Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ национальный исследовательский  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ**

**Н. Г. ЧЕРНЫШЕВСКОГО»**

Факультет компьютерных наук и информационных технологий

**Основы трансляции языков программирования.**

Лабораторная работа №1. Построение лексического анализатора.

Студента 4 курса 441 группы

направления 02.03.03—Математическое обеспечение и администрирование

информационных систем

факультета КНиИТ

Поповича Алексея Сергеевича

Саратов 2024

# Задание на лабораторную работу.

Составить автоматную грамматику и на ее основе реализовать лексический анализатор языка, цепочки которого имеют вид, указанный в задании. Лексический анализатор должен преобразовывать исходный текст в последовательность лексем. По результатам работы анализатора должны формироваться таблицы идентификаторов и констант.

# Описание цепочек анализируемого языка.

Лексический анализатор, разработанный в рамках данной лабораторной работы, способен распознавать следующие типы цепочек, которые состоят из базовых элементов языка:

**Цикл:**

Описание: Конструкция цикла с предусловием.

Формат: do while <логическое выражение> <операторы> loop

Пример: do while x < 10 and y != 5 z = x \* 2 + y / 3 loop

**Условие:**

Описание: Логическое выражение, которое может состоять из одного или нескольких сравнений, объединенных логическими операциями.

Формат: <сравнение> | <условие> <логическая операция> <сравнение>

Пример: x < 10 and y != 5

**Сравнение:**

Описание: Операция сравнения между двумя операндами.

Формат: <операнд> | <операнд> <операция сравнения> <операнд>

Пример: x < 10

**Операция сравнения:**

Описание: Операторы, используемые для сравнения двух операндов.

Формат: < | <= | != | == | > | >=

Пример: a <= b

**Операнд:**

Описание: Операндом может быть либо идентификатор, либо числовая константа.

Формат: <идентификатор> | <константа>

Пример: a, 123

**Логическая операция:**

Описание: Логические операторы для объединения нескольких условий.

Формат: and | or

Пример: a < b and b <= c

**Оператор:**

Описание: Присваивание значения идентификатору с использованием арифметических операций.

Формат: <идентификатор> = <арифметическое выражение>

Пример: b = b + c - 20

**Арифметическое выражение:**

Описание: Выражение, состоящее из операндов и арифметических операций.

Формат: <операнд> | <арифметическое выражение> <арифметическая операция> <операнд>

Пример: b + c - 20

**Арифметическая операция:**

Описание: Операции умножения, деления, сложения и вычитания.

Формат: + | - | \* | /

Пример: +, -,\*, /

**Примечание:**

Лексический анализатор распознает ключевые слова, операторы сравнения, арифметические операторы, идентификаторы и числовые константы. Элементы языка могут быть разделены произвольным количеством пробельных символов, которые игнорируются при анализе.

# Таблица терминальных символов с подробным описанием.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Индекс | Символ | Категория | Тип | Комментарий |
| 0 | do | Ключевое слово | КлючевоеСлово\_Do | Стартовое слово |
| 1 | while | Ключевое слово | КлючевоеСлово\_While | Начало заголовка цикла |
| 2 | loop | Ключевое слово | КлючевоеСлово\_End | Конец тела цикла |
| 3 | and | Ключевое слово | КлючевоеСлово\_And | Логическая операция 'И' |
| 4 | or | Ключевое слово | КлючевоеСлово\_Or | Логическая операция 'ИЛИ' |
| 5 | ; | Ключевое слово | КонецПредусловияЦикла | Конец предусловия цикла |
| 6 | < | Специальный символ | ОператорСравнения | Операция сравнения 'меньше' |
| 7 | <= | Специальный символ | ОператорСравнения | Операция сравнения 'меньше или равно' |
| 8 | != | Специальный символ | ОператорСравнения | Операция сравнения 'неравно' |
| 9 | == | Специальный символ | ОператорСравнения | Операция сравнения 'равно' |
| 10 | = | Специальный символ | ОператорПрисваивания | Операция присваивания |
| 11 | + | Специальный символ | АрифметическийОператор | Операция сложения |
| 12 | - | Специальный символ | АрифметическийОператор | Операция вычитания |
| 13 | \* | Специальный символ | АрифметическийОператор | Операция умножения |
| 14 | / | Специальный символ | АрифметическийОператор | Операция деления |
| 15 | > | Специальный символ | ОператорСравнения | Операция сравнения 'больше' |
| 16 | >= | Специальный символ | ОператорСравнения | Операция сравнения 'больше или равно' |

