МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 “Программное обеспечение информационных технологий”

Специализация 1-40 01 01 10 “Программное обеспечение информационных   
технологий (программирование интернет-приложений)”

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора TDS – 2022»

Выполнил студент Трубач Дмитрий Сергеевич

(Ф.И.О.)

Руководитель проекта преп.-стажер Карпович М. Н.

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

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

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

Консультанты преп.-стажер Карпович М. Н.

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

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

Нормоконтролер преп.-стажер Карпович М. Н.

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 5](#_Toc122449900)

[1 Спецификация языка программирования 6](#_Toc122449901)

[1.1 Характеристика языка программирования 6](#_Toc122449902)

[1.2 Определение алфавит языка программирования 6](#_Toc122449903)

[1.3 Применяемые сепараторы 6](#_Toc122449904)

[1.4 Применяемые кодировки 7](#_Toc122449905)

[1.5 Типы данных 7](#_Toc122449906)

[1.6 Преобразование типов данных 8](#_Toc122449907)

[1.7 Идентификаторы 8](#_Toc122449908)

[1.8 Литералы 8](#_Toc122449909)

[1.9 Область видимости идентификаторов 9](#_Toc122449910)

[1.10 Инициализация данных 9](#_Toc122449911)

[1.11 Инструкции языка 9](#_Toc122449912)

[1.13 Выражения и их вычисления 11](#_Toc122449913)

[1.14 Программные конструкции языка 11](#_Toc122449914)

[1.15 Область видимости 11](#_Toc122449915)

[1.16 Семантические проверки 11](#_Toc122449916)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc122449917)

[1.18 Стандартная библиотека и её состав 12](#_Toc122449918)

[1.19 Ввод и вывод данных 13](#_Toc122449919)

[1.20 Точка входа 13](#_Toc122449920)

[1.21 Препроцессор 13](#_Toc122449921)

[1.22 Соглашения о вызовах 14](#_Toc122449922)

[1.23 Объектный код 14](#_Toc122449923)

[1.24 Классификация сообщений транслятора 14](#_Toc122449924)

[1.25 Контрольный пример 14](#_Toc122449925)

[2 Структура транслятора 15](#_Toc122449926)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc122449927)

[2.2 Перечень входных параметров транслятора 16](#_Toc122449928)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 16](#_Toc122449929)

[3 Разработка лексического анализатора 17](#_Toc122449930)

[3.1 Структура лексического анализатора 17](#_Toc122449931)

[3.2 Контроль входных символов 17](#_Toc122449932)

[3.3 Удаление избыточных символов 18](#_Toc122449933)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 18](#_Toc122449934)

[3.5 Основные структуры данных 19](#_Toc122449935)

[3.6 Структура и перечень сообщений лексического анализатора 20](#_Toc122449936)

[3.7 Принцип обработки ошибок 20](#_Toc122449937)

[3.8 Параметры лексического анализатора 20](#_Toc122449938)

[3.9 Алгоритм лексического анализа 20](#_Toc122449939)

[3.10 Контрольный пример 21](#_Toc122449940)

[4 Разработка синтаксического анализатора 22](#_Toc122449941)

[4.1 Структура синтаксического анализатора 22](#_Toc122449942)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 22](#_Toc122449943)

[4.3 Построение конечного магазинного автомата 24](#_Toc122449944)

[4.4 Основные структуры данных 25](#_Toc122449945)

[4.5 Описание алгоритма синтаксического разбора 25](#_Toc122449946)

[4.6 Структура и перечень сообщений синтаксического анализатора 26](#_Toc122449947)

[4.7 Параметры синтаксического анализатора и режимы его работы 26](#_Toc122449948)

[4.8 Принцип обработки ошибок 26](#_Toc122449949)

[4.9 Контрольный пример 26](#_Toc122449950)

[5 Разработка семантического анализатора 27](#_Toc122449951)

[5.1 Структура семантического анализатора 27](#_Toc122449952)

[5.2 Функции семантического анализатора 27](#_Toc122449953)

[5.3 Структура и перечень сообщений семантического анализатора 27](#_Toc122449954)

[5.4 Принцип обработки ошибок 28](#_Toc122449955)

[5.5 Контрольный пример 28](#_Toc122449956)

[6.1 Выражения, допускаемые языком 29](#_Toc122449957)

[6.2 Польская запись 29](#_Toc122449958)

[6.3 Программная реализация обработки выражений 30](#_Toc122449959)

[6.4 Контрольный пример 30](#_Toc122449960)

[7 Генерация кода 31](#_Toc122449961)

[7.1 Структура генератора кода 31](#_Toc122449962)

[7.2 Представление типов данных в оперативной памяти 31](#_Toc122449963)

[7.3 Статическая библиотека 32](#_Toc122449964)

[7.4 Алгоритм работы генератора кода 33](#_Toc122449965)

[7.5 Контрольный пример 35](#_Toc122449966)

[8 Тестирование транслятора 36](#_Toc122449967)

[8.1 Тестирование фазы проверки на допустимость символов 36](#_Toc122449968)

[8.2 Тестирование лексического анализатора 36](#_Toc122449969)

[8.3 Тестирование синтаксического анализатора 37](#_Toc122449970)

[8.4 Тестирование семантического анализатора 37](#_Toc122449971)

[Заключение 39](#_Toc122449972)

[Список использованных источников 40](#_Toc122449973)

[ПРИЛОЖЕНИЕ А 41](#_Toc122449974)

[ПРИЛОЖЕНИЕ Б 42](#_Toc122449975)

[ПРИЛОЖЕНИЕ В 43](#_Toc122449976)

[ПРИЛОЖЕНИЕ Г 54](#_Toc122449977)

[ПРИЛОЖЕНИЕ Д 57](#_Toc122449978)

[ПРИЛОЖЕНИЕ Е 59](#_Toc122449979)

[ПРИЛОЖЕНИЕ Ж 63](#_Toc122449980)

# **Введение**

Задачей данного курсового проекта является разработка транслятора для своего языка программирования: TDS-2022.

