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 ААА-2018»

Выполнил студент Астахова Анастасия Алексеевна

(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 2017

**Содержание**

[Введение 6](#_Toc469951039)

[Глава 1. Спецификация языка программирования 7](#_Toc469951040)

[1.1 Характеристика языка программирования 7](#_Toc469951041)

[1.2 Алфавит языка 7](#_Toc469951042)

[1.3 Применяемые сепараторы 7](#_Toc469951043)

[1.4 Применяемые кодировки 8](#_Toc469951044)

[1.5 Типы данных 8](#_Toc469951045)

[1.6 Преобразование типов данных 8](#_Toc469951046)

[1.7 Идентификаторы 9](#_Toc469951047)

[1.8 Литералы 9](#_Toc469951048)

[1.9 Объявление данных и область видимости 9](#_Toc469951049)

[1.10 Инициализация данных 9](#_Toc469951050)

[1.11 Инструкции языка 10](#_Toc469951051)

[1.12 Операции языка 10](#_Toc469951052)

[1.13 Выражения и их вычисления 11](#_Toc469951053)

[1.14 Программные конструкции языка 12](#_Toc469951054)

[1.15 Область видимости идентификаторов 12](#_Toc469951055)

[1.16 Семантические проверки 12](#_Toc469951056)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc469951057)

[1.18 Стандартная библиотека и её состав 13](#_Toc469879811)

[1.19 Ввод и вывод данных 13](#_Toc469879812)

[1.20 Точка входа 13](#_Toc469879813)

[1.21 Препроцессор 13](#_Toc469879814)

[1.22 Соглашения о вызовах 13](#_Toc469879815)

[1.23 Объектный код 13](#_Toc469879816)

[1.24 Классификация сообщений транслятора 14](#_Toc469879817)

[1.25 Контрольный пример 14](#_Toc469879818)

[Глава 2. Структура транслятора 14](#_Toc469951058)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc469951059)

[2.2 Перечень входных параметров транслятора 15](#_Toc469951060)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc469951061)

[Глава 3. Разработка лексического анализатора 17](#_Toc469951062)

[3.1 Структура лексического анализатора 17](#_Toc469951063)

[3.2 Контроль входных символов 17](#_Toc469951064)

[3.3 Удаление избыточных символов 17](#_Toc469951065)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 18](#_Toc469879827)

[3.5 Основные структуры данных 19](#_Toc469951067)

[3.6 Принцип обработки ошибок 19](#_Toc469951068)

[3.7 Структура и перечень сообщений лексического анализатора 19](#_Toc469951069)

[3.8 Параметры лексического анализатора и режимы его работы 19](#_Toc469951070)

[3.9 Алгоритм лексического анализа 19](#_Toc469951071)

[3.9 Контрольный пример 20](#_Toc469951072)

[Глава 4. Разработка синтаксического анализатора 21](#_Toc469951073)

[4.1 Структура Синтаксического анализатора 21](#_Toc469951074)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc469951075)

[4.4 Построение конечного магазинного автомата 22](#_Toc469951076)

[4.5 Основные структуры данных 23](#_Toc469951077)

[4.5 Описание алгоритма синтаксического разбора 23](#_Toc469951078)

[4.6 Структура и перечень сообщений синтаксического анализатора 23](#_Toc469951079)

[4.7 Параметры синтаксического анализатора и режимы его работы 23](#_Toc469951080)

[4.8 Принцип обработки ошибок 23](#_Toc469951081)

[4.9 Контрольный пример 24](#_Toc469951082)

[Глава 5. Разработка семантического анализатора 25](#_Toc469951083)

[5.1 Структура семантического анализатора 25](#_Toc469951084)

[5.2 Функции семантического анализа 25](#_Toc469951085)

[5.3 Структура и перечень сообщений семантического анализатора 25](#_Toc469951086)

[5.4 Принцип обработки ошибок 26](#_Toc469951087)

[5.5 Контрольный пример 26](#_Toc469951088)

[Глава 6. Преобразование выражений 27](#_Toc469951089)

[6.1 Выражения, допускаемые языком 27](#_Toc469951090)

[6.2 Польская запись и принцип ее построения 27](#_Toc469951091)

