МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора SYO-2020»

Выполнил студент Шуст Юрий Олегович

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

(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 ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2020

Оглавление

[**Введение** 4](#_Toc58778302)

[**Глава 1. Спецификация языка программирования** 5](#_Toc58778303)

[**1.1** **Характеристика языка программирования** 5](#_Toc58778304)

[**1.2** **Определение алфавита языка программирования** 5](#_Toc58778305)

[**1.3** **Применяемые сепараторы** 5](#_Toc58778306)

[**1.4 Применяемые кодировки** 5](#_Toc58778307)

[**1.5** **Типы данных** 6](#_Toc58778308)

[**1.6** **Преобразование типов данных** 7](#_Toc58778309)

[**1.7** **Идентификаторы** 7](#_Toc58778310)

[**1.8** **Литералы** 7](#_Toc58778311)

[**1.9** **Объявление данных** 8](#_Toc58778312)

[**1.10** **Инициализация данных** 8](#_Toc58778313)

[**1.11** **Инструкции языка** 8](#_Toc58778314)

[**1.12** **Операции языка** 9](#_Toc58778315)

[**1.13** **Выражения и их вычисления** 9](#_Toc58778316)

[**1.14** **Конструкции языка** 10](#_Toc58778317)

[**1.15** **Область видимости идентификаторов** 10](#_Toc58778318)

[**1.16** **Семантические проверки** 10](#_Toc58778319)

[**1.17** **Распределение оперативной памяти на этапе выполнения** 11](#_Toc58778320)

[**1.18** **Стандартная библиотека и ее состав** 11](#_Toc58778321)

[**1.19** **Вывод и ввод данных** 11](#_Toc58778322)

[**1.20** **Точка входа** 12](#_Toc58778323)

[**1.21** **Препроцессор** 12](#_Toc58778324)

[**1.22** **Соглашения о вызове** 12](#_Toc58778325)

[**1.23** **Объектный код** 12](#_Toc58778326)

[**1.24** **Классификация сообщений транслятора** 12](#_Toc58778327)

[**1.25** **Контрольный пример** 13](#_Toc58778328)

[**Глава 2. Структура транслятора** 14](#_Toc58778329)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 14](#_Toc58778330)

[**2.2 Перечень входных параметров транслятора** 15](#_Toc58778331)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 15](#_Toc58778332)

[**Глава 3. Разработка лексического анализатора** 17](#_Toc58778333)

[**3.1 Структура лексического анализатора** 17](#_Toc58778334)

[**3.2. Контроль входных символов** 17](#_Toc58778335)

[**3.3 Удаление избыточных символов** 18](#_Toc58778336)

[**3.4 Перечень ключевых слов** 18](#_Toc58778337)

[**3.5 Основные структуры данных** 19](#_Toc58778338)

[**3.6 Принцип обработки ошибок** 19](#_Toc58778339)

[**3.7 Структура и перечень сообщений лексического анализа** 19](#_Toc58778340)

[**3.8 Параметры лексического анализатора** 20](#_Toc58778341)

[**3.9 Алгоритм лексического анализа** 20](#_Toc58778342)

[**3.10 Контрольный пример** 20](#_Toc58778343)

[**Глава 4. Разработка синтаксического анализатора** 21](#_Toc58778344)

[**4.1 Структура синтаксического анализатора** 21](#_Toc58778345)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис** 21](#_Toc58778346)

[**4.3 Построение конечного магазинного автомата** 23](#_Toc58778347)

[**4.4 Основные структуры данных** 24](#_Toc58778348)

[**4.5 Описание алгоритма синтаксического разбора** 24](#_Toc58778349)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 25](#_Toc58778350)

[**4.7. Параметры синтаксического анализатора и режимы его работы** 25](#_Toc58778351)

[**4.8. Принцип обработки ошибок** 25](#_Toc58778352)

[**4.9. Контрольный пример** 25](#_Toc58778353)

[**Глава 5. Разработка семантического анализатора** 26](#_Toc58778354)

[**5.1 Структура семантического анализатора** 26](#_Toc58778355)

[**5.2 Функции семантического анализатора** 26](#_Toc58778356)

[**5.3 Структура и перечень семантических ошибок** 26](#_Toc58778357)

[**5.4 Принцип обработки ошибок** 26](#_Toc58778358)

[**5.5 Контрольный пример** 26](#_Toc58778359)

[**Глава 6. Вычисление выражений** 28](#_Toc58778360)

[**6.1 Выражения, допускаемые языком** 28](#_Toc58778361)

[**6.3 Программная реализация обработки выражений** 28](#_Toc58778362)

[**6.4 Контрольный пример** 29](#_Toc58778363)

[**Глава 7. Генерация кода** 30](#_Toc58778364)

[**7.2 Представление типов данных в оперативной памяти** 30](#_Toc58778365)

[**7.3 Статическая библиотека** 30](#_Toc58778366)

[**7.4 Особенности алгоритма генерации кода** 31](#_Toc58778367)

[**7.5 Входные параметры генератора кода** 31](#_Toc58778368)

[**7.6 Контрольный пример** 31](#_Toc58778369)

[**Глава 8. Тестирование транслятора** 32](#_Toc58778370)

[**8.1 Общие положения** 32](#_Toc58778371)

[**8.2 Результаты тестирования** 32](#_Toc58778372)

[**Заключение** 34](#_Toc58778373)

[**Список использованных источников** 35](#_Toc58778374)

[**Приложение А** 36](#_Toc58778375)

[**Приложение Б** 38](#_Toc58778376)

[**Приложение В** 41](#_Toc58778377)

[**Приложение Г** 46](#_Toc58778378)

[**Приложение Д** 49](#_Toc58778379)

# **Введение**

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

Классический транслятор состоит из следующих частей:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор кода, или интерпретатор.

Все части транслятора, взаимодействуя между собой, обрабатывают входной текст и строят для него эквивалентный текст на понятном компьютеру языке программирования.

**Глава 1. Спецификация языка программирования**

* 1. **Характеристика языка программирования**

Язык программирования SYO-2020 языком программирования высокого уровня. Он является компилируемым, не объектно-ориентированным. В языке отсутствует преобразование типов. В языке поддерживается 2 типа данных: целочисленный (number) и строковый (text). В стандартной библиотеке имеются функции для работы с целочисленным типом данных: sqroot – возвращает квадратный корень числа, module – возвращает модуль числа, input – позволяет ввести целое число пользователю.

* 1. **Определение алфавита языка программирования**

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], [a…я]символы пробела, табуляции и перевода строки, спецсимволы: [] () , ; : # + - / \* % & > < !.

* 1. **Применяемые сепараторы**

Символы сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены. Они представлены в таблице 1.1.

Таблица 1.1 – Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| “пробел” | Разделитель цепочек. Допускается везде, кроме имен идентификаторов и ключевых слов |
| **[** … **]** | Блок функции или цикла |
| **(** … **)** | Блок фактических или формальных параметров функции, а так же приоритет арифметических операций |
| **,** | Разделитель параметров функций |
| **#** | Символ отделяющий цикл |
| **+ - \* /** | Арифметические операции |
| **> < ! } { &** | Логические операторы (Операции сравнения: больше, меньше, не равно, больше или равно, меньше или равно) |
| **;** | Разделитель программных конструкций |
| **=** | Оператор присваивания |

**1.4 Применяемые кодировки**

Для написания программ на языке SYO-2020 используется кодировка Windows – 1251, представленная на рис.1.1.

