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 BAA-2019»

Выполнил студент Борисов Антон Андреевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Дмитрий Витальевич

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

Заведующий кафедрой к.т.н., доц. Пацей Наталья Владимировна

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

Консультант преп.-стаж. Котович Дмитрий Витальевич

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

Нормоконтролер преп.-стаж. Котович Дмитрий Витальевич

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

Курсовой проект защищен с оценкой

Минск 2019

# **Введение**

Целью курсового проекта является разработка компилятора для своего языка программирования: BAA-2019. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор – это программа, задачей которого является перевод программы, написанной на одном из языков программирования (в моём случае на BAA-2019) в программу на язык ассемблера.

Компиляция состоит из двух частей: анализа и синтеза. Анализ – это разбиение исходной программы на составные части и создание ее промежуточного представления. Синтез – конструирование требуемой целевой программы из промежуточного представления. В данном курсовом проекте мой исходный код транслируется на язык ассемблера. Язык ассемблера – тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

Мой компилятор состоит из следующих составных частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач буду приведены в соответствующих главах курсового проекта, а именно:

В первой главе работы определена спецификация языка программирования, т.е. описан синтаксис и семантика языка.

Во второй главе работы представлена структура транслятора, т.е. перечислены компоненты транслятора, их назначение и принципы взаимодействия, перечень входных параметров, перечень протоколов, формируемых транслятором и их содержимое.

В третьей главе работы показана разработка лексического анализатора, порождающего таблицы лексем и идентификаторов.

В четвертой главе работы рассказывается о синтаксическом анализаторе, который выполняет синтаксический разбор текста с распечаткой протокола разбора и дерева разбора на основе таблицы лексем.

В пятой главе описан семантический анализатор, показана его работа (распечатка выданных сообщений в трёх примерах на разных этапах).

В шестой главе решены вопросы преобразования выражений, допускаемых языком и приведена часть протокола для контрольного примера, отображающая результаты преобразования выражений в польский формат.

В седьмой главе представлена генерация кода, где из промежуточного представления порождается код на целевом языке.

В восьмой главе описывается тестирование транслятора.

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования BAA-2019 является процедурным, строго типизированным, не объектно-ориентированным, компилируемым.

Процедурный язык программирования — язык высокого уровня, в котором используется метод разбиения программ на отдельные связанные между собой модули — подпрограммы (процедуры и функции).

Строго типизированный язык программирования — язык, в котором переменные привязаны к конкретным типам данных. Язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования.

Объектно-ориентированный язык программирования — язык, построенный на принципах объектно-ориентированного программирования. В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Компилируемый язык программирования — язык программирования, исходный код которого преобразуется компилятором в исходный код на другом языке программирования.

* 1. **Определение алфавита языка программирования**

В алфавите языка BAA-2019 используется кодировка ASCII, таблица которой

представлена на рисунке 1.1.



Рисунок 1.1 – Алфавит входных символов

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

* 1. **Применяемые сепараторы**

Применяемые сепараторы в языке BAA-2019, приведены в таблице 1.1. Таблица 1.1 — Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Область применения |
| ‘пробел' | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |
| ; | Разделитель программных конструкций |
| { } | Блок функции |
| = | Оператор присваивания |
| ( ) | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| , | Разделитель параметров функций |
| +,-,\*,/ ,% | Арифметические операции |
| [] | Блок условной конструкции/цикла |
| > < & ! | Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции. |

* 1. **Применяемые кодировки**

Для написания программ язык BAA-2019 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как { } [ ] ( ) , ; ~ + - / \* > < & !.

## **Типы данных**

В языке BAA-2019 есть 3 типа данных: целочисленный и строковый, символьный. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка BAA-2019

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| integer | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 4 байта. Максимальное значение: 2147483647.  Минимальное значение: -2147483647.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  + (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  / (бинарный) – оператор деления;  %(бинарный) – оператор остаток от деления;  = (бинарный) – оператор присваивания.  В качестве условия цикла/условного оператора поддерживаются следующие логические операции:  > (бинарный) – оператор «больше»;  <(бинарный) – оператор «меньше»;  & (бинарный) – оператор проверки на равенство;  ! (бинарный) – оператор проверки на неравенство. |
| string | 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: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, а также использование библиотечных функций. |
| symbol | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 1.  Инициализация по умолчанию: символ нулевой длины “”. Операции над данными строкового типа: присваивание символьному идентификатору значения другого символьного идентификатора, символьного литерала или значения символьной функции, а также использование библиотечных функций. |

## **Преобразование типов данных**

В языке программирования BAA-2019 преобразование типов данных не поддерживается, т.е. язык является строго типизированным.

