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 PDI-2024»

Выполнил студент Подшиваленко Диана Игоревна

(Ф.И.О.)

Руководитель проекта преп.-стаж. Волчек Д.И.

(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

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

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

Нормоконтролер

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

Курсовой проект защищен с оценкой

Минск 2024

Содержание

[Введение 5](#_Toc184675624)

[1 Спецификация языка программирования 6](#_Toc184675625)

[1.1 Характеристика языка программирования 6](#_Toc184675626)

[1.2 Определение алфавита языка программирования 6](#_Toc184675627)

[1.3 Применяемые сепараторы 6](#_Toc184675628)

[1.4 Применяемые кодировки 7](#_Toc184675629)

[1.5 Типы данных 7](#_Toc184675630)

[1.6 Преобразование типов данных 8](#_Toc184675631)

[1.7 Идентификаторы 8](#_Toc184675632)

[1.8 Литералы 9](#_Toc184675633)

[1.9 Объявление данных 9](#_Toc184675634)

[1.10 Инициализация данных 9](#_Toc184675635)

[1.11 Инструкции языка 9](#_Toc184675636)

[1.12 Операции языка 10](#_Toc184675637)

[1.13 Выражения и их вычисление 11](#_Toc184675638)

[1.14 Конструкции языка 11](#_Toc184675639)

[1.15 Область видимости идентификаторов 12](#_Toc184675640)

[1.16 Семантические проверки 12](#_Toc184675641)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc184675642)

[1.18 Стандартная библиотека и ее состав 12](#_Toc184675643)

[1.19 Ввод и вывод данных 13](#_Toc184675644)

[1.20 Точка входа 14](#_Toc184675645)

[1.21 Препроцессор 14](#_Toc184675646)

[1.22 Соглашение о вызовах 14](#_Toc184675647)

[1.23 Объектный код 14](#_Toc184675648)

[1.24 Классификация сообщений транслятора 14](#_Toc184675649)

[1.25 Контрольный пример 14](#_Toc184675650)

[2 Структура транслятора 15](#_Toc184675651)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 15](#_Toc184675652)

[2.2 Перечень входных параметров транслятора 16](#_Toc184675653)

[2.3 Протоколы, формируемые транслятором 16](#_Toc184675654)

[3 Разработка лексического анализатора 18](#_Toc184675655)

[3.1 Структура лексического анализатора 18](#_Toc184675656)

[3.2 Контроль входных символов 18](#_Toc184675657)

[3.3 Удаление избыточных символов 18](#_Toc184675658)

[3.4 Перечень ключевых слов 19](#_Toc184675659)

[3.5 Основные структуры данных 21](#_Toc184675660)

[3.6 Структура и перечень сообщений лексического анализатора 23](#_Toc184675661)

[3.7 Принцип обработки ошибок 23](#_Toc184675662)

[3.8 Параметры лексического анализатора 23](#_Toc184675663)

[3.9 Алгоритм лексического анализа 23](#_Toc184675664)

[3.10 Контрольный пример 24](#_Toc184675665)

[4 Разработка синтаксического анализатора 26](#_Toc184675666)

[4.1 Структура синтаксического анализатора 26](#_Toc184675667)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 26](#_Toc184675668)

[4.3 Построение конечного магазинного автомата 28](#_Toc184675669)

[4.4 Основные структуры данных 29](#_Toc184675670)

[4.5 Описание алгоритма синтаксического разбора 33](#_Toc184675671)

[4.6 Параметры синтаксического анализатора 34](#_Toc184675672)

[4.7 Структура и перечень сообщений синтаксического анализатора 34](#_Toc184675673)

[4.8 Принцип обработки ошибок 35](#_Toc184675674)

[4.9 Контрольный пример 35](#_Toc184675675)

[5 Разработка семантического анализатора 36](#_Toc184675676)

[5.1 Структура семантического анализатора 36](#_Toc184675677)

[5.2 Функции семантического анализатора 36](#_Toc184675678)

[5.3 Структура и перечень сообщений семантического анализатора 37](#_Toc184675679)

[5.4 Принцип обработки ошибок 38](#_Toc184675680)

[5.5 Контрольный пример 38](#_Toc184675681)

[6 Вычисление выражений 39](#_Toc184675682)

[6.1 Выражения, допускаемые языком 39](#_Toc184675683)

[6.2 Польская запись и принципы ее построения 39](#_Toc184675684)

[6.3 Программная реализация обработки выражений 40](#_Toc184675685)

[6.4 Контрольный пример 40](#_Toc184675686)

[7 Генерация кода 40](#_Toc184675687)

[7.1 Структура генератора кода 40](#_Toc184675688)

[7.2 Представление типов данных в оперативной памяти 41](#_Toc184675689)

[7.3 Статическая библиотека 42](#_Toc184675690)

[7.4 Особенности алгоритма генерации кода 42](#_Toc184675691)

[7.5 Входные параметры, управляющие генерацией кода 43](#_Toc184675692)

[7.6 Контрольный пример 43](#_Toc184675693)

[8 Тестирование транслятора 44](#_Toc184675694)

[8.1 Общие положения 44](#_Toc184675695)

[8.2 Результаты тестирования 44](#_Toc184675696)

[Заключение 48](#_Toc184675697)

[Список использованных источников 48](#_Toc184675698)

[Приложение А 49](#_Toc184675699)

[Приложение Б 50](#_Toc184675700)

[Приложение В 55](#_Toc184675701)

[Приложение Г 56](#_Toc184675702)

[Приложение Д 59](#_Toc184675703)

[Приложение Е 61](#_Toc184675704)

[Приложение Ж 70](#_Toc184675705)

[Приложение И 71](#_Toc184675706)

[Приложение К 73](#_Toc184675707)

[Приложение Л 74](#_Toc184675708)

# **Введение**

Данный курсовой проект посвящен разработке собственного языка программирования, который был назван PDI-2024. Он представляет собой простой учебный язык, который содержит 4 типа данных, позволяет выполнять арифметические действия, побитовые операции, операции сравнения, создавать функции, имеет условный оператор и оператор цикла.

Для создания нового языка нужно разработать собственный компилятор.

Этапы разработки компилятора для языка PDI-2024:

* Написание спецификации языка программирования;
* Разработка лексического анализатора;
* Создание синтаксического анализатора;
* Разработка семантического анализатора;
* Преобразование арифметических выражений в ПОЛИЗ;
* Генерация кода в язык ассемблер;
* Тестирование транслятора.

Информация о каждом из этапов приведена в соответствующих разделах пояснительной записки.

В первом разделе представлена спецификация языка – точное формализованное описание набора правил, определяющих синтаксис и семантику языка.

Во втором разделе содержится описание структуры транслятора, перечень входных параметров для управления работой транслятора, а также информация о протоколах, которые формирует транслятор.

В третьем разделе описан принцип работы лексического анализатора, определены допустимые символы языка и ключевые слова, а также описаны основные структуры данных, используемые для реализации лексического анализатора.

В четвертом разделе дано описание принципа работы синтаксического анализатора, определена формальная грамматика языка и приведена в нормальную форму Грейбах для выполнения синтаксического разбора.

Пятый раздел посвящен описанию принципа работы и основных функций семантического анализатора.

В шестом разделе описываются выражения, которые допустимы языком, а также принципы построения и вычисления выражений в ПОЛИЗ.

В седьмом разделе представлено описание процесса генерации кода в язык ассемблер, описана статическая библиотека языка, приведен алгоритм генерации кода.

В восьмом разделе приведены примеры тестирования транслятора.

# **1 Спецификация языка программирования**

# **1.1 Характеристика языка программирования**

Язык программирования PDI-2024 является универсальным языком высокого уровня. Он императивный, компилируемый, со строгой типизацией, которая происходит во время компиляции (статическая типизация).

# **1.2 Определение алфавита языка программирования**

Язык программирования PDI-2024 использует таблицу символов Windows-1251, в которой определены русские и латинские буквы, символы операций, символы-сепараторы и специальные символы. Структура алфавита языка PDI-2024, описанная в формате РБНФ представлена в таблице 1.1.

Таблица 1.1 – Алфавит языка программирования PDI-2024

|  |
| --- |
| <прописная буква латинского алфавита>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |
| <прописная буква русского алфавита>::= А|Б|В|Г|Д|Е|Ё|Ж|З|И|Й|К|Л|М|Н|О|П|Р|С|Т|У|Ф|Х|Ц|Ч|Ш|Щ|Ъ|Ь|Э|Ю|Я |
| <строчная буква латинского алфавита>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |
| <строчная буква русского алфавита>::= а|б|в|г|д|е|ё|ж|з|и|й|к|л|м|н|о|п|р|с|т|у|ф|х|ц|ч|ш|щ|ъ|ь|э|ю|я |
| <сепаратор>::=' '|,|(|)|{|}|; |
| <арифметическая операция>::= +|-|/|\*|% |
| <операция сравнения>::= >|>=|<|<=|==|!= |
| <побитовая операция>::= &| | |
| <цифра>::= 0|1|2|3|4|5|6|7|8|9 |

Код на языке программирования PDI-2024 может содержать символы латинского алфавита в верхнем и нижнем регистре, цифры, символы арифметических, побитовых операций, операций сравнения, а также символы разделители. Символы русского алфавита в верхнем и нижнем регистре разрешены только в строковых литералах.

# **1.3 Применяемые сепараторы**

В языке программирования PDI-2024 определен набор сепараторов для отделения лексем друг от друга. Все сепараторы и их назначение представлены ниже в таблице 1.2.

Таблица 1.2 – Сепараторы языка PDI-2024

| Сепаратор | Назначение сепаратора |
| --- | --- |
| {} | Программный блок |

Продолжение таблицы 1.2 – Сепараторы языка PDI-2024

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| Пробел | Разделитель лексем. Может отсутствовать. В названиях идентификаторов и ключевых слов не допускается |
| ; | Конец инструкции |
| , | Разделитель параметров функции |
| = | Оператор присваивания |
| >, <, !=, ==, >=, <= | Операторы сравнения |
| (,) | Блок параметров функции, изменение приоритетности в выражениях, а также блок условий |
| &, | | Побитовые операции |
| +, -, \*, /, % | Арифметические операции |

# **1.4 Применяемые кодировки**

Для написания исходного кода программы на языке PDI-2024 используется кодировка Windows-1251. На рисунке 1.1 представлена таблица кодировки Windows-1251.



Рисунок 1.1 – Таблица кодировки Windows-1251

# **1.5 Типы данных**

В языке программирования PDI-2024 допустимы 4 типа данных: беззнаковый целочисленный uint (4 байта), строковый string, вещественный double (8 байт), логический bool (1 байт). Базовые типы, поддерживаемые языком, представлены в таблице 1.3.

Таблица 1.3 – Базовые типы данных

|  |  |
| --- | --- |
| Тип | Описание |
| uint | Является беззнаковым целочисленным типом данных.  Занимает 4 байта.  Предусмотрен для объявления беззнаковых целочисленных данных.  Возможные значения: от 0 до 2^32-1  Инициализируется по умолчанию 0.  К данному типу применимы все арифметические и побитовые операции, а также операции сравнения. |
| double | Является вещественным типом данных.  Занимает 8 байт.  Предусмотрен для объявления вещественных данных. Возможные значения: −1.7976931348623157×10^308 до 1.7976931348623157×10^308  Инициализируется по умолчанию 0.0.  Этот тип поддерживает все арифметические операции, кроме получения остатка от деления, и операции сравнения. |
| bool | Является логическим типом данных.  Занимает 1 байт.  Предусмотрен для объявления логических данных.  Принцип представления в памяти:  Если true, то 1, если false, то 0.  Инициализируется по умолчанию false (0).  К логическому типу применимы все операции сравнения. |
| string | Является строковым типом данных.  Вся строка занимает n + 1 байт, где n – количество символов в строке и + 1 для символа конца строки.  Максимальная длина строки 255 символов.  Принцип размещения в памяти:  Каждый символ занимает 1 байт, символы расположены в памяти друг за другом. Строка оканчивается символом конца строки.  Инициализация по умолчанию: \0.  К строковым переменным недопустимо применение операций. |

Пользовательские типы данных не поддерживаются.

# **1.6 Преобразование типов данных**

Преобразования типов данных в языке программирования PDI-2024 не поддерживаются.

# **1.7 Идентификаторы**

Идентификаторы используются для наименования функций, параметров и переменных.

Имена идентификаторов могут содержать символы латинского алфавита нижнего и верхнего регистра. Имя идентификатора не должно совпадать с ключевыми словами.

Максимальная длина имени идентификатора равна шестнадцати символам.

Представление идентификатора в РБНФ:

<идентификатор>::= <прописная буква латинского алфавита>|< строчная буква латинского алфавита>|<прописная буква латинского алфавита><идентификатор>|<строчная буква латинского алфавита><идентификатор>

Пример правильных идентификаторов: a, b, num

Пример неправильных идентификаторов: b$, e#h

# **1.8 Литералы**

Литерал – это запись в исходном коде программы, представляющая собой фиксированное значение. В языке программирования PDI-2024 есть несколько разновидностей литералов: целочисленные, вещественные, строковые, логические. Описание литералов приведено в таблице 1.4.

