МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий Кафедра Программной инженерии Специальность 1-40 01 01 Программное обеспечение информационных технологий Специализация Программирование интернет-приложений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора SMI-2023»

Выполнил студент Сосновец Мария Игоревна

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

(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 ст.пр. Наркевич Аделина Сергеевна

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



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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 5](#_Toc153109721)

[1. Спецификация языка программирования 6](#_Toc153109722)

[1.1 Характеристика языка программирования 6](#_Toc153109723)

[1.2 Определение алфавита языка программирования 6](#_Toc153109724)

[1.3 Применяемые сепараторы 7](#_Toc153109725)

[1.4 Применяемые кодировки 7](#_Toc153109726)

[1.5 Типы данных 7](#_Toc153109727)

[1.6 Преобразование типов данных 8](#_Toc153109728)

[1.7 Идентификаторы 9](#_Toc153109729)

[1.8 Литералы 10](#_Toc153109730)

[1.9 Объявление данных 10](#_Toc153109731)

[1.10 Инициализация данных 11](#_Toc153109732)

[1.11 Инструкции языка 11](#_Toc153109733)

[1.12 Операции языка 12](#_Toc153109734)

[1.13 Выражения и их вычисление 12](#_Toc153109735)

[1.14 Конструкции языка 13](#_Toc153109736)

[1.15 Области видимости идентификаторов 13](#_Toc153109737)

[1.16 Семантические проверки 13](#_Toc153109738)

[1.17 Распределение оперативной памяти на этапе выполнения 14](#_Toc153109739)

[1.18 Стандартная библиотека и её состав 14](#_Toc153109740)

[1.19 Ввод и вывод данных 15](#_Toc153109741)

[1.20 Точка входа 15](#_Toc153109742)

[1.21 Препроцессор 15](#_Toc153109743)

[1.22 Соглашения о вызовах 15](#_Toc153109744)

[1.23 Объектный код 15](#_Toc153109745)

[1.24 Классификация сообщений транслятора 15](#_Toc153109746)

[1.25 Контрольный пример 16](#_Toc153109747)

[2. Структура транслятора 17](#_Toc153109748)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 17](#_Toc153109749)

[2.2 Перечень входных параметров транслятора 18](#_Toc153109750)

[2.3 Протоколы, формируемые транслятором 18](#_Toc153109751)

[3. Разработка лексического анализатора 20](#_Toc153109752)

[3.1 Структура лексического анализатора 20](#_Toc153109753)

[3.2 Входные и выходные данные лексического анализатора 20](#_Toc153109754)

[3.3 Параметры лексического анализатора 20](#_Toc153109755)

[3.4 Алгоритм лексического анализа 21](#_Toc153109756)

[3.5 Контроль входных символов 21](#_Toc153109757)

[3.6 Удаление избыточных символов 22](#_Toc153109758)

[3.7 Перечень ключевых слов 22](#_Toc153109759)

[3.8 Основные структуры данных 23](#_Toc153109760)

[3.9 Структура и перечень сообщений лексического анализатора 24](#_Toc153109761)

[3.10 Принцип обработки ошибок 25](#_Toc153109762)

[3.11 Контрольный пример 25](#_Toc153109763)

[4. Разработка синтаксического анализатора 26](#_Toc153109764)

[4.1 Структура синтаксического анализатора 26](#_Toc153109765)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 26](#_Toc153109766)

[4.3 Построение конечного магазинного автомата 29](#_Toc153109767)

[4.4 Основные структуры данных 30](#_Toc153109768)

[4.5 Описание алгоритма синтаксического разбора 30](#_Toc153109769)

[4.6 Параметры синтаксического анализатора 30](#_Toc153109770)

[4.7. Структура и перечень сообщений синтаксического анализатора 30](#_Toc153109771)

[4.8. Принцип обработки ошибок 32](#_Toc153109772)

[4.9. Контрольный пример 32](#_Toc153109773)

[5. Разработка семантического анализатора 33](#_Toc153109774)

[5.1 Структура семантического анализатора 33](#_Toc153109775)

[5.2 Функции семантического анализатора 33](#_Toc153109776)

[5.3 Структура и перечень сообщений семантического анализатора 34](#_Toc153109777)

[5.4 Принцип обработки ошибок 35](#_Toc153109778)

[5.5 Контрольный пример 36](#_Toc153109779)

[6. Вычисление выражений 37](#_Toc153109780)

[6.1 Выражения, допускаемые языком 37](#_Toc153109781)

[6.2 Польская запись и принцип её построения 38](#_Toc153109783)

[6.3 Программная реализация обработки выражений 39](#_Toc153109784)

[6.4 Контрольный пример 39](#_Toc153109785)

[7. Генерация кода 40](#_Toc153109786)

[7.1 Структура генератора кода 40](#_Toc153109787)

[7.2 Представление типов данных в оперативной памяти 40](#_Toc153109788)

[7.3 Статическая библиотека 40](#_Toc153109789)

[7.4 Особенности алгоритма генерации кода 41](#_Toc153109790)

[7.5 Параметры, управляющие генерацией кода 43](#_Toc153109791)

[7.6 Контрольный пример 43](#_Toc153109792)

[8. Тестирование транслятора 44](#_Toc153109793)

[8.1 Общие положения 44](#_Toc153109794)

[8.2 Результаты тестирования 44](#_Toc153109795)

[Заключение 48](#_Toc153109796)

[Список использованных литературных источников 49](#_Toc153109797)

[Приложение А 50](#_Toc153109798)

[Приложение Б 52](#_Toc153109799)

[Приложение В 55](#_Toc153109800)

[Приложение Д 59](#_Toc153109801)

[Приложение Е 61](#_Toc153109802)

[Приложение Ж 63](#_Toc153109803)

# **Введение**

Целью курсового проектирования является разработка транслятора для языка программирования SMI-2023, уникального языка программирования, который требует эффективной интерпретации и перевода в низкоуровневый язык ассемблера. Основным языком программирования, выбранным для реализации транслятора, является С++, известный своей эффективностью и высокой производительностью.

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования.

Транслятор состоит из следующих частей:

– лексический анализатор;

– семантический анализатор;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были поставлены следующие задачи:

– разработка дизайна языка;

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка и преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

В данном курсовом проекте будет определена спецификация языка программирования, представлена структура транслятора, показана разработка лексического анализатора, порождающего таблицы лексем и идентификаторов, рассказано о синтаксическом анализаторе, который выполняет синтаксический разбор текста с распечаткой протокола разбора и дерева разбора на основе таблицы лексем, описан семантический анализатор, показана его работа, решены вопросы преобразования выражений, допускаемых языком и приведена часть протокола для контрольного примера, отображающая результаты преобразования выражений в польский формат, представлена генерация кода, где из промежуточного представления порождается код на целевом языке, а так же описано тестирование кода.

**1. Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык программирования SMI-2023 является высокоуровневым, процедурным, не объектно-ориентированным, компилируемым. Имеет нестрогую статическую типизацию.

Высокоуровневый язык программирования — средство записи компьютерных программ, обеспечивающее высокую скорость и удобство работы. Его отличительной чертой является абстракция. Высокоуровневый язык программирования обеспечивает возможность введения смысловых конструкций, способных коротко описать форматы данных и операции с ними в тех случаях, когда описания на [низкоуровневом языке](https://gb.ru/blog/yazyki-programmirovania-nizkogo-urovnya/?from=blog_stati_ankor).

Процедурный язык программирования — язык высокого уровня, в котором используется метод разбиения программ на отдельные связанные между собой модули — подпрограммы (процедуры и функции).

Объектно-ориентированный язык программирования — язык, построенный на принципах объектно-ориентированного программирования. В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Компилируемый язык программирования — язык программирования, исходный код которого преобразуется компилятором в исходный код на другом языке программирования.

Нестрогая статическая типизация — это концепция, применяемая в некоторых языках программирования, которая сочетает в себе особенности статической типизации и динамической типизации.

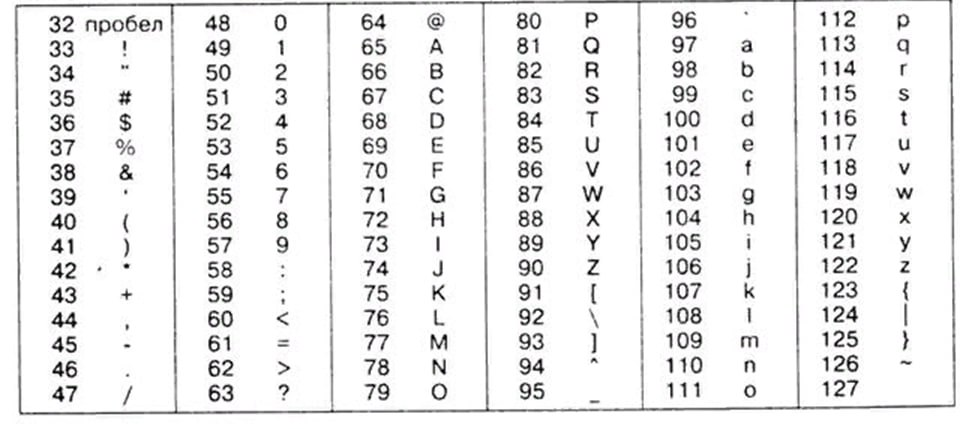
Статическая типизация — приём, широко используемый в языках программирования, при котором переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже (переменная или параметр будут принимать, а функция — возвращать значения только этого типа).

**1.2 Определение алфавита языка программирования**

Алфавит языка программирования – набор символов, которые могут использоваться при написании исходного кода.

Алфавит языка SMI-2023 основывается на таблице ASCII, представленной в таблице 1.1.

Таблица 1.1 – Таблица кодировок ASCII



Используемые в алфавите символы: [a … z], [A … Z], [0 … 9], [А … Я], [а … я] спецсимволы: () , ; :`' , а также символы пробела, табуляции и перевода строки.

**1.3 Применяемые сепараторы**

Сепараторы — это символы, используемые для разделения элементов в исходном коде и указания конца выражений или инструкций.

Сепараторы, которые использовались в языке программирования SMI-2023 представлены в таблице 1.2.

Таблица 1.2 – Сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение и область применения |
| ‘пробел’ | Разделитель лексем языка. Допускается везде кроме названий идентификаторов и ключевых слов |
| { … } | Блок функции или условной конструкции/цикла |
| ( … ) | Установка приоритета арифметических операций |
| **,** | Разделитель параметров функций |
| > < == ^=  >= <= | Логические операции (операции сравнения: больше, меньше, равно, не равно, больше или равно, меньше или равно), используемые в условии цикла/условной конструкции, а также в выражениях. |
| + - \* / % | Арифметические операции (сложение, вычитание, умножение, деление, остаток от деления). Используются в выражениях. |
| ; | Разделитель программных конструкций |
| **=** | Оператор присваивания |

**1.4 Применяемые кодировки**

Для написания программ на языке SMI-2023 используется кодировка Windows – 1251, представленная на рис.1.1.

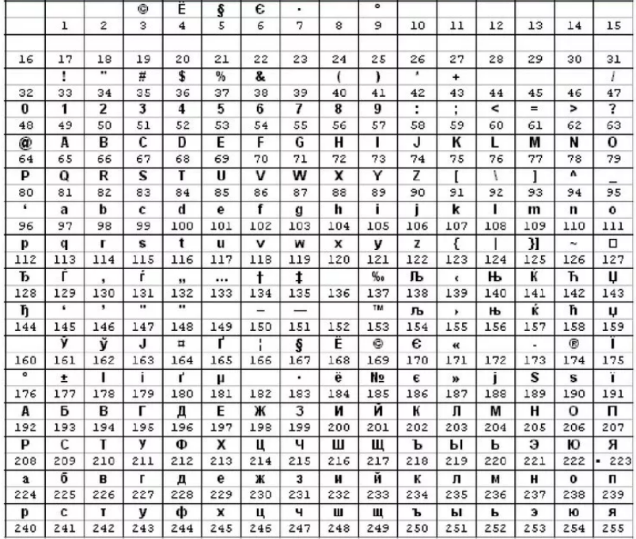


Рисунок 1.1 – Кодировка Windows-1251

Windows-1251 — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий [Microsoft Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows) до 10-й версии.

**1.5 Типы данных**

В языке SMI-2023 есть три типа данных: целочисленный, строковый и логический. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.3.

Таблица 1.3 – Типы данных SMI-2023 и их характеристика

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный тип данных number | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. В памяти занимает 2 байта. Максимальное значение: 32767.  Минимальное значение: -32768.  Инициализация по умолчанию: значение 0.  В качестве операторов условия цикла поддерживаются следующие операторы:  + (бинарный) – оператор сложения;  **-** (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  **/** (бинарный) – оператор целочисленного деления;  % (бинарный) – оператор остатка от деления;  =(бинарный) – оператор присваивания;  > (бинарный) – оператор «больше»;  < (бинарный) – оператор «меньше»;  ==(бинарный) – оператор «равно»;  ^=(бинарный) – оператор «не равно»;  >=(бинарный) – оператор «больше или равно»;  <=(бинарный) – оператор «меньше или равно». |
| Логический тип данных bool | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных. В памяти занимает 1 байт.  Значения: true (истина), false (ложь).  Инициализация по умолчанию: false. |
| Строковый тип данных line | Фундаментальный тип данных. Предусмотрен для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 256.  Инициализация по умолчанию: строка нулевой длины “”. Поддерживаемые операции:  **=** (бинарный) – оператор присваивания. |

Данная таблица демонстрирует, какие значения можно использовать для каждого типа данных, а также возможные операции, применимые к этим типам.

**1.6 Преобразование типов данных**

В языке SMI-2023 предусмотрены неявные преобразования между логическими (bool) и целочисленными (number) типами. Значение логического типа true преобразуется к 1, значение false – к 0. К логическому значению true преобразуются все значения типа number кроме 0, он преобразуется к false.