Написание транслятора будет осуществляться на языке C++, при этом код на языке TDS-2022 будет транслироваться в язык ассемблера.

Транслятор TDS-2022 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;

– разбратка 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 TDS-2022 – это универсальный язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования.

## **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.

## **Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей операций языка. Сепараторы, используемые в языке программирования TDS-2022, 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 |
| /  \  +  -  \*  : | Знаки «косая черта», «обратная косая черта», «плюс», «астерикс», «тильда», | Выражения |

Продолжение таблицы 1.1

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| &  ^  <  > | «амперсанд», «циркумфлекс», знаки «больше» и «меньше» | Выражения в операторе цикла |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования TDS-2022 используется кодировка Windows-1251.

## **Типы данных**

В языке TDS-2022 реализованы четыре типа данных: целочисленный беззнаковый(uint), символьный(char), строковый(string). Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка TDS-2022

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных uint | Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями. В памяти занимает 4 байта.  При попытке инициализации значением больше максимального, инициализируется максимальным. При попытке инициализации значением меньше минимального, инициализируется значением, которое равно разнице между максимальным и исходным значением.  Максимальное значение: 4294967295. Минимальное значение: 0.  Инициализация по умолчанию: значение 0. |
| Строковый тип данных string | 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 char | Фундаментальный тип данных. Используется для работы с символом, который в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |

Пользовательские типы данных не поддерживаются.

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, т.е. язык является строготипизированным. Но в стандартной библиотеке есть функции преобразования типа uint в тип char и наоборот.

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита и знак «\_». 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 TDS-2022 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [1-9]+[0-9]\* | Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Литералы могут быть только rvalue. | declare uint sum = 15;  15 – целочисленный литерал. |
| Символьный литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/] | Символ, заключённый в ‘’ (одинарные кавычки), по умолчанию инициализируются пустой строкой. Литералы могут быть только rvalue. | declare char symbol = ‘T’;  T – символьный литерал. |
| Строковый литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/]\* | Строковые литералы, максимальная длина строки 255 символов | print “text” |

Литералы являются константами и при генерации кода объявляются один раз.

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке TDS-2022 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 не допускается инициализация. Описание способов инициализации переменных языка TDS-2022 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| declare <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа uint инициализируются нулём, переменные типа char – пустым символом. | declare uint sum;  declare symbol chr; |
| <идентификатор> = <значение>; | Присваивание переменной значения. | sum = 9;  chr = ‘D’; |

Соответствие типов проверяется на синтаксическом анализе.

## **Инструкции языка**

Все возможные инструкции языка программирования TDS-2022 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования TDS-2022

|  |  |
| --- | --- |
| Инструкция | Запись на языке TDS-2022 |
| Объявление переменной | declare <тип данных> <идентификатор>; |
| Объявление функции | declare <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Объявление внешней функции | extern <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …); |
| Блок инструкций | main  {  …  } |

Продолжение таблицы 1.5

|  |  |
| --- | --- |
| Инструкция | Запись на языке TDS-2022 |
| Возврат из подпрограммы | return <идентификатор> / <литерал>; |
| Условная инструкция | while(<условие>)[<блок кода>]; |
| Вывод данных | print <идентификатор> / <литерал>; |
| Однострочный комментарий до конца строки | #<любой текст> |

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

* 1. **Операции языка**

Язык программирования TDS-2022 может выполнять операции, представленные в таблице 1.6. Операция сдвига учитывает только первый младший бит оператора, т.к. сдвиг более чем на 255 любого числа кроме нуля вернет число большее, чем можно разместить в типе данных uint.

Таблица 1.6 – Операции языка программирования TDS-2022

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| ( | Приоритет операций | - | sum = (a + b) \* c; |
| ) |
| + | Суммирование | (uint, uint)  (char, char) | sum = a + b; |
| - | Вычитание | (uint, uint) | diff = a – b; |
| \* | Умножение | (uint, uint) | mul = a\*b; |
| : | Деление | (uint, uint) | div = a:b; |
| % | Остаток от деления | (uint, uint) | mod = a%b; |
| / | Сдвиг влево | (uint, uint) | pr = a / b; |
| \ | Сдвиг вправо | (uint, uint) | pr = a \ b; |
| = | Присваивание | (uint, uint)  (char, char)  (string, string) | sum = 15;  chr = ‘T’; |
| <,> | Знаки «больше», «меньше» для условной инструкции | (uint, uint)  (char, char) | while(sum < diff) […]; |
| & | Оператор эквивалентности | (uint, uint)  (char, char) | while(sum & diff) […]; |
| ^ | Оператор неравенства | (uint, uint)  (char, char) | while(sum ^ diff) […]; |

Т.к. отрицательные числа не поддерживаются, если результат операции меньше нуля, он вычитается из максимального значения.

## **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 TDS-2022 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка TDS-2022

|  |  |
| --- | --- |
| Конструкция | Запись на языке TDS-2022 |
| Главная функция (точка входа) | main  {  …  return <идентификатор> / <литерал>;  } |
| Функция | declare <тип данных> func <идентификатор> (<тип> <идентификатор>, …)  {… return <идентификатор> / <литерал>; }; |
| Цикл | while(a^8)[ …]; |
| Условный оператор | if(5>4)[…]; |

Программные конструкции языка TDS-2022 представляют собой базовый функционал для выполнения различных операций, что делает возможным решать задачи различного уровня.

## **Область видимости**

В языке TDS-2022 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций (по принципу С++). 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 string не может быть аргументом условной конструкции |
| 8 | Для типа char определены только операции + и - |
| 9 | Функции не должны подключаться дважды в пределах одной программы |

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

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

Все переменные размещаются в куче.

## **Стандартная библиотека и её состав**

Стандартная библиотека TDS-2022 написана на языке программирования C++.