## **Идентификаторы**

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны начинаться только с символов латинского алфавита, могут содержать цифры. Максимальная длина идентификатора равна восьми символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов. Идентификаторы не должны совпадать с ключевыми словами.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Типы литералов языка BAA-2019 представлены в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не должен отделяться пробелом) |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в “ ” |
| Символьные литералы | Символ , заключенный в ~ ~ |

## **Объявление данных**

Для объявления переменной используется ключевое слово type, после которого указывается тип данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления числового типа с инициализацией:

**type integer** num1 = -1

Пример объявления переменной строкового типа с инициализацией:

**type string** str1= “hello world”

Пример объявления переменной символьного типа с инициализацией:

**type symbol** ch1= ~c~

Для объявления функций используется ключевое слово function, перед которым указывается тип функции, после которого имя функции. Далее обязателен список параметров и тело функции.

## **Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа **integer** и строка длины 0 (“”) для типа **string** и **symbol**.

Таблица 1.4 — Способы инициализации переменных

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

## **Инструкции языка**

Инструкции языка BAA-2019 представлена в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | **type** <тип данных> <идентификатор>. |
| Объявление переменной с явной инициализацией | **type** <тип данных> <идентификатор> = <значение>|<выражение>;  Значение – литерал, идентификатор, вызов функции соответствующего типа данных |
| Объявление функции | <тип данных> **function** <идентификатор> (<тип данных> <идентификатор>, …)  {  / программный блок /  **return** <идентификатор/литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | <идентификатор> = <значение>; |
| Вывод данных | **write** <идентификатор/литерал>; |
| Вывод данных с переходом на новую строку | **writeline** <идентификатор/литерал>; |
| Возвращаемое значение | **return** <литерал/идентификатор>. |

## **Операции языка**

В языке BAA-2019 предусмотрены следующие операции с данными. Приоритетность операции умножения выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.6.

Таблица 1.6 — Операции языка

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические операции языка | + – сложение  - – вычитание  \* – умножение  / – деление  % – остаток от деления  = – присваивание |
| Логические | > – больше  < – меньше  & – проверка на равенство  ! – проверка на неравенство |

## **Выражение и их вычисление**

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

* Допускается использовать скобки для смены приоритета операций;
* Выражение записывается в строку без переносов;
* Использование двух подряд идущих операторов не допускается. (за исключением оператор “-”, идущего после любого другого оператора и предшествующему литералу);
* 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

## **Конструкции языка**

Программа на языке BAA-2019 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 представлены в таблице 1.7.

Таблица 1.7 — Основные конструкции языка

|  |  |  |
| --- | --- | --- |
| Конструкция | | Реализация |
| Главная функция | | main  {  …  } |
| Внешняя функция | | <тип данных> function <идентификатор> (<тип> <идентификатор>, ...)  {…  return <идентификатор/литерал>;  } |
| Условная конструкция | is <идентификатор1> <оператор> <идентификатор2>  true [ … ]  false [ … ]  <идентификатор1>, <идентификатор2> - идентификаторы или литералы. <оператор> - один из операторов сравнения ( > < & ! ), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. Причем с идентификаторами и литералами строкового и символьного типа, могут использоваться только операторы сравнения (& !) При истинности условия выполняется код внутри блока true, иначе – код внутри блока false. Любой из блоков true, false может отсутствовать, но не оба блока одновременно. При отсутствии одного из блоков, в зависимости от истинности или ложности условия программа может как выполнить один из заявленных блоков, так и передать управление инструкции, следующей в коде. | |
| Цикл | while <идентификатор1> <оператор> <идентификатор2>  do [ … ]  Цикл (операторы внутри блока do) выполняется, пока истинно условие “<идентификатор1> <оператор> <идентификатор2>”, имеющее тот же смысл, что и в примере выше. | |

## **Область видимости переменных**

Область видимости: сверху вниз (как и в С++). Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри какого-либо блока. Каждая переменная или параметр функции получают префикс – название функции, внутри 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.