[6.3 Программная реализация обработки выражений 28](#_Toc469879852)

[6.4 Контрольный пример 28](#_Toc469951092)

[Глава 7. Генерация кода 29](#_Toc469951093)

[7.1 Структура генератора кода 30](#_Toc469879855)

[7.2 Представление типов данных в памяти 30](#_Toc469879856)

[7.3 Статическая библиотека 31](#_Toc469879857)

[7.4 Особенности алгоритма генерации кода 31](#_Toc469879858)

[7.5 Контрольный пример 31](#_Toc469879859)

[Глава 8. Тестирование транслятора 31](#_Toc469951094)

[8.1 Тестирование проверки на допустимость символов 32](#_Toc469879861)

[8.2 Тестирование лексического анализатора 32](#_Toc469879862)

[8.3 Тестирование синтаксического анализатора 33](#_Toc469879863)

[8.4 Тестирование семантического анализатора 33](#_Toc469879864)

[Заключение 33](#_Toc469951097)

[Приложение А 34](#_Toc469951098)

[Приложение Б 35](#_Toc469951099)

[Приложение В 36](#_Toc469951100)

[Приложение Г 38](#_Toc469951101)

[Приложение Д 40](#_Toc469951102)

[Приложение Е 42](#_Toc469951103)

[Приложение Ж 43](#_Toc469951104)

[Приложение И 51](#_Toc469951105)

[Приложение К 52](#_Toc469951106)

[Литература 53](#_Toc469951107)

# Введение

Основной целью данного курсового проекта является разработка компилятора для языка программирования ААА-2018. Главная задача компилятора заключается в том, чтобы сделать программу, написанную языке программирования ААА-2018, понятной компьютеру. Этого можно добиться одним из двух способов: компиляцией или интерпретацией. В данном курсовом проекте трансляция будет осуществляться в байткод.

Компиляция состоит из двух частей: анализа и синтеза. Анализ – это разбиение исходной программы на составные части и создание ее промежуточного представления, а синтез, в свою очередь, – конструирование требуемой целевой программы из промежуточного представления

Исходя из цели курсового проекта, были определены следующие задачи:

-разбработка спецификации языка программирования;

-разбратка структуры транслятора;

-разработка лексического анализатора;

-разработка синтаксического анализатора;

-разработка семантического анализатора;

-обработка выражений;

-генерация кода;

-тестирование компилятора.

Язык прогарммирования ААА-2018 предназначен для выполнения простейших арифметический действий и операций над строками.

**Глава 1. Спецификация языка программирования AAA-2018**

* 1. **Характеристика языка программирования**

Язык программирования ААА– это процедурный, универсальный, строго типизированный (Преобразование типов не допускается), компилируемый язык. Не является объектно-ориентированным.

* 1. **Алфавит языка**

Алфавит языка ААА-2018 основан на кодировке Windows-1251, представленной на рисунке 1.1.

Исходный код ААА-2018 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах. символы пробела, табуляции и перевода строки.

**1.3 Применяемые сепараторы**

Символы-разделители, разрешенные к использованию в языке AAA-2018 , приведены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  :  ‘ ‘ (space)  ,  =  +  -  \*  <  >  ! | разделение инструкций |
| {  } | программный блок |
| (  ) | параметры/приоритетность операций (в выражениях) |
| ;  :  ‘ ‘ (space)  ,  =  +  -  \* | разделение инструкций |
| Таблица 1.1-продолжение | |
| Сепаратор | Назначение |
| {  } | разделение инструкций |
| (  ) | параметры/приоритетность операций (в выражениях) |
| [  ] | Блок условий в конструкциях ветвления и цикла |

* 1. **Применяемые кодировки**

Для написания исходного кода на языке программирования ААА-2018 используется кодировка Windows-1251. 

**1.5 Типы данных**

В языке AAA-2018 есть 2 типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| integer | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта).  Автоматически инициализируется нулевым значением.  Возможные операции:  *арифметические*  + – бинарный, суммирование;  - – бинарный, вычитание;  \* – бинарный, умножение;  = – присваивание значения; |
| string | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). Максимальное количество символов в строке – 255. Автоматическая инициализация строкой нулевой длины. |

**1.6 Преобразование типов данных**

Преобразование типов данных не допускается, так как язык AAA-2018 является строго типизированным.