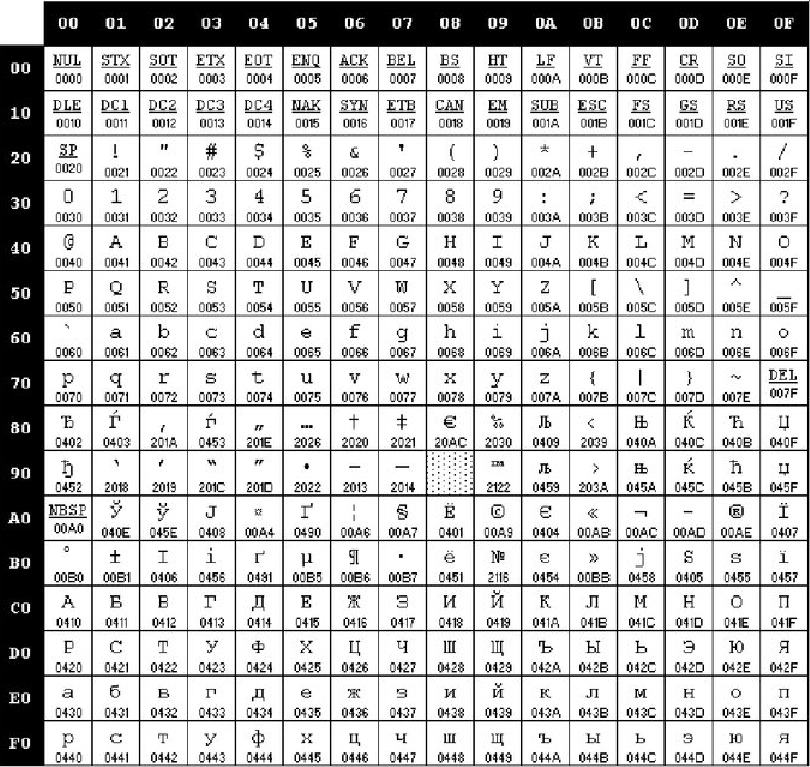


Рисунок 1.1 Алфавит вводных символов

* 1. **Типы данных**

В языке SYO-2020 реализованы 3 фундаментальных типа данных: целочисленный, строковый и логический. Описание типов приведено в таблице 1.2.

Таблица 1.2 – Типы данных языка SYO-2020

|  |  |
| --- | --- |
| Типы данных | Характеристика |
| Целочисленный тип данных **number** | Является целочисленным типом данных. Этот тип данных занимает 4 байта. Предназначен для арифметических операций над числами. Инициализация по умолчанию: 0.  Поддерживаемые операции:  + (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  / (бинарный) – оператор деления;  = (бинарный) – оператор присваивания  %(бинарный) – оператор деления по модулю  В качестве операторов условия цикла поддерживаются следующие операторы:  > (бинарный) – оператор “больше”;  < (бинарный) – оператор “меньше”  ! (бинарный) – оператор “не равно”  } (бинарный) – оператор “больше либо равно”  { (бинарный) – оператор “меньше либо равно”  & (бинарный) – оператор “сравнение” |
| Строковый тип данных **text** | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255.  Инициализация по умолчанию: строка нулевой длины “”.  Операции над данными строкового типа:  = (бинарный) – присваивание значения в момент инициализации. |

* 1. **Преобразование типов данных**

Преобразование типов данных в языке SYO-2020 не поддерживается, так как язык SYO-2020 является строго типизированным.

* 1. **Идентификаторы**

Общее количество идентификаторов ограниченно максимальным размером таблицы идентификаторов (4096). Идентификаторы могут содержать символы как нижнего регистра, так и верхнего. Максимальная длина идентификатора равна 10 символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Данные правила действуют для всех идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции.

Правило составления идентификатора:

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

* 1. **Литералы**

С помощью литералов осуществляется инициализация переменных. Все литералы являются revalue. Имеются литералы целочисленные литералы десятичного и восьмеричного представления, а так же строковые литералы. Подробное описание литералов языка SYO-2020 представлены в таблице  1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не отделяется пробелом) |
| Целочисленные литералы в восьмеричном представлении | Последовательность цифр 0…7 с предшествующим символом “q” |
| Строковые литералы | Набор символов алфавита языка, заключенных в двойные кавычки |

Ограничения на целочисленные литералы: не могут начинаться с 0, если их значение не 0; если литерал отрицательный, после знака “-” не может идти 0.

* 1. **Объявление данных**

Для объявления переменной используется ключевое слово **new,** после которого указывается тип данных и имя идентификатора.

Пример объявления числового типа данных с инициализации:

**new number** x = 14;

**new number** x = q232;

Пример объявления строкового типа данных с инициализацией:

**new text** str = 'привет мир';

Для объявления функций используется ключевое слово **func**, перед которым указывается тип функции. Далее обязателен список параметров и тело функции.

* 1. **Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы и литералы. При объявлении переменные инициализируются значением по умолчанию. Для **number** значение 0, для **text** строка нулевой длины (“”) и значение 0(false) для переменных типа **boolean**.

* 1. **Инструкции языка**

Инструкции языка SYO-2020 представлены в таблице 1.4.

Таблица 1.4 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной | **new**<тип данных>< идентификатор >; |
| Возврат значения из функции | **return** <идентификатор> | <литерал>; |
| Вывод данных | **print** <идентификатор> | <литерал>; |
| Вызов функции | <идентификатор функции>(<список параметров>); |
| Присваивание | <идентификатор> = <выражение>;  Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. |

* 1. **Операции языка**

Операции языка SYO-2020 и их приоритет представлен в таблице 1.5.

Таблица 1.5 – Операции языка SYO-2020

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | **+ -** сложение (приоритет 3)  - **-** разность (приоритет 3)  **\* -** умножение (приоритет 2)  **/ -** деление (приоритет 2)  % - деление по модулю (приоритет 2) |
| Логические | **>** - больше (приоритет 4)  **<** - меньше (приоритет 4)  **!** – не равно (приоритет 5)  **}** – больше или равно (приоритет 4)  **{** - меньше или равно (приоритет 4)  & - сравнение (приоритет 4) |

Для повышения приоритета выполнения операций используются круглые скобки “( )”.

* 1. **Выражения и их вычисления**

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;

2. Выражение записывается в строку без переносов;

3. Использование двух подряд идущих операторов не допускается;

4. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера. Преобразование выражений приведено в главе 5.

* 1. **Конструкции языка**

Программа на языке SYO-2020 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Программные конструкции языка SYO-2020 представлены в таблице 1.6.

Таблица 1.6 – Конструкции языка SYO-2020

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция | **main**  **[**  …  **]** |
| Внешняя функция | <тип данных> **func** <идентификатор> (<тип> <идентификатор>, ...)  **[**  …  **return** <идентификатор/литерал>;  **]** |
| Цикл | **state:** <идентификатор1> <логический оператор><идентификатор2> **#**  **repeat […] #**  Цикл(операторы блока **repeat**) выполняется пока истинно условие стоящие при **state:** |
| Условие | **state:** <идентификатор1> <логический оператор><идентификатор2> **#**  **truest [**<идентификатор> = <литерал> | <идентификатор>**]**  **falsest [**<идентификатор> = <литерал> | <идентификатор>**]#** |

* 1. **Область видимости идентификаторов**

Область видимости: сверху вниз. Переменные, объявленные в одной функции не доступны в другой. Все операции и объявления происходят внутри какого-либо блока или тела функции. Каждая переменная или параметр функции получают префикс – название функции, в которой они находятся.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

* 1. **Семантические проверки**

В языке программирования SYO-2020 выполняются следующие семантические проверки:

1. Наличие функции main – точки входа в программу;

2. Единственность точки входа;

3. Переопределение идентификаторов;

4. Использование идентификаторов без их объявления;

5. Проверка соответствия типа функции и возвращаемого параметра;

6. Правильность передаваемых в функцию параметров: количество, типы;

7. Превышение размера строковых и числовых литералов;

* 1. **Распределение оперативной памяти на этапе выполнения**

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода, несмотря на то, что в оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

* 1. **Стандартная библиотека и ее состав**

В языке SYO-2020 присутствует стандартная библиотека, которая подключается автоматически на этапе трансляции исходного кода в язык ассемблера.

Содержимое стандартной библиотеки представлено в таблице 1.8.

