Оглавление

[ВВЕДЕНИЕ 3](#_Toc469519600)

[Глава 1.СПЕЦИФИКАЦИЯ АЛГОРИТМИЧЕСКОГО ЯЗЫКА BNI-2016 4](#_Toc469519601)

[1.1. Характеристика языка программирования 4](#_Toc469519602)

[1.2. Алфавит языка 4](#_Toc469519603)

[1.3. Символы – сепараторы 5](#_Toc469519604)

[1.4. Применяемые кодировки 6](#_Toc469519605)

[1.5. Типы данных 6](#_Toc469519606)

[1.6. Преобразование типов данных 6](#_Toc469519607)

[1.7. Идентификаторы 7](#_Toc469519608)

[1.8. Литералы 7](#_Toc469519609)

[1.9. Область видимости идентификаторов 7](#_Toc469519610)

[1.10. Инициализация данных 7](#_Toc469519611)

[1.11. Инструкции языка 8](#_Toc469519612)

[1.12. Операции языка 8](#_Toc469519613)

[1.13. Выражения и их вычисления 9](#_Toc469519614)

[1.14. Программные конструкции языка 9](#_Toc469519615)

[1.16. Семантические проверки 9](#_Toc469519616)

[1.17. Распределение оперативной памяти на этапе выполнения 9](#_Toc469519617)

[1.18. Стандартная библиотека и её состав 9](#_Toc469519618)

[1.19. Вывод данных 10](#_Toc469519619)

[1.20. Точка входа 10](#_Toc469519620)

[1.21. Препроцессор 10](#_Toc469519621)

[1.22. Соглашения о вызовах 10](#_Toc469519622)

[1.23. Объектный код 10](#_Toc469519623)

[1.24. Классификация сообщений транслятора 10](#_Toc469519624)

[1.25. Контрольный пример 10](#_Toc469519625)

[Глава 2. Структура транслятора 11](#_Toc469519626)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 11](#_Toc469519627)

[2.2 Перечень входных параметров транслятора 12](#_Toc469519628)

[2.3 Перечень протоколов формируемых транслятором и их содержимое 12](#_Toc469519629)

[Глава 3. Разработка лексического анализатора 12](#_Toc469519630)

[3.1 Структура лексического анализатора 12](#_Toc469519631)

[3.2 Контроль входных символов 13](#_Toc469519632)

[3.3. Удаление избыточных символов 14](#_Toc469519633)

[3.4. Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем, регулярных выражений и конечных автоматов 14](#_Toc469519634)

[3.5. Основные структуры данных 15](#_Toc469519635)

[3.6. Принцип обработки ошибок 16](#_Toc469519636)

[3.7. Структура и перечень сообщений лексического анализатора 16](#_Toc469519637)

[3.8 Параметры лексического анализатора и режимы его работы 16](#_Toc469519638)

[3.9 Алгоритм лексического анализа 16](#_Toc469519639)

[3.10 Контрольный пример 16](#_Toc469519640)

[Глава 4. Разработка синтаксического анализатора 18](#_Toc469519641)

[4.1 Структура синтаксического анализатора 18](#_Toc469519642)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 18](#_Toc469519643)

[**4.3** **Структура и перечень сообщений синтаксического анализатора** 19](#_Toc469519644)

[**4.4** **Принцип обработки ошибок** 19](#_Toc469519645)

[**4.5** **Структура синтаксического анализатора** 20](#_Toc469519646)

[**4.6** **Параметры управления синтаксическим анализатором и выходные данные** 22](#_Toc469519647)

[**4.7** **Принцип работы магазинного автомата** 22](#_Toc469519648)

[**4.8** **Контрольный пример** 23](#_Toc469519649)

[Глава 5. Разработка семантического анализатора 25](#_Toc469519650)

[5.1 Структура семантического анализатора 25](#_Toc469519651)

[5.2 Функции семантического анализатора 25](#_Toc469519652)

[5.3 Структура и перечень сообщений синтаксического анализатора 26](#_Toc469519653)

[5.4 Принцип обработки ошибок 26](#_Toc469519654)

[Глава 6. Вычисление выражений 27](#_Toc469519655)

[6.1 Общие положения 27](#_Toc469519656)

[6.2 Алгоритм преобразования выражения 27](#_Toc469519657)

[Глава 7. Генерация кода 28](#_Toc469519658)

[7.1 Общие положения 28](#_Toc469519659)

[7.2 Представление типов данных в памяти 28](#_Toc469519660)

[7.3 Статическая библиотека 28](#_Toc469519661)

# ВВЕДЕНИЕ

Основной целью данной курсовой работы является разработка транслятора для языка программирования BNI-2016. BNI -2016 не содержит технически сложно реализуемых компонентов и конструкций, поскольку основное внимание при разработке уделяется компонентам, реализующим важнейшие функции любого компилятора, таких как лексический, синтаксический и семантический анализаторы, а также генератор ассемблерного кода.