Таблица 1.4 – Представление литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Целочисленные | Состоят только из цифр.  В восьмеричной системе счисления число начинается с 0. |
| Вещественные | Состоят из цифр и точки, отделяющей дробную часть. |
| Строковые | Состоит из символов, заключенных в ' ' (одинарные кавычки). |
| Логические | Логическое значение true, если истина, и false, если ложно, которые могут быть преобразованы в 1 и 0 соответственно. |

# **1.9 Объявление данных**

Для объявления переменной нужно указать сначала ее тип, а затем имя идентификатора. Допускается инициализация при объявлении. В языке программирования PDI-2024 обязательно объявление переменных и функций до их использования. У каждой переменной есть своя область видимости, ограниченная фигурными скобками, в которой определена эта переменная. Объявление глобальных переменных не допускается.

# **1.10 Инициализация данных**

Допустима инициализация при объявлении или присваивание значения уже объявленной ранее переменной.

Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций.

Значения по умолчанию: значение 0 для беззнакового целочисленного типа данных, 0.0 для вещественного типа данных, пустая строка ' ' для строкового типа данных и false (0) для логического типа данных.

# **1.11 Инструкции языка**

Возможные инструкции языка программирования PDI-2024 представлены в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление переменной | <тип данных><идентификатор>; |
| Инициализация | <тип данных><идентификатор> = <литерал>|<идентификатор>; |
| Объявление функции | function <тип данных><идентификатор>(<тип данных><идентификатор>, …){  тело функции  ret <идентификатор>|<литерал>;  }; |
| Объявление главной функции main | main{  тело функции  ret <идентификатор>|<литерал>;  }; |
| Присваивание | <идентификатор> = <литерал>|<идентификатор>|<выражение>; |
| Вывод данных | print <идентификатор>|<литерал>;  println <идентификатор>|<литерал>; |
| Ввод данных | read <идентификатор>; |
| Условный оператор | if(<логическое выражение>){…};  if(<логическое выражение>){…}  otherwise{…}; |
| Оператор цикла | until(<выражение>){…}; |
| Оператор возврата из функции | ret <идентификатор>|<литерал>; |

# **1.12 Операции языка**

Все операции выполняются над одним и тем же типом данных. Недопустимо выполнение операций над операндами разных типов. Результат будет иметь такой же тип данных, как и операнды, над которыми выполняется операция.

В таблице 1.6 представлены все операции языка PDI-2024.

Таблица 1.6 – Операции языка

|  |  |  |
| --- | --- | --- |
| Категория | Операторы | Пояснение |
| Арифметические | + – сложение  - – вычитание  \* – умножение  / – деление  % – взятие остатка от деления | Применимы к целочисленному и вещественному типу данных. |
| Операции сравнения | > – больше  >= – больше или равно  < – меньше  <= – меньше или равно  == – равно  != – не равно | Применимы к вещественным, целочисленным и логическим типам данных. |
| Побитовые | & – побитовое И  | – побитовое ИЛИ | Применимы только к целочисленным типам данных. |

Если у операций одинаковый приоритет, то первой будет выполнена операция, стоящая левее. С помощью круглых скобок может быть изменен приоритет операций.

# **1.13 Выражения и их вычисление**

Выражения являются важной составляющей языка программирования, так как позволяют удобно описывать алгоритмы. Для вычисления выражений применяются определенные нормы и правила. Правила вычисления выражений языка программирования PDI-2024:

* выражения читаются слева направо и записываются в одну строку;
* в выражении могут присутствовать только операнды одного типа;
* выражение может содержать вызов функций;
* допускается использование круглых скобок для изменения приоритета операций;
* реализация выражений происходит с помощью ПОЛИЗ.

# **1.14 Конструкции языка**

В языке PDI-2024 поддерживаются 4 конструкции: главная функция main, функция, тело которой определяет сам пользователь, условная конструкция и цикл. Функция может возвращать только 1 значение. Главная функция возвращает код, с которым завершилось выполнение программы. Функция обязательно должна возвращать значение. Для вызова функции необходимо указать ее имя, далее в круглых скобках через запятую перечислить параметры. Функции из стандартной библиотеки языка вызываются так же, как и пользовательские функции.

Конструкции языка программирования PDI-2024 представлены в таблице 1.7.

Таблица 1.7 – Конструкции языка

|  |  |
| --- | --- |
| Конструкция | Синтаксис |
| Главная функция(main) | main{  …  ret <идентификатор>|<литерал>;  }; |
| Функция | function <тип данных><идентификатор>(<тип данных><идентификатор>, …)  {  …  ret <идентификатор>|<литерал>;  }; |
| Оператор цикла | until(<условие>){  … }; |
| Условный оператор | if(<условие>){…};  или  if(<условие>){…}  otherwise{…}; |

# **1.15 Область видимости идентификаторов**

Идентификаторы функций имеют глобальную область видимости. Остальные идентификаторы могут быть объявлены локально, то есть внутри конструкции.

# **1.16 Семантические проверки**

Все возможные в языке программирования PDI-2024 семантические проверки представлены ниже в таблице 1.8.

Таблица 1.8 – Список семантических проверок

|  |  |
| --- | --- |
| **№** | **Правило** |
| **1** | **Должна присутствовать функция main** |
| **2** | **Идентификаторы не должны совпадать с ключевыми словами** |
| **3** | **Все функции должны быть объявлены перед main** |
| **4** | **Соответствие типов формальных и фактических параметров при вызове**  **функций** |
| **5** | **Нужно объявить переменную перед использованием** |
| **6** | **Нельзя переопределить идентификатор в той же области видимости** |
| **7** | **Возвращаемое значение функции должно совпадать с типом функции** |

Продолжение таблицы 1.8 – Список семантических проверок

|  |  |
| --- | --- |
| **№** | **Правило** |
| **8** | **Проверка на корректность скобок при выделении блоков кода (функции, блоки условных операторов и циклов)** |
| **9** | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу |
| **10** | Соответствие операторов типам данных, к которым они применимы |

Семантические проверки необходимы для нахождения логических ошибок, обеспечения правильного смысла кода, а также для согласования разных частей программы.

# **1.17 Распределение оперативной памяти на этапе выполнения**

Транслируемый код использует 2 области памяти. В сегмент констант записываются все допустимые литералы языка, в сегмент данных помещаются все имена переменных. Для запоминания промежуточных вычислений выражений используются 2 стека: для вещественных чисел используется стек математического сопроцессора, для всего остального – основной стек.

# **1.18 Стандартная библиотека и ее состав**

Язык программирования PDI-2024 предоставляет доступ к стандартной библиотеке, которая будет подключаться на этапе компоновки. Она написана на языке с++. Функции библиотеки представлены ниже в таблице 1.9.

Таблица 1.9 – Стандартная библиотека PDI-2024

|  |  |
| --- | --- |
| Функция | Характеристика |
| int compare(char\*, char\*) | Функция для сравнения строк. Возвращает значение целого типа. Принимает в качестве параметров две строки и возвращает 1, если строки равны, 2, если первая строка лексикографически больше второй, и 0, если вторая строка лексикографически больше первой. |
| int len(char\*) | Функция для получения длины строки. Возвращает значение целого типа. Принимает в качестве параметров одну строку и возвращает ее длину. |
| int waitPause() | Функция для приостановки закрытия консольного окна до нажатия любой клавиши пользователем. Возвращает 0 в случае успешного выполнения. |
| int readss(int, char\*) | Функция для чтения данных переданного типа. Возвращает значение целого типа. Принимает на вход тип в виде определенного числа и указатель на переменную, в которую должно быть записано значение. Возвращает 0, если операция выполнена успешно, и -1, если при чтении произошла ошибка. |

Продолжение таблицы 1.9 – Стандартная библиотека PDI-2024

|  |  |
| --- | --- |
| Функция | Характеристика |
| int printss(int, char\*) | Функция для вывода данных переданного типа на консоль без перевода строки. Возвращает значение целого типа. Принимает на вход тип в виде определенного числа и указатель на переменную, значение которой требуется вывести. Возвращает 0, если операция выполнена успешно. |
| int printssln(int, char\*) | Функция для вывода данных переданного типа на консоль с переводом строки. Возвращает значение целого типа. Принимает на вход тип в виде определенного числа и указатель на переменную, значение которой требуется вывести. Возвращает 0, если операция выполнена успешно. |
| int getmin(int, int) | Функция для нахождения меньшего из 2 целых чисел. Возвращает значение целого типа. Принимает на вход 2 числа и возвращает меньшее из них. |
| int getmax(int, int) | Функция для нахождения большего из 2 целых чисел. Возвращает значение целого типа. Принимает на вход 2 числа и возвращает большее из них. |

# **1.19 Ввод и вывод данных**

Стандартный вывод данных осуществляется с помощью функций стандартной библиотеки printss и printssln. В качестве аргумента могут быть числовые, строковые и логические литералы, а также идентификаторы:

printss <идентификатор>|<литерал>;

printssln <идентификатор>|<литерал>;

Стандартный ввод данных осуществляется с помощью функции стандартной библиотеки readss. В качестве аргумента выступают идентификаторы.

readss <идентификатор>;

Функции, управляющие выводом и вводом данных, реализованы на языке C++. На этапе генерации кода операторы вывода и ввода языка PDI-2024 заменяются на встроенные функции стандартной библиотеки этого языка.

# **1.20 Точка входа**

В языке программирования PDI-2024 точкой входа является ключевое слово main. Определение main обязательно, причем оно должно быть единственно.

# **1.21 Препроцессор**

Для языка программирования PDI-2024 препроцессор не предусмотрен.

# **1.22 Соглашение о вызовах**

В языке PDI-2024 используется стандартное соглашение о вызовах под названием stdcall – это значит, что в программе все параметры функции будут передаваться через стек справа налево и память будет освобождаться вызываемым кодом.

# **1.23 Объектный код**

Программа, исходный код которой написан на языке программирования PDI-2024, будет транслироваться в ассемблерный код, а далее из ассемблерного кода в объектный код**.**

# **1.24 Классификация сообщений транслятора**

Сообщения будут выводится в файл протокола и на консоль. Классификация сообщений представлена ниже в таблице 1.10.

Таблица 1.10 – Классификация ошибок

|  |  |
| --- | --- |
| **Код** | **Характеристика** |
| 0-99 | Системные ошибки |
| 100-101 | Ошибки параметров |
| 103-109 | Ошибки лексического анализа |
| 110-113 | Ошибки файлов |
| 114-119 | Ошибки таблиц лексем и идентификаторов |
| 102, 120-126 | Ошибки семантического анализа |
| 600-607 | Ошибки синтаксического анализа |

# **1.25 Контрольный пример**

Контрольный пример демонстрирует основные возможности языка программирования PDI-2024. Исходный код приведён в приложении А.

**2 Структура транслятора**

**2.1 Компоненты транслятора их назначение и принципы взаимодействия**

Транслятор – это программа, которая преобразует исходный код на одном языке программирования в исходный код на другом языке программирования.

На рисунке 2.1 представлена структура транслятора.

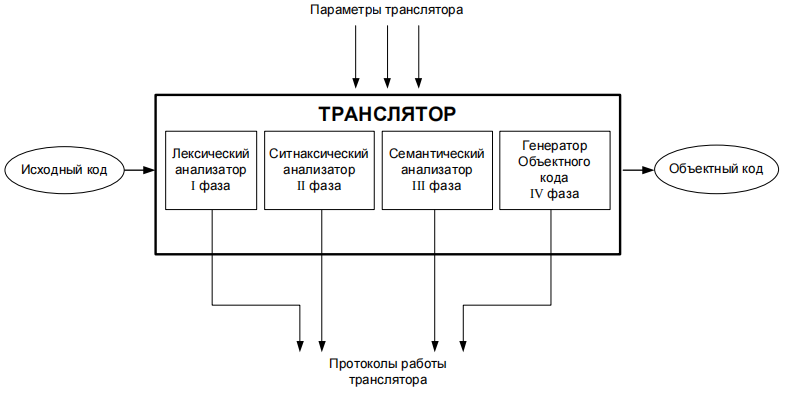


Рисунок 2.1 – Структура транслятора

Этапы, представленные на рисунке, выполняются последовательно. Каждый этап имеет свои входные и выходные данные, которые последовательно передаются на следующую фазу транслятора.

Первой фазой трансляции является лексический анализ. Он принимает на вход исходный код программы на языке высокого уровня. Задача лексического анализатора распознать лексемы и сформировать 2 таблицы: таблицу лексем и таблицу идентификаторов.

Синтаксический анализ представляет собой вторую часть работы транслятора. Входом для него является таблица лексем и таблица идентификаторов, полученные на предыдущем этапе. Задачей синтаксического анализатора является проверка кода на соответствие правилам грамматики и построение дерева разбора.

Третьей фазой является семантический анализ. Его задача – проверить выполнение семантических правил языка. На вход он принимает таблицу лексем, таблицу идентификаторов и дерево разбора.

Последний этап трансляции представляет собой генерацию кода. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с кодом на языке ассемблер.

