|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | | | | | |
|  | | | | | |  | | |  | | | |
| ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | | | | | | | |
|  | | | | |  | | | | |  | | |
|  | | ОТЧЕТ  По разработке компилятора для языка Pascal  Вариант 2 | | | | | | | | |  | |
|  | | | |  | | | | | |  | | |
|  | Работу выполнил  студент гр. ПМИ-3  Антонов А.А.  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | | | |  | Проверил  Преподаватель кафедры МОВС  Пономарев Ф.А.  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | |  |
|  | | | | | | | | | | | | |
|  | | | Пермь 2022 | | | | | | |  | | |

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc97384054)

[1 Постановка задачи 3](#_Toc97384055)

[2 Структура компилятора 5](#_Toc97384056)

[3 Описание языка Pascal 8](#_Toc97384057)

[4 Модуль ввода-вывода 10](#_Toc97384058)

[4.1 Модуль ввода 10](#_Toc97384059)

[4.1.1 Проектирование 10](#_Toc97384060)

[4.1.2 Тестирование 10](#_Toc97384061)

[4.2 Модуль вывода 14](#_Toc97384062)

[5 Лексический анализатор 15](#_Toc97384063)

[5.1 Проектирование 15](#_Toc97384064)

[5.2 Токены 15](#_Toc97384065)

[6 Синтаксический анализатор 17](#_Toc97384066)

[7 Семантический анализатор 18](#_Toc97384067)

[8 Генерация кода 19](#_Toc97384068)

1 Постановка задачи

Глобальное задание: написать компилятор для подмножества языка Паскаль. Задание разбивается на отдельные этапы:

1. Модуль ввода-вывода (8 баллов, оценивается совместно с лексическим анализатором).
2. Лексический анализатор (12 баллов, оценивается совместно с модулем ввода-вывода).
3. Синтаксический анализатор (12 баллов) с нейтрализацией синтаксических ошибок (8 баллов).
4. Семантический анализатор с нейтрализацией семантических ошибок (20 баллов).
5. Генерация кода (25 баллов).

Для получения минимального проходного балла необходимо реализовать указанные этапы для подмножества языка Паскаль, описанного далее в разделе «*Общая минимальная часть*».

Для получения 70% баллов за анализаторы и 100% баллов за генерацию кода необходимо дополнительно реализовать конструкции, описанные далее в разделе «*Общая дополнительная часть*».

Для получения 100% баллов за анализаторы необходимо дополнительно реализовать анализ конструкций, описанных далее в разделе «*Индивидуальная часть*», в соответствии с заданным вариантом.

В случае не сдачи студентом задания по генерации кода на последнем лабораторном занятии (или ранее), данное задание для данного студента заменяется теоретическим экзаменом, который оценивается исходя из максимума в 25 баллов.

Замена задания по генерации на теоретический экзамен возможна также в случае, если студента не устраивают баллы, полученные за задание по генерации. В этом случае набранные баллы обнуляются, и студент сдает экзамен на тех же условиях, что и студенты, не сдававшие задание по генерации кода вообще.

*Общая минимальная часть.* Основные разделы программы: раздел описания переменных, раздел операторов. Переменные стандартных типов (Boolean, integer, real, char). Числовые константы. Арифметическое выражение (в выражении допустимы только константы, переменные, операции +, –, \*, / и скобки). Оператор присваивания и составной оператор.

*Общая дополнительная часть.* Раздел описания типов. Выражение (полностью, включая арифметические, логические операции, сравнения и т.д., но только над константами и простыми переменными (не индексированные, не поля записи, не указатели)). Условный оператор (if). Оператор цикла с предусловием (while).

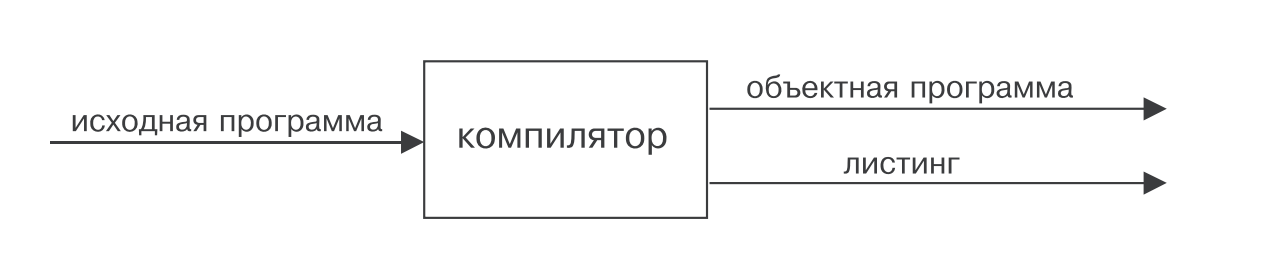
*Индивидуальная часть.*

Описание функций. Вызов функции в выражении. Оператор выбора (case).