Следовательно, основные задачи данной курсовой задачи можно представить в виде:

1. Разработка лексического анализатора;
2. Разработка синтаксического анализатора;
3. Разработка семантического анализатора;
4. Разработка генератора ассемблерного кода.
5. Тестирование транслятора;

Разработка данной программной системы подразумевает проход по каждому из этапов классического цикла разработки программного обеспечения, откуда следует ещё одна цель: развить, усовершенствовать и закрепить навыки проектирования, реализации и тестирования программных продуктов.

# Глава 1.СПЕЦИФИКАЦИЯ АЛГОРИТМИЧЕСКОГО ЯЗЫКА BNI-2016

# 1.1. Характеристика языка программирования

Язык BNI-2016 ­– это процедурный, универсальный, строготипизированный, компилируемый язык. Hе является объектно-ориентированным.

# 1.2. Алфавит языка

Таблица символов, используемых для написания исходного кода программы представлена на рисунке 1. Алфавит языка BNI-2016 основан на кодировке Windows-1251.



Рис. 1. Таблица символов

Символы, используемые на этапе выполнения:

[a…z], [A…Z], [а…я],[А…Я], [0…9], спецсимволы:’**{**’,’**}**’ ,’**(**‘ ,’**)**’,’**;**’,’**,**’, ‘**\***’, ‘**+**’, ‘-’, ‘**/**’, ‘**=**’, ‘ **\’**’;

Русские символы разрешено использовать только в литералах.

# 1.3. Символы – сепараторы

В языке BNI-2016 предусмотрены следующие символы-сепараторы.

Символы-сепараторы служат в качестве разделителей цепочек (единиц) языка на уровне первичной обработки исходного текста программы.

Табл. 1. Символы-сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ; | разделитель инструкций |
| {} | программный блок |
| ( ) | приоритетность операций, параметры |
| , | разделитель параметров функции |
| Пробел | Разделитель цепочек |

# 1.4. Применяемые кодировки

Применяется кодировка Windows-1251.

# 1.5. Типы данных

В BNI-2016 предусмотрены 2 типа данных:

Табл. 2. Типы данных

|  |  |
| --- | --- |
| Целочисленный тип данных int | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных (4 байта).  Автоматическая инициализация нулём.  Возможные операторы:  *арифметические*  **+** – бинарный, суммирование **(int**, **int);**  - – бинарный, вычитание **(int, int);**  **\*** – бинарный, умножение **(int, int);**  = – присваивание значения **(int, int);**  **/** – бинарный, деление **(int, int);** |
| Строковый тип данных str | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт).Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255.  Возможные операторы:  = – копирование значения строки str1в str2 **(str2, str1);** |

# 1.6. Преобразование типов данных

Преобразование типов данных не поддерживается.

# 1.7. Идентификаторы

Идентификаторы состоят из символов латинского алфавита ([a…z]). Длина не должна превышать 7 символов(иначе обрезается до 7 символов). Зарезервированные идентификаторы не предусмотрены.

# 1.8. Литералы

С помощью литералов осуществляется инициализация переменных каким-либо постоянным значением. В языке существует 2 вида литералов: литералы целого типа и строковые\символьные литералы. Могут быть только **rvalue**.

Табл. 3. Литералы

|  |  |
| --- | --- |
| Литералы целого типа | Не имеют дробных частей или экспонент. Целочисленные литералы можно задавать в десятичной форме. Числа интерпретируются как **int.** |
| Строковые литералы | Символы, заключённые в ‘’(одинарные кавычки) инициализируются как **str,** строковые переменные**.** |

Описание литералов языка BNI-2016 с помощью регулярных выражений можно представить в следующем виде:

1. Литералы целого типа: **[1-9]+[0-9]\***

2. Литералы строкового типа: **[a-z|A-Z|А-Я|а-я|0-9| !-/]+**

# 1.9. Область видимости идентификаторов

Область видимости сверху вниз (по принципу С++). Все переменные являются глобальными. Имя каждой функции состоит из Идентификатора функции + порядковый номер функции. Имя переменной содержит Префикс объемлющей функции + имя идентификатора + порядковый номер функции

# 1.10. Инициализация данных

Определение переменной:

var <тип данных> <идентификатор>;

Инициализация значением:

<идентификатор> = <значение>;

Автоматическая инициализация переменных типа **int** нулём. Автоматическая инициализация переменных типа **str** строкой нулевой длины.

# 1.11. Инструкции языка

Табл. 4. Инструкции языка

|  |  |
| --- | --- |
| Определение переменной | var <тип данных> <идентификатор> |
| Инициализация внешней функции | <тип данных> func<идентификатор>(<тип данных> <идентификатор>,... <тип данных> <идентификатор>)  Параметры:  передаются по значению |
| Инициализация идентификатора | <идентификатор> = <значение>; |
| Блок инструкций | main()  {  return <идентификатор> или <литерал>;  } |
| Возврат из подпрограммы | return <идентификатор> или <литерал>; |
| Вывод данных | display <идентификатор> или <литерал>; |

# 1.12. Операции языка

Операции языка BNI-2016 определяют стандартные возможности для манипулирования данными того или иного типа. Перечень операции представлен в таблице 5.

