МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 “Программная инженерия”

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора KNP – 2024»

Выполнил студент Кучерук Николай Петрович

(Ф.И.О.)

Руководитель проекта ст. преп. Наркевич Аделина Сергеевна

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

Заведующий кафедрой к.т.н., доц. Смелов Владимир Владиславович

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

Консультанты ст. преп. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст. преп. Наркевич Аделина Сергеевна.

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

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[**Введение** 5](#_Toc185308664)

[**1 Спецификация языка программирования** 6](#_Toc185308665)

[**1.1 Характеристика языка программирования** 6](#_Toc185308666)

[**1.2 Определение алфавит языка программирования** 6](#_Toc185308667)

[**1.3 Применяемые сепараторы** 6](#_Toc185308668)

[**1.4 Применяемые кодировки** 7](#_Toc185308669)

[**1.5 Типы данных** 7](#_Toc185308670)

[**1.6 Преобразование типов данных** 8](#_Toc185308671)

[**1.7 Идентификаторы** 8](#_Toc185308672)

[**1.8 Литералы** 9](#_Toc185308673)

[**1.9 Область видимости идентификаторов** 9](#_Toc185308674)

[**1.10 Инициализация данных** 10](#_Toc185308675)

[**1.11 Инструкции языка** 10](#_Toc185308676)

[**1.13 Выражения и их вычисления** 11](#_Toc185308677)

[**1.14 Программные конструкции языка** 11](#_Toc185308678)

[**1.15 Область видимости** 12](#_Toc185308679)

[**1.16 Семантические проверки** 12](#_Toc185308680)

[**1.17 Распределение оперативной памяти на этапе выполнения** 13](#_Toc185308681)

[**1.18 Стандартная библиотека и её состав** 13](#_Toc185308682)

[**1.19 Ввод и вывод данных** 14](#_Toc185308683)

[**1.20 Точка входа** 14](#_Toc185308684)

[**1.21 Препроцессор** 14](#_Toc185308685)

[**1.22 Соглашения о вызовах** 14](#_Toc185308686)

[**1.23 Объектный код** 14](#_Toc185308687)

[**1.24 Классификация сообщений транслятора** 15](#_Toc185308688)

[**1.25 Контрольный пример** 15](#_Toc185308689)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 16](#_Toc185308691)

[**2.2 Перечень входных параметров транслятора** 17](#_Toc185308692)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 17](#_Toc185308693)

[**3 Разработка лексического анализатора** 19](#_Toc185308694)

[**3.1 Структура лексического анализатора** 19](#_Toc185308695)

[**3.2 Контроль входных символов** 19](#_Toc185308696)

[**3.3 Удаление избыточных символов** 20](#_Toc185308697)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 20](#_Toc185308698)

[**3.5 Основные структуры данных** 21](#_Toc185308699)

[**3.6 Структура и перечень сообщений лексического анализатора** 22](#_Toc185308700)

[**3.7 Принцип обработки ошибок** 22](#_Toc185308701)

[**3.8 Параметры лексического анализатора** 22](#_Toc185308702)

[**3.9 Алгоритм лексического анализа** 22](#_Toc185308703)

[**3.10 Контрольный пример** 23](#_Toc185308704)

[**4 Разработка синтаксического анализатора** 24](#_Toc185308705)

[**4.1 Структура синтаксического анализатора** 24](#_Toc185308706)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 24](#_Toc185308707)

[**4.3 Построение конечного магазинного автомата** 26](#_Toc185308708)

[**4.4 Основные структуры данных** 27](#_Toc185308709)

[**4.5 Описание алгоритма синтаксического разбора** 27](#_Toc185308710)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 28](#_Toc185308711)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 28](#_Toc185308712)

[**4.8 Принцип обработки ошибок** 28](#_Toc185308713)

[**4.9 Контрольный пример** 29](#_Toc185308714)

[**5 Разработка семантического анализатора** 30](#_Toc185308715)

[**5.1 Структура семантического анализатора** 30](#_Toc185308716)

[**5.2 Функции семантического анализатора** 30](#_Toc185308717)

[**5.3 Структура и перечень сообщений семантического анализатора** 30](#_Toc185308718)

[**5.4 Принцип обработки ошибок** 31](#_Toc185308719)

[**5.5 Контрольный пример** 31](#_Toc185308720)

[**6.1 Выражения, допускаемые языком** 32](#_Toc185308721)

[**6.2 Польская запись** 32](#_Toc185308722)

[**6.3 Программная реализация обработки выражений** 33](#_Toc185308723)

[**6.4 Контрольный пример** 33](#_Toc185308724)

[**7 Генерация кода** 34](#_Toc185308725)

[**7.1 Структура генератора кода** 34](#_Toc185308726)

[**7.2 Представление типов данных в оперативной памяти** 34](#_Toc185308727)

[**7.4 Алгоритм работы генератора кода** 36](#_Toc185308728)

[**7.5 Контрольный пример** 38](#_Toc185308729)

[**8 Тестирование транслятора** 39](#_Toc185308730)

[**8.1 Тестирование фазы проверки на допустимость символов** 39](#_Toc185308731)

[**8.2 Тестирование лексического анализатора** 39](#_Toc185308732)

[**8.3 Тестирование синтаксического анализатора** 40](#_Toc185308733)

[**8.4 Тестирование семантического анализатора** 40](#_Toc185308734)

[**Заключение** 42](#_Toc185308735)

[ПРИЛОЖЕНИЕА 44](#_Toc185308736)

[ПРИЛОЖЕНИЕ Б 46](#_Toc185308737)

[ПРИЛОЖЕНИЕ В 48](#_Toc185308738)

[ПРИЛОЖЕНИЕГ 57](#_Toc185308739)

[ПРИЛОЖЕНИЕД 60](#_Toc185308740)

[ПРИЛОЖЕНИЕ Е 62](#_Toc185308741)

[ПРИЛОЖЕНИЕ Ж 65](#_Toc185308742)

# **Введение**

Задачей данного курсового проекта является разработка транслятора для своего языка программирования: KNP-2024.

Написание транслятора будет осуществляться на языке C++, при этом код на языке KNP-2024 будет транслироваться в язык ассемблера.

Транслятор KNP-2024 состоит из следующих частей:

– семантический анализатор;

– синтаксический анализатор;

– логический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбработка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

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

# **1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования KNP-2024 – это язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным. Язык является транслируемым.

## **Определение алфавит языка программирования**

Совокупность символов, используемых в языке, называется алфавитом языка.

На этапе выполнения могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. Русские символы разрешены только в строковых литералах.

Таблица 1.1 – Алфавит языка программирования RIV-2024

|  |
| --- |
| <строчная буква латинского алфавита>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |
| <прописная буква латинского алфавита>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |
| <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <символ- сепаратор>::= ' '|,|(|)|{|}|;|”|’|=|+|-|\*|:|&| ’|’ |^|[|] |

## **Применяемые сепараторы**

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

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| […] | Квадратные кавычки | Блок кода |
| (…) | Круглые скобки | Приоритет операций, параметры функции |
| “…” | Двойные кавычки | Строковый литерал |
| ‘…’ | Одинарные кавычки | Допускается везде, кроме идентификаторов и ключевых слов |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| Сепаратор | Название | Область применения |
| & |  ^  /  \  +  -  \*  :  % | Знаки «амперсанд», «вертикальная черта», «циркумфлекс», «косая черта», «обратная косая черта», «плюс», «астерикс», «тильда», «процент» | Выражения |
| ==  !  <  > | знаки «больше» и «меньше», двойное «равно», «восклицательный знак» | Выражения в операторе условий и цикла |

## **Применяемые кодировки**

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

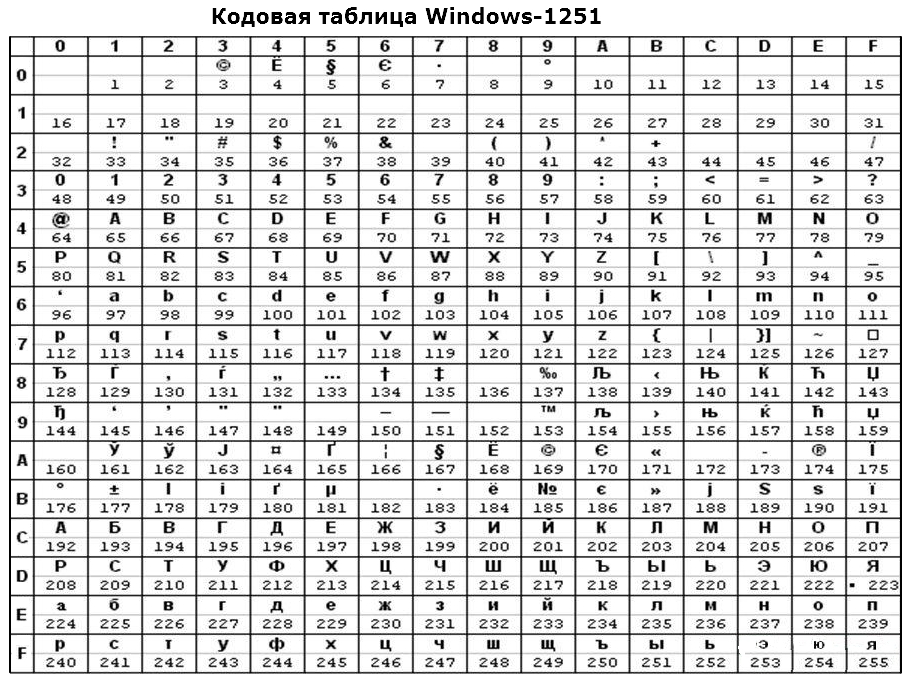


Рисунок 1.1 – Кодовая таблица Windows-1251

## **Типы данных**

В языке KNP-2024 реализованы четыре типа данных: целочисленный (num), символьный(symb), строковый(str) и массив целочисленных(array). Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка KNP-2024

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных num | Фундаментальный тип данных. Используется для работы с целочисленными значениями. В памяти занимает 4 байта.  При попытке инициализации значением больше максимального, инициализируется максимальным. При попытке инициализации значением меньше минимального, инициализируется минимальным.  Максимальное значение: 2147483647. Минимальное значение: -2147483648.  Инициализация по умолчанию: значение 0. |
| Строковый тип данных str | Фундаментальный тип данных. Используется для работы с набором символов, каждый символ в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |
| Символьный тип данных symb | Фундаментальный тип данных. Используется для работы с символом, который в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |
| Массив целых чисел array | Одномерный массив целых чисел. Размерность элементов массива соответствует размерности целочисленного типа данных. Максимальное количество элементов массива – 1000. Инициализация по умолчанию: длина указывается при создании. Значения элементов 0. |

Пользовательские типы данных не поддерживаются.

## **Преобразование типов данных**

Преобразование типов данных реализовано при помощи функций стандартной библиотеки ntc(целочисленный в символьный) и ctn(символьный в целочисленный).

