МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора SDV-2024»

Выполнил студент Семёнов Даниил Вячеславович

(Ф.И.О.)

Руководитель проекта асс. Гончар Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Гончар Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Гончар Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 4](#_Toc185179515)

[Глава 1 Спецификация языка программирования 5](#_Toc185179516)

[1.1 Характеристика языка программирования 5](#_Toc185179517)

[1.2 Определение алфавита языка программирования 5](#_Toc185179518)

[1.3 Применяемые сепараторы 5](#_Toc185179519)

[1.4 Применяемые кодировки 6](#_Toc185179520)

[1.5 Типы данных 6](#_Toc185179521)

[1.6 Преобразование типов данных 7](#_Toc185179522)

[1.7 Идентификаторы 8](#_Toc185179523)

[1.8 Литералы 8](#_Toc185179524)

[1.9 Объявлениe данных 8](#_Toc185179525)

[1.10 Инициализация данных 9](#_Toc185179526)

[1.11 Инструкции языка 9](#_Toc185179527)

[1.12 Операции языка 10](#_Toc185179528)

[1.13 Выражения и их вычисления 10](#_Toc185179529)

[1.14 Конструкции языка 11](#_Toc185179530)

[1.15 Область видимости идентификаторов 11](#_Toc185179531)

[1.16 Семантические проверки 12](#_Toc185179532)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc185179533)

[1.18 Стандартная библиотека и её состав 12](#_Toc185179534)

[1.19 Ввод и вывод данных 13](#_Toc185179535)

[1.20 Точка входа 13](#_Toc185179536)

[1.21 Препроцессор 13](#_Toc185179537)

[1.22 Соглашения о вызовах 13](#_Toc185179538)

[1.23 Объектный код 14](#_Toc185179539)

[1.24 Классификация сообщений транслятора 14](#_Toc185179540)

[1.25 Контрольный пример 14](#_Toc185179541)

[Глава 2 Структура транслятора 15](#_Toc185179542)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc185179543)

[2.2 Перечень входных параметров транслятора 16](#_Toc185179544)

[2.3 Протоколы, формируемые транслятором 16](#_Toc185179545)

[Глава 3 Разработка лексического анализатора 18](#_Toc185179546)

[3.1 Структура лексического анализатора 18](#_Toc185179547)

[3.2 Контроль входных символов 18](#_Toc185179548)

[3.3 Удаление избыточных символов 19](#_Toc185179549)

[3.4 Перечень ключевых слов 19](#_Toc185179550)

[3.5 Основные структуры данных 21](#_Toc185179551)

[3.6 Структура и перечень сообщений лексического анализатора 22](#_Toc185179552)

[3.7 Принцип обработки ошибок 22](#_Toc185179553)

[3.8 Параметры лексического анализатора 22](#_Toc185179554)

[3.9 Алгоритм лексического анализа 22](#_Toc185179555)

[3.10 Контрольный пример 23](#_Toc185179556)

[Глава 4 Разработка синтаксического анализатора 24](#_Toc185179557)

[4.1 Структура синтаксического анализатора 24](#_Toc185179558)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 24](#_Toc185179559)

[4.3 Построение конечного магазинного автомата 25](#_Toc185179560)

[4.4 Основные структуры данных 26](#_Toc185179561)

[4.5 Описание алгоритма синтаксического разбора 26](#_Toc185179562)

[4.6 Параметры синтаксического анализатора 26](#_Toc185179563)

[4.7 Структура и перечень сообщений синтаксического анализатора 26](#_Toc185179564)

[4.8 Принцип обработки ошибок 27](#_Toc185179565)

[4.9 Контрольный пример 27](#_Toc185179566)

[Глава 5 Разработка семантического анализатора 28](#_Toc185179567)

[5.1 Структура семантического анализатора 28](#_Toc185179568)

[5.2 Функции семантического анализатора 28](#_Toc185179569)

[5.3 Структура и перечень сообщений семантического анализатора 28](#_Toc185179570)

[5.4 Принцип обработки ошибок 29](#_Toc185179571)

[5.5 Контрольный пример 29](#_Toc185179572)

[Глава 6 Вычисление выражений 30](#_Toc185179573)

[6.1 Выражения, допускаемые языком 30](#_Toc185179574)

[6.2 Польская запись и принцип ее построения 30](#_Toc185179575)

[6.3 Программная реализация обработки выражений 31](#_Toc185179576)

[6.4 Контрольный пример 31](#_Toc185179577)

[Глава 7 Генерация кода 32](#_Toc185179578)

[7.1 Структура генератора кода 32](#_Toc185179579)

[7.2 Представление типов данных в оперативной памяти 32](#_Toc185179580)

[7.3 Статическая библиотека 33](#_Toc185179581)

[7.4 Особенности алгоритма генерации кода 33](#_Toc185179582)

[7.5 Параметры, управляющие генерацией кода 34](#_Toc185179583)

[7.6 Контрольный пример 34](#_Toc185179584)

[Глава 8 Тестирование транслятора 35](#_Toc185179585)

[8.1 Общие положения 35](#_Toc185179586)

[8.2 Результаты тестирования 35](#_Toc185179587)

[Заключение 37](#_Toc185179588)

[Приложение А 39](#_Toc185179590)

[Приложение Б 41](#_Toc185179591)

[Приложение В 45](#_Toc185179592)

[Приложение Г 47](#_Toc185179593)

[Приложение Д 50](#_Toc185179594)

[Приложение Е 53](#_Toc185179595)

[Приложение Ж 54](#_Toc185179596)

[Приложение З 55](#_Toc185179597)

Введение

Целью курсового проекта является разработка компилятора для нового языка программирования с названием SDV-2024.

Компилятор представляет собой программу, которая преобразует исходный код программы, написанной на одном языке программирования, в машинный код или код на другом языке программирования.

Процесс компиляции состоит из двух ключевых этапов: анализа и генерации. Анализ включает разбиение исходного кода на части и создание промежуточного представления программы. Генерация заключается в построении целевой программы на основе этого представления. В рамках данного проекта исходный код будет транслироваться на язык ассемблера.

Компилятор SDV-2024 включает в себя четыре основных компонента:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор исходного кода на языке ассемблера.

С учетом основной цели курсового проекта, были определены следующие этапы и задачи:

* разработка спецификации языка программирования;
* разработка структуры транслятора;
* разработка лексического анализатора;
* разработка синтаксического анализатора;
* выполнение преобразования выражений;
* генерация кода на язык ассемблер;
* тестирование транслятора.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта.

Глава 1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования SDV-2024 является компилируемым процедурным, высокоуровневым. Имеет нестрогую статическую типизацию.

1.2 Определение алфавита языка программирования

Алфавит языка программирования — это набор допустимых символов, которые могут использоваться для написания программ на этом языке. Он включает буквы, цифры, специальные символы и операторы, определенные правилами языка.

Алфавит языка SDV-2024 использует символы латинского алфавита для записей инструкций, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также символы русского алфавита для записи строковых литералов.

1.3 Применяемые сепараторы

Символы сепараторы служат в качестве разделителей операций языка. Сепараторы, используемые в языке программирования SDV-2024, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| (…) | Круглые скобки | Приоритет операций, условие для одноимённого оператора, параметры функция |
| {…} | Фигурные скобки | Заключение программного блока |
| […] | Квадратные скобки | Блок для тела условных операторов |
| ; | Точка с запятой | Разделение программных конструкций |
| "…" | Двойные кавычки | Строковый литерал |
| '...' | Одинарные кавычки | Символьный литерал |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| + - \* / % | Знаки «плюс», «минус» «астерикс», «косая черта», «процент» | Арифметические выражения |
| < > =< >= == != | Знаки «меньше», «больше», «меньше либо равно», «больше либо равно», «равно», «не равно» | Операции сравнения |

Окончание таблицы 1.1

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| «пробел» | Пробел | Допустим везде, кроме идентификаторов и ключевых слов |

Сепараторы играют ключевую роль в структурировании кода

1.4 Применяемые кодировки

Для написания исходного кода на языке программирования SDV-2024 используется кодировка Windows-1251.

1.5 Типы данных

Допускается использование фундаментальных типов данных. В языке SDV- 2024 реализованы 4 типа данных: целочисленный(numb), строковый(line), символьный(symb), логический(bool). Описание типов данных, предусмотренных в данном языке представлено в таблице 1.2.

Таблица 1.2 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| numb | По умолчанию инициализируется 1.  Размер в байтах: 2.  Допустимый диапазоны значений: от −32768 до 32767.  Применяемые операции:  + (бинарный) – сложение;  - (бинарный) – вычитание;  \* (бинарный) – умножение;  / (бинарный) – целочисленное деление;  % (бинарный) – остаток от деления;  **=** (бинарный) – оператор присваивания;  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»;  **==** (бинарный) – оператор «равно»;  != (бинарный) – оператор «не равно»;  >= (бинарный) – оператор «больше или равно»;  =< (бинарный) – оператор «меньше или равно». |

Окончание таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Описание |
| symb | По умолчанию инициализируется символом конца строки “\0”.  Размер в байтах: 1.  Хранит символ в диапазоне: от 0 до 255.  Применяемые операции:  = (бинарный) – оператор присваивания. |
| bool | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. В памяти занимает 1 байт.  Значения: true (истина), false (ложь).  Инициализация по умолчанию: false.  Поддерживаемые операции:  + (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  / (бинарный) – оператор целочисленного деления;  % (бинарный) – оператор остатка от деления;  = (бинарный) – оператор присваивания.  Также поддерживаются все логические операции. |
| line | По умолчанию инициализируется пустой строкой: ‘’.  Размер в байтах (для одного символа): 1 байт.  Применяемые операции:  = (бинарный) – оператор присваивания. |

Тип данных в программировании определяет характеристики и возможности переменной или значения. Он указывает, какую информацию можно хранить в переменной, как оперировать этой информацией и какую память занимает переменная.

1.6 Преобразование типов данных

В языке программирования SDV-2024 существует неявное преобразования между логическими и целочисленными типами. Значение true преобразуется к 1, значение false – к 0.