Таблица 1.8 – Стандартная библиотека языка SYO-2020

|  |  |
| --- | --- |
| Функция | Описание |
| **number** sqroot(**number** x) | Целочисленная функция, возвращает корень поданного числа. |
| **number** module(**number** x) | Целочисленная функция, возвращает модуль поданного числа |
| **number** input(**number** x) | Целочисленная функция, возвращает введенное пользователем число |

* 1. **Вывод и ввод данных**

Вывод данных осуществляется с помощью оператора **print**. Допускается использование оператора **print** с литералами и идентификаторами. Допускается перевод строки при использовании функции **printline**.

Ввод данных осуществляется с помощью функции **input**.

Функции, управляющие вводом/выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда **print** в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода автоматически. Должна быть и расположена в директории языка.

* 1. **Точка входа**

В языке SYO-2020 каждая программа должна содержать главную функцию (точку входа) **main**, с первой инструкции которой начнётся последовательное выполнение команд программы.

Должна иметься только одна точка входа main.

* 1. **Препроцессор**

Команды препроцессора в языке SYO-2020 отсутствуют.

* 1. **Соглашения о вызове**

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.

* 1. **Объектный код**

Язык SYO-2020 транслируется в язык ассемблера, а затем - в объектный код.

* 1. **Классификация сообщений транслятора**

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения об ошибках имеют специфический постфикс, зависящий от этапа, на котором обнаружена ошибка.

Список постфиксов приведен в таблице 1.9.

Таблица 1.9 – Список префиксов ошибок в языке SYO-2020

|  |  |
| --- | --- |
| Постфикс | Пояснение |
| [SYN#] | Указывает, что ошибка была обнаружена на стадии синтаксического анализа. |
| [LEX#] | Указывает, что ошибка была обнаружена на стадии лексического анализа. |
| [SEM#] | Указывает, что ошибка была обнаружена на стадии семантического анализа. |

* 1. **Контрольный пример**

Контрольный пример демонстрирует главные особенности языка SYO-2020: его фундаментальные типы, основные структуры, функции, использование функция стандартной библиотеки. Исходный код контрольного примера представлен в приложении А.

**Глава 2. Структура транслятора**

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

В языке SYO-2020 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка SYO-2020 приведена на рисунке 2.1.



Рисунок 2.1 Структура транслятора языка программирования SYO-2020

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

* убрать все лишние пробелы;
* выполнить распознавание лексем;
* построить таблицу лексем и таблицу идентификаторов;
* при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

**2.2 Перечень входных параметров транслятора**

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка SYO-2020

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке SYO-2020 , имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |

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

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы.

В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 – Протоколы, формируемые транслятором языка SYO-2020

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования SYO-2020 . 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. |

**Глава 3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка,. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

* удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);
* распознавание идентификаторов и ключевых слов;
* распознавание констант;
* распознавание разделителей и знаков операций.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора

**3.2. Контроль входных символов**

Таблица контроля входных символов представлена в приложении Б

Принцип работы таблицы заключается в соответствии значения каждого элемента значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, S – символ-разделитель, SPC – пробельные символы, LX – символы арифметических операций, скобки и запятые.

**3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

- посимвольно считываем файл с исходным кодом программы;

- в отличие от других символов-разделителей, не записываем пробелы и символы табуляции в таблицу лексем;

- продолжаем считывание файла с исходным кодом программы до встречи с лексемой, отличной от пробела или символа табуляции.

**3.4 Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.1.

Таблица 3.1 – Соответствие токенов и лексем в языке SYO-2020

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| number, text | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 10 символов. |
| Литерал | l | Литерал любого доступного типа. |
| Восьмиричный литерал | l | Литерал в восьмиричном представлении. |
| func | f | Объявление функции. |
| return | r | Выход из функции/процедуры. |
| main | m | Главная функция. |
| new | n | Объявление переменной. |
| print | p | Ввод данных. |
| state: | ? | Указывает начало цикла/условного оператора. |
| repeat | c | Указывает на начало тела цикла. |
| printline | ^ | Оператор вывода символа перевода строки. |
| # | # | Разделение конструкций в цикле/условном операторе. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока/тела функции. |
| ] | ] | Закрытие блока/тела функции. |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +  -  \*  /  }  {  >  <  !  %  & | +  -  \*  /  }  {  >  <  !  %  & | Знаки операций. |

В приложении Б находится пример конечного автомата, используемый для разбора цепочки символов.

**3.5 Основные структуры данных**

Структуры таблиц лексем и идентификаторов данных языка SYO-2020, используемых для хранения, представлены в приложении Б.

В таблице лексем содержатся сами лексемы, строка для каждой лексемы, в которой она была замечена. Так же размер самой таблицы лексем В таблице идентификаторов содержится имя идентификатора, его номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение, а также имя родительской функции.

**3.6 Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами.

В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## **3.7 Структура и перечень сообщений лексического анализа**

Перечень сообщений представлен в приложении Б.

Сообщения об ошибках данной стадии имеют префикс [LEX#] что с легкостью дает пользователю понять, на каком этапе возникла ошибка.

## **3.8 Параметры лексического анализатора**

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, так и в командную строку.

**3.9 Алгоритм лексического анализа**

Последовательность выполнения алгоритма работы лексического анализатора представлен ниже.

1. Разделение текста на отдельные лексемы.
2. Распознавание каждой строки в двумерном массиве с помощью автоматов.
3. При удачном прохождении информация заносится в таблицу лексем и идентификаторов. Возврат к шагу 2).
4. Формирование протокола работы
5. При невозможности обработать строку двумерного массива выводится сообщение об ошибке.
6. Конец работы лексического анализатора

**3.10 Контрольный пример**

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

**Глава 4. Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

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

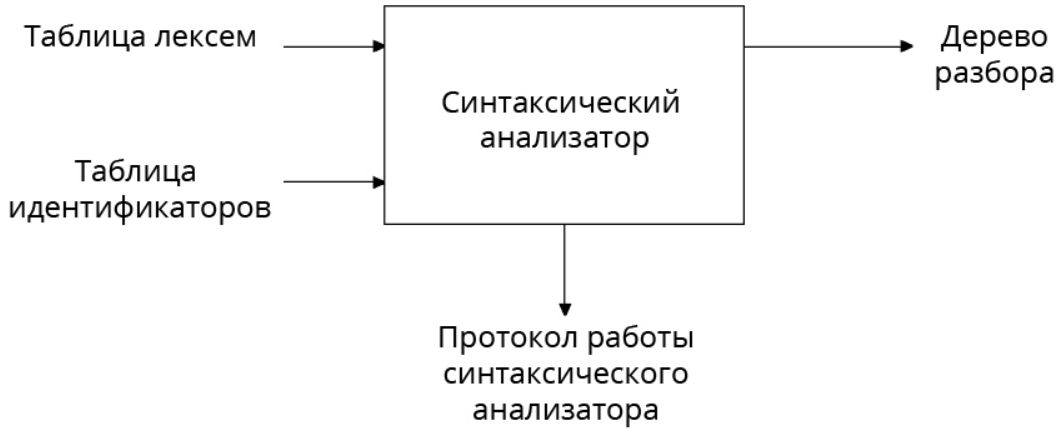
Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка SYO-2020 используется контекстно-свободная грамматика , 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.