2 Структура компилятора

Компилятор – это программа, которая переводит программу на языке высокого уровня в эквивалентную программу на другом (объектном) языке. Обычно также выдает листинг, содержащий текст исходной программы и сообщения обо всех обнаруженных ошибках.

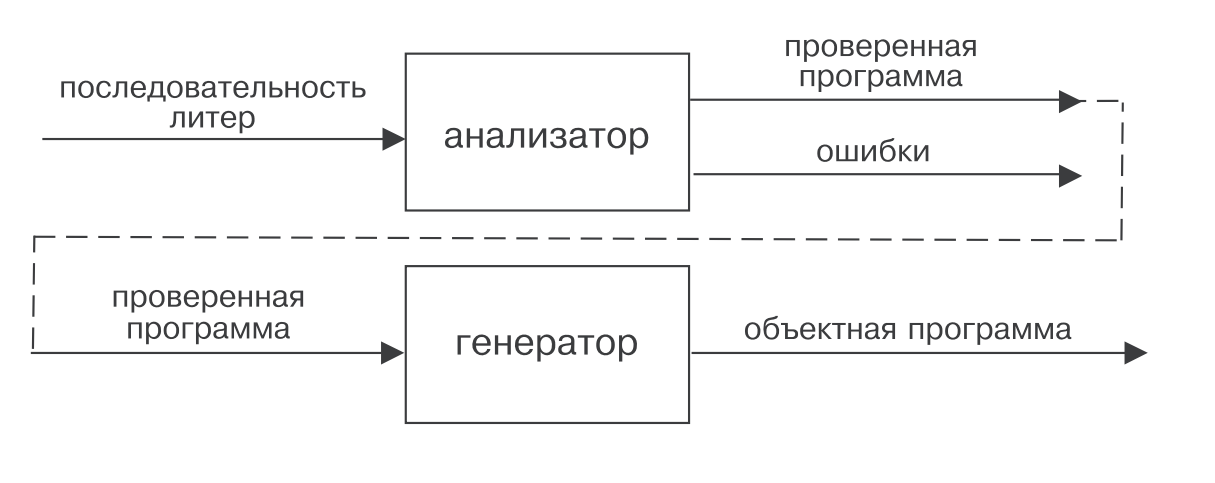
Компилятор можно представить в виде следующей схемы:



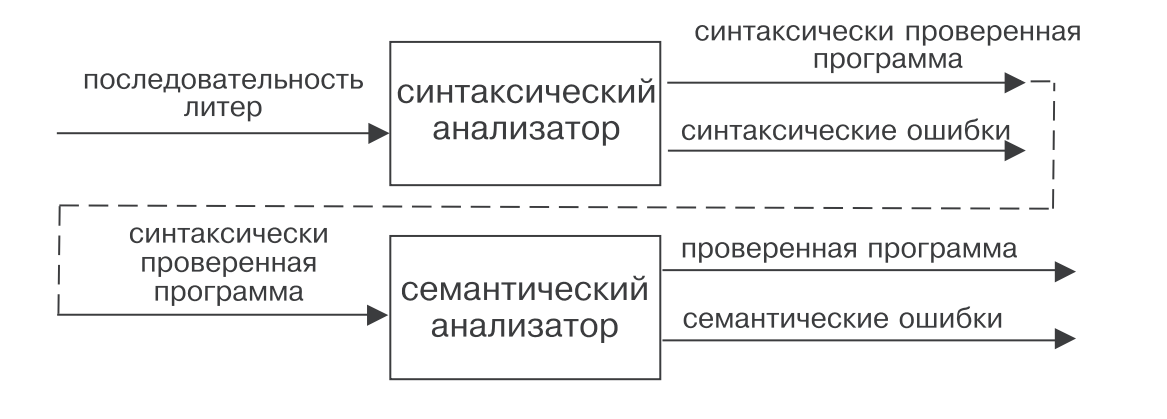
Работу компилятора можно разделить на 2 этапа:

1. Анализ – определение правильности исходной программы и формирование (в случае необходимости) сообщений об ошибках;
2. Синтез – генерация объектной программы; этот этап выполняется для программ, не содержащих ошибок.