Логический тип преобразуется к целочисленному в следующих ситуациях:

* при присваивании результата логического выражения переменной типа numb;
* при присваивании значения логического идентификатора или литерала переменной типа numb;
* при вычислении арифметических выражений.

1.7 Идентификаторы

Идентификаторы используются для именования функций, параметров и переменных. Максимальное количество идентификаторов не должно превышать допустимый размер таблицы идентификаторов, который составляет 4096 записей. Длина идентификатора не должна превышать 10 символов. Идентификаторы не должны совпадать с зарезервированными ключевыми словами языка программирования.

1.8 Литералы

Литерал — это элемент программы, который напрямую обозначает конкретное значение. В программировании литералы представляют собой неизменные значения, прописанные непосредственно в исходном коде. В языке SDV-2024 существует 4 типа литералов: целочисленный, логический, символьный и строковый. Краткое описание литералов языка SDV-2024 представлено в таблице 1.3.

Таблица 1.3 - Описание литералов

|  |  |  |
| --- | --- | --- |
| Тип литерала | Описание | Пример |
| Целочисленный литерал | Последовательность цифр 0…9 с предшествующим знаком минус или без него. | dec numb sum = 2; |
| Символьный литерал | Символ алфавита языка, заключенный в одинарные кавычки (' '). | dec symb a = 'B'; |
| Строковый литерал | Набор символов алфавита языка, заключенный в двойные кавычки (" "). | dec line str1 = "hello"; |
| Логический литерал | Два представления: “true” для истинного значения и “false” для ложного, используемые в логических операциях. | dec bool flag = true; |

Примеры правильных литералов: 52, -1, 0, “Hello, World!”, ‘k’, ‘f’, true, false.

Примеры неправильных литералов: -1233456754, 1002938584, 9876543333 TRUE, “”, ‘ff’.

1.9 Объявлениe данных

Переменная объявляется с использованием ключевого слова **dec**, после которого следует указать тип данных и имя идентификатора. При желании данные можно сразу инициализировать во время объявления.

Для определения функции применяется ключевое слово **func**, перед которым необходимо указать ключевое слово **dec** и тип данных, и после которого указывается имя.

1.10 Инициализация данных

Инициализация данных предполагает присвоение переменной значения прямо в момент её объявления.

Значение, которое присваивается переменной, должно находиться справа от знака равенства и может быть литералом, идентификатором, выражением или результатом вызова функции. При этом важно, чтобы тип присваиваемого значения совпадал с типом переменной.

В языке SDV-2024 предусмотрено значение по умолчанию для всех типов данных, если переменные не были инициализированы.

1.11 Инструкции языка

Все возможные инструкции языка программирования SDV-2024 представлены в общем виде в таблице 1.4.

Таблица 1.4 – Инструкции языка программирования SDV-2024

|  |  |
| --- | --- |
| Инструкция | Запись на языке SDV-2024 |
| Объявление переменной | dec <тип данных> <идентификатор>; |
| Объявление переменной с инициализацией | deс <тип данных> <идентификатор> = <значение>|<выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> |
| Объявление функции | dec <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | recovi <идентификатор> / <литерал>; |
| Условный оператор | provi(<условие>)  [<блок кода>]  otherwise  [<блок кода>]; |
| Вывод данных | write <идентификатор> / <литерал>; |

Окончание таблицы 1.4.

|  |  |
| --- | --- |
| Инструкция | Запись на языке SDV-2024 |
| Вывод данных с переносом на новую строку | writeln <идентификатор> / <литерал>; |
| Вызов функции | <идентификатор> ( <идентификатор>|  <литерал>, …); |

Все инструкции требуют закрывающую «;».

1.12 Операции языка

Арифметические операции, которые можно использовать в языке программирования SDV-2024, представлены в таблице 1.5.

Таблица 1.5 — Операции языка и их приоритеты

|  |  |  |
| --- | --- | --- |
| Тип операций | Операции | Приоритет |
| Логические операции | > – больше  < – меньше  == – равно  != – не равно  >= – больше или равно  =< – меньше или равно | -1  -1  -1  -1  -1  -1 |
| Арифметические операции | + – сложение  - – вычитание  \* – умножение  / – деление  % –остаток от деления | 2  2  3  3  3 |
| Операция запятая | , | 1 |

В языке программирования SDV-2024 операции с одинаковым приоритетом выполняются в порядке их добавления (слева направо). Это значит, что, если в выражении встречаются несколько операций с одинаковым приоритетом, они будут выполняться последовательно, начиная с той, которая стоит первой. Для смены приоритета арифметических операций можно использовать оператор круглые скобки ().

1.13 Выражения и их вычисления

В языке SDV-2024 поддерживаются выражения со всеми типами данных.

Синтаксис и правила работы с выражениями в SDV-2024 включают следующее:

* Каждое выражение и инструкция должны завершаться разделителем точка с запятой: “;”.
* Операнды в одном выражении должны быть одного типа, смешивание типов не допускается (за исключением логического и целочисленного типов).
* Присвоение значения переменной возможно, если типы данных выражения и переменной совпадают (за исключением возможности преобразования логического типа в целочисленный).
* В выражениях допустимо использовать вызовы функций в качестве операндов.
* Последовательное использование двух операторов подряд запрещено.

1.14 Конструкции языка

В языке SDV-2024 предусмотрены разнообразные программные конструкции, что позволяет эффективно решать широкий круг задач, варьирующихся по уровню сложности. Они обеспечивают основной функционал для выполнения вычислений, обработки данных и управления потоком выполнения. Ключевые программные конструкции языка программирования SDV-2024 представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка SDV-2024

|  |  |
| --- | --- |
| Конструкция | Запись на языке SDV-2024 |
| Главная функция | main  {  …  recovi <идентификатор> / <литерал>;  }; |
| Функция | dec <тип данных> func <идентификатор> (<тип> <идентификатор>, …)  {  …  recovi <идентификатор> / <литерал>;  }; |
| Условный оператор | provi(1<4)[…]; |

В языке SDV-2024 представлены основные конструкции, обеспечивающие выполнение различных операций, что позволяет решать задачи любого уровня сложности.

1.15 Область видимости идентификаторов

В языке SDV-2024 все переменные и параметры являются локальными и могут быть объявлены исключительно внутри тела функции. Идентификаторы, определенные в одной функции, недоступны для использования в других функциях.

1.16 Семантические проверки

В языке программирования SDV-2024 предусмотрены различные семантические проверки. Эти проверки необходимы для обеспечения корректности выполнения программы и предотвращения ошибок на стадии компиляции. Перечень некоторых семантических проверок, предусмотренных языком, приведен в таблице 1.7.

Таблица 1.7 - Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Проверка |
| 1 | Наличие главной функции main – точки входа в программу |
| 2 | Количество точек входа в программу (не должно превышать 1) |
| 3 | Переопределение идентификаторов |
| 4 | Использование необъявленных идентификаторов |
| 5 | Правильность передаваемых в функцию параметров: количество, типы |
| 6 | Совпадение типа функции и типа возвращаемого ею значения |
| 7 | Деление на ноль в арифметических операциях |
| 8 | Проверка на вызов функции в логических выражениях |
| 9 | Соответствие типа идентификатора и типа присваиваемого ему значения |
| 10 | Превышение размера строковых и целочисленных литералов; |

Таким образом, при нарушении любого из установленных правил семантический анализатор выдаст сообщение об ошибке.

1.17 Распределение оперативной памяти на этапе выполнения

Во время трансляции кода используется две области памяти: сегмент констант и сегмент данных. Эти области заполняются на основе таблицы лексем и таблицы идентификаторов, которые формируются в ходе лексического анализа.

1.18 Стандартная библиотека и её состав

В языке SDV-2024 присутствует стандартная библиотека, которая подключается автоматически. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 - Стандартная библиотека языка SDV-2024

|  |  |
| --- | --- |
| Функция | Описание |
| lenline(line a) | Целочисленная функция, возвращающая длину строки а. |
| random(min, max) | Функция, возвращающая случайно сгенерированное значение в диапазоне (min, max) |
| concat(line a, line b) | Строковая функция, возвращающая конкатенацию строки a и b. |
| WriteLine(line a) | Функция выводит в консоль строку a. |
| WriteLineLn(line a) | Функция выводит в консоль строку a с переносом на новую строку. |
| WriteSymb(symb a) | Функция выводит в консоль символ a. |
| WriteSymbLn(symb a) | Функция выводит в консоль символ a с переносом на новую строку. |
| WriteNumb(numb a) | Функция выводит в консоль число a. |
| WriteNumbLn(numb a) | Функция выводит в консоль число a с переносом на новую строку. |
| WriteBool(bool a) | Функция выводит в консоль булево значение a. |
| WriteBoolLn(bool a) | Функция выводит в консоль булево значение a с переносом на новую строку. |

1.19 Ввод и вывод данных

Вывод данных в языке SDV-2024 осуществляется с помощью ключевых слов write и writeln. В зависимости от типа параметра определяется функция для вывода из стандартной библиотеки, описанной в таблице 1.8.

Функции ввода не предусмотрены.

1.20 Точка входа

Точкой входа является функция main. Точка входа в приложение не может отсутствовать, и должна быть единственной.

1.21 Препроцессор

В языке SDV-2024 не предусмотрена реализация препроцессора.

1.22 Соглашения о вызовах

В языке используется соглашение о вызовах под названием stdcall – это значит, что в программе все параметры функции будут передаваться через стек справа налево и память будет освобождаться вызываем кодом.

1.23 Объектный код

Исходный код языка SDV-2024 транслируется в язык Ассемблер.

1.24 Классификация сообщений транслятора

В соответствии с префиксами будут различаться сообщения, представленные в таблице 1.9.

Таблица 1.9 - Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-110 | Системные ошибки, ошибки параметров |
| 200-299 | Ошибки лексического анализа |
| 300-399 | Ошибки семантического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 400 – 499, 700 – 999 | Зарезервированные коды ошибок |

Транслятор генерирует сообщения об ошибках пользователю и выводит их в файл протокола.

1.25 Контрольный пример

Контрольный пример на языке SDV-2024 в приложении А.