**2.2 Перечень входных параметров транслятора**

Для того, чтобы получить файлы с результатами работы транслятора, необходимы входные параметры. Информация о данных параметрах представлена в таблице 2.1.

Таблица 2.1 – Входные параметры PDI-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Характеристика | Значение по умолчанию |
| -in:<путь к файлу> | Файл, содержащий исходный код на языке программирования PDI-2024. Файл имеет расширение .txt | Не задано |
| -out:<путь к файлу> | Файл, содержащий код на языке ассемблера. | <имя in-файла>.asm |
| -log:<путь к файлу> | Файл с протоколом работы транслятора | <имя in-файла>.log |

**2.3 Протоколы, формируемые транслятором**

Протоколы, которые будут формироваться транслятором, приведены в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором PDI-2024

|  |  |
| --- | --- |
| Протоколы и файлы | Характеристика |
| Файл, заданный параметром -log: | Файл с протоколом работы транслятора языка программирования PDI-2024. Содержит информацию о входных параметрах, общем количестве символов и строк (исходные данные). В случае ошибки текст, строка и позиция ошибки будет записан в файл. |
| IT.txt | Файл содержит таблицу идентификаторов, сформированную во время лексического анализа. |
| LT.txt | Файл содержит таблицу лексем, сформированную во время лексического анализа. |
| NewLT.txt | Файл содержит новую таблицу лексем, сформированную после преобразования выражений в обратную польскую запись. |
| ST.txt | Файл содержит дерево разбора, сформированное во время работы синтаксического анализатора. |
| Файл, заданный параметром -out: с расширением .asm | Содержит исходный код на языке ассемблера. |

**3 Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

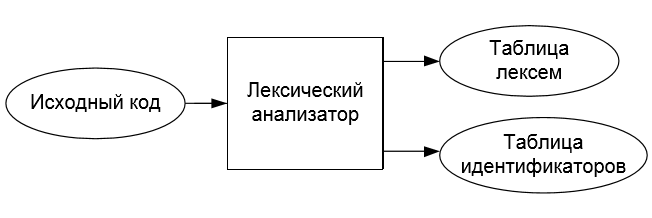
Лексический анализатор – это первая фаза транслятора, которая выполняет лексический анализ: весь текст исходного кода разбивается на лексемы, для создания промежуточного представления исходной программы, а также формируется таблица идентификаторов. Схема работы лексического анализатора представлена на рисунке 3.1.

Рисунок 3.1 – Схема работы лексического анализатора

На вход лексического анализатора поступает исходный код на языке программирования PDI-2024, на выходе формируются таблица лексем и таблица идентификаторов.

**3.2 Контроль входных символов**

Для проверки входных символов на допустимость есть специальная таблица, которая повторяет таблицу Windows-1251. Символы делятся на несколько категорий: T – разрешенные символы, S – пробельные символы и знаки табуляции, Z – знаки операций.

Таблица проверки входных символов представлена на рисунке 3.2.

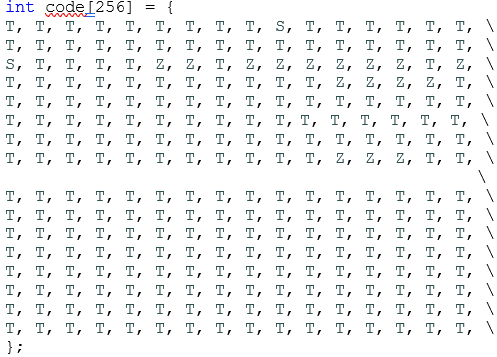


Рисунок 3.2 – Таблица проверки входных символов

**3.3 Удаление избыточных символов**

Избыточный символ – это символ, удаление которого никак не влияет на исходный текст программы. В языке PDI-2024 избыточными символами являются пробельные символы, знаки табуляции. Удаление данных символов предусмотрено до начала разбиения текста на лексемы.

Алгоритм удаления избыточных символов:

Пока не конец файла:

* читается символ;
* если символ является пробелом или знаком табуляции, то есть входит в категорию S:
* если перед или после символа стоят символы, которые относятся к категории Z:
* игнорируется текущий символ из категории S, так как он является избыточным;
* иначе данный символ категории S является сепаратором и его удаление приведет к некорректному анализу программы, а значит нужно его оставить.

**3.4 Перечень ключевых слов**

Все ключевые слова языка PDI-2024, сепараторы, символы операций, а также соответствующие им лексемы приведены в таблице 3.1.

Таблица 3.1 – Все ключевые слова, сепараторы, символы операций и соответствующие им лексемы

|  |  |  |
| --- | --- | --- |
| Ключевое слово | Характеристика | Лексема |
| uint | Целочисленный беззнаковый тип | t |
| string | Строковый тип | t |
| bool | Логический тип | t |
| double | Вещественный тип | t |
| идентификатор | Идентификатор любого доступного типа языка | i |
| литерал | Литерал любого доступного типа языка | l |
| main | Главная функция (точка входа в программу) | m |
| function | Указание на объявление функции | f |
| ret | Указание на выход из функции и возврат значения | r |

Продолжение таблицы 3.1 – Все ключевые слова, сепараторы, символы операций и соответствующие им лексемы

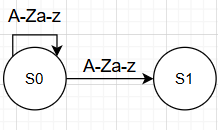
|  |  |  |
| --- | --- | --- |
| Ключевое слово | Характеристика | Лексема |
| print | Вывод данных на консоль без перевода строки | p |
| println | Вывод данных на консоль с переводом строки | x |
| read | Ввод данных | h |
| ; | Признак конца инструкции | ; |
| , | Разделитель параметров функции | , |
| { | Начало конструкции языка | { |
| } | Конец конструкции языка | } |
| ( | Признак начала перечисления параметров функции или приоритет в выражениях | ( |
| ) | Признак конца перечисления параметров функции или приоритет в выражениях | ) |
| + | Арифметический оператор сложения | + |
| - | Арифметический оператор вычитания | - |
| / | Арифметический оператор деления | / |
| \* | Арифметический оператор умножения | \* |
| % | Арифметический оператор взятия остатка от деления | % |
| = | Арифметический оператор присваивания | = |
| == | Оператор проверки на равенство | c |
| != | Оператор проверки на неравенство | n |
| > | Оператор больше | > |
| < | Оператор меньше | < |
| >= | Оператор больше или равно | e |
| <= | Оператор меньше или равно | o |

Продолжение таблицы 3.1 – Все ключевые слова, сепараторы, символы операций и соответствующие им лексемы

|  |  |  |
| --- | --- | --- |
| if | Условный оператор | y |
| otherwise | Условный оператор, ветвь else | d |
| until | Оператор цикла | w |
| & | Оператор побитового И | & |
| | | Оператор побитового ИЛИ | | |

Для распознавания ключевых слов используются конечные автоматы, в которых цепочки записаны в виде регулярных выражений.

На рисунках 3.3 и 3.4 представлены графы переходов для распознавания идентификатора и числового литерала соответственно.



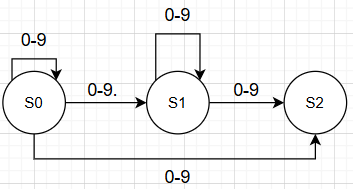
Рисунок 3.3 – Граф перехода для идентификатора

Рисунок 3.4 – Граф перехода для числового литерала

В приложении Б представлены графы конечных автоматов для распознавания ключевых слов.

**3.5 Основные структуры данных**

Основными структурами данных лексического анализатора в языке PDI-2024 являются таблица лексем и таблица идентификаторов.

В листинге 3.1 представлена реализация таблицы лексем.

|  |
| --- |
| struct Entry {  char lexema[LEXEMA\_FIXSIZE];  int sn;  int idxTI;  };  struct LexTable {  int maxsize = 0;  int size = 0;  Entry\* table;  }; |

Листинг 3.1 – Структура таблицы лексем

Структура с именем LexTable представляет собой экземпляр таблицы лексем, в которой maxsize – максимальный размер таблицы, size – текущий размер таблицы, table – указатель на строку таблицы лексем. Данная таблица предназначена для хранения записей обо всех лексемах, которые встречаются в исходном коде программы на языке программирования PDI-2024.

Структура с именем Entry представляет экземпляр строки таблицы лексем, в которой lexema хранит символьное обозначение лексемы, sn – номер строки в исходном коде, в которой располагается данная лексема, idTI – номер строки в таблице идентификаторов.

Перечисление с именем IDDATATYPE используется для задания каждому идентификатору типа данных, который он может хранить. Для беззнакового целочисленного INT = 1, для строкового STR = 2, для логического BOOL = 3, для вещественного DOUBLE = 4.

Перечисление с именем IDTYPE используется для задания каждому идентификатору его типа. Для переменной V = 1, для функции F = 2, для параметра функции P = 3, для литерала L = 4.

Структура с именем IdTable представляет собой экземпляр таблицы идентификаторов, в которой maxsize – максимальный размер таблицы, size – текущий размер таблицы, table – указатель на строку таблицы идентификаторов. Данная таблица предназначена для хранения записей обо всех идентификаторах, которые встречаются в исходном коде программы на языке программирования PDI-2024.

Структура с именем Entry представляет экземпляр строки таблицы идентификаторов, в которой idxfirstLE – индекс первого вхождения идентификатора в таблицу лексем, id – идентификатор, iddatatype – тип данных, который может хранить данный идентификатор, idtype – тип самого идентификатора, scope\_name – имя области видимости, в которой находится данный идентификатор и в которой он будет доступен, value – объединение, которое позволяет хранить значение целочисленного, вещественного и логического идентификатора в переменной vint и структура vstr, которая нужна для хранения значения строкового идентификатора, в которой есть поля len для хранения длины строки и str для хранения самой строки. Также присутствуют несколько конструкторов: c параметрами и по умолчанию.

В листинге 3.2 представлена реализация таблицы идентификаторов.

|  |
| --- |
| Enum IDDATATYPE { UINT = 1, STR = 2, BOOL = 3, DOUBLE = 4};  enum IDTYPE { V = 1, F = 2, P = 3, L = 4 };  struct Entry  {  int idxfirstLE;  char id[17];  IDDATATYPE iddatatype;  IDTYPE idtype;  string scope\_name;  union  {  double vint = NULL;  struct  {  int len;  char str[TI\_STR\_MAXSIZE - 1];  } vstr[TI\_STR\_MAXSIZE];  } value;  Entry(int IdxfirstLE, const char\* Id, IDDATATYPE Datatype, IDTYPE Type, string Scope, int value)  {  this->idxfirstLE = IdxfirstLE;  for (int i = 0; i < strlen(Id); i++)  {  this->id[i] = Id[i];  }  this->id[strlen(Id)] = '\0';  this->iddatatype = Datatype;  this->idtype = Type;  this->scope\_name = Scope;  this->value.vint = value;  }  Entry() = default;  };  struct IdTable {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.2 – Структура таблицы идентификаторов

**3.6 Структура и перечень сообщений лексического анализатора**

В случае возникновения ошибки на стадии лексического анализа формируется экземпляр ошибки в следующем формате: код ошибки в соответствии с таблицей ошибок, текст сообщения, номер строки в исходном коде, позиция в строке.

Номера ошибок и соответствующий им текст сообщения представлены в таблице 3.2.

Таблица 3.2 – Перечень сообщений ошибок лексического анализатора

|  |  |
| --- | --- |
| Номер ошибки | Текст сообщения |
| 103 | Функция main не объявлена |
| 104 | Ключевое слово не может быть идентификатором |
| 105 | Дублирование идентификатора |

Продолжение таблицы 3.2 – Перечень сообщений ошибок лексического анализатора

|  |  |
| --- | --- |
| Номер ошибки | Текст сообщения |
| 106 | Идентификатор не определён |
| 107 | Функция уже определена |
| 108 | Превышено максимально возможное число параметров в функции |
| 109 | Повторное объявление main |

**3.7 Принцип обработки ошибок**

Если была обнаружена ошибка при выполнении лексического анализа, то формируется сообщение об ошибке и выводится на консоль, а также записывается в файл протокола работы, заданный параметром –log: <имя файла>.log. Транслятор прекращает свою работу.

**3.8 Параметры лексического анализатора**

На вход лексического анализатора подается исходный текст программы на языке программирования PDI-2024, в которым заранее были удалены избыточные символы.

**3.9 Алгоритм лексического анализа**

Алгоритм лексического анализа описан ниже.