* 1. **Идентификаторы**

В имени идентификатора допускаются символы латинского алфавита нижнего регистра. Максимальная длина имени - 15 символов.

Идентификаторы не должны совпадать с ключевыми словами. Данные правила справедливы как для переменных, так и для функций.

* 1. **Литералы**

Литералы, используемые для написания программы на языке AAA-2018 описаны в таблице 1.3. Литералы осуществляют инициализацию переменных

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные неотрицательные литералы, инициализируются 0. Литералы только rvalue. |
| Строковые литералы | Символы, заключённые в ‘ ’ (одинарные кавычки), инициализируются пустой строкой, строковые переменные. Только rvalue. |

* 1. **Объявление данных и область видимости**

Для объявления переменной следует указать имя переменной и её тип.

Пример объявления переменной числового типа:

var a:integer;

Пример объявления переменной строкового типа:

var a:string;

Для объявления функций используется ключевое слово function, после которого следует указать имя функции, а после –тип функции.

Объявление внешней функции :

use function s (a:integer):integer;

Все переменные имеют область видимости, а именно префикс – название функции, в которой они находятся, (кроме main) что разрешает использование в различных функция переменных с одинаковым именем. Параметры функции видны только внутри функции. Переменные, объявленные в одной функции, недоступны в другой. – как раз допустимы

* 1. **Инициализация данных**

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| var <идентификатор>:<тип данных>; | Автоматическая инициализация: переменные типа integer инициализируются нулём, переменные типа string – пустой строкой. |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## Инструкции языка

Все возможные инструкции языка программирования AAA-2018 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования AAA-2018

|  |  |
| --- | --- |
| Инструкция | Запись на языке AAA-2018 |
| Объявление переменной | var <идентификатор> : <тип данных>; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Объявление внешней функции | use function<идентификатор> (<идентификатор>: <тип данных>,…):<типданных>; |
| Блок инструкций | main  {  …  } |
| Возврат из подпрограммы | return <идентификатор> / <литерал>; |
| Вывод данных | out <идентификатор> / <литерал>; |

* 1. **Операции языка**

Язык программирования ААА-2018 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования AAA‑2018

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 0 или 4 |
| , | 1 |
| \*/ | 2 |
| +  - | 3 |

Максимальным значением приоритетности является “0”, минимальным “4” соответственно.

Также языком поддерживается операция сравнения операндов. Данная операция не имеет приоритета.

* 1. **Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции, а параметр вызова функции может быть представлен в виде выражения.

* 1. **Программные конструкции языка**

Ключевые программные конструкции языка программирования AAA-2018 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка AAA-2018

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | main  {  } |
| Функция | function <идентификатор> (<идентификатор> : <тип>, …) : <тип>  {  …  return <выражение>;  } |

* 1. **Область видимости идентификаторов**

В языке AAA-2018 переменные обязаны находится внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

* 1. **Семантические проверки**

1. Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.
2. Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы функций не должны повторяться |
| 2 | Операнды в операторах ветвления и выхода из функции должны быть целочисленного типа |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 4 | Тип данных передаваемых значений в функцию стандартной библиотеки должен соответствовать заявленному. |
| 5 | Идентификатор должен быть объявлен до его использования. |
| 6 | Операнды в арифметическом выражении не могут быть разных типов |

**1.15Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в стеке.

## Стандартная библиотека и её состав

Функции стандартной библиотеки с описанием представлены в таблице 1.9. Стандартная библиотека написана на языке программирования C++.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| max | integer | integer x-число  integer y-число | Функция вычисляет максимальное значение |
| min | integer | integer x-число  integer y-число | Функция вычисляет минимальное значение |
| fact | integer | integer x-число | Функция вычисляет факториал x |
| strlen | integer | string x-число | Функция вычисляет длину строки |
| out |  | string x – строка  integer x-число | Функция выводит на консоль x |

Таблица 1.9 – Состав стандартной библиотеки

**1.17 Ввод и вывод данных**

Для вывода данных в языке AAA-2018 используется функция out(идентификатор).

