МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-49 01 01«Программное обеспечение информационных технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка транслятора BNI-2016

Студент Борсук Николай Игоревич

(Ф.И.О.)

Руководитель проекта (работы): с.п. Наркевич А.С.

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

Заведующий кафедрой: к.т.н., доцент каф. ИСИТ В. В. Смелов

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

Консультанты: с.п. Наркевич А.С.

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

Нормоконтролеры: с.п. Наркевич А.С.

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

Курсовой проект защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Минск 2016

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИТ Кафедра ИСИТ

Специальность 1-49 01 01

Специализация Программирование интернет-приложений

«Утверждаю»

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_

(подпись) (инициалы и фамилия)

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_\_ г.

**ЗАДАНИЕ**

**НА КУРСОВОЙ ПРОЕКТ СТУДЕНТА**

Студенту Борсуку Николаю Игоревичу

(фамилия, имя, отчество)

**1. Тема проекта**: Разработка транслятора BNI-2016

утверждена приказом по университету от «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. №\_

**2. Срок сдачи студентом законченного проекта**: 16 декабря 2016 г.

**3. Исходные данные к проекту**:

Разработка программы осуществляется на языке C++ (стандартизации International Standard ISO/IEC 14882:2014(E) Programming Language C++ 14) в среде разработки Visual Studio 2015 update 2. Операционная система под которой происходит разработка Windows 7 SP1 (64-bit). Типы данных: int, str, bool; имеются литералы для каждого типа. Предусмотрены программные конструкций типа функций. Функции стандартной библиотеки: 3 функции для работы со строками, 1 для чисел. Операторы: арифметические(с выставлением приоритетов), строковые, логические. Оператор вывода в стандартный поток: out.

**4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)**

4.1 Введение

4.2 Спецификация алгоритмического языка BNI-2016

4.3 Разработка лексического анализатора

4.4 Разработка синтаксического анализатора

4.5 Разработка семантического анализатора

4.6 Генерация кода

4.7 Контрольный пример

4.8 Руководство программиста

4.9 Приложение

4.10 Литература

**5. Перечень графического материала (с точным указанием обязательных чертежей):**

Дерево разбора синтаксического анализатора

**6. Консультанты по проекту с указанием относящихся к ним разделов проекта**

|  |  |
| --- | --- |
| Раздел | Консультант |
| Оформление пояснительной записки к курсовому проект. | Наркевич А. С. |

7. Календарный план

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Спецификация алгоритмического языка BNI-2016 | 15.10.2016 |  |
| 2 | Разработка лексического анализатора | 05.11.2016 |  |
| 3 | Разработка синтаксического анализатора | 11.11.2016 |  |
| 4 | Разработка семантического анализатора | 16.11.2016 |  |
| 5 | Генерация кода | 25.11.2016 |  |
| 6 | Тестирование | 05.11.2016 |  |
| 7 | Оформление руководства программиста | 10.11.2016 |  |
| 8 | Оформление курсового проекта | 13.12.2016 |  |

**8.Дата выдачи задания** 16.09.2016

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)

**\**

**1.СПЕЦИФИКАЦИЯ АЛГОРИТМИЧЕСКОГО ЯЗЫКА BNI-2016**

**1.1. Характеристика языка программирования**

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

**1.2. Алфавит языка**

Алфавит языка BNI-2016 основан на кодировке Windows-1251.

Таблица символов, используемых для написания кода программы:



**Рис. 1**

Символы, используемые на этапе выполнения:

[a…z], [A…Z], [а…я],[А…Я], [0…9], спецсимволы:’**{**’,’**}**’ ,’**(**‘ ,’**)**’,’**;**’,’**,**’, ‘**\***’, ‘**+**’, ‘-’, ‘**/**’, ‘**>**’, ‘**&**’, ‘**~**’, ‘**!**’, ‘**=**’, ‘ **”** ’, ‘**<**’, ‘ ’,‘\**t**’;

**1.3. Символы – сепараторы(таблица)**

В языке предусмотрены следующие символы-сепараторы:

Табл. 1. Символы-сепараторы

|  |  |
| --- | --- |
| **;** | разделитель инструкций |
| **{}** | программный блок |
| **( )** | параметры |
| **( )** | приоритетность операций |
| // | однострочный комментарий |
| “” | допускается везде кроме идентификаторов и ключевых слов |

**1.4. Применяемые кодировки**

Применяется кодировка Windows-1251.