Логический тип преобразуется к целочисленному в следующих случаях:

* при присваивании результата логического выражения переменной типа number;
* при присваивании значения логического идентификатора или литерала переменной типа number;
* при вычислении арифметических выражений;
* при передаче логического типа в качестве целочисленного аргумента функции.

Целочисленный тип преобразуется к логическому в следующих случаях:

* при присваивании результата арифметического выражения переменной типа bool;
* при присваивании значения целочисленного идентификатора или литерала переменной типа bool;
* при передаче целочисленного типа в качестве логического аргумента функции;
* при передаче целочисленного идентификатора или литерала в качестве условия для оператора цикла или условного оператора.

Преобразование строкового типа line в какой-либо другой фундаментальный тип не производится.

**1.7 Идентификаторы**

Идентификатор — это последовательность символов, используемая для обозначения одного из следующих элементов:

* имени объекта или переменной;
* имени класса, структуры или объединения;
* имени перечисленного типа;
* члена класса, структуры, объединения или перечисления;
* функции или функции члена класса;
* имени определения типа (typedef);
* имени макроса;
* параметра макроса;

Идентификаторы в языке SMI-2023 используются для именования функций, процедур, параметров функций и переменных. Идентификаторы, объявленные внутри блока функции или процедуры получают префикс, идентичный имени функции/процедуры для разрешения области видимости.

Предусмотрены несколько правил составления идентификатора:

– состоит из символов латинского алфавита любого регистра и цифр;

– могут начинаться с символа нижнего подчеркивания «\_»;

– максимальная длина идентификатора равна 10. При превышении длины идентификатора она будет урезаться;

– не должны совпадать с ключевыми словами языка, а также с именами команд ассемблера (за исключением идентификаторов переменных и параметров функций/процедур);

Формальное описание идентификатора приведено в таблице 1.4.

Таблица 1.4 – Формальное описание идентификатора (БНФ)

|  |
| --- |
| *<идентификатор> ::= <буква> | <идентификатор><буква>*  *<буква>::= "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"|"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"* |

Примеры правильных идентификаторов: numA, count, factorial, getName, result, a, myVariable.

Примеры неправильных идентификаторов: num1, number, getFourthNumberFibonacci, числоА, number one, result+, 123variable, my variable, @value, class.

Примеры неправильных идентификаторов для функций или процедур: byte, offset, mov, push.

**1.8 Литералы**

Литералы — это постоянные значения, которые присваиваются постоянным переменным. Литералы представляют собой фиксированные значения, которые нельзя изменить. Литералы содержат память, но у них нет ссылок в качестве переменных.

В языке SMI-2023 существует три типа литералов: целочисленные, логические и строковые. Они используются для инициализации переменных, внутри условия оператора цикла или условного оператора, а также могут выступать в качестве аргумента функции. Все литералы являются rvalue. Краткое описание литералов представлено в таблице 1.5.

Таблица 1.5 – Описание литералов

|  |  |
| --- | --- |
| Тип | Описание |
| Логические литералы | Значения true, false. В памяти представляется как 0 и 1. |
| Строковые литералы | Символы, заключенные в двойные кавычки (“”), только rvalue. Максимальное число символов в литерале – 255. Использование двойных кавычек внутри строкового литерала не допускается. |
| Целочисленные  литералы | Последовательность цифр 0…9 с предшествующим знаком минус или без него.  Диапазоны значений: от -32768 до 32767. |

Примеры правильных литералов: 26, -19, 0, “Hello, С++!”, “Привет, Друг!”, true, false.

Примеры неправильных литералов: -9999999, 10754949585, TRUE, “”, “Фабрика “Коммунарка””.

**1.9 Объявление данных**

При объявлении переменной указывается ключевое слово "def", за которым следует указание имени переменной и её типа данных. Можно также присвоить начальное значение переменной в момент объявления. Для создания функции используется ключевое слово "function", после которого следует указание имени функции и её возвращаемого типа. При объявлении процедуры используется ключевое слово "procedure", за которым сразу следует имя процедуры без указания возвращаемого типа.

**1.10 Инициализация данных**

Инициализация по умолчанию при объявлении: переменные типа bool – инициализируются false, переменные типа number инициализируются нулём, переменные типа line – пустой строкой. Например:

def <тип данных> <идентификатор>.

Присваивание значения идентификатора, литерала, результата вычисления выражения или функции при объявлении. Например:

def <тип данных> <идентификатор> = <значение>|<выражение>;

<значение>::= <литерал>| <идентификатор>|<вызов функции>.

**1.11 Инструкции языка**

Инструкции языка программирования SMI-2023 и примеры применения инструкций представлены в таблице 1.6.

Таблица 1.6 – Инструкции языка программирования SMI-2023

|  |  |  |
| --- | --- | --- |
| Инструкция | Запись на языке SMI-2023 | Пример |
| Объявление переменной | def <тип данных> <идентификатор>; | def number value; |
| Объявление переменной с явной инициализацией | def <тип данных> <идентификатор> = <значение>|<выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> | def bool a = true;  def number value = 7 \* 6; |
| Присваивание | <идентификатор> = <значение>|  <выражение>;  <значение>::= <литерал>| <идентификатор>|<вызов функции> | value = 23;  value = result;  value = factorial[4]; |
| Вызов внешней функции или процедуры | <идентификатор> [ <идентификатор>|  <литерал>, …]; | factorial[d];  factorial[4];  max[2, 19]; |
| Возврат из  подпрограммы | give <идентификатор> / <литерал>; | give result;  give 2; |
| Вывод данных | out [<идентификатор> / <литерал>]; | out[“Bye”]; |
| Вывод данных с переходом на новую строку | outln [<идентификатор> / <литерал>]; | outln[result] |
| Условный оператор | when [<условие>]  { … }  otherwise  { … } | when [i > 0]  {  outln[“Positive”];  }  otherwise  {  outln[“Negative”];} |

Инструкции в языке программирования играют важную роль в организации и управлении выполнением программы. Они позволяют указывать, какие действия должны быть выполнены компьютером в определенном порядке.

**1.12 Операции языка**

В языке программирования SMI-2023 предусмотрены логические и арифметические операции, представленные в таблице 1.7.

Таблица 1.7 – Операции языка SMI-2023

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

Максимальным значением приоритетности является “3”, минимальным “0” соответственно. Логические операции приоритета не имеют. Для смены приоритета арифметических операций можно использовать оператор круглые скобки ().

**1.13 Выражения и их вычисление**

Выражения в языке SMI-2023 составляются согласно следующим правилам:

– использование логических и арифметических операторов в одном выражении не допустимо;

– использование двух и более идущих подряд операторов (за исключением оператора смены приоритета) запрещено;  
– использование двух и более логических операторов в одном выражении запрещено;

– вычисление сложных выражений (как минимум с одним оператором) внутри оператора возврата, в аргументах функции или процедуры, внутри условия цикла или условного оператора (за исключением логических операций) не производится.

**1.14 Конструкции языка**

Ключевые программные конструкции языка программирования SMI-2023 представлены в таблице 1.8.

Таблица 1.8 – Программные конструкции языка SMI-2023

|  |  |
| --- | --- |
| Главная функция | program  { …} |
| Процедура | procedure <идентификатор> [<тип данных> <идентификатор>, …]  {  …  } |
| Функция | function <тип данных> <идентификатор> [<тип данных> <идентификатор>, …]  {  …  give <выражение>  } |

Конструкции языка программирования играет важную роль в разработке программного кода.

Функция в языке программирования SMI-2023 может принимать максимум 3 параметра.

**1.15 Области видимости идентификаторов**

В языке SMI-2023 переменные обязаны находится внутри программного блока функций или процедур. Переменные, объявленные в одной функции, недоступны в другой. Внутри разных областей видимости разрешено объявление переменных с одинаковыми именами. Все переменные и параметры внутри области видимости получают префикс, который отображается в таблице идентификаторов. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости также не предусмотрено.

**1.16 Семантические проверки**

В языке программирования SMI-2023 выполняются следующие семантические проверки:

* наличие точки входа в программу;
* единственность точки входа;
* переопределение идентификаторов;
* использование идентификаторов без их объявления;
* проверка соответствия типа функции и возвращаемого параметра;
* правильность передаваемых в функцию параметров: количество, типы;
* правильность строковых выражений;
* превышение размера строковых и целочисленных литералов;
* деление на ноль в арифметических операциях;
* проверка на вызов функции в логических выражениях;
* корректность использования операторов в выражениях.

**1.17 Распределение оперативной памяти на этапе выполнения**

Используемые в языке ассемблера исходный код включает в себя два различных сегмента памяти. В первом из них, который является сегментом констант, размещаются все литералы. Во втором сегменте, который представляет собой область данных, хранятся переменные и параметры функций. Контроль над локальной областью видимости в коде осуществляется при помощи префиксов идентификаторов. Этот подход обеспечивает локальность переменных на уровне исходного кода, несмотря на то, что в языке ассемблера все переменные имеют глобальную область видимости. код

**1.18 Стандартная библиотека и её состав**

Функции стандартной библиотеки языка SMI-2023 реализованы на языке C++. Описание функций приведено в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| concat | line | line a, line b | Функция производит конкатенацию строк a и b, возвращает строку. |
| linelen | number | line a | Функция вычисляет длину строки a. |
| random | number | number min, number max | Функция возвращает случайно сгенерированное число в диапазоне [min, max]. |
| sqrt | number | number a | Функция возвращает результат вычисление квадратного корня из числа a. |
| OutLine | - | line a | Функция выводит в консоль строку a. |
| OutNumber | - | number a | Функция выводит в консоль число a. |
| OutBool | - | bool a | Функция выводит в консоль булево значение a. |
| OutLineLn | - | line a | Функция выводит в консоль строку a с переносом на новую строку. |
| OutNumberLn | - | number a | Функция выводит в консоль число a с переносом на новую строку. |
| OutBoolLn | - | bool a | Функция выводит в консоль булево значение a с переносом на новую строку. |

Одной из основных преимуществ стандартной библиотеки является то, что она обеспечивает возможность повторного использования кода.

**1.19 Ввод и вывод данных**

Вывод данных осуществляется с помощью функцийout иoutln. Допускается использование функций с литералами и идентификаторами.

В зависимости от типа параметра определяется функция: OutLine, OutBool, OutNumber, OutLineLn, OutBoolLn, OutNumberLn которые входят в состав стандартной библиотеки и описаны в таблице 1.9.

Функции ввода данных в языке SMI-2023 не предусмотрены.

**1.20 Точка входа**

Точкой входа является функция с именем program.

**1.21 Препроцессор**

Препроцессор в языке программирования SMI-2023 не предусмотрен.

**1.22 Соглашения о вызовах**

В языке SMI-2023 вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

**1.23 Объектный код**

Исходный код на языке SMI-2023 транслируется в язык ассемблера, а затем в объектный код.

**1.24 Классификация сообщений транслятора**

При обнаружении ошибки в программном коде, написанном на языке SMI-2023, и её выявлении транслятором, соответствующее сообщение отображается в текущем файле протокола. Спецификация классификации сообщений представлена в таблице 1.10.

Таблица 1.10 – ­­­­­­­­Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-110 | Системные ошибки, ошибки параметров |
| 200-299 | Ошибки лексического анализа |
| 300-399 | Ошибки семантического анализа |
| 600-699 | Ошибки синтаксического анализа |

Окончание таблицы 1.10

|  |  |
| --- | --- |
| 400 – 499,  700 – 999 | Зарезервированные коды ошибок |

**1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка SMI-2023: функции, процедуры, фундаментальные типы, основные инструкции и операции, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

**2. Структура транслятора**

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

В языке программирования SMI-2023 исходный текст программы преобразуется в язык ассемблера при помощи транслятора, включающего лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. В процессе лексического анализа создаются таблицы лексем и идентификаторов, которые передаются входными данными для синтаксического и семантического анализаторов. После успешного прохождения проверки исходного кода всеми указанными анализаторами, таблицы лексем и идентификаторов идут на вход генератора кода. Структура транслятора языка SMI-2023 представлена на рисунке 2.1.



Рисунок 2.1 – Структура транслятора SMI-2023

Первым этапом работы транслятора является проведение лексического анализа, который выполняется специальным инструментом - лексическим анализатором. На вход лексическому анализатору поступает исходный текст программы, преобразованный в последовательность символов. Анализатор разбирает текст, разделяя его на отдельные слова, заменяя их внутренним представлением - лексемами. Эти лексемы используются для создания промежуточного представления исходного текста программы. Все полученные лексемы регистрируются в таблице лексем. Также формируется таблица идентификаторов, где хранится информация о каждом идентификаторе, включая его имя, тип данных, значение и первое вхождение в таблицу лексем. Обе таблицы - лексем и идентификаторов - служат входными данными для следующего этапа транслятора, а именно синтаксического анализа. Назначение лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Вторым этапом процесса трансляции является синтаксический анализ. Синтаксический анализатор выполняет проверку исходного кода на соответствие правилам грамматики языка программирования SMI-2023.

Назначение синтаксического анализатора:

− сопоставить последовательности лексем правилам грамматики;

− построить дерево разбора;

− выявить синтаксические ошибки при их наличии и выдать сообщение об ошибке.

Третьим этапом процесса трансляции является выполнение семантического анализа. На вход семантического анализатора поступают таблицы лексем и идентификаторов. Анализатор осуществляет проверку на наличие ошибок, которые не могут быть выявлены с использованием как регулярной, так и контекстно-свободной грамматики. Семантические проверки, осуществляемые анализатором, описаны в пункте 1.16 раздела 1.

Четвертым этапом процесса трансляции является генерация ассемблерного кода на основе полученных данных на предыдущих этапах трансляции (таблица лексем, таблица идентификаторов). Более подробно структура генератора описана в главе 7.

