "read":

out\_vars = []

count = 0

prev = self.stack.pop()

while prev == ",":

count += 1

prev = self.stack.pop()

out\_vars.append(prev)

for \_ in range(count):

out\_vars.append(self.stack.pop())

for var in out\_vars[::-1]:

res += "\tGET\_HEX 4, "+var+"\n"

elif el[0] == "[" and (self.main\_rpn[i+1] == "if" or self.main\_rpn[i+1] == "elseif"):

res += "\tfinit\n"

res += "\tfld dword[RES]\n"

res += "\tmov dword[TMP], 0x" + single(str(0)).hex()[0] + "\n\tfld dword[TMP]\n"

res += "\tfcomi\n"

res += "\tjnz label"+str(self.labels\_count)+"\n"

self.labels\_count += 1

res += "\tjmp "+el[1:-1:]+"\n"

res += "label"+str(self.labels\_count-1) + ":\n"

self.calced = False

elif el[0] == "[" and self.main\_rpn[i+1] == "for":

second = self.stack.pop()

first = self.stack.pop()

res += "\tfinit\n"

res += "\tfld dword["+first+"]\n"

if second in self.var\_names:

res += "\tfld dword["+second+"]\n"

else:

res += "\tmov dword[TMP], 0x" + single(second).hex()[0] + "\n\tfld dword[TMP]\n"

res += "\tfcomi\n"

res += "\tjz " + el[1:-1:] + "\n"

elif el[0] == "[" and self.main\_rpn[i+1] == "while":

res += "\tfinit\n"

res += "\tfld dword[RES]\n"

res += "\tmov dword[TMP], 0x" + single(str(0)).hex()[0] + "\n\tfld dword[TMP]\n"

res += "\tfcomi\n"

res += "\tjz "+el[1:-1:]+"\n"

elif el[0] == "[":

res += "\tjmp "+el[1:-1:]+"\n"

elif el[0] == "(":

res += el[1:-1:] + ":\n"

else:

self.stack.append(el)

res += "\tret\n"

return res

**# gui.py**

import PySimpleGUI as sg

import io

import os

import lexer

import matrix

import parser

import semantic

import RPN

import ASM

layout = [[sg.Text("Choose a file: "), sg.Input(), sg.FileBrowse(key="-IN-FILE-"), sg.Button("Load"), sg.Button('Compile', size=(12, 2), key="\_COMPILE\_"), sg.Button('Build EXE', size=(12, 2), key="\_BUILD\_")],

[sg.Text("Source code:", size=(46, 1)), sg.Text("Lexemes:", size=(45, 1)), sg.Text("Reverse Polish notation:", size=(45, 1)), sg.Text("Assembler (NASM):", size=(45, 1)),],

[sg.Multiline(s=(50, 30), key="-INPUT-"), sg.Multiline(s=(50, 30), disabled=True, key="-LEXER-"), sg.Multiline(s=(50, 30), disabled=True, key="-RPN-"), sg.Multiline(s=(50, 30), disabled=True, key="-ASM-")],

[sg.Text("Errors:")],

[sg.Multiline(s=(50, 7), key="-ERROR-", disabled=True)]]

window = sg.Window('Compiler', layout)

def do\_compile(stream):

lex = lexer.Lexer(stream)

operators = []

operators\_semantic = []

operator\_values = []

res = ""

err = ""

while lex.symbol != lexer.Lexer.EOF:

lex.next\_token()

if lex.error:

break

operators.append(lexer.decrypt\_to\_operators[lex.symbol])

if lex.symbol == lexer.Lexer.ID or lex.symbol == lexer.Lexer.NUM or lex.symbol == lexer.Lexer.REAL or lex.symbol == lexer.Lexer.BOOL\_VAL:

operators\_semantic.append(semantic.Operator(lex.symbol, lexer.decrypt\_to\_operators[lex.symbol], lex.value))

operator\_values.append([lexer.decrypt[lex.symbol], lex.value])

else:

operators\_semantic.append(semantic.Operator(lex.symbol, lexer.decrypt\_to\_operators[lex.symbol], None))

operator\_values.append([lexer.decrypt[lex.symbol]])

if lex.symbol == lexer.Lexer.ID:

res += lexer.decrypt\_to\_lexemes[lex.symbol] + ": " + str(lex.value) + "\n"

elif lex.symbol == lexer.Lexer.ID or lex.symbol == lexer.Lexer.NUM or lex.symbol == lexer.Lexer.REAL:

res += "CONSTANT: " + lexer.decrypt\_to\_lexemes[lex.symbol] + "," + str(lex.value) + "\n"

elif lex.symbol in lexer.Lexer.SYMBOLS.values() or lex.symbol == lexer.Lexer.EOF:

res += "SYMBOL: " + lexer.decrypt\_to\_lexemes[lex.symbol] + "\n"

else:

res += "WORD: " + lexer.decrypt\_to\_lexemes[lex.symbol] + "\n"

if lex.error:

window["-ERROR-"].update(lex.error\_msg)

return

window["-LEXER-"].update(res)

err += lex.error\_msg + "\n"

stream.close()

source\_path = "grammar.txt"

stream = open(source\_path, 'r')

gen = matrix.MatrixGenerator(stream)

gen.generate()

if gen.error:

window["-ERROR-"].update(gen.error\_msg)

return

err += gen.error\_msg + "\n"

stream.close()

rules = gen.rules

for rule\_name in rules.keys():

for rule in rules[rule\_name]:

for i in range(len(rule)):

if rule[i] not in gen.T:

rule[i] = "E"

pars = parser.Parser(gen.operator\_matrix, operators, rules)

pars.check()

if pars.error:

window["-ERROR-"].update(pars.error\_msg)

return

err += pars.error\_msg + "\n"

sem = semantic.Semantic(operators\_semantic)

sem.check()

if sem.error:

window["-ERROR-"].update(sem.error\_msg)

return

err += sem.error\_msg + "\n"

rpn = RPN.RPN(operator\_values)

declare\_res, res = rpn.convert()

if rpn.error:

window["-ERROR-"].update(rpn.error\_msg)

return

res\_string = ""

for el in declare\_res:

res\_string += str(el) + " "

for el in res:

res\_string += str(el) + " "

window["-RPN-"].update(res\_string)

err += rpn.error\_msg + "\n"

asm\_gen = ASM.ASM(declare\_res, res)

res\_string = asm\_gen.generate()

if asm\_gen.error:

window["-ERROR-"].update(asm\_gen.error\_msg)

return

window["-ASM-"].update(res\_string)

err += asm\_gen.error\_msg + "\n"

window["-ERROR-"].update(err)

while True:

event, values = window.read()

if event == sg.WINDOW\_CLOSED:

break

if event == "\_COMPILE\_":

do\_compile(io.StringIO(values["-INPUT-"]))

elif event == "Load":

if values["-IN-FILE-"] != "":

file = open(values["-IN-FILE-"])

window["-INPUT-"].update(file.read())

elif event == "\_BUILD\_":

asm\_file = open("asm.asm", "w")

asm\_file.write(values["-ASM-"])

asm\_file.close()

os.system('.\\NASM\\nasm.exe --gprefix \_ -f win32 asm.asm -o asm.obj')

os.system(".\MinGW\\bin\gcc.exe asm.obj .\\NASM\macro.o -g -o asm.exe -m32")

window.close()