Глава 2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Трансляторы играют важную роль в разработке программного обеспечения, обеспечивая перевод программ с одного языка программирования на другой. В контексте языка SDV-2024 транслятор выполняет преобразование программ, написанных на SDV-2024, в код на языке ассемблера. Основные компоненты транслятора включают лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода, который отвечает за создание выходного кода на языке ассемблера. Каждый из этих компонентов выполняет свою уникальную функцию, и их взаимодействие основано на четко определённых принципах и правилах. Такое взаимодействие компонентов обеспечивает правильность и точность процесса трансляции, что критично для создания эффективных и работоспособных программ. Принцип их взаимодействия иллюстрируется на рисунке 2.1.

Принцип их взаимодействия представлен на рисунке 2.1.

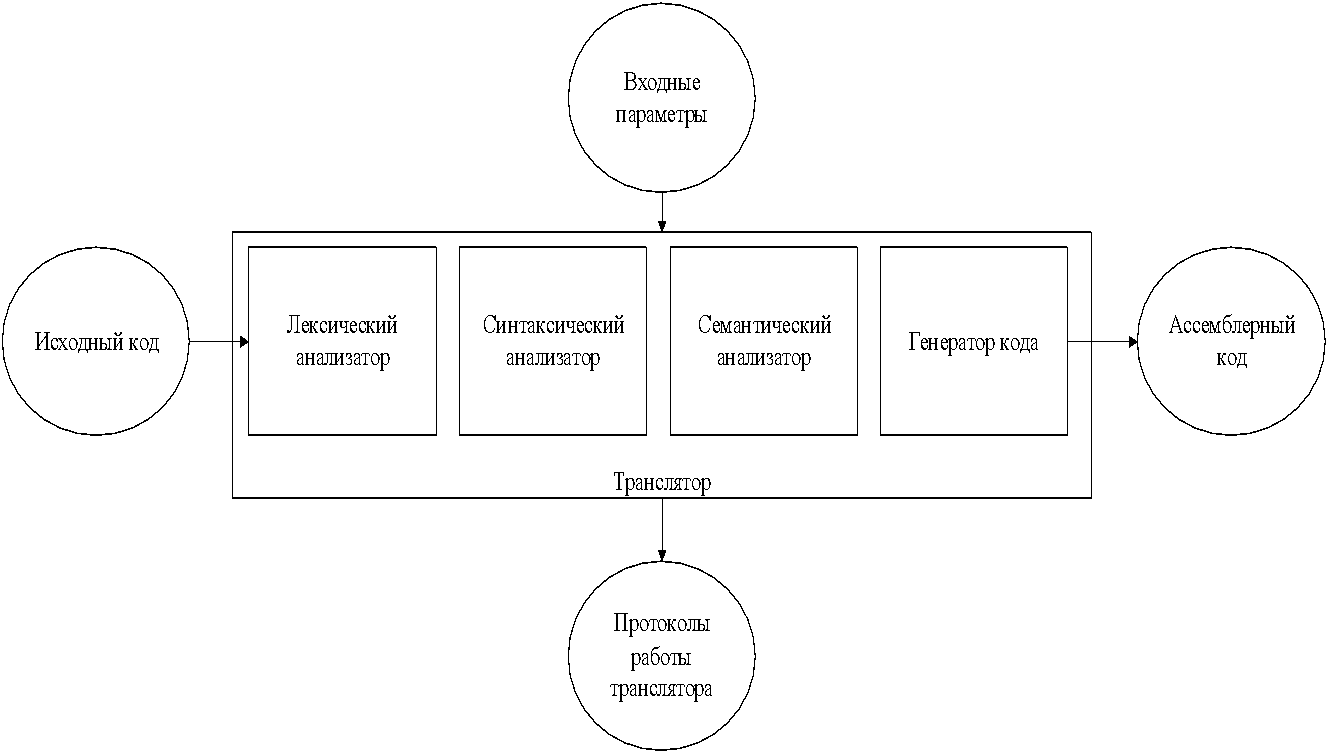


Рисунок 2.1 – Структура транслятора

Лексический анализ — это первая фаза трансляции, которая играет ключевую роль в процессе обработки программы. Лексический анализатор отвечает за выявление ошибок в лексической структуре языка и формирование таблиц лексем и идентификаторов. Подробности этого этапа будут рассмотрены в третьем разделе.

Синтаксический анализ представляет собой важную часть транслятора и выполняет задачи распознавания синтаксических конструкций, а также создания промежуточного представления программы. Входными данными для синтаксического анализа служат таблицы лексем, полученные от лексического анализатора. Этот этап будет подробно описан в четвертом разделе.

Семантический анализ отвечает за проверку исходной программы на соответствие семантике языка, что включает в себя согласованность с его правилами и определениями. Этот этап обеспечивает более глубокий анализ программы, сосредотачиваясь на её смысловом содержании. Подробная информация будет представлена в пятом разделе.

Генерация кода — это заключительный этап транслятора, который преобразует данные, полученные на предыдущих этапах, в код на ассемблере. Генератор кода принимает на вход таблицы идентификаторов и лексем и, используя эту информацию, создает ассемблерные инструкции, соответствующие исходной программе на языке SDV-2024. Этот процесс будет описан более подробно в седьмом разделе.

2.2 Перечень входных параметров транслятора

В таблице 2.1 представлены входные параметры, которые обязаны либо могут использоваться для управления работой транслятора.

Таблица 2.1 Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Тип | Характеристика |
| -in: | Обязательный | Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением \*.txt |
| -out: | Не обязательный | Указывает на файл для вывода необходимых данных. Если не указан явно, то имя файла формируется путем конкатенации имени файла исходного кода и постфикса «.asm» |
| -log: | Не обязательный | Указывает на файл для вывода протокола работы программы. Если не указан явно, то имя файла формируется путем конкатенации имени файла исходного кода и постфикса «.log» |
| -poliz | Не обязательный | Ключ для вывода на консоль промежуточного представления кода после преобразования в польскую инверсную запись |
| -lt | Не обязательный | Ключ для вывода таблицы лексем на консоль |
| -id | Не обязательный | Ключ для вывода таблицы идентификаторов на консоль |

Данные входные параметры транслятора используются для форматирования файлов с результатами работы лексического, синтаксического и семантического анализаторов.

2.3 Протоколы, формируемые транслятором

Таблица с перечнем протоколов, формируемых транслятором языка SDV-2024 и их назначением представлена в таблице 2.2.

Таблица 2.2 Протоколы, формируемые транслятором языка SDV-2024

|  |  |
| --- | --- |
| Протокол | Характеристика |
| Файл для вывода протокола работы программы (-log:) | Содержит информацию о времени выполнения приложения, входных параметрах, таблицу лексем, таблицу идентификаторов промежуточное представление кода; трассировку синтаксического анализа; дерево разбора, время выполнения разбора; промежуточное представление кода после приведения его к польской нотации. |
| Выходной файл с расширением \*.asm | Содержит код на языке ассемблера, сгенерированный на основе исходного кода. |

Результатом работы транслятора языка SDV-2024 является исходный код на языке ассемблера и протокол работы транслятора, содержащий основную информацию о процессе обработки исходного кода.

Глава 3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Структура лексического анализатора представлена на рисунке 3.1

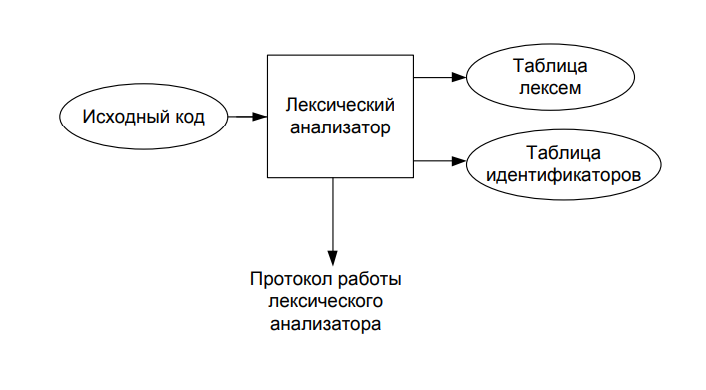


Рисунок 3.1 — Структура лексического анализатора

Входными данными для лексического анализатора является исходный код на языке SDV-2024. Таблица лексем и таблица идентификаторов, сформированные в результате лексического анализа, являются результатом работы лексического анализатора.

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

|  |
| --- |
|  |

Рисунок 3.2 — Таблица контроля входных символов

Таблица используется для определения допустимости символов в соответствии с их значениями в таблице ASCII. Каждый элемент таблицы имеет значение: F обозначает запрещённые символы, а T — разрешённые.

При чтении файла символы проверяются на соответствие этой таблице. Если символ помечен как T, программа считывает его и добавляет в буфер. Если символ помечен как F, работа транслятора прекращается, и выводится сообщение об ошибке.

## **3.3 Удаление избыточных символов**

Избыточными символами являются лишние символы пробела.

Эти символы удаляются перед этапом разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. Посимвольное считывание исходного кода, занесенного в структуру In.
2. Пробелы, которые стоят рядом с другими сепараторами или пробелами, становятся избыточными.
3. Избыточные символы подлежат удалению.

## **Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие ключевых слов и лексем приведено в таблице 3.1.

Таблица 3.1 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | deс | d |
| numb, line, bool, symb | t |
| main | m |
| func | f |
| recovi | r |
| write | p |
| writeln | w |
| provi | c |
| otherwise | ! |
| Иное | Идентификатор | i |
| Литерал | l |
| true, false | y |

Окончание таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Цепочка | | Лексема |
| Функции стандартной библиотеки | concat | | + |
| lenline | | / |
| random | | z |
| Сепараторы | | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| [ | [ |
| ] | ] |
| Операторы | | Арифметические (+, -, \*, /, %) | v |
| Логические (== != > < >= =<) | s |
| Присваивание (=) | = |

Каждому выражению соответствует детерминированный конечный Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. При успешном разборе выражение записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация дополнительно заносится в таблицу идентификаторов. Пример конечного автомата представлен на рисунке 3.3

|  |
| --- |
|  |

Рисунок 3.3 — Пример конечного автомата с графом для ключевого слов main

В приложении Б представлены конечные автоматы для других слов языка.