**2.2 Перечень входных параметров транслятора**

Входные параметры транслятора представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка SMI-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на SMI-2023 | Не предусмотрено |
| -log:<имя\_файла> | Файл для записи полного протокола работы транслятора | <имя\_файла>.log |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_файла>.out.asm |
| -poliz | Ключ для вывода на консоль промежуточного представления кода после преобразования в польскую инверсную запись | По умолчанию отсутствует |
| -lt | Ключ для вывода таблицы лексем на консоль | По умолчанию отсутствует |
| -id | Ключ для вывода таблицы идентификаторов на консоль | По умолчанию отсутствует |

Входные параметры транслятора – это специфические данные или настройки, которые передаются транслятору для обработки и преобразования исходного кода.

**2.3 Протоколы, формируемые транслятором**

В ходе трансляции формируются протоколы работы лексического, синтаксического и семантического анализаторов. Перечень протоколов, формируемых транслятором языка программирования их описание представленные в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка SMI-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, заданный параметром -log | Содержит информацию о времени выполнения приложения; входных параметрах в приложение; код на языке SMI-2023 с сепараторами и без избыточных пробелов, табуляций и переходов на новую строку; таблицу идентификаторов; таблицу лексем; промежуточное представление кода; трассировку синтаксического анализа; дерево разбора, время выполнения разбора; промежуточное представление кода после приведения его к польской нотации. |
| Выходной файл, заданный параметром -out: | Содержит сгенерированный код на языке Ассемблера. |

Таким образом, протоколы работы лексического, синтаксического и семантического анализаторов играют важную роль в трансляции программного кода, обеспечивая его правильность и оптимизацию перед его фактическим выполнением. Эти протоколы являются фундаментальными инструментами.

# **3.** **Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор — это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы. Результатом работы лексического анализатора является перечень всех найденных в тексте исходной программы лексем. Этот перечень лексем представляется в виде таблицы, называемой таблицей лексем. Таблица лексем фактически содержит весь текст исходной программы, обработанный лексическим анализатором. Структура лексического анализатора представлена на рисунке 3.1.

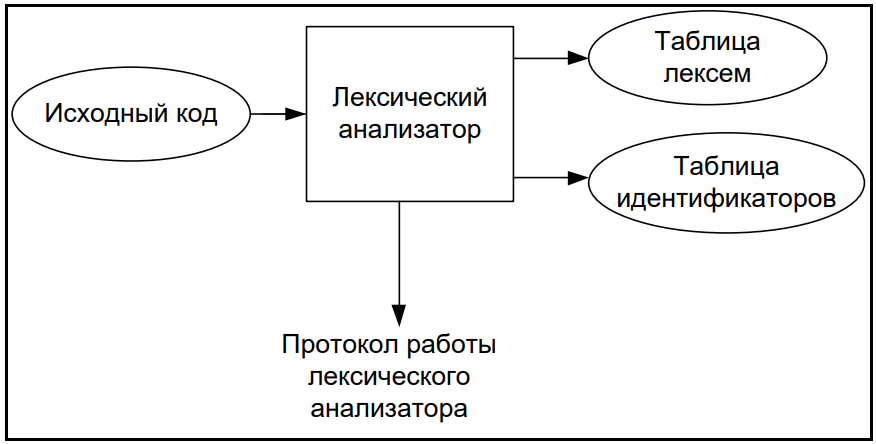


Рисунок 3.1 – Структура лексического анализатора

Данный рисунок содержит обработанный лексическим анализатором текст исходной программы.

## **3.2 Входные и выходные данные лексического анализатора**

Входными параметрами для лексического анализатора является исходный текст программы, написанный на языке SMI-2023, а также файл протокола, в который записываются входные данные (таблица лексем и таблица идентификаторов).

Лексический анализатор является важной частью процесса компиляции, предназначенной для преобразования исходного кода программы в набор лексем. На этапе лексического анализа осуществляется разбор исходного текста программы с целью выделения лексем - минимальных значимых единиц, таких как ключевые слова, идентификаторы, числа, знаки операций и другие.

## **3.3 Параметры лексического анализатора**

Параметры, управляющие работой лексического анализатора представлены в таблице 3.1.

Таблица 3.1 – Параметры, управляющие работой лексического анализатора языка SMI-2023

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Прототип функции | Входные данные | Выходные данные | Назначение | Принцип применения |
| GetWord | unsigned char source[] - массив символов, представляющий исходный код программы  int size – размер массива символов. | возвращает массив структур, который содержит информацию о словах, выделенных из исходного кода программы. | Выполняет анализ символов в массиве и разбивает его на слова с использованием разделителей, замена символов перевода строки на другие символы, сдвиг индекса на следующую позицию после найденного слова. | Вызывает функцию, передавая ей массив символов их размер, проход по исходному коду, выделяя слова и заполняя структуру. |
| Analyze | In in - содержит информацию о тексте исходного кода для анализа. | возвращает две таблицы: lextable (лексическая таблица) и idtable (таблица идентификаторов). | Проанализировать предоставленный входной исходный код и создать лексическую и таблицу идентификаторов. | Принимает на вход исходный код программы, выполнение лексического анализа, распознание лексемы, построение таблицы лексем и идентификаторов. |

В данном контексте, таблица 3.3 является центральным элементом, представляющим параметры, которые играют роль в управлении лексическим анализатором. Лексический анализатор – это важный компонент программного обеспечения, который отвечает за разбор входного текста на отдельные лексемы или токены.

## **3.4 Алгоритм лексического анализа**

Лексический анализ основан на функционировании конечных автоматов, которые анализируют регулярные выражения. Регулярные выражения представляют собой метод или формальный способ определения регулярных языков, которые состоят из констант и операторов, которые определяют множество строк и набор операций, выполняемых над ними. Любое регулярное выражение может быть представлена в виде графа.

Алгоритм лексического анализа:

− из входного потока символов программы на исходном языке удаляются лишние пробелы и добавляется сепаратор для вычисления номера строки для каждой лексемы;

− формируется массив из слов языка;

− для каждого слова выполняется функция распознавания лексемы;

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

− при неуспешном распознавании выдается сообщение об ошибке;

− формируется протокол работы.

## **3.5 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2.

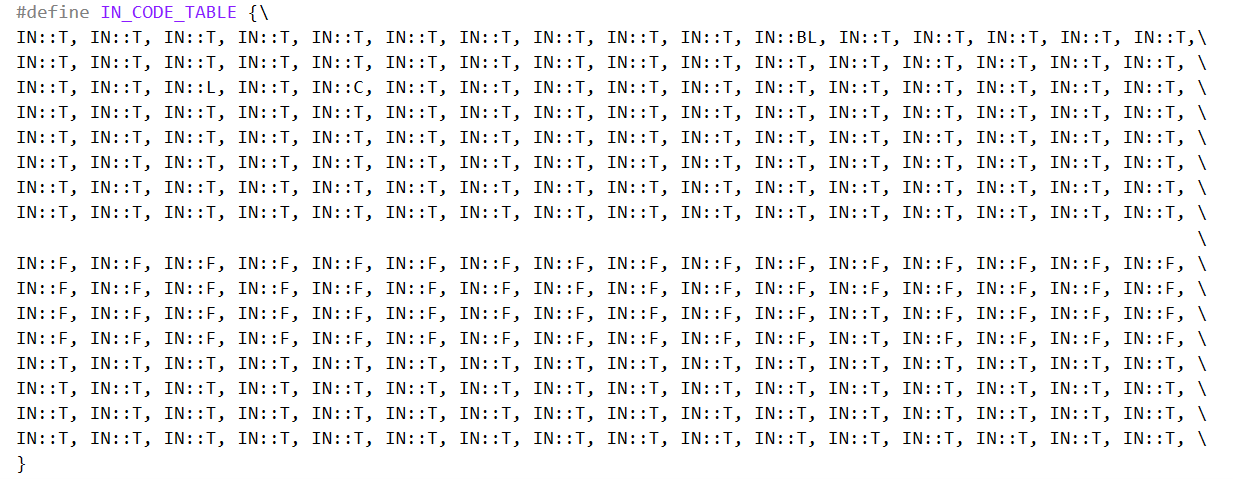


Рисунок 3.2 – Таблица контроля входных символов

Категории входных символов представлены в таблице 3.2.

Таблица 3.2 – Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Символы | Значение в таблице входных символов |
| T | Разрешенный |
| F | Запрещенный |
| BL | Перевод строки |
| C | Признак начала комментария |
| L | Строковый литерал |

В данной таблице перечислены и классифицированы все символы, которые могут быть использованы в качестве входных данных.

## **3.6 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

− посимвольное считывание исходного кода, занесенного в структуру In;

− встреча пробела или знака табуляции вне пределов строкового литерала или комментария является встречей символа-сепаратора;

− в отличие от других символов-сепараторов табуляции и пробелы игнорируются, т.е. не записываются в таблицу лексем.

## **Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие ключевых слов и лексем приведено в таблице 3.3.

Таблица 3.3 – Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | def | d |
| number, line, bool | t |
| program | p |
| function | f |
| procedure | s |
| give | r |
| out | o |
| when | w |
| otherwise | ! |
| Иное | Идентификатор | i |
| Литерал | l |

Окончание таблицы 3.3

|  |  |  |
| --- | --- | --- |
| Функции стандартной библиотеки | concat | + |
| linelen | % |
| sqrt | q |
| outln | B |
|  | random | z |
| Сепараторы | ; | ; |
|  | , | , |
|  | { | { |
|  | } | } |
|  | ( | ( |
|  | ) | ) |
|  | [ | [ |
|  | ] | ] |
| Операторы | Арифметические (+, -, \*, /, %) | v |
|  | Логические (== != > < >= <=) | g |
|  | Присваивание (=) | = |

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата представлена в листинге 3.1.

|  |
| --- |
| struct RELATION  {  unsigned char symbol;  short nnode;  RELATION(char c = 0x00, short ns = NULL);  };  struct NODE  {  short n\_relation;  RELATION\* relations;  NODE();  NODE(short n, RELATION rel, ...);  };  struct FST  {  unsigned char\* string;  short position;  short nstates;  NODE\* nodes;  short\* rstates;  FST(unsigned char\* s, short ns, NODE n, ...);  }; |

Листинг 3.1 – Структура конечного автомата

Пример графа перехода конченого автомата представлен на рисунке 3.3.

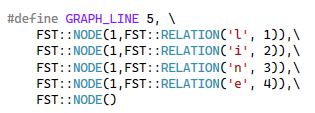


Рисунок 3.3 – Граф конечного автомата для ключевого слова line

В приложении Б представлены графы конечных автоматов для других слов языка.

## **3.8 Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Код на языке C++ со структурой таблицы лексем представлен в листинге 3.2.

|  |
| --- |
| struct Entry  {  unsigned char lexema;  int line;  int idxTI;  int priority;  OPER operation;  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.2 – Структура таблицы лексем

Код на языке C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

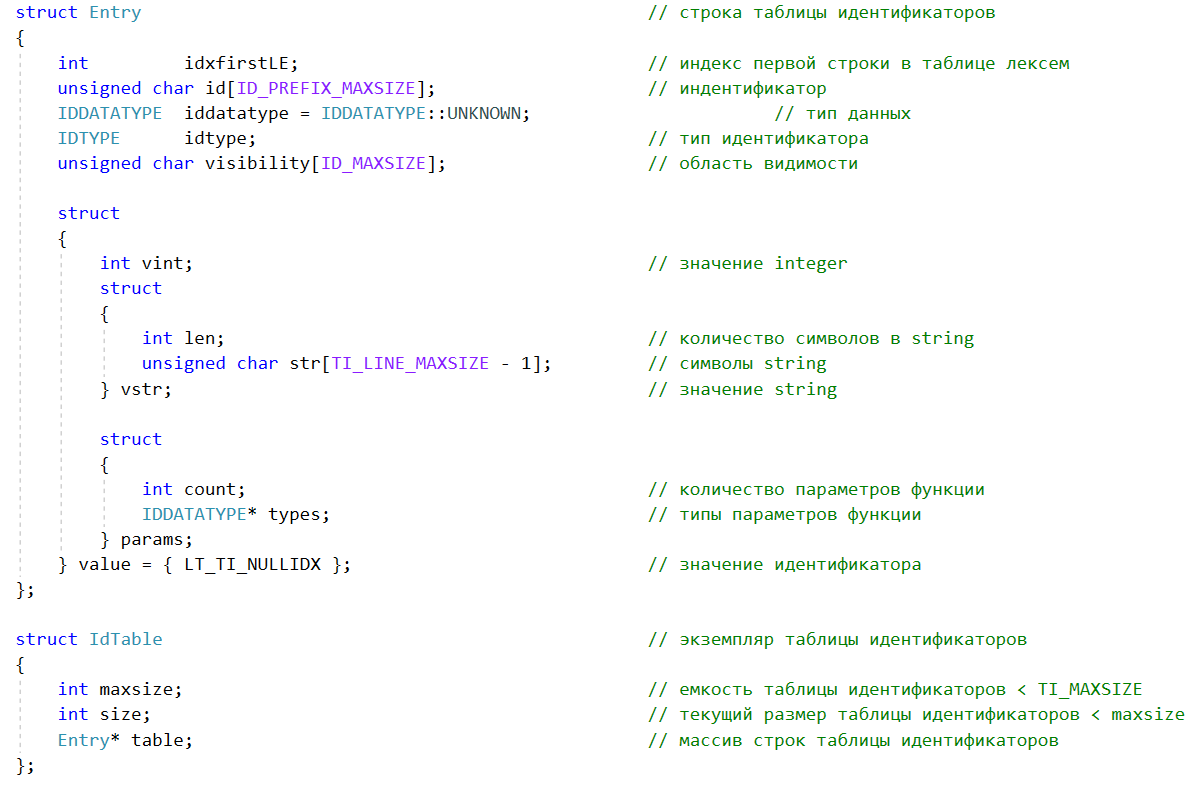


Рисунок 3.4 – Структура таблицы идентификаторов

Таблица идентификаторов состоит из набора полей данных (записей), каждое из которых может соответствовать одному элементу исходной программы.

