#### Содержание

[Введение](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557373) 3

[1 Спецификация языка программирования](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557374) 4

[1.1 Характеристики языка программирования](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557375) 4

[1.2 Определение алфавита языка программирования](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557376) 4

[1.3 Применяемые сепараторы](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557377) 4

[1.4 Применяемые кодировки](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 5

[1.5 Типы данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 5

[1.6 Преобразование типов данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 6

[1.7 Идентификаторы](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 6

[1.8 Литералы](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 6

[1.9 Объявление данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 6

[1.10 Инициализация данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 6

[1.11 Инструкция языка](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 6

[1.12 Операции языка](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 7

[1.13 Выражения и их вычисление](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 7

[1.14 Конструкции языка](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 8

[1.15 Область видимости идентификаторов](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 8

[1.16 Семантические проверки](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 9

[1.17 Распределение оперативной памяти на этапе выполнения](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 9

[1.18 Стандартная библиотека и её состав](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 9

[1.19 Ввод и вывод данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 9

[1.20 Точка входа](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 10

[1.21 Препроцессор](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 11

[1.22 Соглашения о вызовах](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 11

[1.23 Объектный код](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 11

[1.24 Классификация сообщений транслятора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 11

[1.25 Контрольный пример](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557378) 11

[2 Структура транслятора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557383) 12

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557384) 12

[2.2 Перечень входных параметров](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557385) 13

[2.3 Протоколы, формируемые транслятором](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557386) 13

[3 Разработка лексического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557388) 14

[3.1 Структура лексического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557389) 14

[3.2 Контроль входных символов](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557390) 15

[3.3 Удаление избыточных символов](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557391) 15

[3.4 Перечень ключевых слов](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 16

[3.5 Основные структуры данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 17

[3.6 Структура и перечень сообщений лексического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 18

[3.7 Принцип обработки ошибок](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 18

[3.8 Параметры лексического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 18

[3.9 Алгоритм лексического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 18

[3.10 Контрольный пример](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557392) 19

[4 Разработка синтаксического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557395) 20

[4.1 Структура синтаксического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557396) 20

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557397) 20

[4.3 Построение конечного магазинного автомата](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557398) 21

[4.4 Основные структуры данных](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 21

[4.5 Описание алгоритма синтаксического разбора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 23

[4.6 Структура и перечень сообщений синтаксического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 24

[4.7 Параметры синтаксического анализатора и режимы его работы](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 24

[4.8 Принципы обработки ошибок](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 24

[4.9 Контрольные примеры](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 24

[5 Разработка семантического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557395) 25

[5.1 Структура семантического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557396) 25

[5.2 Функции семантического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557397) 25

[5.3 Структура и перечень сообщений семантического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557398) 25

[5.4 Принцип обработки ошибок](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 26

[5.5 Контрольный пример](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 26

[6 Вычисление выражений](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557395) 27

[6.1 Выражения, допускаемые языком](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557396) 27

[6.2 Польская запись и принцип её построения](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557397) 27

[6.3 Программная реализация обработки выражений](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557398) 28

[6.4 Контрольный пример](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 28

[7 Генерация кода](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557395) 29

[7.1 Структура генератора языка](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557396) 29

[7.2 Представление типов данных в оперативной памяти](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557397) 29

[7.3 Статическая библиотека](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557398) 30

[7.4 Особенности алгоритма генерации кода](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557399) 30

[7.5 Входные параметры, управляющие генерацией кода](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557398) 31

[7.6 Контрольный пример](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557398) 31

[8 Тестирование транслятора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557395) 32

[8.1 Структура синтаксического анализатора](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557396) 32

[8.2 Результаты тестирования](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557397) 32

[Заключение](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557400) 33

[Список литературных источников](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557409) 34

[ПРИЛОЖЕНИЕ А](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557426) 35

[ПРИЛОЖЕНИЕ Б](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557427) 38

[ПРИЛОЖЕНИЕ В](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557428) 55

[ПРИЛОЖЕНИЕ Г](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557429) 61

[ПРИЛОЖЕНИЕ Д](file:///C:\Users\User\Downloads\Курсовой_проект_Горощеня_ПОИТ4%20(2).docx#_Toc41557430) 65

#### Введение

Основной задачей данного курсового проекта является разработка собственного языка и транслятора к нему. Название языка – GVS-2022. Разработка транслятора будет вестись на языке программирования С++, а код на разработанном языке будет транслироваться в ассемблерный.

Задание курсового проекта делится на следующие этапы:

1. Разработка спецификации языка GVS-2022
2. Разработка лексического анализатора
3. Разработка синтаксического анализатора
4. Разработка синтаксического анализатора
5. Разбор вычислительных выражений
6. Разработка генератора кода
7. Тестирование транслятора

#### 1 Спецификация языка программирования

#### 1.1 Характеристика языка программирования

Предназначение языка GVS – 2022 – обучение азам программирования людей, которые хотят начать свои первые шаги в IT-сфере. Поэтому данный язык является процедурным, строго типизированным, компилируемым и не поддерживающим объектно-ориентированное программирование.

#### 1.2 Алфавит языка программирования

В качестве алфавита языка GVS – 2022 будет выступать кодировка ASCII, поскольку английский язык – один из самых распространённых лингвистических языков в мире. (БНФ для алфавита)

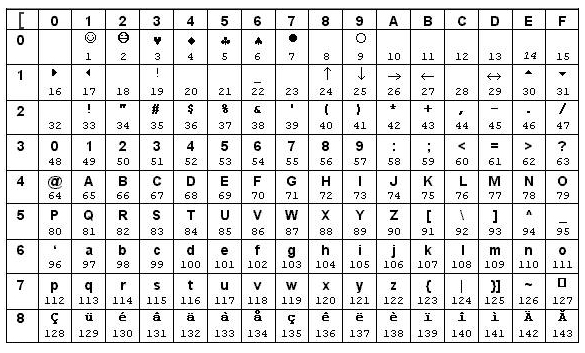


Рисунок 1.1 Кодировка ASCII

Используемые символы на этапе выполнения: [a…z], [A...Z], [0…9], а также спецсимволы {},.()=[]+-\*/.

#### 1.3 Применяемые сепараторы

Сепараторы служат для разделения символьных цепочек в ходе обработки исходного текста на токены. Токен – замещение объекта каким – либо идентификатором. В таблице 1.1 приведены сепараторы и их назначение.

Таблица 1.1 Символы - сепараторы

|  |  |
| --- | --- |
| Сепараторы | Назначение |
| {…} | Тело функции, условия и цикла |
| (…) | Параметры функции, приоритет при обработке численных выражений, параметры условия и цикла |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| Сепараторы | Назначение |
| ‘ ‘ (пробел) | Разделитель цепочек, в именах идентификаторов и ключевых слов пользоваться нельзя |
| , | Разделитель параметров функции |
| . | Используется при записи чисел с плавающей точкой |
| = | Присваивание |
| / | Деление нацело |
| % | Остаток от деления |
| \* | Умножение |
| + | Сложение |
| - | Вычитание |

#### 1.4 Применяемые кодировки

Как уже было упомянуто, в качестве кодировки язык GVS – 2022 использует Американскую стандартную кодировку для обмена информацией, она же ASCII. Данная кодировка отличается простотой, что соответствует его назначению. Также, она состоит из латинских символов, что позволит распространению языка не только в Беларуси, но и по всему миру.

#### 1.5 Типы данных

В языке GVS – 2022 приведены три типа данных: целочисленный, символьный, число с плавающей точкой. Их описание приведено в таблице 1.2.

Таблица 1.2. Описание типов данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| Целочисленный тип integer | Фундаментальный тип данных, занимающий в памяти 4 байта. Используется для работы с целыми числами.  Минимальное значение: 65355  Максимальное значение: -65356  Поддерживает сложение, вычитание, умножение, деление, побитовые и логические операции. |
| Символьный/строковый тип symbol | Фундаментальный тип данных, занимающий 1 байт в памяти в случае, если его длина занимает 1, в остальных – 1\*N, где N – число символов, заключённых в кавычки. Используется для работы с символами/строками. |

#### 1.6 Преобразование типов данных

Поскольку GVS – 2022 является строго типизированным, то преобразование типов он не предусматривает.

#### 1.7 Идентификаторы

Идентификаторы предназначены для определения имени переменной, типа или члена. В GVS – 2022 они могут включать в себя латинские буквы верхнего и нижнего регистра, так и цифры. Максимальная длина – шестнадцать символов. +БНФ

#### 1.8 Литералы

Литералы позволяют инициализировать переменные. Например, целочисленные литералы могут объявляться последовательностью цифр от 0 до 9, в случае если число отрицательное – перед ним ставится знак минус, не отделяясь пробелом. Символьный тип заключается в одинарные кавычки, строка задаётся в виде массива символов. Шестнадцатеричное целое число начинается с символа х, после чего следует последовательность цифр от 0 до 9 или символов от A до F. В то же время имеются ограничения. Внутри символьного литерала не используются одинарные или двойные кавычки, а также не допускается объявления символов, не входящих в кодировку ASCII. Для целочисленных в случае равенства литерала 0, знак минус перед ним не ставится.

Число в БНФ: [-][0...9]|x[0…9]

Символ в БНФ: “[0…9]|[A…Z]|[a…z]|сепараторы”

#### 1.9 Объявление данных

Данные объявляются следующим образом:

create <тип><имя\_переменной>=<значение>

#### 1.10 Инициализация данных

При инициализации данных переменной будет присвоено значение, находящееся справа от знака равенства. Инициализаторами являются идентификаторы и литералы. Присвоение значения по умолчанию не предусмотрено, пользователь обязан его указывать.

#### 1.11 Инструкции языка

Инструкции языка в последовательности формируют программу, ибо это наименьшая автономная часть языка программирования. Она служит для исполнения кода.

Инструкции GVS – 2022 приведены в таблице 1.3.

Таблица 1.3. Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Её реализация |
| Объявление переменной | <тип> <имя\_переменной>=<значение> |
| Объявление переменной с явной инициализацией | <тип> <имя\_переменной>=<значение> |
| Возврат значения из функции | bring <идентификатор|литерал> |
| Вывод данных | output < идентификатор|литерал > |
| Объявление функции | <тип> func <идентификатор>(<список параметров>) |
| Перевод строки | Jump |
| Присваивание | <идентификатор>=<выражение>, в качестве выражения может быть и результат, возвращаемый функцией |

#### 1.12 Операции языка

В языке GVS – 2022 предусмотрены три вида операций: арифметические, побитовые, побитовые и логические. Каждая из операций обладает приоритетом. Круглые скобки используются для повышения приоритета. Операции приведены в таблице 1.5.

Таблица 1.5. Операции языка GVS – 2022

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | + - сложение (приоритет - 2)  - - вычитание (приоритет - 2)  \* - умножение (приоритет - 3)  / - деление нацело (приоритет - 3)  % - остаток от деления (приоритет - 3)  = - присваивание |
| Логические | < - больше (приоритет - 0)  > - меньше (приоритет - 0) |
| Побитовые | | – ИЛИ (приоритет - 3)  & – И (приоритет - 3)  ? – ИНВЕРСИЯ (приоритет - 3) |

#### 1.13 Вычисление выражений

Вычислить выражение – значит выполнить все арифметические действия, указанные в нём, по определённым правилам:

1. Скобки позволяют изменять приоритет операции
2. Выражение записывается в одну строку, без переносов
3. Не допускается следование одного оператора за другим
4. В выражении допускается вызов функции, возвращающей численное значение.