* 1. **Точка входа**

Точкой входа в языке AAA-2018является функция main(). Если она присутствует выполнение продолжается, иначе фатальная ошибка и прекращение работы.

* 1. **Препроцессор**

В данном языке программирования AAA-2018 препроцессор не предусмотрен.

* 1. **Соглашения о вызовах**

В языке AAA-2018 используются соглашение о вызовах stdcall, то есть все параметры передаются в стек справа налево, память освобождает вызываемый код.

* 1. **Объектный код**

Исходный код, написанный на языке AAA-2018 , транслируется в

язык CIL (Common Intermediate Language)

* 1. **Классификация сообщений транслятора**

Транслятор, в ходе своей работы, генерирует сообщения, которые информируют пользователя о допущенных ошибка. Все сообщения транслятора разделены на интервалы, в зависимости от того на каком этапе была обнаружена ошибка. Все интервалы представлены в таблице 1.9.

Таблица 1.9 - Сообщения транслятора

|  |  |
| --- | --- |
| 1. Интервал | Описание ошибок |
| 0-19 | Системные ошибки |
| 20-49 | Ошибки лексического анализа |
| 100-109 | Ошибки параметров |
| 110-119 | Ошибки открытия и чтения файлов |
| 600-699 | Ошибки синтаксического анализа |
| 700-799 | Ошибки семантического анализа |

* 1. **Контрольный пример**

Контрольный пример демонстрирует главные особенности языка AAA-2018. Исходный код представлен в приложении .

**Глава 2. Структура транслятора**

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

Основными компонентами транслятора являются лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода CIL, приведенные на рисунке 2.1.

Лексический анализатор – принимает на вход уже первично обработанный и разбитый на отдельные компоненты исходный код на языке AAA-2018 . Формирует таблицу идентификаторов и таблицу лексем, а также занимается обнаружением ошибок, связанных с лексикой языка.

Синтаксический анализатор – принимает на вход таблицу лексем, сформированную лексическим анализатором. Перебирая каждое правило языка (допустимую конструкцию) он выявляет синтаксические ошибки, допущенные в исходном коде. Формирует дерево разбора, а также выводит трассировку (разбор) цепочек.

Семантический анализатор – состоит из нескольких функций, отвечающих за выявления тех или иных ошибок, а также некоторых проверок, выполняемых на этапе лексического анализатора. В на вход подается таблица лексем и таблица лексем и идентификаторов.

Генератор кода в байткод – принимает на вход таблицу идентификаторов и таблицу лексем. Задача этого компонента заключается в трансляции, уже пройденного все предыдущие этапы кода на языке AAA-2018 , в код в байткод



Рисунок 2.1 - Структура транслятора AAA-2018

**2.2 Перечень входных параметров транслятора**

С помощью входных параметров транслятора осуществляется контроль за выводом протоколов работы разных компонент, а также указывается файл с исходным кодом программы. Перечень входных параметров представлена в таблице 2.1.

Таблица 2.1 - Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Тип параметра |
| -in:\*.txt | Файл с исходным кодом на языке AAA-2018, имеющий расширение \*.txt – не обязательно | Имя файла |
| -LT | Параметр для вывода таблицы лексем в файл “log.txt” | Генерируется по умолчанию. |
| -IT | Параметр для вывода таблицы идентификаторов в файл “log.txt” | Генерируется по умолчанию. |
| -R | Отвечает за вывод правил разбора, а также трассировочную информацию в лог-файл. | Генерируется по умолчанию. |
| -log:\*.txt | Файл с журналом работы транслятора. | Имя файла |
| -out: | Файл с результатом трансляции | Имя файла |
| -logToConsole | Добавить вывод сообщений в консоль |  |

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

Транслятор AAA-2018 в ходе своей работы формирует файлы, в которых содержатся таблица лексем, таблица идентификаторов, служебная информация и дерево разбора. Протоколы работы описаны в таблице 2.2.

Таблица 2.2 - Протоколы транслятора AAA-2018

|  |  |
| --- | --- |
| Протокол | Содержимое |
| “log.txt” | Файл с протоколом работы транслятора языка программирования AAA-2018 . Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа. |
| “doc.il ” | Сгенерированный файл |

Протоколы работы нужны для отображения хода выполнения трансляции языка AAA-2018 , благодаря им пользователь может обнаружить некорректно введенные данные или ошибку в исходном коде программы.