Таким образом компилятор разбивается на следующие модули:



Анализатор в свою очередь можно разбиваться на синтаксический и семантический анализаторы:



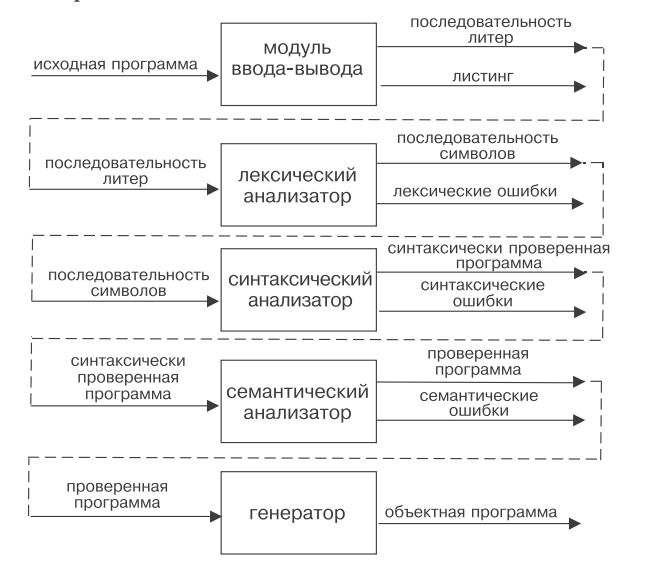
Синтаксический анализатор проверяет, удовлетворяет ли программа формальным правилам. Назначение же семантического анализатора состоит в том, чтобы выяснить, не нарушены ли неформальные правила описания языка.

Дальнейшее разбиение на модули выполняется внутри синтаксического анализатора.

Первый модуль – Лексический анализатор (Lexer) просматривает символы исходной программы и строит символы (лексемы) – идентификаторы, ключевые слова и константы.

Второй модуль – Синтаксический анализатор выполняет синтаксический анализ последовательности символов. На этом этапе символы рассматриваются как неделимые, и их представление как последовательности литер несущественно.

В результате получаем следующую структуру компилятора:



Для реализации компилятора необходимо реализовать каждый из модулей, представленных выше.

3 Описание языка Pascal

Будем рассматривать 3 типа токенов: ключевые слова, идентификаторы и константы.

Ключевые слова – это слова ЯП, которые имеют специальное, раз и навсегда закрепленное за ними значение. В программе нельзя использовать идентификаторы, совпадающие с ключевыми словами.

К ключевым словам будем также относить знаки операций, ограничители и специальные символы.

Идентификаторы – имена программ, модулей, функций, типов, переменных и констант. Идентификаторы начинаются с буквы (A-Z, a-z) или с символа нижнего подчеркивания и могут содержать буквы, символ нижнего подчеркивания и цифры. В языке Pascal идентификаторы нечувствительные к регистру.

Константы – это числа или строки, которые встречают в выражениях.

Ключевые слова:

|  |  |
| --- | --- |
| Лексема | Описание |
| program | Название программы |
| begin | Составной оператор |
| end | Составной оператор |
| var | Описание переменной |
| integer |  |
| real |  |
| string |  |
| boolean |  |
| type |  |
| function |  |
| if |  |
| then |  |
| else |  |
| while |  |
| do |  |
| and |  |
| or |  |
| xor |  |
| not |  |
| @ | Оператор адреса |
| ^ | Оператор переадресации указателя |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| ( | Открывающая скобка |
| ) | Закрывающая скобка |
| := | Оператор присваивания |
| . |  |
| , |  |
| : |  |
| ; |  |
| = |  |
| < |  |
| > |  |
| <= |  |
| >= |  |
| <> |  |
|  |  |
|  |  |
|  |  |
|  |  |

4 Модуль ввода-вывода

4.1 Модуль ввода

4.1.1 Проектирование

Для реализации модуля ввода-вывода используется класс CInputOutput, который содержит следующие поля и методы:

class CInputOutput {

private:

int currentLine;

int currentLinePosition;

std::ifstream inputFileStream;

std::ofstream outputFileStream;

public:

CInputOutput(std::string filePath, std::string outputPath);

std::unique\_ptr<CLiteral> NextChar();

void WriteError(CError error);

};

inputFileStream и outputFileStream хранят ссылки на потоки файлового ввода и вывода из стандартной библиотеки соответственно.