## **3.5** **Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Код на языке C++ со структурой таблицы лексем представлен на рисунке 3.4. Код на языке C++ со структурой таблицы идентификаторов представлен на рисунке 3.5.

|  |
| --- |
|  |

Рисунок — 3.4 Структура таблицы лексем

Эти таблицы служат для хранения информации о лексемах.

|  |
| --- |
|  |

Рисунок 3.5 — Структура таблицы идентификаторов

Эти таблицы служат для хранения информации о идентификаторах, обеспечивая быстрый доступ к данным для последующих этапов обработки программы.

* 1. **Структура и перечень сообщений лексического анализатора**

Перечень сообщений лексического анализатора представлен на рисунке 3.6.

|  |
| --- |
|  |

Рисунок 3.6 — Сообщения лексического анализатора

Таким образом, при написании кода, достаточно просто понять, с чем конкретно связана та или иная ошибка.

* 1. **Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, а также информацию об ошибке. При возникновении сообщения, лексический анализатор останавливает работу и далее сообщение выводится в файл протокола.

* 1. **Параметры лексического анализатора**

Входными параметрами для лексического анализатора является исходный текст программы, написанный на языке SDV-2024, а также файл протокола в который записываются выходные данные (таблица лексем и таблица идентификаторов).

* 1. **Алгоритм лексического анализа**

Лексический анализ основывается на работе конечных автоматов, разбирающих регулярные выражения. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Алгоритм лексического анализа:

− из входного потока символов программы на исходном языке удаляются лишние пробелы;

− формируется вектор из слов языка;

− для каждого слова выполняется функция распознавания лексемы;

− при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;

− при неуспешном распознавании выдается сообщение об ошибке;

− формируется протокол работы.

* 1. **Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении В.

Глава 4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Структура синтаксического анализатора представлена на рисунке 4.1.

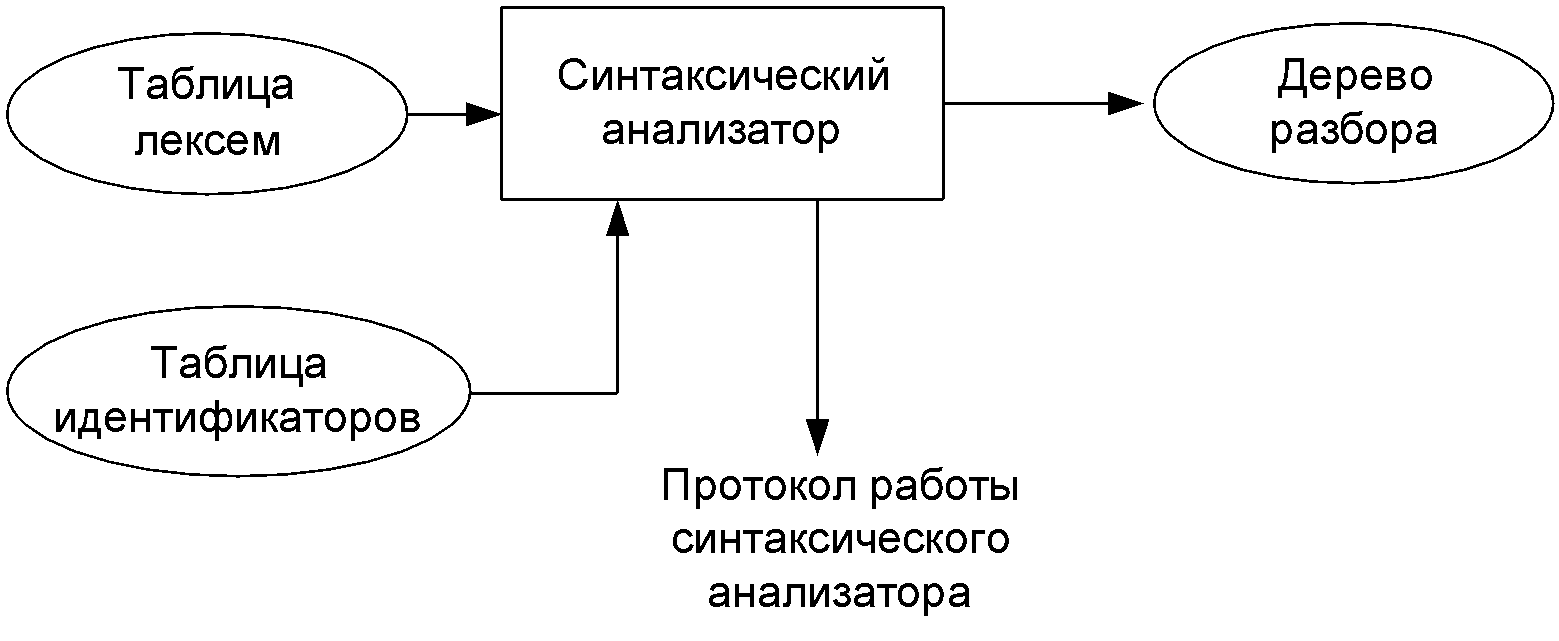


Рисунок 4.1 – Структура синтаксического анализатора

Исходными данными синтаксического анализатора являются таблицы лексем и идентификаторов, полученными на первой фазе трансляции. Результат работы синтаксического анализатора – дерево разбора (промежуточное представление кода).

4.2 Контекстно свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка SDV-2024 используется контекстно-свободная грамматика , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или );
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

T – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

N – нетерминальные символы, представленные заглавными буквами латинского алфавита.

Таблица 4.1, описывающая правила грамматики языка SDV-2024 представлена в приложении Г.

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $). |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

На основании данной таблицы можно заключить, что магазинный автомат применяется для анализа и обработки языка SDV-2024 с использованием контекстно-свободной грамматики Автомат включает состояния, алфавит символов, функцию переходов, а также начальное и конечные состояния. Используя эти компоненты, автомат выполняет разбор и трансляцию программного кода, написанного на языке SDV-2024.

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка SDV-2024. Данные структуры представлены в приложении Д.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ.
2. На основе полученных ранее таблиц формируется входная лента.
3. Запускается автомат.
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку.
6. Если в магазине встретился нетерминал, переходим к пункту 4.
7. Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. После 3 исключений синтаксический анализатор завершает работу и генерирует последнее исключение.

4.6 Параметры синтаксического анализатора

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем и дерево разбора, которые записываются в файл протокола данного этапа обработки.

4.7 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен на рисунке 4.2.

|  |
| --- |
|  |

Рисунок 4.2 — Сообщения синтаксического анализатора

Таким образом, при написании кода, достаточно просто понять, с чем конкретно связана та или иная ошибка.

4.8 Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка, которая записывается в протокол работы и работа транслятора останавливается.

4.9 Контрольный пример

Результатом работы синтаксического анализатора является дерево разбора. Результат работы синтаксического анализатора представлен в приложении Е.

Глава 5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов, а также дерево разбора, и последовательно ищет необходимые ошибки. Общая структура семантического анализатора представлена на рисунке 5.1.

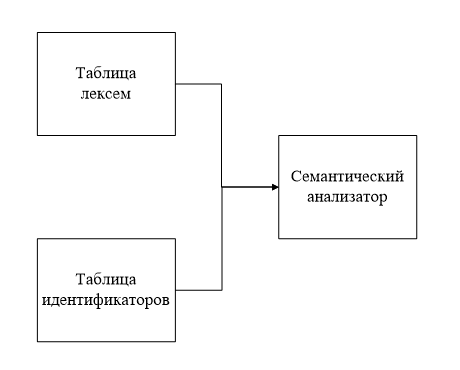


Рисунок 5.1 — структура семантического анализатора

Некоторые проверки (такие как проверка на единственность точки входа, проверка на повторную инициализацию переменной) осуществляются в процессе лексического анализа.

5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основные правила языка (семантики языка).

За семантический анализ отвечает функция AnalyzeSem. Ее входными параметрами является таблица лексем и таблица идентификаторов.

5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

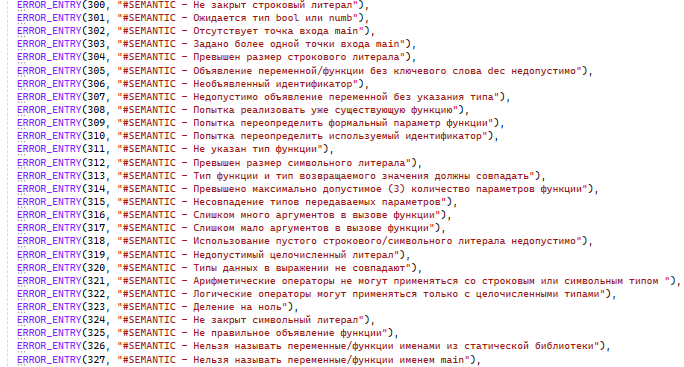


Рисунок 5.2 — Перечень сообщений семантического анализатора

Таким образом, при написании кода, достаточно просто понять, с чем конкретно связана та или иная ошибка.

5.4 Принцип обработки ошибок

Все семантические ошибки являются критическими, из-за чего транслятор прекращает свою работу и в протокол работы транслятора выводится соответствующее сообщение об ошибке.

5.5 Контрольный пример

Результат работы семантических функций описан в главе 8.

Глава 6 Вычисление выражений

6.1 Выражения, допускаемые языком

В языке SDV-2024 допустимы выражения с использованием целочисленных литералов и идентификаторов. Однако использование отрицательных целочисленных литералов в условных выражениях не допустимо. При необходимости сравнения отрицательных чисел, следует использовать целочисленные идентификаторы. В выражениях языка также допустимо использование вызова функции в качестве операнда, за исключением логических выражений. К допустимым операторам выражений относятся арифметические и логические операторы, описанные в пункте 1.12

6.2 Польская запись и принцип ее построения

В языке SDV-2024 выражения преобразуются в обратную польскую запись.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит открывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все извлекаются в строку;

– открывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая круглая скобка выталкивает все до открывающей и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

В таблице 6.2 представлен пример преобразования выражения в обратную польскую запись.