В дальнейшем, выражение преобразуется в форму обратной польской записи для облегчения дальнейшего вычисления выражения на ассемблере, а после его генерации.

#### 1.14 Конструкции языка

Код программы на GVS – 2022 обязательно включает в себя главную функцию, но также может содержать и функции, разработанные пользователем. Для улучшения читаемости и понятности кода необходимо отделять конструкции блоками и рекомендуется делать отступы. Конструкции языка представлены в таблице 1.6.

Таблица 1.6. Конструкции языка

|  |  |
| --- | --- |
| Конструкция | Её реализация |
| Главная функция | start{  …  } |
| Внешняя функция | <тип>func<имя\_функции>(<тип><идентификатор>,…){  …  }  Если функция ничего не возвращает, то тип не указывается |
| Цикл | until(<идентификатор><оператор><значение2>){  …  } |
| Условный блок | if(<идентификатор><оператор><значение|идентификатор>) …  Otherwise …  Блок if автоматически имеет значение true. В противном случае дописывается блок otherwise**.** |

#### 1.15 Области видимости идентификатора

Область видимости схожа с С++. Переменные, объявленные в одной области видимости, недоступны в другой, если это локальная, а не глобальная переменная. Параметры видны только внутри функции, где они объявлены.

##### **1.16 Семантические проверки**

В языке GVS – 2022 будут осуществляться следующие проверки:

1. Проверки на наличие входной точки **start**, и её единственность в коде
2. Переопределение идентификаторов и наличие их в коде, в нужной области видимости.
3. Проверка на тип функции и тип возвращающего значения
4. Правильность передаваемых параметров: типы, количество
5. Правильность структуры конструкций языка

#### 1.17 Распределение памяти

Для записи литерал используется сегмент констант, также в данный сегмент можно записывать идентификаторы. Сегмент данных используется для записи идентификаторов, функции и параметры функций.

#### 1.18 Стандартная библиотека

В GVS – 2022 имеется стандартная библиотека, включающая в себя две функции: возведения в степень и модуля. Она подключается автоматически при трансляции в язык ассемблера. Содержимое библиотеки и его описание в таблице 1.7.

Таблица 1.7. Содержимое библиотеки

|  |  |
| --- | --- |
| Функция стандартной библиотеки | Описание |
| integer module(int val) | Численная функция, принимающая на вход число типа integer. Возвращает число по модулю. |
| integer raise(int val, int num) | Численная функция, принимающая на вход числа типа integer. Первое значение – значение, второе - степень. Возвращает значение, возведённое в степень. |

Стандартная библиотека написана на языке С++. Она подключается к транслируемому коду на этапе генерации. Их вызов доступен там же, где пользовательские функции. +преобразование

#### 1.19 Ввод и вывод данных

Вывод данных осуществляется с помощью ключевого слова output. Синтаксис данной команды выглядит следующим образом:

Output <имя\_переменной|идентификатор|литерал>

#### 1.20 Точка входа

Как уже было упомянуто, главной точкой входа программы является функция start. Она не имеет типа и является единственной. Именно с этой конструкции и начинается последовательное выполнение команд программы.

#### 1.21 Препроцессор

В языке GVS – 2022 препроцессоры не предусмотрены.

#### 1.22 Соглашение о вызовах

В качестве соглашений о вызове применяется соглашение о вызовах stdcall. Это одно из самых удобных и широко распространённых соглашений о вызовах. Особенностями данного соглашения являются:

1. Параметры размещены в стеке справа – налево
2. Стек чистится вызываемой функцией
3. Все параметры также записываются через стек
4. В дизассемблерном коде, в последней инструкции вызываемого кода RET указывается значение суммарного размера всех параметров в байтах, прибавляясь к ESP, и при этом возвращаемое значение передаётся через EAX.

#### 1.23 Объектный код

Язык GVS – 2022, как уже было упомянуто, транслируется в ассемблер, а из него в объектный код.

#### 1.24 Классификация сообщений транслятора

Сообщения, генерируемые транслятором, должны давать подробную информацию, чтобы пользователь понял, где и какие имеются недочёты и ошибки в написанном им коде. Сообщения транслятора и их характеристики приведены в таблице 1.9.

Таблица 1.9. Сообщения транслятора и их характеристики

|  |  |
| --- | --- |
| Номера | Характеристика |
| 0 – 200 | Системные ошибки |
| 201 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |
| 400 – 799 | Прочие ошибки |

#### 1.25 Контрольный пример

Контрольный пример с демонстрацией главных особенностей GVS – 2022 находится в приложении A.

#### 2 Структура транслятора

#### 2.1 Компоненты транслятора, их назначение и принцип взаимодействия

В языке GVS-2022 исходный код транслируется в Assembler. Сам же транслятор имеет несколько составляющих, взаимодействующих между собой и выполняющие определённые функции. Для получения ассемблерного кода используются выходные данные работы транслятора: таблица лексем и таблица идентификаторов. Схема транслятора представлена на рисунке 2.1



Рисунок 2.1 Схема транслятора

Первой стадией является лексический анализ, реализующийся с помощью программы, называемой лексическим анализатором. На вход поступает последовательность символов входного языка. При этом производится разбор текста, и единый массив текстовых символов преобразуется в «токены». К лексическим единицам относятся такие конструкции как идентификаторы, числа, символы операций, служебные слова. При преобразовании текста лексические единицы заменяются лексемами, для создания промежуточного представления программы. При этом, каждой лексеме в таблице в соответствие ставятся её тип и запись в таблице идентификаторов, где имеется дополнительная информация. Соответственно, результаты работы первой стадии являются входом во вторую, третью и последнюю.

Вторая стадия – синтаксический анализ. Он позволяет проверить исходный код на соответствие правилам грамматики. На вход синтаксический анализатор (программа, выполняющая синтаксический анализ) принимает таблицы лексем и идентификаторов. Выходная информация – дерево разбора.

Третья стадия – семантический анализ. Он проверяет программу на наличие ошибок, не отслеживающихся при помощи регулярной и контекстно – свободной грамматик. Как уже было отмечено, на вход семантический анализатор принимает таблицы лексем и идентификаторов.

Последняя стадия – генератор. Это часть транслятора, выполняющая генерацию ассемблерного кода на основе результатов работы предыдущих этапов. На вход подаются таблицы лексем и идентификаторов, а на выходе получаем сгенерированный на их основе ассемблерный файл.

#### 2.2 Перечень входных параметров

Для формирования файлов, хранящих результаты трёх этапов, используется следующие параметры, приведённые в таблице 2.1.

Таблица 2.1. Входные параметры

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<path\_to\_in-file> | Файл с исходным кодом на языке GVS-2022 , имеющий расширение .txt | Отсутствует |
| -log:<path\_to\_log-file > | Файл лога для вывода протоколов работы программы. | <in-file name>.log |

#### 2.3 Протоколы, формируемые транслятором

В ходе работы лексического, синтаксического и семантического анализаторов формируются протоколы работы, содержащие в себе перечень протоколов. Формируемые протоколы приведены в таблице 2.2

|  |  |
| --- | --- |
| Входной параметр | Описание |
| Файл лога, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования GVS-2022 . Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

#### 3 Разработка лексического анализатора

#### 3.1 Структура лексического анализатора

Лексический анализатор – программа осуществляющая лексический анализ. На вход она принимает последовательность символов языка, а на выходе даёт таблицы лексем и идентификаторов. В качестве лексических единиц выступают идентификаторы, числа, символы операций и так далее. Они преобразуются в лексемы, причём каждой из них в соответствие ставится тип, запись и прочая дополнительная запись.

Функции лексического анализатора:

1. Удаление пустых символов, комментариев.
2. Распознавание идентификаторов и ключевых слов.
3. Распознавание констант
4. Распознавание разделителей и знаков операций.

Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1. Структура лексического анализатора

#### 3.2 Контроль входных символов

Для упрощения работы с кодом при его передаче в лексический анализатор, все символы делятся по категориям. Категории представлены на рисунке 3.2 и таблице 3.1.

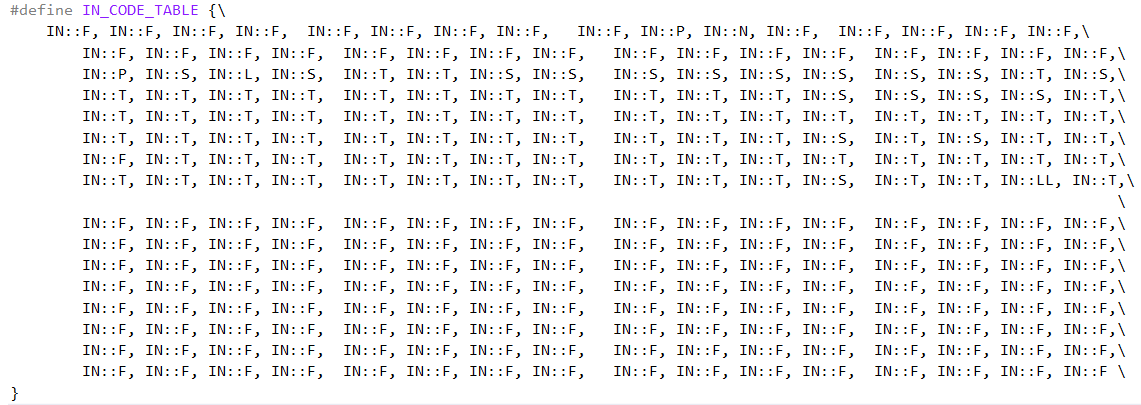


Рисунок 3.2 Таблица входных символов

Таблица 3.1 Соответствие символов и значения в таблице

|  |  |
| --- | --- |
| Значение в таблице | Описание |
| T | Разрешённые символы |
| F | Запрещённые символы |
| I | Игнорируемые символы |
| L | Одинарные кавычки |
| LL | Двойные кавычки |
| E | Пустой символ |
| S | Сепаратор |

#### 3.3 Удаление избыточных символов

В качестве избыточных символов у нас выступают пробелы и табуляции. Их удаление осуществляется на этапе разбиения исходного кода на токены. Для этого файл с исходным кодом посимвольно считывается, если встречаем пробел или табуляцию, являющихся по своей сути символами – сепараторами. Но в отличии от сепараторов, они игнорируются.

#### 3.4 Перечень ключевых слов

При лексическом анализе исходный код преобразуется, заменяя лексические единицы лексемами. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2. Соответствие токенов и лексем

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| integer, string, symbol | t | Типы данных языка |
| Идентификатор | i | Имя переменной, максимальная длина - 5 |
| Функция | f | Функция |
| Литерал | l | Литерал любого из доступных в языке типов |
| bring | b | Возвращает значение |
| Start | s | Точка входа |
| create | c | Ключевое слово для объявления переменной |
| output | o | Вывод |
| jump | ^ | Переход на новую строку |
| if | y | Начало условной конструкции |
| until | u | Цикл |
| otherwise | e | Ложная ветвь условной конструкции |
| ; | ; | Разделитель выражений |
| , | , | Разделитель аргументов |
| { | { | Начало тела функции |
| } | } | Конец тела функции |
| ( | ( | Начало передачи параметров в функцию, приоритет операций, а также выражение цикла/условия |
| ) | ) | Конец передачи параметров в функцию, приоритет операций, а также выражение цикла/условия |
| = | = | Присваивание значения |
| +  -  /  \*  &  |  ? | v | Арифметические, побитовые операции |

#### 3.5 Основные структуры данных

Основные структуры данных: таблица лексем и таблица идентификаторов. Таблица лексем содержит лексему (lexema), полученную при разборе, номер строки в исходном коде и номер в таблице идентификаторов. Таблица идентификаторов содержит в себе имя, номер в таблице лексем, тип данных и тип идентификатора, а также (если имеется) значение. Коды таблицы идентификаторов и таблицы лексем представлены на рисунках 3.3 и 3.4.