В currentLine и currentLinePosition поддерживаются номер строки и позиция последнего считанного символа в этой строке соответственно.

В качестве аргументов конструктора класса принимаются строки с путем к исходному коду программы на языке Pascal и путем к файлу в который будут выводится ошибки в текстовом виде.

Метод NextChar возвращает указатель на объект класса CLiteral.

Класс CLiteral инкапсулирует в себе считываемые символы и их позицию.

class CLiteral {

public:

int lineNumber, linePosition;

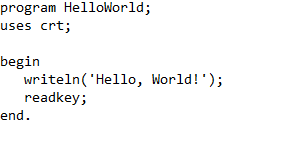
char c;

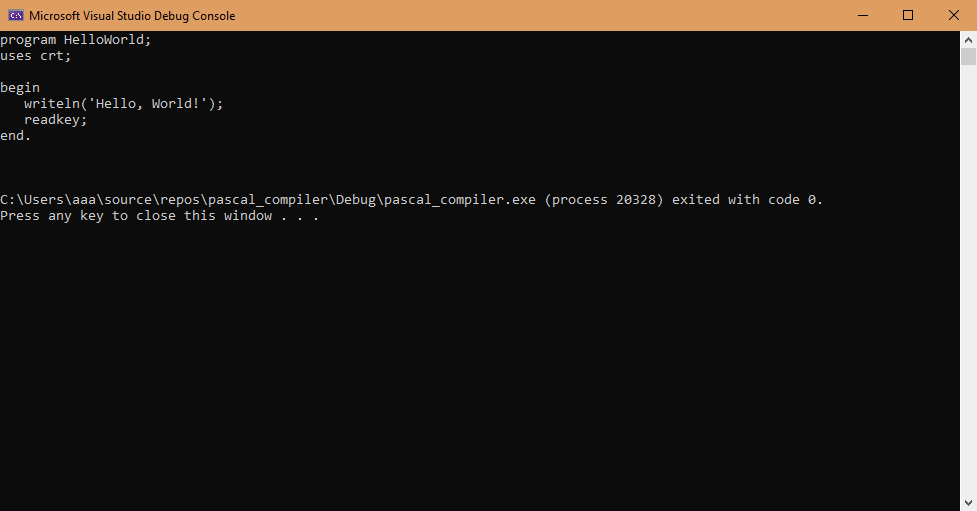
CLiteral(char \_c, int \_lineNumber, int \_linePosition);

};

4.1.2 Тестирование

Для тестирования рассмотрим пример паскаль программы и посимвольно выведем все, что возвращает нам модуль ввода.





Текст программы, который был выведен в консоль соответствует исходному.

Протестируем корректность сохранения позиции символа. Выведем каждый возвращаемый модулем символ вместе с его позицией в исходном тексте программы.