1. Читается посимвольно исходный код программы. Если текущий символ является концом строки, то переходим к пункту 8;
2. Если текущий символ является сепаратором, то он записывается в таблицу лексем, иначе записывается в буфер, пока не встретится сепаратор;
3. После этого строка, записанная в буфере, передается на распознавание конечным автоматам. При успешном разборе автомат вернет лексическому анализатору лексему, соответствующую данному ключевому слову, которое записывается в таблицу лексем. Если текущая лексема представляет собой идентификатор или литерал, то переход к пункту 4. В противном случае переход к пункту 1;
4. Если предыдущие лексемы являются ключевыми словами для обозначения типа данных, то текущему идентификатору или литералу присваивается данный тип;
5. Если текущая лексема идентификатор и происходит его объявление, то осуществляется поиск по таблице идентификаторов. В случае, если такой идентификатор уже существует в данной области видимости, то выводится ошибка и транслятор прекращает свою работу. В противном случае в таблицу идентификаторов заносится запись о текущем идентификаторе;
6. Если текущая лексема идентификатор и перед ним не указан тип данных, то есть предполагается, что он был объявлен ранее, то осуществляется поиск по таблице идентификаторов. В случае, если такого идентификатора не существует в данной области видимости, то поиск осуществляется в области видимости, предшествующей данной, пока не дойдет до глобальной. Если запись в таблице идентификаторов не была найдена, то выводится соответствующая ошибка и транслятор завершает свою работу. В противном случае запись в таблицу идентификаторов производится не будет, но будет сделана запись в таблицу лексем с ссылкой на найденную запись в таблице идентификаторов;
7. Если текущая лексема является литералом, то выявляется его тип и значение. Далее в таблицу идентификаторов помещается запись о текущем литерале. В таблицу лексем заносится данная лексема с ссылкой на таблицу идентификаторов. Переход в пункт 1;
8. Если идентификатор является функцией, то запись о ней будет помещена в таблицу идентификаторов с соответствующими типом возвращаемого значений. Последующие идентификаторы в круглых скобках будут записаны как параметры. Переход в пункт 1;
9. Если не дошли до конца исходного текста, то переход в пункт 1.

**3.10 Контрольный пример**

Результатом работы лексического анализатора является таблица лексем и таблица идентификаторов для контрольного примера, которые представлены в приложении В.

**4 Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

Синтаксический анализ является второй фазой транслятора языка PDI-2024 и выполняется сразу после завершения фазы лексического анализатора. Он предназначен для распознавания заранее заданных синтаксических правил. Структура синтаксического анализатора представлена на рисунке 4.1.

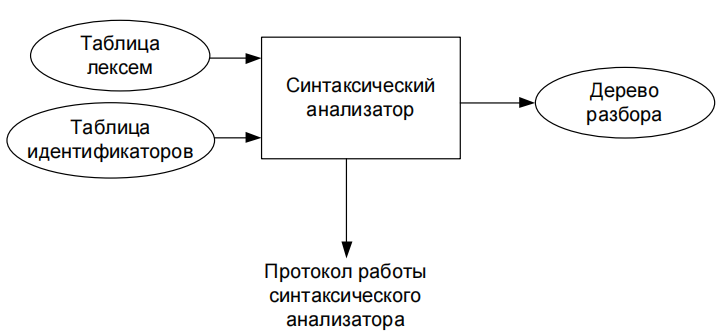


Рисунок 4.1 – Структура синтаксического анализатора

Входными данными для синтаксического анализатора являются таблица лексем и идентификаторов. Результатом работы будет дерево разбора.

**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка**

Синтаксический анализатор языка PDI-2024 использует грамматику типа 2 иерархии Хомского (контекстно-свободная грамматика) G = {N, T, P, S}, в которой:

* N – конечный алфавит нетерминальных символов;
* T – конечный алфавит терминальных символов;
* P – конечное множество правил грамматики;
* S – стартовый символ грамматики G.

Данная грамматика G имеет нормальную форму Грейбах, так как она не является леворекурсивной (не содержит леворекурсивных правил) и все правила из конечного множества P имеют вид:

* A → aα, где a ∈ T, α ∈ N\*;
* S → λ, где S ∈ N – начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Перед приведением грамматики к нормальной форме Грейбах необходимо:

* удалить все бесплодные символы;
* удалить все недостижимые символы из грамматики;
* удалить все лямбда-правила из грамматики;
* удалить все цепные правила.

Все синтаксические правила в нормальной форме Грейбах приведены в таблице 4.1.

Таблица 4.1 – Синтаксические правила в форме Грейбах

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Характеристика |
| S | m{Nri;};S  m{Nri;};  m{Nrl;};S  m{Nrl;};  fti(F){Nri;};S  fti(F){Nri;};  fti(F){Nrl;};S  fti(F){Nrl;};  fti(F){ri;};S  fti(F){ri;};  fti(F){rl;};S  fti(F){rl;}; | Правила, которые описывают порождение функции main, а также пользовательские функции. |
| N | ti;N  ti;  ti=E;N  ti=E;  i=E;N  i=E;  pi;N  pi;  pl;N  pl;  xi;N  xi;  xl;N  xl;  hi;N  hi;  wU{N};N  wU{N};  yU{N};N  yU{N};  yU{N};d{N};N  yU{N};d{N}; | Правила, которые порождают основные операторы языка. |
| E | i  iM  l  lM  (E)  (E)M  i(W)  i(W)M | Правила, которые порождают выражения. |

Продолжение таблицы 4.1 – Синтаксические правила в форме Грейбах

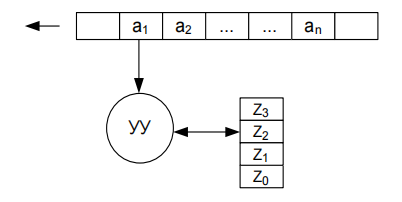
|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Характеристика |
| F | ti  ti,F | Правила, которые порождают параметры объявления функции |
| W | i  i,W  l  l,W | Правила, которые порождают параметры вызываемой функции. |
| M | +E  -E  \*E  /E  >E  <E  cE  nE  eE  oE  %E  &E  |E | Правила, которые порождают подвыражения. |
| L | i  l | Правила, которые порождают идентификатор или литерал для конструкций. |
| U | (L>L)  (L<L)  (LcL)  (LnL)  (LeL)  (LoL) | Правила, которые порождают условия для циклов и условных конструкций. |

**4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью – это семерка M = {Q, V, Z, δ, q0, z0, F}, в которой

* Q – множество возможных состояний;
* V – алфавит входных символов;
* Z – специальный алфавит магазинных символов;
* δ – функция переходов автомата Q × (V ∪{λ})× Z → P(Q × Z \*), где P(Q × Z \*) – множество подмножеств Q × Z \*;
* q0 ∈ Q – начальное состояние автомата;
* z0 ∈ Z – начальное состояние магазина (маркер дна);
* F ⊆ Q – множество конечных состояний.

Схема работы конечного автомата с магазинной памятью представлена на рисунке 4.2.

Рисунок 4.2 – Схема работы конечного автомата с магазинной памятью

Во время работы автомата на каждом шаге возможны 3 случая:

* функция δ(q, a, z) определена – осуществляется переход в новое состояние;
* функция δ(q, a, z) не определена, но определена δ(q, λ, z) – осуществляется переход в новое состояние (лента не продвигается);
* функция δ(q, a, z) и δ(q, λ, z) не определены – дальнейшая работа автомата не возможна (цепочка не разобрана).

Успешный результат разбора цепочки для контрольного примера приведен в приложении Г.

**4.4 Основные структуры данных**

Структуры синтаксического анализатора и их описание представлены в таблице 4.2.

Таблица 4.2 – Структуры синтаксического анализатора

|  |  |
| --- | --- |
| Структура | Характеристика |
| Greibach | Структура: представление грамматики. Все символы (алфавит) грамматики представляются в формате **GRBALPHABET** (short). Причем терминалы – положительные значения, нетерминалы – отрицательные значения.  Структура включает:  - множество правил: переменная rules (типа структура **Rule**);  - количество правил: переменная **size** (short);  - стартовый символ грамматики: **startN** (**GRBALPHABET**);  - служебный символ (дно стека и последняя лексема таблице лексем):  **stbottomT** (**GRBALPHABET**);  - два конструктора;  - методы **getRule**:  1) позволяет получить номер правила или -1 (к точке возврата) и правило (второй параметр типа Rule) по левому символу правила (первый параметр);  2) позволяет получить правило (возвращает к точке вызова параметр типа Rule) по его номеру |

Продолжение таблицы 4.2 – Структуры синтаксического анализатора

|  |  |
| --- | --- |
| Структура | Характеристика |
| Rule | Структура: представление одного правила, имеющего вид: A→xxx|yyy|….  Структура включает:  - нетерминал – левый символ правила: **nn** (**GRBALPHABET**);  - идентификатор ошибки, связанной с правилом: **iderror** (int) – код ошибки в подсистеме **Error**;  - количество цепочек в правой стороне правила: **size** (short);  - цепочки-правые стороны правила: **chains** (типа **Rule::Chain**);  - два конструктора;  - метод **getCRule**: позволяет получить правило в виде строки вида N-> цепочка (в символьном ASCII-виде, для отображения);  - метод **getNextChain**: позволяет найти следующую за номером (3й параметр **j** типа short) цепочку (параметр **pchain** типа **Rule::Chain**) и ее номер (к точке возврата типа short |
| Rule::Chain | Структура: представление цепочки – правой стороны правила.  Структура включает:  - размер цепочки: **size** (short) в символах;  - цепочка: **nt** (**GRBALPHABET**);  - два конструктора;  - метод **getCChain**: позволяет получить строку-цепочку в  символьном виде для отображения;  - методы **T** и **N**: преобразовывают ASCI-символы в **GRBALPHABET-**символы (терминалы и нетерминалы);  - методы **isT** и **isN**: проверяют является **GRBALPHABET-**символ терминалом или нетерминалом;  - метод **aplphabet\_to\_char**: преобразует заданный (параметр) **GRBALPHABET-**символ в ASCII-символ |
| MfstState | Структура: для сохранения состояния автомата;  сохранять состояние автомата необходимо для того, чтобы иметь возможность к этому состоянию вернуться и осуществить альтернативный вариант синтаксического разбора (в силу недетерминированности автомата).  Структура включает:  - текущую позицию на входной ленте автомата: **lenta\_position** (short);  - номер текущей цепочки, текущего правила: **nrulechain** (short);  - стек автомата с содержимым на момент сохранения **st**  (**MFSTSTACK**);  - два конструктора. |

Продолжение таблицы 4.2 – Структуры синтаксического анализатора

|  |  |
| --- | --- |
| Структура | Характеристика |
| Mfst | Структура: представление магазинного конечного автомата.  Структура включает:  - перечисление, содержащее возможные коды возврата метода step: **RC\_STEP**;  - массив структур для строк диагностики: **diagnosis (MstDiagnosis**, описание ниже**)**;  - входную ленту: **lenta (GRBALPHABET\*)**;  - текущая позиция на входной ленте: **lenta\_position** (short);  - номер текущего правила грамматики: **nrule** (short);  - номер текущей цепочки текущего правила грамматики: **nrulechain** (short);  - количество символов на ленте: **lenta\_size** (short);  - грамматика языка: **grebach (**GRB::Greibach);  - результат, предварительно выполненного лексического анализа (таблицы лексем и идентификаторов): **lex** (**LEX::LEX**);  - стек автомата: **st** (**MFSSTATACK**);  - стек для хранения состояний (структур **MfstState**) автомата: **storestate** (std::stack< **MfstState**>);  - два конструктора;  - функция **getCSt:** принимает один параметр – буфер; заполняет буфер содержимым стека для отображения, в конце 0x00; возвращает к точке вызова указатель на буфер;  - функция **getCLenta:** заполняет буфер (первый параметр) содержимым ленты с заданной позиции (второй параметр) заданное количество символов (третий параметр) в формате ASCII-строки для отображения, в конце строки 0x00; возвращает к точке вызова указатель на строки буфер;  - функция **getDiagnosis:** по заданному номеру (первый параметр) строки диагностики записывает строку в буфер (второй параметр) в формате ASCII-строки для отображения и возвращает указатель на буфер;  - функция **savestate:** сохраняет текущее состояние автомата в **storestate,** всегда возвращает **true**;  - функция **reststate:** восстанавливает последнее сохраненное состояние автомата из **storestate**, возвращает **true**, если восстановление выполнено (есть данные для восстановления);  - функция **push\_chain:** помещает реверс цепочки в стек автомата**,** всегда возвращает **true**;  - функция **step:** выполняет такт работы автомата, формирует диагностические сообщения, осуществляет отладочный вывод на консоль;  - функция **start:** запускает работу автомата, в цикле выполняет функцию **step**, осуществляет вывод диагностических сообщений;  - функция **savediagnosis:** сохраняет в массиве **diagnosis** строку диагностики; в массиве **diagnosis** сохраняются диагностические сообщения в порядке убывания позиции ленты (вызвавшей диагностику) и только в равным длине массива (макрос **MFST\_DIAGN\_NUMBER**). |
| MfstDiagnosis | Структура (внутренняя для **Mfst**): представление строки диагностики.  Структура включает:  - позиция входной ленты: **lenta\_position** (short);  - код возврата, сформированный функцией **step**;  - номер действующего на момент диагностики правила грамматики: **nrule** (short);  - номер текущей цепочки действующего на момент диагностики правила грамматики: **nrule\_chain**(short);  - два конструктора. |

Программный код основных структур данных на языке C++, описывающих контекстно-свободную грамматику, представлен в приложении Д.

**4.5 Описание алгоритма синтаксического разбора**

Синтаксический разбор выполняется по алгоритму, представленному ниже.