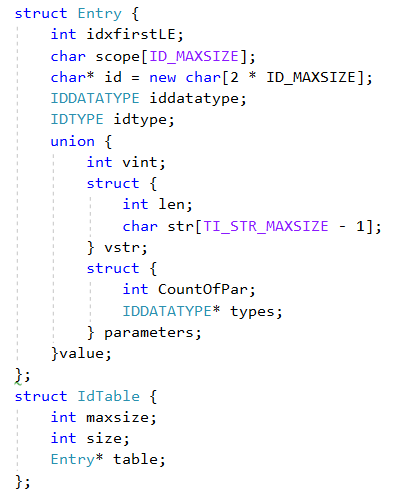


Рисунок 3.3 Структура таблицы идентификаторов

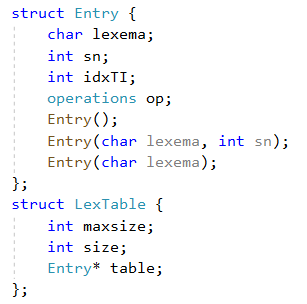


Рисунок 3.4. Структура таблицы лексем

#### 3.6 Структура и перечень сообщений лексического анализатора

Возникшие в процессе работы ошибки фиксируются в файле протоколирования, заданный в аргументах командной строки. При этом ошибки протоколируются с номером ошибки и соответствующим сообщением.

#### 3.7 Принцип обработки ошибок

Для обработки ошибок, лексический анализатор использует специальную таблицу с ошибками, содержащую в себе номер ошибки и текст с сообщением, и дополнительно номер строки и позиции, где она была встречена. Если ошибка встречается, то лексический анализатор вновь продолжает свою работу с исходным кодом. Структура сообщений лексического анализатора приведена на рисунке 3.5.

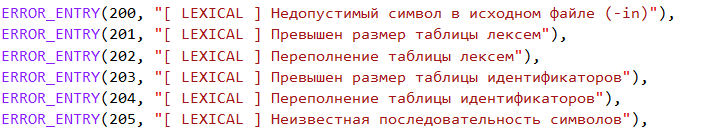


Рисунок 3.5. Структура ошибок лексического анализатора

#### 3.8 Параметры лексического анализатора

В качестве параметров лексический анализатор принимает таблицы лексем и идентификаторов – результат его работы. Они выводятся как файл лога, так и в командную строку.

#### 3.9 Алгоритм лексического анализатора

Алгоритм работы лексического анализатора выглядит следующим образом:

1. Проверка входного потока символов на исходном языке на их допустимость, удаление лишних пробелов, добавление сепаратора для определения номера строки.
2. Распознавание лексемы.
3. Если распознавание прошло успешно, то она заносится в таблицы идентификаторов и таблицы лексем и возвращается на первый этап. В противном случае, выдаёт сообщение об ошибке.
4. Формирование протокола работы.

Важно отметить, что распознавание лексемы ведётся на основе конечных автоматов, чью работу можно легко проиллюстрировать с помощью графа переходов.

Пример графа переходов показан на рисунке 3.6.

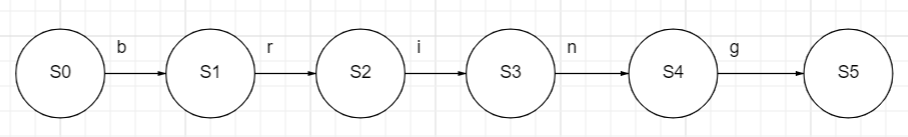


Рисунок 3.6 Граф переходов для ключевого слова bring

#### 3.10 Контрольный пример

Таблицы лексем и идентификаторов приведены в приложении Б.

#### 4 Синтаксический анализ

#### Структура синтаксического анализатора

Синтаксический анализ – часть программы, проверяющая проверку исходного кода на соответствие правилам грамматики. В качестве входного параметра она принимает таблицы лексем и идентификаторов, а на выход выдаёт дерево разбора. Подробное описание структуры синтаксического анализатора представлено на рисунке 4.1.



Рисунок 4.1. Структура синтаксического анализатора

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

Для осуществления синтаксического анализа язык GVS – 2022 использует контекстно – свободную, вторую по иерархии Хомского, грамматику. Она имеет вид , где

* T – набор множества терминальных символов.
* N – набор нетерминальных символов.
* P – множество правил языка.
* S – начальный символ, являющийся нетерминалом.

Поскольку грамматика не является леворекурсивной, то она имеет нормальную форму Грейбах, для которой определены следующие правила P:

1. , где или .
2. , где – начальный символ. Данный символ не встречается в правой части правил, если такое правило имеет место быть.

Описание нетерминалов представлено в таблице 4.2.

#### Построение конечного магазинного автомата

Конечный магазинный автомат – математическая абстракция, представляющая собой семёрку , имеющую один вход и один выход, а также находящуюся в одном из состояний в каждый момент времени. Подробное описание компонентов автомата представлено в таблице 4.1.

Таблица 4.1. Компоненты автомата с магазинной памятью

|  |  |  |
| --- | --- | --- |
| Компонент | Описание | Разъяснение |
| Q | Множество состояний автомата | Состояние автомата – структура, содержащая позицию во входной ленте, номер текущего правила, цепочки и его стек |
| V | Алфавит входных символов | Множество терминальных и нетерминальных символов |
| Z | Алфавит специальных магазинных символов | Множество, включающее в себя начальный символ и маркер дна стека $ |
|  | Функция переходов автомата | Множество правил грамматики Грейбах |
|  | Начальное состояние автомата | Состояние, которое автомат имеет в начале своей работы. Представляется в виде стартового правила грамматики |
|  | Начальное состояние магазина автомата | Символ маркера дна стека |
|  | Множество конечных состояний | Состояние, позволяющие автомату завершить свою работу, то есть магазин пуст. |

#### 4.4 Основные структуры синтаксического анализатора

Основными структурами данных синтаксического анализатора являются структура магазинного автомата, выполняющей разбор входящей ленты, и структура правил грамматики Грейбах. Правила грамматики приведены в таблице 4.2.

Таблица 4.2. Правила грамматики Грейбах

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Пояснение |
| S | S->tfiFBS  S->s{N}  S->tfiFB | Стартовое состояние с описанием общей структуры программы |
| F | F->(P)  F->() | Правило параметров функции |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| P | P->ti  P->ti,F | Правила списка параметров функции |
| B | B->{NbI;}  B->{bI;}  B->{N} | Правила для тела функции |
| I | I->i  I->l | Правила для выражения |
| U | U->{N} | Правила для тела функции или цикла |
| N | N->qti;N  N->qti=E;N  N->i=E;N  N->u(R){X}N  N->y(R){X}N  N->y(R){X}n{X}N  N->oI;N;  N->^;N  N->bE;N  N->iK;N  N->qti;  N->qti=E;  N->i=E;  N->u(R){X}  N->y(R){X}  N->y(R){X}n{X}  N->oI;  N->^;  N->bE;  N->iK; | Правила для конструкций в теле функции |
| R | R->i  R->ivi  R->ivl  R->lvi | Правила для условного выражения |
| K | K->(W)  K->() | Правила для вызова функции |
| E | E->i  E->l  E->(E)  E->iK  E->iM  E->lM  E->(E)M  E->iKM | Правила для арифметических выражений |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| W | W->i  W->l  W->i,W  W->l,W | Правила для списка параметров при вызове функции |
| M | M->vE  M->vEM | Правила для арифметического выражения |
| X | X->qti;N  X->qti=E;N  X->i=E;N  X->oI;N  X->^;N  X->bE;N  X->iK;N  X->qti;  X->qti=E;  X->i=E;  X->oI;  X->^;  X->bE;  X->iK; | Правила для тела условия или цикла |

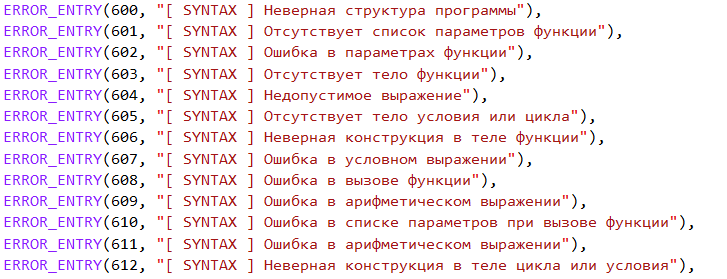
#### 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора кода на исходном языке имеет следующий вид:

1. Запуск стартового символа в магазин.
2. Формирование входной ленты на основе результатов работы лексического анализатора.
3. Запуск автомата
4. Выбор цепочки, соответствующей нетерминальному символу, запись в магазин в обратном порядке.
5. При совпадении терминалов в стеке и ленте, терминал удаляется из ленты из стека. В противном случае, переход в предыдущее состояние и выбор другой цепочки нетерминала.
6. При встрече нетерминала переход снова на 4 этап.
7. При достижении символом дна стека и полной очистке ленты синтаксический разбор выполнился успешно, иначе генерируется исключение.

#### Структура и перечень сообщений синтаксического анализатора

Также, как и структура сообщений лексического анализатора, таблица ошибок включает в себя номер ошибки и её текст, а также номер строки и позиции в ней. Структура представлена на рисунке 4.2.



#### Параметры синтаксического анализатора и принцип его работы

Входными параметрами синтаксического анализатора являются таблицы лексем и идентификаторов, результатом же работы является дерево разбора. Помимо этого, протокол работы автомата с магазинной памятью записывается в файл логирования.

#### Принцип обработки ошибок

Синтаксический анализ выполняется до тех пор, пока не символ ленты не дойдёт до дна стека или не найдётся исключение. В этом случае, выводится сообщение об ошибке, и если таких случаев 3, то анализатор прекращает свою работу.

## **Контрольный пример**

Контрольный пример приведён в приложении В.

# **5 Семантический анализ**

# **Структура семантического анализатора**

Семантический анализатор – программа, выполняющая семантический анализ, то есть, на основе результатов работы лексического, таблиц идентификаторов и лексем, и синтаксического, дерева разбора, и ищет необходимые ошибки. Многие проверки, такие как проверка на единственность точки входа или предварительное объявление переменной, осуществляются на этапе лексического анализа, но большая часть работы всё же лежит на лексическом анализаторе. Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. Структура семантического анализатора

# **Функции семантического анализатора**

Функционал семантического анализатора представлен в разделе 1.16

# **Структура и перечень сообщений семантического анализатора**

Структура сообщений семантического анализатора абсолютно идентична двум предыдущим случаям. Она представлена на рисунке 4.2.

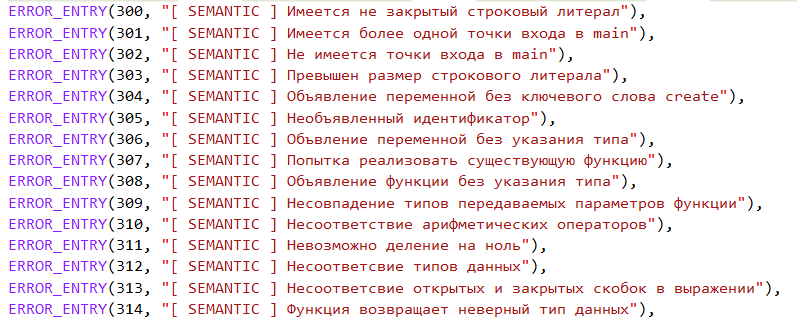


Рисунок 4.2 Структура ошибок семантического анализатора

# **Принцип обработки ошибок**

Ошибки при семантическом анализе фиксируются в протоколе, с учётом её номера и диагностического сообщения. Процесс завершается после нахождения всех ошибок.