Для использования функций стандартной библиотеки, нужно явно подключить необходимую функцию с помощью ключевого 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 |
| int ord(char) | char | Возвращает код символа |
| char chr(int) | char | Возвращает символ с заданным кодом |
| int GetMonth(int number) | uint | Возвращает номер месяца |

Продолжение таблицы 1.9

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| int GetDate(int number) | uint | Возвращает дату в формате ДДММГГГГ |
| int GetHours(int number) | uint | Возвращает час |
| int GetMinutes(int number) | uint | Возвращает минуты |

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 outputuint (unsigned int a) | – | Выводит число на экран Вызывается оператором print |
| void outputchar (char a) | – | Выводит символ на экран Вызывается оператором print |
| void outputstring (void\* in) | – | Выводит строку на экран Вызывается оператором print |

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

## **Ввод и вывод данных**

В языке TDS-2022 не реализованы средства ввода данных.

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

## **Точка входа**

В языке TDS-2022 каждая программа должна содержать главную функцию main, т. е. точку входа, с которой начнется последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования TDS-2022 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**

TDS-2022 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке TDS-2022 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 |
| 120-129 | Ошибки лексического анализа |
| 130-139 | Ошибки таблиц лексем и таблиц идентификаторов |
| 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 преобразует программу, написанную на языке TDS-2022 в программу на языке ассемблера [1][2]. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка TDS-2022 и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы TDS-2022 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода TDS-2022. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке TDS-2022, прошедший успешно все предыдущие этапы, в код на языке Ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка TDS-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке TDS-2022. 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 TDS-2022. 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 лог журнал.

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

Таблица с перечнем протоколов, формируемых транслятором языка TDS-2022 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка TDS-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Файл содержит в себе краткую информацию об исходном коде на языке TDS-2022. 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 TDS-2022. 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 TDS-2022 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 TDS-2022 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 | uint | i |
| Строковый тип данных | string | i |
| Символьный тип данных | char | i |
| Лексема | Объявление переменной | declare | v |
| Подключение функции библиотеки | extern | e |
| Оператор вывода | print | p |
| Объявление функции | func | f |
| Возврат значения из функции | return | r |
| Инструкция цикла | while | 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 |
| 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 TDS-2022.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка TDS-2022, 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.

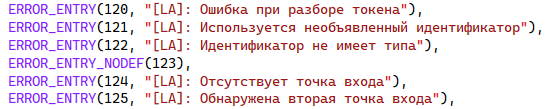


Рисунок 3.2 – Перечень ошибок лексического анализатора

## **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 – “$LEX”, 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.

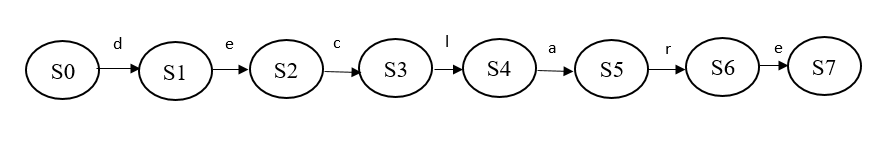


Рисунок 3.3 – Граф переходов для цепочки «declare»

## **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 TDS-2022. 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 TDS-2022

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 TDS-2022 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 TDS-2022 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 TDS-2022

|  |  |  |
| --- | --- | --- |
| 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;  rE;  ivE;  etfi(F);  o(B)[N]N  o(B)[N]  dti;N  ivE;N  etfi(F);N  pi;  pl;  pi;N  pl;N  u(B)[N];  u(B)[N];N  etfi();N  etfi(); | Порождает правила, описывающие конструкции языка |
| E | i  l  (E)  i(W)  iM  lM  (E)  i(W)  i()  i(M)  l(M) | Порождает правила, описывающие выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| E | (E)M  i(W)M  i(W) | Порождает правила, описывающие выражения |
| F | ti  ti,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 | 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 |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | 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 TDS-2022. 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.

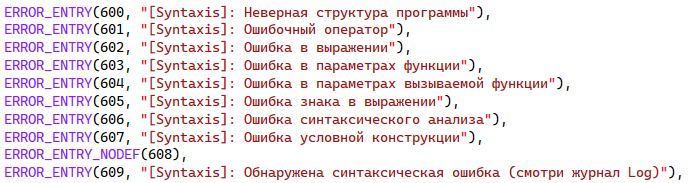


Рисунок 4.1 – Перечень сообщений синтаксического анализатора

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 TDS-2022 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.

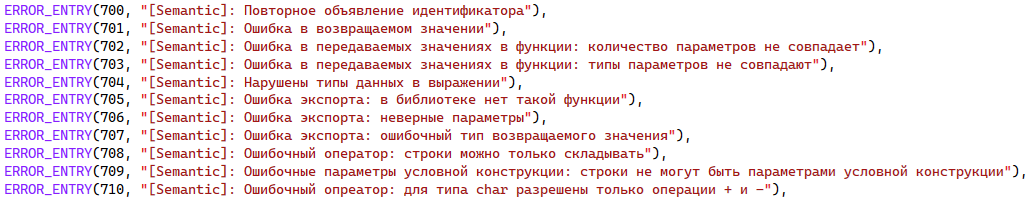


Рисунок 5.2 – Перечень сообщений семантического анализатора

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 TDS-2022 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические 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 TDS-2022

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

Некоторые из операций таблицы 6.1 используются для типов, отличных от целочисленных.