1. Первым делом инициализируется магазин, в который помещается стартовый символ.
2. Формируется входная лента по полученной ранее таблице лексем.
3. Встретившийся нетерминальный символ раскрывается в соответствии с правилами, и реверс полученной цепочки записывается в магазин.
4. Если терминалы на вершине стека и в ленте совпадают, то лента продвигается, а терминал удаляется с вершины стека. Иначе происходит возврат к предыдущему сохраненному состоянию и выбирается другое правило.
5. Если в магазине встретился нетерминал, то осуществляется переход в пункт 3.
6. Если достигнуто дно стека и при этом входная цепочка пуста, то значит синтаксический анализ выполнен успешно. Если нет, то генерируется исключение.

**4.6 Параметры синтаксического анализатора**

Синтаксический анализатор не имеет параметров, которые управляют режимом его работы, поэтому он всегда работает в одном режиме.

**4.7 Структура и перечень сообщений синтаксического анализатора**

В ситуации возникновения ошибки в синтаксическом анализаторе формируется ошибка в формате: код ошибки, текст сообщения, строка в исходном коде программы и позиция.

Все сообщения синтаксического анализатора с кодом ошибки представлены в таблице 4.3.

Таблица 4.3 – Сообщения синтаксического анализатора с кодом ошибки

|  |  |
| --- | --- |
| Номер ошибки | Текст сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибочный оператор |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах функции |
| 604 | Ошибка в параметрах вызываемой функции |
| 605 | Ошибка в подвыражении |
| 606 | Ошибка в логическом выражении |
| 607 | Ошибка в логическом подвыражении |

**4.8 Принцип обработки ошибок**

Синтаксический анализатор использует следующий алгоритм обработки ошибок:

* синтаксический анализатор пытается подобрать цепочку грамматики для текущей конструкции таблицы лексем;
* если подобрать цепочку невозможно, то генерируется соответствующая ошибка;
* все ошибки записываются в общую структуру ошибок;
* если была найдена ошибка после выполнения всего синтаксического анализа, то выводится соответствующее сообщение об ошибке, транслятор завершает свою работу.

**4.9 Контрольный пример**

Дерево разбора, полученное при синтаксическом анализе контрольного примера, представлено в приложении Г.

**5 Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор нужен для проверки смысловой структуры программы и ее семантической правильности, для обнаружения ошибок, которые связаны с типами данных, допустимыми операциями над переменными определенного типа данных, проверки совпадения возвращаемого значения функции с ее типом и т.д. Некоторые семантические проверки производятся на стадии лексического анализа. Входными параметрами семантического анализатора является таблица лексем и таблица идентификаторов. Выходным параметром является протокол работы.

Структура семантического анализатора представлена на рисунке 5.1.

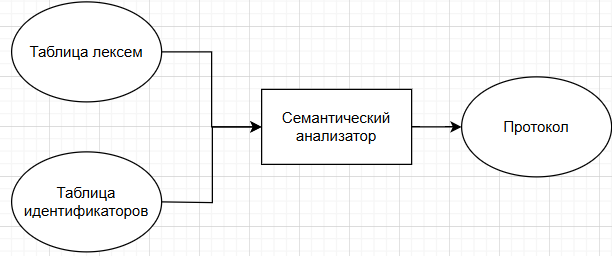


Рисунок 5.1 – Структура семантического анализатора

**5.2 Функции семантического анализатора**

В таблице 5.1 представлены семантические проверки с указанием фазы транслятора, на которой они происходят.

Таблица 5.1 – Семантические проверки языка PDI-2024

|  |  |
| --- | --- |
| Семантическая проверка | Фаза выполнения |
| Все функции должны быть объявлены до main | Семантический анализ |
| Функция main не объявлена | Лексический анализ |
| Ключевое слово не может быть идентификатором | Лексический анализ |
| Присваиваемое значение не соответствует типу переменной | Семантический анализ |
| Возвращаемое значение в функции не соответствует ее типу | Семантический анализ |
| Фактические и формальные параметры функции не соответствуют по типу или количеству | Семантический анализ |
| При присваивании, слева должна быть переменная | Семантический анализ |
| Недопустимо деление на ноль | Лексический анализ, выполнения программы |
| Ошибка в строковом литерале | Лексический анализ |
| Функция не может быть строкового типа | Лексический анализ |

В приложении Е представлен код функции void checkSemantic(LT::LexTable lextable, IT::IdTable idtable, map<string, vector<IT::IDDATATYPE>> funcs), реализующей проверку всех семантических правил.

**5.3 Структура и перечень сообщений семантического анализатора**

Перечень сообщений семантического анализатора с кодом ошибки представлен в таблице 5.2.

Таблица 5.2 – Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Номер ошибки | Текст сообщение |
| 102 | Все функции должны быть объявлены до main |
| 120 | Присваиваемое значение не соответствует типу переменной |
| 121 | Возвращаемое значение в функции не соответствует ее типу |
| 122 | Фактические и формальные параметры не соответствуют по типу или количеству |
| 123 | При присваивании, слева должна быть переменная или должен быть параметр |
| 124 | Недопустимо деление на ноль |
| 125 | Ошибка в строковом литерале |
| 126 | Функция не может быть строкового типа |

**5.4 Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибке и выводит его на консоль и в файл протокола .log. Транслятор завершает работу.

**5.5 Контрольный пример**

Примеры кодов с семантическими ошибками и сообщениями семантического анализатора представлены в приложении Ж.

**6 Вычисление выражений**

**6.1 Выражения, допускаемые языком**

Язык PDI-2024 допускает выражения, применимые к целочисленным и вещественным типам данных. Также допустим вызов функций, которые возвращают целочисленные и вещественные значения. В таблице 6.1 приведены операции и их приоритетность.

Таблица 6.1 – Операции и их приоритеты

|  |  |
| --- | --- |
| Операция | Приоритет |
| \* | 3 |
| / | 3 |
| % | 3 |
| + | 2 |
| - | 2 |
| & | 2 |
| | | 2 |
| , | 1 |
| ( | 0 |
| ) | 0 |

Пример выражения из контрольного примера:

(10 + a) / 2 – 5 и т.д.

**6.2 Польская запись и принципы ее построения**

Польская запись, или ПОЛИЗ, является альтернативным способом записи арифметических выражений, которые затем удобно вычислять с помощью стека. Обратная польская запись – это форма записи выражений, когда операнды идут перед знаками операций.

Алгоритм построения обратной польской записи:

* если встретился операнд, то он переносится в строку результата преобразования;
* если встретился вызов функции, то он помещается в стек;
* если встретилась операция, то, если стек не пуст, сравнивается приоритет текущей операции с приоритетом операции на вершине стека. Пока приоритет операции, которая находится на вершине стека выше или равен приоритету текущей операции, из стека извлекается операция и помещается в результирующую строку. Текущая операция помещается в стек. Иначе, если изначально стек был пуст, то текущая операция сразу помещается в стек;
* если встретилась открывающаяся скобка, то она помещается в стек;
* если встретилась закрывающаяся скобка, то извлекаются все операции в результирующую строку, пока не встретится открывающаяся скобка. После этого открывающаяся скобка также извлекается из стека;
* если выражение рассмотрено, но стек не пуст, то извлекаются все символы из стека в результирующую строку.

Пример преобразования выражения из контрольного примера:

(10 + a) / 2 – 5 – исходное выражение в инфиксной записи.

10а+2/5- – выражение в постфиксной записи.

**6.3 Программная реализация обработки выражений**

Фрагмент кода с программной реализацией обработки выражений и преобразования их в обратную польскую запись представлен в приложении И.

**6.4 Контрольный пример**

В приложении К представлена таблица лексем с выполненными преобразованиями выражений в ПОЛИЗ.

**7 Генерация кода**

**7.1 Структура генератора кода**

Трансляция языка PDI-2024 производится в язык ассемблера. Структура генератора кода представлена на рисунке 7.1.

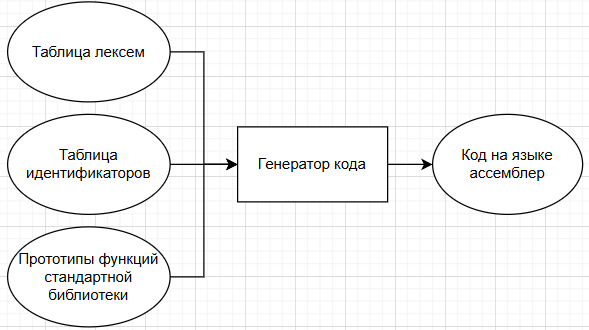


Рисунок 7.1 – Структура генератора кода

Генерация базируется на том, что каждый оператор представляет собой заранее заданный набор команд, который будет генерироваться при каждом использовании этого оператора. Функции на исходном языке программирования PDI-2024 транслируются в процедуры на ассемблере, возврат значений происходит через регистры и стек. При генерации условного оператора и оператора цикла используются метки и переходы по ним.

Генератору на вход поступает таблица лексем, таблица идентификаторов и прототипы функций стандартной библиотеки языка PDI-2024. На выходе получаем код на языке ассемблера.

**7.2 Представление типов данных в оперативной памяти**

Используется плоская модель памяти FLAT. Она представляет собой архитектурный подход к организации памяти в современных вычислительных системах. В этой модели память предоставляется программе как единое, непрерывное адресное пространство, что упрощает доступ к данным и коду. Данный сегмент делится на следующие области:

1. .STACK – сегмент стека;
2. .CONST – сегмент констант;
3. .DATA – сегмент данных;
4. .CODE – сегмент кода.

В таблице 7.1 приведено соответствие типов данных в языке программирования PDI-2024 и ассемблере, а также краткое пояснение.

Таблица 7.1 – Соответствие типов данных языка PDI-2024 и ассемблера

|  |  |  |
| --- | --- | --- |
| Тип данных в языке PDI-2024 | Тип данных в ассемблере | Пояснение |
| uint | DWORD | Нужен для хранения беззнакового целочисленного типа. |
| string | BYTE | Нужен для хранения строкового типа, представляет собой набор байтов, которые идут друг за другом. |
| bool | DWORD | Нужен для хранения логического значения. |
| double | real8 | Нужен для хранения вещественного типа. |

**7.3 Статическая библиотека**

Для языка PDI-2024 предусмотрена статическая библиотека, которая включает в себя функции, представленные в таблице 1.9.

Библиотека написана на языке С++. Ее подключение происходит на этапе компоновки. Прототипы функций указываются в прологе программы на ассемблере. На рисунке 7.2 приведен фрагмент кода на языке ассемблер, в котором демонстрируется подключение библиотеки и объявление прототипов функций.

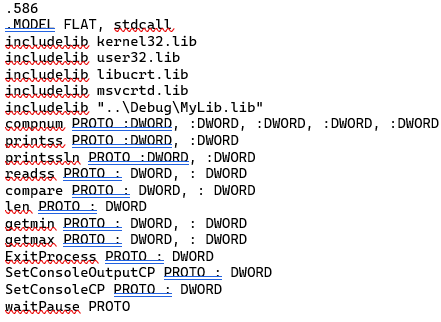


Рисунок 7.2 – Подключение библиотек и объявление прототипов функций на языке ассемблер

**7.4 Особенности алгоритма генерации кода**

Генерация кода реализуется с помощью вспомогательных макросов и основной функции void asmGenerator(LexA::LEX lex, wchar\_t outfile[]). Данная функция используется для генерации кода ассемблера, опираясь на таблицу лексем и идентификаторов.

Алгоритм генерации кода описан ниже.

1. Генерация пролога программы (указание модели процессора, модели памяти, соглашения о вызовах, подключаемых библиотек).
2. Генерация прототипов функций.
3. Генерация сегмента стека.
4. Генерация и заполнение сегментов констант и данных, используя таблицу идентификаторов.
5. Генерация сегмента кода.
6. Генерация пользовательских функций.
7. Генерация основной функции main.
8. Генерация кода, который соответствует окончанию процедуры main.

**7.5 Входные параметры, управляющие генерацией кода**

Генератор не имеет параметров для управления генерацией кода. Данная фаза трансляции происходит по одному принципу.

**7.6 Контрольный пример**

Результат генерации кода для контрольного примера представлен в приложении Л.

**8 Тестирование транслятора**

**8.1 Общие положения**

Тесты используются для проверки транслятора на работоспособность и для выявления недочетов и ошибок, а также последующего их исправления.

При возникновении ошибки транслятор завершает свою работу и выводит сообщение об ошибке с соответствующим ей кодом на консоль и в файл протокола .log.

**8.2 Результаты тестирования**

В таблице 8.1 представлены ошибки, которые будут генерироваться на разных стадиях транслятора с соответствующими фрагментами кода, в которых содержится ошибка.