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита и знак «\_» и цифры. Максимальная длина имени идентификатора – 8 символов. Максимальная длина имени идентификатора функции – 11 символов. При вводе идентификатора длиной более разрешенного количества символов, он будет усекаться. Имя идентификатора не может совпадать с именем функции, уже содержащаяся в стандартной библиотеке, если только это функция подключена через оператор extern.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует три типа литералов. Краткое описание литералов языка KNP-2024 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал (десятичный) | [1-9]+[0-9]\* | Литералы могут быть только rvalue. | declare num sum = 15;  15 – десятичный литерал. |
| Целочисленный литерал (шестнадцатеричный) | [0]+[x]+[1-9|a-f|A-F]+[0-9|a-f|A-F]\* | Литералы могут быть только rvalue. | declare num sum = 0xA5;  0xA15 – шестнадцатеричный литерал. |
| Символьный литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/] | Литералы могут быть только rvalue. | declare symb symbol = ‘T’;  T – символьный литерал. |
| Строковый литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/]\* | Строковые литералы, максимальная длина строки 255 символов | write “text”;  text – строковый литерал |

Литералы являются константами и при генерации кода объявляются один раз.

## **Область видимости идентификаторов**

Область видимости «сверху вниз». В языке KNP-2024 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Каждая переменная получает префикс – название функции, в которой она объявлена. Объявление функций стандартной библиотеки можно производить в любом месте кода.

## **Инициализация данных**

При объявлении переменной происходит инициализация с стандартными значениями. Описание способов инициализации переменных языка KNP-2024 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| declare <тип данных> <идентификатор> / <идентификатор>[<значение>]; | Автоматическая инициализация: переменные типа num инициализируются нулём, переменные типа symb – пустым символом. Массиву требуется значение длины для инициализации. | declare num sum;  declare symbol chr;  declare array id[5]; |
| <идентификатор> /<идентификатор>[<значение>] = <значение>; | Присваивание переменной значения. | sum = 9;  chr = ‘D’;  id[3]=12; |

Соответствие типов проверяется на семантическом анализе.

## **Инструкции языка**

Все возможные инструкции языка программирования KNP-2024 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования KNP-2024

|  |  |
| --- | --- |
| Инструкция | Запись на языке KNP-2024 |
| Объявление переменной | declare <тип данных> <идентификатор> / <идентификатор>[<значение>]; |
| Объявление функции | declare <тип данных> action<идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> /<идентификатор>[<значение>]= <значение>/<идентификатор>; |
| Объявление внешней функции | extern <тип данных> action<идентификатор> (<тип данных> <идентификатор>, …); |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | return <идентификатор> / <литерал>; |
| Условная инструкция | if(<условие>){<блок кода>}; |
| Инструкция цикла | until(<условие>){<блок кода>}; |
| Вывод данных | write <идентификатор> / <литерал>; |
| Однострочный комментарий до конца строки | #<любой текст> |

Инструкции (кроме функции входа в программу) требуют закрывающую «;».

* 1. **Операции языка**

Язык программирования KNP-2024 может выполнять операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка программирования KNP-2024

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| ( | Приоритет операций | - | sum = (a + b) \* c; |
| ) |
| + | Суммирование | (num, num) | sum = a + b; |
| - | Вычитание | (num, num) | diff = a – b; |
| \* | Умножение | (num, num) | mul = a \* b; |
| : | Деление | (num, num) | div = a : b; |
| % | Остаток от деления | (num, num) | mod = a % b; |
| / | Сдвиг влево | (num, num) | shiftL = a / b; |
| \ | Сдвиг вправо | (num, num) | shiftR = a \ b; |
| & | Поразрядное «и» | (num, num) | and = a & b; |
| | | Поразрядное «или» | (num, num) | or = a | b; |
| ^ | Поразрядная инверсия | (num) | not = ^ a; |
| = | Присваивание | (num, num)  (symb, symb)  (str, str) | sum = 15;  chr = ‘T’; |
| <,> | Знаки «больше», «меньше» для условной инструкции | (num, num) | until(sum < diff) {…} |
| == | Оператор эквивалентности | (num, num) | if(sum == diff) {…} |
| ! | Оператор неравенства | (num, num) | until(sum ! diff) {…} |

## **Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции, если эта функция уже содержится в стандартной библиотеке. Выражения вычисляются только после оператора присваивания.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования KNP-2024 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка KNP-2024

|  |  |
| --- | --- |
| Конструкция | Запись на языке KNP-2024 |
| Главная функция (точка входа) | main  {  …  return <идентификатор> / <литерал>;  } |
| Функция | declare <тип данных> action<идентификатор> (<тип> <идентификатор>, …)  {… return <идентификатор> / <литерал>; };  }; |
| Цикл | until(a!8){ …}; |
| Условный оператор | if(5>4){…}; |

Программные конструкции языка KNP-2024 представляют собой базовый функционал для выполнения различных операций, что делает возможным решать задачи различного уровня.

## **Область видимости**

В языке KNP-2024 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **Семантические проверки**

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования. Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторно объявляться в пределах одной функции. |
| 2 | Тип возвращаемого значения должен совпадать с типом функции при её объявлении или подключении |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении или подключении |
| 4 | В функцию должно быть передано то число параметров, сколько ожидается |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 6 | Типы данных операндов выражения должны быть одинаковыми |
| 7 | Тип данных str не может быть аргументом условной конструкции |
| 8 | Функции не должны подключаться дважды в пределах одной программы |
| 9 | К элементу массива можно обращаться только целочисленным типом |

Если семантическая проверка не проходит, то в лог журнал записывается соответствующая ошибка.

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

Все переменные размещаются в стеке.

Стек — это структура данных, организованная по принципу LIFO (последний вошел - первый вышел). Данное решение идеально подходит для хранения данных, к которым вскоре предстоит обратиться (легко извлекаются с вершины стека).

## **Стандартная библиотека и её состав**

Стандартная библиотека KNP-2024 написана на языке программирования C++.

Для использования функций стандартной библиотеки, нужно использовать ключевое слова extern, далее работа с ними производится как с пользовательскими функциями. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| num ctn(symb) | num | Конвертирует передаваемый символ в цифру. Если символ не является цифрой, будет возвращен 0. |
| symb ntc(num) | symb | Конвертирует последнюю цифру передаваемого числа в символ. |
| num GetSize(str) | num | Возвращает длину переданной строки |
| str Copy(str) | str | Возвращает копию переданной строки |
| num Compare(str,str) | num | Сравнивает переданные строки. Возвращает 0 – если равны, 1 – если первая строка больше, 2 – если вторая строка больше. |
| num Random(num) | num | Генерирует случайное число в диапазоне 0- переданное значение |

Так же в библиотеке присутствуют приватные функции. Их описание представлено в таблице 1.10.

Таблица 1.10 – Приватные функции стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| void outputnum (unsigned num a) | – | Выводит число на экран Вызывается оператором write |
| void outputsymb (symb a) | – | Выводит символ на экран Вызывается оператором write |
| void outputstr (void\* in) | – | Выводит строку на экран Вызывается оператором write |

Приватные функции не могут быть вызваны явно и не требуют предварительного пользовательского подключения. Они вызываются специальными операторами языка.

## **Ввод и вывод данных**

В языке KNP-2024 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрен оператор write, который базируется на приватных функциях стандартной библиотеки.

## **Точка входа**

В языке KNP-2024 каждая программа должна содержать главную функцию main, т. е. точку входа, с которой начнется последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования KNP-2024 не предусмотрен.

## **Соглашения о вызовах**

В языке 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.

## **Объектный код**

KNP-2024 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке KNP-2024 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-109 | Ошибки параметров |
| 110-119 | Ошибки открытия и чтения файлов |
| 120-139 | Ошибки лексического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 700-900 | Ошибки семантического анализа |

Компилятор может обрабатывать до 1000 различных ошибок.

## **Контрольный пример**

# Код контрольного примера представлен в Приложении А.**2 Структура транслятора**

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

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

Основные компоненты транслятора (представлены на рисунке 2.1):

- лексический анализатор

- синтаксический анализатор

- семантический анализатор

- генератор кода



Рисунок 2.1 - Структура транслятора

Назначение лексического анализатора: текст исходной программы считывается посимвольно слева направо и группируется в отдельные лексемы, представляющие собой последовательности символов с определенным значением. В результате работы формируются таблица лексем и таблица идентификаторов, а также генерируются ошибки, связанные с лексикой данного языка. (Принцип работы описан в главе «Разработка лексического анализатора»)

Назначение синтаксического анализатора: сопоставляет линейные последовательности лексем из таблицы лексем с формальной грамматикой языка. Формирует полное дерево разбора и генерирует ошибки, если дерево построено не было. (Принцип работы описан в главе «Разработка синтаксического анализатора»)

Назначение семантического анализатора: проверяет наличие семантических ошибок в исходной программе и накапливает информацию для генератора кода. (Принцип работы описан в главе «Разработка семантического анализатора»)

Назначение генератора кода: генерирует ассемблерный код. При этом каждая инструкция транслируется в последовательность машинных инструкций, выполняющих ту же самую работу, а переменные назначаются регистрам. (Принцип работы описан в главе «Генерация кода»)

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка KNP-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке KNP-2024. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке KNP-2024. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат трансляции кода на язык assembler | <имя\_файла>.asm |
| m | Вывод дерева разбора синтаксического анализатора. | – |
| l | Вывод таблицы лексем | – |
| i | Вывод таблицы идентификаторов | – |

Таблицы лексем и дерево разбора синтаксического анализатора выводятся в консоль и записываются в лог файл.

## **2.3 Перечень протоколов, формируемых транслятором и их содержимое**

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

Таблица 2.2 – Протоколы, формируемые транслятором языка KNP-2024

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Файл содержит в себе краткую информацию об исходном коде на языке KNP-2024. В этот файл выводится протокол работы анализаторов, а так же различные ошибки |
| “\*.asm” | Содержит сгенерированный код на языке Ассемблера. |

В log файл выводятся все ошибки, за исключением тех, что связаны с открытием файла log или считывания параметров.

# **3 Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке KNP-2024. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 – Структура лексического анализатора

Результатом работы лексического анализатора являются заполненные таблица лексем и таблица идентификаторов.

## **3.2 Контроль входных символов**

Исходный код на языке программирования KNP-2024 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена в приложении Б.

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – пробельный символ, С – символ одинарной кавычки, L – символ-разделитель, D – символ двойной кавычки, O – символ начала комментария, N – символ новой строки.

## **3.3 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования KNP-2024 разбивается на токены, которые записываются в очередь.

## **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
| Тип данных | Целочисленный беззнаковый тип данных | num | t |
| Строковый тип данных | str | t |
| Символьный тип данных | symb | t |
| Массив | array | a |
| Лексема | Объявление переменной | declare | d |
| Подключение функции библиотеки | extern | e |
| Оператор вывода | write | p |
| Объявление функции | action | f |
| Возврат значения из функции | return | r |
| Инструкция цикла | until | u |
| Инструкция Условного оператора | if | o |
| Блок инструкции цикла | [ | [ |
| ] | ] |
| Блок функции | { | { |
| } | } |
| Изменение приоритетности в выражении и отделение параметров функций | ( | ( |
| ) | ) |
| Сепараторы | ; | ; |
| , | , |
| Оператор присваивания | = | v |
| Условный оператор | < | b |
| > | b |
| & | b |
| ^ | b |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
| Оператор | Знаки арифметических операций | + | v |
| - | v |
| \* | v |
| / | v |
| \ | v |
| : | v |
| % | v |
| Знаки побитовых операций | & | v |
| | | v |
| ^ | v |
| Идентификатор |  | [a-z|A-Z]+  [a-z|A-Z|0-9]\* | i |
| Литерал | Целочисленный литерал | [1-9]+[0-9]\* | l |
| Символьный литерал | [a-z|A-Z|0-9]\*  кроме ‘ | l |
| Строковый литерал | [a-z|A-Z|0-9]\* кроме ” | l |
| Точка входа |  | main | m |

Каждому выражению соответствует детерминированный конечный автомат, то есть автомат с конечным состоянием, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся фраза и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Пример реализации таблицы лексем представлен в приложении В.