# **Контрольный пример**

Контрольный пример приведён в разделе 8.1.

# **6 Вычисление выражений**

# **6.1 Выражения, допускаемые языком**

Язык GVS-2022 допускает вычисление выражений целочисленного типа и поддерживает вызов функций в выражение. Выражение включает в себя операторы, которые указывают на порядок его вычисления. Этот порядок определяется приоритетами. Чем выше приоритет, тем раньше будет выполняться подвыражение, если выражение имеет несколько операторов. Таблица приоритетов операторов приведена в таблице 6.1.

Таблица 6.1. Таблица приоритетов

|  |  |
| --- | --- |
| Оператор | Приоритет |
| ( | 0 |
| ) | 0 |
| , | 1 |
| + | 2 |
| - | 2 |
| \* | 3 |
| / | 3 |
| & | 3 |
| | | 3 |
| ? | 3 |

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

Все выражения в языке преобразуются в польскую обратную запись, или кратко, ПОЛИЗ.

ПОЛИЗ представляет собой альтернативную запись арифметических выражений, являющейся более простой для распознавания выражения ассемблером. Самыми распространёнными её видами являются постфиксная и префиксная формы. В постфиксной знаки операторов записываются перед аргументами, в префиксной – после. Также следует подчеркнуть, что ПОЛИЗ удобна тем, что в ней отсутствуют скобки.

Основным алгоритмом построения польской записи является алгоритм Дейкстры, он же алгоритм сортировочной станции. Он выглядит следующим образом:

* Исходная строка – выражение.
* Выходная: преобразованная строка, полученная в результате ПОЛИЗ
* Стек: пустой.
* Просмотр строки слева направо.
* Операнды переносятся в строку.
* Операция записывается в стек, если тот пуст.
* Если встречается операция с большим или равным приоритетом, то операции в стеке выталкиваются в результирующую строку.
* Если встречается разделитель аргументов функции, то он в стек не помещается.
* Открывающая скобка помещается в стек.
* При встрече с закрывающей, содержимое стека выталкивается в строку. Если определённые операнды являлись аргументами функции, то генерируется последовательность @n, где n – число параметров функции, и она помещается в результирующую строку.
* По концу разбора все операции, находящиеся в этот момент в стеке, выталкиваются в строку.

# **6.3 Программная реализация обработки выражения**

Реализация обработки выражения приведена в приложении Г.

# **6.4 Контрольный пример**

Контрольный пример приведён в приложении Г.

# **7 Генерация кода**

# **Структура генератора кода**

Генерация кода – заключительный этап работы трансляции. Принимая на вход таблицы лексем и идентификаторов, он строит выходной файл на основе языка ассемблера, результатом работы транслятора, ведь транслятор – программа для преобразования исходного кода на одном языке в язык машинного уровня. Генерация не будет осуществляться, если предыдущие этапы работы завершаться с ошибками. Структура генератора приведена на рисунке 7.1.

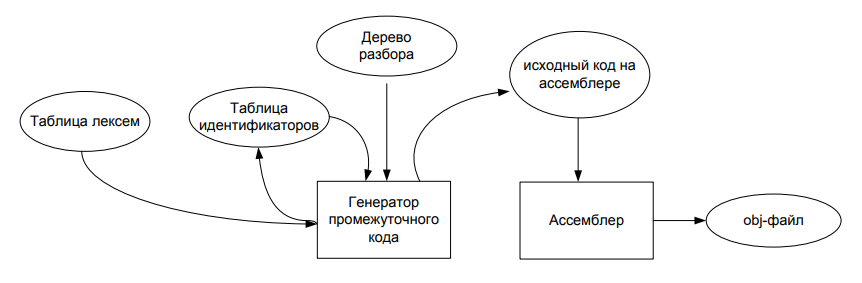


Рисунок 7.1 Структура генератора кода

# **Представление типов в памяти**

При генерации на ассемблер, элементы таблицы идентификаторов располагаются в сегментах данных (.data) и констант (.const). Описание представления типов в оперативной памяти приведено в таблице 7.1.

Таблица 7.1 Представление типов данных в оперативной памяти

|  |  |  |
| --- | --- | --- |
| Тип | Тип идентификатора в языке ассемблера | Пояснение |
| Integer | dword | Хранит целочисленный тип данных, число байт integer и dword одинаковое |
| symbol, string | byte | Хранит символ или строку, byte занимает 1 байт |

# **Статическая библиотека**

Статическая библиотека представляет собой файл с функциями, написанные на языке С++ и автоматически генерируемые в языке ассемблера. Список функций статической библиотеки приведён в таблице 7.2.

Таблица 7.2. Список функций статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| int raise(int a, int b) | Функция возведения в степень |
| int module(int a) | Функция, возвращающая модуль числа |
| int outputint(int a) | Функция вывода числа |
| int outputstr(char\*value) | Функция вывода строки |
| int outputsym(char value) | Функция вывода символа |

# **Особенности алгоритма генерации кода**

Генерация кода осуществляется на основе результатов работы лексического анализатора, на выходе получается asm - файл, являющийся аргументом командной строки проекта или же входным параметром самого транслятора. Блок схема работы генерации приведена на рисунке 7.2.

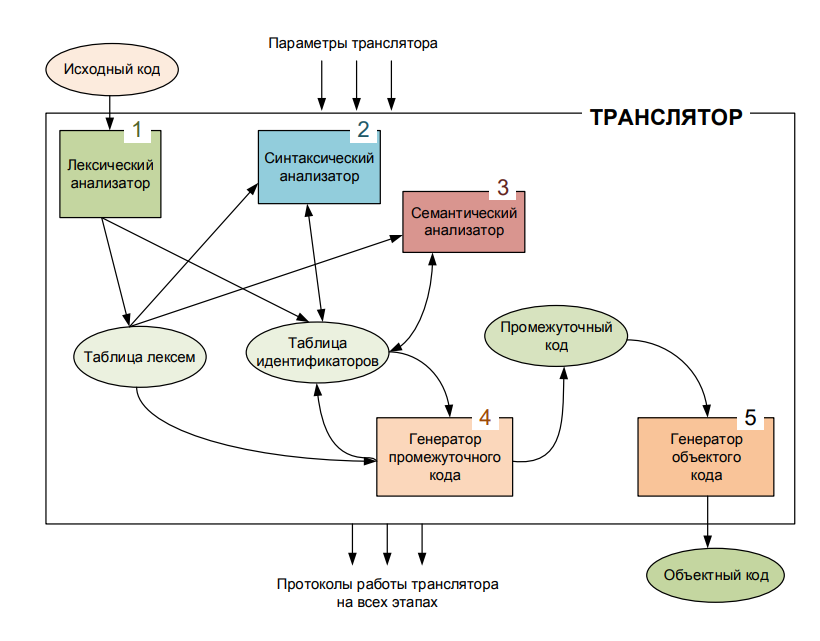


Рисунок 7.2 Структура работы генерации кода

# **Входные параметры генератора кода**

Входными параметрами генератора – таблицы идентификатора и лексем, полученные в результате лексического анализа. Подробное описание приведено в разделе 7.1.

# **Контрольный пример**

Контрольный пример приведён на рисунке 7.3

# **8 Тестирование транслятора**

# **8.1 Общие положения**

Если транслятор обнаруживает ошибку, то она выводится в консоль, а также записывается в файл логирования. Сама ошибка обрабатывается в главном файле программы.

# **8.2 Результаты тестирования**

Результаты тестирования приведены в таблице 8.1.

Таблица 8.1. Результаты тестирования

|  |  |
| --- | --- |
| Код | Результат |
| {  create integer z=1;  output z;  } | Ошибка 302: [Semantic]: Отсутствует точка входа main |
| start{  integer k=8;  } | Ошибка 304: [Semantic]: Объявление переменной без ключевого слова create, Строка 2 |
| Старт | Ошибка 200: [Lexical]: Недопустимый символ в исходном файле (-in) |

# **Заключение**

В ходе выполнения курсового проекта был разработан транслятор для языка программирования GVS – 2022. Были выполнены все необходимые требования, включающие в себя:

* Формулировка спецификации языка GVS – 2022.
* Разработка лексического, синтаксического и семантического анализаторов.
* Разработка контекстно – свободной грамматики, приведённой к форме Грейбах.
* Разработка генератора кода на ассемблер.
* Проведение тестирование транслятора.

Конечная версия языка GVS – 2022 состоит из:

* 3 типа данных
* Поддержка операторов вывода
* Вызов стандартной библиотеки
* Наличие арифметических, побитовых операторов для вычисления выражений.
* Поддержка функций, процедур, цикла и условных конструкций.
* Структурная система для обработки ошибок пользователя.

Таким образом, данная работа позволила нам получить представление о работе транслятора и языков программирования в целом, показала нам сильные и слабые стороны многих средств реализации.

# **Список использованных источников**

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.
3. Вирт Н. Построение компиляторов/ Пер. с англ. Борисов Е. В., Чернышов Л.Н. – М.: ДМК Пресс, 2010 – 192 с.
4. Скляров И. Изучаем Ассемблер за 7 дней/ http://www.sklyaroff.ru
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