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 – Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->tfiPTS  S->pfiPGS  S->m[K] | Стартовые правила, описывающее общую структуру программы |
| P | P->(E)  P->() | Правила для параметров объявляемых функций |
| T | T->[eV;]  T->[KeV;] | Правила для тела функций |
| E | E->ti,E  E->ti | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вывозов функций(в т.ч. и в выражениях) |
| N | N->i  N>l  N->l,N  N->I,N | Правила для параметров вызываемых функций |
| R | R->rY#  R>wY#  R>cY#  R->rYwY#  R->wYrY# | Правила составления цикла/условного оператора |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла/условного оператора |
| L | L->}  L->{  L->!  L->>  L-><  L->& | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->{  A->} | Правила для арифметических операторов |
| V | V->l  V->i | Правила для простых выражений |
| Y | Y->[X] | Правила для тела цикла/условного выражения |
| W | W->l  W->i  W->(W) | Правила для сложных выражений |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| W | W->(W)AW  W->iF  W->iAW  W->lAW  W->iFAW | Правила для сложных выражений |
| K | K->nti=V;K  K->nti;K  K->i=W;K  K->oV;K  K->^;K  K->&Z#RK  K->iF;K  K->nti=V;  K->nti;  K->i=W;  K->oV;  K->^;  K->&Z#R  K->iF; | Программные конструкции |
| X | X->i=W;X  X->oV;X  X->^;X  X->iF;X  X->i=W;  X->oV;  X->^;  X->iF; | Программные конструкции внутри цикла/условного оператора |

**4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку.

Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | 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 |
|  | 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 |

**4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка. Данные структуры в приложении В.

**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. Иначе генерируется исключение.

**4.6 Структура и перечень сообщений синтаксического анализатора**

Сообщения генерируемые синтаксическим анализатором представлены в приложении В.

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

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

**4.8. Принцип обработки ошибок**

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

**4.9. Контрольный пример**

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

**Глава 5. Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

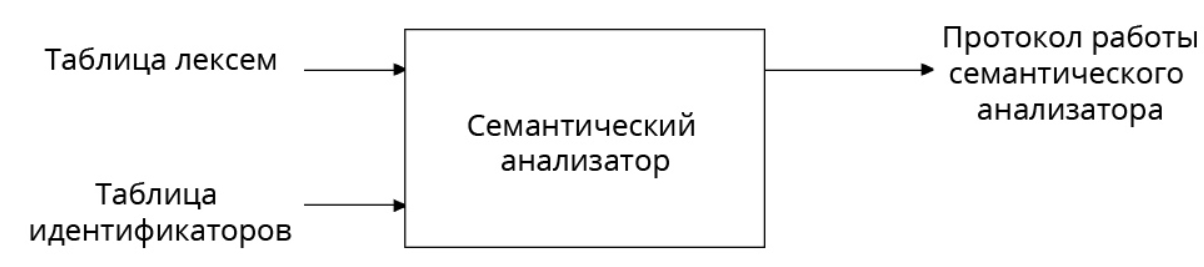


Рисунок 5.1. Структура семантического анализатора

**5.2 Функции семантического анализатора**

Семантический анализатор проверяет правильность составления программных конструкций. При невозможности подобрать правило перехода будет выведен код ошибки, а так же код этой ошибки. Информация об ошибках выводится в консоль, а так же в протокол работы.

**5.3 Структура и перечень семантических ошибок**

Сообщения, формируемые семантическим анализатором, представлены в приложении Г.

## **5.4 Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после того, как будут найдены все ошибки.

## **5.5 Контрольный пример**

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1 – Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main[  number x = 9;  print x;  ] | Ошибка 304: [SEM]# В объявлении отсутствует ключевое слово new Строка: 2 |
| main[  new number x = 9;  new text y =x;  ] | Ошибка 314: [SEM]# Типы данных в выражении не совпадают Строка: 3 |
| main[  new number x = 9;  ]  main[  new text y = "qwerty";  ] | Ошибка 302: [SEM]# Обнаружено несколько точек входа main Строка: 0 |

**Глава 6. Вычисление выражений**

**6.1 Выражения, допускаемые языком**

В языке SYO-2020 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 – Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 0 |
| \* | 3 |
| / | 3 |
| + | 2 |
| - | 2 |
| % | 3 |

**6.2 Польская запись и принцип ее построения**

Все выражения языка SYO-2020 преобразовываются к обратной польской записи.

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов. Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции;

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2.

Преобразование выражений в формат польской записи в нашем случае необходимо для построения более простых алгоритмов при последующей обработки таблицы лексем.

Таблица 6.2 – Преобразование выражений к ПОЛИЗ

|  |  |  |
| --- | --- | --- |
| Исходное выражение | Стек | Результирующая строка |
| ( i - (l \* i)) - l |  |  |
| i - (l \* i)) - l | ( |  |
| - (l \* i)) - l | ( | i |
| (l \* i)) - l | -( | i |
| l \* i)) - l | (-( | i |
| \* i)) - l | (-( | il |
| i)) - l | \*(-( | il |
| )) - l | \*(-( | ili |
| ) - l | -( | ili\* |
| - l |  | ili\*- |
| l | - | ili\*- |
|  |  | ili\*-l- |

В приложении Г приведена изменённая таблица лексем, отображающая результаты преобразования выражений в польский формат.

**Глава 7. Генерация кода**

**7.1 Структура генератора кода**

В языке SYO-2020 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода SYO-2020 представлена на рисунке 7.1.

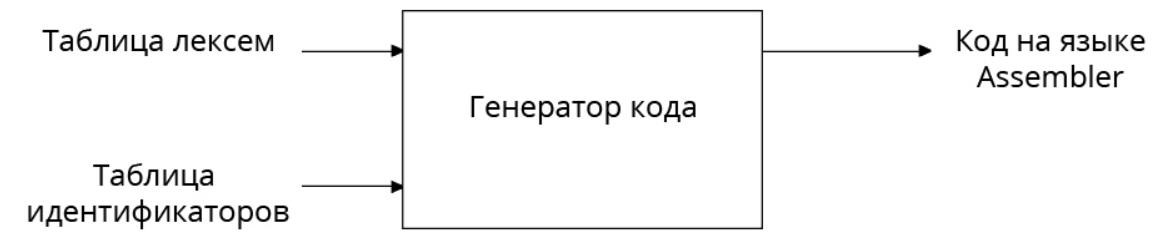


Рисунок 7.1 – Структура генератора кода

**7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке SYO-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SYO-2020 | Тип идентификатора на языке ассемблера | Пояснение |
| number | sdword | Хранит целочисленный тип данных. |
| text | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

**7.3 Статическая библиотека**

В языке SYO-2020 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Стандартная библиотека находится в директории языка и при генерации кода подключается автоматически. Путь к библиотеке генерируется автоматически на стадии генерации кода.

Функции статической библиотеки приведены в таблице 7.3.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void outlstr(char\* str) | Вывод на консоль строки str |
| void outnum(int num) | Вывод на консоль целочисленной переменной num |
| int sqroot(int num) | Возвращает квадратный корень числа |
| int module (int num) | Возвращает модуль числа |
| int input(int num) | Ввод числа из консоли |

**7.4 Особенности алгоритма генерации кода**

В процессе генерации используются векторы и строки. Отдельные сегменты сначала записываются в строки, а затем отправляются в вектор. В конце работы весь вектор последовательно выводится в файл.

**7.5 Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке SYO-2020 . Результаты работы генератора кода выводятся в файл с расширением .asm.

**7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён в приложении Д.

**Глава 8. Тестирование транслятора**

**8.1 Общие положения**

В языке SYO-2020 при возникновении ошибки на одном из этапов она обрабатывается в главной функции. Затем код ошибки и сообщение выводится в консольное окно, а так же записывается в протокол работы.

**8.2 Результаты тестирования**

В таблице 8.1 приведены ошибки возникающие при считывании из файла, а так же на стадии лексического, синтаксического и семантического анализа.