Таблица 6.1 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| (a + b)\*5 |  |  |
| a + b)\*5 | ( |  |
| + b)\*5 | ( | a |
| b)\*5 | ( + | b |
| )\*5 | ( + | a b |
| \*5 |  | a b + |
| 5 | \* | a b + |
|  | \* | a b + 5 |
|  |  | a b + 5 \* |

Таким образом, в результате преобразования все выражения языка SDV-2024 представляются в виде обратной польской записи, что позволяет выполнять их вычисление без использования скобок и с легкостью определять порядок операций.

6.3 Программная реализация обработки выражений

Программная реализация алгоритма преобразования выражений в обратную польскую запись основана на функциях Poliz и StartPoliz. Функция StartPoliz принимает как параметр таблицу лексем и таблицу идентификаторов и содержит цикл, в ходе которого перебираются все лексемы исходного кода. Если последовательность лексем соответствует началу выражения, вызывается функция Poliz, где и проводится преобразование выражений к польской нотации.

6.4 Контрольный пример

В приложении Ж приведено представление промежуточного кода, отображающее результаты преобразования выражений в польский формат.

Глава 7 Генерация кода

7.1 Структура генератора кода

На завершающем этапе трансляции языка SDV-2024 выполняется генерация кода. Этот процесс представляет собой преобразование внутреннего представления исходной программы в последовательность символов выходного языка с помощью компилятора. Для генерации используются таблицы лексем и идентификаторов, на основе которых создается файл с ассемблерным кодом.

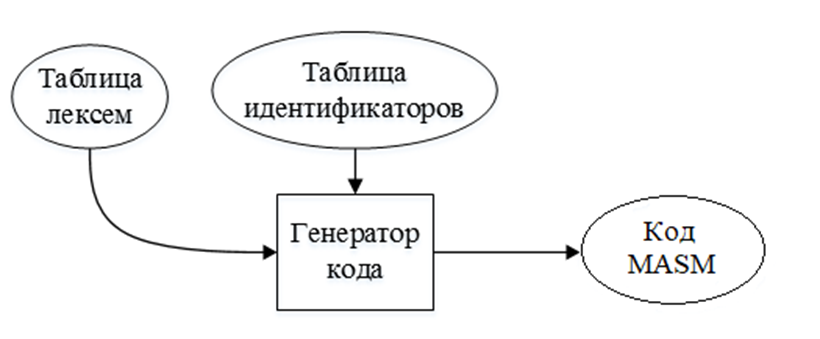


Рисунок 7.1 – Структура генератора кода

Таким образом, генератор кода выполняет не менее значимую часть компиляции.

7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .data и.const языка ассемблера. Соответствия между типами данных идентификаторов на языке SDV-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка SDV-2024 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SDV-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| Numb | sword | Хранит целочисленный тип данных. |
| Bool | sword | Хранит булевый тип данных (в виде целого числа) |
| Line, symb | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

Следовательно, таблица 7.1 показывает соответствия между типами идентификаторов на языке SDV-2024 и языке ассемблера. Это важно при переводе кода с языка SDV-2024 на язык ассемблера, чтобы правильно определить типы данных и использовать соответствующие инструкции и регистры для работы с идентификаторами.

7.3 Статическая библиотека

В языке SDV-2024 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Подключение библиотеки происходит с помощью includelib на этапе генерации кода. C помощью оператора EXTRN объявляются функции из библиотеки. Пример подключения библиотеки в исходном коде на языке ассемблера представлен на рисунке 7.2.

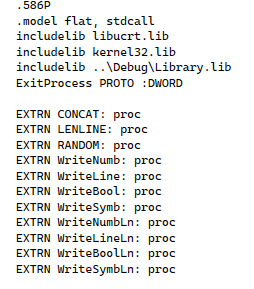


Рисунок 7.2 – Подключение статической библиотеки

Состав статической библиотеки указан в пункте 1.18.

7.4 Особенности алгоритма генерации кода

В языке SDV-2024 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.3.

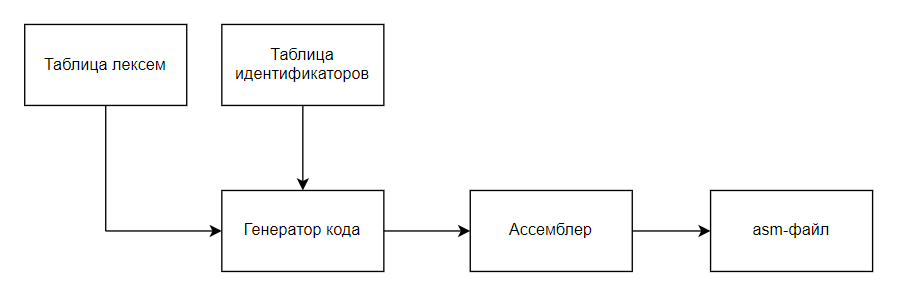


Рисунок 7.3 - Структура генератора кода

В таблице 7.2 представлены прототипы функций, осуществляющих генерацию кода, и их описание.

Таблица 7.2 – Прототипы функций, осуществляющих генерацию кода

|  |  |  |
| --- | --- | --- |
| void Generation(LT::LexTable, IT::IdTable, wchar\_t) | | Основная функция. Формирует поток выходного файла и вызывает другие генерирующие функции. |
| void Head(ofstream\*); | | Функция, генерирующая заголовок ассемблерного файла (подключение библиотек, указание прототипов функций и т.д.). |
| void ConstSegment(IT::IdTable, ofstream\*); | | Функция, генерирующая сегмент констант. |
| void DataSegment (LT::LexTable, IT::IdTable, ofstream\*) | | Функция, генерирующая сегмент данных. |
| void CodeSegment (LT::LexTable, IT::IdTable, ofstream\*) | Функция, генерирующая сегмент кода. | |

7.5 Параметры, управляющие генерацией кода

На вход генератора кода поступают таблицы лексем и идентификаторов исходного кода программы на языке SDV-2024. Результаты работы генератора кода выводятся в файл с расширением \*.asm.

7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении З.

Глава 8 Тестирование транслятора

8.1 Общие положения

Тестирование должно покрывать как можно больше сценариев использования языка и его конструкций. Все тесты были представлены для типичных ошибок пользователей при использовании языка. Когда компилятор обнаруживает ошибку, он записывает информацию о ней в протокол, содержащий номер ошибки и диагностическое сообщение, помогающее разработчику понять причину ошибки компиляции. Результаты тестирования записываются в файл .log.

8.2 Результаты тестирования

В таблице 8.1 приведены результаты тестов для разных этапов трансляции.

Таблица 8.1 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Проверка на допустимость символов | |
| ma.in  {  dec line “abcd”  }; | Ошибка200:#LEXICAL - Недопустимый символ в исходном файле (-in)  Строка 1, позиция 3 |
| Лексический анализ | |
| main  {  dec line addresOfDepartment "adress";  }; | Ошибка208:#LEXICAL - Длина идентификатора не должна превышать 10 символов  Строка -1, позиция 1 |
| main  {  dec numb@ count = 5;  }; | Ошибка207:#LEXICAL - Лексема не распознана  Строка 3, позиция 3 |
| Синтаксический анализ | |
| dec numb identif;  main  {recovi 0;}; | Ошибка 600:#SYNTAX - Неверная структура программы строка 1 |
| dec numb func sayHi  {  recovi 0;  };  main  {…}; | Ошибка 601:#SYNTAX - Отсутствует список параметров функции при её объявлении строка 2 |

Продолжение таблицы 8.1

|  |  |  |
| --- | --- | --- |
| dec numb func sayHi()  dec numb func test()  {  recovi 3;  };  main  { recovi 0; }; | Ошибка 603:#SYNTAX - Возможно отсутствует тело функции строка 2 | |
| Семантический анализ | | |
| main  {  dec numb size= 8;  recovi 0;  };  main  {  recovi 0;  } | | Ошибка303:#SEMANTIC - Задано более одной точки входа main  Строка -1, позиция 1 |
| main  {  size= 8;  }; | | Ошибка306:#SEMANTIC - Необъявленный идентификатор  Строка 3, позиция 2 |
| main  {  dec numb size = 8;  dec numb size = 9;  }; | | Ошибка310:#SEMANTIC - Попытка переопределить используемый идентификатор  Строка 4, позиция 10 |
| dec numb func myFunc(symb x)  {  dec line x = "qwerty";  recovi 0;  };  main  {  recovi = 0;  }; | | Ошибка309:#SEMANTIC - Попытка переопределить формальный параметр функции  Строка 3, позиция -1 |

Таким образом данный раздел предоставляет набор тестов для проверки лексического, синтаксического и семантического анализаторов.

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования SDV-2024. Таким образом, были выполнены основные задачи данной курсовой работы:

– разработана спецификация языка программирования;

– разработана структура транслятора;

– разработан лексический анализатор;

– разработан синтаксический анализатор;

– разработан семантический анализатор;

– разработан генератор кода на язык ассемблера;

– проведено тестирование транслятора.

Итоговая версия языка SDV-2024 включает:

* 4 типа данных;
* поддержку операторов вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 5 арифметических операторов для вычисления выражений;
* Наличие 6 логических операторов для вычисления выражений;
* Поддержка функций и условного оператора;

Таким образом, в ходе выполнения курсового проекта были получены новые знания и навыки в проектировании систем программирования и в разработке программного обеспечения для систем программирования.