## **3.9 Структура и перечень сообщений лексического анализатора**

Перечень сообщений лексического анализатора, в котором макросом является макрос ERROR\_ENTRY, а категория ошибки LEXICAL представлен в таблице 3.4.

Таблица 3.4 – Перечень сообщений лексического анализатора языка SMI-2023

|  |  |
| --- | --- |
| Код ошибки | Сообщение об ошибке |
| 200 | Недопустимый символ в исходном файле (-in) |
| 201 | Превышен размер таблицы лексем |
| 202 | Переполнение таблицы лексем |
| 203 | Превышен размер таблицы идентификаторов |
| 204 | Переполнение таблицы идентификаторов |
| 205 | Лексема не распознана |

Окончание таблицы 3.4

|  |  |
| --- | --- |
| 206 | Длина идентификатора не должна превышать 10 символов |

Макрос ERROR\_ENTRY служит для удобства и эффективности работы с лексическим анализатором. Категория ошибок LEXICAL включает в себя различные семантические и синтаксические ошибки, которые могут возникать при обработке текста.

## **3.10 Принцип обработки ошибок**

Лексический анализатор использует таблицу сообщений для обработки ошибок. Эта таблица имеет структуру, которая включает в себя несколько полей: номер сообщения, номер строки и позицию в исходном коде, где данное сообщение было вызвано, а также информацию о самой ошибке. Если возникает сообщение, лексический анализатор приостанавливает свою работу, выводит сообщение в файл протокола и завершает выполнение.

## **3.11 Контрольный пример**

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

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор – это часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные на этапе лексического анализа. Выходной информацией является дерево разбора.

Структура синтаксического анализатора представлена на рисунке 4.1.

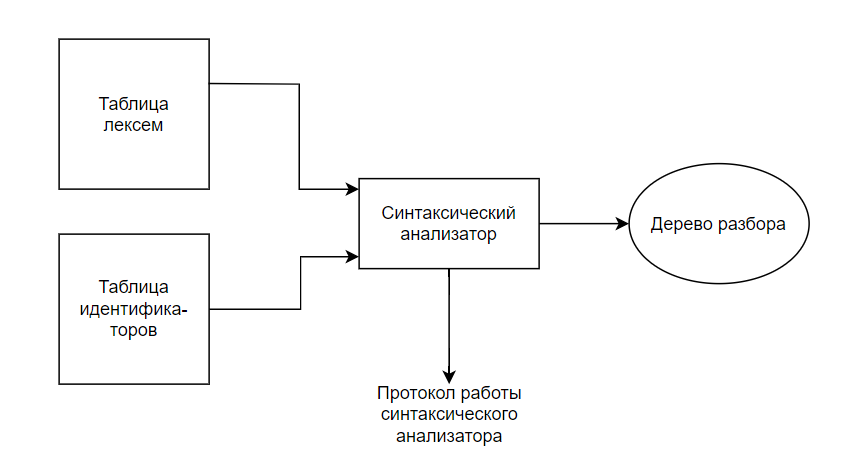


Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка SMI-2023 используется контекстно-свободная грамматика , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Перечень правил, составляющих грамматику языка программирования SMI-2023 представлен в таблице 4.1.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | ftiFBS  siFUS  p{N}  ftiFB  siFU | Проверка правильности структуры программы |
| F | [P]  [] | Проверка наличия списка параметров функции |
| P | ti  ti,P | Проверка на ошибку в параметрах функции при ее объявлении |
| B | {Nr[I];}  {r[I];} | Проверка наличия тела функции |
| I | l  i | Проверка на недопустимое выражение (ожидается только литерал или идентификатор) |
| N | dti;N  dti;  dti=R;N  dti=E;N  dti=E;  i=R;N  i=E;N  i=E;  w[R]{X}N  w[R]{X}  w[R]{X}!{X}N  w[R]{X}!{X}  %K;N  %K;  +K;N  +K;  qK;N  qK;  zK;N  zK;  o[I];N  o[I];  b[I];N  oK;  bK;  iK;N  iK; | Проверка на неверную конструкцию в теле функции |
| U | {N} | Проверка на ошибку в теле процедуры |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| R | i  l  igi  igl  lgi  lgl | Проверка на ошибку в условном выражении |
| K | [W]  [] | Проверка на ошибку в вызове функции |
| E | i  iM  l  lM  (E)  (E)M  iK  iKM  %K  %KM  +K  +KM  qK  qKM  zK  zKM | Проверка на ошибку в арифметическом выражении |
| W | i  l  i,W  l,W | Проверка на ошибку в параметрах вызываемой функции |
| M | vE  vEM | Проверка арифметических действий |
| X | dti;N  dti;  dti=E;N  dti=E;  dti=R;N  dti=R;  i=R;N  i=R;  i=E;N  i=E;  %K;N  %K;  +K;N  +K;  qK;N | Проверка на неверную конструкцию в теле цикла или условного выражения |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  | qK;  zK;N  zK;  o[I];N  oK;  b[I];N  bK;  iK;N  iK;  r[I];N  r[I]; |  |

Правила языка SMI-2023 реализованные на языке C++ представлены в приложении В.

## **4.3 Построение конечного магазинного автомата**

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

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата. |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $). |
|  | Функция переходов автомата | Функция, которая представляет из себя множество правил грамматики. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики. |
|  | Начальное состояние магазина автомата | Символ маркера дна стека $. |

Окончание таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты. |

Конечный автомат с магазинной памятью, также известный как автомат с "стеком", представляет собой устройство, способное принимать некоторое количество символов, выполнять определенные действия на основе входных данных и изменять свое состояние в соответствии с заранее заданными правилами.

## **4.4 Основные структуры данных**

Основными структурами данных синтаксического анализатора являются структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка SMI-2023. Данные структуры представлены в приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

− в магазин записывается стартовый символ;

− на основе полученных ранее таблиц формируется входная лента;

− запускается автомат;

− выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;

− если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбирается другая цепочка;

− если в магазине встретился нетерминал, переход к пункту 4;

− если символ достиг дна стека и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

## **4.6 Параметры синтаксического анализатора**

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

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

## **4.7. Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора, в котором макросом является макрос ERROR\_ENTRY, а категория ошибки SYNTAX представлен в таблице 4.3.

Таблица 4.3 – Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение об ошибке |
| 600 | Неверная структура программы |
| 601 | Отсутствует список параметров функции при её объявлении |
| 602 | Ошибка в параметрах функции при её объявлении |
| 603 | Ошибка в параметрах функции при её объявлении |
| 604 | Недопустимое выражение. Ожидаются только литералы и идентификаторы |
| 605 | Возможно отсутствует тело процедуры |
| 606 | Неверная конструкция в теле функции/процедуры |
| 607 | Ошибка в условном выражении |
| 608 | Ошибка в вызове функции |
| 609 | Ошибка при вычислении выражения |
| 610 | Ошибка в списке параметров при вызове функции |
| 611 | Неверная конструкция в теле цикла/условного выражения |
| 612 | Требуется закрывающая фигурная скобка |
| 613 | Требуется открывающая фигурная скобка |
| 614 | Отрицательное число требуется взять в скобки |
| 615 | Вызов функции в логическом выражении недопустим |

Макрос ERROR\_ENTRY служит для удобства и эффективности работы с синтаксическим анализатором. Ошибка категории SYNTAX является синтаксической ошибкой, которая возникает во время компиляции или интерпретации программы. Эта ошибка указывает на то, что в коде программы присутствует недопустимая конструкция или ошибка в синтаксисе языка программирования.

## **4.8. Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

− синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в та блице лексем;

− если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка;

− все ошибки записываются в общую структуру ошибок;

− в случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

## **4.9. Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода предоставлен в приложении Е в виде фрагмента трассировки и дерева разбора исходного кода.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализатор является важной составляющей в процессе проверки корректности программного кода. Он принимает на вход результаты работы лексического и синтаксического анализаторов, включающие таблицы лексем, идентификаторов и дерево разбора соответственно.

Задачей семантического анализатора является поиск и обнаружение ошибок, происходящих на более высоком уровне анализа программы. Для этого происходит последовательная проверка на соответствие различным семантическим правилам.

Некоторые проверки могут быть выполнены еще на этапе лексического анализа, такие как проверка на единственность точки входа или предварительное объявление переменной. Однако, более сложные проверки требуют использования семантического анализатора.

Общая структура семантического анализатора представлена на рисунке 5.1. Она обозначает взаимодействие с другими компонентами системы анализа кода и обнаружения ошибок.

Задача семантического анализатора - выявление ошибок, связанных с некорректным использованием языка программирования и нарушением правил его семантики. Он проверяет правильность применения операторов, типы данных, использование переменных и функций, а также другие аспекты программного кода.

При обнаружении ошибок, семантический анализатор может предоставить информацию о характере ошибки и ее местоположении разработчику, что позволяет ему вносить необходимые исправления. Это в свою очередь способствует созданию более эффективного и безопасного программного продукта.

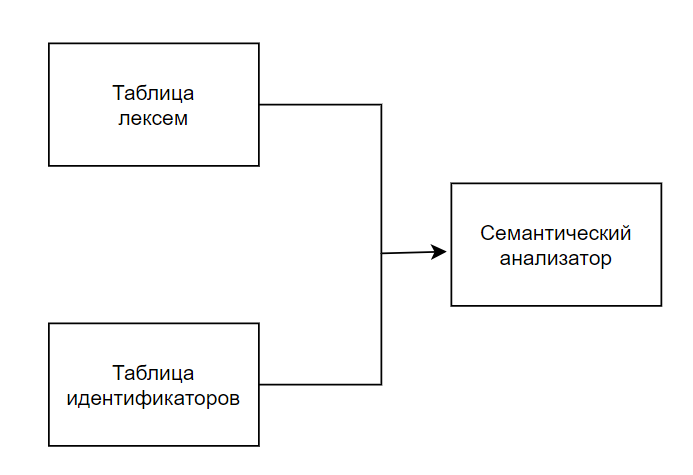


Рисунок 5.1 – Структура семантического анализатора

## **5.2 Функции семантического анализатора**

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

За семантический анализ отвечает функция AnalyzeSem. Ее входными параметрами является таблица лексем и таблица идентификаторов. Нет явного возвращаемого значения. Функция может выбросить исключение в случае нарушения семантических правил. Назначениями функции AnalyzeSem являются выполнение семантического анализа программы, проверка корректности использования идентификаторов, типов данных, операторов и других конструкций в программе.

Функция реализует следующие виды семантических проверок:

− проверка вызова функций и процедур с правильным числом и типами параметров;

− проверка правильности присваивания значений переменным;

− проверка корректности использования логических операторов.

## **5.3 Структура и перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, в котором макросом является макрос ERROR\_ENTRY, а категория ошибки SEMANTIC представлены на рисунке 5.1.

Таблица 5.1 – Сообщения, формируемые семантическим анализатором

|  |  |
| --- | --- |
| Код ошибки | Сообщение об ошибке |
| 300 | Не закрыт строковый литерал |
| 301 | Ожидается тип bool или number |
| 302 | Отсутствует точка входа program |
| 303 | Задано более одной точки входа program |
| 304 | Превышен размер строкового литерала |
| 305 | Объявление переменной без ключевого слова def недопустимо |
| 306 | Необъявленный идентификатор |
| 307 | Недопустимо объявление переменной без указания типа |
| 308 | Попытка реализовать уже существующую функцию |
| 309 | Попытка переопределить формальный параметр функции |
| 310 | Попытка переопределить переменную |
| 311 | Не указан тип функции |
| 312 | Процедура не должна иметь тип |
| 313 | Тип функции и тип возвращаемого значения должны совпадать |
| 314 | Превышено максимально допустимое (3) количество параметров функции |
| 315 | Несовпадение типов передаваемых параметров |
| 316 | Слишком много аргументов в вызове функции |
| 317 | Слишком мало аргументов в вызове функции |
| 318 | Использование пустого строкового литерала недопустимо |
| 319 | Недопустимый целочисленный литерал |
| 320 | Типы данных в выражении не совпадают |
| 321 | Арифметические операторы не могут применяться со строковым типом |

Окончание таблицы 5.1

|  |  |
| --- | --- |
| 322 | Логические операторы могут применяться только с целочисленными типами |
| 323 | Деление на ноль |

Макрос ERROR\_ENTRY служит для удобства и эффективности работы с семантическим анализатором. Ошибка категории SEMANTIC является ошибкой, которая возникает, когда программа компилируется без ошибок, но при выполнении проявляются некорректные или непредсказуемые результаты.

## **5.4 Принцип обработки ошибок**

В процессе трансляции программы могут возникать ошибки, которые фиксируются и записываются в протокол. Этот протокол задается входными параметрами программы. В случае возникновения ошибок, они протоколируются с указанием номера ошибки и диагностического сообщения. После этого семантический анализ программы останавливается.

Однако, несмотря на возникшие ошибки, поиск и анализ ошибок не прекращается. Даже после остановки семантического анализа продолжается изучение и поиск ошибок в программе. Это важный этап, который позволяет выявить и исправить все возможные проблемы, чтобы программа функционировала без сбоев и неправильного выполнения задач.

## **5.5 Контрольный пример**

Соответствие примеров некоторых семантических ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.2.