Табл. 5. Операции языка

|  |  |
| --- | --- |
| Арифметические | **()** – приоритет операций  **\*** - умножение  **/** - деление  **+** - сложение  **-** - вычитание  = - присваивание (работает и для типа str) |

# 1.13. Выражения и их вычисления

Вычисление выражений является одной из важнейших задач большинства языков программирования. Всякое выражение подчиняется правилам его составления, каждое из которых описано ниже

Табл. 6. Выражения

|  |
| --- |
| Выражения записываются в строку без переносов |
| Допуск круглых скобок для изменения приоритета |
| Не допускается запись двух подряд идущих операций |
| Выражения преобразовываются в формат польской записи |
| В выражении возможен вызов функции |

На этапе генерации кода каждое выражение преобразуется в формат польской записи.

# 1.14. Программные конструкции языка

Табл. 7. Конструкции языка

|  |  |
| --- | --- |
| **Главная функция** | main()  {  return <integer-идентификатор> или <integer-литерал> или true или false;  } |
| **Функция** | Объявление внешней функции  <тип данных> func <идентификатор> (<тип данных> <идентификатор>, ...)  {  return <идентификатор>;  } |

# 1.15. Область видимости

Все переменные в языке BNI-2016 являются глобальными, содержащими номер префикс+идентификатор+объемлющей функции.

# 1.16. Семантические проверки

Таблица семантических правил представлена в пункте 2 главы 5.

# 1.17. Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в куче.

# 1.18. Стандартная библиотека и её состав

Табл. 9. Стандартная библиотека

|  |  |
| --- | --- |
| **int** strl (**str**) | возврат длины строки |
| **int** ipow(**int, int**) | возведение в степень |

# 1.19. Вывод данных

Вывод на экран в системе программирования BNI-2016 осуществляется с помощью встроенных операторов

display <литерал/идентификатор>;

Запись

display;

переносит каретку на новую строку.

# 1.20. Точка входа

Программа должна содержать одну точку вход – **main** является точкой входа. Две точки входа не допускаются.

# 1.21. Препроцессор

Препроцессор, принимающий некоторые данные на свой вход и выдающий другие данные на вход транслятору, в системе BNI-2016 отсутствует.

# 1.22. Соглашения о вызовах

Используется соглашение \_stdcall, то есть все параметры передаются в стек справа налево при вызове функции; память освобождает вызываемый код.

# 1.23. Объектный код

BNI-2016 транслируется в язык ассемблера, а далее - в объектный код.

# 1.24. Классификация сообщений транслятора

|  |  |
| --- | --- |
| Кодысообщений | Принадлежность |
| 0 – 99 | Системные ошибки |
| 100 – 119 | Ошибки входных параметров |
| 110 – 119 | Ошибки открытия и чтения файлов |
| 120 – 139 | Ошибки лексического анализа |
| 140-599 | Зарезервированные коды ошибок |
| 600 – 699 | Ошибки синтаксического анализа |
| 700 – 799 | Ошибки семантического анализа |
| 800– 999 | Зарезервированные коды ошибок |

# 1.25. Контрольный пример

Контрольный пример демонстрирует главные особенности языка BNI-2016. Исходный код представлен в приложении А.

# Глава 2. Структура транслятора

# 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Лексический анализатор - принимает на вход исходный код программы на языке MAP-2016, предварительно разбитый на отдельные компоненты. Основной задачей лексического анализатора является формирование таблицы лексем и таблицы идентификаторов, необходимых для дальнейших этапов работы транслятора, а также обнаружение определённых ошибок. Результат работы лексического анализа фиксируется в файле-протоколе.



Рис. 2 Структура транслятора языка программирования BNI -2016

Лексический анализатор – часть транслятора, выполняющая лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов. На вход лексического анализатора приходит файл с исходным кодом. По окончании данного этапа получаем таблицу лексем и таблицу идентификаторов.

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. Проверяется правильность расположения идентификаторов и ключевых слов в исходном коде. Для того, чтобы провести данную операцию используются таблица лексем и идентификаторов. На выходе получаем дерево разбора.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов. Код семантического анализатора разбросан по синтаксическим и лексическим анализам.

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

# 2.2 Перечень входных параметров транслятора

Входные параметры принимаются в командной строке.

Табл. 10. Входные параметры

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:\*.txt | Файл с исходным кодом на языке BNI -2016, имеющий расширение \*.txt | Не предусмотрено, должен быть задан обязательно |
| -lt | Параметр для вывода таблицы лексем в файл log | Таблица лексем не выводится |
| -it | Параметр для вывода таблицы идентификаторов на консоль | Таблица идентификаторов не выводится |
| -tr | Параметр для вывода трассировки на консоль | Трассировка не выводится |
| -st | Параметр для вывода дерева разбора в файл log | Дерево разбора не выводится |

# 2.3 Перечень протоколов формируемых транслятором и их содержимое

Входные параметры определяют имена файлов для отчетов работы транслятора. Файл log.txt включает в себя таблицу лексем, полученную на этапе лексического анализа, таблицу идентификаторов, трассировку и дерево разбора, полученные на этапе синтаксического анализа, ошибки выполнения.