Таблица 8.1 – Тестирование транслятора

|  |  |  |
| --- | --- | --- |
| Код | Сообщение об ошибке | Фаза транслятора |
| function double r(double a){  print a;  ret a;  };  main{  double t = r(10.5);  ret 0;  };  function uint g(uint a, uint b){  a = a + b;  ret a;  }; | Ошибка 102: Все функции должны быть объявлены до main | Семантический анализатор |
| function double r(double a){  print a;  ret a;  };  function uint g(uint a, uint b){  a = a + b;  ret a;  }; | Ошибка 103: Функция main не объявлена | Лексический анализатор |
| main{  double ret = 10.6;  }; | Ошибка 104: Ключевое слово не может быть идентификатором  Строка 2 позиция 11 | Лексический анализатор |

Продолжение таблицы 8.1 – Тестирование транслятора

|  |  |  |
| --- | --- | --- |
| Код | Сообщение об ошибке | Фаза транслятора |
| main{  double a = 10.5;  uint a = 2;  }; | Ошибка 105: Дублирование идентификатора  Строка 3 позиция 7 | Лексический анализатор |
| main{  a = 10 + 35;  }; | Ошибка 106: Идентификатор не определён  Строка 2 позиция 2 | Лексический анализатор |
| function double h(double b){  print b;  ret b;  };  function uint h(uint b){  print b;  ret b;  };  main{  ret 0;  }; | Ошибка 107: Функция уже определена  Строка 6 позиция 16 | Лексический анализатор |
| function uint k(uint a, uint b, uint r, uint t, uint u, uint i, uint p, uint y, uint o){  ret 1;  };  main{  uint a = k(1, 2, 3, 4, 5, 6, 7, 8, 9);  ret 0;  }; | Ошибка 108: Превышено максимально возможное число параметров в функции  Строка 1 позиция 79 | Лексический анализатор |
| main{  uint a = 10;  ret 0;  };  main{  uint b = 10;  ret 0;  }; | Ошибка 109: Повторное объявление main  Строка 6 позиция 5 | Лексический анализатор |

Продолжение таблицы 8.1 – Тестирование транслятора

|  |  |  |
| --- | --- | --- |
| Код | Сообщение об ошибке | Фаза транслятора |
| main{  double t = 'no';  ret 0;  }; | Ошибка 120: Присваиваемое значение не соответствует типу переменной  Строка 2 позиция 0 | Семантический анализатор |
| function double h(double a){  ret 'error';  }; | Ошибка 121: Возвращаемое значение в функции не соответствует ее типу  Строка 2 позиция 6 | Семантический анализатор |
| function double h(double t, double a){  t = t + a;  ret t;  };  main{  double n = h('error', 8.2);  ret 0;  }; | Ошибка 122: Фактические и формальные параметры не соответствуют по типу или количеству  Строка 7 позиция 6 | Семантический анализатор |
| main{  uint a = 10 / 0;  ret 0;  }; | Ошибка 124: Недопустимо деление на ноль  Строка 2 позиция 0 | Семантический анализатор |
| main{  string a = 'error;  ret 0;  }; | Ошибка 125: Ошибка в строковом литерале  Строка 2 позиция 17 | Семантический анализатор |
| function string a(string b){  ret b;  };  main{  string v = a('error');  ret 0;  }; | Ошибка 126: Функция не может быть строкового типа  Строка 1 позиция 0 | Семантический анализатор |

Продолжение таблицы 8.1 – Тестирование транслятора

|  |  |  |
| --- | --- | --- |
| Код | Сообщение об ошибке | Фаза транслятора |
| uint t = 10;  main{  uint f = 10;  ret 0;  }; | Ошибка 600: Неверная структура программы  Строка 1 позиция 0 | Синтаксический анализатор |
| function double h(double a){  ret a;  };  main{  double k = h(5.5,,);  }; | Ошибка 604: Ошибка в параметрах вызываемой функции  Строка 6 позиция 0 | Синтаксический анализатор |

**Заключение**

В ходе выполнения курсовой работы был разработан компилятор для языка PDI-2024. Были выполнены все минимальные требования, указанные в задании, а также были добавлены 2 дополнительных типа данных bool и double, арифметические и побитовые операции, оператор цикла, оператор ввода данных, дополнительные функции стандартной библиотеки языка.

Язык PDI-2024 включает в себя:

* 4 типа данных;
* оператор вывода данных с переводом строки;
* оператор вывода данных без перевода строки;
* оператор ввода данных;
* функции стандартной библиотеки языка;
* 5 арифметических операторов;
* 6 операторов сравнения;
* 2 побитовые операции;
* условный оператор if-otherwise;
* оператор цикла until.

В итоге была достигнута поставленная цель по разработке компилятора PDI-2024 со всеми требованиями. Данная работа позволила получить ценный опыт в сфере написания компиляторов, а также были изучены основы теории формальных грамматик и основы теории построения компиляторов.

**Список использованных источников**

1 Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. –М.: Вильямс, 2003. – 768с.

2 Ирвин, К. Язык ассемблера для процессоров Intel, 4-е издание.: Пер. с англ. – М.: Вильямс, 2005. – 912с.

3 Вирт, Н. Построение компиляторов / Пер. с англ. Борисов Е. И., Чернышов Л.Н. – М.: ДМК Пресс, 2010. – 192с.: ил.

4 Курс лекций по КПО / Наркевич А.С.

**Приложение А**

Контрольный пример

|  |
| --- |
| function double h(double a, double b){  a = a + b;  ret a;  };  main{  double k = 10.7;  string s = 'Проверка работы функции';  println s;  double t = h(100.5, k) - 12.7 - 7.25;  print 'h(100.5, k) - 12.7 - 7.25 = ';  println t;  uint a = 10;  uint b = (10 + a) / 2 - 5;  print 'b = ';  println b;  uint r;  read r;  if(r > b){  println 'Введенное число больше b';  };  otherwise{  println 'Введенное число меньше или равно b';  };  println 'Проверка работы цикла. Вывод цифр от 0 до 3';  uint j = 0;  until(j < 4){  println j;  j = j + 1;  };  println 'Конец';  ret 0;  }; |

**Приложение Б**

Графы конечных автоматов для распознавания ключевых слов

|  |  |
| --- | --- |
| #define FST\_UINT FST::FST \_uint(str,\  5,\  FST::NODE(1, FST::RELATION('u', 1)),\  FST::NODE(1, FST::RELATION('i', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('t', 4)),\  FST::NODE()\  );  #define FST\_STRING FST::FST \_string(str,\  7,\  FST::NODE(1, FST::RELATION('s', 1)),\  FST::NODE(1, FST::RELATION('t', 2)),\  FST::NODE(1, FST::RELATION('r', 3)),\  FST::NODE(1, FST::RELATION('i', 4)),\  FST::NODE(1, FST::RELATION('n', 5)),\  FST::NODE(1, FST::RELATION('g', 6)),\  FST::NODE()\  );  #define FST\_BOOL FST::FST \_bool(str,\  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()\  );  #define FST\_FUNC FST::FST \_function(str,\  9,\  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(1, FST::RELATION('t', 5)),\  FST::NODE(1, FST::RELATION('i', 6)),\  FST::NODE(1, FST::RELATION('o', 7)),\  FST::NODE(1, FST::RELATION('n', 8)),\  FST::NODE()\  );  #define FST\_RETURN FST::FST \_return(str,\  4,\  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE()\  );  #define FST\_MAIN FST::FST \_main(str,\  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()\  );  #define FST\_PRINT FST::FST \_print(str,\  6,\  FST::NODE(1, FST::RELATION('p', 1)),\  FST::NODE(1, FST::RELATION('r', 2)),\  FST::NODE(1, FST::RELATION('i', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE()\  );  #define FST\_PRINTLN FST::FST \_println(str,\  8,\  FST::NODE(1, FST::RELATION('p', 1)),\  FST::NODE(1, FST::RELATION('r', 2)),\  FST::NODE(1, FST::RELATION('i', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE(1, FST::RELATION('l', 6)),\  FST::NODE(1, FST::RELATION('n', 7)),\  FST::NODE()\  );  #define FST\_READ FST::FST \_read(str,\  5,\  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('e', 2)),\  FST::NODE(1, FST::RELATION('a', 3)),\  FST::NODE(1, FST::RELATION('d', 4)),\  FST::NODE()\  );  #define FST\_TRUE FST::FST \_true(str,\  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()\  );  #define FST\_FALSE FST::FST \_false(str,\  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()\  );  #define FST\_DOUBLE FST::FST \_double(str,\  7,\  FST::NODE(1, FST::RELATION('d', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('u', 3)),\  FST::NODE(1, FST::RELATION('b', 4)),\  FST::NODE(1, FST::RELATION('l', 5)),\  FST::NODE(1, FST::RELATION('e', 6)),\  FST::NODE()\  );  #define FST\_WHILE FST::FST \_while(str,\  6,\  FST::NODE(1, FST::RELATION('u', 1)),\  FST::NODE(1, FST::RELATION('n', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE(1, FST::RELATION('i', 4)),\  FST::NODE(1, FST::RELATION('l', 5)),\  FST::NODE()\  ); | #define FST\_IF FST::FST \_if(str,\  3,\  FST::NODE(1, FST::RELATION('i', 1)),\  FST::NODE(1, FST::RELATION('f', 2)),\  FST::NODE()\  );  #define FST\_ELSE FST::FST \_else(str,\  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()\  );  #define FST\_LITERAL FST::FST \_literal(str,\  3,\  FST::NODE(21,\  FST::RELATION('0', 0), FST::RELATION('1', 0), FST::RELATION('2', 0),\  FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0),\  FST::RELATION('6', 0), FST::RELATION('7', 0), FST::RELATION('8', 0),\  FST::RELATION('9', 0), FST::RELATION('.', 1), FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2),\  FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2),\  FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2),\  FST::RELATION('9', 2)),\  FST::NODE(20,\  FST::RELATION('0', 1), FST::RELATION('1', 1), FST::RELATION('2', 1),\  FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1),\  FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1),\  FST::RELATION('9', 1), FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2', 2),\  FST::RELATION('3', 2), FST::RELATION('4', 2), FST::RELATION('5', 2),\  FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2),\  FST::RELATION('9', 2)),\  FST::NODE()\  );  #define FST\_IDEN FST::FST \_iden(str,\  2,\  FST::NODE(104,\  FST::RELATION('a', 1), FST::RELATION('a', 0), FST::RELATION('b', 1), FST::RELATION('b', 0),\  FST::RELATION('c', 1), FST::RELATION('c', 0), FST::RELATION('d', 1), FST::RELATION('d', 0), FST::RELATION('e', 1), FST::RELATION('e', 0),\  FST::RELATION('f', 1), FST::RELATION('f', 0), FST::RELATION('g', 1), FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('h', 1), FST::RELATION('i', 0), FST::RELATION('i', 1),\  FST::RELATION('j', 0), FST::RELATION('j', 1), FST::RELATION('k', 0), FST::RELATION('k', 1), FST::RELATION('l', 0), FST::RELATION('l', 1),\  FST::RELATION('m', 0), FST::RELATION('m', 1), FST::RELATION('n', 0), FST::RELATION('n', 1), FST::RELATION('o', 0), FST::RELATION('o', 1),\  FST::RELATION('p', 0), FST::RELATION('p', 1), FST::RELATION('q', 0), FST::RELATION('q', 1), FST::RELATION('r', 0), FST::RELATION('r', 1),\  FST::RELATION('s', 0), FST::RELATION('s', 1), FST::RELATION('t', 0), FST::RELATION('t', 1), FST::RELATION('u', 0), FST::RELATION('u', 1),\  FST::RELATION('v', 0), FST::RELATION('v', 1), FST::RELATION('w', 0), FST::RELATION('w', 1), FST::RELATION('x', 0), FST::RELATION('x', 1),\  FST::RELATION('y', 0), FST::RELATION('y', 1), FST::RELATION('z', 0), FST::RELATION('z', 1),FST::RELATION('A', 1), FST::RELATION('A', 0), FST::RELATION('B', 1), FST::RELATION('B', 0),\  FST::RELATION('C', 1), FST::RELATION('C', 0), FST::RELATION('D', 1), FST::RELATION('D', 0), FST::RELATION('E', 1), FST::RELATION('E', 0),\  FST::RELATION('F', 1), FST::RELATION('F', 0), FST::RELATION('G', 1), FST::RELATION('G', 0), FST::RELATION('H', 0), FST::RELATION('H', 1), FST::RELATION('I', 0), FST::RELATION('I', 1),\  FST::RELATION('J', 0), FST::RELATION('J', 1), FST::RELATION('K', 0), FST::RELATION('K', 1), FST::RELATION('L', 0), FST::RELATION('L', 1),\  FST::RELATION('M', 0), FST::RELATION('M', 1), FST::RELATION('N', 0), FST::RELATION('N', 1), FST::RELATION('O', 0), FST::RELATION('O', 1),\  FST::RELATION('P', 0), FST::RELATION('P', 1), FST::RELATION('Q', 0), FST::RELATION('Q', 1), FST::RELATION('R', 0), FST::RELATION('R', 1),\  FST::RELATION('S', 0), FST::RELATION('S', 1), FST::RELATION('T', 0), FST::RELATION('T', 1), FST::RELATION('U', 0), FST::RELATION('U', 1),\  FST::RELATION('V', 0), FST::RELATION('V', 1), FST::RELATION('W', 0), FST::RELATION('W', 1), FST::RELATION('X', 0), FST::RELATION('X', 1),\  FST::RELATION('Y', 0), FST::RELATION('Y', 1), FST::RELATION('Z', 0), FST::RELATION('Z', 1)),\  FST::NODE()\  ); |