Таблица 5.2 – Примеры семантических ошибок

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| program  {  def line a = "Test;  outln[a];  } | Ошибка 300: #SEMANTIC - Не закрыт строковый литерал  Строка 3 позиция 15 |
| {  def number a = 5;  outln[a];  } | Ошибка 302: #SEMANTIC - Отсутствует точка входа program  Строка -1 позиция -1 |
| program  {  line a = "Test";  outln[a];  } | Ошибка 305: #SEMANTIC - Объявление переменной без ключевого слова def недопустимо  Строка 3 позиция 7 |
| program  { def a = 67;  outln[a]; } | Ошибка 307: #SEMANTIC - Недопустимо объявление переменной без указания типа  Строка 3 позиция 6 |

Окончание таблицы 5.2

|  |  |
| --- | --- |
| function MyFunc [number a, number b]  {  …  } | Ошибка 311: #SEMANTIC - Не указан тип функции  Строка 1 позиция -1 |
| procedure number MyProc [number a]  {  …  } | Ошибка 312: #SEMANTIC - Процедура не должна иметь тип  Строка 1 позиция -1 |
| function number MyFunc [number a]  {  def line lnA = "Hello";  give[lnA];  } program { … } | Ошибка 313: #SEMANTIC - Тип функции и тип возвращаемого значения должны совпадать  Строка 4 позиция -1 |
| program  {  def line a = 99999999999;  outln[a];  } | Ошибка 319: #SEMANTIC - Недопустимый целочисленный литерал  Строка 3 позиция 15 |
| program  {  def line a = 67;  outln[a];  } | Ошибка 320: #SEMANTIC - Типы данных в выражении не совпадают  Строка 3 позиция -1 |
| program  {  def line a = "Hello, " + "World!";  } | Ошибка 321: #SEMANTIC - Арифметические операторы не могут применяться со строковым типом  Строка 3 позиция -1 |
| program  {  def bool b = true < false;  } | Ошибка 322: #SEMANTIC - Логические операторы могут применяться только с целочисленными типами  Строка 3 позиция -1 |

Использование таблицы облегчает и улучшает процесс разработки программного обеспечения.

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке SMI-2023 допускается вычисление выражений целочисленного типа, также поддерживается вызов функций внутри арифметических выражений. В выражениях могут использоваться арифметические и операции сравнения. Операции и их приоритет представлен в таблице 6.1.

Таблица 6.1 – Операции и их приоритет

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

Примеры выражений языка SMI-2023 представлены на рисунке 6.1.

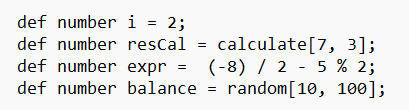


Рисунок 6.1 – Примеры выражений

В итоге, язык SMI-2023 предоставляет удобство и гибкость в использовании целочисленных типов данных. Можно производить вычисления, вызывать функции, а также использовать арифметические и операции сравнения.

## **6.2 Польская запись и принцип её построения**

Выражения в языке SMI-2023 преобразовываются к обратной польской записи.

Обратная польская запись — это форма записи математических выражений, в которой операторы расположены после своих операндов.

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

–операнды переносятся в результирующую строку в порядке их следования;

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

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

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

– отрывающая скобка помещается в стек;

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

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

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

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

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| z[l,l]vl | - | - |
| [l,l]vl | z | - |
| l,l]vl | z | [ |
| ,l]vl | zl | [ |
| l]vl | zl | [ |
| ]vl | zll | [ |
| vl | zll@2 | - |
| l | zll@2 | v |
| - | zll@2l | v |
| - | zll@2lv | - |

Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

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

# **7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. Структура генератора кода представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

Из рисунка можно заметить, что на вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

## **7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке SMI-2023 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка SMI-2023 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SMI-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| number | sword | Хранит целочисленный тип данных. |
| bool | sword | Хранит булевый тип данных (в виде целого числа) |
| line | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## **7.3 Статическая библиотека**

В языке SMI-2023 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Пример подключения библиотеки в исходном коде на языке ассемблера представлен в листинге 7.1.

|  |
| --- |
| includelib ..\Debug\Library.lib  ExitProcess PROTO :DWORD  EXTRN CONCAT: proc  EXTRN LINELEN: proc  EXTRN RANDOM: proc  EXTRN SQRT: proc  EXTRN OutNumber: proc  EXTRN OutLine: proc  EXTRN OutBool: proc  EXTRN OutNumberLn: proc  EXTRN OutLineLn: proc  EXTRN OutBoolLn: proc |

Листинг 7.1 – Подключение статической библиотеки

Подключение библиотеки происходит с помощью includelib на этапе генерации кода. Для доступа к функциям из библиотеки includelib в исходном коде на языке ассемблера необходимо использовать оператор EXTRN.

## **7.4 Особенности алгоритма генерации кода**

В языке SMI-2023 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

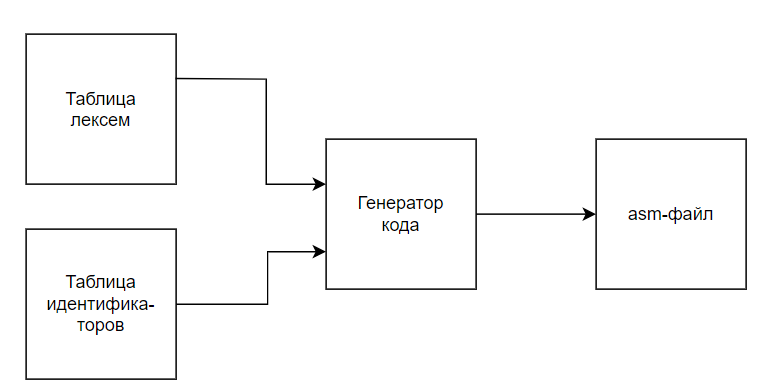


Рисунок 7.2 – Схема работы генератора кода

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

Таблица 7.2 – Прототипы функций, осуществляющих генерацию кода

|  |  |  |  |
| --- | --- | --- | --- |
| Прототип функции | Входные данные | Выходные данные | Описание функции |
| void Generation(LT::LexTable, IT::IdTable, wchar\_t) | На вход функция принимает таблицу лексем (LexTable), таблицу идентификаторов (IdTable), массив символов. | Выходными данными являются: заголовок программы (Head), Сегмент констант (ConstSegment), Сегмент данных (DataSegment), Сегмент кода (CodeSegment). | Основная функция. Формирует поток выходного файла и вызывает другие генерирующие функции. |
| void Head(ofstream\*); | На вход функция принимает указатель на объект типа ofstream, который представляет собой поток вывода в файл. | Выходные данные этой функции представляются в виде записанных строк в файл, который передан по указателю в качестве аргумента ofstream\* out. | Функция, генерирующая заголовок ассемблерного файла (подключение библиотек, указание прототипов функций и т.д.). |
| void ConstSegment(IT::IdTable, ofstream\*); | На вход функция принимает таблицу идентификаторов (IdTable), указатель на объект типа ofstream, который представляет собой поток вывода в файл. | Выходными данными является запись строк ассемблерного кода в указанный файловый поток. | Функция, генерирующая сегмент констант. |
| void CodeSegment (LT::LexTable, IT::IdTable, ofstream\*); | На вход функция принимает таблицу лексем (LexTable), таблицу идентификаторов (IdTable), указатель на файловый поток. | Выходными данными является ассемблерный код, который генерируется в процессе обработки лексической таблицы и идентификаторной таблицы. | Функция, генерирующая сегмент кода. |

Окончание таблицы 7.2

|  |  |  |  |
| --- | --- | --- | --- |
| void DataSegment (LT::LexTable, IT::IdTable, ofstream\*); | На вход функция принимает таблицу лексем (LexTable), таблицу идентификаторов (IdTable), указатель на объект типа ofstream, представляющий собой поток вывода в файл. | Выходными данными являются записи строк ассемблерного кода в указанный файловый поток. | Функция, генерирующая сегмент данных. |

В данной таблице описаны прототипы функций, а также входные, выходные данные и описание прототипов функций.

## **Параметры, управляющие генерацией кода**

Генератор кода является важным инструментом в разработке программного обеспечения. Его задачей является преобразование таблиц лексем и идентификаторов в машинный код, понятный компьютеру. При этом результаты работы генератора кода выводятся в специальный файл с расширением .asm.

Таблицы лексем и идентификаторов содержат информацию о синтаксических элементах программы и их значениях. Лексемы представляют собой отдельные элементы языка программирования, такие как ключевые слова, операторы, константы и т.д. Идентификаторы, в свою очередь, являются названиями переменных, функций или других элементов программы.

## **7.6 Контрольный пример**

Результат работы контрольного примера приведён на рисунке 7.3.

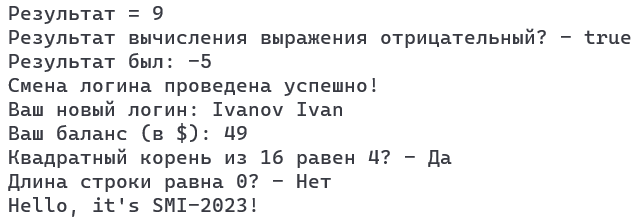


Рисунок 7.3 – Результат работы программы на языке SMI-2023

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

# **8. Тестирование транслятора**

## **8.1 Общие положения**

В процессе работы транслятора могут возникать ошибки на различных этапах трансляции. Эти этапы включают анализ исходного текста программы, лексический анализ, синтаксический анализ и семантический анализ. Когда возникают ошибки, транслятор отслеживает их и записывает в файл протокола. В этом файле указывается идентификатор ошибки, сообщение об ошибке, а также строка и позиция в исходном тексте программы, где ошибка была обнаружена. Как правило, после возникновения ошибки работа транслятора прекращается, поскольку ошибка на одном из этапов трансляции может вызвать ошибки на последующих этапах, за исключением синтаксического анализатора.

## **8.2 Результаты тестирования**

В таблице 8.1 приведены результаты тестов для разных этапов трансляции.

Таблица 8.1 – Результаты тестов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Проверка на допустимость символов | |
| pro№gram  {  } | Ошибка 200: #LEXICAL - Недопустимый символ в исходном файле (-in)  Строка 1 позиция 4 |
| Лексический анализ | |
| function number func! [number a, number b]  { …} | Ошибка 205: #LEXICAL - Лексема не распознана  Строка 1 позиция 17 |
| function number MyLongNamedFunction [number a]  {…} | Ошибка 206: #LEXICAL - Длина идентификатора не должна превышать 10 символов  Строка 1 позиция 17 |
| Синтаксический анализ | |
| program def number a  {…} | Ошибка 600: строка 1, #SYNTAX - Неверная структура программы |
| function number myFunc  {  give[0];  }  program  {…} | Ошибка 601: строка 2, #SYNTAX - Отсутствует список параметров функции при её объявлении |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| function number func [number]  {  give[8];  }  program  {…} | Ошибка 602: строка 1, #SYNTAX - Ошибка в параметрах функции при её объявлении |
| function number myFunc []  program  {…} | Ошибка 603: строка 3, #SYNTAX - Возможно отсутствует тело функции |
| function number myFunc []  {  give[1 + 1];  }  program  {…} | Ошибка 604: строка 3, #SYNTAX - Недопустимое выражение. Ожидаются только литералы и идентификаторы |
| procedure myProc []  program  {…} | Ошибка 605: строка 3, #SYNTAX - Возможно отсутствует тело процедуры |
| function number myFunc []  {  program  {  def number a = 9;  }  } | Ошибка 606: строка 3, #SYNTAX - Неверная конструкция в теле функции |
| program  {  when[1 > 8 > 9]  {  def number a = 1;  }  } | Ошибка 607: строка 3, #SYNTAX - Ошибка в условном выражении |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| function number myFunc [number a]  { a = a + 2;  give[a];  }  program{def number num = myFunc;} | Ошибка 608: #SYNTAX - Ошибка в вызове функции  Строка 9 позиция -1 |
| program  {  def number num = 9 \* 8);  } | Ошибка 609: строка 3, #SYNTAX - Ошибка при вычислении выражения |
| function number myFunc[number a, number b]  {  def number res = a + b;  give[res];  }  program  {  def number res = myFunc[2,,3];  } | Ошибка 610: строка 8, #SYNTAX - Ошибка в списке параметров при вызове функции |
| program  def number a = 9;  } | Ошибка 613: #SYNTAX - Требуется открывающая фигурная скобка |
| program  {  def number a; | Ошибка 612: #SYNTAX - Требуется закрывающая фигурная скобка |
| program{  def number a = 8 - -9;  } | Ошибка 614: #SYNTAX - Отрицательное число требуется взять в скобки  Строка 3 позиция -1 |
| Семантический анализ | |
| program{  when["line"]  {  out[true];  }  } | Ошибка 301: #SEMANTIC - Ожидается тип bool или number  Строка 3 позиция -1 |
| program{  when[a == 1] {…}  } | Ошибка 306: #SEMANTIC - Необъявленный идентификатор  Строка 3 позиция -1 |

Окончание таблицы 8.1

|  |  |
| --- | --- |
| function number MyFunc[number a]{…}  function number MyFunc[number a, number b]{…} | Ошибка 308: #SEMANTIC - Попытка реализовать уже существующую функцию  Строка 5 позиция -1 |
| function number MyFunc[number a, number b]  { def number a = 9; } | Ошибка 309: #SEMANTIC - Попытка переопределить формальный параметр функции  Строка 3 позиция -1 |
| def line ln = "hello" + "world"; | Ошибка 321: #SEMANTIC - Арифметические операторы не могут применяться со строковым типом  Строка 3 позиция -1 |
| def number a = 9 / 0; | Ошибка 323: #SEMANTIC - Деление на ноль  Строка 3 позиция -1 |

Тестирование прошло успешно.

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования SMI-2023, а также генератор кода, который переводит в ассемблер. Были выполнены основные задачи курсового проекта:

– разработан дизайн языка;

– разработана спецификация языка программирования;

– разработана структура транслятора;

– разработан лексический анализатор;

– разработан синтаксический анализатор;

– разработан семантический анализатор;

– разработан генератор кода на язык ассемблера;

– проведено тестирование транслятора.

Итоговая версия языка SMI-2023 включает:

* 3 типа данных;
* поддержку операторов вывода;
* возможность вызова функций стандартной библиотеки;
* наличие 5 арифметических операторов для вычисления выражений;
* наличие 6 операторов сравнения для вычисления выражений;
* поддержка функций, процедур, условный оператор.

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

# **Список использованных литературных источников**

1. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.