**1.5. Типы данных**

Используем 3 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);**  **++ -** унарный, постинкремент **(int);**  **-- -** унарный, постдекремент **(int)**  *сравнения*  **~ -** бинарный, равенство (**int, int)** > - бинарный, больше (**int, int)**  < - бинарный, меньше **(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 одной строки в другую **(str, str);**  **+** – бинарный, конкатенация **(str, str).**  **~ -** бинарный, сравнение строк(**str,str**) |
| Логический тип данных bool | Фундаментальный тип данных. Предусмотрен для логических переменных. Автоматическая инициализация **false(0).**  Занимаемый размер 1 байт. Инициализирована только ключевыми словами **true(1)** или **false(0).**  Возможные операторы:  *логические*  & – бинарный, логическое умножение **(bool, bool);**  **|** – бинарный, логическое суммирование **(bool, bool).**  сравнения  **~ -** бинарный, равенство (**bool, bool)** |

**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 10 символов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы свыше 10 символов обрезаются до 10.

**1.8. Литералы**

В языке существует 3 вида литералов: литералы целого типа и строковые\символьные литералы, литералы логического типа. Могут быть только **rvalue**.

Табл. 3. Литералы

|  |  |
| --- | --- |
| Литералы целого типа | Не имеют дробных частей или экспонент. Целочисленные литералы можно задавать в десятичной форме. Могут иметь знак. числа инициализируются как **int.(**переписать) |
| Строковые литералы | Символы, заключённые в “”(двойные кавычки) инициализируются как **str,** строковые переменные**.** |
| Логические литералы | Ключевые слова **true** и **false**. |

**1.9. Область видимости идентификаторов**

Область видимости сверху вниз (по принципу С++). Все переменные являются глобальными.

**1.10. Инициализация данных**

var <тип данных> <идентификатор>;

<идентификатор> = <значение>;

Автоматическая инициализация переменных типа **int** нулём. Автоматическая инициализация переменных типа **str** строкой нулевой длины. Автоматическая инициализация переменных типа **bool** ключевым словом **false.**

**1.11. Инструкции языка**

Табл. 4. Инструкции языка

|  |  |
| --- | --- |
| Инициализация переменной | var <тип данных> <идентификатор> |
| Инициализация внешней функции | var <тип данных> func<идентификатор>(<тип данных> <идентификатор>,... (<тип данных> <идентификатор>)  Параметры:  передаются по значению  Область видимости сверху вниз (по принципу С++). Все переменные являются глобальными.(цэ перенести) |
| Присваивание | <идентификатор> = <значение>; |
| Блок инструкций | main()  {  …  return <идентификатор> или <литерал>;  } |
| Цикл c предусловием | while(<bool-выражение/ идентификатор/литерал><>0)  {…}; |
| Возврат из подпрограммы | return <идентификатор> или <литерал>; |
| Вывод данных | display <идентификатор> или <литерал>; |

**1.12. Операции языка**

Все операции расставлены по приоритету, начиная с наивысшего.

Табл. 5. Операции языка

|  |  |
| --- | --- |
| aрифметические | **()** – приоритетность операций(меняют приоритет)  **++** - инкремента  **--**дикремента  **\*** - умножение  **/** - деление  **+** - сложение  **-** - вычитание |
| строковые | **+** - конкатенация |
| логические | **&** - бинарное умножение  **|**  - бинарное сложение |
| условные | **~** - сравнение  **<** - меньше  **>**- больше |

**1.13. Выражения и их вычисления**

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 |

**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>;  } |

**1.16. Семантические проверки(ДОПИСАТЬ):**

Табл. 8. Семантика

|  |
| --- |
| длина идентификатора не превышает 10 букв; |
| cтрока содержит в себе максимум 255 символов. |
| обязательное наличие главной функции main(). |
| предварительное объявление идентификатора с ключевым словом var. |

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

Все переменные размещаются в стеке.