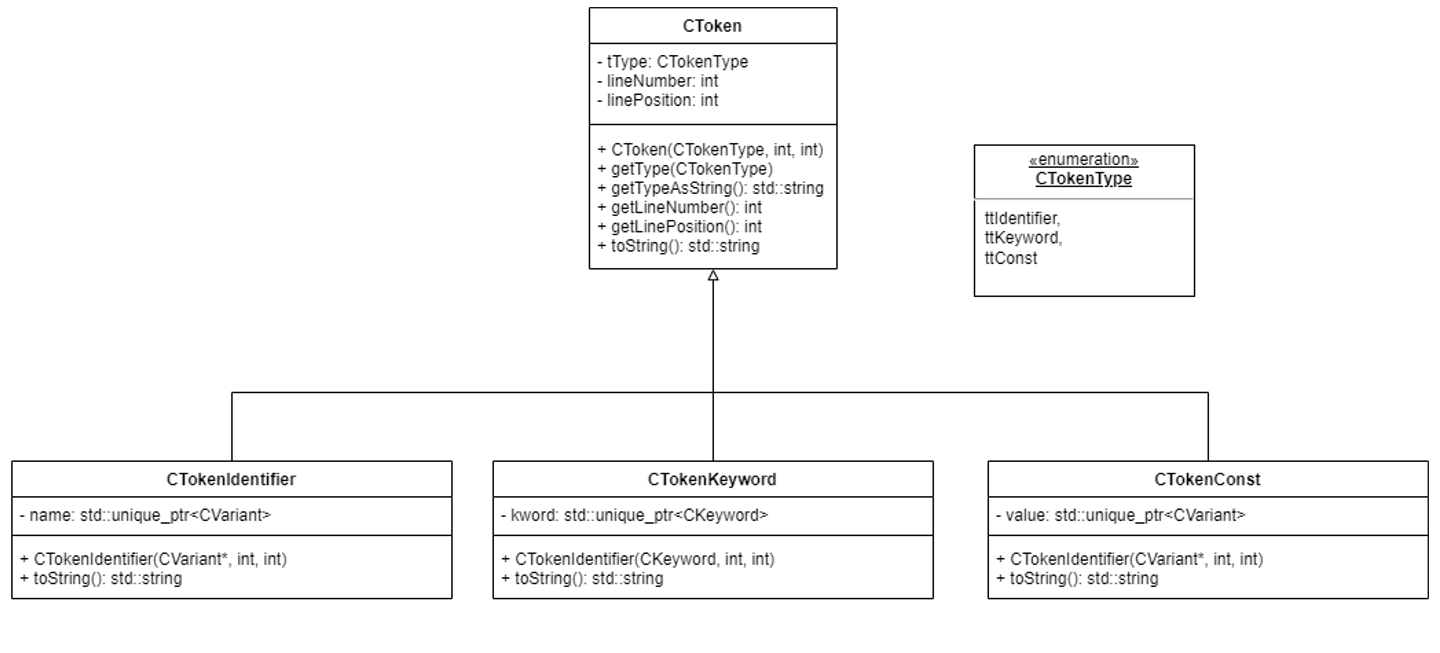


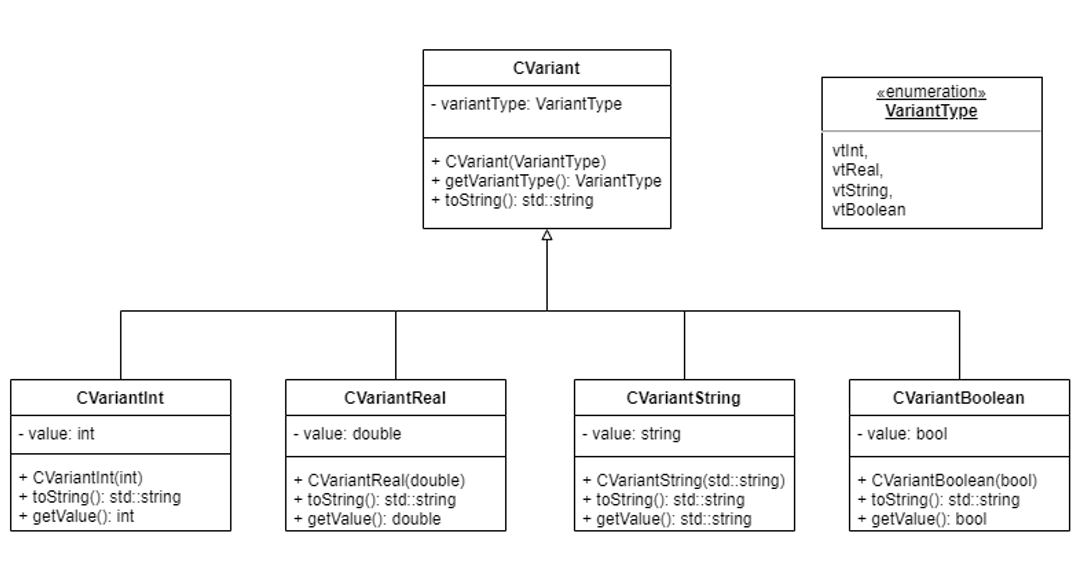
Позиции символов соответствуют позициям в исходном тексте программы на языке Pascal.