Список использованных источников

1. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт.  - 3-е изд. – Москва: Вильямс, 2003. – 429 с.
2. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.
3. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.
4. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768 с.
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

# **Приложение А**

dec numb func sayHi(line name)

{

dec line message = concat("Hello ", name);

writeln message;

dec numb length = lenline(message);

recovi length;

};

dec bool func checkAge(numb age)

{

dec bool isAdult;

provi(age >= 18)

[

write "The person is an adult";

writeln '!';

isAdult = true;

]

otherwise

[

write "The person is not an adult";

writeln '$';

isAdult = false;

];

recovi isAdult;

};

dec numb func checkAv(line name, bool flag)

{

provi(flag)

[

write name;

writeln ": вы имеете возможность получить водительские права";

]

otherwise

[

write name;

writeln ": вы не можете получить водительские права";

];

recovi 0;

};

main

{

dec line name1 = "Daniil";

dec numb length1 = sayHi(name1);

dec numb age1 = 18;

dec bool isAdult1 = checkAge(age1);

checkAv(name1, isAdult1);

dec line name2 = "AnotherPerson";

dec numb length2 = sayHi(name2);

dec numb age2 = 15;

dec bool isAdult2 = checkAge(age2);

checkAv(name2, isAdult2);

writeln " ";

writeln "\*\*\*\*\*Анализ шанса на успех\*\*\*\*\*";

write "Шанс сдать на права составляет: ";

dec numb chance = random(length1, length2) + 6 \* (12 / 4) - 15 - (-1 \* -3) + random(12, 87);

dec symb percent = '%';

write chance;

write percent;

writeln " ";

provi(chance == 100)

[

writeln "Да вы счастливчик!";

];

};

Листинг 1 – Исходный код на языке SDV-2024

**Приложение Б**

FST::FST typeInteger("", 5,

FST::NODE(1, FST::RELATION('n', 1)),

FST::NODE(1, FST::RELATION('u', 2)),

FST::NODE(1, FST::RELATION('m', 3)),

FST::NODE(1, FST::RELATION('b', 4)),

FST::NODE()

);

FST::FST typeString("", 5,

FST::NODE(1, FST::RELATION('l', 1)),

FST::NODE(1, FST::RELATION('i', 2)),

FST::NODE(1, FST::RELATION('n', 3)),

FST::NODE(1, FST::RELATION('e', 4)),

FST::NODE()

);

FST::FST typeBool("", 5,

FST::NODE(1, FST::RELATION('b', 1)),

FST::NODE(1, FST::RELATION('o', 2)),

FST::NODE(1, FST::RELATION('o', 3)),

FST::NODE(1, FST::RELATION('l', 4)),

FST::NODE()

);

FST::FST typeChar("", 5,

FST::NODE(1, FST::RELATION('s', 1)),

FST::NODE(1, FST::RELATION('y', 2)),

FST::NODE(1, FST::RELATION('m', 3)),

FST::NODE(1, FST::RELATION('b', 4)),

FST::NODE()

);

FST::FST typeFunction("", 5,

FST::NODE(1, FST::RELATION('f', 1)),

FST::NODE(1, FST::RELATION('u', 2)),

FST::NODE(1, FST::RELATION('n', 3)),

FST::NODE(1, FST::RELATION('c', 4)),

FST::NODE()

);

FST::FST typeDeclare("", 4,

FST::NODE(1, FST::RELATION('d', 1)),

FST::NODE(1, FST::RELATION('e', 2)),

FST::NODE(1, FST::RELATION('c', 3)),

FST::NODE()

);

FST::FST typeReturn("", 7,

FST::NODE(1, FST::RELATION('r', 1)),

FST::NODE(1, FST::RELATION('e', 2)),

FST::NODE(1, FST::RELATION('c', 3)),

FST::NODE(1, FST::RELATION('o', 4)),

FST::NODE(1, FST::RELATION('v', 5)),

FST::NODE(1, FST::RELATION('i', 6)),

FST::NODE()

);

FST::FST typeMain("", 5,

FST::NODE(1, FST::RELATION('m', 1)),

FST::NODE(1, FST::RELATION('a', 2)),

FST::NODE(1, FST::RELATION('i', 3)),

FST::NODE(1, FST::RELATION('n', 4)),

FST::NODE()

);

FST::FST typeWrite("", 6,

FST::NODE(1, FST::RELATION('w', 1)),

FST::NODE(1, FST::RELATION('r', 2)),

FST::NODE(1, FST::RELATION('i', 3)),

FST::NODE(1, FST::RELATION('t', 4)),

FST::NODE(1, FST::RELATION('e', 5)),

FST::NODE()

);

FST::FST typeWriteln("", 8,

FST::NODE(1, FST::RELATION('w', 1)),

FST::NODE(1, FST::RELATION('r', 2)),

FST::NODE(1, FST::RELATION('i', 3)),

FST::NODE(1, FST::RELATION('t', 4)),

FST::NODE(1, FST::RELATION('e', 5)),

FST::NODE(1, FST::RELATION('l', 6)),

FST::NODE(1, FST::RELATION('n', 7)),

FST::NODE()

);

FST::FST typeIf("", 6,

FST::NODE(1, FST::RELATION('p', 1)),

FST::NODE(1, FST::RELATION('r', 2)),

FST::NODE(1, FST::RELATION('o', 3)),

FST::NODE(1, FST::RELATION('v', 4)),

FST::NODE(1, FST::RELATION('i', 5)),

FST::NODE()

);

FST::FST typeElse("", 10,

FST::NODE(1, FST::RELATION('o', 1)),

FST::NODE(1, FST::RELATION('t', 2)),

FST::NODE(1, FST::RELATION('h', 3)),

FST::NODE(1, FST::RELATION('e', 4)),

FST::NODE(1, FST::RELATION('r', 5)),

FST::NODE(1, FST::RELATION('w', 6)),

FST::NODE(1, FST::RELATION('i', 7)),

FST::NODE(1, FST::RELATION('s', 8)),

FST::NODE(1, FST::RELATION('e', 9)),

FST::NODE()

);

FST::FST typeLENLINE("", 8,

FST::NODE(1, FST::RELATION('l', 1)),

FST::NODE(1, FST::RELATION('e', 2)),

FST::NODE(1, FST::RELATION('n', 3)),

FST::NODE(1, FST::RELATION('l', 4)),

FST::NODE(1, FST::RELATION('i', 5)),

FST::NODE(1, FST::RELATION('n', 6)),

FST::NODE(1, FST::RELATION('e', 7)),

FST::NODE()

);

FST::FST typeRANDOM("", 7,

FST::NODE(1, FST::RELATION('r', 1)),

FST::NODE(1, FST::RELATION('a', 2)),

FST::NODE(1, FST::RELATION('n', 3)),

FST::NODE(1, FST::RELATION('d', 4)),

FST::NODE(1, FST::RELATION('o', 5)),

FST::NODE(1, FST::RELATION('m', 6)),

FST::NODE()

);

FST::FST typeCONCAT("", 7,

FST::NODE(1, FST::RELATION('c', 1)),

FST::NODE(1, FST::RELATION('o', 2)),

FST::NODE(1, FST::RELATION('n', 3)),

FST::NODE(1, FST::RELATION('c', 4)),

FST::NODE(1, FST::RELATION('a', 5)),

FST::NODE(1, FST::RELATION('t', 6)),

FST::NODE()

);

FST::FST typeTrue("", 5,

FST::NODE(1, FST::RELATION('t', 1)),

FST::NODE(1, FST::RELATION('r', 2)),

FST::NODE(1, FST::RELATION('u', 3)),

FST::NODE(1, FST::RELATION('e', 4)),

FST::NODE()

);

FST::FST typeFalse("", 6,

FST::NODE(1, FST::RELATION('f', 1)),

FST::NODE(1, FST::RELATION('a', 2)),

FST::NODE(1, FST::RELATION('l', 3)),

FST::NODE(1, FST::RELATION('s', 4)),

FST::NODE(1, FST::RELATION('e', 5)),

FST::NODE()

);