# Глава 3. Разработка лексического анализатора

# 

## 3.1 Структура лексического анализатора

Входными данными для лексического анализатора является массив цепочек, предварительно сформированный на первичной обработке исходного кода программы. Выходными данными являются таблица лексем, таблица идентификаторов, а также протокол работы. Структура лексического анализатора AAA-2018 представлена на рисунке 3.1.

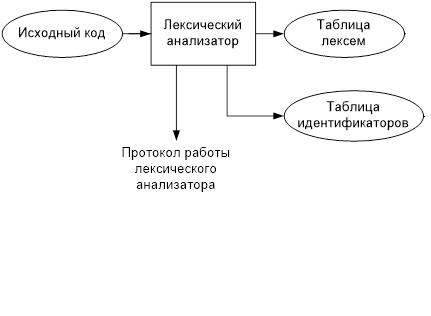


Рисунок 3.1 - Структура лексического анализатора AAA-2018

## 

## 3.2 Контроль входных символов

Таблица для контроля входных символов представлена в приложении .

В таблице: F - запрещённый символ, T - разрешённый символ, I - игнорируемый символ.

## 3.3 Удаление избыточных символов

Описание алгоритма удаления избыточных символов, а также разбиения исходного кода на цепочки:  
1. Посимвольно считываем файл с исходным кодом программы.  
2. Если встречаем кавычку, то срабатывает флаг, запрещающий удаление каких-либо символов внутри кавычек (т.к. внутри кавычек содержатся любые литералы).   
3. Встреча пробела включает проверку предыдущего (буферного) символа.   
3.1. Если предыдущий символ является пробелом, сепаратором или переходом на новую строку, то текущий символ игнорируется.   
3.2. Если предыдущий символ является допустимым символом, то текущий пробел – разделитель единиц языка, следовательно, сохраняем считанную единицу в формируемую структуру первично обработанного исходного кода. – не понятно что за логика  
4. Встреча сепаратора подключает проверку предыдущего (буферного) символа, осуществляет аналогичные пункту 3 инструкции.   
5. В конце каждой итерации посимвольного считывания текущий символ сохраняется в буфер.

## 3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам

Благодаря замене цепочек, написанных на языке AAA-2018 , лексемами -упрощается дальнейшая обработка исходного кода программы. Перечень цепочек, соответствующих им автоматов и лексем, представлена в таблице 3.1.

Таблице 3.1 – Таблица соответствий цепочек, их типов и лексем

|  |  |  |
| --- | --- | --- |
| Цепочка | Тип | Лексема |
| min | Ключевое слово |  |
| max, min | Ключевое слово |  |
| integer | Ключевое слово | t |
| string | Ключевое слово | t |
| Идентификатор |  | i |
| Целочисленный литерал |  | l |
| Строковый литерал |  | l |
| main | Ключевое слово | m |
| function | Ключевое слово | f |
| var | Ключевое слово | v |
| return | Ключевое слово | r |
| fact, strlen | Ключевое слово | c |
| +,-,\*,/ | Операторы |  |
| = | Оператор | = |
| ., {}() | Сепараторы | ., {}() |
| use |  | u |
| out | Ключевое слово | o |
|  | | |

## 3.5 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему, полученную при разборе, номер строки в исходном коде, является ли лексема арифметической операцией и номер в таблице идентификаторов, если лексема является идентификатором. Таблица идентификаторов содержит имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и значение. Описание основных структур данных, используемых для хранения таблиц лексем и идентификаторов.

## 3.6 Принцип обработки ошибок

В трансляторе AAA-2018 предусмотрена генерация ошибок разной степени, а именно: критических и предупреждений. При обнаружении предупреждения, работа транслятора не останавливается, предупреждения записываются в специальную структуру, которая наполняется по ходу работы транслятора, но при обнаружении критической ошибки, работа транслятора останавливается. – сейчас не так – в случае ошибки работа прекращается

## 3.7 Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор использует таблицу с сообщениями, структура которой представлена в приложении Б. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор продолжает работу с исходным кодом, проверяет его до конца и выводит оставшиеся сообщения об ошибках, если они существуют.