Таблица 8.1 – Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  new text a = 'hello';  ] | 600: строка 1, [SYN]# Неверная структура программы |
| main  [  new number x = 2;  state: x < 15  repeat [  x = x + 1;  ] #  ] | 610: строка 4, [SYN]# Неверный условный оператор |
| main  [  new number x = 2;  state: x < 15 #  repeat [  x = x + 1;  ]  ] | 615: строка 7, [SYN]# Недопустимая синтаксическая конструкция в теле цикла/условного выражения |
| main  [  new number x = 2;  new text k = 'hello';  x = x + k;  ] | Ошибка 314: [SEM]# Типы данных в выражении не совпадают  Строка 5 позиция -1 |
| [  ] | Ошибка 301: [SEM]# Отсутствует точка входа main  Строка -1 позиция -1 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| main  [  new number a;  new text b;  a = sqroot(b);  print a;  ] | Ошибка 309: [SEM]# Несовпадение типов передаваемых параметров  Строка 5 позиция -1 |
| main  [  вввввввввв;  new number a;  print a;  ] | Ошибка 200: [LEX]# Невозможно распознать цепочку сиволов  Строка 3 позиция -1 |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования SYO-2020 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка SYO-2020;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка SYO-2020 включает:

1. 2 типа данных;
2. Поддержка операторов ввода и перевода строки;
3. Наличие 5 арифметических операторов для вычисления выражений
4. Наличие 5 логических операторов для использования в условиях цикла и условной конструкции
5. Поддержка функций; Операторов цикла и условия;
6. Наличие библиотеки стандартных функций языка
7. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

# **Список использованных источников**

1. Курс лекций по ЯП Наркевич А.С.

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

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

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

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

**Приложение А**

Листинг 1 – Исходный код программы на языке SYO-2020

|  |
| --- |
| number func odd(number t, number n)  [  new number i;  state: t < n #  repeat [  i = i + 1;  print t;  print ' ';  t = t + 2;  ] #  return i;  ]  number func max(number a, number b)  [  new number res;  state: a > b #  truest [res = a;]  falsest [res = b;] #  return res;  ]  main  [  new number a;  print 'Введите число нижний предел: ';  a = input(a);  print 'введите число верхний предел: ';  new number b;  b = input(b);  b = odd(a, b);  new number k;  k = max(a, b);  new number c = q144;  print 'результат после вычисления выражения: ';  c = (c - (2 \* c)) - 125;  print c;  c = module(c);  printline;  print 'модуль результата: ';  print c;  printline;  print 'квадратный корень результата: ';  c = sqroot(c);  print c;  printline;  print 'остаток от деления на 2 результата квадратного корня: ';  c = c % 2;  print c;  ] |

**Приложение Б**

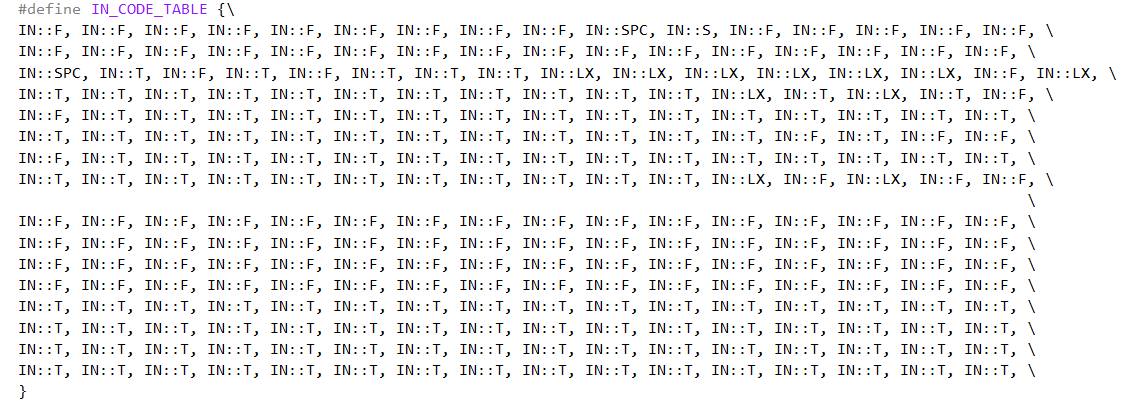


Рисунок 1 - Таблица контроля входных символов

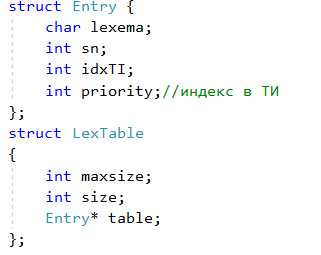


Рисунок 2 – Структура таблицы лексем

Листинг 1 – Пример конечного автомата

|  |
| --- |
| #define FST\_PRINT 6, \  FST::NODE(1, FST::RELATION('p', 1)),\  FST::NODE(1, FST::RELATION('r', 2)),\  FST::NODE(1, FST::RELATION('i', 3)),\  FST::NODE(1, FST::RELATION('n', 4)),\  FST::NODE(1, FST::RELATION('t', 5)),\  FST::NODE() |

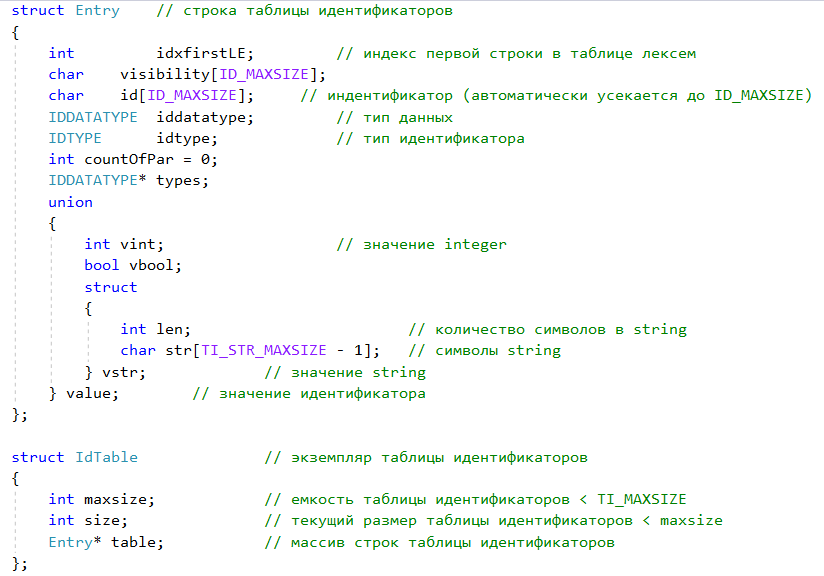


Рисунок 3 – Структура таблицы идентификаторов

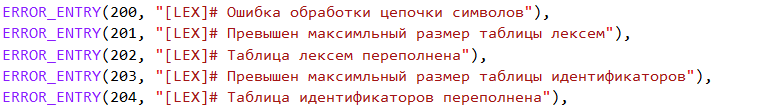


Рисунок 3 – Сообщения об ошибках стадии лексического анализа

Листинг 2 – Таблица идентификаторов контрольного примера

|  |
| --- |
| ---------------------------------------------------------------------------------------  № | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение  ---------------------------------------------------------------------------------------  0000 | odd | number | функция | 2 | -  0001 | oddt | number | параметр | 5 | -  0002 | oddn | number | параметр | 8 | -  0003 | oddi | number | переменная | 13 | 0  0004 | L1 | number | литерал | 26 | 1  0005 | L2 | text | литерал | 32 | [1]" "  0006 | L3 | number | литерал | 38 | 2  0007 | max | number | функция | 48 | -  0008 | maxa | number | параметр | 51 | -  0009 | maxb | number | параметр | 54 | -  0010 | maxres | number | переменная | 59 | 0 |
| 0011 | maina | number | переменная | 89 | 0  0012 | L4 | text | литерал | 92 | [29]"Введите число нижний предел: "  0013 | input | number | стандартная функция | 96 | -  0014 | L5 | text | литерал | 102 | [30]"введите число верхний предел: "  0015 | mainb | number | переменная | 106 | 0  0016 | maink | number | переменная | 126 | 0  0017 | mainc | number | переменная | 139 | 0  0018 | L6 | number | литерал | 141 | 100  0019 | L7 | text | литерал | 144 | [38]"результат после вычисления выражения: "  0020 | L8 | number | литерал | 158 | 125  0021 | module | number | стандартная функция | 165 | -  0022 | L9 | text | литерал | 173 | [19]"модуль результата: "  0023 | L10 | text | литерал | 181 | [30]"квадратный корень результата: "  0024 | sqroot | number | стандартная функция | 185 | -  0025 | L11 | text | литерал | 196 | [54]"остаток от деления на 2 результата квадратного корня: "  ---------------------------------------------------------------------------------------  Количество идентификаторов: 26  --------------------------------------------------------------------------------------- |