Листинг 2 – Конечные автоматы для ключевых слов языка

**Приложение В**

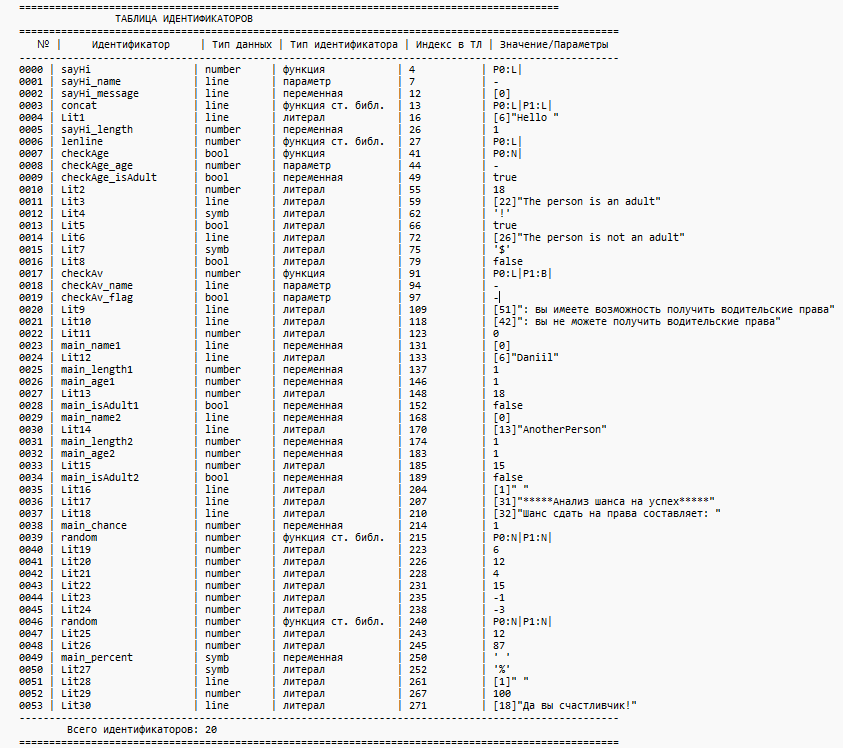


Рисунок В.1 – Таблица идентификаторов

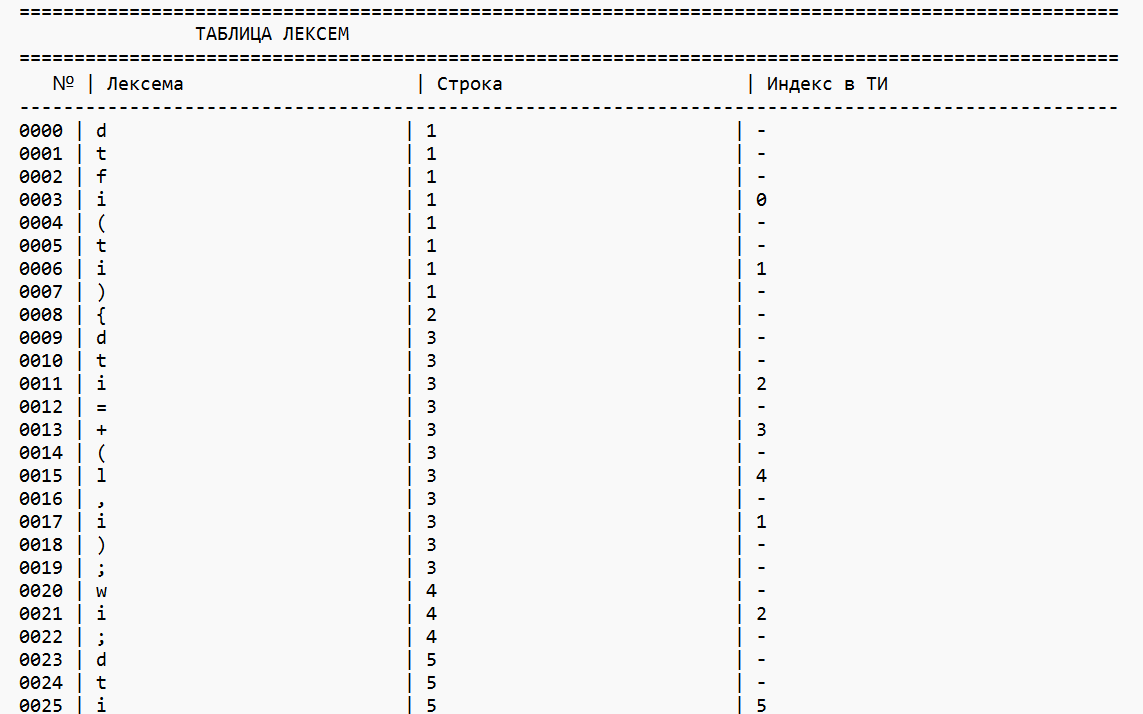


Рисунок В.2 – Начало таблицы лексем

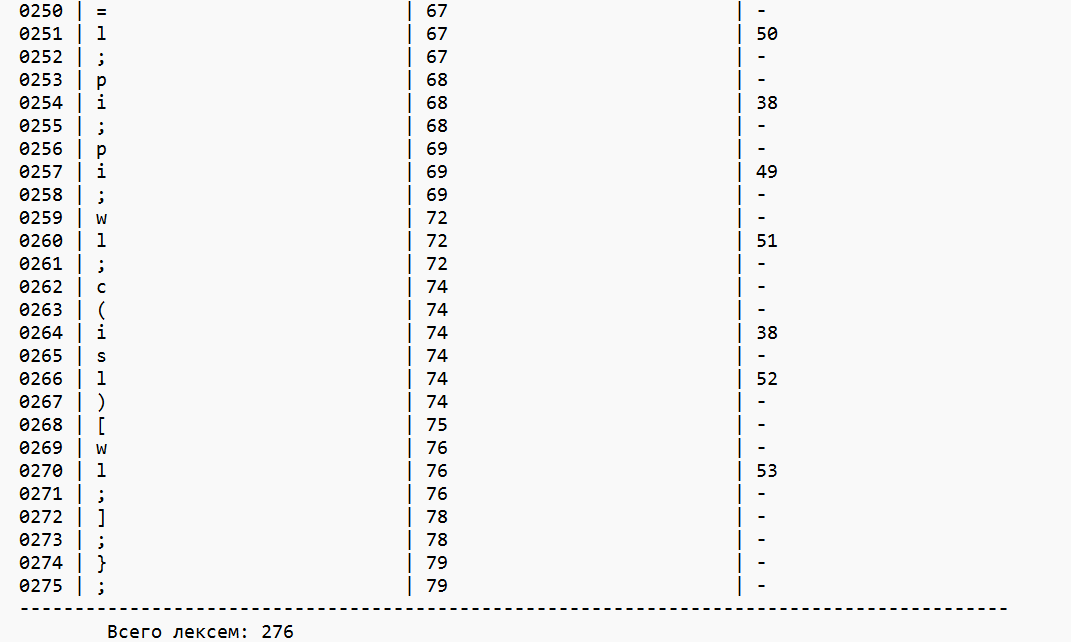


Рисунок В.2 Окончание таблицы лексем

# **Приложение Г**

Таблица 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Описание |
| S | S->dtfiOBS  S->m{N}; | Стартовые правила, описывающее общую структуру программы |
| O | O->()  O->(F) | Правила списка параметров функции |
| F | F->ti  F->ti,F | Правила для параметров функции при её объявлении |
| B | B->{NrI;};  B->{rI;}; | Правила для тела функции |
| N | N->+K;  N->/K;  N->zK;  N->+K;N  N->/K;N  N->zK;N  N->dti;  N->dti;N  N->dti = E;  N->dti=E;N  N->i = E;  N->i=E;N  N->pE;  N->pE;N  N->wE;  N->wE;N  N->c(R)[X];  N->c(R)[X];N  N->c(R)[X]![X];  N->c(R)[X]![X];N  N->iK;N  N->iK;  N->rI;  N->rI;N  N->dti=R;  N->dti=R;N  N->i=R;  N->i=R;N | Правила для конструкций внутри функций |
| K | K->(W)  K->() | Правила для списка параметров, передаваемых в функцию |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| W | W->i  W->l  W->y  W->-l  W->-l,W  W->i,W  W->l,W  W->y,W | Правила для параметров, передаваемых в функцию |
| E | E->i  E->l  E->y  E->-l  E->/KM  E->+KM  E->zKM  E->/K  E->+K  E->zK  E->iK  E->iKM  E->iM  E->lM  E->yM  E->-lM  E->(E)  E->(E)M | Правила для выражений |
| M | M->vE  M->vEM | Правила для арифметических операций |
| R | R->i  R->l  R->y  R->isi  R->isl  R->lsi  R->lsl  R->ysy  R->ysi  R->ysl  R->isy  R->lsy | Правила для логических операций |
| X | X->dti;N  X->dti;  X->dti=E; | Правила для тела условного оператора |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  | X->dti=E;X  X->dti=R;  X->dti=R;X  X->i=R;X  X->i=R;  X->rI;X  X->rI;  X->i=E;X  X->i=E;  X->pE;X  X->pE;  X->wE;X  X->wE;  X->+K;X  X->+K;  X->zK;X  X->zK;  X->/K;X  X->/K;  X->iK;X  X->iK; |  |
| I | I->i  I->l  I->y | Правила для возвращаемого значения из функции |

Приложение Д

namespace GRB

{

struct Rule

{

GRBALPHABET nn;

int iderror;

short size;

struct Chain

{

short size;

GRBALPHABET\* nt;

Chain() { size = 0; nt = 0; }

Chain(short psize, GRBALPHABET s, ...);

char\* getCChain(char\* b);

static GRBALPHABET T(char t) { return GRBALPHABET(t); }

static GRBALPHABET N(char n) { return -GRBALPHABET(n); }

static bool isT(GRBALPHABET s) { return s > 0; }

static bool isN(GRBALPHABET s) { return !isT(s); }

static char alphabet\_to\_char(GRBALPHABET s) { return isT(s) ? char(s) : char(-s); }

}\*chains;

Rule() { nn = 0x00; size = 0; }

Rule(GRBALPHABET pnn, int iderror, short psize, Chain c, ...);

char\* getCRule(char\* b, short nchain);

short getNextChain(GRBALPHABET t, Rule::Chain& pchain, short j);

};

struct Greibach

{

short size;

GRBALPHABET startN;

GRBALPHABET stbottomT;

Rule\* rules;

Greibach() { short size = 0; startN = 0; stbottomT = 0; rules = 0; }

Greibach(GRBALPHABET pstartN, GRBALPHABET pstbottomT, short psize, Rule r, ...);

short getRule(GRBALPHABET pnn, Rule& prule);

Rule getRule(short n);

};

Greibach getGreibach();

}

Листинг 3 – Структура грамматики Грейбах

namespace MFST

{

struct MfstState

{

short lenta\_position;

short nrule;

short nrulechain;

MFSTSTSTACK st;

MfstState();

MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);

MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);

};

struct Mfst

{

enum RC\_STEP

{

NS\_OK,

NS\_NORULE,

NS\_NORULECHAIN,

NS\_ERROR,

TS\_OK,

TS\_NOK,

LENTA\_END,

SURPRISE

};

struct MfstDiagnosis

{

short lenta\_position;

RC\_STEP rc\_step;

short nrule;

short nrule\_chain;

MfstDiagnosis();

MfstDiagnosis(short plenta\_position, RC\_STEP prc\_step, short pnrule, short pnrule\_chain);

} diagnosis[MFST\_DIAGN\_NUMBER];

GRBALPHABET\* lenta;

short lenta\_position;

short nrule;

short nrulechain;

short lenta\_size;

GRB::Greibach grebach;

LT::LexTable lexTable;

MFSTSTSTACK st;

std::stack<MfstState> storestate;

Mfst();

Mfst(LT::LexTable plex, GRB::Greibach pgrebach);

char\* getCSt(char\* buf);

char\* getCLenta(char\* buf, short pos, short n = 25);

char\* getDiagnosis(short n, char\* buf);

bool savestate(ofstream\* stream);

bool reststate(ofstream\* stream);

bool push\_chain(GRB::Rule::Chain chain);

RC\_STEP step(ofstream\* stream);

bool start(ofstream\* stream);

bool savediagnosis(RC\_STEP pprc\_step);

void printrules(ofstream\* stream);

struct Deducation

{

short size;

short\* nrules;

short\* nrulechains;

Deducation() { size = 0; nrules = 0; nrulechains = 0; }

} deducation;

bool savededucation();

};