Также в приложении В находятся конечные автоматы, соответствующие лексемам языка KNP-2024.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка KNP-2024, используемых для хранения, представлены в приложении В. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, номер столбца в исходном коде, индекс таблицы идентификаторов (если нет соответствующего идентификатора, то индекс равен -1), а также специальное поле, в котором хранится значение лексемы. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, его значение, а также бинарное поле для определения внешний ли идентификатор.

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

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-125. Также сам текст ошибки содержит в себе префикс [LA]. Перечень сообщений лексического анализатора представлен на рисунке 3.3.

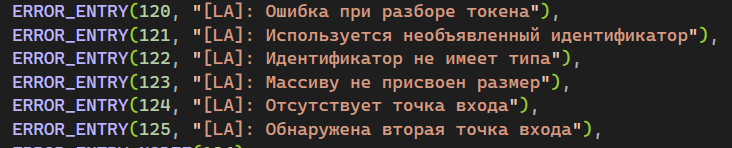


Рисунок 3.2 – Перечень ошибок лексического анализатора

## **3.7 Принцип обработки ошибок**

Все ошибки являются критическими и приводят к прекращению работы транслятора и выводу диагностического сообщения в log-файл.

## **3.8 Параметры лексического анализатора**

Входные параметры используются для вывода результата работы лексического анализатора. Они передаются аргументами через командную строку и рассмотрены в таблице 2.1

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

В случае, если токен является идентификатором, перед его именем записывается название функции, в которой он объявлен и после этого он заносится в таблицу идентификаторов.

В случае, если токен является идентификатором функции, название функции в которой он объявлен не записывается.

В случае, если токен является литералом, то он заносится в таблицу идентификаторов в виде abi, где a – имя функции, где объявлен литерал, b – “&LIT ”, c –количество определённых литералов+1.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных или вид идентификатора, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных или вид идентификатора, которому он соответствует.

В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и видом идентификатора, и именем вида “ab”, где a – имя функции, где объявлен идентификатор, b – имя идентификатора.

Пример. Регулярное выражение для ключевого слова declare: «declare».

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S7  – конечное состояние автомата.

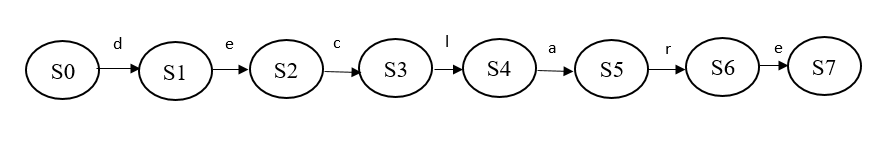


Рисунок 3.3 – Граф переходов для цепочки «declare»

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении В.

# **4 Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций KNP-2024. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора KNP-2024

Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией – дерево разбора.

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

В синтаксическом анализаторе транслятора языка KNP-2024 используется контекстно-свободная грамматика [2], где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка KNP-2024 представлена в приложении Г.

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

NS – нетерминальные символы, представленные несколькими заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил и описание нетерминальных символов KNP-2024

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | m{NrE;};  m{rE;};  dtfi(F){NrE;};S  dtfi(F){rE;};S  dtfi(){NrE;};S  dtfi(){rE;};S  m{rE;};S | Порождает правила, описывающее общую структуру программы |
| N | dti;  dai[l];  dai[l];N  f(F);  i(F);N  ivE;  ivE;N  i[i]vE;  i[i]vE;N  i[l]vE;  i[l]vE;N  etfi(F);  etfi(F);N  o(B){N}N  o(B){N}  dti;N  ivE;N  etfi();N  etfi();  pi;  pl;  pi;N  pl;N  pi[l];  pi[l];N  pi[i];  pi[i];N  u(B){N};  u(B){N};N | Порождает правила, описывающие конструкции языка |
| E | i  vi | Порождает правила, описывающие выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| E | l  (E)  i(W)  iM  lM  i()  i(W)M  (E)M  i[l]M  i[i]M  i[i]  i[l] | Порождает правила, описывающие выражения |
| F | ti  ti,F  i  i,F  i  l,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | i  l  i,W  l,W | Порождает правила, описывающие принимаемые параметры функции |
| B | ibi  ibl  lbi  lbl | Порождает правила, описывающие условное выражение в операторе цикла |
| M | v  vE  v(E)  v(E)M  vEM | Порождает правила, описывающие знаки арифметических операций |

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал.

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку [1], описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении Г.

Таблица 4.2 – Описание компонентов магазинного автомата

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

Для вывода результата работы синтаксического анализатора нужно использовать флаг m.

## **4.4 Основные структуры данных**

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

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

1. в магазин записывается стартовый символ;
2. на основе полученных ранее таблиц формируется входная лента;
3. запускается автомат;
4. выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. если в магазине встретился нетерминал, переходим к пункту 4;
7. если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно и формируется дерево разбора. Иначе генерируется исключение.

## **4.6 Структура и перечень сообщений синтаксического анализатора**

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 600-609. Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

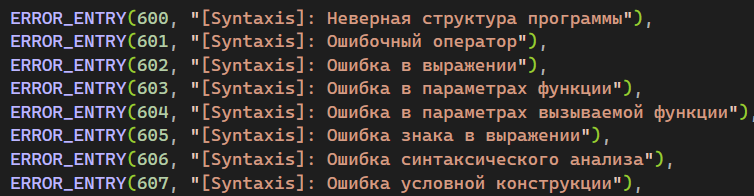


Рисунок 4.1 – Перечень сообщений синтаксического анализатора

Текст синтаксической ошибки содержит в себе префикс [Syntaxis].

## **4.7 Параметры синтаксического анализатора и режимы его работы**

Для вывода результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в таблице 2.1.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. В случае ошибки выводится соответствующее сообщение в журнал лога и компилятор прекращает работу.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке KNP-2024 представлен в приложении Д. Дерево разбора исходного кода также представлено в приложении Д.

# **5 Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки). Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 – Структура семантического анализатора

Функции семантического анализатора частично реализованы в лексическом анализаторе.

## **5.2 Функции семантического анализатора**

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

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

Все ошибка семантического анализатора имеют идентификатор свыше 700. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

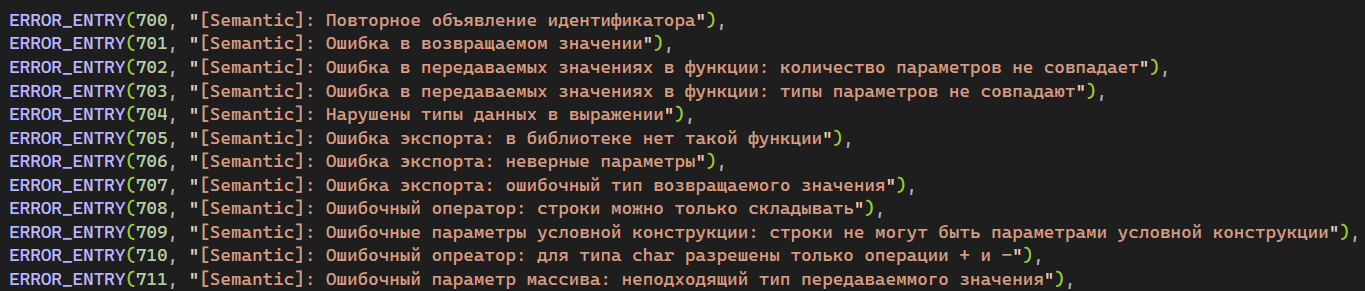


Рисунок 5.2 – Перечень сообщений семантического анализатора

Текст семантической ошибки содержит в себе префикс [Semantic].

## **5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор завершит свою работу c

Записью информации об ошибке в лог файл.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

**6 Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке KNP-2024 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические и побитовые операции, такие как +, -, \*, ^, :, /, \,&,|,^ и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке KNP-2024

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 1 |
| ) | 1 |
| + | 2 |
| - | 2 |
| \* | 3 |
| : | 3 |
| % | 3 |
| / | 3 |
| \ | 3 |
| & | 4 |
| | | 4 |
| ^ | 4 |

## **6.2 Польская запись**

Выражения в языке KNP-2024 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

– также, если идентификатор является именем функции, то он заменяется на спецсимвол «@».

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| a-b/5\*(c+4) |  |  |
| -b\*5/(c-2) | a |  |
| b\*5/(c-2) | a | - |
| \*5/(c-2) | ab | - |
| 5/(c-2) | ab | -/ |
| /(c-2) | ab5 | -/ |
| (c-2) | ab5/ | -/ |
| c-2) | ab5/ | -\*( |
| -2) | ab5/c | -\*( |
| 2) | ab5/c | -\*(+ |
| ) | ab5/c2 | -\*(+ |
|  | ab5/c2+ | -\* |
|  | ab5/c2+\* | - |
|  | ab5/c2+\*- |  |

Как результат успешного разбора, мы получаем пустой стек и заполненную результирующую строку выражения.

## **6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Е.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

# **7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы KNP-2024 в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 – Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const [4][5]. Соответствия между типами данных идентификаторов на языке KNP-2024 и на языке ассемблера приведены в таблице 7.1.

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

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KNP-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| symb | BYTE | Хранит символьный тип данных. |
| str | DWORD | Хранит указатель на начало строки. |
| num | SDWORD | Хранит целочисленный тип данных. |
| Лексема | BYTE  SDWORD  DWORD | Литералы: символьные,  целочисленные, строковые |

Идентификаторы языка KNP-2024 размещены в сегменте данных (.data). Литералы – в сегменте констант (.const).

**7.3 Статическая библиотека**

Статическая библиотека реализована на языке программирования C++. Её код находится в проекте KNP-2024LIB, в свойствах которого был выбран пункт «статическая библиотека .lib». Подключение библиотеки в языке ассемблера происходит с помощью директивы includelib на этапе генерации кода. Далее объявляются имена функций из библиотеки. Вышеописанное проиллюстрировано в листинге 7.1.

|  |
| --- |
| void Head(std::ofstream\* stream, LEX::LEX t) {  \*stream << ".586\n";  \*stream << "\t.model flat, stdcall\n";  \*stream << "\tincludelib libucrt.lib\n";  \*stream << "\tincludelib kernel32.lib\n";  \*stream << "\tincludelib ../StaticLibraries/KNP-2024LIB.lib\n";  \*stream << "\tExitProcess PROTO :DWORD\n\n";  for (int i = 0; i < t.idtable.size; i++)  {  if (t.idtable.table[i].idtype == IT::F)  { //Если библиотечная  if (t.idtable.table[i].isExternal == true)  {  \*stream << "\n\t" << t.idtable.table[i].id << " PROTO";  int pos = 1;  bool commaFlag = false;  while (true)  {  if (t.lextable.table[t.idtable.table[i].idxfirstLE + pos].lexema == LEX\_ID  &&  t.idtable.table[t.lextable.table[t.idtable.table[i].idxfirstLE + pos].idxTI].idtype == IT::P)  {  if (commaFlag)  {  \*stream << ',';  }  commaFlag = true;  switch (t.idtable.table[t.lextable.table[t.idtable.table[i].idxfirstLE + pos].idxTI].iddatatype)  {  case IT::INT: {  \*stream << " :SDWORD ";  break;  }  case IT::CHR: {  \*stream << " :BYTE";  break;  }  case IT::STR: {  \*stream << " :DWORD";  break;  }  }  }  if (t.lextable.table[t.idtable.table[i].idxfirstLE + pos].lexema == LEX\_RIGHTHESIS)  break;  pos++;  }  }  }  }  \*stream << "\noutputuint PROTO :SDWORD";  \*stream << "\noutputchar PROTO :BYTE";  \*stream << "\noutputstr PROTO :DWORD\n";  \*stream << "\n.stack 4096\n";  } |

Листинг 7.1 – Фрагмент функции генерации кода

## **7.4 Алгоритм работы генератора кода**

Алгоритм генерации кода выглядит следующим образом:

1. Генерирует заголовочную информацию (Лист. 7.2): модель памяти, подключение библиотек, прототипы внешних функций, размер стека.