2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

3. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

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

5. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

**Приложение А**

|  |
| --- |
| $Пользовательские функции  function number power [number a, number b]  {  def number i = 2;  def number res = a \* 3 - ( b + 2 );  give [res];  }  $Процедура  procedure yourLogin [line login]  {  outln["Смена логина проведена успешно!"];  out["Ваш новый логин: "];  outln[login];  }  program  {  $Вызов пользовательских функций  out["Результат: "];  def number resPow = power[6, 3];  outln[resPow];  out["Результат вычисления выражения отрицательный? - "];  def number expr = (-8) / 2 - 5 % 2;  def bool resNeg = expr <= 0;  outln[resNeg];  out["Результат был: "]; outln[expr];  $Вызов процедуры  def line login = "Ivanov Ivan";  yourLogin[login];  $Вызов функций стандартной библиотеки  out["Ваш баланс (в $): "];  def number balance = random[10, 100] + 5;  outln[balance];  out["Квадратный корень из 10 равен 4? - "];  def number resSqrt = sqrt[10];  when [resSqrt == 4]  {  outln["Да"];  }  otherwise  {  outln["Нет"];  }  out["Длина строки равна 0? - "];  def line lineA = "abcdef";  def number lengthA = linelen[lineA];  def bool resA = lengthA ^= 0;  when [resA]  {  outln["Нет"];  }  otherwise  {  outln["Да"];  }  def line resConcat = concat["Hello, ", "it's SMI-2023!"];  outln[resConcat];  } |

**Приложение Б**

|  |
| --- |
| #define GRAPH\_DEFINE 4, \  FST::NODE(1, FST::RELATION('d', 1)), \  FST::NODE(1, FST::RELATION('e', 2)), \  FST::NODE(1, FST::RELATION('f', 3)), \  FST::NODE()  #define GRAPH\_NUMBER 7, \  FST::NODE(1,FST::RELATION('n', 1)),\  FST::NODE(1,FST::RELATION('u', 2)),\  FST::NODE(1,FST::RELATION('m', 3)),\  FST::NODE(1,FST::RELATION('b', 4)),\  FST::NODE(1,FST::RELATION('e', 5)),\  FST::NODE(1,FST::RELATION('r', 6)),\  FST::NODE()  #define GRAPH\_LINE 5, \  FST::NODE(1,FST::RELATION('l', 1)),\  FST::NODE(1,FST::RELATION('i', 2)),\  FST::NODE(1,FST::RELATION('n', 3)),\  FST::NODE(1,FST::RELATION('e', 4)),\  FST::NODE()  #define GRAPH\_BOOL 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 GRAPH\_TRUE 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 GRAPH\_FALSE 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 GRAPH\_OUTLN 6,\  FST::NODE(1, FST::RELATION('o', 1)), \  FST::NODE(1, FST::RELATION('u', 2)), \  FST::NODE(1, FST::RELATION('t', 3)), \  FST::NODE(1, FST::RELATION('l', 4)), \  FST::NODE(1, FST::RELATION('n', 5)), \  FST::NODE()  #define GRAPH\_OPERATOR 2, \  FST::NODE(5, FST::RELATION('+', 1), \  FST::RELATION('-', 1), \  FST::RELATION('\*', 1), \  FST::RELATION('/', 1), \  FST::RELATION('%', 1)),\  FST::NODE()  #define GRAPH\_LOG\_OPERATOR 2, \  FST::NODE(2, FST::RELATION('>', 1), \  FST::RELATION('<', 1)), \  FST::NODE()  #define GRAPH\_EQUALS 3, \  FST::NODE(1,FST::RELATION('=', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_NOTEQUALS 3, \  FST::NODE(1,FST::RELATION('^', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_MOREEQUALS 3, \  FST::NODE(1,FST::RELATION('>', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_LESSEQUALS 3, \  FST::NODE(1,FST::RELATION('<', 1)), \  FST::NODE(1,FST::RELATION('=', 2)), \  FST::NODE()  #define GRAPH\_SEMICOLON 2, \  FST::NODE(1, FST::RELATION(';', 1)),\  FST::NODE()  #define GRAPH\_COMMA 2, \  FST::NODE(1, FST::RELATION(',', 1)),\  FST::NODE()  #define GRAPH\_LEFTBRACE 2, \  FST::NODE(1, FST::RELATION('{', 1)),\  FST::NODE()  #define GRAPH\_BRACELET 2, \  FST::NODE(1, FST::RELATION('}', 1)),\  FST::NODE()  #define GRAPH\_LEFTTHESIS 2, \  FST::NODE(1, FST::RELATION('[', 1)),\  FST::NODE()  #define GRAPH\_RIGHTTHESIS 2, \  FST::NODE(1, FST::RELATION(']', 1)),\  FST::NODE()  #define GRAPH\_LEFTBRACKET 2, \  FST::NODE(1, FST::RELATION('(', 1)),\  FST::NODE()  #define GRAPH\_RIGHTBRACKET 2, \  FST::NODE(1, FST::RELATION(')', 1)),\  FST::NODE()  #define GRAPH\_EQUAL 2, \  FST::NODE(1, FST::RELATION('=', 1)),\  FST::NODE()  #define GRAPH\_ID 2, \  FST::NODE(104, \  FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('c', 0), \  FST::RELATION('d', 0), FST::RELATION('e', 0), FST::RELATION('f', 0), \  FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('i', 0), \  FST::RELATION('j', 0), FST::RELATION('k', 0), FST::RELATION('l', 0), \  FST::RELATION('m', 0), FST::RELATION('n', 0), FST::RELATION('o', 0), \  FST::RELATION('p', 0), FST::RELATION('q', 0), FST::RELATION('r', 0), \  FST::RELATION('s', 0), FST::RELATION('t', 0), FST::RELATION('u', 0), \  FST::RELATION('v', 0), FST::RELATION('w', 0), FST::RELATION('x', 0), \  FST::RELATION('y', 0), FST::RELATION('z', 0),\  FST::RELATION('A', 0), FST::RELATION('B', 0), FST::RELATION('C', 0), \  FST::RELATION('D', 0), FST::RELATION('E', 0), FST::RELATION('F', 0), \  FST::RELATION('G', 0), FST::RELATION('H', 0), FST::RELATION('I', 0), \  FST::RELATION('J', 0), FST::RELATION('K', 0), FST::RELATION('L', 0), \  FST::RELATION('M', 0), FST::RELATION('N', 0), FST::RELATION('O', 0), \  FST::RELATION('P', 0), FST::RELATION('Q', 0), FST::RELATION('R', 0), \  FST::RELATION('S', 0), FST::RELATION('T', 0), FST::RELATION('U', 0), \  FST::RELATION('V', 0), FST::RELATION('W', 0), FST::RELATION('X', 0), \  FST::RELATION('Y', 0), FST::RELATION('Z', 0),\  \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), \  FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1), \  FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), \  FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1), \  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), \  FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1), \  FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), \  FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1), \  FST::RELATION('y', 1), FST::RELATION('z', 1),\  FST::RELATION('A', 1), FST::RELATION('B', 1), FST::RELATION('C', 1), \  FST::RELATION('D', 1), FST::RELATION('E', 1), FST::RELATION('F', 1), \  FST::RELATION('G', 1), FST::RELATION('H', 1), FST::RELATION('I', 1), \  FST::RELATION('J', 1), FST::RELATION('K', 1), FST::RELATION('L', 1), \  FST::RELATION('M', 1), FST::RELATION('N', 1), FST::RELATION('O', 1), \  FST::RELATION('P', 1), FST::RELATION('Q', 1), FST::RELATION('R', 1), \  FST::RELATION('S', 1), FST::RELATION('T', 1), FST::RELATION('U', 1), \  FST::RELATION('V', 1), FST::RELATION('W', 1), FST::RELATION('X', 1), \  FST::RELATION('Y', 1), FST::RELATION('Z', 1)),\  FST::NODE()  #define GRAPH\_LINE\_LITERAL 3,\  FST::NODE(1, FST::RELATION('\"', 1)),\  FST::NODE(157, \  FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1),\  FST::RELATION('e', 1), FST::RELATION('f', 1), FST::RELATION('g', 1), FST::RELATION('h', 1),\  FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\  FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1),\  FST::RELATION('q', 1), FST::RELATION('r', 1), FST::RELATION('s', 1), FST::RELATION('t', 1),\  FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\  FST::RELATION('y', 1), FST::RELATION('z', 1),\  FST::RELATION('A', 1), FST::RELATION('B', 1), FST::RELATION('C', 1), FST::RELATION('D', 1),\  FST::RELATION('E', 1), FST::RELATION('F', 1), FST::RELATION('G', 1), FST::RELATION('H', 1),\  FST::RELATION('I', 1), FST::RELATION('J', 1), FST::RELATION('K', 1), FST::RELATION('L', 1),\  FST::RELATION('M', 1), FST::RELATION('N', 1), FST::RELATION('O', 1), FST::RELATION('P', 1),\  FST::RELATION('Q', 1), FST::RELATION('R', 1), FST::RELATION('S', 1), FST::RELATION('T', 1),\  FST::RELATION('U', 1), FST::RELATION('V', 1), FST::RELATION('W', 1), FST::RELATION('X', 1),\  FST::RELATION('Y', 1), FST::RELATION('Z', 1), FST::RELATION('а', 1), FST::RELATION('б', 1), \  FST::RELATION('в', 1), FST::RELATION('г', 1), FST::RELATION('д', 1), FST::RELATION('е', 1), \  FST::RELATION('ё', 1), FST::RELATION('ж', 1), FST::RELATION('з', 1), FST::RELATION('и', 1), \  FST::RELATION('й', 1), FST::RELATION('к', 1), FST::RELATION('л', 1), FST::RELATION('м', 1), \  FST::RELATION('н', 1), FST::RELATION('о', 1), FST::RELATION('п', 1), FST::RELATION('р', 1), \  FST::RELATION('с', 1), FST::RELATION('т', 1), FST::RELATION('у', 1), FST::RELATION('ф', 1), \  FST::RELATION('х', 1), FST::RELATION('ц', 1), FST::RELATION('ч', 1), FST::RELATION('ш', 1), \  FST::RELATION('щ', 1), FST::RELATION('ь', 1), FST::RELATION('ъ', 1), FST::RELATION('ы', 1), \  FST::RELATION('э', 1), FST::RELATION('ю', 1), FST::RELATION('я', 1), FST::RELATION('А', 1), \  FST::RELATION('Б', 1), FST::RELATION('В', 1), FST::RELATION('Г', 1), FST::RELATION('Д', 1), \  FST::RELATION('Е', 1), FST::RELATION('Ё', 1), FST::RELATION('Ж', 1), FST::RELATION('З', 1), \  FST::RELATION('И', 1), FST::RELATION('Й', 1), FST::RELATION('К', 1), FST::RELATION('Л', 1), \  FST::RELATION('М', 1), FST::RELATION('Н', 1), FST::RELATION('О', 1), FST::RELATION('П', 1), \  FST::RELATION('Р', 1), FST::RELATION('С', 1), FST::RELATION('Т', 1), FST::RELATION('У', 1), \  FST::RELATION('Ф', 1), FST::RELATION('Х', 1), FST::RELATION('Ц', 1), FST::RELATION('Ч', 1), \  FST::RELATION('Ш', 1), FST::RELATION('Щ', 1), FST::RELATION('Ь', 1), FST::RELATION('Ъ', 1), \  FST::RELATION('Ы', 1), FST::RELATION('Э', 1), FST::RELATION('Ю', 1), FST::RELATION('Я', 1), \  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('$', 1), FST::RELATION('%', 1),\  FST::RELATION(' ', 1), FST::RELATION(',', 1), FST::RELATION('.', 1), FST::RELATION(';', 1),\  FST::RELATION('-', 1), FST::RELATION('+', 1), FST::RELATION('\*', 1), FST::RELATION('/', 1),\  FST::RELATION('=', 1), FST::RELATION(':', 1), FST::RELATION(')', 1), FST::RELATION('(', 1),\  FST::RELATION('}', 1), FST::RELATION('{', 1), FST::RELATION(']', 1), FST::RELATION('[', 1),\  FST::RELATION('!', 1), FST::RELATION('?', 1), FST::RELATION('#', 1), FST::RELATION('&', 1),\  FST::RELATION('>', 1), FST::RELATION('<', 1), FST::RELATION('[', 1), FST::RELATION(']', 1),\  FST::RELATION('^', 1), FST::RELATION('\'', 1),\  FST::RELATION('\"', 2)),\  FST::NODE()  #define GRAPH\_NUM\_LITERAL 3, \  FST::NODE(19, \  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(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 GRAPH\_FUNCTION 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 GRAPH\_PROCEDURE 10, \  FST::NODE(1, FST::RELATION('p', 1)),\  FST::NODE(1, FST::RELATION('r', 2)),\  FST::NODE(1, FST::RELATION('o', 3)),\  FST::NODE(1, FST::RELATION('c', 4)),\  FST::NODE(1, FST::RELATION('e', 5)),\  FST::NODE(1, FST::RELATION('d', 6)),\  FST::NODE(1, FST::RELATION('u', 7)),\  FST::NODE(1, FST::RELATION('r', 8)),\  FST::NODE(1, FST::RELATION('e', 9)),\  FST::NODE()  #define GRAPH\_PROGRAM 8, \  FST::NODE(1,FST::RELATION('p', 1)),\  FST::NODE(1,FST::RELATION('r', 2)),\  FST::NODE(1,FST::RELATION('o', 3)),\  FST::NODE(1,FST::RELATION('g', 4)),\  FST::NODE(1,FST::RELATION('r', 5)),\  FST::NODE(1,FST::RELATION('a', 6)),\  FST::NODE(1,FST::RELATION('m', 7)),\  FST::NODE()  #define GRAPH\_WHEN 5, \  FST::NODE(1,FST::RELATION('w', 1)),\  FST::NODE(1,FST::RELATION('h', 2)),\  FST::NODE(1,FST::RELATION('e', 3)),\  FST::NODE(1,FST::RELATION('n', 4)),\  FST::NODE()  #define GRAPH\_OTHERWISE 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 GRAPH\_CYCLE 6, \  FST::NODE(1,FST::RELATION('c', 1)),\  FST::NODE(1,FST::RELATION('y', 2)),\  FST::NODE(1,FST::RELATION('c', 3)),\  FST::NODE(1,FST::RELATION('l', 4)),\  FST::NODE(1,FST::RELATION('e', 5)),\  FST::NODE()  #define GRAPH\_OUT 4, \  FST::NODE(1, FST::RELATION('o', 1)),\  FST::NODE(1, FST::RELATION('u', 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE()  #define GRAPH\_GIVE 5, \  FST::NODE(1,FST::RELATION('g',1)),\  FST::NODE(1,FST::RELATION('i',2)),\  FST::NODE(1,FST::RELATION('v',3)),\  FST::NODE(1,FST::RELATION('e',4)),\  FST::NODE()  #define GRAPH\_LINELEN 8, \  FST::NODE(1, FST::RELATION('l', 1)),\  FST::NODE(1, FST::RELATION('i', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('e', 4)),\  FST::NODE(1, FST::RELATION('l', 5)),\  FST::NODE(1, FST::RELATION('e', 6)),\  FST::NODE(1, FST::RELATION('n', 7)),\  FST::NODE()  #define GRAPH\_CONCAT 7, \  FST::NODE(1, FST::RELATION('c', 1)),\  FST::NODE(1, FST::RELATION('o', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('c', 4)),\  FST::NODE(1, FST::RELATION('a', 5)),\  FST::NODE(1, FST::RELATION('t', 6)),\  FST::NODE()  #define GRAPH\_SQRT 5, \  FST::NODE(1, FST::RELATION('s', 1)),\  FST::NODE(1, FST::RELATION('q', 2)),\  FST::NODE(1, FST::RELATION('r', 3)),\  FST::NODE(1, FST::RELATION('t', 4)),\  FST::NODE()  #define GRAPH\_RANDOM 7, \  FST::NODE(1, FST::RELATION('r', 1)),\  FST::NODE(1, FST::RELATION('a', 2)),\  FST::NODE(1, FST::RELATION('n', 3)),\  FST::NODE(1, FST::RELATION('d', 4)),\  FST::NODE(1, FST::RELATION('o', 5)),\  FST::NODE(1, FST::RELATION('m', 6)),\  FST::NODE() |