# **Приложение А**

Исходный код программы

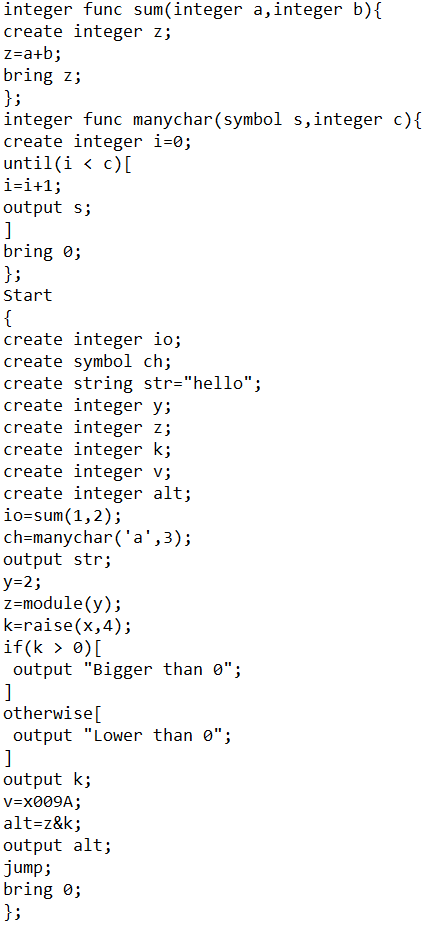
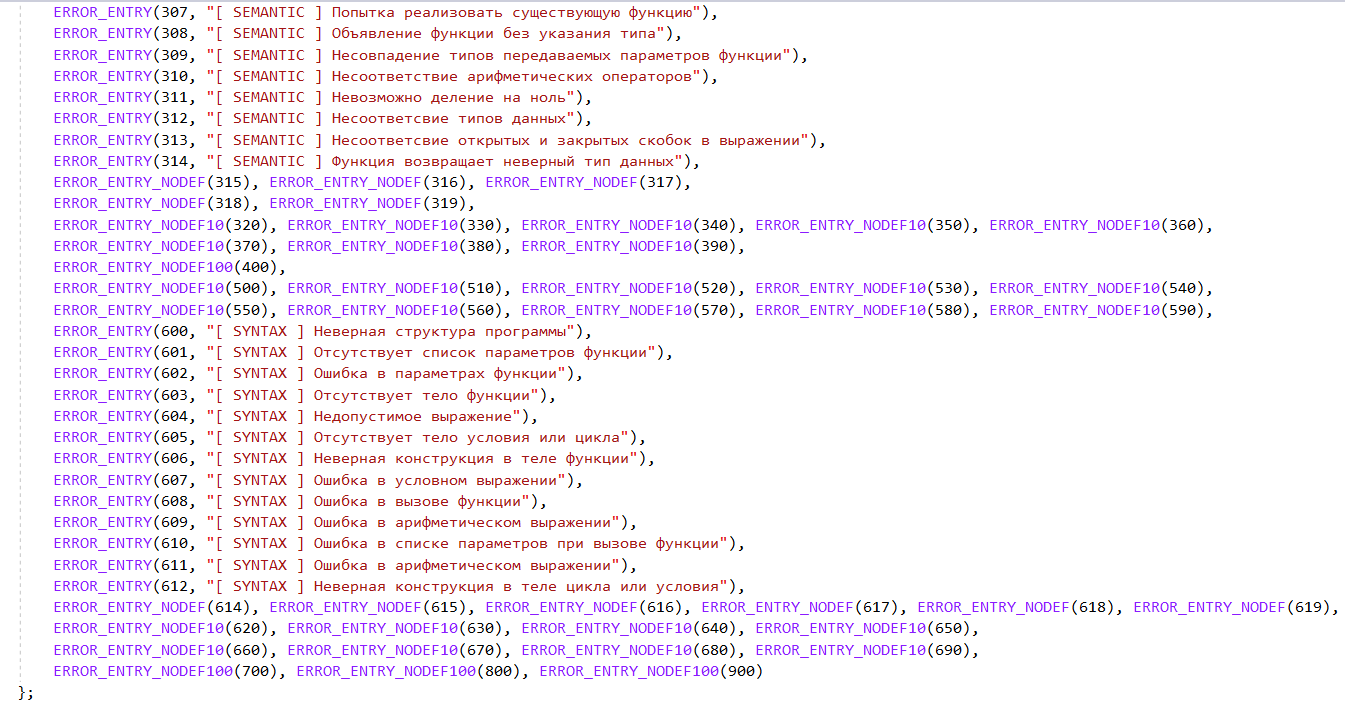
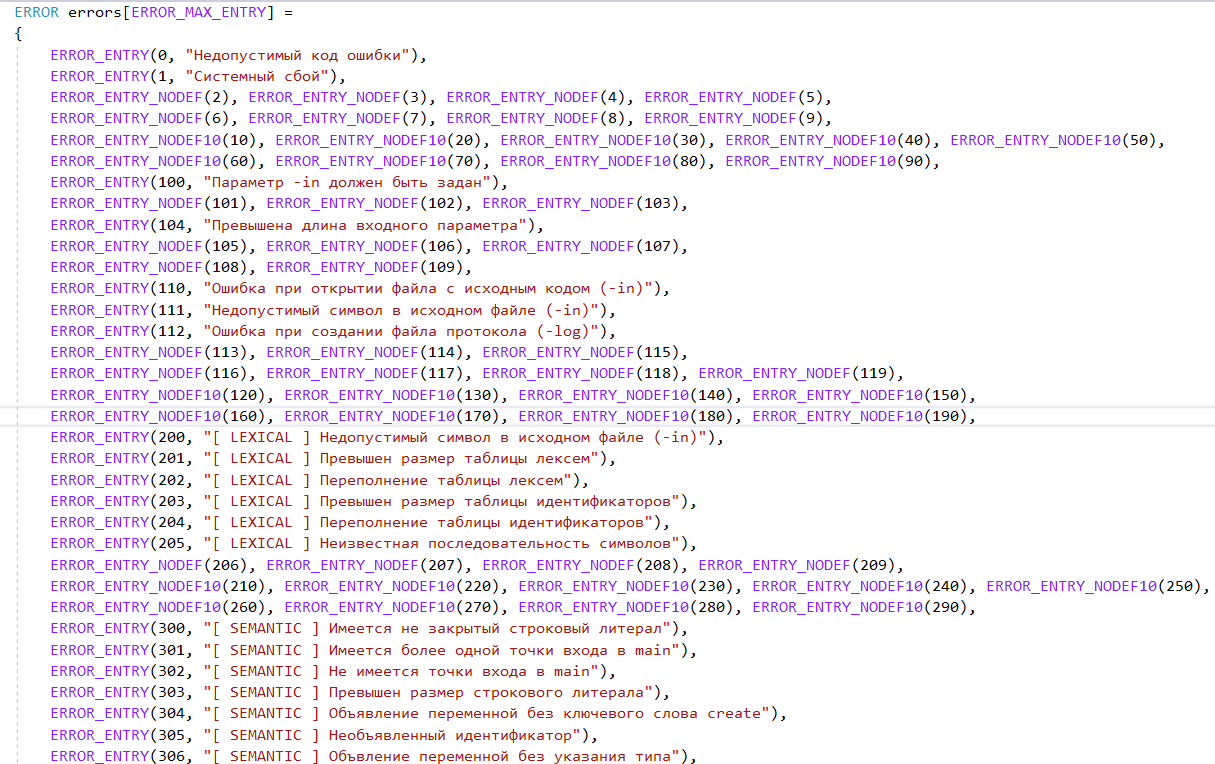