Описание:

Ключевые слова: while, do, loop, and, or — используются для управления потоком выполнения программы.

Операции сравнения: < | <= | != | == | > | >= — для проверки условий.

Операторы присваивания и арифметические операторы: =, +, \*, / — для работы с выражениями.

# Описание основных типов лексем с использованием грамматик.

В данной работе лексический анализатор распознает следующие типы лексем:

**Ключевые слова:**

Описание: Ключевые слова языка, зарезервированные для управления потоком выполнения программы.

Тип лексемы: КлючевоеСлово\_While, КлючевоеСлово\_Do, КлючевоеСлово\_End, КлючевоеСлово\_And, КлючевоеСлово\_Or, КонецПредусловияЦикла

Правило грамматики:

<ключевое слово> ::= "while" | "do" | "loop" | "and" | "or" | ";"

**Идентификаторы:**

Описание: Идентификаторы, которые используются для обозначения переменных или функций в программе. Идентификаторы должны начинаться с буквы и могут содержать буквы и цифры.

Тип лексемы: Идентификатор

Правило грамматики:

<идентификатор> ::= <буква> {<буква> | <цифра>}

где <буква> — это любой символ латинского алфавита, а <цифра> — это цифры от 0 до 9.

**Константы**

Описание: Числовые константы (целые числа), состоящие только из цифр.

Тип лексемы: Константа

Правило грамматики:

<константа> ::= <цифра> {<цифра>}

где <цифра> — это цифры от 0 до 9.

**Операторы сравнения**

Описание: Операторы, которые используются для сравнения двух значений.

Тип лексемы: ОператорСравнения

Правило грамматики:

<оператор сравнения> ::= "<" | "<=" | "!=" | "==" | ">" | ">="

**Арифметические операторы**

Описание: Операторы, которые используются для выполнения арифметических операций (сложение и вычитание).

Тип лексемы: АрифметическийОператор

Правило грамматики:

<арифметическая операция> ::= "+" | "-" | "\*" | "/"

**Операторы присваивания**

Описание: Операторы, которые используются для присваивания значения переменной.

Тип лексемы: ОператорПрисваивания

Правило грамматики:

<оператор присваивания> ::= "="

# Описание грамматики автоматного языка в виде графа

**Состояния автомата:**

S — начальное состояние.

A — состояние для распознавания идентификаторов.

B — состояние для распознавания констант.

C1 — состояние для распознавания операторов сравнения начинающихся с <, >.

D — состояние для распознавания <=, >=.

C2 — состояние для распознавания операторов сравнения начинающихся с !.

C31 — состояние для распознавания !=.