**1.18. Стандартная библиотека и её состав**

Табл. 9. Стандартная библиотека

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

**1.19. Вывод данных**

display <аргумент>;

**1.20. Точка входа**

Главная функция – **main** является точкой входа.

**1.21. Препроцессор**

Препроцессор не предусмотрен.

**1.22. Соглашения о вызовах**

*cdecl* – соглашение о вызовах для всех функций.  
Особенности *cdecl*:

все параметры функции передаются в стек;

память высвобождает вызывающий код;

занесение в стек параметров идёт справа налево.

**1.23. Объектный код**

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

**1.24. Классификация сообщений транслятора**

Вывод ошибок при их наличии в каждой фазе транслятора.

**1.25. Контрольный пример**

*int func fi(int x, int y)*

*{*

*var int z;*

*var int s;*

*x++;*

*y--;*

*z = x\*(x+y);*

*z = s+z;*

*return z;*

*};*

*int func fs (str a, int b)*

*{*

*var int c;*

*return c;*

*};*

*bool func pl(bool a, bool b)*

*{*

*var bool rc;*

*rc = a&b | a&(a|b);*

*return rc;*

*}*

*main()*

*{*

*var int x;*

*x = 5;*

*var int y;*

*y=4;*

*var int z;*

*z = fi(x,y);*

*display z;*

*var str sa;*

*sa = "Hello World";*

*display sa;*

*var str sb;*

*sb = "13.02.1998";*

*var str sc;*

*sc = sa+sb;*

*var bool ba;*

*ba = false;*

*var bool bb;*

*var bool bc;*

*bc = pl(ba,bb);*

*while(z>0)*

*{*

*display z;*

*z--;*

*}*

*return 0;*

*}*

**Глава 2. Структура транслятора**

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



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 и дерево разбора, полученное 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 представлена на рисунке8**