Таблица ошибок

****

# **Приложение Б**

Графы переходов

|  |
| --- |
| #define INTEGER 8,\  FSTN::NODE(1, FSTN::RELATION('i', 1)),\  FSTN::NODE(1, FSTN::RELATION('n', 2)),\  FSTN::NODE(1, FSTN::RELATION('t', 3)),\  FSTN::NODE(1, FSTN::RELATION('e', 4)),\  FSTN::NODE(1, FSTN::RELATION('g', 5)),\  FSTN::NODE(1, FSTN::RELATION('e', 6)),\  FSTN::NODE(1, FSTN::RELATION('r', 7)),\  FSTN::NODE()  #define GRAPH\_INT\_NEGATIVE 6,\  FSTN::NODE(1, \  FSTN::RELATION('-', 1)),\  FSTN::NODE(2, \  FSTN::RELATION('0', 2),FSTN::RELATION('0', 4)),\  FSTN::NODE(1, \  FSTN::RELATION('x', 3)),\  FSTN::NODE(32,\  FSTN::RELATION('1',3),FSTN::RELATION('0',3), FSTN::RELATION('2',3),FSTN::RELATION('3',3),\  FSTN::RELATION('A',3),FSTN::RELATION('B',3), FSTN::RELATION('4',3),FSTN::RELATION('5',3),\  FSTN::RELATION('C',3),FSTN::RELATION('D',3), FSTN::RELATION('6',3),FSTN::RELATION('7',3),\  FSTN::RELATION('E',3),FSTN::RELATION('F',3), FSTN::RELATION('8',3),FSTN::RELATION('9',3),\  FSTN::RELATION('1',5),FSTN::RELATION('0',5), FSTN::RELATION('2',5),FSTN::RELATION('3',5),\  FSTN::RELATION('A',5),FSTN::RELATION('B',5), FSTN::RELATION('4',5),FSTN::RELATION('5',5),\  FSTN::RELATION('C',5),FSTN::RELATION('D',5), FSTN::RELATION('6',5),FSTN::RELATION('7',5),\  FSTN::RELATION('E',5),FSTN::RELATION('F',5), FSTN::RELATION('8',5),FSTN::RELATION('9',5)),\  FSTN::NODE(16,\  FSTN::RELATION('1',4),FSTN::RELATION('0',4), FSTN::RELATION('2',2),FSTN::RELATION('3',4),\  FSTN::RELATION('4',4),FSTN::RELATION('5',4), FSTN::RELATION('6',2),FSTN::RELATION('7',4),\  FSTN::RELATION('1',5),FSTN::RELATION('0',5), FSTN::RELATION('2',5),FSTN::RELATION('3',5),\  FSTN::RELATION('4',5),FSTN::RELATION('5',5), FSTN::RELATION('6',5),FSTN::RELATION('7',5)),\  FSTN::NODE()  #define GRAPH\_INT\_LITERAL 6, \  FSTN::NODE(23, \  FSTN::RELATION('-', 1),FSTN::RELATION('0', 2),FSTN::RELATION('0', 5),\  FSTN::RELATION('1',1),FSTN::RELATION('0',3),FSTN::RELATION('1',5),\  FSTN::RELATION('2',1),FSTN::RELATION('3',1),FSTN::RELATION('2',5),FSTN::RELATION('3',5),\  FSTN::RELATION('6',1),FSTN::RELATION('7',1),FSTN::RELATION('4',5),FSTN::RELATION('5',5),\  FSTN::RELATION('1',1),FSTN::RELATION('6',5),FSTN::RELATION('7',5),\  FSTN::RELATION('4',1),FSTN::RELATION('5',1),FSTN::RELATION('8',5),FSTN::RELATION('9',5),\  FSTN::RELATION('8',1),FSTN::RELATION('9',1)),\  FSTN::NODE(20,\  FSTN::RELATION('1',1),FSTN::RELATION('0',1), FSTN::RELATION('0',5),\  FSTN::RELATION('2',1),FSTN::RELATION('3',1), FSTN::RELATION('4',1),FSTN::RELATION('5',1),\  FSTN::RELATION('6',1),FSTN::RELATION('7',1), FSTN::RELATION('8',1),FSTN::RELATION('9',1),\  FSTN::RELATION('1',5),FSTN::RELATION('2',5), FSTN::RELATION('3',5),FSTN::RELATION('4',5),\  FSTN::RELATION('5',5),FSTN::RELATION('6',5), FSTN::RELATION('7',5),FSTN::RELATION('8',5),\  FSTN::RELATION('9',5)),\  FSTN::NODE(16,\  FSTN::RELATION('1',2),FSTN::RELATION('0',2), FSTN::RELATION('2',2),FSTN::RELATION('3',2),\  FSTN::RELATION('4',2),FSTN::RELATION('5',2), FSTN::RELATION('6',2),FSTN::RELATION('7',2),\  FSTN::RELATION('1',5),FSTN::RELATION('0',5), FSTN::RELATION('2',5),FSTN::RELATION('3',5),\  FSTN::RELATION('4',5),FSTN::RELATION('5',5), FSTN::RELATION('6',5),FSTN::RELATION('7',5)),\  FSTN::NODE(1,FSTN::RELATION('x',4)),\  FSTN::NODE(32,\  FSTN::RELATION('1',4),FSTN::RELATION('0',4), FSTN::RELATION('2',4),FSTN::RELATION('3',4),\  FSTN::RELATION('A',4),FSTN::RELATION('B',4), FSTN::RELATION('4',4),FSTN::RELATION('5',4),\  FSTN::RELATION('C',4),FSTN::RELATION('D',4), FSTN::RELATION('6',4),FSTN::RELATION('7',4),\  FSTN::RELATION('E',4),FSTN::RELATION('F',4), FSTN::RELATION('8',4),FSTN::RELATION('9',4),\  FSTN::RELATION('1',5),FSTN::RELATION('0',5), FSTN::RELATION('2',5),FSTN::RELATION('3',5),\  FSTN::RELATION('A',5),FSTN::RELATION('B',5), FSTN::RELATION('4',5),FSTN::RELATION('5',5),\  FSTN::RELATION('C',5),FSTN::RELATION('D',5), FSTN::RELATION('6',5),FSTN::RELATION('7',5),\  FSTN::RELATION('E',5),FSTN::RELATION('F',5), FSTN::RELATION('8',5),FSTN::RELATION('9',5)),\  FSTN::NODE()  #define SYMBOL 7,\  FSTN::NODE(1, FSTN::RELATION('s', 1)),\  FSTN::NODE(1, FSTN::RELATION('y', 2)),\  FSTN::NODE(1, FSTN::RELATION('m', 3)),\  FSTN::NODE(1, FSTN::RELATION('b', 4)),\  FSTN::NODE(1, FSTN::RELATION('o', 5)),\  FSTN::NODE(1, FSTN::RELATION('l', 6)),\  FSTN::NODE()  #define STRING 7,\  FSTN::NODE(1, FSTN::RELATION('s', 1)),\  FSTN::NODE(1, FSTN::RELATION('t', 2)),\  FSTN::NODE(1, FSTN::RELATION('r', 3)),\  FSTN::NODE(1, FSTN::RELATION('i', 4)),\  FSTN::NODE(1, FSTN::RELATION('n', 5)),\  FSTN::NODE(1, FSTN::RELATION('g', 6)),\  FSTN::NODE()  #define FUNC 5,\  FSTN::NODE(1, FSTN::RELATION('f', 1)),\  FSTN::NODE(1, FSTN::RELATION('u', 2)),\  FSTN::NODE(1, FSTN::RELATION('n', 3)),\  FSTN::NODE(1, FSTN::RELATION('c', 4)),\  FSTN::NODE()  #define JUMP 5,\  FSTN::NODE(1, FSTN::RELATION('j', 1)),\  FSTN::NODE(1, FSTN::RELATION('u', 2)),\  FSTN::NODE(1, FSTN::RELATION('m', 3)),\  FSTN::NODE(1, FSTN::RELATION('p', 4)),\  FSTN::NODE()  #define BRING 6,\  FSTN::NODE(1, FSTN::RELATION('b', 1)),\  FSTN::NODE(1, FSTN::RELATION('r', 2)),\  FSTN::NODE(1, FSTN::RELATION('i', 3)),\  FSTN::NODE(1, FSTN::RELATION('n', 4)),\  FSTN::NODE(1, FSTN::RELATION('g', 5)),\  FSTN::NODE()  #define OUTPUT 7,\  FSTN::NODE(1, FSTN::RELATION('o', 1)),\  FSTN::NODE(1, FSTN::RELATION('u', 2)),\  FSTN::NODE(1, FSTN::RELATION('t', 3)),\  FSTN::NODE(1, FSTN::RELATION('p', 4)),\  FSTN::NODE(1, FSTN::RELATION('u', 5)),\  FSTN::NODE(1, FSTN::RELATION('t', 6)),\  FSTN::NODE()  #define STR\_LIT 3,\  FSTN::NODE(1, FSTN::RELATION('\"', 1)),\  FSTN::NODE(87, \  FSTN::RELATION('a', 1), FSTN::RELATION('b', 1), FSTN::RELATION('c', 1), FSTN::RELATION('d', 1),\  FSTN::RELATION('e', 1), FSTN::RELATION('f', 1), FSTN::RELATION('g', 1), FSTN::RELATION('h', 1),\  FSTN::RELATION('i', 1), FSTN::RELATION('j', 1), FSTN::RELATION('k', 1), FSTN::RELATION('l', 1),\  FSTN::RELATION('m', 1), FSTN::RELATION('n', 1), FSTN::RELATION('o', 1), FSTN::RELATION('p', 1),\  FSTN::RELATION('q', 1), FSTN::RELATION('r', 1), FSTN::RELATION('s', 1), FSTN::RELATION('t', 1),\  FSTN::RELATION('u', 1), FSTN::RELATION('v', 1), FSTN::RELATION('w', 1), FSTN::RELATION('x', 1),\  FSTN::RELATION('y', 1), FSTN::RELATION('z', 1),\  FSTN::RELATION('A', 1), FSTN::RELATION('B', 1), FSTN::RELATION('C', 1), FSTN::RELATION('D', 1),\  FSTN::RELATION('E', 1), FSTN::RELATION('F', 1), FSTN::RELATION('G', 1), FSTN::RELATION('H', 1),\  FSTN::RELATION('I', 1), FSTN::RELATION('J', 1), FSTN::RELATION('K', 1), FSTN::RELATION('L', 1),\  FSTN::RELATION('M', 1), FSTN::RELATION('N', 1), FSTN::RELATION('O', 1), FSTN::RELATION('P', 1),\  FSTN::RELATION('Q', 1), FSTN::RELATION('R', 1), FSTN::RELATION('S', 1), FSTN::RELATION('T', 1),\  FSTN::RELATION('U', 1), FSTN::RELATION('V', 1), FSTN::RELATION('W', 1), FSTN::RELATION('X', 1),\  FSTN::RELATION('Y', 1), FSTN::RELATION('Z', 1),\  FSTN::RELATION('0', 1), FSTN::RELATION('1', 1), FSTN::RELATION('2', 1), FSTN::RELATION('3', 1),\  FSTN::RELATION('4', 1), FSTN::RELATION('5', 1), FSTN::RELATION('6', 1), FSTN::RELATION('7', 1),\  FSTN::RELATION('8', 1), FSTN::RELATION('9', 1),\  FSTN::RELATION(' ', 1), FSTN::RELATION(',', 1), FSTN::RELATION('.', 1), FSTN::RELATION(';', 1),\  FSTN::RELATION('-', 1), FSTN::RELATION('+', 1), FSTN::RELATION('\*', 1), FSTN::RELATION('/', 1),\  FSTN::RELATION('=', 1), FSTN::RELATION(':', 1), FSTN::RELATION(')', 1), FSTN::RELATION('(', 1),\  FSTN::RELATION('}', 1), FSTN::RELATION('{', 1), FSTN::RELATION(']', 1), FSTN::RELATION('[', 1),\  FSTN::RELATION('!', 1), FSTN::RELATION('?', 1), FSTN::RELATION('#', 1), FSTN::RELATION('&', 1),\  FSTN::RELATION('>', 1), FSTN::RELATION('<', 1), FSTN::RELATION('[', 1), FSTN::RELATION(']', 1),\  FSTN::RELATION('\"', 2)),\  FSTN::NODE()  #define IF 3,\  FSTN::NODE(1,FSTN::RELATION('i',1)),\  FSTN::NODE(1,FSTN::RELATION('f',2)),\  FSTN::NODE()  #define OTHERWISE 10, \  FSTN::NODE(1, FSTN::RELATION('o',1)),\  FSTN::NODE(1, FSTN::RELATION('t',2)),\  FSTN::NODE(1, FSTN::RELATION('h',3)),\  FSTN::NODE(1, FSTN::RELATION('e',4)),\  FSTN::NODE(1, FSTN::RELATION('r',5)),\  FSTN::NODE(1, FSTN::RELATION('w',6)),\  FSTN::NODE(1, FSTN::RELATION('i',7)),\  FSTN::NODE(1, FSTN::RELATION('s',8)),\  FSTN::NODE(1, FSTN::RELATION('e',9)),\  FSTN::NODE()  #define STD\_MODULE 7, \  FSTN::NODE(1, FSTN::RELATION('m',1)),\  FSTN::NODE(1, FSTN::RELATION('o',2)),\  FSTN::NODE(1, FSTN::RELATION('d',3)),\  FSTN::NODE(1, FSTN::RELATION('u',4)),\  FSTN::NODE(1, FSTN::RELATION('l',5)),\  FSTN::NODE(1, FSTN::RELATION('e',6)),\  FSTN::NODE()  #define STD\_RAISE 6, \  FSTN::NODE(1, FSTN::RELATION('r',1)),\  FSTN::NODE(1, FSTN::RELATION('a',2)),\  FSTN::NODE(1, FSTN::RELATION('i',3)),\  FSTN::NODE(1, FSTN::RELATION('s',4)),\  FSTN::NODE(1, FSTN::RELATION('e',5)),\  FSTN::NODE()  #define UNTIL 6, \  FSTN::NODE(1, FSTN::RELATION('u',1)),\  FSTN::NODE(1, FSTN::RELATION('n',2)),\  FSTN::NODE(1, FSTN::RELATION('t',3)),\  FSTN::NODE(1, FSTN::RELATION('i',4)),\  FSTN::NODE(1, FSTN::RELATION('l',5)),\  FSTN::NODE()  #define START 6, \  FSTN::NODE(1, FSTN::RELATION('S',1)),\  FSTN::NODE(1, FSTN::RELATION('t',2)),\  FSTN::NODE(1, FSTN::RELATION('a',3)),\  FSTN::NODE(1, FSTN::RELATION('r',4)),\  FSTN::NODE(1, FSTN::RELATION('t',5)),\  FSTN::NODE()  #define CREATE 7, \  FSTN::NODE(1, FSTN::RELATION('c',1)),\  FSTN::NODE(1, FSTN::RELATION('r',2)),\  FSTN::NODE(1, FSTN::RELATION('e',3)),\  FSTN::NODE(1, FSTN::RELATION('a',4)),\  FSTN::NODE(1, FSTN::RELATION('t',5)),\  FSTN::NODE(1, FSTN::RELATION('e',6)),\  FSTN::NODE()  #define PLUS 2, \  FSTN::NODE(1, FSTN::RELATION('+', 1)),\  FSTN::NODE()  #define MINUS 2, \  FSTN::NODE(1, FSTN::RELATION('-', 1)),\  FSTN::NODE()  #define STAR 2, \  FSTN::NODE(1, FSTN::RELATION('\*', 1)),\  FSTN::NODE()  #define DIRSLASH 2, \  FSTN::NODE(1, FSTN::RELATION('/', 1)),\  FSTN::NODE()  #define AND 2, \  FSTN::NODE(1, FSTN::RELATION('&', 1)),\  FSTN::NODE()  #define OR 2, \  FSTN::NODE(1, FSTN::RELATION('|', 1)),\  FSTN::NODE()  #define NO 2, \  FSTN::NODE(1, FSTN::RELATION('?', 1)),\  FSTN::NODE()  #define LESS 2, \  FSTN::NODE(1, FSTN::RELATION('>', 1)),\  FSTN::NODE()  #define MORE 2, \  FSTN::NODE(1, FSTN::RELATION('<', 1)),\  FSTN::NODE()  #define SEMICOLON 2, \  FSTN::NODE(1, FSTN::RELATION(';', 1)),\  FSTN::NODE()  #define COMMA 2, \  FSTN::NODE(1, FSTN::RELATION(',', 1)),\  FSTN::NODE()  #define LEFTBRACE 2, \  FSTN::NODE(1, FSTN::RELATION('{', 1)),\  FSTN::NODE()  #define BRACELET 2, \  FSTN::NODE(1, FSTN::RELATION('}', 1)),\  FSTN::NODE()  #define LEFTTHESIS 2, \  FSTN::NODE(1, FSTN::RELATION('(', 1)),\  FSTN::NODE()  #define RIGHTTHESIS 2, \  FSTN::NODE(1, FSTN::RELATION(')', 1)),\  FSTN::NODE()  #define EQUALS 2, \  FSTN::NODE(1, FSTN::RELATION('=', 1)),\  FSTN::NODE()  #define NEXTLINE 2, \  FSTN::NODE(1, FSTN::RELATION('\n', 1)),\  FSTN::NODE()  #define IDENTIFICATOR 1, \  FSTN::NODE(54, FSTN::RELATION('a', 0), \  FSTN::RELATION('b', 0), \  FSTN::RELATION('c', 0), \  FSTN::RELATION('d', 0), \  FSTN::RELATION('e', 0), \  FSTN::RELATION('f', 0), \  FSTN::RELATION('g', 0), \  FSTN::RELATION('h', 0), \  FSTN::RELATION('i', 0), \  FSTN::RELATION('j', 0), \  FSTN::RELATION('k', 0), \  FSTN::RELATION('l', 0), \  FSTN::RELATION('m', 0), \  FSTN::RELATION('n', 0), \  FSTN::RELATION('o', 0), \  FSTN::RELATION('p', 0), \  FSTN::RELATION('q', 0), \  FSTN::RELATION('r', 0), \  FSTN::RELATION('s', 0), \  FSTN::RELATION('t', 0), \  FSTN::RELATION('u', 0), \  FSTN::RELATION('v', 0), \  FSTN::RELATION('w', 0), \  FSTN::RELATION('x', 0), \  FSTN::RELATION('y', 0), \  FSTN::RELATION('z', 0), \  FSTN::RELATION('\_', 0)),\  FSTN::RELATION('A', 0), \  FSTN::RELATION('B', 0), \  FSTN::RELATION('C', 0), \  FSTN::RELATION('D', 0), \  FSTN::RELATION('E', 0), \  FSTN::RELATION('F', 0), \  FSTN::RELATION('G', 0), \  FSTN::RELATION('H', 0), \  FSTN::RELATION('I', 0), \  FSTN::RELATION('J', 0), \  FSTN::RELATION('K', 0), \  FSTN::RELATION('L', 0), \  FSTN::RELATION('M', 0), \  FSTN::RELATION('N', 0), \  FSTN::RELATION('O', 0), \  FSTN::RELATION('P', 0), \  FSTN::RELATION('Q', 0), \  FSTN::RELATION('R', 0), \  FSTN::RELATION('S', 0), \  FSTN::RELATION('T', 0), \  FSTN::RELATION('U', 0), \  FSTN::RELATION('V', 0), \  FSTN::RELATION('W', 0), \  FSTN::RELATION('X', 0), \  FSTN::RELATION('Y', 0), \  FSTN::RELATION('Z', 0), \  FSTN::NODE() |