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../StaticLibraries/KNP-2024LIB.lib  ExitProcess PROTO :DWORD  outputuint PROTO :SDWORD  outputchar PROTO :BYTE  outputstr PROTO :DWORD  .stack 4096 |

Листинг 7.2 –Заголовочная информация

1. Проходит полностью таблицу идентификаторов и заполняет поле .const литералами (Лист. 7.3).

|  |
| --- |
| .const  divideOnZeroExeption BYTE "Попытка деления на ноль.", 0  main$LIT2 SDWORD 5 ;INT  main$LIT3 BYTE "If works", 0 ;STR  main$LIT5 BYTE "If doesn't work", 0 ;STR  main$LIT6 SDWORD 10 ;INT  main$LIT7 SDWORD 0 ;INT  main$LIT9 SDWORD 1 ;INT  main$LIT10 BYTE "element", 0 ;STR  main$LIT13 SDWORD 6 ;INT  main$LIT15 SDWORD 4 ;INT  main$LIT16 SDWORD 69 ;INT  main$LIT17 BYTE "Random:", 0 ;STR  main$LIT18 SDWORD 50 ;INT  main$LIT20 BYTE "shift!!!!", 0 ;STR  main$LIT21 SDWORD 12 ;INT  main$LIT22 SDWORD 2 ;INT  main$LIT23 BYTE "Hello, World!", 0 ;STR  main$LIT24 BYTE "Длина:", 0 ;STR  main$LIT26 BYTE "Строки одинаковы", 0 ;STR  main$LIT28 BYTE "Первая строка больше", 0 ;STR  main$LIT30 BYTE "Вторая строка больше", 0 ;STR  main$LIT31 BYTE "operations", 0 ;STR  main$LIT34 SDWORD 3 ;INT  main$LIT35 SDWORD 0 ;INT |

Листинг 7.3 – Пример заполнения поля .const

1. Проходим таблицу идентификаторов и объявляем переменные в поле .data. (Лист. 7.4).

|  |
| --- |
| .data  sumresult SDWORD 0 ;INT  mainnumber SDWORD 0 ;INT  mainmassiv SDWORD 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ;ARR  mainline DWORD 0 ;STR  maini SDWORD 0 ;INT  maingreet DWORD 0 ;STR  maincomp SDWORD 0 ;INT  maingreet2 DWORD 0 ;STR  maina SDWORD 0 ;INT  mainb SDWORD 0 ;INT  mainc SDWORD 0 ;INT  mainres SDWORD 0 ;INT |

Листинг 7.4 – Пример заполнения поля .data

1. Генерируем сегмент данных .code (Лист. 7.5). Сперва проходим по таблице идентификаторов и ищем функции. Объявляем их и генерируем код, содержащийся в функциях. Так же перед именем функции дописываем знак «$», чтобы исключить совпадение имени функции с ключевым словом ассемблера. При генерации кода, при встрече оператора присваивания, описываем вычисление выражения. Описание алгоритма преобразования выражений представлено в пункте 7.3.

|  |
| --- |
| .code  $sum PROC uses ebx ecx edi esi , sumfirst: SDWORD , sumsecond: SDWORD  ; expression #3 :iviiv  push sumfirst  push sumsecond  pop ebx  pop eax  add eax, ebx  push eax  pop sumresult  mov eax, sumresult  ret  $sum ENDP  main PROC  mov esi,0 ;для работы с массивами  ; expression #17 :ivl  push main$LIT1  pop mainnumber  If94Start:  mov eax, mainnumber  mov ebx, main$LIT2  cmp eax, ebx  je If94End  push offset main$LIT3  CALL outputstr  If94End: |

Листинг 7.5 – Пример заполнения поля .code

После генерации всех пользовательских функций, генерируется функция начала программы main в функции main по такому же принципу.

## **7.5 Контрольный пример**

Генерируемый код записывается в файл заданный параметром “-out”. Сгенерированный код можно находится в приложении Ж.

# **8 Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В языке KNP-2024 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{в} | Ошибка 111: [IN]: Недопустимый символ в исходном файле (-in), строка 1, столбец 4 |

Запрещённые символы можно посмотреть в приложении Б.

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{  declare num 1s;  } | Ошибка 120: [LA]: Ошибка при разборе токена, строка 2, столбец 11 |
| main{  s = 5;  } | Ошибка 121: [LA]: Используется необъявленный идентификатор, строка 2, столбец 1 |
| declare num actiona(){  declare q;  } | Ошибка 122: [LA]: Идентификатор не имеет типа, строка 2, столбец 6 |
| declare array arr; |  |
| declare num actiona(){  declare num q;  } | Ошибка 124: [LA]: Отсутствует точка входа |
| main(){  declare symb a;  }  main(){  declare symb z;  } | Ошибка 125: [LA]: Обнаружена вторая точка входа, строка 4, столбец 1 |

Ошибка лексического анализатора приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{  declare symb a  } | Ошибка 609: [Syntaxis]: Обнаружена синтаксическая ошибка(смотри журнал Log) |

Ошибка синтаксического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{  declare symb a;  declare str a;  } | Ошибка 700: [Semantic]: Повторное объявление идентификатора, строка 3, столбец 10 |
| declare num action foo(num q){  declare str a;  return a;  };  main{  declare num a;  a = foo(3);  return 0;  }; | Ошибка 701: [Semantic]: Ошибка в возвращаемом значении, строка 3, столбец 6 |

Продолжение таблицы 8.4

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| declare num action foo(num a, num b){  return 5;  };  main{  declare num a;  a = foo(a);  return a;  }; | Ошибка 702: [Semantic]: Ошибка в передаваемых значениях в функции: количество параметров не совпадает, строка 6, столбец 8 |
| declare num action foo(str a){  return 5;  };  main{  declare num a;  a = foo(a);  return a;  }; | Ошибка 703: [Semantic]: Ошибка в передаваемых значениях в функции: типы параметров не совпадают, строка 6, столбец 7 |
| declare num action foo(num q){  declare num a;  return a;  };  main{  declare str a;  a = foo(3);  return 0;  }; | Ошибка 704: [Semantic]: Нарушены типы данных в выражении, строка 7, столбец 5 |
| main{  extern num action foo(num max);  declare num a;  a = f(a);  return a;  }; | Ошибка 705: [Semantic]: Ошибка экспорта: в библиотеке нет такой функции, строка 2, столбец 18 |

Ошибка семантического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

# **Заключение**

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования KNP-2024 на язык ассемблера.

Язык KNP-2024 поддерживает 4 типа данных: целочисленный (num), строковый (str), символьный(symb), массив целочисленных(array).

Для целочисленного типа реализована обработка 10 арифметических действий, скобок, обозначающих приоритет операций.

На этапе семантического анализа производится проверка соответствия исходного кода спецификации по 11 правилам.

Реализованы 6 публичные и 3 приватные функции стандартной библиотеки.

1. Карпов Ю. Теория и технология программирования. Основы построения трансляторов, 2005. – 272с.
2. Введение в теорию трансляторов [Электронный ресурс]. – Режим доступа: http://bourabai.ru/tpoi/compilers.htm. – Дата доступа: 15.11.2022.
3. Википедия: Обратная польская запись [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Reverse\_Polish\_notation. – Дата доступа: 20.11.2022.
4. MASM для x86 [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/cpp/assembler/masm/masm-for-x64-ml64-exe?view=msvc-160>. – Дата доступа: 29.11.2022.
5. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

## ПРИЛОЖЕНИЕА

|  |
| --- |
| declare num action sum(num first,num second){  declare num result;  result = first+second;  return result;  };  main  {  extern num action Random(num max);  extern num action Compare(str str1,str str2);  extern num action GetSize(str string);  extern str action Copy(str string);  extern num action ctn(symb char);  extern symb action ntc(num numb);  declare num number; #comment    number = -11;  if(number ! 5){  write "If works";  };  if(number == 5){  write "If doesn't work";  };      declare array massiv[10];  declare str line;    write line;  declare num i;  i = 0;  until( i ! 10){  massiv[i] = i;  write massiv[i];  i = i + 1;  };  write "element";  write massiv[1];  number=massiv[5]+massiv[6];  write number;  number = sum(5,4);  write number;  number = 0x45;  write number;  line = "Random:";  write line;  number = Random(50);  write number;  number = Random(10);  write number;  write "shift!!!!";  number = 12;  write number;  number = number / 2;  write number;  declare str greet;  greet="Hello, World!";  write greet;  write "Длина:";  declare num comp;  comp = GetSize(greet);  write comp;    write greet;  declare str greet2;  greet2 = Copy(greet);  write greet2;  comp = Compare(greet,greet2);  if(comp==0){  write "Строки одинаковы";  };  if(comp==1){  write "Первая строка больше";  };  if(comp==2){  write "Вторая строка больше";  };  write "operations";  declare num a;  declare num b;  declare num c;  a=5;  b=4;  c=3;  declare num res;  res=(a+c)\*b;  write res;  res=a/b;  write res;  res= ^b;  write res;  res=a|b;  write res;    return 0;  } |