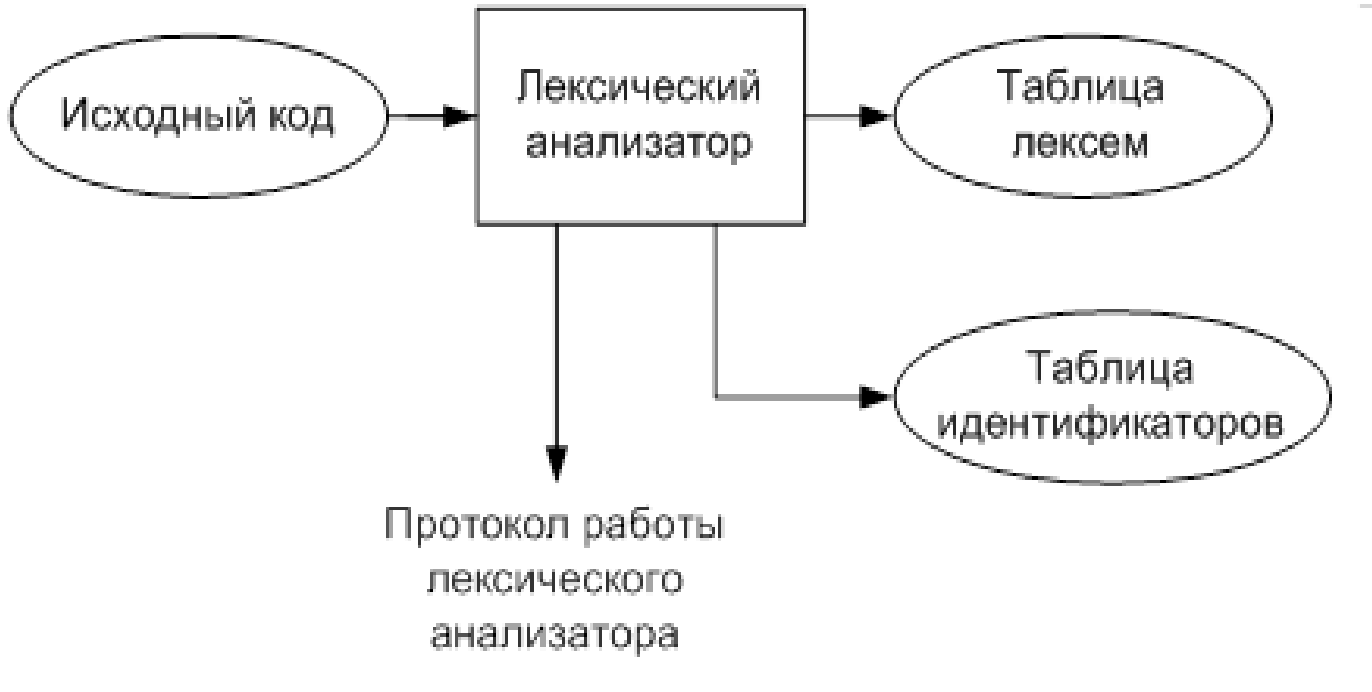


Рис 3. Структура лексического анализатора

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**

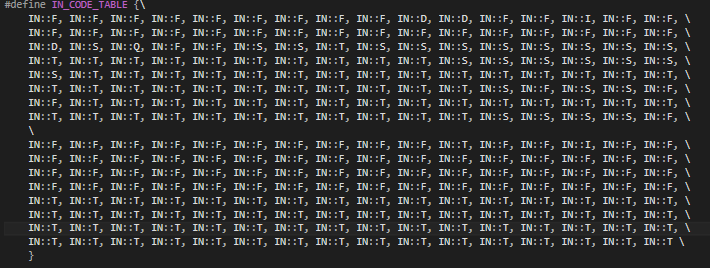


Рис 4. Таблица допустимых символов

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.

??переписать Каждый символ имеет определённое значение:

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

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 |
| bool | t |
| func | f |
| return | r |
| main | m |
| display | d |
| if | u |
| else | e |
| true | l |
| while | w |
| false | l |
| Сепараторы(разделяют ключевые слова) | , | , |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| ; | ; |
| Символы операций(определяют символы арифметических операций) | + | v |
| - | v |
| \* | v |
| / | v |
| ++ | k |
| -- | k |
| & | v |
| | | v |
| > | v |
| < | v |
| ~ | v |
| = | = |

??Сказать, для чего язык, и на примере одной цепочки показать КА, граф и т.д.

Регулярное выражение для идентификатора имеет вид

[a-b]+

**3.5. Основные структуры данных**

Структура для таблицы лексем

struct LexTable

{

int maxize; //емкость таблицы лексем

int size; //текущий размер таблицы лексем

Entry\* table; //массив строк табилцы лексем

};

Структура для содержимого таблицы лексем

struct Entry

{

char lexema; //лексема

int sn; //номер строки в исходном коде

int idxTI; //индекс в таблице идентификаторов

};

Структура для таблицы идентификаторов

struct IdTable //экземпляр таблицы идентификаторов

{

int maxsize; //емкость таблицы идентификаторов < TI\_MAXSIZE

int size; //текущий размер таблицы идентификаторов < maxsize

Entry\* table; //массив строк таблицы идентификаторов

};

Структура содержимого таблицы идентификаторов

struct Entry

{

int idxfirstLE; //индекс первой строки в таблице лексем

char id[ID\_MAXSIZE]; //идентификатор (автоматические усекается до ID\_MAXSIZE)

DATATYPE iddatatype; //тип данных

TYPE idtype; //тип идентификатора

struct

{

int vbool; //значение bool

int vind; //значение integer

struct {

int len; //длина string

char str[TI\_STR\_MAXSIZE]; //символы string

} vstr; //значение string

}value; //значение идентификатора

}; Описание enum для заполнения таблицы идентификаторов

??вынести все эти структуру в отдельное приложение

enum IDDATATYPE {OFF = 0, INT = 1, STR = 2, BOOL = 3 };