Таблица идентификаторов

|  |
| --- |
| ============================= Identificator table ============================  | N |Line in LT | Identificator type | Name | Value(parameters)  | 0 | 2 | integer func | sum |-  | 1 | 5 | integer parameter | suma |-  | 2 | 8 | integer parameter | sumb |-  | 3 | 13 | integer variable | sumz |0  | 4 | 16 | UNDEFINED | + |-  | 5 | 23 | integer func | Start |-  | 6 | 27 | integer variable | Startd |0  | 7 | 29 | integer literal | L1 |1  | 8 | 33 | integer variable | Startx |0  | 9 | 35 | integer literal | L2 |2  | 10 | 39 | integer variable | Startio |0  | 11 | 51 | UNDEFINED | > |-  | 12 | 52 | integer literal | L3 |0  | 13 | 73 | string variable | Startstr |[0]  | 14 | 75 | string literal | L4 |[5]hello  | 15 | 82 | symbol variable | Startch |  | 16 | 84 | symbol literal | L5 |f  | 17 | 93 | integer variable | Starty |0  | 18 | 99 | integer variable | Startz |0  | 19 | 101 | integer func | module |-  | 20 | 113 | integer variable | Startk |0  | 21 | 115 | integer func | raise |-  | 22 | 117 | integer literal | L6 |5  | 23 | 119 | integer literal | L7 |4  | 24 | 129 | integer variable | Startv |0  | 25 | 131 | integer literal | L8 |154  | 26 | 135 | integer variable | Startalt |0  | 27 | 138 | UNDEFINED | & |-  | 28 | 156 | UNDEFINED | - |-  ------------------------------------------------------------------------- |

Таблица лексем

|  |
| --- |
| 1 tfi(ti,ti)  2 {  3 qti=ivi;  4 bi;  5 }  6 s  7 {  8 qti=l;  9 qti=l;  10 qti=i(i,i);  11 y(ivl)  12 {  13 i=ivl;  14 }  15 n  16 {  17 i=l;  18 }  19 ^;  20 qti=l;  21 oi;  22 qti=l;  23 oi;  24 ^;  25 qti=l;  26 qti=i(i);  27 oi;  28 ^;  29 qti=i(l,l);  30 oi;  31 ^;  32 qti=l;  33 qti=ivi;  34 oi;  35 ^;  36 u(ivl)  37 {  38 i=ivl;  39 oi;  40 }  41 ^;  42 bl;  43 } |

Таблица лексем после преобразования в ПОЛИЗ

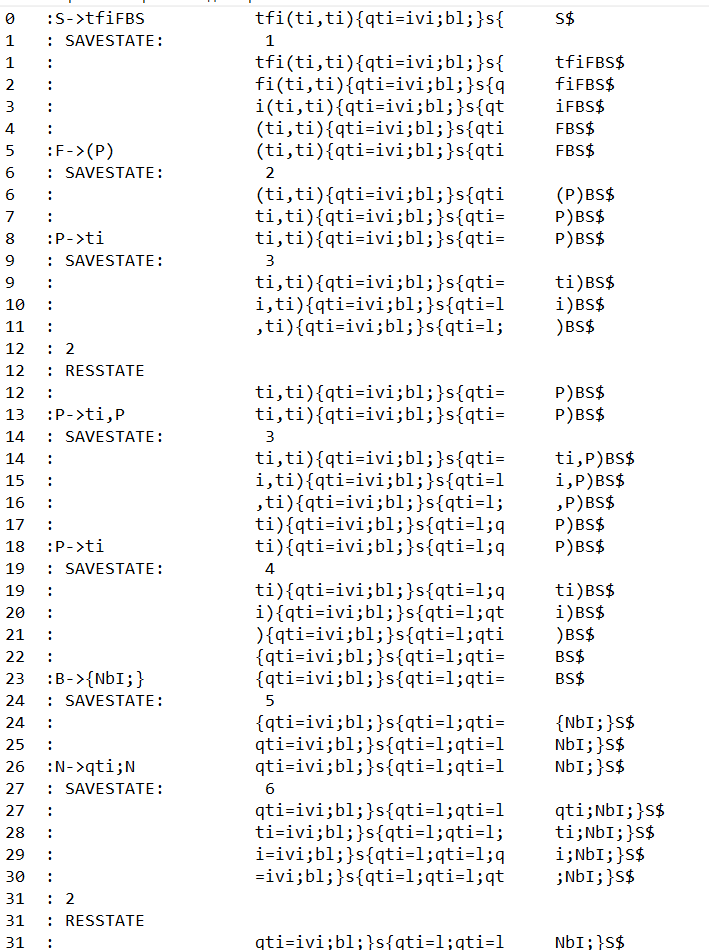
|  |
| --- |
| 1 tfi(ti,ti)  2 {  3 qti=iiv;  4 bi;  5 }  6 s  7 {  8 qti=l;  9 qti=l;  10 qti=ii@2##;  11 y(ivl)  12 {  13 i=ilv;  14 }  15 n  16 {  17 i=l;  18 }  19 ^;  20 qti=l;  21 oi;  22 qti=l;  23 oi;  24 ^;  25 qti=l;  26 qti=i@1#;  27 oi;  28 ^;  29 qti=ll@2##;  30 oi;  31 ^;  32 qti=l;  33 qti=iiv;  34 oi;  35 ^;  36 u(ivl)  37 {  38 i=ilv;  39 oi;  40 }  41 ^;  42 bl;  43 } |