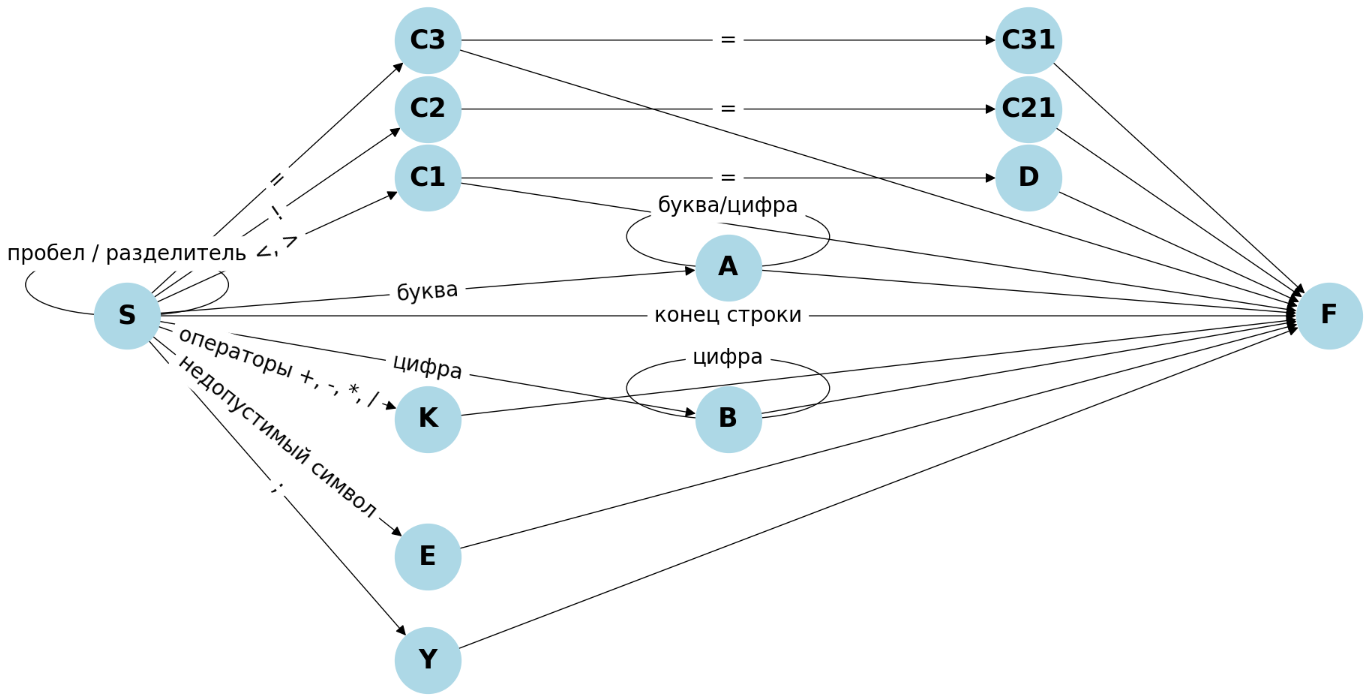
C3 — состояние для распознавания операторов сравнения начинающихся с =.

C31 — состояние для распознавания ==.

F — принимающее состояние (конец строки или успешное распознавание лексемы).

E — ошибка (недопустимый символ).

**Переходы автомата:**



# Описание основных алгоритмов и структур данных, используемых в программе

**А) Описание структур данных, используемых для представления последовательности лексем, таблиц идентификаторов и констант**

**Лексемы (Lexem):**

Структура: Класс Lexem представляет каждую лексему, распознанную лексическим анализатором. Он содержит следующие поля:

Value: строковое значение лексемы (например, "while", "a", "123").

Type: тип лексемы, представлен через enum LexemType, где указаны типы ключевых слов, операторов, идентификаторов и констант.

Position: позиция лексемы в исходной строке.

Использование: Лексемы собираются в список (List<Lexem>) после анализа входной строки и используются для дальнейшего синтаксического анализа или вывода.

**Таблица идентификаторов и констант (SymbolTable):**

Структура: Два словаря (Dictionary<string, int>) для хранения:

Идентификаторов (\_identifiers): сопоставляет строковое значение идентификатора с его порядковым номером.

Констант (\_constants): аналогичная структура для числовых констант.

Использование: При распознавании идентификаторов или констант они добавляются в таблицу с уникальным идентификатором (если они еще не были добавлены).