1. Лексический анализатор

4.1 Проектирование

4.1.1 Токены





4.1.2 Обработка ошибок

4.1.3 Лексер

* 1. Тестирование

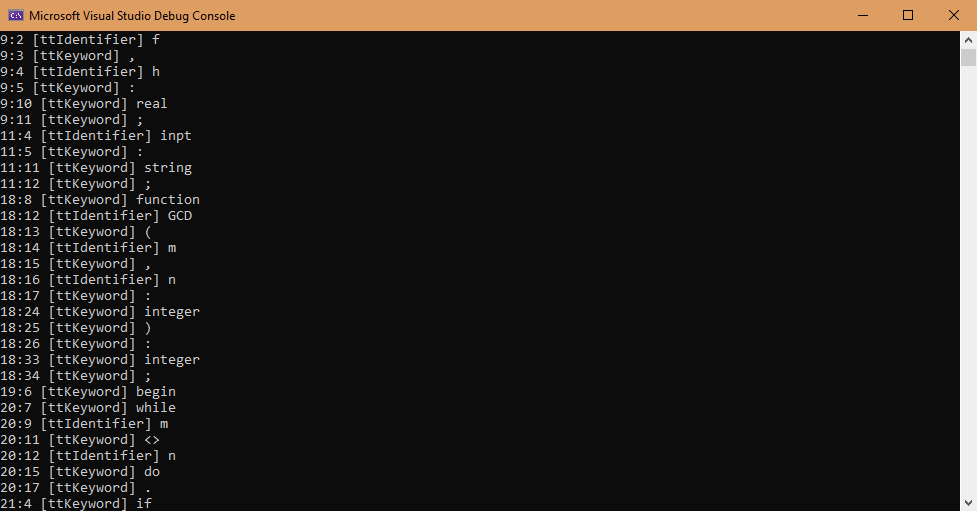
Для тестирования лексического анализатора возьмём комплексную программу со всеми ключевыми словами, комментариями и идентификаторами.