Листинг 3 – Таблица лексем контрольного примера

|  |  |
| --- | --- |
| tfi(ti,ti)[nti;  ?i<i#  c[i=i+l;  pi;  pl;  i=i+l;  ]#  ri;  ]tfi(ti,ti)[nti;  ?i>i#  u[i=i;  ]w[i=i;  ]#  ri;  ]m[nti;  pl;  i=i(i);  pl;  nti;  i=i(i);  i=i(i,i); | nti;  i=i(i,i);  nti=l;  pl;  i=(i-(l\*i))-l;  pi;  i=i(i);  ^;  pl;  pi;  ^;  pl;  i=i(i);  pi;  ^;  pl;  i=i%l;  pi; |

**Приложение В**

Листинг 1 – Грамматика языка SYO-2020

reibach greibach(NS('S'), TS('$'), 16,

Rule(NS('S'), GRB\_ERROR\_SERIES, 3, // Неверная структура программы

Rule::Chain(6, TS('t'), TS('f'), TS('i'), NS('P'), NS('T'), NS('S')),

Rule::Chain(4, TS('m'), TS('['), NS('K'), TS(']'))

),

Rule(NS('T'), GRB\_ERROR\_SERIES + 2, 2, // Ошибка в теле функции

Rule::Chain(5, TS('['), TS('e'), NS('V'), TS(';'), TS(']')),

Rule::Chain(6, TS('['), NS('K'), TS('e'), NS('V'), TS(';'), TS(']'))

),

Rule(NS('P'), GRB\_ERROR\_SERIES + 1, 2, // Не найден список параметров функции

Rule::Chain(3, TS('('), NS('E'), TS(')')),

Rule::Chain(2, TS('('), TS(')'))

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 4, 2, // Ошибка в списке параметров функции

Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('E')),

Rule::Chain(2, TS('t'), TS('i'))

),

Rule(NS('F'), GRB\_ERROR\_SERIES + 5, 2, // Ошибка в вызове функции

Rule::Chain(3, TS('('), NS('N'), TS(')')),

Rule::Chain(2, TS('('), TS(')'))),

Rule(NS('N'), GRB\_ERROR\_SERIES + 6, 4, // Ошибка в списке параметров функции

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('i'), TS(','), NS('N')),

Rule::Chain(3, TS('l'), TS(','), NS('N'))),

Rule(NS('R'), GRB\_ERROR\_SERIES + 7, 5, // Ошибка при констуировании цикла/условного выражения

Rule::Chain(3, TS('u'), NS('Y'), TS('#')),

Rule::Chain(3, TS('w'), NS('Y'), TS('#')),

Rule::Chain(3, TS('c'), NS('Y'), TS('#')),

Rule::Chain(5, TS('u'), NS('Y'), TS('w'), NS('Y'), TS('#')),

Rule::Chain(5, TS('w'), NS('Y'), TS('r'), NS('Y'), TS('#'))),

Rule(NS('Y'), GRB\_ERROR\_SERIES + 8, 1, // Ошибка в теле цикла/условного выражения

Rule::Chain(3, TS('['), NS('X'), TS(']'))),

Rule(NS('Z'), GRB\_ERROR\_SERIES + 9, 3, // Ошибка в условии цикла/условного выражения

Rule::Chain(3, TS('i'), NS('L'), TS('i')),

Rule::Chain(3, TS('i'), NS('L'), TS('l')),

Rule::Chain(3, TS('l'), NS('L'), TS('i'))),

Rule(NS('L'), GRB\_ERROR\_SERIES + 10, 6,// Неверный условный оператор

Rule::Chain(1, TS('<')),

Rule::Chain(1, TS('>')),

Rule::Chain(1, TS('!')),

Rule::Chain(1, TS('{')),

Листинг 1(продолжение) – Грамматика языка SYO-2020

Rule::Chain(1, TS('}')),

Rule::Chain(1, TS('&'))

),

Rule(NS('A'), GRB\_ERROR\_SERIES + 11, 5, // Неверный арифметический оператор

Rule::Chain(1, TS('+')),

Rule::Chain(1, TS('-')),

Rule::Chain(1, TS('\*')),

Rule::Chain(1, TS('/')),

Rule::Chain(1, TS('%'))

),

Rule(NS('V'), GRB\_ERROR\_SERIES + 12, 2, // Неверное выражение. Rule::Chain(1, TS('l')),

Rule::Chain(1, TS('i'))

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 13, 8,// Ошибка в арифметичском выражении

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('('), NS('W'), TS(')')),

Rule::Chain(5, TS('('), NS('W'), TS(')'), NS('A'), NS('W')),

Rule::Chain(2, TS('i'), NS('F')),

Rule::Chain(3, TS('i'), NS('A'), NS('W')),

Rule::Chain(3, TS('l'), NS('A'), NS('W')),

Rule::Chain(4, TS('i'), NS('F'), NS('A'), NS('W'))),

Rule(NS('K'), GRB\_ERROR\_SERIES + 14, 14,// Недопустимая синтаксическая конструкция

Rule::Chain(7, TS('n'), TS('t'), TS('i'), TS('='), NS('V'), TS(';'), NS('K')), // декларация + присваивание

Rule::Chain(5, TS('n'), TS('t'), TS('i'), TS(';'), NS('K')), // декларация

Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('K')), // присваивание

Rule::Chain(4, TS('o'), NS('V'), TS(';'), NS('K')), // вывод

Rule::Chain(3, TS('^'), TS(';'), NS('K')), // перевод строки

Rule::Chain(5, TS('?'), NS('Z'), TS('#'), NS('R'), NS('K')), // state

Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('K')), // вызов функции

Rule::Chain(6, TS('n'), TS('t'), TS('i'), TS('='), NS('V'), TS(';')), // декларация + присваивание

Rule::Chain(4, TS('i'), TS('='), NS('W'), TS(';')), // присваивание

Rule::Chain(4, TS('n'), TS('t'), TS('i'), TS(';')), // декларация

Rule::Chain(3, TS('p'), NS('V'), TS(';')), // вывод

Rule::Chain(2, TS('^'), TS(';')), // перевод строки

Rule::Chain(4, TS('?'), NS('Z'), TS('#'), NS('R')), // state

Rule::Chain(3, TS('i'), NS('F'), TS(';')) // вызов функции

),

Rule(NS('X'), GRB\_ERROR\_SERIES + 15, 8, // Недопустимая синтаксическая конструкция в теле цикла/условного выражения

Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('X')), //присваивание

Rule::Chain(4, TS('p'), NS('V'), TS(';'), NS('X')), // вывод

Rule::Chain(3, TS('^'), TS(';'), NS('X')), // перевод строки

Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('X')), // вызов функции

Rule::Chain(4, TS('i'), TS('='), NS('W'), TS(';')), // присваивание

Rule::Chain(3, TS('p'), NS('V'), TS(';')), // вывод

Rule::Chain(2, TS('^'), TS(';')), // перевод строки

Rule::Chain(3, TS('i'), NS('F'), TS(';')) // вызов функции

)

);