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 - Перечень сообщений лексического анализатора

Скопировать из программы Error.cpp

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 106 | Отсутствие главной функции |
| 107 | Попытка переопределения |
| 108 | Отсутствует предварительное определение |
| 117 | Превышен размер таблицы лексем |

## 3.8 Параметры лексического анализатора и режимы его работы

Транслятор AAA-2018 допускает использование параметров для управления работой лексического анализатора, а именно: выводом таблицы лексем и таблицы идентификаторов. Описание параметров представлено в таблице 3.3.

Таблица 3.3 Входные параметры лексического анализатора

|  |  |
| --- | --- |
| Параметр | Описание |
| -LT | Параметр для вывода таблицы лексем в файл “log.txt” |
| -IT | Параметр для вывода таблицы идентификаторов в файл “log.txt” |

НЕТ ТАКОГО

## 3.9 Алгоритм лексического анализа

Лексический анализ – первая и наиболее простая фаза трансляции. Алгоритм работы лексического анализатора заключается распознавании и разборе цепочек исходного кода. Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно представить в виде графа. Пример графа представлен на рисунке 3.2. На рисунке осуществляется разбор цепочки “return”, где S0 – начальное состояние, а S3 – конечное.

return

S0

t

u

r

n

r

e

S6

S4

S2

S1

S5

S3

Рисунок 3.2 – Граф переходов для цепочки “return”

## 3.9 Контрольный пример

Результатом работы лексического анализатора являются таблица идентификаторов и таблица лексем. Содержимое таблиц на основе исходного кода.

Контрольный пример в виде таблиц лексем и идентификаторов представлен в приложении

# 

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура Синтаксического анализатора

Синтаксический анализатор – следующая ступень транслятора. На свой вход он принимает таблицу лексем, которую в ходе свой работы он перебирает с целью обнаружений в ней неверных синтаксических конструкций. Начинает свою работу только при условии отсутствия ошибок в их общей структуре. Если в ходе его работы не было обнаружено ошибок, то формируется дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика для синтаксического разбора языка AAA-2018 представляется четверкой G = <T, N, P, S>, где Т – множество терминальных символов, N – множество нетерминальных символов, P – множество правил языка, S – начальный символ грамматики, являющийся нетерминалом.

В грамматике языка AAA-2018 множество нетерминальных символов представлено в таблице 4.1.

Табл. 4.1 Таблица правил переходов нетерминальных символов

|  |  |
| --- | --- |
| S→ | Порождает стартовые правила, описывающее общую конструкцию программы (функции) |
| N→ | Порождает правила, описывающие инструкции языка |
| E→ | Порождает правила, описывающие выражения |
| F→ | Порождает правила, описывающие формальные параметры функции |
| W→ | Порождает правила, описывающие принимаемые параметры функции |
| M→ | Порождает правила, описывающие арифметические действия |

## Правил больше – все в Greibach.cpp

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку,где Q – множество состояний автомата, V – алфавит входных символов, Z – алфавит специальных магазинных символов,  - функция переходов автомата,  - начальное состояние автомата,  - начальное состояние магазинного автомата, F – множество конечных состояний.

|  |  |  |
| --- | --- | --- |
| 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 4.2. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики |
|  | Начальное состояние магазина автомата | Представляет из себя 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 |

Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка AAA-2018 .Данные структуры представлены в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора:

1.Поиск и выделение синтаксических конструкций в исходном тексте (разбор).

2.Распознавание (проверка правильности) синтаксических конструкций.

3.Выявление ошибок и продолжение процесса распознавания после обработки ошибок.

4.Если нет ошибок, формирование дерева разбора.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Табл. 4.3 - Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 600 | Неверная структура программы. |
| 601 | Ошибка в параметрах функции. |
| 602 | Ошибочный оператор. |
| 603 | Ошибка в выражении. |
| 604 | Ошибка в параметрах вызываемой функции. |
| 605 | Ошибка в построении выражения. |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Транслятор AAA-2018 допускает использование параметров для управления работой синтаксического анализатора, а именно: вывода дерева разбора и трассировки. Описание параметров представлено в таблице 2.1.