Табл. 11. Выходные файлы

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| “log.txt” | Файл с протоколом работы транслятора языка программирования BNI -2016, |
| “out.txt” | Файл с ассемблерным кодом. |

# Глава 3. Разработка лексического анализатора

# 3.1 Структура лексического анализатора

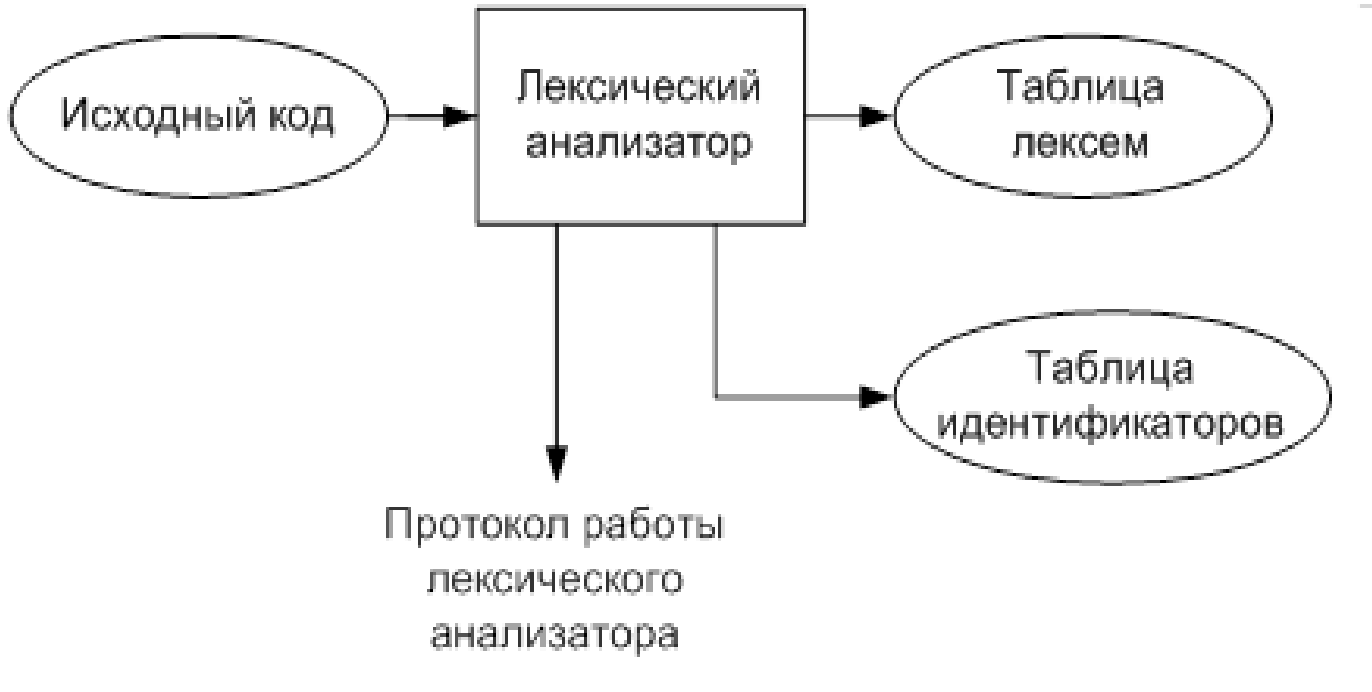


Рис 3. Структура лексического анализатора

Лексический анализатор принимает структуру из файла парсера, содержащую массив цепочек, их количество и номера строк цепочек. В процессе лексического анализа заполняются таблицы лексем и идентификаторов. На выходе получаются заполненные таблицы лексем и идентификаторов.

# 3.2 Контроль входных символов

Таблица, контролирующая входные символы, представлена на рисунке 4.

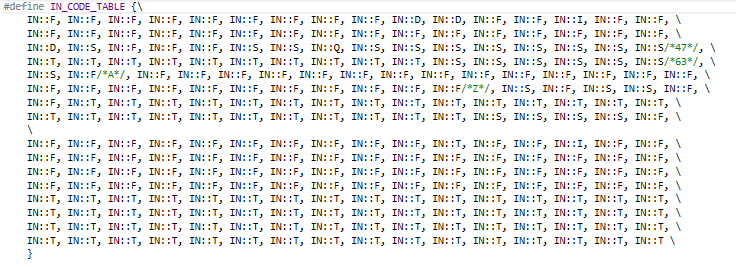
Рис 4. Таблица допустимых символов

Таблица содержит массив, содержащий типы входных символов, соответствующим символам таблицы ASCII(0-255). При проверке исходного кода идет сравнение типа входного символа и типа символа, соответствующего его индексу в этом массиве.