}

Листинг 4 – Структура магазинного конечного автомата

Приложение Е

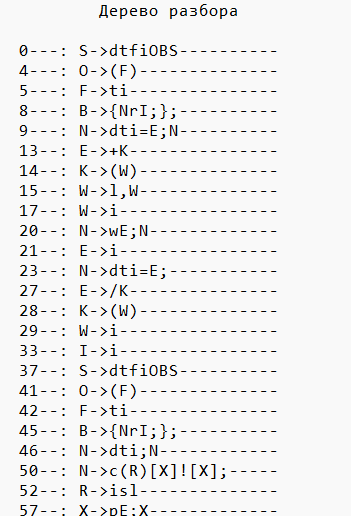


Рисунок E.1 – Начало дерева разбора

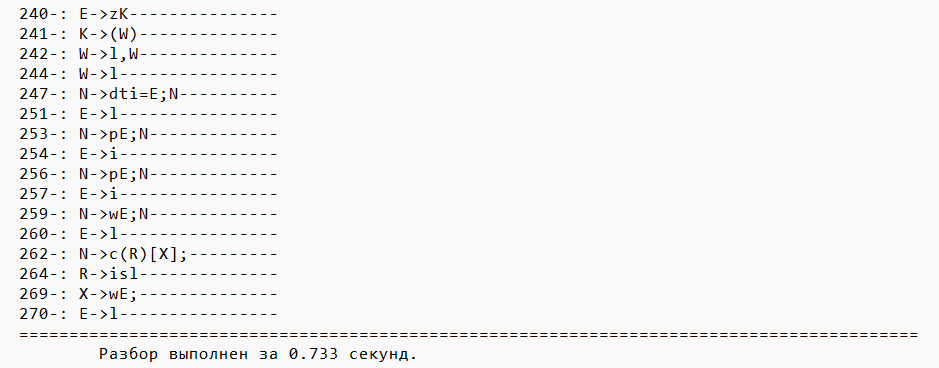


Рисунок Е.2 – Окончание дерева разбора

Приложение Ж

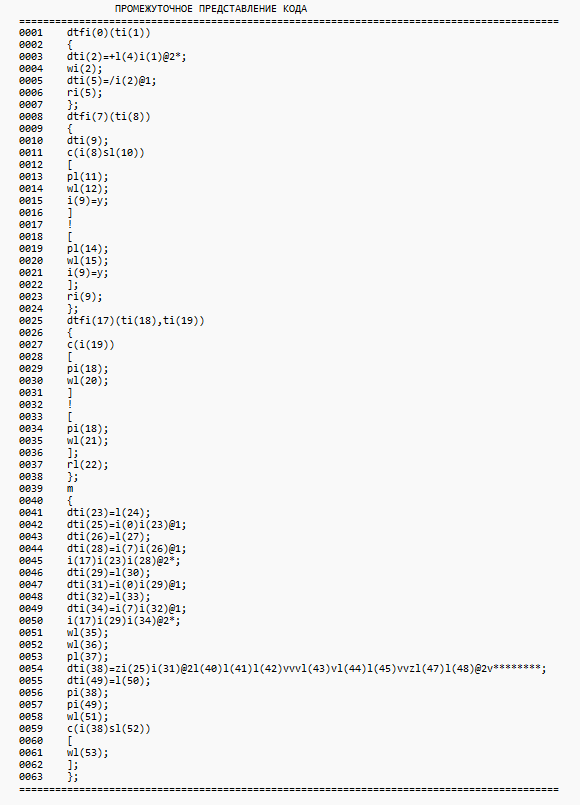
****

Рисунок Ж.1 – Промежуточное представление кода

Приложение З

.586P

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib ..\Debug\Library.lib

ExitProcess PROTO :DWORD

EXTRN CONCAT: proc

EXTRN LENLINE: proc

EXTRN RANDOM: proc

EXTRN WriteNumb: proc

EXTRN WriteLine: proc

EXTRN WriteBool: proc

EXTRN WriteSymb: proc

EXTRN WriteNumbLn: proc

EXTRN WriteLineLn: proc

EXTRN WriteBoolLn: proc

EXTRN WriteSymbLn: proc

.stack 4096

.const

null\_division BYTE "Exception: деление на ноль", 0

Lit1 BYTE "Hello ", 0

Lit2 SWORD 18

Lit3 BYTE "The person is an adult", 0

Lit4 BYTE "!", 0

Lit5 SWORD 1

Lit6 BYTE "The person is not an adult", 0

Lit7 BYTE "$", 0

Lit8 SWORD 0

Lit9 BYTE ": вы имеете возможность получить водительские права", 0

Lit10 BYTE ": вы не можете получить водительские права", 0

Lit11 SWORD 0

Lit12 BYTE "Daniil", 0

Lit13 SWORD 18

Lit14 BYTE "AnotherPerson", 0

Lit15 SWORD 15

Lit16 BYTE " ", 0

Lit17 BYTE "\*\*\*\*\*Анализ шанса на успех\*\*\*\*\*", 0

Lit18 BYTE "Шанс сдать на права составляет: ", 0

Lit19 SWORD 6

Lit20 SWORD 12

Lit21 SWORD 4

Lit22 SWORD 15

Lit23 SWORD -1

Lit24 SWORD -3

Lit25 SWORD 12

Lit26 SWORD 87

Lit27 BYTE "%", 0

Lit28 BYTE " ", 0

Lit29 SWORD 100

Lit30 BYTE "Да вы счастливчик!", 0

.data

buffer BYTE 256 dup(0)

sayHi\_message DWORD ?

sayHi\_length SWORD 1

checkAge\_isAdult SWORD 1

main\_name1 DWORD ?

main\_length1 SWORD 1

main\_age1 SWORD 1

main\_isAdult1 SWORD 1

main\_name2 DWORD ?

main\_length2 SWORD 1

main\_age2 SWORD 1

main\_isAdult2 SWORD 1

main\_chance SWORD 1

main\_percent DWORD ?

.code

f\_sayHi PROC sayHi\_name : DWORD

push offset Lit1

push sayHi\_name

pop dx

pop dx

push sayHi\_name

push offset Lit1

push offset buffer

call CONCAT

push eax

pop sayHi\_message

push sayHi\_message

call WriteLineLn

push sayHi\_message

pop dx

push sayHi\_message

call LENLINE

push ax

pop sayHi\_length

push sayHi\_length

jmp local0

local0:

pop eax

ret

f\_sayHi ENDP

f\_checkAge PROC checkAge\_age : SWORD

mov ax, checkAge\_age

cmp ax, Lit2

jge m0

jle m1

m0:

push offset Lit3

call WriteLine

push offset Lit4

call WriteSymbLn

push Lit5

pop ax

cmp ax, 0

je l0

jne l1

l0:

mov ax, 0

push ax

jmp endofexpr0

l1:

mov ax, 1

push ax

endofexpr0:

pop checkAge\_isAdult

jmp e0

m1:

push offset Lit6

call WriteLine

push offset Lit7

call WriteSymbLn

push Lit8

pop ax

cmp ax, 0

je l2

jne l3

l2:

mov ax, 0

push ax

jmp endofexpr1

l3:

mov ax, 1

push ax

endofexpr1:

pop checkAge\_isAdult

e0:

push checkAge\_isAdult

jmp local1

local1:

pop eax

ret

f\_checkAge ENDP

f\_checkAv PROC checkAv\_name : DWORD, checkAv\_flag : SWORD

mov ax, checkAv\_flag

cmp ax, 0

je l4

jne l5

l4:

mov ax, 0

push ax

jmp endofexpr2

l5:

mov ax, 1

push eax

endofexpr2:

pop checkAv\_flag

mov ax, checkAv\_flag

cmp ax, 0

jnz m2

jz m3

m2:

push checkAv\_name

call WriteLine

push offset Lit9

call WriteLineLn

jmp e1

m3:

push checkAv\_name

call WriteLine

push offset Lit10

call WriteLineLn

e1:

push 0

jmp local2

local2:

pop eax

ret

f\_checkAv ENDP

main PROC

push offset Lit12

pop main\_name1

push main\_name1

pop dx

push main\_name1

call f\_sayHi

push ax

pop main\_length1

push Lit13

pop main\_age1

push main\_age1

pop dx

movsx eax, main\_age1

push eax

call f\_checkAge

push ax

pop ax

cmp ax, 0

je l6

jne l7

l6:

mov ax, 0

push ax

jmp endofexpr3

l7:

mov ax, 1

push ax

endofexpr3:

pop main\_isAdult1

movsx eax, main\_isAdult1

push eax

push main\_name1

call f\_checkAv

push offset Lit14

pop main\_name2

push main\_name2

pop dx

push main\_name2

call f\_sayHi

push ax

pop main\_length2

push Lit15

pop main\_age2

push main\_age2

pop dx

movsx eax, main\_age2

push eax

call f\_checkAge

push ax

pop ax

cmp ax, 0

je l8

jne l9

l8:

mov ax, 0

push ax

jmp endofexpr4

l9:

mov ax, 1

push ax

endofexpr4:

pop main\_isAdult2

movsx eax, main\_isAdult2

push eax

push main\_name2

call f\_checkAv

push offset Lit16

call WriteLineLn

push offset Lit17

call WriteLineLn

push offset Lit18

call WriteLine

push main\_length1

push main\_length2

pop dx

pop dx

movsx eax, main\_length2

push eax

movsx eax, main\_length1

push eax

call RANDOM

push ax

push Lit19

push Lit20

push Lit21

pop bx

pop ax

cmp bx, 0

je nulldiv

cwd

idiv bx

push ax

pop ax

pop bx

mul bx

push ax

pop ax

pop bx

add ax, bx

push ax

push Lit22

pop bx

pop ax

sub ax, bx

push ax

push Lit23

push Lit24

pop ax

pop bx

mul bx

push ax

pop bx

pop ax

sub ax, bx

push ax

push Lit25

push Lit26

pop dx

pop dx

movsx eax, Lit26

push eax

movsx eax, Lit25

push eax

call RANDOM

push ax

pop ax

pop bx

add ax, bx

push ax

pop main\_chance

push offset Lit27

pop main\_percent

movsx eax, main\_chance

push eax

call WriteNumb

push main\_percent

call WriteSymb

push offset Lit28

call WriteLineLn

mov ax, main\_chance

cmp ax, Lit29

je m4

jne m5

m4:

push offset Lit30

call WriteLineLn

m5:

theend:

push 0

call ExitProcess

nulldiv PROC

push offset null\_division

call WriteLineLn

push -1

call ExitProcess

nulldiv ENDP

main ENDP

end main

Листинг 5 – Результат генерации ассемблерного кода