## **6.2 Польская запись**

Выражения в языке TDS-2022 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 |
| x+y\*5/(z-2) |  |  |
| +y\*5/(z-2) | x |  |
| y\*5/(z-2) | x | + |
| \*5/(z-2) | xy | + |
| 5/(z-2) | xy | +\* |
| /(z-2) | xy5 | +\* |
| (z-2) | xy5\* | +/ |
| z-2) | xy5\* | +/( |
| -2) | xy5\*z | +/( |
| 2) | xy5\*z | +/(- |
| ) | xy5\*z2 | +/(- |
|  | xy5\*z2- | +/ |
|  | xy5\*z2-/ | + |
|  | xy5\*z2-/+ |  |

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

## **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 TDS-2022 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 TDS-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка TDS-2022 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке TDS-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| char | BYTE | Хранит символьный тип данных. |
| string | DWORD | Хранит указатель на начало строки. |
| uint | DWORD | Хранит целочисленный тип данных без знака. |
| Лексема | BYTE  DWORD  DWORD | Литералы: символьные,  целочисленные, строковые |

Идентификаторы языка TDS-2022 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 TDS-2022LIB, 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 ../Debug/TDS-2022LIB.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 << " :DWORD ";  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 :DWORD";  \*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 ../Debug/TDS-2022LIB.lib  ExitProcess PROTO :DWORD  GetHours PROTO :DWORD  GetMonth PROTO :DWORD  GetMinutes PROTO :DWORD  GetDate PROTO :DWORD  outputuint PROTO :DWORD  outputchar PROTO :BYTE  outputstring PROTO :DWORD |

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 ;STRING, для вывода ошибки при делении на ноль  FindFactor$LEX1 DWORD 1 ;INT  main$LEX4 BYTE "Stringing:", 0 ;STRING  main$LEX5 BYTE "okay", 0 ;STRING  main$LEX6 BYTE 'q' ;CHR  main$LEX7 BYTE "Symbol", 0 ;STRING  main$LEX8 BYTE "Factorial of number 5", 0 ;STRING  main$LEX9 DWORD 5 ;INT  main$LEX10 BYTE "Number to be circilar shifted:", 0 ;STRING  main$LEX11 DWORD 32 ;INT  main$LEX12 DWORD 3 ;INT  main$LEX14 BYTE "32<<3:", 0 ;STRING  main$LEX15 BYTE "32>>1:", 0 ;STRING  main$LEX16 DWORD 6 ;INT  main$LEX18 BYTE "If constringuction works", 0 ;STRING  main$LEX21 BYTE "If constringuction not works", 0 ;STRING  main$LEX26 BYTE "Hours:", 0 ;STRING  main$LEX27 BYTE "Minutes:", 0 ;STRING  main$LEX28 BYTE "Date:", 0 ;STRING  main$LEX29 BYTE "Month:", 0 ;STRING  main$LEX30 DWORD 0 ;INT |

Листинг 7.3 – Пример заполнения поля .const

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

|  |
| --- |
| .data  FindFactoranswer DWORD 0 ;INT  mainstring1 DWORD 0 ;STRING  mainsymb BYTE 0 ;CHR  mainnumber DWORD 0 ;INT  maindemo DWORD 0 ;INT  maindemo1 DWORD 0 ;INT  maindemo2 DWORD 0 ;INT  mainnumber1 DWORD 0 ;INT  mainnumber2 DWORD 0 ;INT  mainnumber3 DWORD 0 ;INT  mainnumber4 DWORD 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  $FindFactor PROC uses ebx ecx edi esi , FindFactora: DWORD  ; Stringing #3 :ivl  push FindFactor$LEX1  pop FindFactoranswer  While17Start:  mov eax, FindFactora  mov ebx, FindFactor$LEX1  cmp eax, ebx  jl While17End  ; Stringing #6 :iviiv  push FindFactoranswer  push FindFactora  pop ebx  pop eax  mul ebx  push eax  pop FindFactoranswer  ; Stringing #7 :ivilv  push FindFactora  push FindFactor$LEX1  pop ebx  pop eax  sub eax, ebx  push eax  pop FindFactora  jmp While17Start  While17End:  mov eax, FindFactoranswer  ret  $FindFactor ENDP |

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 посмотреть в приложении Ж.

# **8 Тестирование транслятора**

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

В языке TDS-2022 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 uint 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 uint func a(){  declare q;  } | Ошибка 122: [LA]: Идентификатор не имеет типа, строка 2, столбец 6 |
| declare uint func a(){  declare uint q;  } | Ошибка 124: [LA]: Отсутствует точка входа |
| main(){  declare char a;  }  main(){  declare char 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 char 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 char a;  declare string a;  } | Ошибка 700: [Semantic]: Повторное объявление идентификатора, строка 3, столбец 10 |
| declare uint func f(uint q){  declare string a;  return a;  };  main{  declare uint a;  a = f(3);  return 0;  }; | Ошибка 701: [Semantic]: Ошибка в возвращаемом значении, строка 3, столбец 6 |

Продолжение таблицы 8.4

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| declare uint func f(uint a, uint b){  return 5;  };  main{  declare uint a;  a = f(a);  return a;  }; | Ошибка 702: [Semantic]: Ошибка в передаваемых значениях в функции: количество параметров не совпадает, строка 6, столбец 8 |
| declare uint func f(string a){  return 5;  };  main{  declare uint a;  a = f(a);  return a;  }; | Ошибка 703: [Semantic]: Ошибка в передаваемых значениях в функции: типы параметров не совпадают, строка 6, столбец 7 |
| declare uint func f(uint q){  declare uint a;  return a;  };  main{  declare string a;  a = f(3);  return 0;  }; | Ошибка 704: [Semantic]: Нарушены типы данных в выражении, строка 7, столбец 5 |
| main{  extern uint func random(uint max);  declare uint 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 TDS-2022 на язык ассемблера.

Язык TDS-2022 поддерживает 3 типа данных: беззнаковый целочисленный (uint), строковый (string), символьный(char).

Для целочисленного типа реализована обработка 7 арифметических действий, скобок, обозначающих приоритет операций.

На этапе семантического анализа производится проверка соответствия исходного кода спецификации по 10 правилам.

Реализованы 6 публичные и 3 приватные функции стандартной библиотеки.