T – символ разрешён;

F – символ запрещён;

I – символ игнорируется обработчиком;

S – символ является сепаратором.

Q – кавычка(для строчного литерала)

D – перенос строки + пробел

Каждый считанный из файла с исходным кодом символ проверяется на допустимость, в случае, если символ помечен значением T либо S - записывается в цепочку, I –игнорируется и обрабатывается следующий символ, F – срабатывает обработчик исключений и в таблицу ошибок добавляется ошибки запрещенного символа.

# 3.3. Удаление избыточных символов

Предусмотрен разбор исходного кода в массив цепочек, в алгоритме которого избыточные символы игнорируются.

# 3.4. Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем, регулярных выражений и конечных автоматов

Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем представлен в таблице 12

Табл. 12. Цепочки и лексемы

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | лексема |
| Ключевые слова(определяют основные операторы/типы данных и т.д.) | var | v |
| str | t |
| int | t |
| func | f |
| return | r |
| main | m |
| display | d |
| Сепараторы(разделяют ключевые слова) | , | , |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| ; | ; |
| Символы операций(определяют символы арифметических операций) | + | v |
| - | v |
| \* | v |
| / | v |

Пример конечного автомата для цепочки int.Структура конченого автомата, содержит узлы(NODE), и инцидентные им ребра.

FST fst\_int("", LEX\_INT, FST\_INT, 4,

NODE(1, RELATION('i', 1)),

NODE(1, RELATION('n', 2)),

NODE(1, RELATION('t', 3)),

NODE()

);

# 3.5. Основные структуры данных

Структуры данных для лексического анализа представлены в приложении Б.

# 3.6. Принцип обработки ошибок

В случае обнаружения ошибки, она заносится в массив ошибок. Максимально допустимое количество ошибок в лексическом анализе ограничено числом 10. По достижении данного количества ошибок транслятор прекращает свою работу и выводит на экран все ошибки, найденные в исходном коде.

# 3.7. Структура и перечень сообщений лексического анализатора

Номера ошибок лексического анализатора находятся в промежутке 120-139 в общем массиве сообщений. Каждое сообщение сопровождается дополнительной информацией в виде номер строки ошибки и, быть может, ошибочной цепочки. Перечень сообщений представлен на рисунке 5.

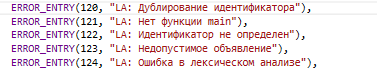


Рис 5. Таблица ошибок для лексического анализа.

# 3.8 Параметры лексического анализатора и режимы его работы

Алгоритм лексического анализа не имеет входных параметров

# 3.9 Алгоритм лексического анализа

Лексический анализ - один из простейших этапов трансляции, освобождающий синтаксический анализ от рутинных алгоритмов. Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «main» представлен на рисунке 6, где S0 – начальное, а Sf – конечное состояние автомата.

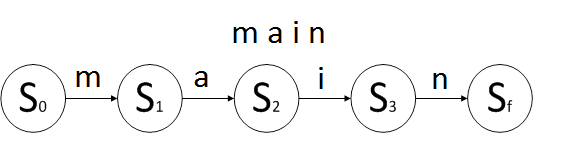


Рис 6. Экземпляр таблицы идентификатора

# 3.10 Контрольный пример

Разбор на таблицы идентификаторов и лексем контрольного примера(Приложение B) представлен ниже.

# Глава 4. Разработка синтаксического анализатора

# 4.1 Структура синтаксического анализатора

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы. Наиболее удобным формализмом для описания синтаксических конструкций языка программирования являются контекстно-свободные грамматики( например нормальная форма Бэкуса-Наура).

Синтаксический анализ – это процесс, который определяет, принадлежит ли некоторая последовательность лексем языку, порождаемому грамматикой.

Вход синтаксического анализатора – последовательность лексических и таблицы, например, таблица внешних представлений, которые являются выходом лексического анализатора.



Рис. 8. Схема работы синтаксического анализатора

Выход синтаксического анализатора – дерево разбора и таблицы, например, таблица идентификаторов и таблица типов, которые являются входом для следующего просмотра компилятора (например, это может быть просмотр, осуществляющий контроль типов).

# 4.2 Контекстно свободная грамматика, описывающая синтаксис языка

Синтаксис языка BNI-2016 описывается при помощи контекстно свободной грамматики.

Контекстно-свободная грамматика – грамматика типа 2 по иерархии Хомского. Данная грамматика имеет вид , где

T – множество терминальных символов

Используемые лексемы представлены в таблице 12

N – множество нетерминальных символов

Множество нетерминальных символов можно увидеть в структуре грамматики Грейбах в приложении Г

P – множество правил переходов

Множество правил переходов можно увидеть в структуре грамматики Грейбах в приложении Г

S – стартовый символ

В контекстно-свободной грамматике правила имеют вид: , где





 - словарь грамматики 

??переделать

Грамматика представлена в приложении В в виде заполненной структуры на языке С++.