## **Семантические проверки**

* В языке программирования BAA-2019 выполняются следующие семантические проверки:
* Наличие функции main – точки входа в программу;
* Единственность точки входа;
* Переопределение идентификаторов;
* Использование идентификаторов без их объявления;
* Проверка соответствия типа функции и возвращаемого параметра;
* Правильность передаваемых в функцию параметров: количество, типы;
* Правильность строковых выражений;
* Превышение размера строковых и числовых литералов;
* Правильность составленного условия цикла/условного оператора.

## **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.

## **Стандартная библиотека и её состав**

В языке BAA-2019 предусмотрена стандартная библиотека. Функции, входящие в состав библиотеки, описаны в табл. 1.8. Стандартная библиотека подключается автоматически на этапе генерации кода.

Таблица 1.8 - Функции стандартной библиотеки языка BAA-2019

|  |  |
| --- | --- |
| Функция | Описание |
| random(integer a) | Данная функция целочисленного типа принимает 1 параметр. Функция генерирует случайное число в диапазоне от -a до а и возвращает это число. |
| pow (integer a, integer b) | Данная функция целочисленного типа принимает два целочисленных параметра. Функция возводит число a в степень b и возвращает результат. |
| power (integer a, integer b) | Данная функция целочисленного типа принимает два целочисленных параметра. Функция возводит число a в степень b и возвращает результат. |
| length (string a) | Данная функция целочисленного типа принимает 1 параметр.  Функция возвращает длину строки a. |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор **write** и **writeline**. Эти функции представлены в таблице 1.9.

Таблица 1.9 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void outnum(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void outstr(char\* line) | Функция для вывода в стандартный поток значения строкового, символьного идентификатора/литерала. |
| void outstrline(char\* line) | Функция для вывода в стандартный поток значения строкового, символьного идентификатора/литерала и перехода на новую строку. |
| void outnumline(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала и перехода на новую строку. |
| void system\_pause() | Функция которая запускается по окончанию программы и требует нажать любой символ для завершения программы |

## **Ввод и вывод данных**

Вывод данных осуществляется с помощью операторов **write** и **writeline**. Допускается использование операторов **write** и **writeline** с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательские команды **write** и **writeline** в транслированном коде будут заменены вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

## **Точка** **входа**

В языке BAA-2019 каждая программа должна содержать главную функцию (точку входа) **main**, с первой инструкции которой начнётся последовательное выполнение команд программы.

## **Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке BAA-2019 отсутствует.

## **Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

BAA-2019 транслируется в язык ассемблера, а затем - в объектный код

## **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 транслятора приведены в таблице 1.10, а также в приложении А.

Таблица 1.10 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 200 | Системные ошибки |
| 200 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |
| 400-499, 700-999 | Зарезервированные коды ошибок |

## **Контрольный пример**

Контрольный пример демонстрирует главные особенности языка BAA-2019: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

# **Глава 2. Структура транслятора**

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

В языке BAA-2019 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 BAA-2019 приведена на рисунке 1.



Рисунок 2.1 Структура транслятора языка программирования BAA-2019

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 BAA-2019

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке BAA-2019, имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<путь к out-файлу> | Выходной файл – результат работы транслятора. Содержит исходный код на языке асемблера. | Значение по умолчанию:  <имя in-файла>.asm |

## **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 BAA-2019

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования BAA-2019 . 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, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