Рисунок 1 – Контрольный пример

## ПРИЛОЖЕНИЕ Б

|  |  |
| --- | --- |
| #define IN\_CODE\_TABLE {\  IN::T, /\*\*/, IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\* | \*/ IN::F, /\*\*/ IN::S, /\*tabulation\*/ IN::N, /\*New line\*/, \  IN::F, /\* \*/ IN::F, /\*\*/ IN::T, /\*carriage return\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*left-arrow\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*‑\*/, \  IN::F, /\*­\*/ IN::S, /\*Space\*/ IN::L, /\*!\*/ IN::D, /\*"\*/ IN::I, /\*#\*/, \  IN::F, /\*$\*/ IN::L, /\*%\*/ IN::L, /\*&\*/ IN::C, /\*'\*/ IN::L, /\*(\*/ IN::L, /\*)\*/, \  IN::L, /\*\*\*/ IN::L, /\*+\*/ IN::L, /\*,\*/ IN::L, /\*-\*/ IN::F, /\*.\*/, \  IN::L, /\*/\*/ IN::T, /\*0\*/ IN::T, /\*1\*/ IN::T, /\*2\*/ IN::T, /\*3\*/, \  IN::T, /\*4\*/ IN::T, /\*5\*/ IN::T, /\*6\*/ IN::T, /\*7\*/ IN::T, /\*8\*/, \  IN::T, /\*9\*/ IN::L, /\*:\*/ IN::L, /\*;\*/ IN::L, /\*<\*/ IN::L, /\*=\*/ \  IN::L, /\*>\*/ IN::F, /\*?\*/ IN::F, /\*@\*/ IN::T, /\*A\*/ IN::T, /\*B\*/, \  IN::T, /\*C\*/ IN::T, /\*D\*/ IN::T, /\*E\*/ IN::T, /\*F\*/ IN::T, /\*G\*/, \  IN::T, /\*H\*/ IN::T, /\*I\*/ IN::T, /\*J\*/ IN::T, /\*K\*/ IN::T, /\*L\*/, \  IN::T, /\*M\*/ IN::T, /\*N\*/ IN::T, /\*O\*/ IN::T, /\*P\*/ IN::T, /\*Q\*/, \  IN::T, /\*R\*/ IN::T, /\*S\*/ IN::T, /\*T\*/ IN::T, /\*U\*/ IN::T, /\*V\*/, \  IN::T, /\*W\*/ IN::T, /\*X\*/ IN::T, /\*Y\*/ IN::T, /\*Z\*/ IN::L, /\*[\*/, \  IN::L, /\*\\*/ IN::L, /\*]\*/ IN::L, /\*^\*/ IN::T, /\*\_\*/ IN::F, /\*`\*/, \  IN::T, /\*a\*/ IN::T, /\*b\*/ IN::T, /\*c\*/ IN::T, /\*d\*/ IN::T, /\*e\*/, \  IN::T, /\*f\*/ IN::T, /\*g\*/ IN::T, /\*h\*/ IN::T, /\*i\*/ IN::T, /\*j\*/, \  IN::T, /\*k\*/ IN::T, /\*l\*/ IN::T, /\*m\*/ IN::T, /\*n\*/ IN::T, /\*o\*/, \  IN::T, /\*p\*/ IN::T, /\*q\*/ IN::T, /\*r\*/ IN::T, /\*s\*/ IN::T, /\*t\*/, \  IN::T, /\*u\*/ IN::T, /\*v\*/ IN::T, /\*w\*/ IN::T, /\*x\*/ IN::T, /\*y\*/, \  IN::T, /\*z\*/ IN::L, /\*{\*/ IN::L, /\*|\*/ IN::L, /\*}\*/ IN::F, /\*~\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*¤\*/ IN::F, /\*\*/ IN::F, /\*¦\*/ IN::F, /\*§\*/, \  IN::F, /\*\*/ IN::F, /\*©\*/ IN::F, /\*\*/ IN::F, /\*«\*/ IN::F, /\*¬\*/, \  IN::F, /\*­\*/ IN::F, /\*®\*/ IN::F, /\*\*/ IN::F, /\*°\*/ IN::F, /\*±\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*µ\*/ IN::F, /\*¶\*/, \  IN::F, /\*·\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*»\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/ IN::F, /\*\*/, \  } |

Рисунок 1 – Таблица входных символов

## ПРИЛОЖЕНИЕ В

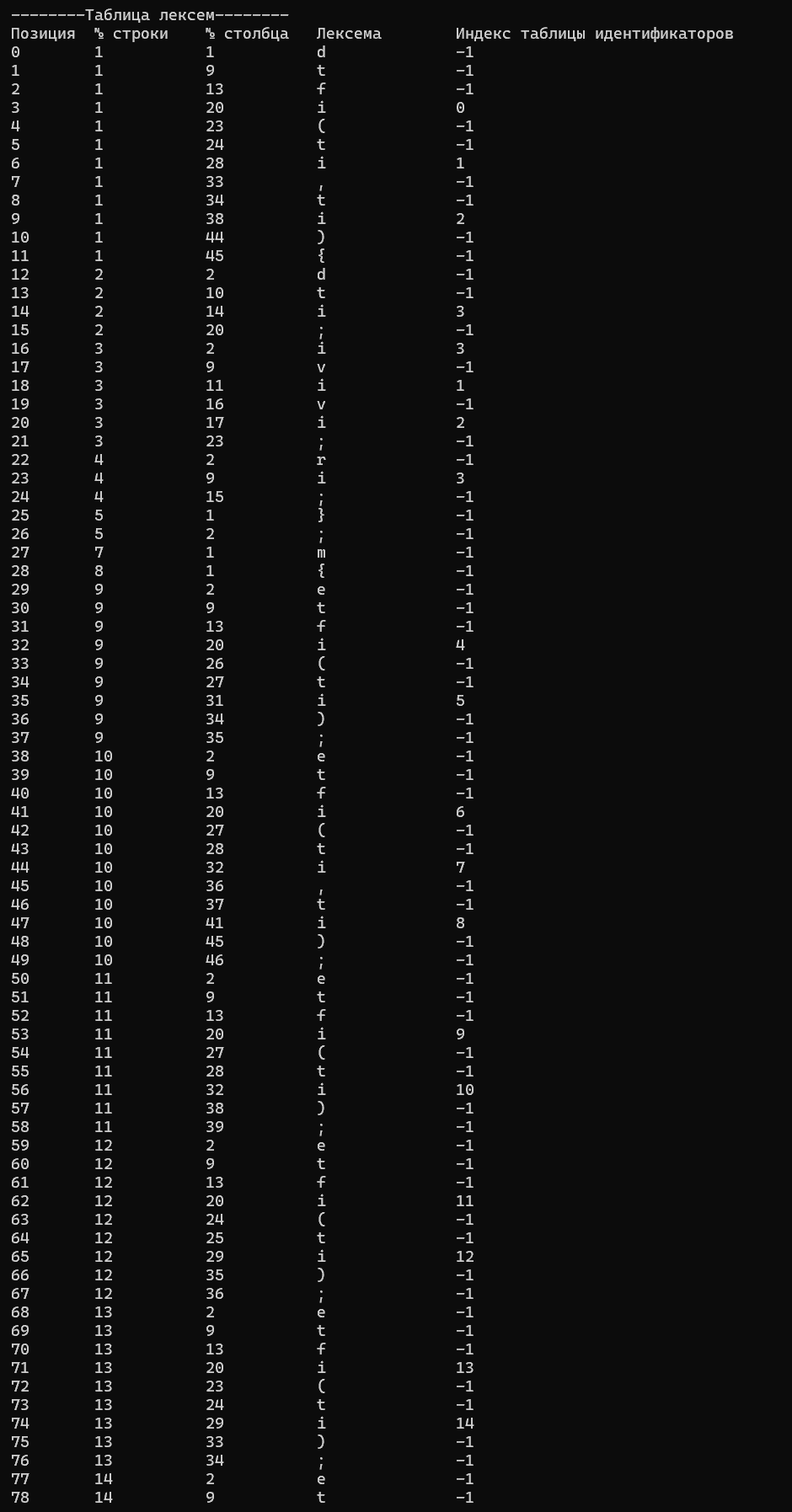


Рисунок 1– Таблица лексем

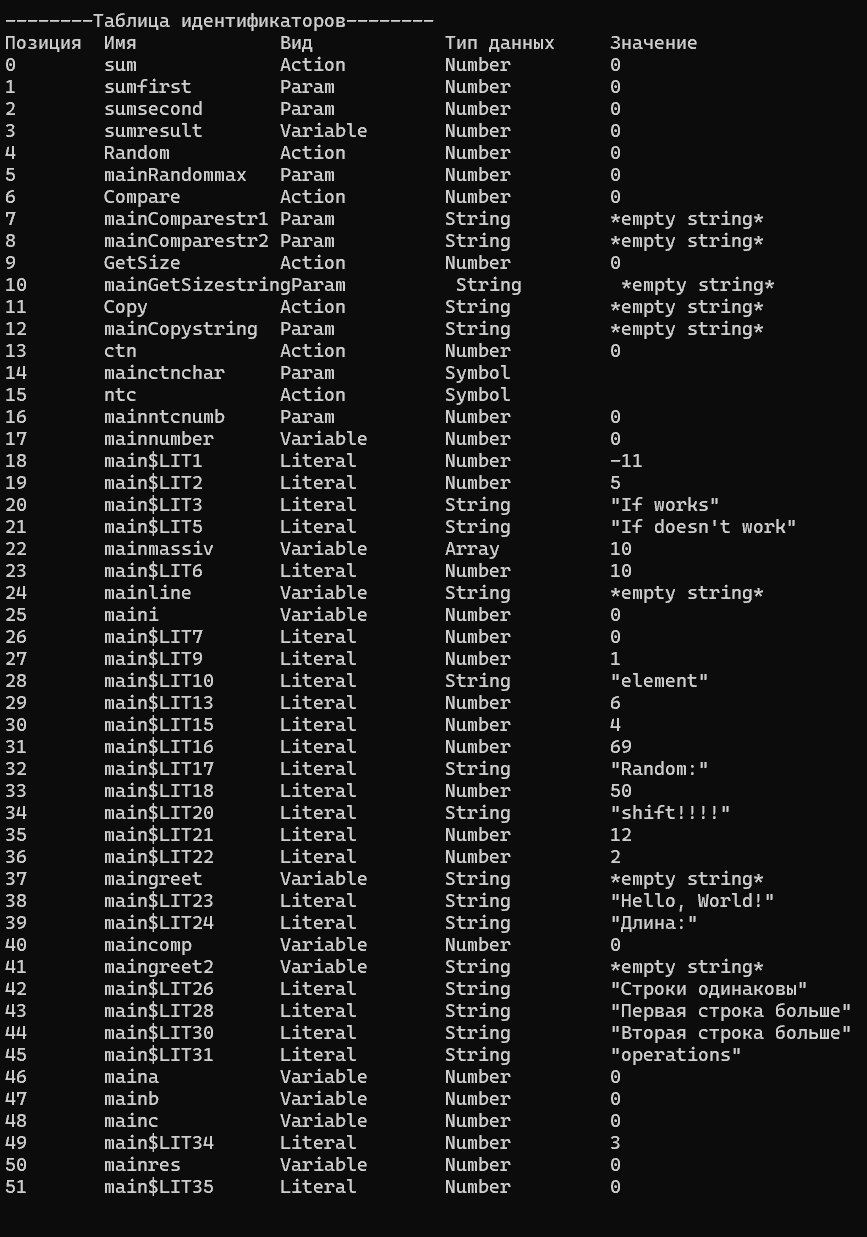
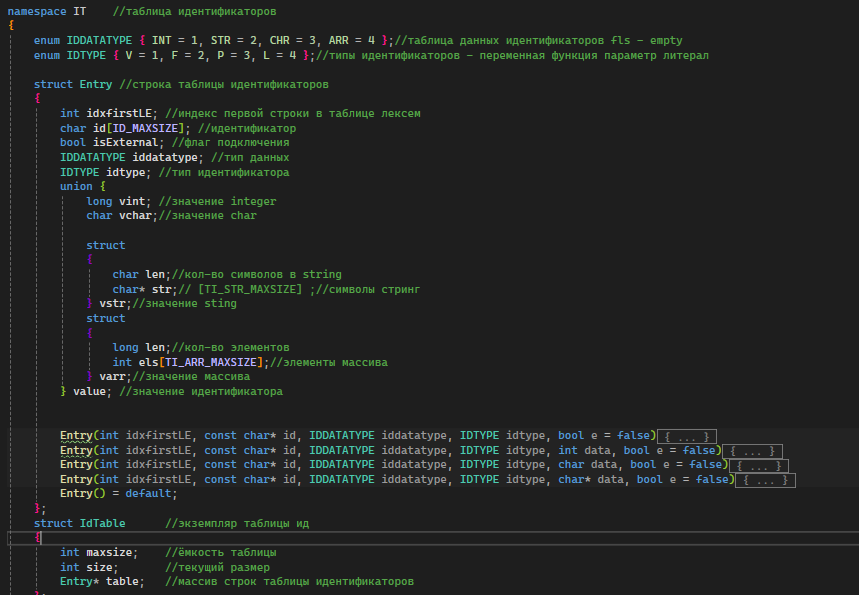