**Приложение В**

Таблица идентификаторов

|  |
| --- |
| id datatype idtype Line value Scope idTi  compare UINT F -1 global -1  len UINT F -1 global -1  getmin UINT F -1 global -1  getmax UINT F -1 global -1  h DOUBLE F 1 global 1  a DOUBLE P 1 h 1  b DOUBLE P 1 h 1  main UINT F 6 global 6  k DOUBLE V 7 main 7  L0 DOUBLE L 7 10.7 main 7  s STR V 8 main 8  L1 STR L 8 'Проверка работы функции' main 8  t DOUBLE V 10 main 10  L2 DOUBLE L 10 100.5 main 10  L3 DOUBLE L 10 12.7 main 10  L4 DOUBLE L 10 7.25 main 10  L5 STR L 11 'h(100.5,k)-12.7-7.25=' main 11  a UINT V 13 main 13  L6 UINT L 13 10 main 13  b UINT V 14 main 14  L7 UINT L 14 10 main 14  L8 UINT L 14 2 main 14  L9 UINT L 14 5 main 14  L10 STR L 15 'b=' main 15  r UINT V 17 main 17  L11 STR L 20 'Введенное число больше b' if0 20  L12 STR L 23 'Введенное число меньше или равно b' else0 23  L13 STR L 25 'Проверка работы цикла. Вывод цифр от 0 до 3' main 25  j UINT V 26 main 26  L14 UINT L 26 0 main 26  L15 UINT L 27 4 until0 27  L16 UINT L 29 1 until0 29  L17 STR L 31 'Конец' main 31  L18 UINT L 32 0 main 32 |

Таблица лексем

|  |
| --- |
| 1 fti(ti,ti){  2 i=i+i;  3 ri;  4 };  6 m{  7 ti=l;  8 ti=l;  9 xi;  10 ti=i(l,i)-l-l;  11 pl;  12 xi;  13 ti=l;  14 ti=(l+i)/l-l;  15 pl;  16 xi;  17 ti;  18 hi;  19 y(i>i){  20 xl;  21 };  22 d{  23 xl;  24 };  25 xl;  26 ti=l;  27 w(i<l){  28 xi;  29 i=i+l;  30 };  31 xl;  32 rl;  33 }; |

**Приложение Г**

Последовательность правил грамматики для контрольного примера

|  |
| --- |
| 0 : всего строк 33 , синтаксический анализ выполнен без ошибок  0 : S->fti(F){Nri;};S  4 : F->ti,F  7 : F->ti  11 : N->i=E;  13 : E->iM  14 : M->+E  15 : E->i  22 : S->m{Nrl;};  24 : N->ti=E;N  27 : E->l  29 : N->ti=E;N  32 : E->l  34 : N->xi;N  37 : N->ti=E;N  40 : E->i(W)M  42 : W->l,W  44 : W->i  46 : M->-E  47 : E->lM  48 : M->-E  49 : E->l  51 : N->pl;N  54 : N->xi;N  57 : N->ti=E;N  60 : E->l  62 : N->ti=E;N  65 : E->(E)M  66 : E->lM  67 : M->+E  68 : E->i  70 : M->/E  71 : E->lM  72 : M->-E  73 : E->l  75 : N->pl;N  78 : N->xi;N  81 : N->ti;N  84 : N->hi;N  94 : N->xl;  101 : N->xl; |

|  |
| --- |
| 109 : N->ti=E;N  112 : E->l  114 : N->wU{N};N  115 : U->(L<L)  116 : L->i  118 : L->l  121 : N->xi;N  124 : N->i=E;  126 : E->iM  127 : M->+E  128 : E->l  132 : N->xl; |

**Приложение Д**

Структуры данных, описывающие контекстно-свободную грамматику

|  |
| --- |
| struct Rule {  GRBALPHABET nn;  int iderror;  short size;  struct Chain {  short size;  GRBALPHABET\* nt;  Chain() { this->size = 0; this->nt = 0; }  Chain(short size, 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() { this->nn = 0x00; this->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() { this->size = 0; this->startN = 0; this->stbottomT = 0; this->rules = 0; };  Greibach(GRBALPHABET pstartN, GRBALPHABET pstbottomT, short psize, Rule r, ...);  short getRule(GRBALPHABET pnn, Rule& prule);  Rule getRule(short n);  };  Greibach getGreibach(); |

**Приложение Е**

Функция проверки семантических правил

|  |
| --- |
| #include "SemA.h"  #include <vector>  namespace SemA {  bool flagMain = false;  int curpos = 0;  int curline = 1;  void checkSemantic(LT::LexTable lextable, IT::IdTable idtable, map<string, vector<IT::IDDATATYPE>> funcs) {  for (int i = 0; i < lextable.size; i++) {  if (curline != lextable.table[i].sn) {  curline = lextable.table[i].sn;  curpos = 0;  }  curpos++;  if (lextable.table[i].lexema[0] == LEX\_MAIN) {  flagMain = true;  }  if (lextable.table[i].lexema[0] == LEX\_FUNCTION && flagMain) {  throw ERROR\_THROW\_IN(102, curline, curpos);  }  //if (lextable.table[i].lexema[0] == '=') {  // if (idtable.table[lextable.table[i - 1].idxTI].iddatatype != idtable.table[lextable.table[i + 1].idxTI].iddatatype) {  // throw ERROR\_THROW\_IN(120, curline, curpos);  // }  //}  if ((lextable.table[i].lexema[0] == LEX\_ID && idtable.table[lextable.table[i].idxTI].idtype == IT::F && lextable.table[i - 1].lexema[0] == TYPE) || lextable.table[i].lexema[0] == LEX\_MAIN) {  int j = i;  int localcurpos = 0;  while (lextable.table[j].lexema[0] != LEX\_RETURN) {  if (lextable.table[j - 1].sn != lextable.table[j].sn) {  localcurpos = 0;  }  j++;  localcurpos++;  }  if (idtable.table[lextable.table[j + 1].idxTI].iddatatype != idtable.table[lextable.table[i].idxTI].iddatatype) {  throw ERROR\_THROW\_IN(121, lextable.table[j + 1].sn, localcurpos);  }  } |
| if ((lextable.table[i].lexema[0] == LEX\_ID && idtable.table[lextable.table[i].idxTI].idtype == IT::F && lextable.table[i - 1].lexema[0] != TYPE) && lextable.table[i].lexema[0] != LEX\_MAIN) {  int indexparm = 0;  for (int k = 1; lextable.table[i + k].lexema[0] != LEX\_RIGHTTHESIS; k++) {  if (lextable.table[i + k].lexema[0] == LEX\_ID || lextable.table[i + k].lexema[0] == LEX\_LITERAL)  {  if (indexparm >= funcs[idtable.table[lextable.table[i].idxTI].id].size()) {  throw ERROR\_THROW\_IN(122, curline, curpos + k);  }  if (idtable.table[lextable.table[i + k].idxTI].iddatatype != funcs[idtable.table[lextable.table[i].idxTI].id][indexparm]) {  throw ERROR\_THROW\_IN(122, curline, curpos + k);  }  indexparm++;  }  }  }  }  // проверка на наличие main  if (!flagMain) {  throw ERROR\_THROW(103);  }  for (int i = 0; i < lextable.size; i++)  {  if (lextable.table[i].lexema[0] == LEX\_EQUAL)  {  // проверка на то, что мы присваиваем значение переменной или параметру функции  if (idtable.table[lextable.table[i - 1].idxTI].idtype != IT::IDTYPE::V && idtable.table[lextable.table[i - 1].idxTI].idtype != IT::IDTYPE::P)  throw ERROR\_THROW\_IN(123, lextable.table[i - 1].sn, 0);  IT::IDDATATYPE curtype = idtable.table[lextable.table[i - 1].idxTI].iddatatype;  int indexB = 0;  bool isFirst = true;  IT::IDDATATYPE type1 = IT::IDDATATYPE::UINT;  switch (curtype)  {  case IT::IDDATATYPE::BOOL: {  while (lextable.table[i + indexB].lexema[0] != LEX\_SEMICOLON)  {  if (lextable.table[i + indexB].lexema[0] == LEX\_PLUS || lextable.table[i + indexB].lexema[0] == LEX\_MINUS  || lextable.table[i + indexB].lexema[0] == LEX\_DIRSLASH || lextable.table[i + indexB].lexema[0] == LEX\_STAR  || lextable.table[i + indexB].lexema[0] == LEX\_OST) {  throw ERROR\_THROW\_IN(602, lextable.table[i + indexB].sn, 0);  }  if (lextable.table[i + indexB].lexema[0] == LEX\_ID || lextable.table[i + indexB].lexema[0] == LEX\_LITERAL)  {  if (isFirst)  {  type1 = idtable.table[lextable.table[i + indexB].idxTI].iddatatype;  isFirst = false;  if (lextable.table[i + indexB + 1].lexema[0] == LEX\_SEMICOLON && type1 != IT::BOOL) {  throw ERROR\_THROW\_IN(120, lextable.table[i + indexB].sn, 0);  }  }  if (idtable.table[lextable.table[i + indexB].idxTI].idtype == IT::IDTYPE::F)  {  if (idtable.table[lextable.table[i + indexB].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + indexB].sn, 0);  int countOfHesis = 0;  if (lextable.table[i + indexB + 1].lexema[0] != LEX\_LEFTTHESIS)  throw ERROR\_THROW\_IN(602, lextable.table[i + indexB].sn, 0);  do  {  indexB++;  if (lextable.table[i + indexB].lexema[0] == LEX\_LEFTTHESIS)  countOfHesis++;  if (lextable.table[i + indexB].lexema[0] == LEX\_RIGHTTHESIS)  countOfHesis--;  } while (countOfHesis != 0);  }  else if (idtable.table[lextable.table[i + indexB].idxTI].iddatatype != type1)  throw ERROR\_THROW\_IN(120, lextable.table[i + indexB].sn, 0);  }  indexB++;  continue;  }  break;  }  case IT::IDDATATYPE::STR: {  for (int j = 0; lextable.table[i + j].lexema[0] != LEX\_SEMICOLON; j++)  {  if (lextable.table[i + j].lexema[0] == LEX\_PLUS || lextable.table[i + j].lexema[0] == LEX\_MINUS  || lextable.table[i + j].lexema[0] == LEX\_DIRSLASH || lextable.table[i + j].lexema[0] == LEX\_STAR  || lextable.table[i + j].lexema[0] == LEX\_LESS  || lextable.table[i + j].lexema[0] == LEX\_MORE || lextable.table[i + j].lexema[0] == LEX\_ISEQUAL  || lextable.table[i + j].lexema[0] == LEX\_NOT\_EQUAL || lextable.table[i + j].lexema[0] == LEX\_LESS\_OR\_EQUAL  || lextable.table[i + j].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + j].lexema[0] == LEX\_OST  || lextable.table[i + j].lexema[0] == LEX\_OR || lextable.table[i + j].lexema[0] == LEX\_AND)  throw ERROR\_THROW\_IN(602, lextable.table[i + j].sn, 0);  if (lextable.table[i + j].lexema[0] == LEX\_ID || lextable.table[i + j].lexema[0] == LEX\_LITERAL)  {  if (idtable.table[lextable.table[i + j].idxTI].idtype == IT::IDTYPE::F)  {  if (idtable.table[lextable.table[i + j].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + j].sn, 0);  int countOfHesis = 0;  if (lextable.table[i + j + 1].lexema[0] != LEX\_LEFTTHESIS)  throw ERROR\_THROW\_IN(602, lextable.table[i + j].sn, 0);  do  {  j++;  if (lextable.table[i + j].lexema[0] == LEX\_LEFTTHESIS)  countOfHesis++;  if (lextable.table[i + j].lexema[0] == LEX\_RIGHTTHESIS)  countOfHesis--;  } while (countOfHesis != 0);  }  else if (idtable.table[lextable.table[i + j].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + j].sn, 0);  }  }  break;  }  case IT::IDDATATYPE::UINT: {  int index = 0;  while (lextable.table[i + index].lexema[0] != LEX\_SEMICOLON)  {  if ((lextable.table[i + index].lexema[0] == LEX\_DIRSLASH || lextable.table[i + index].lexema[0] == LEX\_OST) && idtable.table[lextable.table[i + index + 1].idxTI].value.vint == 0)  {  throw ERROR\_THROW\_IN(124, lextable.table[i + index].sn, 0);  }  if (lextable.table[i + index].lexema[0] == LEX\_LESS || lextable.table[i + index].lexema[0] == LEX\_MORE  || lextable.table[i + index].lexema[0] == LEX\_ISEQUAL || lextable.table[i + index].lexema[0] == LEX\_NOT\_EQUAL)  throw ERROR\_THROW\_IN(602, lextable.table[i + index].sn, 0);  if (lextable.table[i + index].lexema[0] == LEX\_ID || lextable.table[i + index].lexema[0] == LEX\_LITERAL)  {  if (idtable.table[lextable.table[i + index].idxTI].idtype == IT::IDTYPE::F)  {  if (idtable.table[lextable.table[i + index].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + index].sn, 0);  int countOfHesis = 0;  if (lextable.table[i + index + 1].lexema[0] != LEX\_LEFTTHESIS)  throw ERROR\_THROW\_IN(602, lextable.table[i + index].sn, 0);  do  {  index++;  if (lextable.table[i + index].lexema[0] == LEX\_LEFTTHESIS)  countOfHesis++;  if (lextable.table[i + index].lexema[0] == LEX\_RIGHTTHESIS)  countOfHesis--;  } while (countOfHesis != 0);  }  else  {  if (idtable.table[lextable.table[i + index].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + index].sn, 0);  }  }  index++;  continue;  }  break;  }  case IT::IDDATATYPE::DOUBLE: {  int indexD = 0;  while (lextable.table[i + indexD].lexema[0] != LEX\_SEMICOLON)  {  if ((lextable.table[i + indexD].lexema[0] == LEX\_DIRSLASH) && idtable.table[lextable.table[i + indexD + 1].idxTI].value.vint == 0.0)  {  throw ERROR\_THROW\_IN(124, lextable.table[i + indexD].sn, 0);  }  if (lextable.table[i + indexD].lexema[0] == LEX\_LESS || lextable.table[i + indexD].lexema[0] == LEX\_MORE  || lextable.table[i + indexD].lexema[0] == LEX\_ISEQUAL || lextable.table[i + indexD].lexema[0] == LEX\_NOT\_EQUAL || lextable.table[i + indexD].lexema[0] == LEX\_OST  || lextable.table[i + indexD].lexema[0] == LEX\_OR || lextable.table[i + indexD].lexema[0] == LEX\_AND)  throw ERROR\_THROW\_IN(602, lextable.table[i + indexD].sn, 0);  if (lextable.table[i + indexD].lexema[0] == LEX\_ID || lextable.table[i + indexD].lexema[0] == LEX\_LITERAL)  {  if (idtable.table[lextable.table[i + indexD].idxTI].idtype == IT::IDTYPE::F)  {  if (idtable.table[lextable.table[i + indexD].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + indexD].sn, 0);  int countOfHesis = 0;  if (lextable.table[i + indexD + 1].lexema[0] != LEX\_LEFTTHESIS)  throw ERROR\_THROW\_IN(602, lextable.table[i + indexD].sn, 0);  do  {  indexD++;  if (lextable.table[i + indexD].lexema[0] == LEX\_LEFTTHESIS)  countOfHesis++;  if (lextable.table[i + indexD].lexema[0] == LEX\_RIGHTTHESIS)  countOfHesis--;  } while (countOfHesis != 0);  }  else  {  if (idtable.table[lextable.table[i + indexD].idxTI].iddatatype != curtype)  throw ERROR\_THROW\_IN(120, lextable.table[i + indexD].sn, 0);  }  }  indexD++;  continue;  }  break;  }  }  }  if (lextable.table[i].lexema[0] == LEX\_IF)  {  if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::STR || idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::STR)  {  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::UINT && idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::UINT)  {  if (!(lextable.table[i + 3].lexema[0] == LEX\_MORE || lextable.table[i + 3].lexema[0] == LEX\_LESS || lextable.table[i + 3].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_LESS\_OR\_EQUAL ||  lextable.table[i + 3].lexema[0] == LEX\_ISEQUAL || lextable.table[i + 3].lexema[0] == LEX\_NOT\_EQUAL))  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::DOUBLE && idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::DOUBLE)  {  if (!(lextable.table[i + 3].lexema[0] == LEX\_MORE || lextable.table[i + 3].lexema[0] == LEX\_LESS || lextable.table[i + 3].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_LESS\_OR\_EQUAL ||  lextable.table[i + 3].lexema[0] == LEX\_ISEQUAL || lextable.table[i + 3].lexema[0] == LEX\_NOT\_EQUAL))  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::BOOL && idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::BOOL)  {  if (!(lextable.table[i + 3].lexema[0] == LEX\_MORE || lextable.table[i + 3].lexema[0] == LEX\_LESS || lextable.table[i + 3].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_LESS\_OR\_EQUAL ||  lextable.table[i + 3].lexema[0] == LEX\_ISEQUAL || lextable.table[i + 3].lexema[0] == LEX\_NOT\_EQUAL))  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else {  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  }  if (lextable.table[i].lexema[0] == LEX\_WHILE)  {  if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::STR || idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::STR)  {  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::UINT && idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::UINT)  {  if (!(lextable.table[i + 3].lexema[0] == LEX\_MORE || lextable.table[i + 3].lexema[0] == LEX\_LESS || lextable.table[i + 3].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_LESS\_OR\_EQUAL ||  lextable.table[i + 3].lexema[0] == LEX\_ISEQUAL || lextable.table[i + 3].lexema[0] == LEX\_NOT\_EQUAL))  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else if (idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::DOUBLE && idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::DOUBLE)  {  if (!(lextable.table[i + 3].lexema[0] == LEX\_MORE || lextable.table[i + 3].lexema[0] == LEX\_LESS || lextable.table[i + 3].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_LESS\_OR\_EQUAL ||  lextable.table[i + 3].lexema[0] == LEX\_ISEQUAL || lextable.table[i + 3].lexema[0] == LEX\_NOT\_EQUAL))  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else if(idtable.table[lextable.table[i + 2].idxTI].iddatatype == IT::IDDATATYPE::BOOL && idtable.table[lextable.table[i + 4].idxTI].iddatatype == IT::IDDATATYPE::BOOL)  {  if (!(lextable.table[i + 3].lexema[0] == LEX\_MORE || lextable.table[i + 3].lexema[0] == LEX\_LESS || lextable.table[i + 3].lexema[0] == LEX\_MORE\_OR\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_LESS\_OR\_EQUAL ||  lextable.table[i + 3].lexema[0] == LEX\_ISEQUAL || lextable.table[i + 3].lexema[0] == LEX\_NOT\_EQUAL || lextable.table[i + 3].lexema[0] == LEX\_OR || lextable.table[i + 3].lexema[0] == LEX\_AND))  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  else {  throw ERROR\_THROW\_IN(606, lextable.table[i].sn, 0);  }  }  }  }} |