# 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку,. Подробное описание компонентов магазинного автомата представлено в таблице 13.

Таблица 13. 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 |

**4.4** **Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора содержится в таблице 13.

Таблица 13. Сообщения синтаксического анализатора

|  |  |
| --- | --- |
| **Номер сообщения** | **Содержание сообщения** |
| Ошибка 600 | Неверная структура программы |
| Ошибка 601 | Параметры функции заданы неверно |
| Ошибка 602 | Структура цикла или условия составлено неверно |
| Ошибка 603 | Конструкция функции составлено неверно |
| Ошибка 604 | Выражение составлено неверно |

**4.5** **Принцип обработки ошибок**

Синтаксический анализатор перебирает все возможные правила и цепочки правила грамматики в целях поиска подходящего соответствия. Если ни одна из цепочек правила не подошла для рассматриваемой конструкции, то генерируется ошибка в соответствии с таблицей 4.2. Ошибка записывается в общую структуру ошибок.

* 1. **Структура синтаксического анализатора**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка BNI-2016. Эти структуры представлены в приложении Г

* 1. **Параметры управления синтаксическим анализатором и выходные данные**

Параметры, управляющие синтаксическим анализатором представлены ранее в таблице 10.

Выходными данными являются протоколы работы синтаксического анализатора, которые описаны в таблице 11.

* 1. **Принцип работы синтаксического анализатора**

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 в стеке, записывается в магазин в обратном порядке;
5. если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку, правила, которому соответствует данные нетерминал;
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 вызывается ошибка.

Структура магазинного автомата, записанная на языке программирования С++, представлена в приложении ??.

* 1. **Контрольный пример**

Результатом работы синтаксического анализатора является дерево разбора, содержимое которого представлено в приложении Г.

# Глава 5. Разработка семантического анализатора

# 5.1 Структура семантического анализатора

Семантические проверки предназначены для обеспечения целостности и непротиворечивости конструкций языка. Структура семантического анализатора представлена на рисунке 14.

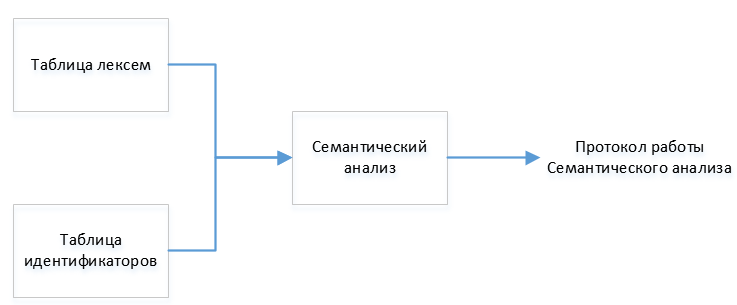


Рис. 14. Структура семантического анализатора

Часть семантического анализатора расположены в лексическом анализаторе. Основной семантический анализ расположен в файле SemAn.cpp.

Семантический анализ проводится после лексического и синтаксического анализов. При наличии хотя бы одной ошибки после семантического анализа в общем массиве ошибок генерация кода не запускается.

# 5.2 Функции семантического анализатора

Семантические проверки, размещенные в двух файлов представлены в таблице 14.

Таблица 14. Таблица семантических правил

|  |  |
| --- | --- |
| № | Правило |
| 1 | СМА: Наличие функции main |
| 2 | Усечение слишком длинных идентификаторов до 7 символов ( [a-z] ) |
| 3 | СМА: Операнды в выражении должны соответствовать одному типу |
| 4 | ЛА: Нет повторяющихся наименований функций |
| 5 | ЛА: Нет повторяющихся объявлений идентификаторов |
| 6 | ЛА: Предварительное объявление и определение применяемых функций |
| 7 | ЛА: Предварительное объявление идентификаторов. |
| 8 | СМА: Тип возвращаемого значения функции должен соответствовать типу функции |
| 9 | СМА: Соответствие типов формальных и фактических параметров при вызове функций |
| 10 | ЛА: Усечение слишком длинного значения str –литерала до 255 символов |
| 11 | ЛА: Округление слишком большого значения int –литерала до INT\_MAX |
| 12 | Сначала идет проверка на ключевые слова, а потом на идентификаторы |

Максимальное число ошибок – 5 штук. После чего будет произведена остановка трансляции с выводом всех найденных ошибок.

Семантические проверки, расположенные в лексическом анализаторе и семантическом обозначены ЛА и СМА соответственно.

# 5.3 Структура и перечень сообщений синтаксического анализатора

Структуру сообщений семантического анализатора представляет собой общий массив ошибок, предназначенный на 5 ошибок.

Перечень сообщений семантического анализатора представлен на рисунке 15.

