|  |
| --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» |

|  |
| --- |
| Пермь 2021 |

**1. Задание**

Разработать компилятор для программ на языке Pascal. Язык программирования, на котором должен быть написан компилятор, должен быть объектно-ориенитрованным.

**Проектирование**

Компиляция состоит из следующих фаз: модуль ввода-вывода, анализатор, генератор. Анализатор, в свою очередь, состоит из 3 модулей: лексический, синтаксический и семантический анализаторы. Весь компилятор принимает следующую структуру:

Модуль ввода-вывода получает исходный код программы и преобразует в последовательность литер

Лексический анализатор осуществляет лексический анализ, строит идентификаторы, ключевые слова, разделители, числа

Синтаксический анализатор проверяет, удовлетворяет ли программа формальным правилам

Семантический анализатор проверяет смысловую правильность программы

Генератор генерирует объектную программу

**Модуль ввода-вывода**

Модуль ввода-вывода содержит класс IO

public IO(string path).

В конструктор класса IO передается путь к файлу с текстом программы на языке Pascal, в строковое поле ProgramText.

public string ProgramText { get; private set; }

считывается текст всей программы построчно. Из модуля ввода-вывода лексический анализатор берет очередной символ, для этого есть метод. В данном методе обновляется позиция символа по порядку, по номеру строки и позиция в самой строке

public char Nextch()

Для проверки лексем с длиной более 1 необходимо брать следующий символ. В случае, если он не подошел к лексеме, нужно вернуться на символ обратно, для этого есть метод

public void Back()

**Лексический анализатор**

Я разделил лексемы на идентификаторы, ключевые слова, целые и вещественные числа (беззнаковые), разделители, символы операции, спецификаторы, простые типы, строки и символы. Конструктор анализатора получает экземпляр класса ввода вывода

public Lexical(IO Input\_Reader)

Лексема характеризуется типом, значением, номером строки и положением в строке.

Разные типы лексем имеют разные типы значения. Для удобного хранения лексем было организовано наследование классов.

Базовый класс – Лексема

Производные: Вещественные, целые, идентификаторы, ключевые слова, разделители, операции, спецификаторы, простые типы, строки и символы

public class Lexem

{

public int LinePosition;

public int LineNumber;

public string type;

public Lexem (int pos, int num, string input\_type)

{

LineNumber = num;

LinePosition = pos;

type = input\_type;

}

};

Производные классы содержат поле value соответствующего типа (вещественные, целые, строковые и т.д.)

public class Keyword : Lexem

public class Simple\_Type : Lexem

public class Real : Lexem

public class Int : Lexem

public class Operation : Lexem

public class Id : Lexem

public class Limiter : Lexem

public class Specifier : Lexem

public class String : Lexem

public class Char : Lexem

Получение следующей лексемы происходит в методе

public Lexem NextSym(out string raw\_value)

Здесь создается экземпляр класса Лексема. Изначально создается базовый класс типа «Лексама», в процессе обработки литер создаются производные классы соответствующего типа, на вход подаются полученные слова и внутри конструктора преобразуются в соответствующие типы. Метод возвращает ссылку типа лексема на объект производного типа.

Определение идентификатора: первый символ из латинского алфавита, либо нижнее подчеркивание. Остальные могут быть теми же + цифры.

Определение ключевого слова: все символы должны быть латинскими, список ключевых слов находится внутри соответствующего массива

Определение целочисленной беззнаковой лексемы: все символы – цифры

Определение вещественной беззнаковой лексемы: получена константа в экспоненциальной форме и с фиксированной точкой

Определение операции, разделителя, спецификатора: если список операций содержит данный набор символов

Простые типы также содержатся в соответствующем массиве

Идентификаторы и ключевые слова начинаются с латиницы, константы с цифры.