1 – integer

2 – string

3 – bool

enum IDTYPE { N = 0, V = 1, F = 2, P = 3, L = 4, O = 5 };

0 – значение по умолчанию

1 – арифметическая операция

2 – функция

3 – параметр функции

4 – литерал

5 – оператор

**3.6. Принцип обработки ошибок**

В случае обнаружения ошибки, она заносится в массив ошибок. Максимально допустимое количество ошибок в лексическом анализе ограничено числом 5. 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 начинаются на 12х??словами. Также при нахождении ошибки записывается позиция этой ошибки в исходном файле, то есть строка. Каждая ошибка имеет префикс LA.



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**

?? ссылаемся на код из приложения, где будет все обозначено

Пример определения идентификатора

*var int z;*

Его разбор лексическим анализатором



Рис 6. Экземпляр таблицы идентификатора

**3.10Контрольный пример**

??ссылка на выражение

Пример исходного кода

*main()*

*{*

*var int x;*

*x = 5;*

*var int y;*

*y=4;*

*var int z;*

*display (z);*

*var str sa;*

*sa = "Hello World";*

*display (sa);*

*while(true)*

*{*

*display(5);*

*}*

*}*

Таблица лексем:

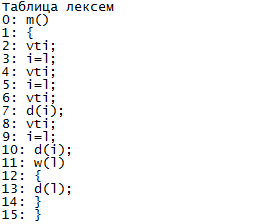


Таблица идентификаторов:

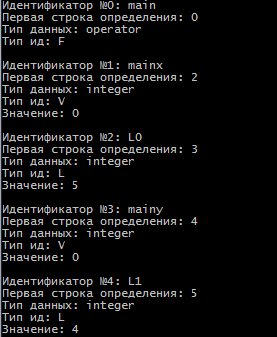
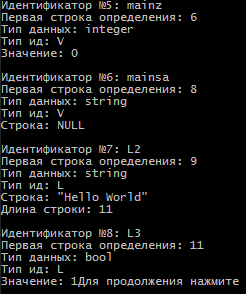
 

Рис. 6,7 Таблица идентификаторов

**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.

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).



Рис. 8. Схема работы синтаксического анализатора

**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

N – множество нетерминальных символов

P – множество правил переходов

S – стартовый символ

В контекстно-свободной грамматике правила имеют вид: , где





 - словарь грамматики 

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

* 1. **Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора содержится в таблице 4.1

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 |

* 1. **Принцип обработки ошибок**

При невозможности найти подходящую цепочку, КА возвращается к предыдущему состоянию и подбирает следующую цепочку в правиле, если это возможно. После чего повторяет шаг по следующей цепочке правил, в случае отсутствия нужной цепочки вызывается специализированная ошибка для правила, до которого максимально добрался анализатор.