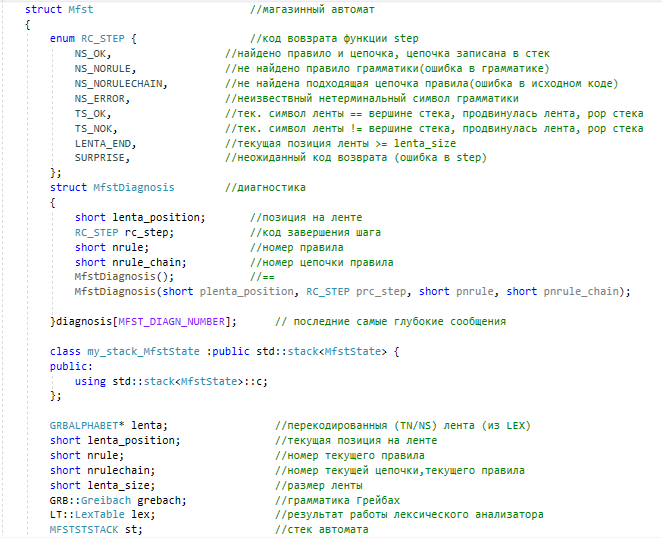


Рисунок 1 – Структура магазинного автомата

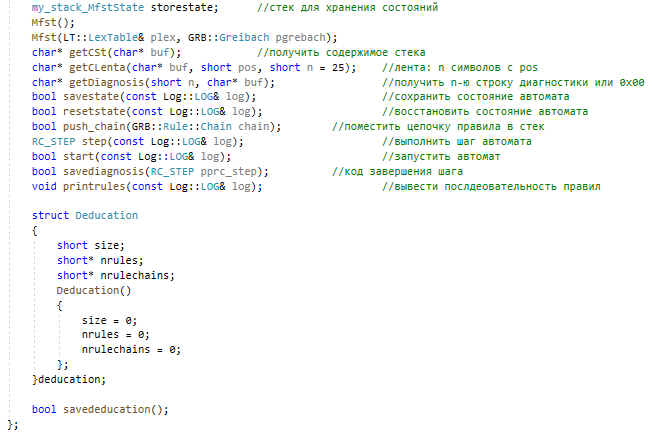


Рисунок 2 – Структура магазинного автомата (продолжение)

Листинг 2 – Разбор исходного кода синтаксическим анализатором

|  |
| --- |
| 0 : S->tfiPTS S$  0 : SAVESTATE: 1  0 : tfiPTS$  1 : fi(ti,ti)[nti;?i<i#c[i=i+ fiPTS$  2 : i(ti,ti)[nti;?i<i#c[i=i+l iPTS$ |

Листинг 2 (продолжение) – Разбор исходного кода синтаксическим анализатором

|  |
| --- |
| 1379: i;]]]]]]]]]]]]]]]]]]]]]]] i;K]$  1380: ;]]]]]]]]]]]]]]]]]]]]]]]] ;K]$  1381: ]]]]]]]]]]]]]]]]]]]]]]]]] K]$  1382: TNS\_NORULECHAIN/NS\_NORULE  1382: RESSTATE  1382: i;]]]]]]]]]]]]]]]]]]]]]]] V;K]$  1383: TNS\_NORULECHAIN/NS\_NORULE  1383: RESSTATE  1383: pi;]]]]]]]]]]]]]]]]]]]]]] K]$  1384: K->pV; pi;]]]]]]]]]]]]]]]]]]]]]] K]$  1384: SAVESTATE: 109  1384: pi;]]]]]]]]]]]]]]]]]]]]]] pV;]$  1385: i;]]]]]]]]]]]]]]]]]]]]]]] V;]$  1386: V->i i;]]]]]]]]]]]]]]]]]]]]]]] V;]$  1386: SAVESTATE: 110  1386: i;]]]]]]]]]]]]]]]]]]]]]]] i;]$  1387: ;]]]]]]]]]]]]]]]]]]]]]]]] ;]$  1388: ]]]]]]]]]]]]]]]]]]]]]]]]] ]$  1389: ]]]]]]]]]]]]]]]]]]]]]]]]] $  1390: 6  1391: ------>LENTA\_END |

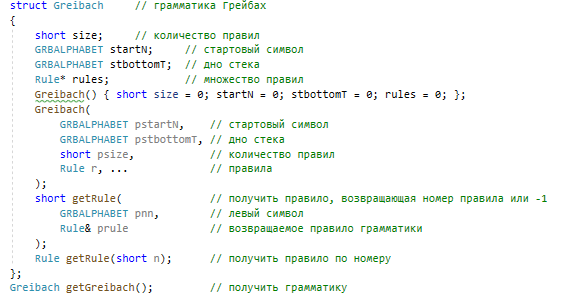


Рисунок 3 – Структура грамматики Грейбах

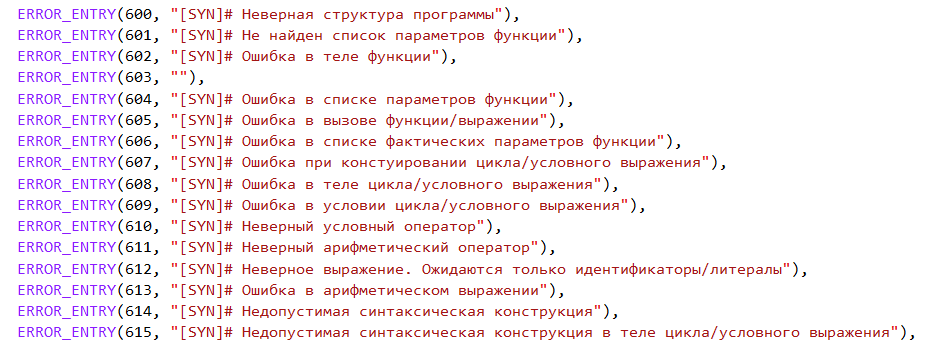


Рисунок 4 – Сообщения об ошибках стадии синтаксического анализа

# **Приложение Г**

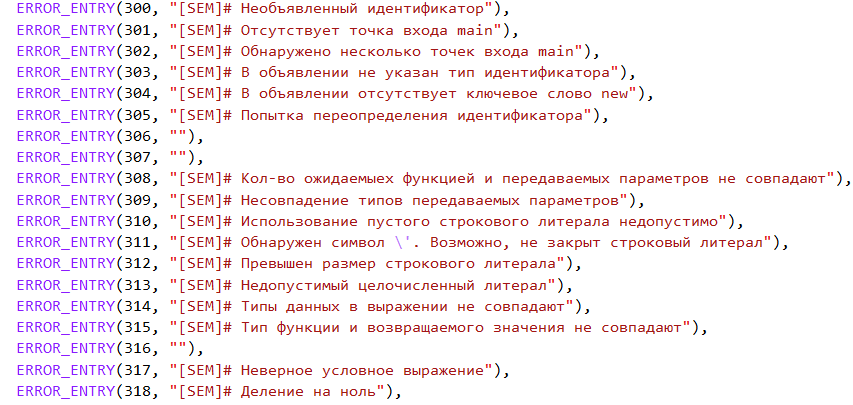


Рисунок 1 – Сообщения об ошибках стадии семантического анализа

Листинг 1 – Программная реализация механизма преобразования в ПОЛИЗ

|  |
| --- |
| **bool** **PolishNotation**(**int** i, Lex::LEX& lex)  {  stack<LT::Entry> stack; // стек для операций  queue<LT::Entry> queue; // очередь для операндов  LT::Entry temp; // запрещенная лексема  temp.idxTI = -**1**;  temp.lexema = ' ';  temp.sn = -**1**;  LT::Entry func;  func.lexema = '^';  LT::Entry num;  num.lexema = ' ';  **int** countLex = **0**; // количество преобразованных лексем  **int** posLex = i; // запоминаем номер лексемы перед преобразованием  **bool** findFunc = false;  **int** buf = **0**;  **for** (i; lex.lextable.table[i].lexema != LEX\_SEMICOLON; i++, countLex++)  {  **switch** (lex.lextable.table[i].lexema){  **case** LEX\_ID: // если идентификатор  {  queue.push(lex.lextable.table[i]);  **continue**;  }  **case** LEX\_LITERAL: {  queue.push(lex.lextable.table[i]); //в очередь  **continue**;  }  **case** LEX\_COMMA: |