Рисунок 2 – Таблица идентификаторов

|  |
| --- |
| FST l\_num(  str,  4,  NODE(1, RELATION('n', 1)),  NODE(1, RELATION('u', 2)),  NODE(1, RELATION('m', 3)),  NODE()  );  FST l\_array(  str,  6,  NODE(1, RELATION('a', 1)),  NODE(1, RELATION('r', 2)),  NODE(1, RELATION('r', 3)),  NODE(1, RELATION('a', 4)),  NODE(1, RELATION('y', 5)),  NODE()  );  FST l\_until(  str,  6,  NODE(1, RELATION('u', 1)),  NODE(1, RELATION('n', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('i', 4)),  NODE(1, RELATION('l', 5)),  NODE()  );  FST l\_if(  str,  3,  NODE(1, RELATION('i', 1)),  NODE(1, RELATION('f', 2)),  NODE()  );  FST l\_extern(  str,  7,  NODE(1, RELATION('e', 1)),  NODE(1, RELATION('x', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('e', 4)),  NODE(1, RELATION('r', 5)),  NODE(1, RELATION('n', 6)),  NODE()  );  FST l\_str(  str,  4,  NODE(1, RELATION('s', 1)),  NODE(1, RELATION('t', 2)),  NODE(1, RELATION('r', 3)),  NODE()  );  FST l\_decimalLiteral(  str,  2,  NODE(20,  RELATION('-', 0),  RELATION('1', 0),  RELATION('2', 0),  RELATION('3', 0),  RELATION('4', 0),  RELATION('5', 0),  RELATION('6', 0),  RELATION('7', 0),  RELATION('8', 0),  RELATION('9', 0),  RELATION('0', 1),  RELATION('1', 1),  RELATION('2', 1),  RELATION('3', 1),  RELATION('4', 1),  RELATION('5', 1),  RELATION('6', 1),  RELATION('7', 1),  RELATION('8', 1),  RELATION('9', 1)  ),  NODE()  );  FST l\_hexLiteral(  str,  4,  NODE(2,  RELATION('-', 0),  RELATION('0', 1)  ),  NODE(1, RELATION('x', 2)),  NODE(43,  RELATION('1', 2),  RELATION('2', 2),  RELATION('3', 2),  RELATION('4', 2),  RELATION('5', 2),  RELATION('6', 2),  RELATION('7', 2),  RELATION('8', 2),  RELATION('9', 2),  RELATION('a', 2),  RELATION('b', 2),  RELATION('c', 2),  RELATION('d', 2),  RELATION('e', 2),  RELATION('f', 2),  RELATION('A', 2),  RELATION('B', 2),  RELATION('C', 2),  RELATION('D', 2),  RELATION('E', 2),  RELATION('F', 2),    RELATION('0', 3),  RELATION('1', 3),  RELATION('2', 3),  RELATION('3', 3),  RELATION('4', 3),  RELATION('5', 3),  RELATION('6', 3),  RELATION('7', 3),  RELATION('8', 3),  RELATION('9', 3),  RELATION('a', 3),  RELATION('b', 3),  RELATION('c', 3),  RELATION('d', 3),  RELATION('e', 3),  RELATION('f', 3),  RELATION('A', 3),  RELATION('B', 3),  RELATION('C', 3),  RELATION('D', 3),  RELATION('E', 3),  RELATION('F', 3)  ),  NODE()  );    FST l\_symb(  str,  5,  NODE(1, RELATION('s', 1)),  NODE(1, RELATION('y', 2)),  NODE(1, RELATION('m', 3)),  NODE(1, RELATION('b', 4)),  NODE()  );  FST l\_action(  str,  7,  NODE(1, RELATION('a', 1)),  NODE(1, RELATION('c', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('i', 4)),  NODE(1, RELATION('o', 5)),  NODE(1, RELATION('n', 6)),  NODE()  );  FST l\_declare(  str,  8,  NODE(1, RELATION('d', 1)),  NODE(1, RELATION('e', 2)),  NODE(1, RELATION('c', 3)),  NODE(1, RELATION('l', 4)),  NODE(1, RELATION('a', 5)),  NODE(1, RELATION('r', 6)),  NODE(1, RELATION('e', 7)),  NODE()  );  FST l\_return(  str,  7,  NODE(1, RELATION('r', 1)),  NODE(1, RELATION('e', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('u', 4)),  NODE(1, RELATION('r', 5)),  NODE(1, RELATION('n', 6)),  NODE()  );  FST l\_write(  str,  6,  NODE(1, RELATION('w', 1)),  NODE(1, RELATION('r', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('t', 4)),  NODE(1, RELATION('e', 5)),  NODE()  );  FST l\_main(  str,  5,  NODE(1, RELATION('m', 1)),  NODE(1, RELATION('a', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('n', 4)),  NODE()  );  FST l\_conditional(  str,  3,  NODE(1, RELATION('i', 1)),  NODE(1, RELATION('f', 2)),  NODE()  );  FST l\_semicolon(  str,  2,  NODE(1, RELATION(';', 1)),  NODE()  );  FST l\_comma(  str,  2,  NODE(1, RELATION(',', 1)),  NODE()  );  FST l\_braceleft(  str,  2,  NODE(1, RELATION('{', 1)),  NODE()  );  FST l\_braceright(  str,  2,  NODE(1, RELATION('}', 1)),  NODE()  );  FST l\_lefthesis(  str,  2,  NODE(1, RELATION('(', 1)),  NODE()  );  FST l\_righthesis(  str,  2,  NODE(1, RELATION(')', 1)),  NODE()  );  FST l\_leftsquare(  str,  2,  NODE(1, RELATION('[', 1)),  NODE()  );  FST l\_rightsquare(  str,  2,  NODE(1, RELATION(']', 1)),  NODE()  );  FST l\_verb(  str,  2,  NODE(11, RELATION('+', 1), RELATION('-', 1), RELATION('\*', 1),  RELATION('/', 1), RELATION(':', 1), RELATION('\\', 1), RELATION('%', 1), RELATION('=', 1), RELATION('&', 1), RELATION('|', 1), RELATION('^', 1)),  NODE()  );  FST l\_boolVerb(  str,  2,  NODE(4, RELATION('!', 1), RELATION('<', 1), RELATION('>', 1), RELATION('$', 1)),  NODE()  ); |

Листинг 1 – Конечные автоматы



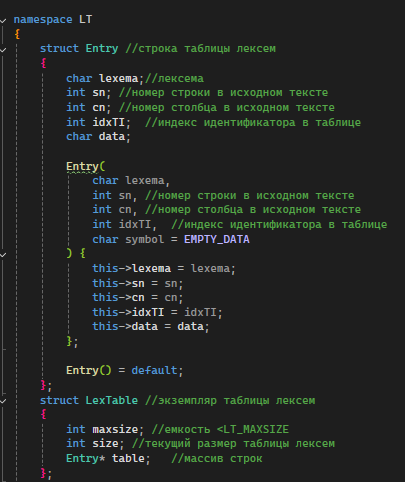


Рисунок 3 – Структура таблиц лексем и идентификаторов

## ПРИЛОЖЕНИЕГ

|  |
| --- |
| Greibach greibach(NS('S'), TS('$'),  9,  Rule(NS('S'), GRB\_ERROR\_SERIES + 0,  6,  Rule::Chain(7, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),  Rule::Chain(6, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}')),  Rule::Chain(15, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(13, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S'))  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1,  28,  Rule::Chain(7, TS('d'), TS('a'), TS('i'), TS('['), TS('l'), TS(']'), TS(';')),  Rule::Chain(8, TS('d'), TS('a'), TS('i'), TS('['), TS('l'), TS(']'), TS(';'), NS('N')),  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(5, TS('f '), TS('('), NS('F'), TS(')'), TS(';')),  Rule::Chain(6, TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),  Rule::Chain(9, TS('o'), TS('('), NS('B'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')),  Rule::Chain(8, TS('o'), TS('('), NS('B'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), TS('v'), NS('E'), TS(';')),  Rule::Chain(5, TS('i'), TS('v'), NS('E'), TS(';'), NS('N')),  Rule::Chain(9, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),  Rule::Chain(8, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';')),  Rule::Chain(8, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),  Rule::Chain(7, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';')),  Rule::Chain(4, TS('p'), TS('i'), TS(';'), NS('N')),  Rule::Chain(3, TS('p'), TS('i'), TS(';')),  Rule::Chain(4, TS('p'), TS('l'), TS(';'), NS('N')),  Rule::Chain(3, TS('p'), TS('l'), TS(';')),  Rule::Chain(7, TS('p'), TS('i'), TS('['), TS('l'), TS(']'), TS(';'), NS('N')),  Rule::Chain(6, TS('p'), TS('i'), TS('['), TS('l'), TS(']'), TS(';')),  Rule::Chain(7, TS('p'), TS('i'), TS('['), TS('i'), TS(']'), TS(';'), NS('N')),  Rule::Chain(6, TS('p'), TS('i'), TS('['), TS('i'), TS(']'), TS(';')),  Rule::Chain(7, TS('i'), TS('['), TS('l'), TS(']'), TS('v'), NS('E'), TS(';')),  Rule::Chain(8, TS('i'), TS('['), TS('l'), TS(']'), TS('v'), NS('E'), TS(';'), NS('N')),  Rule::Chain(7, TS('i'), TS('['), TS('i'), TS(']'), TS('v'), NS('E'), TS(';')),  Rule::Chain(8, TS('i'), TS('['), TS('i'), TS(']'), TS('v'), NS('E'), TS(';'), NS('N')),  Rule::Chain(8, TS('u'), TS('('), NS('B'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(9, TS('u'), TS('('), NS('B'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2,  14,  Rule::Chain(1, TS('i')),  Rule::Chain(2, TS('v'), TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),  Rule::Chain(3, TS('i'), TS('('), TS(')')),  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(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),  Rule::Chain(5, TS('i'), TS('['), TS('i'), TS(']'), NS('M')),  Rule::Chain(5, TS('i'), TS('['), TS('l'), TS(']'), NS('M')),  Rule::Chain(4, TS('i'), TS('['), TS('l'), TS(']')),  Rule::Chain(4, TS('i'), TS('['), TS('i'), TS(']'))  ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 3,  5,  Rule::Chain(1, TS('v')),  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(4, TS('v'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('v'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('v'), NS('E'), NS('M'))  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 4,  6,  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F')),  Rule::Chain(1, TS('i')),  Rule::Chain(3, TS('i'), TS(','), NS('F')),  Rule::Chain(1, TS('i')),  Rule::Chain(3, TS('l'), TS(','), NS('F'))  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 5,  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('B'), GRB\_ERROR\_SERIES + 6,  4,  Rule::Chain(3, TS('i'), TS('b'), TS('i')),  Rule::Chain(3, TS('i'), TS('b'), TS('l')),  Rule::Chain(3, TS('l'), TS('b'), TS('i')),  Rule::Chain(3, TS('l'), TS('b'), TS('l'))  )  ); |