**Таблица терминальных символов (TerminalSymbolTable):**

Структура: Список объектов TerminalSymbol, каждый из которых описывает терминальный символ (например, "while", "+", "<") с указанием индекса, символа, категории, типа и комментария.

Использование: Используется для вывода всех возможных терминальных символов языка. Структура не участвует непосредственно в анализе, но служит для представления возможных символов.

**Б) Описание процедур и функций, отвечающих за работу с введенными структурами данных**

**Процедуры для работы с лексемами (класс Lexer):**

**Analyze():** Главная функция, которая выполняет лексический анализ входной строки. Она последовательно читает символы и формирует лексемы на основе правил:

* Пробелы и разделители игнорируются.
* Буквы распознаются как идентификаторы или ключевые слова.
* Цифры распознаются как константы.
* Символы <, =, > и т.д. распознаются как операторы сравнения.
* Символы +, - — как арифметические операторы.
* Неизвестные символы классифицируются как Unknown.

**ReadWhile():** Читает символы строки, пока они удовлетворяют заданному условию (например, пока символ — буква или цифра).

**IdentifyKeyword():** Определяет, является ли строка ключевым словом (например, "while", "do"). Если не является, строка считается идентификатором.

**Процедуры для работы с таблицей терминальных символов (класс TerminalSymbolTable):**

* GetTable(): Возвращает список всех терминальных символов.
* PrintTable(): Выводит таблицу терминальных символов в консоль.

**В) Описание алгоритма анализа автоматного языка**

Алгоритм лексического анализа реализован в классе Lexer и работает как конечный автомат, переходя из одного состояния в другое на основе прочитанных символов:

**Инициализация:**

Входная строка передается в конструктор класса Lexer, где сохраняется как \_input, а индекс текущей позиции устанавливается в 0.

**Цикл анализа:**

* Алгоритм начинается с начального состояния и перебирает символы входной строки.
* Для каждого символа проверяется, является ли он пробелом, буквой, цифрой, оператором или неизвестным символом.
* В зависимости от символа, автомат переходит в одно из состояний:
  + Идентификаторы и ключевые слова: если символ — буква, запускается процесс распознавания идентификатора или ключевого слова.
  + Константы: если символ — цифра, запускается процесс распознавания числовой константы.
  + Операторы: если символ — один из операторов сравнения или арифметических операций, он классифицируется как соответствующий оператор.
  + Неизвестные символы: если символ не соответствует ни одному правилу, он классифицируется как Unknown.

**Создание токенов:**

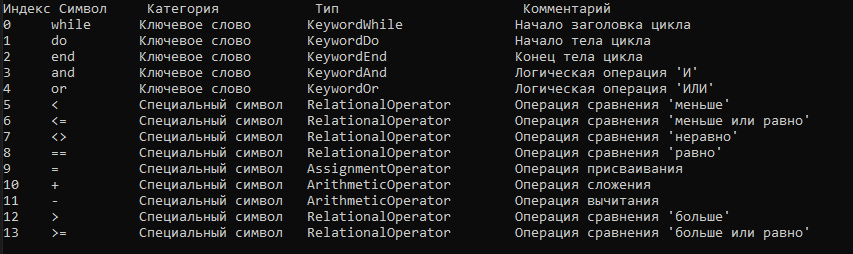
* Когда очередная лексема распознана, создается объект класса Lexem, содержащий значение лексемы, ее тип и позицию в строке.
* Все токены добавляются в список, который возвращается после завершения анализа.