# **Список использованных источников**

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

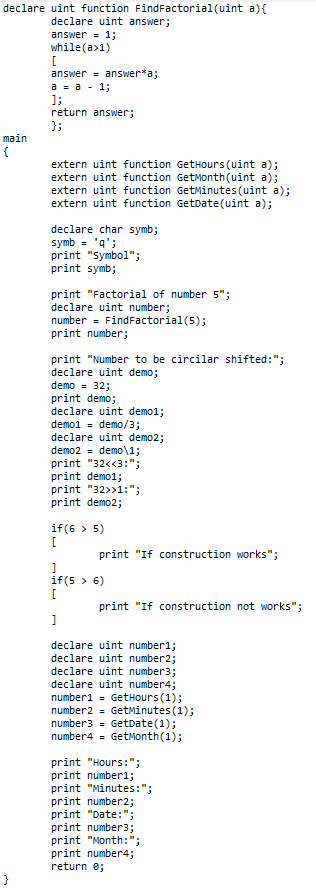


Рисунок 1 – Контрольный пример

## ПРИЛОЖЕНИЕ Б

|  |
| --- |
|  |

Рисунок 1 – Таблица входных символов

## ПРИЛОЖЕНИЕ В

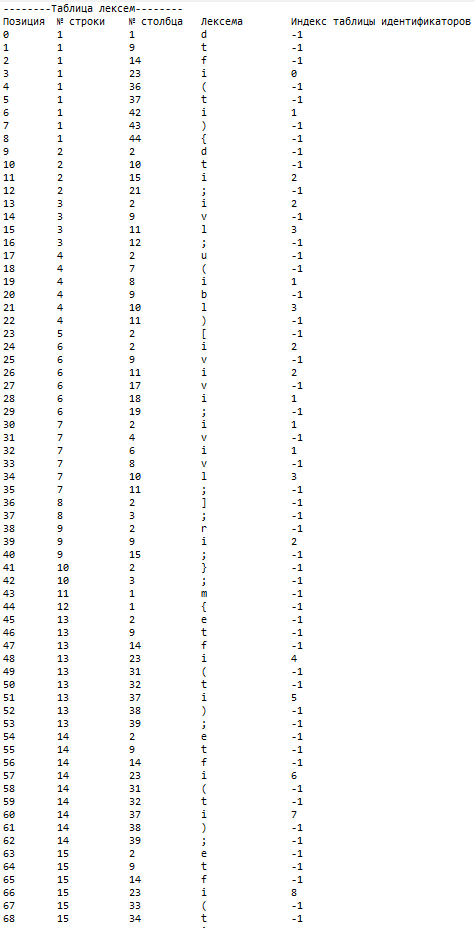


Рисунок 1– Таблица лексем

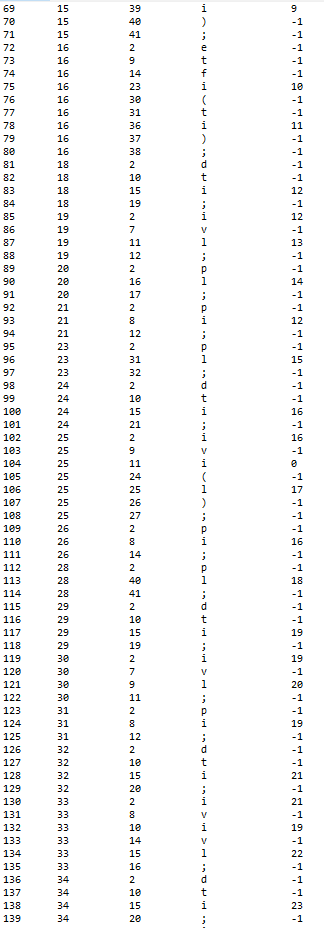


Рисунок 1 (продолжение) – Таблица лексем

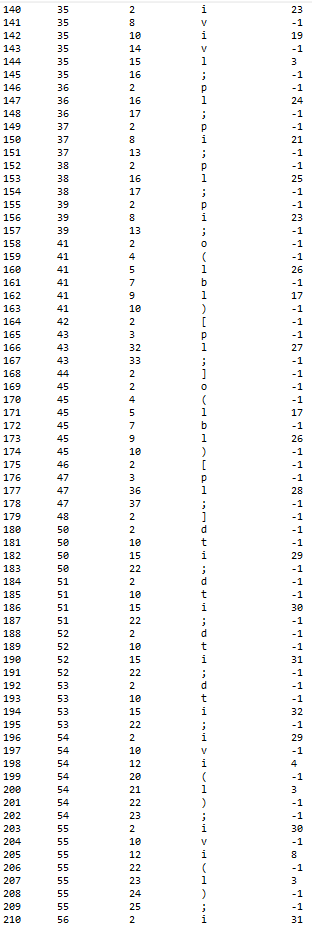


Рисунок 1 (продолжение) – Таблица лексем

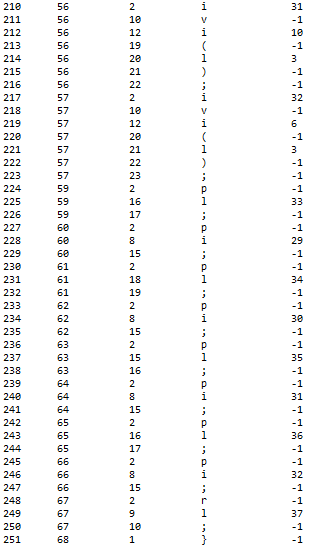


Рисунок 1 (конец) – Таблица лексем

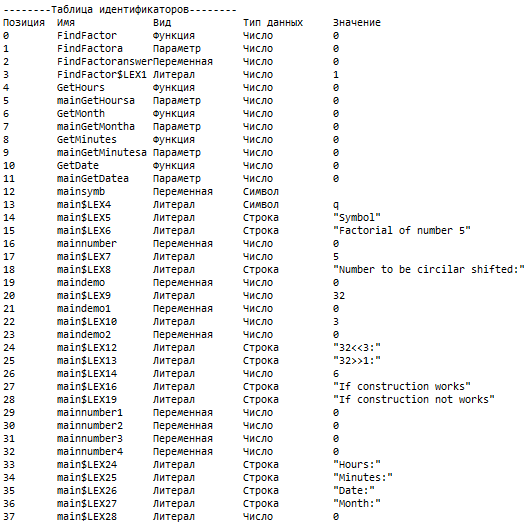


Рисунок 2 – Таблица идентификаторов

|  |
| --- |
| FST l\_uint(  str,  5, //количество состояний  NODE(1, RELATION('u', 1)),  NODE(1, RELATION('i', 2)),  NODE(1, RELATION('n', 3)),  NODE(1, RELATION('t', 4)),  NODE()  );  FST l\_until(  str,  6, //количество состояний  NODE(1, RELATION('w', 1)),  NODE(1, RELATION('h', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('l', 4)),  NODE(1, RELATION('e', 5)),  NODE()  );  FST l\_if(  str,  3, //количество состояний  NODE(1, RELATION('i', 1)),  NODE(1, RELATION('f', 2)),  NODE()  );  FST l\_extern(  str,  7, //количество состояний  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,  7, //количество состояний  NODE(1, RELATION('s', 1)),  NODE(1, RELATION('t', 2)),  NODE(1, RELATION('r', 3)),  NODE(1, RELATION('i', 4)),  NODE(1, RELATION('n', 5)),  NODE(1, RELATION('g', 6)),  NODE()  );  FST l\_numberLiteral(  str,  2, //количество состояний  NODE(20,  RELATION('0', 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\_string(  str,  5, //количество состояний  NODE(1, RELATION('c', 1)),  NODE(1, RELATION('h', 2)),  NODE(1, RELATION('a', 3)),  NODE(1, RELATION('r', 4)),  NODE()  );  FST l\_function(  str,  9, //количество состояний  NODE(1, RELATION('f', 1)),  NODE(1, RELATION('u', 2)),  NODE(1, RELATION('n', 3)),  NODE(1, RELATION('c', 4)),  NODE(1, RELATION('t', 5)),  NODE(1, RELATION('i', 6)),  NODE(1, RELATION('o', 7)),  NODE(1, RELATION('n', 8)),  