# **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**

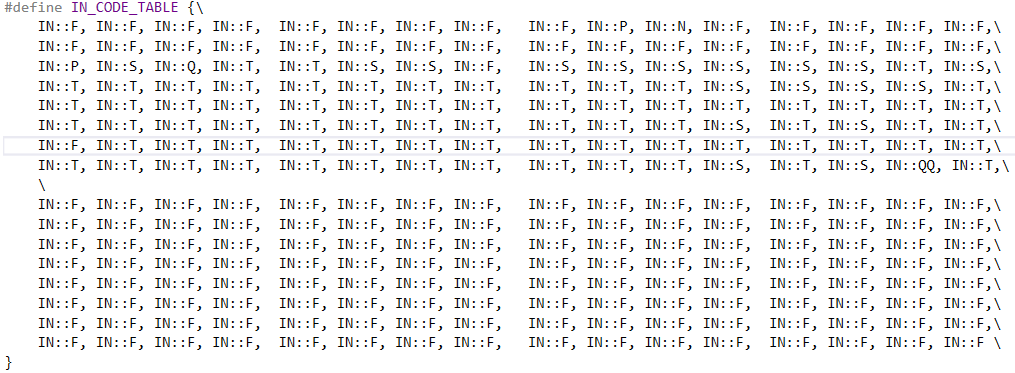
Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2, категории входных символов представлены в таблице 3.1.

Рисунок 3.2. Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Строковый литерал | Q |
| Символьный литерал | QQ |
| Сепаратор | S |
| Перевод строки | N |
| Пробел, табуляция | P |

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы;

2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора;

3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

## **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 3.2.

Таблица 3.2 Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| integer, string,symbol | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 8 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| Функции стандартной библиотеки | p | Ключевое слово для стандартных функций. |
| return | e | Выход из функции/процедуры. |
| main | m | Главная функция. |
| type | n | Объявление переменной. |
| write | o | Вывод данных. |
| writeline | ^ | Вывод данных c переводом строки. |
| is | ? | Указывает начало условного оператора. |
| while | c | Указывает начало цикла. |
| true | r | Истинная ветвь условного оператора. |
| false | w | Ложная ветвь условного оператора. |
| do | d | Указывает на начало тела цикла. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока цикла/условного выражения. |
| ] | ] | Конец блока цикла/условного выражения. |
| { | { | Начало тела функции. |
| } | } | Конец тела функции |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +  -  \*  /  % | +  -  \*  /  % | Знаки операций. |
| >  <  &  ! | >  <  &  ! | Знаки логических операторов |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.

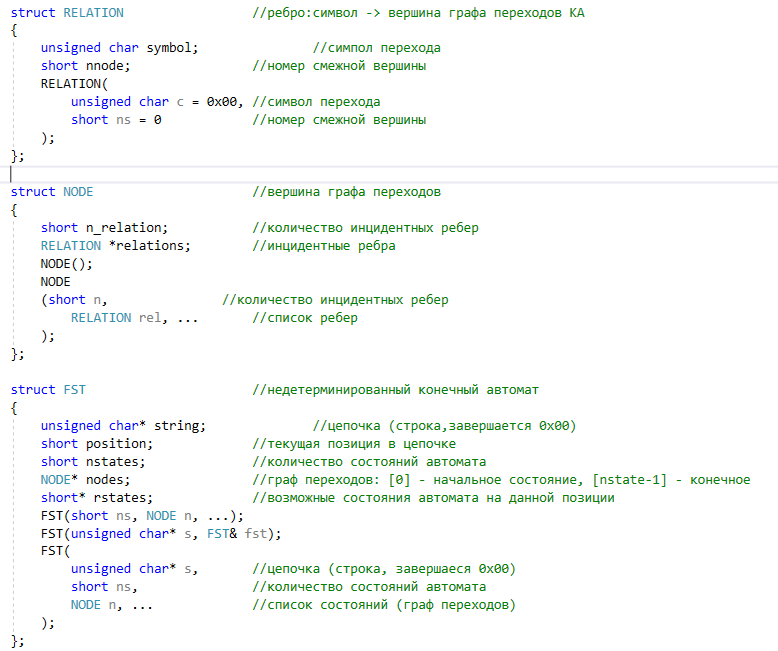


Рисунок 3.3 Структура конечного автомата

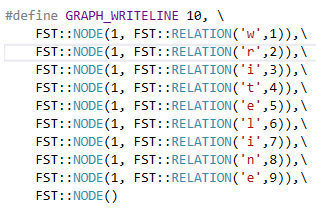


Рисунок 3.4 Пример реализации графа конечного автомата для токена writeline

## **3.5 Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы лексем представлен на рисунке 3.3. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