**Завершение анализа:**

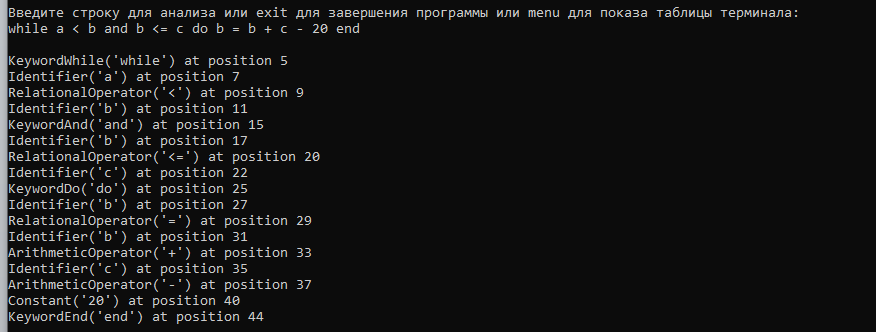
Когда все символы входной строки обработаны, цикл завершает работу, и возвращается список токенов.

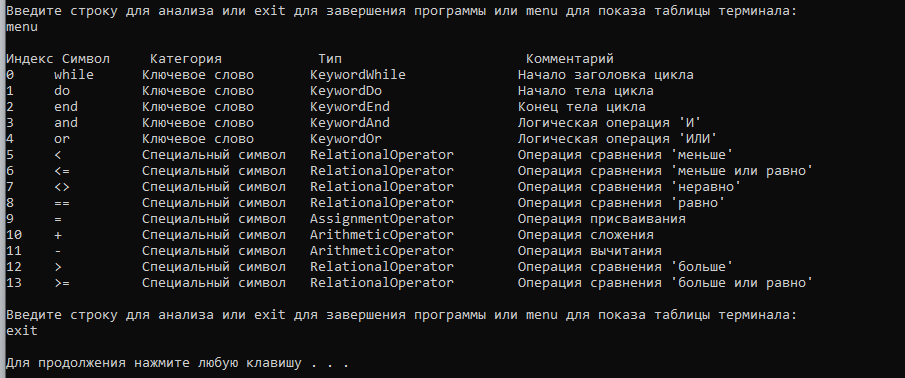
# Описание интерфейса пользователя программы.

Изначально выводятся все наши ключевые слова:



Дальше можно ввести 2 слова menu для показа таблицы терминала или exit для выхода. Либо ввести строку для анализа.





# Листинг программы

public class SymbolTable

{

public static List<Tuple<string, string, ТипЛексемы, string>> ТаблицаЗаданныхСимволов()

{

return new List<Tuple<string, string, ТипЛексемы, string>>

{

new Tuple<string, string, ТипЛексемы, string>("do", "Ключевое слово", ТипЛексемы.КлючевоеСлово\_Do, "Стартовое слово"),

new Tuple<string, string, ТипЛексемы, string>("while", "Ключевое слово", ТипЛексемы.КлючевоеСлово\_While, "Начало заголовка цикла"),

new Tuple<string, string, ТипЛексемы, string>("loop", "Ключевое слово", ТипЛексемы.КлючевоеСлово\_End, "Конец тела цикла"),

new Tuple<string, string, ТипЛексемы, string>("and", "Ключевое слово", ТипЛексемы.КлючевоеСлово\_And, "Логическая операция 'И'"),

new Tuple<string, string, ТипЛексемы, string>("or", "Ключевое слово", ТипЛексемы.КлючевоеСлово\_Or, "Логическая операция 'ИЛИ'"),

new Tuple<string, string, ТипЛексемы, string>(";", "Ключевое слово", ТипЛексемы.КонецПредусловияЦикла, "Конец предусловия цикла"),

new Tuple<string, string, ТипЛексемы, string>("<", "Специальный символ", ТипЛексемы.ОператорСравнения, "Операция сравнения 'меньше'"),

new Tuple<string, string, ТипЛексемы, string>("<=", "Специальный символ", ТипЛексемы.ОператорСравнения, "Операция сравнения 'меньше или равно'"),

new Tuple<string, string, ТипЛексемы, string>("!=", "Специальный символ", ТипЛексемы.ОператорСравнения, "Операция сравнения 'неравно'"),

new Tuple<string, string, ТипЛексемы, string>("==", "Специальный символ", ТипЛексемы.ОператорСравнения, "Операция сравнения 'равно'"),