**Приложение Ж**

Примеры кодов с семантическими ошибками

|  |  |
| --- | --- |
| main{  double t = 'no';  ret 0;  }; | Ошибка 120: Присваиваемое значение не соответствует типу переменной  Строка 2 позиция 0 |
| function double h(double a){  ret 'error';  }; | Ошибка 121: Возвращаемое значение в функции не соответствует ее типу  Строка 2 позиция 6 |
| function double h(double t, double a){  t = t + a;  ret t;  };  main{  double n = h('error', 8.2);  ret 0;  }; | Ошибка 122: Фактические и формальные параметры не соответствуют по типу или количеству  Строка 7 позиция 6 |

**Приложение И**

Фрагмент кода с программной реализацией обработки выражений и преобразования их в обратную польскую запись

|  |
| --- |
| void Rpn(LT::LexTable lextable, IT::IdTable idtable, int curPos)  {  queue<LT::Entry> q;  stack<LT::Entry> s;  LT::Entry bottom = { '$', LT\_TI\_NULLIDX, LT\_TI\_NULLIDX };  int curpos = curPos;  int kolLex = 0;  for (curPos; lextable.table[curPos].lexema[0] != LEX\_SEMICOLON; curPos++, kolLex++)  {  switch (lextable.table[curPos].lexema[0])  {  case LEX\_ID:  {  if (idtable.table[lextable.table[curPos].idxTI].idtype == IT::IDTYPE::F)  {  s.push(lextable.table[curPos]);  continue;  }  q.push(lextable.table[curPos]);  continue;  }  case LEX\_LITERAL:  {  q.push(lextable.table[curPos]);  continue;  }  case LEX\_LEFTTHESIS:  {  s.push(lextable.table[curPos]);  continue;  }  case LEX\_RIGHTTHESIS:  {  if (!s.empty())  {  while (s.top().lexema[0] != LEX\_LEFTTHESIS)  {  q.push(s.top());  s.pop();  }  s.pop();  }  continue;  } |

**Приложение К**

Таблица лексем с выполненными преобразованиями выражений в ПОЛИЗ

|  |
| --- |
| 1 fti(ti,ti){  2 i=ii+;  3 ri;  4 };  6 m{  7 ti=l;  8 ti=l;  9 xi;  10 ti=liil-l-  -1 $$$  10 ;  11 pl;  12 xi;  13 ti=l;  14 ti=li+l/l-  -1 $$  14 ;  15 pl;  16 xi;  17 ti;  18 hi;  19 y(i>i){  20 xl;  21 };  22 d{  23 xl;  24 };  25 xl;  26 ti=l;  27 w(i<l){  28 xi;  29 i=il+;  30 };  31 xl;  32 rl;  33 }; |

**Приложение Л**

Результат генерации кода для контрольного примера

|  |
| --- |
| .586  .MODEL FLAT, stdcall  includelib kernel32.lib  includelib user32.lib  includelib libucrt.lib  includelib msvcrtd.lib  includelib "..\Debug\MyLib.lib"  compnum PROTO :DWORD, :DWORD, :DWORD, :DWORD, :DWORD  printss PROTO :DWORD, :DWORD  printssln PROTO :DWORD, :DWORD  readss PROTO : DWORD, : DWORD  compare PROTO : DWORD, : DWORD  len PROTO : DWORD  getmin PROTO : DWORD, : DWORD  getmax PROTO : DWORD, : DWORD  ExitProcess PROTO : DWORD  SetConsoleOutputCP PROTO : DWORD  SetConsoleCP PROTO : DWORD  waitPause PROTO  .STACK 4096  .CONST  null\_division BYTE 'ERROR: DIVISION BY ZERO', 0  error\_read BYTE 'ERROR: READING ERROR', 0  TRUE BYTE 'true', 0  FALSE BYTE 'false', 0  L0 real8 10.7  L1 BYTE 'Проверка работы функции', 0  L2 real8 100.5  L3 real8 12.7  L4 real8 7.25  L5 BYTE 'h(100.5,k)-12.7-7.25=', 0  L6 DWORD 10  L7 DWORD 10  L8 DWORD 2  L9 DWORD 5  L10 BYTE 'b=', 0  L11 BYTE 'Введенное число больше b', 0  L12 BYTE 'Введенное число меньше или равно b', 0  L13 BYTE 'Проверка работы цикла. Вывод цифр от 0 до 3', 0  L14 DWORD 0 |
| L15 DWORD 4  L16 DWORD 1  L17 BYTE 'Конец', 0  L18 DWORD 0  .DATA  k real8 0.0  s BYTE 128 dup(0)  t real8 0.0  a DWORD 0  b DWORD 0  r DWORD 0  j DWORD 0  .CODE  push 1251d  call SetConsoleOutputCP  push 1251d  call SetConsoleCP  theend:  call waitPause  push 0  call ExitProcess  SOMETHINGWRONG:  push OFFSET null\_division  push 4  call printss  jmp THEENDER  ERRORREAD:  push OFFSET error\_read  push 4  call printss  THEENDER:  call waitPause  push -1  call ExitProcess  h PROC b\_ : real8, a\_ : real8  fld b\_  fld a\_  fadd  fstp a\_  fld a\_  jmp local0  local0:    ret  h ENDP  main PROC  fld L0  fstp k  lea esi, [L1]  lea edi, [s]  mov ecx, lengthof L1  rep movsb  INVOKE printssln, 4, addr s  INVOKE h, L2, k  fld L3  fsub  fld L4  fsub  fstp t  INVOKE printss, 4, addr L5  INVOKE printssln, 2, addr t  push L6  pop a  push a  push L7  pop eax  pop ebx  add eax, ebx  push eax  push L8  pop eax  pop ebx  cmp ebx, 0  je SOMETHINGWRONG  cdq  idiv ebx  push eax  push L9  pop eax  pop ebx  sub eax, ebx  push eax  pop b  INVOKE printss, 4, addr L10  INVOKE printssln, 1, addr b  INVOKE readss, 1, addr r  cmp eax, -1  jz THEENDER  mov al, '>'  push eax  push OFFSET b  push 1  push OFFSET r  push 1  call compnum  cmp eax, -1  je THEENDER  cmp eax, 1  je ifi0  cmp eax, 0  je elsei0  ifi0:  INVOKE printssln, 4, addr L11  jmp ifEnd0  elsei0:  INVOKE printssln, 4, addr L12  ifEnd0:  INVOKE printssln, 4, addr L13  push L14  pop j  while0:  mov al, '<'  push eax  push OFFSET L15  push 1  push OFFSET j  push 1  call compnum  cmp eax, -1  je THEENDER  cmp eax, 0  je notwhile0  INVOKE printssln, 1, addr j  push L16  push j  pop eax  pop ebx  add eax, ebx  push eax  pop j  jmp while0  notwhile0:  INVOKE printssln, 4, addr L17  push L18  jmp theend  main ENDP  end main |