Листинг 1 – Грамматика языка KNP-2024

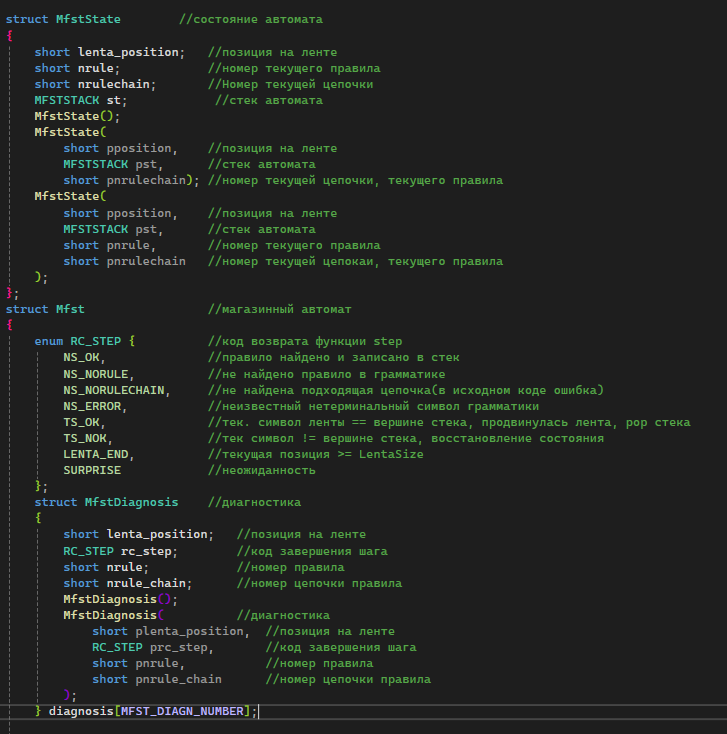


Рисунок 1 – Структура конечного магазинного автомата

## ПРИЛОЖЕНИЕД

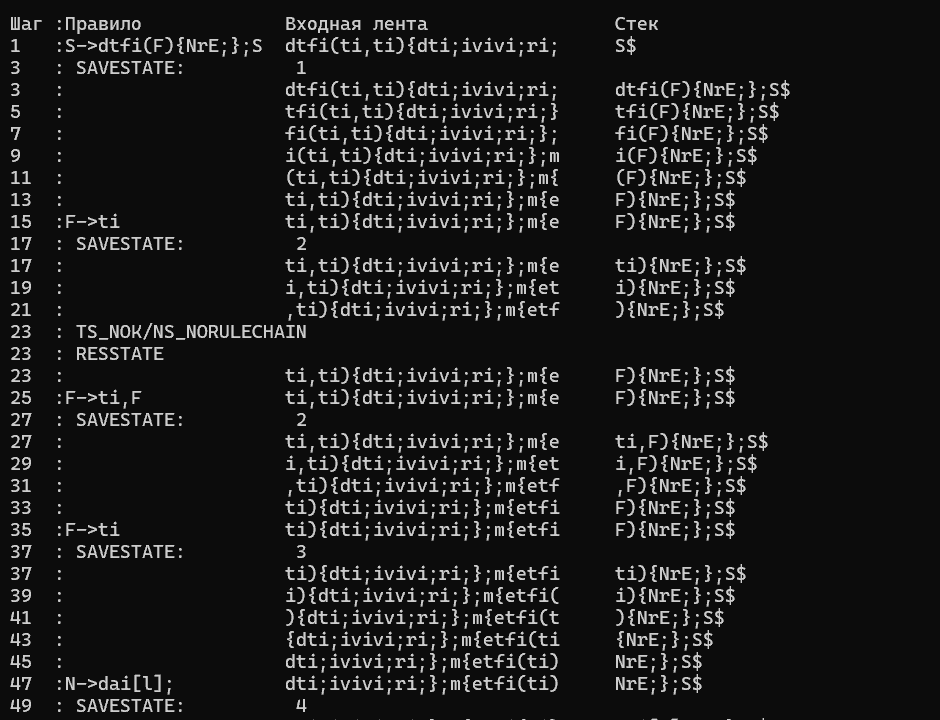


Рисунок 1 – Начало синтаксического анализа

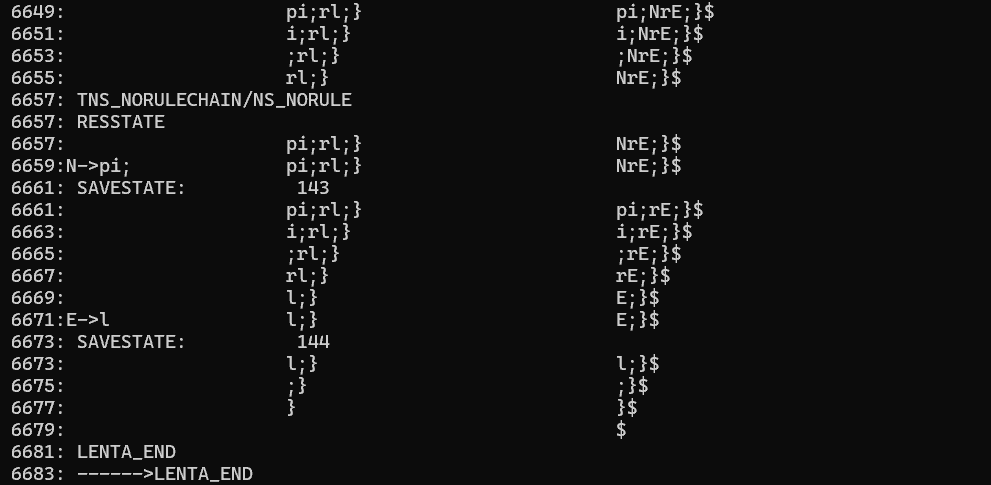


Рисунок 2 – Конец синтаксического анализа

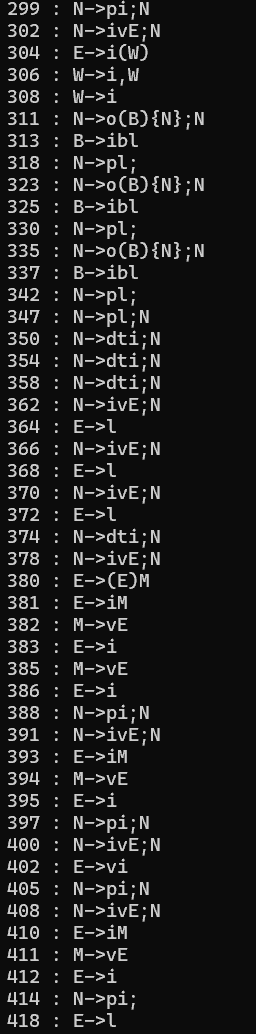
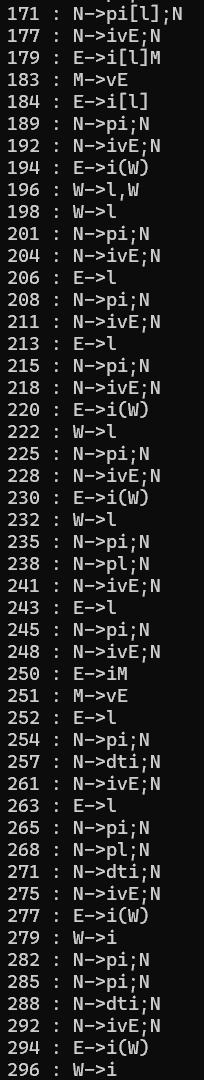
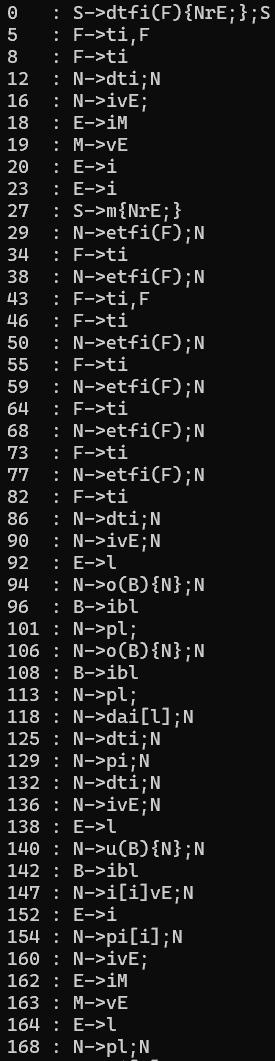
****