Листинг 1 (Прод.) – Программная реализация механизма преобразования в ПОЛИЗ

|  |
| --- |
| {  **if** (findFunc)  stack.push(lex.lextable.table[i]);  }  **case** LEX\_LEFTTHESIS: // если (  {  **if** (lex.lextable.table[i-**1**].lexema == LEX\_ID/\*lex.idtable.table[lex.lextable.table[i - 1].idxTI].idtype == IT::F || lex.idtable.table[lex.lextable.table[i - 1].idxTI].idtype == IT::SF\*/)  {  **if** (lex.idtable.table[lex.lextable.table[i - **1**].idxTI].idtype == IT::F || lex.idtable.table[lex.lextable.table[i - **1**].idxTI].idtype == IT::SF) {  findFunc = true;  num.lexema = (**char**)lex.idtable.table[lex.lextable.table[i - **1**].idxTI].countOfPar + '0';  }  }  **if** (findFunc)  queue.push(lex.lextable.table[i]);  **else**  stack.push(lex.lextable.table[i]); //в стек  **continue**;  }  **case** LEX\_RIGHTTHESIS: // если )  {  **if** (findFunc)  queue.push(lex.lextable.table[i]);  **else** {  **while** (stack.top().lexema != LEX\_LEFTTHESIS) // пока не встретим (  {  queue.push(stack.top()); //со стека в очередь  stack.pop();  **if** (stack.empty())  **return** false;  }  stack.pop(); // уничтожаем (  }  **continue**;  }  **case** LEX\_PLUS: **case** LEX\_MINUS: **case** LEX\_STAR: **case** LEX\_DIRSLASH: **case** LEX\_PROC: // если знак оператора  {  **while** (!stack.empty() && lex.lextable.table[i].priority <= stack.top().priority) // пока приоритет текущего оператора //меньше или равен приоритету оператора в вершине стека  {  queue.push(stack.top()); // выталкиваем со стека в выходную строку  stack.pop();  }  stack.push(lex.lextable.table[i]);  **continue**;}  } |

Листинг 1 (Прод.) – Программная реализация механизма преобразования в ПОЛИЗ

|  |
| --- |
| }  **while** (!stack.empty()) // если стек не пустой  {  **if** (stack.top().lexema == LEX\_LEFTTHESIS || stack.top().lexema == LEX\_RIGHTTHESIS)  **return** false;  queue.push(stack.top()); // выталкиваем все в очередь  stack.pop();  }  **while** (countLex != **0**) // замена текущего выражения в таблице лексем на выражение в ПОЛИЗ  {  **if** (!queue.empty()) {  lex.lextable.table[posLex++] = queue.front();  queue.pop();  }  **else**  {  lex.lextable.table[posLex++] = temp;  }  countLex--;  }  **for** (**int** i = **0**; i < posLex; i++) // восстановление индексов первого вхождения в таблицу лексем у операторов из таблицы идентификаторов  {  **if** (lex.lextable.table[i].lexema == LEX\_PLUS || lex.lextable.table[i].lexema == LEX\_MINUS || lex.lextable.table[i].lexema == LEX\_STAR || lex.lextable.table[i].lexema == LEX\_DIRSLASH || lex.lextable.table[i].lexema == LEX\_LITERAL)  lex.idtable.table[lex.lextable.table[i].idxTI].idxfirstLE = i;  }  **return** true;  } |

Листинг 2 – измененная таблица лексем после преобразования выражений

|  |  |  |
| --- | --- | --- |
| tfi(ti,ti)[nti;  ?i<i#  c[i=il+;  pi;  pl;  i=il+;  ]#  ri;  ]tfi(ti,ti)[nti;  ?i>i#  u[i=i;  ]w[i=i;  ]#  ri; | ]m[nti;  pl;  i=i(i);  pl;  nti;  i=i(i);  i=i(i,i);  nti;  i=i(i,i);  nti=l;  pl;  i=ili\*-l- ;  pi;  i=i(i); | ^;  pl;  pi;  ^;  pl;  i=i(i);  pi;  ^;  pl;  i=il%;  pi;  ] |

# **Приложение Д**

Листинг 1 – Результат генерации кода контрольного примера в Ассемблер

|  |  |
| --- | --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib "D:\User\Desktop\CourseWork\SYO-2020\Generation\Debug\GenLib.lib  ExitProcess PROTO:DWORD  .stack 4096  outnum PROTO : DWORD  outstr PROTO : DWORD  sqroot PROTO : DWORD  module PROTO : DWORD  input PROTO : DWORD  .const  printline byte 13, 10, 0  L1 sdword 1  L2 byte ' ', 0  L3 sdword 2  L4 byte 'Введите число нижний предел: ', 0  L5 byte 'введите число верхний предел: ', 0  L6 sdword 100  L7 byte 'результат после вычисления выражения: ', 0  L8 sdword 125  L9 byte 'модуль результата: ', 0  L10 byte 'квадратный корень результата: ', 0  L11 byte 'остаток от деления на 2 результата квадратного корня: ', 0  .data  temp sdword ?  buffer byte 256 dup(0)  oddi sdword 0  maxres sdword 0  maina sdword 0  mainb sdword 0  maink sdword 0  mainc sdword 0  .code  ;----------- odd ------------  odd PROC,  oddt : sdword, oddn : sdword  ; --- save registers ---  push ebx  push edx  ; --- save registers ---  push ebx  push edx  ; ----------------------  mov edx, maxa  cmp edx, maxb  jg right2  jl wrong2  right2:  push maxa  pop ebx  mov maxres, ebx  jmp next2  wrong2:  push maxb  pop ebx  mov maxres, ebx  next2:  ; --- restore registers ---  pop edx  pop ebx  ; -------------------------  mov eax, maxres  ret  max ENDP  ;------------------------------  ;----------- MAIN ------------  main PROC  push offset L4  call outstr  push maina  call input  push eax  pop ebx  mov maina, ebx  push offset L5  call outstr  push mainb  call input  push eax  pop ebx  pop ebx  mov mainc, ebx  push offset printline  call outstr  push offset L9  call outstr  push mainc  call outnum  push offset printline  call outstr  push offset L10  call outstr  push mainc  call sqroot  push eax  pop ebx  mov mainc, ebx  push mainc  call outnum  push offset printline  call outstr  push offset L11  call outstr  push mainc  push L3  pop ebx  pop eax  cdq  idiv ebx  push edx  pop ebx  mov mainc, ebx  push mainc  call outnum | ; ----------------------  mov edx, oddt  cmp edx, oddn  jl repeat1  jmp repeatnext1  repeat1:  push oddi  push L1  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov oddi, ebx  push oddt  call outnum  push offset L2  call outstr  push oddt  push L3  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov oddt, ebx  mov edx, oddt  cmp edx, oddn  jl repeat1  repeatnext1:  ; --- restore registers ---  pop edx  pop ebx  ; -------------------------  mov eax, oddi  ret  odd ENDP  ;------------------------------  ;----------- max ------------  max PROC,  maxa : sdword, maxb : sdword  mov mainb, ebx  push mainb  push maina  call odd  push eax  pop ebx  mov mainb, ebx  push mainb  push maina  call max  push eax  pop ebx  mov maink, ebx  push L6  pop ebx  mov mainc, ebx  push offset L7  call outstr  push mainc  push L3  push mainc  pop ebx  pop eax  imul eax, ebx  push eax  pop ebx  pop eax  sub eax, ebx  push eax  push L8  pop ebx  pop eax  sub eax, ebx  push eax  pop ebx  mov mainc, ebx  push mainc  call outnum  push mainc  call module  push eax  push 0  call ExitProcess  main ENDP  end main |

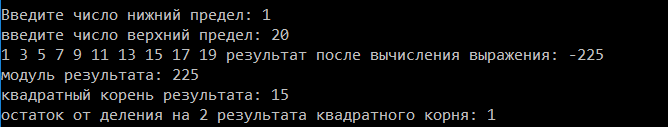


Рисунок 1 – Результат работы программы на языке SYO-2020