Таблица идентификаторов:

|  |
| --- |
| ==============================================================================  № | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение/Параметры  ----------------------------------------------------------------------------------------------------  0000 | power | number | функция | 2 | P0:N|P1:N|  0001 | power\_a | number | параметр | 5 | -  0002 | power\_b | number | параметр | 8 | -  0003 | power\_i | number | переменная | 13 | 0  0004 | lit1 | number | литерал | 15 | 2  0005 | power\_res | number | переменная | 19 | 0  0006 | lit2 | number | литерал | 23 | 3  0007 | yourLogin | - | функция | 38 | P0:L|  0008 | yourLogin\_login | line | параметр | 41 | -  0009 | lit3 | line | литерал | 46 | [31]"Смена логина проведена успешно!"  0010 | lit4 | line | литерал | 51 | [17]"Ваш новый логин: "  0011 | lit5 | line | литерал | 64 | [11]"Результат: "  0012 | program\_resPow | number | переменная | 69 | 0  0013 | lit6 | number | литерал | 73 | 6  0014 | lit7 | line | литерал | 85 | [48]"Результат вычисления выражения отрицательный? - "  0015 | program\_expr | number | переменная | 90 | 0  0016 | lit8 | number | литерал | 94 | -8  0017 | lit9 | number | литерал | 99 | 5  0018 | program\_resNeg | bool | переменная | 105 | false  0019 | lit10 | number | литерал | 109 | 0  0020 | lit11 | line | литерал | 118 | [15]"Результат был: "  0021 | program\_login | line | переменная | 128 | [0]""  0022 | lit12 | line | литерал | 130 | [11]"Ivanov Ivan"  0023 | lit13 | line | литерал | 139 | [18]"Ваш баланс (в $): "  0024 | program\_balance | number | переменная | 144 | 0  0025 | random | number | функция ст. библ. | 146 | P0:N|P1:N|  0026 | lit14 | number | литерал | 148 | 10  0027 | lit15 | number | литерал | 150 | 100  0028 | lit16 | line | литерал | 162 | [35]"Квадратный корень из 10 равен 4? -"  0029 | program\_resSqrt | number | переменная | 167 | 0  0030 | sqrt | number | функция ст. библ. | 169 | P0:N|  0031 | lit17 | number | литерал | 178 | 4  0032 | lit18 | line | литерал | 183 | [2]"Да"  0033 | lit19 | line | литерал | 191 | [3]"Нет"  0034 | lit20 | line | литерал | 197 | [24]"Длина строки равна 0? - "  0035 | program\_lineA | line | переменная | 202 | [0]""  0036 | lit21 | line | литерал | 204 | [6]"abcdef"  0037 | program\_lengthA | number | переменная | 208 | 0  0038 | linelen | number | функция ст. библ. | 210 | P0:L|  0039 | program\_resA | bool | переменная | 217 | false  0040 | program\_resConcat | line | переменная | 244 | [0]""  0041 | concat | line | функция ст. библ. | 246 | P0:L|P1:L|  0042 | lit22 | line | литерал | 248 | [7]"Hello, "  0043 | lit23 | line | литерал | 250 | [14]"it's SMI-2023!" |

Начало таблицы лексем:

|  |
| --- |
| № | Лексема | Строка | Индекс в ТИ  ----------------------------------------------------------------------------------------------------  0000 | f | 3 | -  0001 | t | 3 | -  0002 | i | 3 | 0  0003 | [ | 3 | -  0004 | t | 3 | -  0005 | i | 3 | 1  0006 | , | 3 | -  0007 | t | 3 | -  0008 | i | 3 | 2  0009 | ] | 3 | -  0010 | { | 4 | -  0011 | d | 5 | -  0012 | t | 5 | -  0013 | i | 5 | 3  0014 | = | 5 | -  0015 | l | 5 | 4  0016 | ; | 5 | -  0017 | d | 6 | -  0018 | t | 6 | -  0019 | i | 6 | 5  0020 | = | 6 | -  0021 | i | 6 | 1  0022 | v | 6 | -  0023 | l | 6 | 6  0024 | v | 6 | -  0025 | ( | 6 | -  0026 | i | 6 | 2  0027 | v | 6 | -  0028 | l | 6 | 4  0029 | ) | 6 | -  0030 | ; | 6 | - |

Окончание таблицы лексем:

|  |
| --- |
| 0228 | [ | 57 | -  0229 | l | 57 | 33  0230 | ] | 57 | -  0231 | ; | 57 | -  0232 | } | 58 | -  0233 | ! | 59 | -  0234 | { | 60 | -  0235 | b | 61 | -  0236 | [ | 61 | -  0237 | l | 61 | 32  0238 | ] | 61 | -  0239 | ; | 61 | -  0240 | } | 62 | -  0241 | d | 64 | -  0242 | t | 64 | -  0243 | i | 64 | 40  0244 | = | 64 | -  0245 | + | 64 | 41  0246 | [ | 64 | -  0247 | l | 64 | 42  0248 | , | 64 | -  0249 | l | 64 | 43  0250 | ] | 64 | -  0251 | ; | 64 | -  0252 | b | 65 | -  0253 | [ | 65 | -  0254 | i | 65 | 40  0255 | ] | 65 | -  0256 | ; | 65 | -  0257 | } | 68 | - |

**Приложение В**

Правила языка SMI-2023

|  |
| --- |
| Greibach greibach(  NS('S'), TS('$'),  13,  Rule(  NS('S'), GRB\_ERROR\_SERIES + 0,  5,  Rule::Chain(6, TS('f'), TS('t'), TS('i'), NS('F'), NS('B'), NS('S')),  Rule::Chain(5, TS('s'), TS('i'), NS('F'), NS('U'), NS('S')),  Rule::Chain(4, TS('p'), TS('{'), NS('N'), TS('}')),  Rule::Chain(5, TS('f'), TS('t'), TS('i'), NS('F'), NS('B')),  Rule::Chain(4, TS('s'), TS('i'), NS('F'), NS('U'))  ),  Rule(  NS('F'), GRB\_ERROR\_SERIES + 1,  2,  Rule::Chain(2, TS('['), TS(']')),  Rule::Chain(3, TS('['), NS('P'), TS(']'))  ),  Rule(  NS('P'), GRB\_ERROR\_SERIES + 2,  2,  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('P'))  ),  Rule(  NS('B'), GRB\_ERROR\_SERIES + 3,  2,  Rule::Chain(8, TS('{'), NS('N'), TS('r'), TS('['), NS('I'), TS(']'), TS(';'), TS('}')),  Rule::Chain(7, TS('{'), TS('r'), TS('['), NS('I'), TS(']'), TS(';'), TS('}'))  ),  Rule(  NS('I'), GRB\_ERROR\_SERIES + 4,  2,  Rule::Chain(1, TS('l')),  Rule::Chain(1, TS('i'))  ),  Rule(  NS('U'), GRB\_ERROR\_SERIES + 5,  1,  Rule::Chain(3, TS('{'), NS('N'), TS('}'))  ),  Rule(  NS('N'), GRB\_ERROR\_SERIES + 6,  33,  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('R'), TS(';'), NS('N')),  Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('R'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(8, TS('u'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'), NS('N')),  Rule::Chain(8, TS('w'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'), NS('N')),  Rule::Chain(12, TS('w'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'), TS('!'), TS('{'), NS('X'), TS('}'), NS('N')),  Rule::Chain(6, TS('%'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('+'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('q'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('z'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('o'), TS('['), NS('I'), TS(']'), TS(';'), NS('N')),  Rule::Chain(6, TS('b'), TS('['), NS('I'), TS(']'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), NS('K'), TS(';'), NS('N')),  Rule::Chain(5, TS('r'), TS('['), NS('I'), TS(']'), TS(';')),  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(6, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(6, TS('d'), TS('t'), TS('i'), TS('='), NS('R'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('R'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(7, TS('u'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}')),  Rule::Chain(7, TS('w'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}')),  Rule::Chain(11, TS('w'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'), TS('!'), TS('{'), NS('X'), TS('}')),  Rule::Chain(3, TS('+'), NS('K'), TS(';')),  Rule::Chain(3, TS('%'), NS('K'), TS(';')),  Rule::Chain(3, TS('q'), NS('K'), TS(';')),  Rule::Chain(3, TS('z'), NS('K'), TS(';')),  Rule::Chain(5, TS('o'), TS('['), NS('I'), TS(']'), TS(';')),  Rule::Chain(3, TS('o'), NS('K'), TS(';')),  Rule::Chain(5, TS('b'), TS('['), NS('I'), TS(']'), TS(';')),  Rule::Chain(3, TS('b'), NS('K'), TS(';')),  Rule::Chain(3, TS('i'), NS('K'), TS(';'))  ),  Rule(  NS('R'), GRB\_ERROR\_SERIES + 7,  6,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS('g'), TS('i')),  Rule::Chain(3, TS('i'), TS('g'), TS('l')),  Rule::Chain(3, TS('l'), TS('g'), TS('i')),  Rule::Chain(3, TS('l'), TS('g'), TS('l'))  ),  Rule(  NS('K'), GRB\_ERROR\_SERIES + 8,  2,  Rule::Chain(3, TS('['), NS('W'), TS(']')),  Rule::Chain(2, TS('['), TS(']'))  ),  Rule(  NS('E'), GRB\_ERROR\_SERIES + 9,  16,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(2, TS('i'), NS('K')),  Rule::Chain(2, TS('%'), NS('K')),  Rule::Chain(2, TS('+'), NS('K')),  Rule::Chain(2, TS('q'), NS('K')),  Rule::Chain(2, TS('z'), NS('K')),  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('i'), NS('K'), NS('M')),  Rule::Chain(3, TS('%'), NS('K'), NS('M')),  Rule::Chain(3, TS('+'), NS('K'), NS('M')),  Rule::Chain(3, TS('q'), NS('K'), NS('M')),  Rule::Chain(3, TS('z'), NS('K'), NS('M'))  ),  Rule(  NS('W'), GRB\_ERROR\_SERIES + 10,  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(  NS('M'), GRB\_ERROR\_SERIES + 9,  2,  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(3, TS('v'), NS('E'), NS('M'))  ),  Rule(  NS('X'), GRB\_ERROR\_SERIES + 11,  26,  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('R'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('R'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(6, TS('%'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('+'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('q'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('z'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('o'), TS('['), NS('I'), TS(']'), TS(';'), NS('N')),  Rule::Chain(6, TS('b'), TS('['), NS('I'), TS(']'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), NS('K'), TS(';'), NS('N')),  Rule::Chain(6, TS('r'), TS('['), NS('I'), TS(']'), TS(';'), NS('N')),  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(6, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(6, TS('d'), TS('t'), TS('i'), TS('='), NS('R'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('R'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(3, TS('+'), NS('K'), TS(';')),  Rule::Chain(3, TS('q'), NS('K'), TS(';')),  Rule::Chain(3, TS('z'), NS('K'), TS(';')),  Rule::Chain(3, TS('%'), NS('K'), TS(';')),  Rule::Chain(3, TS('o'), NS('K'), TS(';')),  Rule::Chain(3, TS('b'), NS('K'), TS(';')),  Rule::Chain(3, TS('i'), NS('K'), TS(';')),  Rule::Chain(5, TS('r'), TS('['), NS('I'), TS(']'), TS(';'))  )  ); |