Рисунок 3 – Пример результата синтаксического анализа

## ПРИЛОЖЕНИЕ Е

|  |
| --- |
| #include <stack>  #include <vector>  #include <iostream>  #include "PolishNotation.h"  #include "Error.h"  namespace PolishNotation {  template <typename T>  struct container : T  {  using T::T;  using T::c;  };  std::string toString(int n) {  char buf[40];  sprintf\_s(buf, "%d", n);  return buf;  }  bool find\_elem(std::stack<char> stack, size\_t size, char elem) {  for (size\_t i = 0; i < size; i++)  if (stack.top() == elem)  return true;  else  stack.pop();  return false;  }  int get\_priority(char a)  {  switch (a)  {  case '(':  return 0;  case ')':  return 0;  case ',':  return 1;  case '-':  return 2;  case '+':  return 2;  case '\*':  return 3;  case '%':  return 3;  case '/':  return 3;  case '\\':  return 3;  case ':':  return 3;  case '&':  return 4;  case '|':  return 4;  case '^':  return 4;  default:  return 0;  }  }  void fixIt(LT::LexTable& lextable, const std::string& str, size\_t length, size\_t pos, const std::vector<int>& ids) {  for (size\_t i = 0, q = 0; i < str.size(); i++) {  lextable.table[pos + i].lexema = str[i];  if (lextable.table[pos + i].lexema == LEX\_ID || lextable.table[pos + i].lexema == LEX\_LITERAL) {  lextable.table[pos + i].idxTI = ids[q];  q++;  }  else  lextable.table[pos + i].idxTI = LT\_TI\_NULLIDX;  }  int temp = str.size() + pos;  int size = length - str.size();  for (size\_t i = 0; i < size; i++) {  lextable.table[temp + i].idxTI = LT\_TI\_NULLIDX;  lextable.table[temp + i].lexema = '\_';  lextable.table[temp + i].sn = -1;  }  }  bool PolishNotation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)  {  container<std::stack<char>> stack;  std::string PolishString;  std::vector<int> ids;  int operators\_count = 0, operands\_count = 0, iterator = 0, right\_counter = 0, left\_counter = 0, params\_counter = 0,square\_counter=0;  bool isInv = false;  bool isArrEl = false;  for (int i = lextable\_pos; i < lextable.size; i++, iterator++) {  char lexem = lextable.table[i].lexema;  char data = lextable.table[i].data;  size\_t stack\_size = stack.size();  if(lextable.table[i].idxTI!=TI\_NULLIDX){  if (idtable.table[lextable.table[i].idxTI].idtype == IT::IDTYPE::F) {  stack.push('@');  operands\_count--;  }  }  switch (lexem) {  case LEX\_OPERATOR:  {  if (!stack.empty() && stack.top() != LEX\_LEFTHESIS) {  while (!stack.empty() && get\_priority(data) <= get\_priority(stack.top())) {  PolishString += stack.top();  stack.pop();  }  }  if (data == '^')isInv = true;  stack.push(data);  operators\_count++;  break;  }  case LEX\_COMMA:  {  while (!stack.empty()) {  if (stack.top() == LEX\_LEFTHESIS)  break;  PolishString += stack.top();  stack.pop();  }  operands\_count--;  break;  }  case LEX\_LEFTHESIS:  {  left\_counter++;    stack.push(lexem);  break;  }    case LEX\_RIGHTHESIS:  {  right\_counter++;  if (!find\_elem(stack, stack\_size, LEX\_LEFTHESIS))  return false;  while (stack.top() != LEX\_LEFTHESIS) {  PolishString += stack.top();  stack.pop();  }  stack.pop();  if (!stack.empty() && stack.top() == '@') {  PolishString += stack.top() + toString(params\_counter - 1);  params\_counter = 0;  stack.pop();  }  break;  }  case LEX\_LEFT\_SQUAREBRACE:  {  square\_counter++;  PolishString += lexem;  isArrEl = true;  break;  }  case LEX\_RIGHT\_SQUAREBRACE:  {  square\_counter--;  PolishString += lexem;  isArrEl = false;  break;  }  case LEX\_SEMICOLON:  {  if (operators\_count != 0 && operands\_count != 0)  if ((!stack.empty() && (stack.top() == LEX\_RIGHTHESIS || stack.top() == LEX\_LEFTHESIS))  || right\_counter != left\_counter || (operands\_count - operators\_count != 1 && !isInv))  return false;  while (!stack.empty()) {  PolishString += stack.top();  stack.pop();  }  fixIt(lextable, PolishString, iterator, lextable\_pos, ids);  return true;  break;  }  case LEX\_ID: {  if (std::find(stack.c.begin(), stack.c.begin(), '@') != stack.c.end())  params\_counter++;  PolishString += lexem;  if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)  {  IT::Entry item = IT::GetEntry(idtable, lextable.table[i].idxTI);    ids.push\_back(lextable.table[i].idxTI);  }  if (!isArrEl) {  operands\_count++;  }  break;  }  case LEX\_LITERAL: {  if (std::find(stack.c.begin(), stack.c.begin(), '@') != stack.c.end())  {  params\_counter++;  }  PolishString += lexem;  if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)  ids.push\_back(lextable.table[i].idxTI);  if(!isArrEl){  operands\_count++;  }  break;  }  }  }  return true;  }  void DoPolish(LEX::LEX t) {  for (int i = 0; i < t.lextable.size; i++)  if (t.lextable.table[i].lexema == LEX\_EQUAL)  if (!PolishNotation(i + 1, t.lextable, t.idtable))  throw ERROR\_THROW(130);  for (int i = 0; i < t.lextable.size; i++)  if (t.lextable.table[i].lexema == '+' || t.lextable.table[i].lexema == '-' || t.lextable.table[i].lexema == '\*' ||  t.lextable.table[i].lexema == '/' || t.lextable.table[i].lexema == '\\' || t.lextable.table[i].lexema == ':' ||  t.lextable.table[i].lexema == '%' || t.lextable.table[i].lexema == '&' || t.lextable.table[i].lexema == '|' || t.lextable.table[i].lexema == '^')  {  t.lextable.table[i].data = t.lextable.table[i].lexema;  t.lextable.table[i].lexema = LEX\_OPERATOR;  }  }  } |

Листинг 1 – Алгоритм преобразования выражения к польской записи

## ПРИЛОЖЕНИЕ Ж

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../StaticLibraries/KNP-2024LIB.lib  ExitProcess PROTO :DWORD  Random PROTO :SDWORD  Compare PROTO :DWORD, :DWORD  GetSize PROTO :DWORD  Copy PROTO :DWORD  ctn PROTO :BYTE  ntc PROTO :SDWORD  outputuint PROTO :SDWORD  outputchar PROTO :BYTE  outputstr PROTO :DWORD  .stack 4096  .const  divideOnZeroExeption BYTE "Попытка деления на ноль.", 0 ;STR, для вывода ошибки при делении на ноль  main$LIT1 SDWORD -11 ;INT  main$LIT2 SDWORD 5 ;INT  main$LIT3 BYTE "If works", 0 ;STR  main$LIT5 BYTE "If doesn't work", 0 ;STR  main$LIT6 SDWORD 10 ;INT  main$LIT7 SDWORD 0 ;INT  main$LIT9 SDWORD 1 ;INT  main$LIT10 BYTE "element", 0 ;STR  main$LIT13 SDWORD 6 ;INT  main$LIT15 SDWORD 4 ;INT  main$LIT16 SDWORD 69 ;INT  main$LIT17 BYTE "Random:", 0 ;STR  main$LIT18 SDWORD 50 ;INT  main$LIT20 BYTE "shift!!!!", 0 ;STR  main$LIT21 SDWORD 12 ;INT  main$LIT22 SDWORD 2 ;INT  main$LIT23 BYTE "Hello, World!", 0 ;STR  main$LIT24 BYTE "Длина:", 0 ;STR  main$LIT26 BYTE "Строки одинаковы", 0 ;STR  main$LIT28 BYTE "Первая строка больше", 0 ;STR  main$LIT30 BYTE "Вторая строка больше", 0 ;STR  main$LIT31 BYTE "operations", 0 ;STR  main$LIT34 SDWORD 3 ;INT  main$LIT35 SDWORD 0 ;INT  .data  sumresult SDWORD 0 ;INT  mainnumber SDWORD 0 ;INT  mainmassiv SDWORD 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ;ARR  mainline DWORD 0 ;STR  maini SDWORD 0 ;INT  maingreet DWORD 0 ;STR  maincomp SDWORD 0 ;INT  maingreet2 DWORD 0 ;STR  maina SDWORD 0 ;INT  mainb SDWORD 0 ;INT  mainc SDWORD 0 ;INT  mainres SDWORD 0 ;INT  .code  $sum PROC uses ebx ecx edi esi , sumfirst: SDWORD , sumsecond: SDWORD  ; expression #3 :iviiv  push sumfirst  push sumsecond  pop ebx  pop eax  add eax, ebx  push eax  pop sumresult  mov eax, sumresult  ret  $sum ENDP  main PROC  mov esi,0 ;для работы с массивами  ; expression #17 :ivl  push main$LIT1  pop mainnumber  If94Start:  mov eax, mainnumber  mov ebx, main$LIT2  cmp eax, ebx  je If94End  push offset main$LIT3  CALL outputstr  If94End:  If106Start:  mov eax, mainnumber  mov ebx, main$LIT2  cmp eax, ebx  jne If106End  push offset main$LIT5  CALL outputstr  If106End:  push mainline  CALL outputstr  ; expression #32 :ivl  push main$LIT7  pop maini  While140Start:  mov eax, maini  mov ebx, main$LIT6  cmp eax, ebx  je While140End  ; expression #34 :i[i]vi  push maini  push maini  pop eax  imul eax,4  mov esi, OFFSET mainmassiv ;начало массива в еси  add esi, eax ;переход к нужному элементу  pop SDWORD PTR[esi]  push maini  pop eax  imul eax,4  mov esi, OFFSET mainmassiv ;начало массива в еси  add esi, eax ;переход к нужному элементу  push SDWORD PTR[esi]  CALL outputuint  ; expression #36 :ivilv  push maini  push main$LIT9  pop ebx  pop eax  add eax, ebx  push eax  pop maini  jmp While140Start  While140End:  push offset main$LIT10  CALL outputstr  mov esi, OFFSET mainmassiv ;начало массива в еси  add esi, 4 ;переход к нужному элементу  push SDWORD PTR[esi]  CALL outputuint  ; expression #42 :ivi[l]i[l]v  mov esi, OFFSET mainmassiv ; получаем начальный адрес  add esi, 20 ;переходим в адресе на нужное место(4 байта - элемент)  push SDWORD PTR[esi]  mov esi, OFFSET mainmassiv ; получаем начальный адрес  add esi, 24 ;переходим в адресе на нужное место(4 байта - элемент)  push SDWORD PTR[esi]  pop ebx  pop eax  add eax, ebx  push eax  pop mainnumber  push mainnumber  CALL outputuint  ; expression #45 :ivill@2\_  invoke $sum, main$LIT2, main$LIT15  mov mainnumber, eax ;результат функции  push mainnumber  CALL outputuint  ; expression #47 :ivl  push main$LIT16  pop mainnumber  push mainnumber  CALL outputuint  ; expression #49 :ivl  push offset main$LIT17  pop mainline  push mainline  CALL outputstr  ; expression #51 :ivil@1  invoke Random, main$LIT18  mov mainnumber, eax ;результат функции  push mainnumber  CALL outputuint  ; expression #53 :ivil@1  invoke Random, main$LIT6  mov mainnumber, eax ;результат функции  push mainnumber  CALL outputuint  push offset main$LIT20  CALL outputstr  ; expression #56 :ivl  push main$LIT21  pop mainnumber  push mainnumber  CALL outputuint  ; expression #58 :ivilv  push mainnumber  push main$LIT22  pop ebx  pop eax  push ecx ; сохраняем данные регистра ecx  mov ecx, ebx  SHL eax, cl  pop ecx  push eax  pop mainnumber  push mainnumber  CALL outputuint  ; expression #62 :ivl  push offset main$LIT23  pop maingreet  push maingreet  CALL outputstr  push offset main$LIT24  CALL outputstr  ; expression #67 :ivii@1  invoke GetSize, maingreet  mov maincomp, eax ;результат функции  push maincomp  CALL outputuint  push maingreet  CALL outputstr  ; expression #72 :ivii@1  invoke Copy, maingreet  mov maingreet2, eax ;результат функции  push maingreet2  CALL outputstr  ; expression #76 :iviii@2\_  invoke Compare, maingreet, maingreet2  mov maincomp, eax ;результат функции  If311Start:  mov eax, maincomp  mov ebx, main$LIT7  cmp eax, ebx  jne If311End  push offset main$LIT26  CALL outputstr  If311End:  If323Start:  mov eax, maincomp  mov ebx, main$LIT9  cmp eax, ebx  jne If323End  push offset main$LIT28  CALL outputstr  If323End:  If335Start:  mov eax, maincomp  mov ebx, main$LIT22  cmp eax, ebx  jne If335End  push offset main$LIT30  CALL outputstr  If335End:  push offset main$LIT31  CALL outputstr  ; expression #91 :ivl  push main$LIT2  pop maina  ; expression #92 :ivl  push main$LIT15  pop mainb  ; expression #93 :ivl  push main$LIT34  pop mainc  ; expression #95 :iviiviv\_\_  push maina  push mainc  pop ebx  pop eax  add eax, ebx  push eax  push mainb  pop ebx  pop eax  mul ebx  push eax  pop mainres  push mainres  CALL outputuint  ; expression #97 :iviiv  push maina  push mainb  pop ebx  pop eax  push ecx ; сохраняем данные регистра ecx  mov ecx, ebx  SHL eax, cl  pop ecx  push eax  pop mainres  push mainres  CALL outputuint  ; expression #99 :iviv  push mainb  pop ebx  pop eax  not ebx  push ebx  pop eax  push eax  pop mainres  push mainres  CALL outputuint  ; expression #101 :iviiv  push maina  push mainb  pop ebx  pop eax  or eax, ebx  push eax  pop mainres  push mainres  CALL outputuint  mov eax, main$LIT35  jmp endPoint  div\_by\_0:  push offset divideOnZeroExeption  CALL outputstr  endPoint:  invoke ExitProcess, eax  main ENDP  end main |

Листинг 1 – Код исходной программы на языке Ассемблер