Вывод всех лексем в цикле в методе

public void PrintLexem()

Также лексический анализатор содержит список всех системных ошибок в методе

public static string Get\_Error(int code)

Метод

public void Raise\_Error(int code)

Получает номер ошибки на вход, выводит значение ошибки, строку, положение ошибки в строке

В следующем методе происходит проверка невидимых символов:

public bool Is\_Void(char symbol)

Для отката определения последовательности лексем в синтаксическом анализаторе предусмотрена структура

public struct Position\_L

Для возврата состояния лексического анализатора и модуля ввода-вывода

Также реализованы методы для сохранения и отката состояния:

public Position\_L Save\_Position()

public void Set\_Position(Position\_L Backup)

**Синтаксический анализатор**

Данный анализатор работает одновременно с лексическим анализатором и модулем ввода-вывода для отката ошибок.

Описание синтаксиса языка Паскаль в виде диаграмм Бэкуса-Наура я взял в справочнике

Для вывода нестандартных ошибок реализован метод

public void Unhandled\_Error(string expected)

Который получает на вход строку ошибки

Метод для обновления текущей лексемы:

public void NextSym()

Метод для обработки строковых лексем:

public bool Accept\_Raw(string raw\_value)

Далее идет реализация обработки конструкций через диаграммы Бэкуса-Наура:

Общая структура программы

public void Accept\_Program()

Блок:

public void Accept\_Block()

Часть описаний:

public void Accept\_Description\_Chapter()

Часть описания операторов:

public bool Accept\_Operators\_Chapter()

Часть описания переменных:

public void Accept\_Variables\_Chapter()

Принять составной оператор:

public bool Accept\_Compound\_Operator()

Принять оператор:

public bool Accept\_Operator()

Принять основной оператор:

public bool Accept\_Main\_Operator()

Принять производный оператор:

public bool Accept\_Derivative\_Operator()

Принять описание переменных:

public bool Accept\_Variables\_Description()

Принять оператор присваивания:

public bool Accept\_Assignment\_Operator()

Принять пустой оператор:

public bool Accept\_Void\_Operator()

Принять оператор присоединения:

public bool Accept\_Connection\_Operator()

Принять имя переменной:

public bool Accept\_Variable\_Name()

Принять тип:

public bool Accept\_Type()

Принять имя функции:

public bool Accept\_Function\_Name()

Принять выражение:

public bool Accept\_Expression()

Принять переменную:

public bool Accept\_Variable()

Принять имя поля:

public bool Accept\_Field\_Name()

Принять арифметическое выражение:

public bool Accept\_Arithmetic\_Expression()

Принять слагаемое:

public bool Accept\_Summand()

Принять множитель:

public bool Accept\_Multiplier()

Принять функцию:

public bool Accept\_Function()

Принять фактический параметр:

public bool Accept\_Actual\_Parameter()

Принять литерное выражение:

public bool Accept\_Literal\_Expression()

Принять имя константы:

public bool Accept\_Const\_Name()

Принять логическое выражение:

public bool Accept\_Logical\_Expression()

Принять простое логическое выражение:

public bool Accept\_Simple\_Logical\_Expression()

Принять логическое слагаемое:

public bool Accept\_Logical\_Summand()

Принять логический множитель:

public bool Accept\_Logical\_Multiplier()

Принять отношение:

public bool Accept\_Relationship()

Принять скалярное отн ение:

public bool Accept\_Scalar\_Relationship()

Принять строковое отношение:

public bool Accept\_String\_Relastionship()

Принять множественное отношение:

public bool Accept\_Multiple\_Relastionship()

Принять множественное выражение:

public bool Accept\_Multiple\_Expression()

Получить класс лексемы:

public string Get\_Class()

Принять оператор сравнения:

public bool Accept\_Comparison\_Operaion()

Принять без знаковое вещественное:

public bool Accept\_Real\_Without\_Sign()

Принять без знаковое целочисленное:

public bool Accept\_Unsigned\_Integer()

Принять имя:

public bool Accept\_Name()

Принять простой тип:

public bool Accept\_Type\_Name()

Принять строковую константу:

public bool Accept\_String\_Const()

Принять true/false:

public bool Accept\_True\_False()

Принять литерную константу:

public bool Accept\_Literal\_Const()