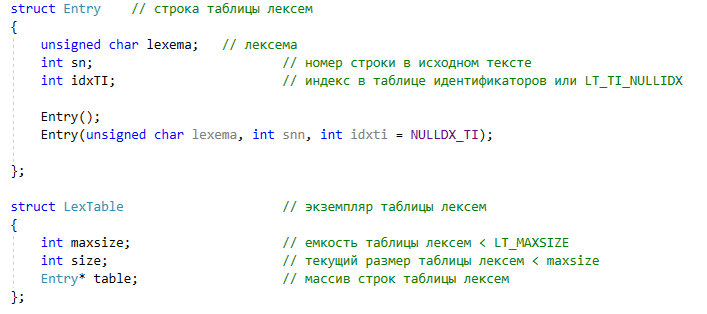


Рисунок 3.3 Структура таблицы лексем

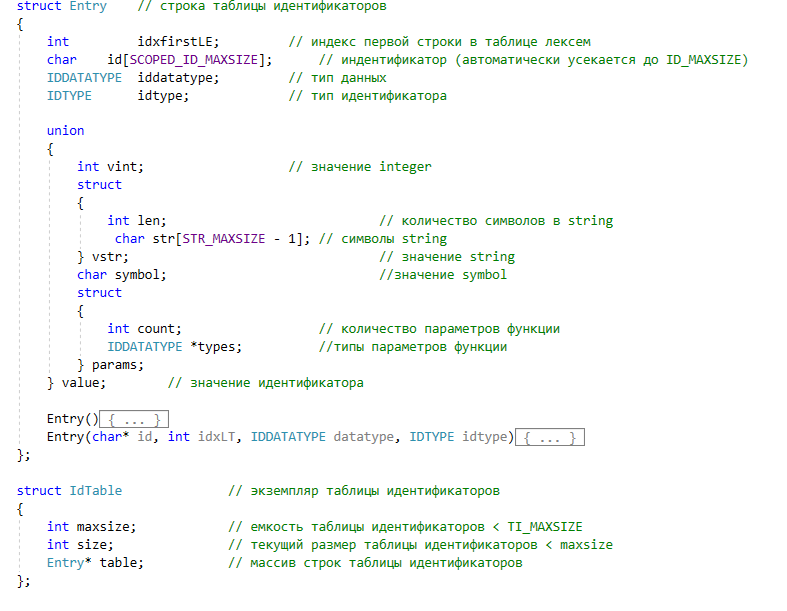


Рисунок 3.4 Структура таблицы идентификаторов

## **3.6 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. 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 игнорирует найденную ошибку и продолжает работу с исходным кодом. Перечень сообщений представлен на рисунке 3.5.



Рисунок 3.5 - Сообщения лексического анализатора

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

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.

## **3.8 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**

* проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
* для выделенной части входного потока выполняется функция распознавания лексемы;
* при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
* формирует протокол работы;
* при неуспешном распознавании выдается сообщение об ошибке.
* Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «string» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «string» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