## 4.8 Принцип обработки ошибок

Синтаксический анализатор перебирает всевозможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. Если не была найдена ни одна подходящая цепочка, то формируется соответствующая ошибка из таблицы 4.3. Все ошибки записываются в общую структуру ошибок.

## 4.9 Контрольный пример

Результатом работы синтаксического анализатора, на примере программы представленной в приложении , является дерево разбора, а также графическое представление дерева на графической работе 1. Дерево разбора исходного кода представлено в приложении .

# Глава 5. Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход таблицы лексем и идентификаторов. Он включает в себя несколько обособленных функций, осуществляющих проверки семантического характера, а также проверки на этапе лексического анализатора, которые не были вынесенные в отдельные функции. 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 анализа

Основные проверки семантического анализатора описаны в пункте 1.12.

Сообщения семантического анализа ошибки, приведены в таблице 5.1. Каждой ошибке соответствует функция, которая проверяет наличие данной ошибки в исходном коде языка. Функции вызываются параллельно с работой лексического анализа, а также после работы лексического анализа.

## 5.3 Структура и перечень сообщений семантического анализатора

Принцип обработки ошибок семантическим анализатором заключается в следующем: все обнаруженные ошибки записываются в общую структуру и продолжается дальнейшая работа транслятора. РАБОТА ПРЕКРАЩАЕТСЯ В СЛУЧАЕ ОШИБКИ Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 - Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| Ошибка 113 | Обнаружена вторая главная функция |
| Ошибка 114 | Несоответствие типов |
| Ошибка 115 | Запрещено присваивать значение функции |
| Ошибка 116 | Неверное значение |
| Ошибка 118 | Пропущен символ окончание литерала |
| Ошибка 119 | Запрещено возвращать функцию |
| Ошибка 120 | Неверное количество принимаемых параметров |
| Ошибка 121 | Ошибка инициализации |
| Ошибка 113 | Обнаружена вторая главная функция |
| Ошибка 114 | Несоответствие типов |
| Ошибка 115 | Запрещено присваивать значение функции |

## 5.4 Принцип обработки ошибок

Семантический анализатор выполняет проверки в соответствии с пунктом 5.2. В случае возникновении ошибки, он сохраняет ее в общую структуру ошибок и продолжает свою работу дальше.

## 5.5 Контрольный пример

Контрольный пример заключается в тестировании функций семантического анализатора при наличии соответствующих ошибок в исходном коде.

# Глава 6. Преобразование выражений

## 6.1 Выражения, допускаемые языком

Выражения, допускаемые языком AAA-2018 , выполняются над целочисленными типами данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / и (), и вызовы функций из них.

Приоритетность арифметических операций представлена в таблице 6.1.

Таблица 6.1 – Приоритетность операций

|  |  |
| --- | --- |
| Операция | Приоритет |
| ( | 4 |
| ) | 4 |
| \* | 3 |
| + | 2 |
| - | 2 |
| ( | 4 |

## 6.2 Польская запись и принцип ее построения

Польская запись — это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Ее отличие заключается в том, что знаки операций пишутся не между аргументами, а до или после них. 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 до открывающей скобки, после чего обе скобки уничтожаются.

## 6.3 Программная реализация

Программная реализация преобразования выражений к польской записи представлена в приложении .

## 6.4 Контрольный пример

Контрольный пример польской записи выражений представлен в приложении , на основе исходного кода из приложения. В качестве примера хода разбора выражения в польский формат записи приведена таблица 6.2.

Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Исходный код программы, написанный на языке AAA-2018, необходимо транслировать в байт код для последующей его компиляции.

Трансляция будет осуществляться функцией Generation, принимающей таблицу лексем и таблицу идентификаторов. Выходным будет являться файл, генерируемый во время компиляции, который будет содержать в себе транслированный исходный код.