NODE()  );  FST l\_declare(  str,  8, //количество состояний  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, //количество состояний  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\_printi(  str,  6, //количество состояний  NODE(1, RELATION('p', 1)),  NODE(1, RELATION('r', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('n', 4)),  NODE(1, RELATION('t', 5)),  NODE()  );  FST l\_prints(  str,  5, //количество состояний  NODE(1, RELATION('s', 1)),  NODE(1, RELATION('a', 2)),  NODE(1, RELATION('y', 3)),  NODE(1, RELATION('s', 4)),  NODE()  );  FST l\_main(  str,  5, //количество состояний  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, //количество состояний  NODE(1, RELATION('i', 1)),  NODE(1, RELATION('f', 2)),  NODE()  );  FST l\_semicolon(  str,  2, //количество состояний  NODE(1, RELATION(';', 1)),  NODE()  );  FST l\_comma(  str,  2, //количество состояний  NODE(1, RELATION(',', 1)),  NODE()  );  FST l\_braceleft(  str,  2, //количество состояний  NODE(1, RELATION('{', 1)),  NODE()  );  FST l\_braceright(  str,  2, //количество состояний  NODE(1, RELATION('}', 1)),  NODE()  );  FST l\_lefthesis(  str,  2, //количество состояний  NODE(1, RELATION('(', 1)),  NODE()  );  FST l\_cycleStart(  str,  2, //количество состояний  NODE(1, RELATION('[', 1)),  NODE()  );  FST l\_cycleEnd(  str,  2, //количество состояний  NODE(1, RELATION(']', 1)),  NODE()  );  FST l\_conditionalStart(  str,  2, //количество состояний  NODE(1, RELATION('[', 1)),  NODE()  );  FST l\_conditionalEnd(  str,  2, //количество состояний  NODE(1, RELATION(']', 1)),  NODE()  );  FST l\_righthesis(  str,  2, //количество состояний  NODE(1, RELATION(')', 1)),  NODE()  );  FST l\_verb(  str,  2, //количество состояний  NODE(8, RELATION('+', 1), RELATION('-', 1), RELATION('\*', 1),  RELATION('/', 1), RELATION(':', 1), RELATION('\\', 1), RELATION('%', 1), RELATION('=', 1)),  NODE()  );  FST l\_boolVerb(  str,  2, //количество состояний  NODE(4, RELATION('^', 1), RELATION('<', 1), RELATION('>', 1), RELATION('&', 1)),  NODE()  ); |