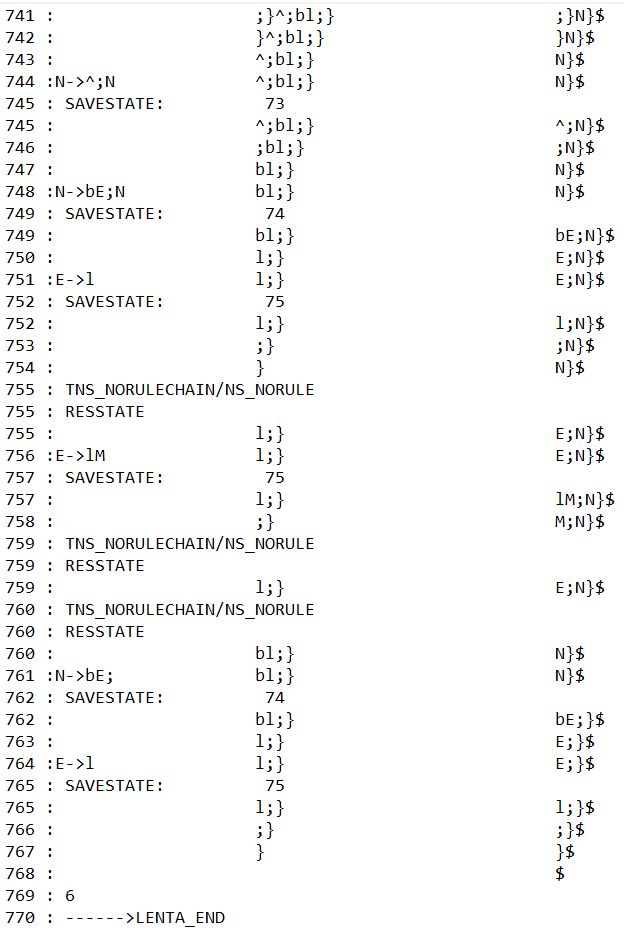
# **Приложение В**

Грамматика

|  |
| --- |
| NS('S'), GRB\_ERROR\_SERIES + 0,  3,  Rule::Chain(6, TS('t'), TS('f'), TS('i'), NS('F'), NS('B'), NS('S')),  Rule::Chain(4, TS('s'), TS('{'), NS('N'), TS('}')),  Rule::Chain(5, TS('t'), TS('f'), TS('i'), NS('F'), NS('B'))  ),  Rule(  NS('F'), GRB\_ERROR\_SERIES + 1,  2,  Rule::Chain(3, TS('('), NS('P'), TS(')')),  Rule::Chain(2, TS('('), TS(')'))  ),  Rule(  NS('P'), GRB\_ERROR\_SERIES + 2,  2,  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('P'))  ),  Rule(  NS('B'), GRB\_ERROR\_SERIES + 3,  3,  Rule::Chain(6, TS('{'), NS('N'), TS('b'), NS('I'), TS(';'), TS('}')),  Rule::Chain(5, TS('{'), TS('b'), NS('I'), TS(';'), TS('}')),  Rule::Chain(3, TS('{'), NS('N'), TS('}'))  ),  Rule(  NS('I'), GRB\_ERROR\_SERIES + 4,  2,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l'))  ),  Rule(  NS('U'), GRB\_ERROR\_SERIES + 5,  1,  Rule::Chain(3, TS('{'), NS('N'), TS('}'))  ),  Rule(  NS('N'), GRB\_ERROR\_SERIES + 6,  20,  Rule::Chain(5, TS('q'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(7, TS('q'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(8, TS('u'), TS('('), NS('R'), TS(')'), TS('{'), NS('X'), TS('}'), NS('N')),  Rule::Chain(8, TS('y'), TS('('), NS('R'), TS(')'), TS('{'), NS('X'), TS('}'), NS('N')),  Rule::Chain(12, TS('y'), TS('('), NS('R'), TS(')'), TS('{'), NS('X'), TS('}'), TS('n'), TS('{'), NS('X'), TS('}'), NS('N')),  Rule::Chain(4, TS('o'), NS('I'), TS(';'), NS('N')),  Rule::Chain(3, TS('^'), TS(';'), NS('N')),  Rule::Chain(4, TS('b'), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), NS('K'), TS(';'), NS('N')),  Rule::Chain(4, TS('q'), TS('t'), TS('i'), TS(';')),  Rule::Chain(6, TS('q'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(7, TS('u'), TS('('), NS('R'), TS(')'), TS('{'), NS('X'), TS('}')),  Rule::Chain(7, TS('y'), TS('('), NS('R'), TS(')'), TS('{'), NS('X'), TS('}')),  Rule::Chain(11, TS('y'), TS('('), NS('R'), TS(')'), TS('{'), NS('X'), TS('}'), TS('n'), TS('{'), NS('X'), TS('}')),  Rule::Chain(3, TS('o'), NS('I'), TS(';')),  Rule::Chain(2, TS('^'), TS(';')),  Rule::Chain(3, TS('b'), NS('E'), TS(';')),  Rule::Chain(3, TS('i'), NS('K'), TS(';'))  ),  Rule(  NS('R'), GRB\_ERROR\_SERIES + 7,  4,  Rule::Chain(1, TS('i')),  Rule::Chain(3, TS('i'), TS('v'), TS('i')),  Rule::Chain(3, TS('i'), TS('v'), TS('l')),  Rule::Chain(3, TS('l'), TS('v'), TS('i'))  ),  Rule(  NS('K'), GRB\_ERROR\_SERIES + 8,  2,  Rule::Chain(3, TS('('), NS('W'), TS(')')),  Rule::Chain(2, TS('('), TS(')'))  ),  Rule(  NS('E'), GRB\_ERROR\_SERIES + 9,  8,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(2, TS('i'), NS('K')),  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, TS('i'), NS('K'), NS('M'))  ),  Rule(  NS('W'), GRB\_ERROR\_SERIES + 10,  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(  NS('M'), GRB\_ERROR\_SERIES + 11,  2,  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(3, TS('v'), NS('E'), NS('M'))  ),  Rule(  NS('X'), GRB\_ERROR\_SERIES + 12,  14,  Rule::Chain(5, TS('q'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(7, TS('q'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('o'), NS('I'), TS(';'), NS('N')),  Rule::Chain(3, TS('^'), TS(';'), NS('N')),  Rule::Chain(4, TS('b'), NS('E'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), NS('K'), TS(';'), NS('N')),  Rule::Chain(4, TS('q'), TS('t'), TS('i'), TS(';')),  Rule::Chain(6, TS('q'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(3, TS('o'), NS('I'), TS(';')),  Rule::Chain(2, TS('^'), TS(';')),  Rule::Chain(3, TS('b'), NS('E'), TS(';')),  Rule::Chain(3, TS('i'), NS('K'), TS(';'))  ) |

Работа конечного автомата и дерево разбора





Дерево разбора

|  |
| --- |
| 0 : S->tfiFBS  3 : F->(P)  4 : P->ti,P  7 : P->ti  10 : B->{NbI;}  11 : N->qti=E;  15 : E->iM  16 : M->vE  17 : E->i  20 : I->l  23 : S->s{N}  25 : N->qti=E;N  29 : E->l  31 : N->qti=E;N  35 : E->l  37 : N->qti=E;N  41 : E->iK  42 : K->(W)  43 : W->i,W  45 : W->i  48 : N->y(R){X}n{X}N  50 : R->ivl  55 : X->i=E;  57 : E->iM  58 : M->vE  59 : E->l  64 : X->i=E;  66 : E->l  69 : N->^;N  71 : N->qti=E;N  75 : E->l  77 : N->oI;N  78 : I->i  80 : N->qti=E;N  84 : E->l  86 : N->oI;N  87 : I->i  89 : N->^;N  91 : N->qti=E;N  95 : E->l  97 : N->qti=E;N  101 : E->iK  102 : K->(W)  103 : W->i  106 : N->oI;N  107 : I->i  109 : N->^;N  111 : N->qti=E;N  115 : E->iK  116 : K->(W)  117 : W->l,W  119 : W->l  122 : N->oI;N  123 : I->i  125 : N->^;N  127 : N->qti=E;N  131 : E->l  133 : N->qti=E;N  137 : E->iM  138 : M->vE  139 : E->i  141 : N->oI;N  142 : I->i  144 : N->^;N  146 : N->u(R){X}N  148 : R->ivl  153 : X->i=E;N  155 : E->iM  156 : M->vE  157 : E->l  159 : N->oI;  160 : I->i  163 : N->^;N  165 : N->bE;  166 : E->l |

# **Приложение Г**

ПОЛИЗ

|  |
| --- |
| #include"PolishNotation.h"  #include"parm.h";  #include"Log.h"  #include"LT.h"  #include<stack>  #include<queue>  #include<map>  using namespace std;  map<char, int>Priorities = {  {'<',0},  {'>',0},  {'(',0},  {')',0},  {',',1},  {'+',2},  {'-',2},  {'\*',3},  {'/',3},  {'&',3},  {'|',3},  {'?',3}  };  namespace PN {  bool polishNotation(int i, LexAn::LEX& lex) {  std::stack<LT::Entry> stack;  std::queue<LT::Entry> queue;  LT::Entry aggregate\_symbol;  aggregate\_symbol.idxTI = -1;  aggregate\_symbol.lexema = '#';  aggregate\_symbol.sn = lex.lextable.table[i].sn;  LT::Entry function\_symbol;  function\_symbol.idxTI = LT\_TI\_NULLIDX;  function\_symbol.lexema = '@';  function\_symbol.sn = lex.lextable.table[i].sn;  int idx;  int lexem\_counter = 0;  int parm\_counter = 0;  int lexem\_position = i;  char\* buf = new char[i];  bool is\_function = false;  for (i; lex.lextable.table[i].lexema != LEX\_SEMICOLON; i++, lexem\_counter++) {  switch (lex.lextable.table[i].lexema) {  case LEX\_ID:  case LEX\_LITERAL:  if (lex.idtable.table[lex.lextable.table[i].idxTI].idtype == IT::F) {  is\_function = true;  idx = lex.lextable.table[i].idxTI;  }  else {  if (is\_function)  parm\_counter++;  queue.push(lex.lextable.table[i]);  }  continue;  case LEFTHESIS:  stack.push(lex.lextable.table[i]);  continue;  case RIGHTHESIS:  while (stack.top().lexema != LEFTHESIS) {  queue.push(stack.top());  stack.pop();  if (stack.empty())  return false;  }  if (!is\_function)  stack.pop();  else {  function\_symbol.idxTI = idx;  idx = LT\_TI\_NULLIDX;  lex.lextable.table[i] = function\_symbol;  queue.push(lex.lextable.table[i]);  \_itoa\_s(parm\_counter, buf, 2, 10);  stack.top().lexema = buf[0];  stack.top().idxTI = LT\_TI\_NULLIDX;  stack.top().sn = function\_symbol.sn;  queue.push(stack.top());  stack.pop();  parm\_counter = 0;  is\_function = false;  }  continue;  case LEX\_OPERATOR:  while (!stack.empty() && Priorities[lex.lextable.table[i].lexema] <= Priorities[stack.top().lexema]) {  queue.push(stack.top());  stack.pop();  }  stack.push(lex.lextable.table[i]);  continue;  }  }  while (!stack.empty()) {  if (stack.top().lexema == LEFTHESIS || stack.top().lexema == RIGHTHESIS)  return false;  queue.push(stack.top());  stack.pop();  }  while (lexem\_counter != 0) {  if (!queue.empty()) {  lex.lextable.table[lexem\_position++] = queue.front();  queue.pop();  }  else  lex.lextable.table[lexem\_position++] = aggregate\_symbol;  lexem\_counter--;  }  for (int i = 0; i < lexem\_position; i++) {  if (lex.lextable.table[i].lexema == LEX\_OPERATOR || lex.lextable.table[i].lexema == LEX\_LITERAL)  lex.idtable.table[lex.lextable.table[i].idxTI].idxfirstLE = i;  }  return true;  }  void startPolish(LexAn::LEX& lex) {  for (int i = 0; i < lex.lextable.size; i++) {  if (lex.lextable.table[i].lexema == '=') {  polishNotation(i + 1, lex);  }  }  }  } |

Лексемы после преобразования в ПОЛИЗ

|  |
| --- |
| 1 tfi(ti,ti)  2 {  3 qti=iiv;  4 bl;  5 }  6 s  7 {  8 qti=l;  9 qti=l;  10 qti=ii@2##;  11 y(ivl)  12 {  13 i=ilv;  14 }  15 n  16 {  17 i=l;  18 }  19 ^;  20 qti=l;  21 oi;  22 qti=l;  23 oi;  24 ^;  25 qti=l;  26 qti=i@1#;  27 oi;  28 ^;  29 qti=ll@2##;  30 oi;  31 ^;  32 qti=l;  33 qti=iiv;  34 oi;  35 ^;  36 u(ivl)  37 {  38 i=ilv;  39 oi;  40 }  41 ^;  42 bl;  43 } |

# **Приложение Д**

Генерация в Ассемблер

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ..\GVS-2022\Debug\GVS-2022Libs.lib  ExitProcess PROTO:DWORD  .stack 4096  module PROTO:DWORD  raise PROTO:DWORD,:DWORD  outputint PROTO:DWORD  outputstr PROTO:DWORD  outputsym PROTO:DWORD  .const  L1 SDWORD 0  L2 SDWORD 1  L3 SDWORD 2  L4 BYTE "hello", 0  L5 BYTE "f", 0  L6 SDWORD 5  L7 SDWORD 4  L8 SDWORD 154  .data  buffer BYTE 256 dup(0)  sumz SDWORD 0  Startd SDWORD 0  Startx SDWORD 0  Startio SDWORD 0  Startstr DWORD ?  Startch DWORD ?  Starty SDWORD 0  Startz SDWORD 0  Startk SDWORD 0  Startv SDWORD 0  Startalt SDWORD 0  .code  sum PROC suma : SDWORD, sumb : SDWORD  push suma  push sumb  pop eax  pop ebx  add eax, ebx  push eax  pop sumz  push 0  jmp local0  local0:  pop eax  ret  sum ENDP  main PROC  push L2  pop Startd  push L3  pop Startx  push Startd  push Startx  pop edx  pop edx  push Startx  push Startd  call sum  push eax  pop Startio  mov eax, Startio  cmp eax, L1  jl m0  jg m1  je m1  m0:  push Startio  push L2  pop eax  pop ebx  add eax, ebx  push eax  pop Startio  jmp e0  m1:  push L1  pop Startio  e0:  call outputstr  push offset L4  pop Startstr  push Startstr  call outputstr  push offset L5  pop Startch  push Startch  call outputstr  call outputstr  push L3  pop Starty  push Starty  pop edx  push Starty  call module  push eax  pop Startz  push Startz  call outputint  call outputstr  push L6  push L7  pop edx  pop edx  push L7  push L6  call raise  push eax  pop Startk  push Startk  call outputint  call outputstr  push L8  pop Startv  push Startz  push Startk  pop ebx  pop eax  and eax, ebx  push edx  pop Startalt  push Startalt  call outputint  call outputstr  mov eax, Startz  cmp eax, L2  jl cycle0  jmp cyclenext0  cycle0:  push Startk  push L2  pop ebx  pop eax  sub eax, ebx  push eax  pop Startk  push Startk  call outputint  mov eax, Startz  cmp eax, L2  jl cycle0  cyclenext0:  call outputstr  push 0  jmp theend  theend:  call ExitProcess  main ENDP  end main |