s t r i n g

Рисунок 3.2 Пример графа переходов для цепочки string

## **3.10 Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении Б.

# **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 языка**

В синтаксическом анализаторе транслятора языка BAA-2019 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->m{K} | Стартовые правила, описывающее общую структуру программы |
| P | P->(E)  P->() | Правила для параметров объявляемых функций |
| T | T->{Q;}  T->[KQ;] | Правила для тела функций |
| Q | Q->eV;  Q->e(l);  Q->e(-i); | Правила для возврата значений |
| 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->rYwY  R->wYrY | Правила составления условного оператора |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла/условного оператора |
| L | L-><  L->>  L->&  L->! | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->% | Правила для арифметических операторов |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| V | V->l  V->i | Правила для простых выражений |
| Y | Y->[XQ]  Y->[X]  Y->[Q] | Правила для тела условного выражения |
| W | W->l  W->i  W->(W)  W->(-i)  W->(-i)AW  W->(W)AW  W->iF  W->pF  W->iAW  W->lAW  W->iFAW  W->pFAW | Правила для сложных выражений |
| K | K->nti=V;K  K->nti=WK  K->nti;K  K->i=W;K  K->oV;K  K->^VZ;K  K->?ZdHK  K->cZdHK  K->i=UF;K  K->nti=V;  K->nti=W;  K->i=W;  K->nti;  K->oV;  K->^V;  K->?ZR  K->cZdH  K->i=UF; | Программные конструкции |
| X | X->i=W;X  X->oV;X  X->^V;X  X->i=UF;X  X->i=W;  X->oV;  X->^V;  X->i=UF; | Программные конструкции внутри цикла/условного оператора |
| U | U->i  U->p | Правила для идентификатора функции |
| B | B->?ZR  B->?ZRX  B->?ZRXB  B->?ZRB | Правила для конструировании условного выражения в цикле |
| H | H->[XB]  H->[B]  H->[X] | Правила для тела цикла |

## **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 ВAA-2019 . 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**

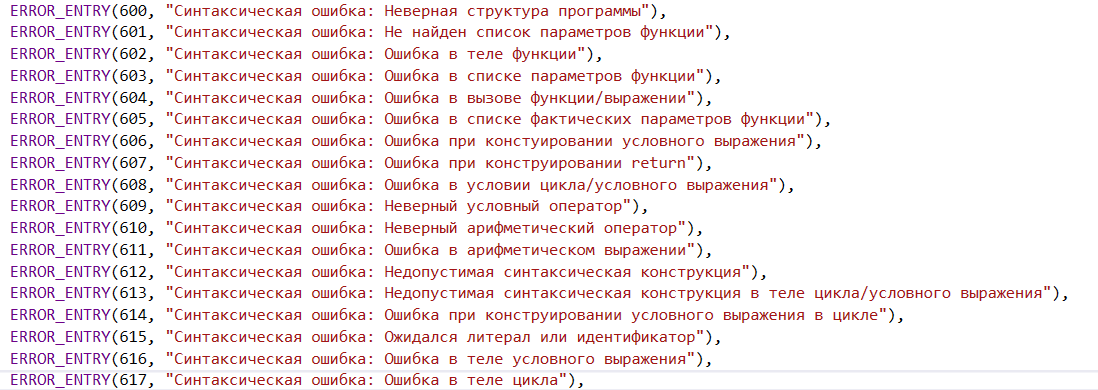
 Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

Рисунок 4.3 - Сообщения синтаксического анализатора

## **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.

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

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

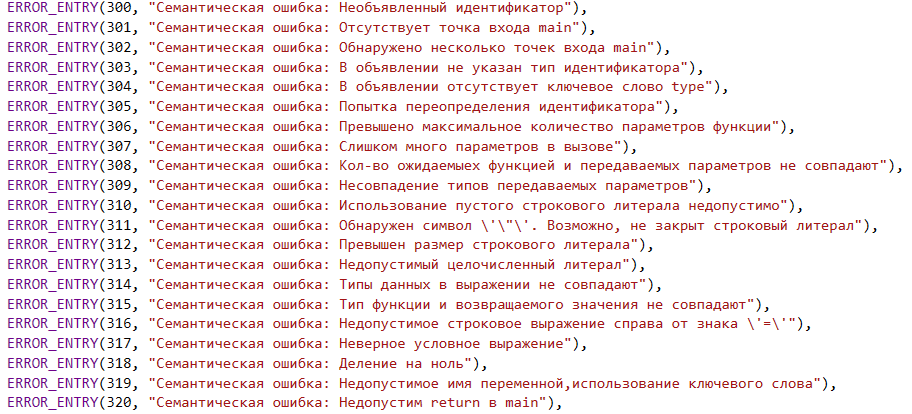


Рисунок 5.2 – Перечень сообщений семантического анализатора

## **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{  integer x = 9;  write x;  } | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово type Строка: 2 |
| main{  type integer x = 9;  type integer y =x;  } | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3 |
| main{  type integer x = 9;  }  main{  type string y = "qwerty";  } | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main |