Листинг 1 – Конечные автоматы

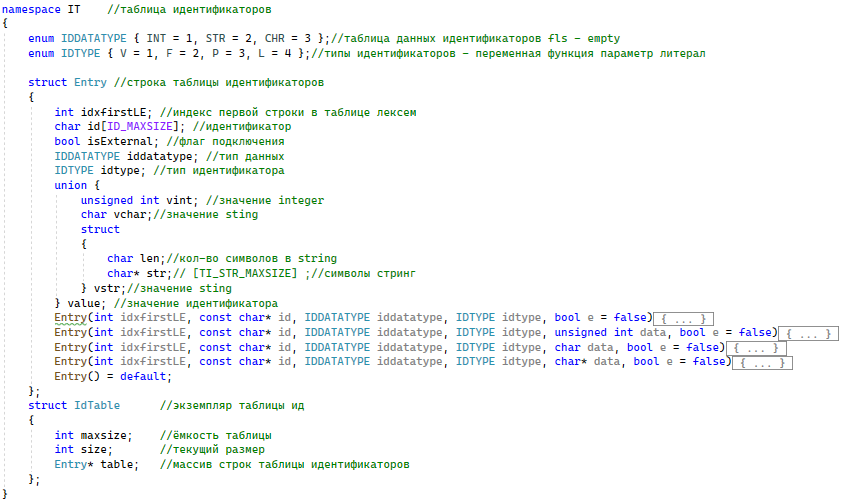


Рисунок 3 – Структура таблиц лексем и идентификаторов

## ПРИЛОЖЕНИЕГ

|  |
| --- |
| Rule(NS('S'), GRB\_ERROR\_SERIES + 0,  6, //todo m{NrE;}; tfi(F){NrE;};S m{NrE;};S tfi(F){NrE;};  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,  18, //todo dti; rE; i=E; dtfi(F); dti;N rE;N i=E;N dtfi(F);N pl;N pi;N pl; pi; pi(W);  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),//+  //Rule::Chain(5, TS('d'), TS('t'), TS('i'), NS('M'), 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(8, TS('o'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), NS('N')),//+  Rule::Chain(5, TS('o'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), 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(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,  9, //todo i l (E) i(W) iM lM (E)M i(W)M  Rule::Chain(1, 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(NS('M'), GRB\_ERROR\_SERIES + 3,  4, //todo vE vEM v(E) v(E)M  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, //todo ti ti,F  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, //todo i l i,W l,W  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, //todo ibi ibl lbi lbl  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 TDS-2022



Рисунок 1 – Структура конечного магазинного автомата

## ПРИЛОЖЕНИЕД

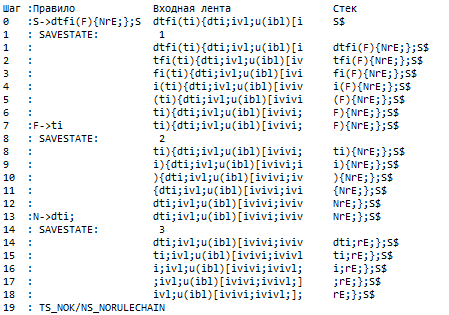


Рисунок 1 – Начало синтаксического анализа

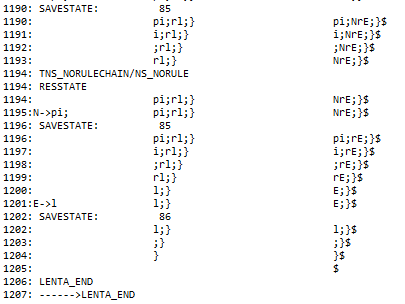


Рисунок 2 – Конец синтаксического анализа

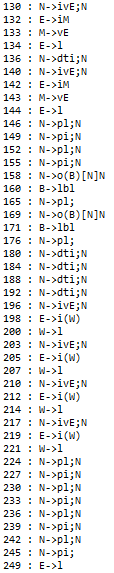
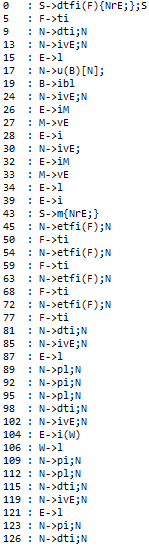