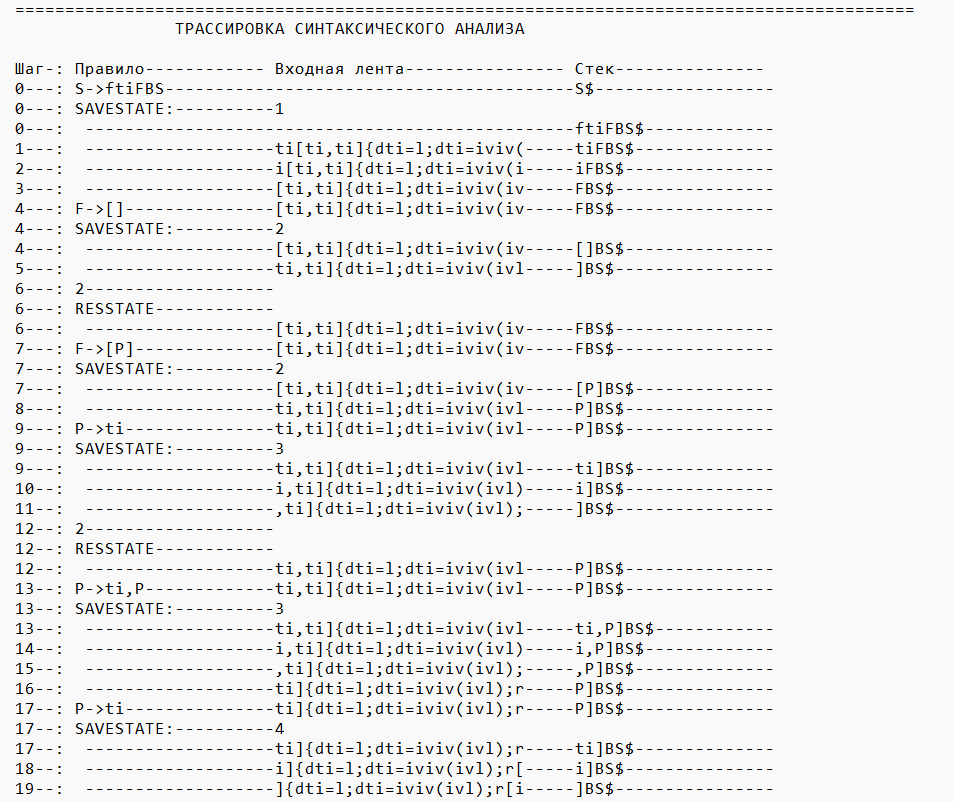
|  |
| --- |
|  |

**Приложение Г**

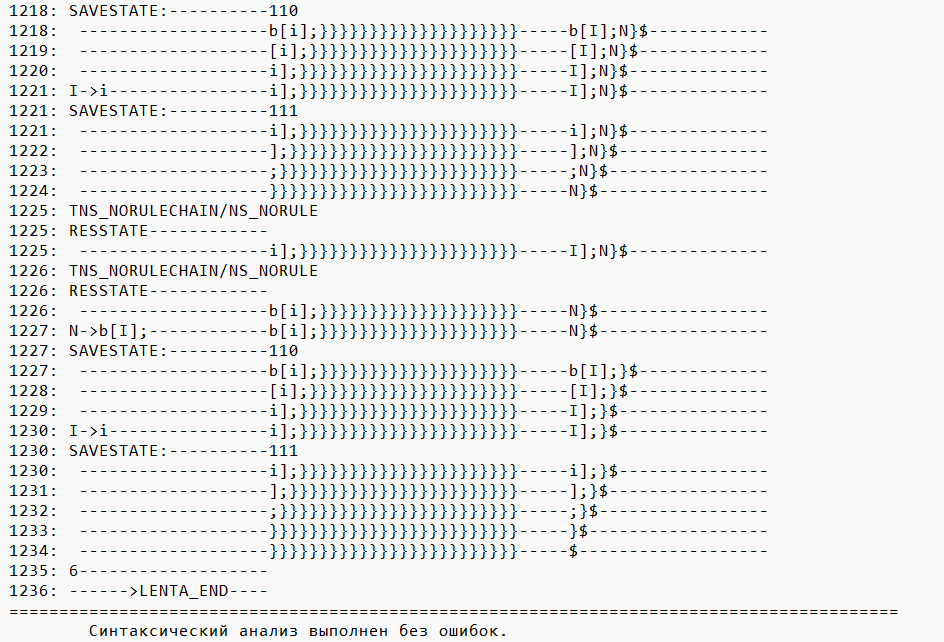
Данные структуры:

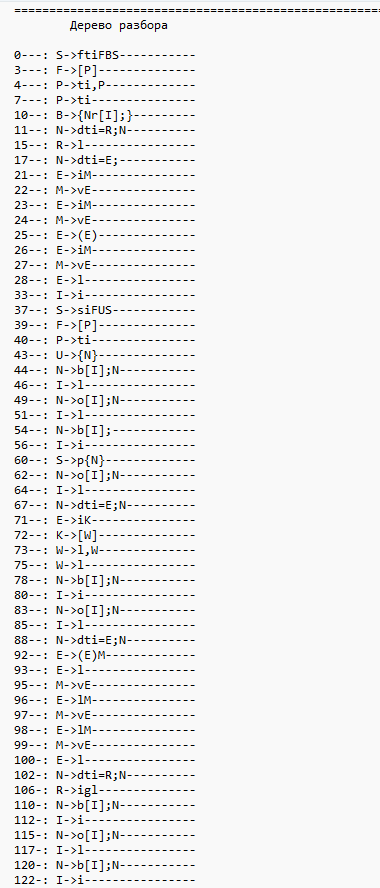
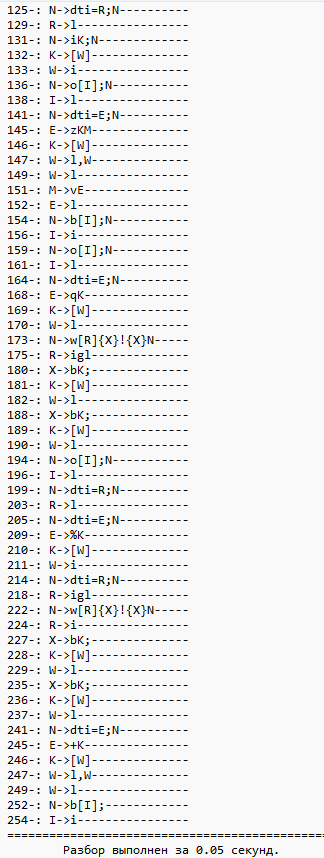
|  |
| --- |
| struct MfstState  {  short lenta\_position;  short nrule;  short nrulechain;  MFSTSTSTACK st;  MfstState();  MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);  MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);  };  struct Mfst  {  enum RC\_STEP  {  NS\_OK,  NS\_NORULE,  NS\_NORULECHAIN,  NS\_ERROR,  TS\_OK,  TS\_NOK,  LENTA\_END,  SURPRISE,  };  struct MfstDiagnosis  {  short lenta\_position;  RC\_STEP rc\_step;  short nrule;  short nrule\_chain;  MfstDiagnosis();  MfstDiagnosis(  short plenta\_position,  RC\_STEP prt\_step,  short pnrule,  short pnrule\_chain  );  } diagnosis[MFST\_DIAGN\_NUMBER];  class my\_stack\_MfstState :public stack<MfstState>  {  public:  using stack<MfstState>::c;  };  GRBALPHABET\* lenta;  short lenta\_position;  short nrule;  short nrulechain;  short lenta\_size;  GRB::Greibach grebach;  LT::LexTable lex;  MFSTSTSTACK st;  my\_stack\_MfstState storestate;  Mfst();  Mfst(LT::LexTable& plex, GRB::Greibach pgrebach);  char\* getCSt(char\* buf);  char\* getCLenta(char\* buf, short pos, short n = 25);  char\* getDiagnosis(short n, char\* buf);  bool savestate(Log::LOG log);  bool resetstate(Log::LOG log);  bool push\_chain(GRB::Rule::Chain chain);  RC\_STEP step(Log::LOG log);  bool start(Log::LOG log);  bool savediagnosis(RC\_STEP pprc\_step);  void printrules(Log::LOG log);  struct Deducation  {  short size;  short\* nrules;  short\* nrulechains;  Deducation()  {  size = 0;  nrules = 0;  nrulechains = 0;  };  } deducation;  bool savededucation();  }; |

**Приложение Д**

Начало разбора синтаксическим анализатором исходного кода :

Конец разбора:



** **

**Приложение Е**

|  |
| --- |
|  |

**Приложение Ж**

Результат генерации кода контрольного примера в Ассемблере:

|  |
| --- |
| .586P  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ..\Debug\Library.lib  ExitProcess PROTO :DWORD  EXTRN CONCAT: proc  EXTRN LINELEN: proc  EXTRN RANDOM: proc  EXTRN SQRT: proc  EXTRN OutNumber: proc  EXTRN OutLine: proc  EXTRN OutBool: proc  EXTRN OutNumberLn: proc  EXTRN OutLineLn: proc  EXTRN OutBoolLn: proc  .stack 4096  .const  null\_division BYTE "Exception: деление на ноль", 0  lit1 SWORD 2  lit2 SWORD 8  lit3 BYTE "Смена логина проведена успешно!", 0  lit4 BYTE "Ваш новый логин: ", 0  lit5 BYTE "Результат: ", 0  lit6 SWORD 6  lit7 SWORD 3  lit8 BYTE "Результат вычисления выражения отрицательный? - ", 0  lit9 SWORD -8  lit10 SWORD 5  lit11 SWORD 0  lit12 BYTE "Результат был: ", 0  lit13 BYTE "Ivanov Ivan", 0  lit14 BYTE "Ваш баланс (в $): ", 0  lit15 SWORD 10  lit16 SWORD 100  lit17 BYTE "Квадратный корень из 9 равен 4? - ", 0  lit18 SWORD 9  lit19 SWORD 4  lit20 BYTE "Да", 0  lit21 BYTE "Нет", 0  lit22 BYTE "Длина строки равна 0? - ", 0  lit23 BYTE "abcdef", 0  lit24 BYTE "Hello, ", 0  lit25 BYTE "it's SMI-2023!", 0  .data  buffer BYTE 256 dup(0)  calculate\_i SWORD 0  calculate\_res SWORD 0  program\_resCal SWORD 0  program\_expr SWORD 0  program\_resNeg SWORD 0  program\_login DWORD ?  program\_balance SWORD 0  program\_resSqrt SWORD 0  program\_lineA DWORD ?  program\_lengthA SWORD 0  program\_resA SWORD 0  program\_resConcat DWORD ?  .code  calculate PROC calculate\_a : SWORD, calculate\_b : SWORD  push lit1  pop calculate\_i  push calculate\_a  push calculate\_i  pop ax  pop bx  mul bx  push ax  push calculate\_b  push lit2  pop bx  pop ax  sub ax, bx  push ax  pop bx  pop ax  sub ax, bx  push ax  pop calculate\_res  push calculate\_res  jmp local0  local0:  pop eax  ret  calculate ENDP  yourLogin PROC yourLogin\_login : DWORD  push offset lit3  call OutLineLn  push offset lit4  call OutLine  push yourLogin\_login  call OutLineLn  local1:  pop eax  ret  ret  yourLogin ENDP  program PROC  push offset lit5  call OutLine  push lit6  push lit7  pop dx  pop dx  movsx eax, lit7  push eax  movsx eax, lit6  push eax  call calculate  push ax  pop program\_resCal  movsx eax, program\_resCal  push eax  call OutNumberLn  push offset lit8  call OutLine  push lit9  push lit1  pop bx  pop ax  cmp bx, 0  je nulldiv  cwd  idiv bx  push ax  push lit10  push lit1  pop bx  pop ax  cmp bx, 0  je nulldiv  cwd  idiv bx  push dx  pop bx  pop ax  sub ax, bx  push ax  pop program\_expr  push program\_expr  push lit11  pop bx  pop ax  cmp ax, bx  jle l0  jg l1  l0:  mov ax, 1  push ax  jmp endofexpr0  l1:  mov ax, 0  push ax  endofexpr0:  pop ax  cmp ax, 0  je l2  jne l3  l2:  mov ax, 0  push ax  jmp endofexpr1  l3:  mov ax, 1  push ax  endofexpr1:  pop program\_resNeg  movsx eax, program\_resNeg  push eax  call OutBoolLn  push offset lit12  call OutLine  movsx eax, program\_expr  push eax  call OutNumberLn  push offset lit13  pop program\_login  push program\_login  call yourLogin  push offset lit14  call OutLine  push lit15  push lit16  pop dx  pop dx  movsx eax, lit16  push eax  movsx eax, lit15  push eax  call RANDOM  push ax  push lit10  pop ax  pop bx  add ax, bx  push ax  pop program\_balance  movsx eax, program\_balance  push eax  call OutNumberLn  push offset lit17  call OutLine  push lit18  pop dx  movsx eax, lit18  push eax  call SQRT  push ax  pop program\_resSqrt  mov ax, program\_resSqrt  cmp ax, lit19  je m0  jne m1  m0:  push offset lit20  call OutLineLn  jmp e0  m1:  push offset lit21  call OutLineLn  e0:  push offset lit22  call OutLine  push offset lit23  pop program\_lineA  push program\_lineA  pop dx  push program\_lineA  call LINELEN  push ax  pop program\_lengthA  push program\_lengthA  push lit11  pop bx  pop ax  cmp ax, bx  jne l4  je l5  l4:  mov ax, 1  push ax  jmp endofexpr2  l5:  mov ax, 0  push ax  endofexpr2:  pop ax  cmp ax, 0  je l6  jne l7  l6:  mov ax, 0  push ax  jmp endofexpr3  l7:  mov ax, 1  push ax  endofexpr3:  pop program\_resA  mov ax, program\_resA  cmp ax, 0  jnz m2  jz m3  m2:  push offset lit21  call OutLineLn  jmp e1  m3:  push offset lit20  call OutLineLn  e1:  push offset lit24  push offset lit25  pop dx  pop dx  push offset lit25  push offset lit24  push offset buffer  call CONCAT  push eax  pop program\_resConcat  push program\_resConcat  call OutLineLn  theend:  push 0  call ExitProcess  nulldiv:  push offset null\_division  call OutLineLn  push -1  call ExitProcess  program ENDP  end program |