new Tuple<string, string, ТипЛексемы, string>("=", "Специальный символ", ТипЛексемы.ОператорПрисваивания, "Операция присваивания"),

new Tuple<string, string, ТипЛексемы, string>("+", "Специальный символ", ТипЛексемы.АрифметическийОператор, "Операция сложения"),

new Tuple<string, string, ТипЛексемы, string>("-", "Специальный символ", ТипЛексемы.АрифметическийОператор, "Операция вычитания"),

new Tuple<string, string, ТипЛексемы, string>("\*", "Специальный символ", ТипЛексемы.АрифметическийОператор, "Операция умножения"),

new Tuple<string, string, ТипЛексемы, string>("/", "Специальный символ", ТипЛексемы.АрифметическийОператор, "Операция деления"),

new Tuple<string, string, ТипЛексемы, string>(">", "Специальный символ", ТипЛексемы.ОператорСравнения, "Операция сравнения 'больше'"),

new Tuple<string, string, ТипЛексемы, string>(">=", "Специальный символ", ТипЛексемы.ОператорСравнения, "Операция сравнения 'больше или равно'")

};

}

public static void ВывестиТаблицу()

{

Console.WriteLine("{0,-7} | {1,-20} | {2,-25} | {3,-40} |", "Символ", "Категория", "Тип", "Комментарий");

foreach (var символ in ТаблицаЗаданныхСимволов())

{

Console.WriteLine("{0,-7} | {1,-20} | {2,-25} | {3,-40} |",

символ.Item1, символ.Item2, символ.Item3, символ.Item4);

}

}

}

public class Лексема

{

public string Value { get; }

public ТипЛексемы Type { get; }

public int Position { get; }

public Лексема(string value, ТипЛексемы type, int position)

{

Value = value;

Type = type;

Position = position;

}

}

public class ЛексическийАнализ

{

private readonly string \_входнаяСтрока;

private int \_позиция;

public ЛексическийАнализ(string input)

{

\_входнаяСтрока = input;

\_позиция = 0;

}

public List<Лексема> Анализ()

{

var лексемы = new List<Лексема>();

while (\_позиция < \_входнаяСтрока.Length)

{

if (char.IsWhiteSpace(ТекущийСимвол()))

{

\_позиция++;

}

else if (IsLetter(ТекущийСимвол()))

{

var Идентификатор = ReadWhile(IsLetterOrDigit);

лексемы.Add(new Лексема(Идентификатор, IdentifyKeyword(Идентификатор), \_позиция));

}

else if (char.IsDigit(ТекущийСимвол()))

{

var Константа = ReadWhile(char.IsDigit);

лексемы.Add(new Лексема(Константа, ТипЛексемы.Константа, \_позиция));

}

else if (ТекущийСимвол() == ';')

{

var символ = ReadWhile(c => c == ';');

лексемы.Add(new Лексема(символ, ТипЛексемы.КонецПредусловияЦикла, \_позиция));

}

else if (ТекущийСимвол() == '<' || ТекущийСимвол() == '=' || ТекущийСимвол() == '>' || ТекущийСимвол() == '!')

{

var символ = ReadWhile(c => c == '<' || c == '=' || c == '!' || c == '>');

var операторыСравнения = SymbolTable.ТаблицаЗаданныхСимволов().GroupBy(i => i.Item3).FirstOrDefault(i => i.Key == ТипЛексемы.ОператорСравнения).Select(i => i.Item1).ToList();

if (символ == "=")

{

лексемы.Add(new Лексема(символ, ТипЛексемы.ОператорПрисваивания, \_позиция));

}

else if (операторыСравнения.Contains(символ))

{

лексемы.Add(new Лексема(символ, ТипЛексемы.ОператорСравнения, \_позиция));

}

else

{

лексемы.Add(new Лексема(символ, ТипЛексемы.Неизвестно, \_позиция));