Структура генератора кода AAA-2018 представлена на рисунке 7.1. – НА КАРТИКЕ ПРО АСМ

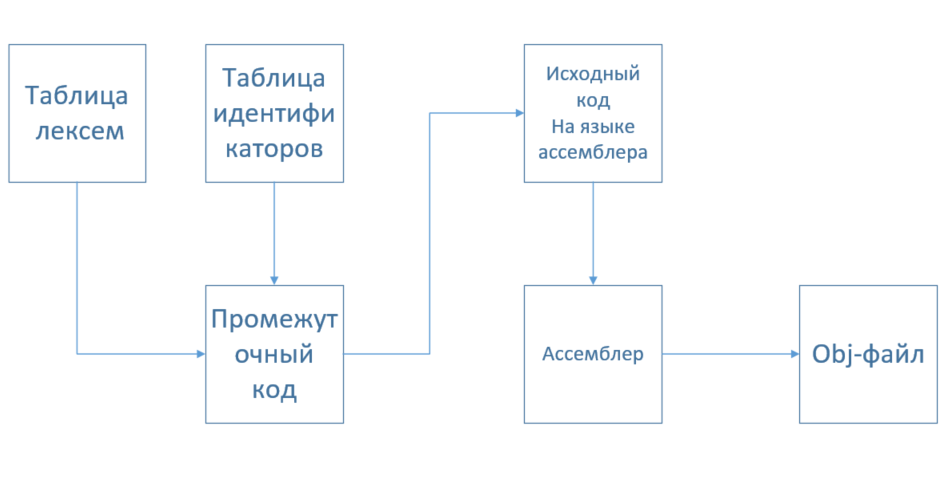


Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в памяти

Элементы таблицы идентификаторов расположены сегментах Соответствия между типами данных идентификаторов на языке AAA-2018 и в байткоде приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка AAA-2018 и байткода

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке AAA-2018 | Тип идентификатора на языке ассемблера | Пояснение |
| integer |  | Хранит целочисленный тип данных. |
| string |  | Хранит целочисленный тип данных без знака. |

## 7.3 Статическая библиотека

В языке AAA-2018 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++ (на CIL). Объявление функций статической библиотеки генерируется автоматически в байткоде. Объявление функций статической библиотеки генерируется автоматически.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| char\*\_\_out(int a) | Вывод на экран числа |
| char\*\_\_out(int a) | Вывод на экран строки |
| Int \_\_stdcall min(int a, int b) | Определяет минимальное значенние |
| int \_\_stdcall max(int a, int b) | Определяет максимальное значенние |
| int \_\_stdcall fact(int a) | Вычисление факториала. |
| Int\_\_\_stdcall strlen(char\*s) | Длинна строки |

## 7.4 Особенности алгоритма генерации кода

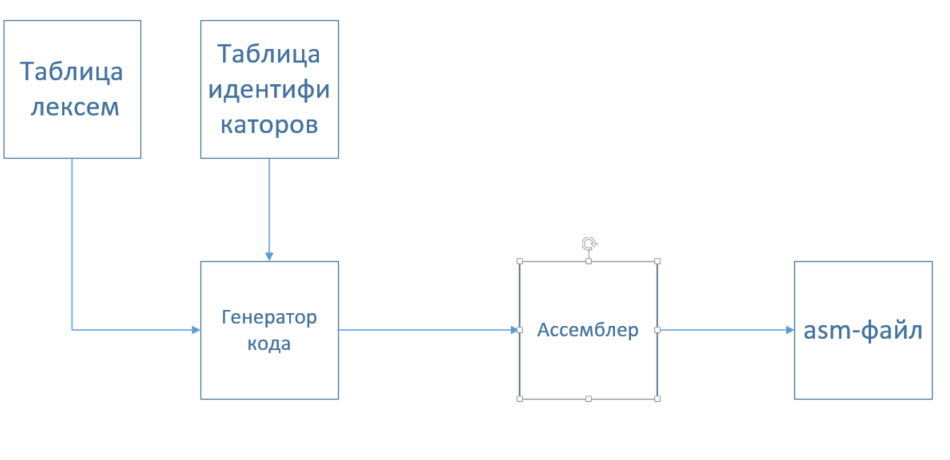
В языке AAA-2018 генерация кода строится на основе таблиц лексем и идентификаторов (И ДЕРЕВА РАЗБОРА). Общая схема работы генератора кода представлена на рисунке 7.2. – РИСУНОК ПРО АСМ

Рисунок 7.2 – Структура генератора кода

## 7.5 Контрольный пример

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