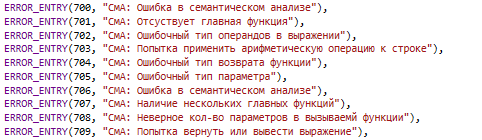


Рис. 15. Структура сообщений семантического анализатора

# 5.4 Принцип обработки ошибок

Если ошибка возникает на этапе лексического анализа, синтаксический анализ не выполняется.

При возникновении ошибки в процессе лексического анализа, сообщение ошибочной фразы заносится в общий массив ошибок и осуществляется попытка разбора следующей фразы.

При возникновении ошибки в процессе синтаксического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы, вплоть до 3 ошибок.

При накоплении 5 ошибок транслятор завершает свою работу с кодом ошибки 2 и сообщением "SYSTEM: Недопустимое количество ошибок".

# 5.5 Контрольный пример

Контрольный пример основывается на тестировании функций семантического анализатора при наличии соответствующих ошибок в исходном коде. Результат работы контрольного примера представлен в приложении Д.

# Глава 6. Вычисление выражений

# 6.1 Общие положения

В языке BNI-2016 предусмотрено вычисление выражений. В выражениях может использоваться вызов функции как один из операндов.

Для трансляции в ассемблерный код выражения формируются в формат обратной польской записи.

# 6.2 Алгоритм преобразования выражения

На этапе генерации кода идет поиск правил, содержащих выражение. Далее номер позиции в таблице лексем данного выражения отправляется в функцию преобразования к обратной польской записи. По заданному алгоритму выражения формируется в вид обратной польской записи, удобной для вычисления выражения через стек. На выходе получаем массив элементов таблицы лексем, содержащих нужные лексемы.

Функция, генерирующая ассемблерный код из формата польской записи представлена в приложении E.

# Глава 7. Генерация кода

# 7.1 Общие положения

Генерация кода происходит в язык ассемблера. Входными данными генерации являются таблицы лексем и идентификаторов. В процессе генерации использует дерева разбора для вставки шаблонов кода на каждое правило, полученного на этапе синтаксического анализа.

# 7.2 Представление типов данных в памяти

Элементы таблицы идентификаторов расположены в сегментах языка ассемблера.

Переменные(идентификаторы) языка BNI-2016 размещены в сегменте данных:

Типу данных INT соответствует тип данных языка ассемблера SDWORD, где хранится целочисленное значение со знаком.

Типу данных STR соответствует тип данных языка ассемблера DWORD, где хранится смещение адреса строки.

Литералы языка BNI-2016 располагаются в сегменте констант. Им соответствуют следующие типы данных:

Литералу типа INT соответствует тип данных языка ассемблера SDWORD, где хранится целочисленное значение со знаком.

Литералу типа INT Типу данных STR соответствует тип данных языка ассемблера BYTE, где хранится значение литерала.

Генерируется сегмент стека размером 4096 байта. Удобство стека заключается в том, что его область используется многократно, причем сохранение в стеке данных и выборка их оттуда выполняется с помощью эффективных команд push и pop без указания каких-либо имен.

Стек традиционно используется, например, для сохранения содержимого регистров, используемых программой, перед вызовом подпрограммы, которая, в свою очередь, будет использовать регистры процессора "в своих личных целях".

# 7.3 Статическая библиотека

Имеется статическая библиотека, содержащая как стандартные функции языка BNI-2016, так и функции, использующиеся во время выполнения кода.

Статическая библиотека содержит функции, написанные на языке C++. Она включает в себя стандартные функции BNI-2016:

Таблица 15. Функции статической библиотеки

|  |  |
| --- | --- |
| Прототип функции | Описание |
| int strl(char\* s) | возвращает длину строки |
| int ipow(int a,int b) | возвращает значение a^b |
| void writei(int i) | Выводит значение i на консоль |
| void writes(const char\* s) | Выводит значение s на консоль |

Параметры в функцию передаются через стек. Применяется соглашение о вызовах stdcall. Параметры передаются в стек справа налево. Стек очищается вызывающей функцией. Возврат значения из функции осуществляется через регистр edx.

# 7.4 Особенности алгоритма генерации кода

Алгоритм конвертирует [синтаксически](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) корректную программу на одном языке, в нашем случае BNI-2016, в последовательность инструкций, которые могут выполняться на машине. Алгоритм генерации кода основан на использовании дерева разбора и таблицы лексем и идентификаторов, полученных на предыдущих этапах работы транслятора. Общая схема работы генератора кода представлена на рисунке 16.

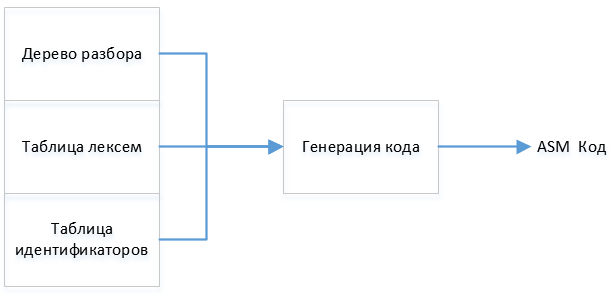


Рис. 16. Общая схема работы генератора кода

В начале кодогенерации формируются заголовочные строки кода ASM, а именно:

1.Установка используемой архитектуры (.586)

2. Модель памяти и соглашение о вызовах функций

3. Подключаемые библиотеки

4. Прототипы используемых функций

Далее выделяется память для стека в размере 4096 байта.