}

}

else if (ТекущийСимвол() == '+' || ТекущийСимвол() == '-' || ТекущийСимвол() == '/' || ТекущийСимвол() == '\*')

{

var символ = ReadWhile(c => c == '+' || c == '-' || c == '/' || c == '\*');

var арифметическиеСравнения = SymbolTable.ТаблицаЗаданныхСимволов().GroupBy(i => i.Item3).FirstOrDefault(i => i.Key == ТипЛексемы.АрифметическийОператор).Select(i => i.Item1).ToList();

if (арифметическиеСравнения.Contains(символ))

{

лексемы.Add(new Лексема(символ, ТипЛексемы.АрифметическийОператор, \_позиция));

}

else

{

лексемы.Add(new Лексема(символ, ТипЛексемы.Неизвестно, \_позиция));

}

}

else

{

лексемы.Add(new Лексема(ТекущийСимвол().ToString(), ТипЛексемы.Неизвестно, \_позиция));

\_позиция++;

}

}

return лексемы;

}

private char ТекущийСимвол()

{

return \_входнаяСтрока[\_позиция];

}

private string ReadWhile(Func<char, bool> condition)

{

var start = \_позиция;

while (\_позиция < \_входнаяСтрока.Length && condition(\_входнаяСтрока[\_позиция]))

{

\_позиция++;

}

return \_входнаяСтрока.Substring(start, \_позиция - start);

}

private bool IsLetter(char ch) => char.IsLetter(ch);

private bool IsLetterOrDigit(char ch) => char.IsLetterOrDigit(ch);

private ТипЛексемы IdentifyKeyword(string value)

{

switch (value)

{

case "while": return ТипЛексемы.КлючевоеСлово\_While;

case "do": return ТипЛексемы.КлючевоеСлово\_Do;

case "loop": return ТипЛексемы.КлючевоеСлово\_End;

case "and": return ТипЛексемы.КлючевоеСлово\_And;

case "or": return ТипЛексемы.КлючевоеСлово\_Or;

default: return ТипЛексемы.Идентификатор;

}

}

}

public enum ТипЛексемы

{

КлючевоеСлово\_While,

КлючевоеСлово\_Do,

КлючевоеСлово\_End,

КлючевоеСлово\_And,

КлючевоеСлово\_Or,

КонецПредусловияЦикла,

ОператорСравнения,

АрифметическийОператор,

ОператорПрисваивания,

Идентификатор,

Константа,

Неизвестно

}

public class Program

{

static void Main(string[] args)

{

SymbolTable.ВывестиТаблицу();

Console.WriteLine();

while (true)

{

Console.WriteLine("Введите строку для анализа или exit для завершения программы или menu для показа таблицы терминала:");

string input = Console.ReadLine();

Console.WriteLine();

if (input == "exit")

{

break;

}

else if (input == "ter")

{

SymbolTable.ВывестиТаблицу();

}

else

{

var лексемы = new ЛексическийАнализ(input).Анализ();

foreach (var лексема in лексемы)

{

Console.ForegroundColor = ConsoleColor.Yellow;

if (лексема.Type == ТипЛексемы.Неизвестно)

{

Console.ForegroundColor = ConsoleColor.Red;

}

Console.Write($"{лексема.Type} : ");

Console.ResetColor();

Console.BackgroundColor = ConsoleColor.DarkBlue;

Console.ForegroundColor = ConsoleColor.Red;

Console.Write($"{лексема.Value}");

Console.ResetColor();

Console.Write($" столбец ");

Console.WriteLine($"{лексема.Position}");

}

}

Console.WriteLine();

}

}

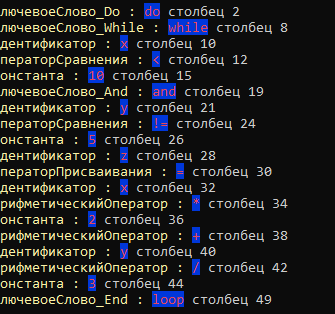
}

# Контрольный пример и результаты тестирования.

Пример:

do while x < 10 and y != 5 z = x \* 2 + y / 3 loop

Результат работы программы:



# Контрольные вопросы.

Контрольные вопросы:

1. Опишите основные этапы компиляции:
   * **Лексический анализ**: Разделение исходного кода на токены — минимальные значимые элементы. Например, int x = 42; превращается в токены: int (тип), x (идентификатор), = (оператор), 42 (константа), ; (конец выражения).
   * **Синтаксический анализ**: Проверка структуры кода согласно правилам языка, создание синтаксического дерева. Например, ошибка int x = 42 42; нарушает синтаксис.
   * **Семантический анализ**: Проверка логики программы на соответствие правилам языка. Например, int x = "hello"; вызовет ошибку типа данных.
   * **Генерация промежуточного кода**: Построение промежуточного представления, независимого от платформы.
   * **Оптимизация промежуточного кода**: Улучшение промежуточного кода для повышения эффективности.
   * **Генерация объектного кода**: Преобразование кода в машинный, пригодный для выполнения.
2. Дайте краткую характеристику этапа лексического анализа:
   * Лексический анализатор разбивает код на последовательности символов (лексемы), формируя токены для дальнейшего анализа.
3. Что такое лексема? Приведите примеры:
   * Лексема — минимальная значимая единица программы, такие как ключевые слова (if, for), идентификаторы (переменные), операторы (+, =).
4. С какими таблицами работает лексический анализатор?
   * Таблица символов (для переменных и функций), таблица ключевых слов, таблица литералов и таблица токенов.
5. Определите грамматику:
   * Грамматика — это формальное описание синтаксиса языка, определяющее правила для создания корректных выражений.
6. Определите автоматную грамматику:
   * Грамматика, описывающая язык, распознаваемый конечным автоматом (например, регулярная грамматика).
7. Назовите основные этапы создания анализатора по автоматной грамматике:
   * Определение правил грамматики, построение конечного автомата, определение состояний и переходов, создание программы для распознавания языка.
8. Какие способы задания автоматного языка существуют?
   * **Регулярные выражения**: Описание последовательностей символов.
   * **Контекстно-свободные грамматики**: Для сложных вложенных структур.
   * **Сети Петри**: Для параллельных процессов.
   * **Диаграммы переходов состояний**: Представление автоматов графически.
   * **Формальные грамматики**: Описание через правила (по классификации Хомского).
   * **Автоматы с магазинной памятью**: Для работы с контекстно-свободными языками.
9. Что такое конечный автомат?
   * Конечный автомат — устройство с конечным числом состояний, переходящее между ними по входным данным.
10. Определите детерминированную, недетерминированную и полнодетерминированную формы автоматной грамматики:
    * **Детерминированная**: Для каждого состояния и символа есть один переход.
    * **Недетерминированная**: Возможны несколько переходов из одного состояния.
    * **Полнодетерминированная**: Определены переходы для всех символов алфавита из каждого состояния, исключены эпсилон-переходы.
11. Что такое ДКА и НКА?
    * **ДКА**: Автомат с единственным переходом для каждого состояния и символа.
    * **НКА**: Автомат с несколькими переходами для одного состояния и символа.
12. Как преобразовать НКА в ДКА?
    * Построение множества состояний ДКА как подмножеств состояний НКА. Создание новых состояний по всем входным символам для каждого состояния НКА.
13. Как преобразовать детерминированную грамматику в полнодетерминированную?
    * Дополнение переходов для всех символов, включая "ловушечное" состояние для неразрешенных переходов.
14. Как создать программу анализа по полнодетерминированной грамматике?
    * Использовать таблицу переходов, где строки — состояния, а столбцы — входные символы, для последовательного перехода по состояниям на основе входных данных.