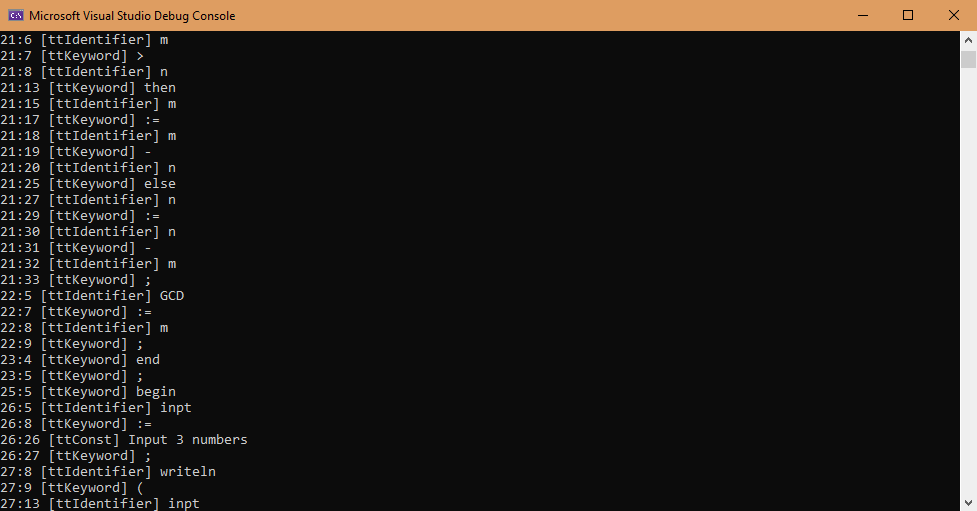
Изображение выглядит как текст

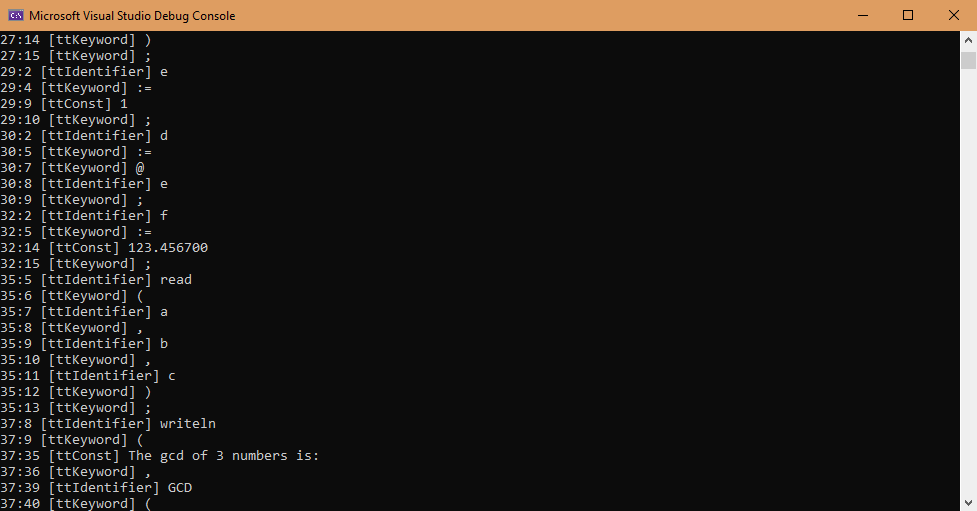
Автоматически созданное описание

Результат выполнения программы:













6 Синтаксический анализатор

6.1 Использованные БНФ

Ниже представлены использованные БНФ для языка Pascal.

<program> ::= program <identifier>;<block>.|<block>.

<block> ::= <typeDeclarationPart><varDeclarationPart><funcionDeclarationPart><statementPart>

<typeDeclarationPart> ::= <empty> | type <typeDeclaration>; {<typeDeclaration>;}

<typeDeclaration> ::= <identifier> = <type>

<type> ::= <identifier>

<varDeclarationPart>::= <empty> | var <varDeclaration>; {<varDeclaration>;}

<varDeclaration> ::= <identifier> {,<identifier>} : <type>

<funcionDeclarationPart> ::= {<funcionDeclaration>;}

<funcionDeclaration> ::= <functionHeading> <block>

<functionHeading> ::= function <identifier>:<type>; |

function <identifier>(<formalParameterSection>{;<formalParameterSection>}):<type>;

<formalParameterSection> ::= <empty> | <parameterGroup> | var <parameterGroup>

<parameterGroup> ::= <identifier>{,<identifier>}: <type>

<statementPart> ::= begin <compoundStatement> end

<compoundStatement> ::= <empty> | <statement>{;<statement>}

<statement> ::= <simpleStatement>|<structuredStatement>

<simpleStatement> ::= <assignmentStatement>|<procedureStatement>

<assignmentStatement> ::= <variable> := <expression>

<variable> ::= <identifier>

<expression> ::= <simpleExpression>{<relationalOperator><simpleExpression>}

<relationalOperator> ::= <|>|>=|<=|=|<>

<simpleExpression> ::= <term>{<addingOperator><term>}

<addingOperator> ::= + | - | or | xor

<term> ::= <factor>{<multiplyingOperator><factor>}

<multiplyingOperator> ::= \* | / | and

<factor> ::= <variable> | <unsignedConst> | (<expression>) | <functionDesignator> | <unaryOperator><factor>

<unaryOperator> ::= + | - | not | @

<unsignedConst> ::= <unsignedNumber> | <stringConst>

<unsignedNumber> ::= <unsignedInteger> | <unsignedReal>

<functionDesignator> ::= <identifier>() | <identifier>(<actualParameter>{,<actualParameter>})

<procedureStatement> ::= <identifier>() | <identifier>(<actualParameter>{,<actualParameter>})

<actualParameter> ::= <expression>|<empty>

<empty> ::=

<structuredStatement> ::= <ifStatement>|<whileStatement>|<caseStatement>

<ifStatement> ::= if <expression> then <statement> | if <expression> then <statementPart> | if<expression> then <statement> else <statement>

| if<expression> then <statementPart> else <statementPart> | if<expression> then <statement> else <statementPart> | if<expression> then <statementPart> else <statement>

<whileStatement> ::= while <expression> do <statement> | while <expression> do <statementPart>

<caseStatement> ::= case <expression> of <case list element> {; <case list element> } end

<caseListElement> ::= <empty> | <unsignedConst>{, <unsignedConst> } : <statement> | <unsignedConst>{, <unsignedConst> } : <statementPart>

6.2 Общее описание

Синтаксический анализатор представлен в виде класса CSyntax.

Объекты данного класса содержат два поля - указатель на объект класса лексического анализатора и указатель на текущий рассматриваемый токен.

Для каждой БНФ в классе реализован метод, который ей соответствует по логике, т.е. вызывает методы других БНФ или обрабатывает терминальные символы. Реализация каждого метода сделана учитывая следующее условие: если мы находимся в начале нового правила, то текущий токен - всегда самый первый токен в левой части БНФ.

6.3 Нейтрализация ошибок

7 Семантический анализатор

8 Генерация кода