Функции CreateDataSeg и CreateConstSeg создают области данных и констант кода ASM соответственно. Каждая из этих функций, используя таблицу идентификаторов, составляет переменные и константы соответствующих типов.

Далее организуется сегмент ASM кода .code, содержащий весь основной код. Алгоритм использует дерево разбора. На правила из дерева разбора подготовлен специализированный шаблон. Шаблоны для кодогенерации представлены в приложении Ж. Шаблоны могут содержать в себе нетерминал.

Общий алгоритм работы генератора кода:

1. В исходную строку записывается начальный символ ‘S’.

2.Очередная итерация по дереву разбора получает правило. Предполагается, что нетерминал для данного правила содержится в исходной строке генерируемого кода.

3.Далее идет поиск нетерминала левой части правила в исходной строке.

4. Удаляется символ данного нетерминала в исходной строке.

5. На место удаленного нетерминала устанавливается шаблон для данного правила.

Пример работы алгоритма представлен в таблице 16.

Таблица 16. Алгоритм работы генератора кода

|  |  |
| --- | --- |
| Правило | Исходная строка |
| NULL | ;S |
| S->tfi(F){NrE;};S | retpow0 PROC ;F  ;N  ;E  pop edx  ret  retpow0 ENDP |
| F->ti,F | retpow0 PROC retpow0x:SDWORD, ;F  ;N  ;E  pop edx |
| F->ti | retpow0 PROC retpow0x:SDWORD, retpow0y:SDWORD  ;N  ;E  pop edx  ret  retpow0 ENDP |
| N->dE; | retpow0 PROC retpow0x:SDWORD, retpow0y:SDWORD  ;E  call writei  ;E  pop edx  ret  retpow0 ENDP |
| E->i | retpow0 PROC retpow0x:SDWORD, retpow0y:SDWORD  push main0x  call writei  ;E  pop edx  ret  retpow0 ENDP |
| E->i | retpow0 PROC retpow0x:SDWORD, retpow0y:SDWORD  push main0x  call writei  push 0h  pop edx  ret  retpow0 ENDP |

Примечание: алгоритм использует не все правила дерева разбора. Например, правило объявления переменной N->dti; не используется в алгоритме, т.к. переменные были объявлены ранее.

# 7.5 Генерация выражений

Обработкой выражения в алгоритме генерации кода занимается специальная функция. Ее вызов осуществляется только тогда когда в дереве разбора было найдено правило N->i=E; либо N->i=E;N, т.е. правила определяющие выражения. В языке BNI-2016 выражения могут быть только rvalue после оператора присваивания. Внутри выражений возможен вызов функций.

Функция генерации выражения возвращает строку с кодом ASM. Данная строка формируется на основе выражения, полученного в результате преобразования его к виду польской записи.

Применение функции преобразования выражения представлено в таблице 17.

Таблица 17. Преобразования выражения

|  |  |  |
| --- | --- | --- |
| Исходное выражение | Польская запись | Генерация ASM кода |
| x=8/(fi(1,z)+3) | x=8z1fi3+/ | push 8  push z  push 1  call fi  push edx  push 3  pop eax  pop ebx  add eax,ebx  jo EXIT\_overflow  push eax  pop ebx  pop eax  test ebx, ebx  jz EXIT\_div\_on\_NULL  cdq  idiv ebx  push eax  pop x |

Типы переменных по обе стороны от оператора присваивания (‘=’) должны быть одного типа данных.

После каждой арифметической операции осуществляются проверки на возможные ошибки.

Для операций + - \* осуществляется проверка на переполнение регистра.

jo EXIT\_overflow

Переход на EXIT\_overflow вызывает завершение ASM программы с кодом ошибки -2 и соответствующим диагностическим сообщением.

Для операции / осуществляется проверка на деление на ноль.

test ebx, ebx

jz EXIT\_div\_on\_NULL

Переход на EXIT\_div\_on\_NULL вызывает завершение ASM программы с кодом ошибки -1 и соответствующим диагностическим сообщением.

# 7.6 Вывод на консоль

Операторы display в языке BNI-2016 позволяют выводить литералы/идентификаторы на консоль, при том оператор display; переводит каретку на новую строку.

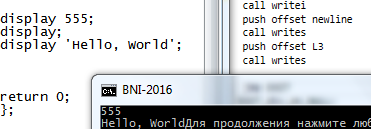


Рис. 17. Вывод на консоль

Файл статической библиотеки содержит функции вывода на консоль для числовых значений и для строковых. В соответствии с типом идентификатора/литерала вызывается соответствующая функция вывода значения на консоль. Пример вывода представлен на рисунке 17.