Рисунок 3 – Пример разбора синтаксическим анализатором

## ПРИЛОЖЕНИЕ Е

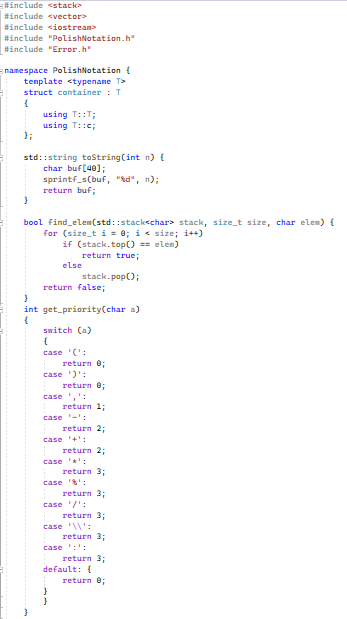


Рисунок 1 – Алгоритм преобразования выражения к польской записи

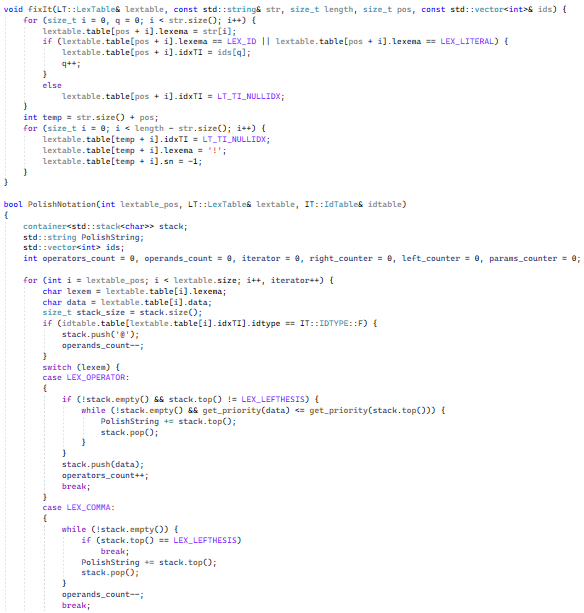


Рисунок 1 (продолжение) – Алгоритм преобразования выражения к польской записи

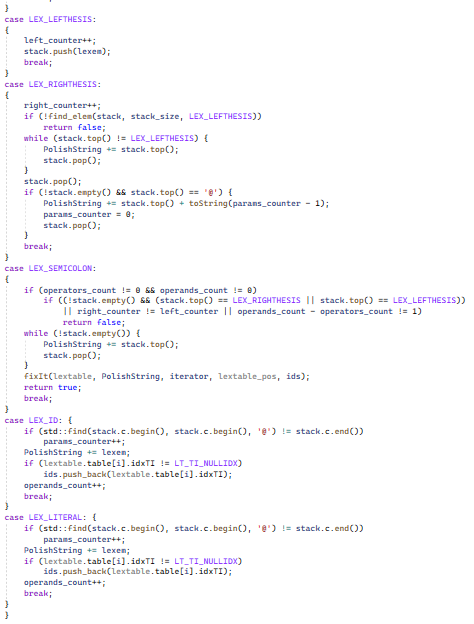


Рисунок 1 (продолжение) – Алгоритм преобразования выражения к польской записи

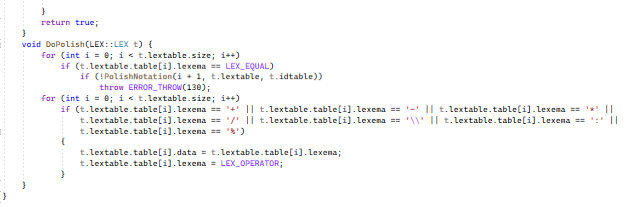


Рисунок 1 (конец) – Алгоритм преобразования выражения к польской записи

## ПРИЛОЖЕНИЕ Ж

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../Debug/TDS-2022LIB.lib  ExitProcess PROTO :DWORD  GetHours PROTO :DWORD  GetMonth PROTO :DWORD  GetMinutes PROTO :DWORD  GetDate PROTO :DWORD  outputuint PROTO :DWORD  outputchar PROTO :BYTE  outputstr PROTO :DWORD  .stack 4096  .const  divideOnZeroExeption BYTE "Попытка деления на ноль.", 0 ;STR, для вывода ошибки при делении на ноль  FindFactor$LEX1 DWORD 1 ;INT  main$LEX4 BYTE 'q' ;CHR  main$LEX5 BYTE "Symbol", 0 ;STR  main$LEX6 BYTE "Factorial of number 5", 0 ;STR  main$LEX7 DWORD 5 ;INT  main$LEX8 BYTE "Number to be circilar shifted:", 0 ;STR  main$LEX9 DWORD 32 ;INT  main$LEX10 DWORD 3 ;INT  main$LEX12 BYTE "32<<3:", 0 ;STR  main$LEX13 BYTE "32>>1:", 0 ;STR  main$LEX14 DWORD 6 ;INT  main$LEX16 BYTE "If construction works", 0 ;STR  main$LEX19 BYTE "If construction not works", 0 ;STR  main$LEX24 BYTE "Hours:", 0 ;STR  main$LEX25 BYTE "Minutes:", 0 ;STR  main$LEX26 BYTE "Date:", 0 ;STR  main$LEX27 BYTE "Month:", 0 ;STR  main$LEX28 DWORD 0 ;INT  .data  FindFactoranswer DWORD 0 ;INT  mainsymb BYTE 0 ;CHR  mainnumber DWORD 0 ;INT  maindemo DWORD 0 ;INT  maindemo1 DWORD 0 ;INT  maindemo2 DWORD 0 ;INT  mainnumber1 DWORD 0 ;INT  mainnumber2 DWORD 0 ;INT  mainnumber3 DWORD 0 ;INT  mainnumber4 DWORD 0 ;INT  .code  $FindFactor PROC uses ebx ecx edi esi , FindFactora: DWORD  ; String #3 :ivl  push FindFactor$LEX1  pop FindFactoranswer  While17Start:  mov eax, FindFactora  mov ebx, FindFactor$LEX1  cmp eax, ebx  jl While17End  ; String #6 :iviiv  push FindFactoranswer  push FindFactora  pop ebx  pop eax  mul ebx  push eax  pop FindFactoranswer  ; String #7 :ivilv  push FindFactora  push FindFactor$LEX1  pop ebx  pop eax  sub eax, ebx  push eax  pop FindFactora  jmp While17Start  While17End:  mov eax, FindFactoranswer  ret  $FindFactor ENDP  main PROC  ; String #19 :ivl  movzx eax, main$LEX4  push eax  pop eax  mov mainsymb, al  push offset main$LEX5  CALL outputstr  push eax  movzx eax, mainsymb  push eax  CALL outputchar  pop eax  push offset main$LEX6  CALL outputstr  ; String #25 :ivil@1  invoke $FindFactor, main$LEX7  push eax ;результат функции  pop mainnumber  push mainnumber  CALL outputuint  push offset main$LEX8  CALL outputstr  ; String #30 :ivl  push main$LEX9  pop maindemo  push maindemo  CALL outputuint  ; String #33 :ivilv  push maindemo  push main$LEX10  pop ebx  pop eax  push ecx ; сохраняем данные регистра ecx  mov ecx, ebx  SHL eax, cl  pop ecx  push eax  pop maindemo1  ; String #35 :ivilv  push maindemo  push FindFactor$LEX1  pop ebx  pop eax  push ecx ; сохраняем данные регистра ecx  mov ecx, ebx  SHR eax, cl  pop ecx  push eax  pop maindemo2  push offset main$LEX12  CALL outputstr  push maindemo1  CALL outputuint  push offset main$LEX13  CALL outputstr  push maindemo2  CALL outputuint  If158Start:  mov eax, main$LEX14  mov ebx, main$LEX7  cmp eax, ebx  jl If158End  push offset main$LEX16  CALL outputstr  If158End:  If169Start:  mov eax, main$LEX7  mov ebx, main$LEX14  cmp eax, ebx  jl If169End  push offset main$LEX19  CALL outputstr  If169End:  ; String #54 :ivil@1  invoke GetHours, FindFactor$LEX1  push eax ;результат функции  pop mainnumber1  ; String #55 :ivil@1  invoke GetMinutes, FindFactor$LEX1  push eax ;результат функции  pop mainnumber2  ; String #56 :ivil@1  invoke GetDate, FindFactor$LEX1  push eax ;результат функции  pop mainnumber3  ; String #57 :ivil@1  invoke GetMonth, FindFactor$LEX1  push eax ;результат функции  pop mainnumber4  push offset main$LEX24  CALL outputstr  push mainnumber1  CALL outputuint  push offset main$LEX25  CALL outputstr  push mainnumber2  CALL outputuint  push offset main$LEX26  CALL outputstr  push mainnumber3  CALL outputuint  push offset main$LEX27  CALL outputstr  push mainnumber4  CALL outputuint  mov eax, main$LEX28  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