* 1. **Структура синтаксического анализатора**

Структура синтаксического анализатора представлена на рисунках 8,9,10

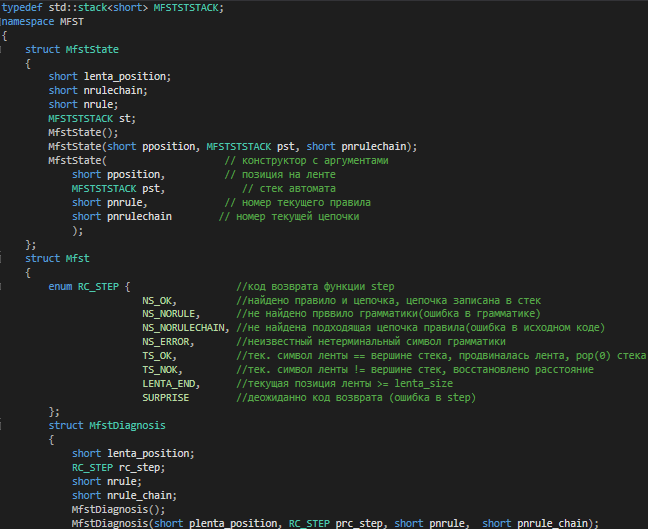


Рис. 8. Магазинный автомат

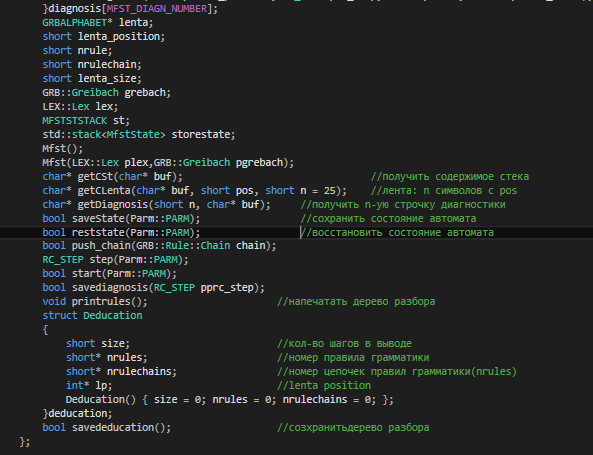


Рис. 9. Магазинный автомат

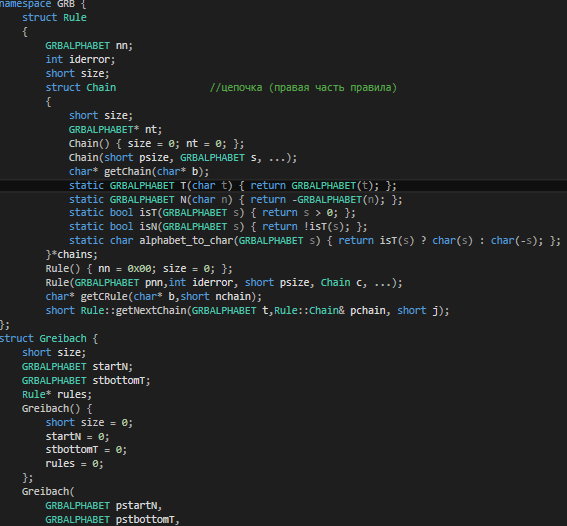


Рис. 10. Грамматика Грейбах

* 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 магазинного автомата**

Принцип работы автомата следующий:

1. в магазин записывается стартовый символ;
2. на основе полученных раннее 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 с 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 вызывается ошибка.

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

* 1. **Контрольный пример**

Пример разбора исходного кода на языке программирования BNI-2016 синтаксическим анализатором представлена на рисунках 11, 12, 13

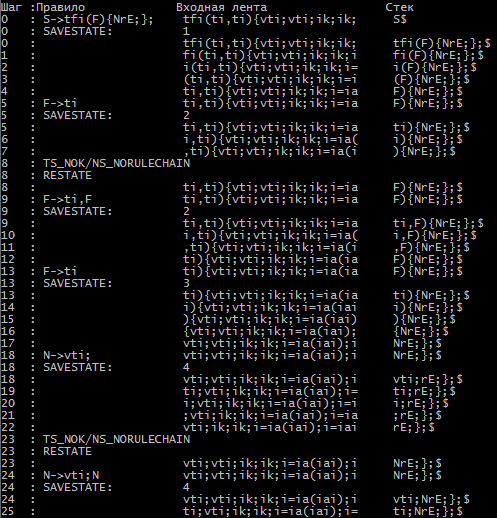


Рис. 11. Трассировка СА

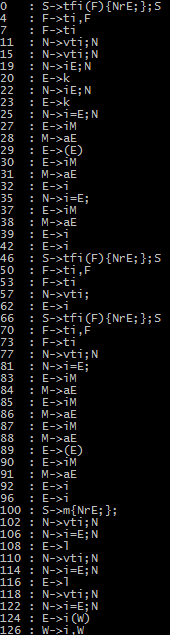
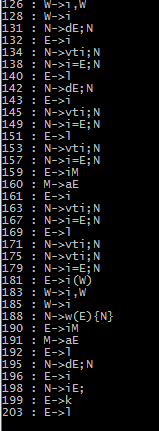
 

Рис. 12, 13. Дерево разбора

На выходе синтаксического анализатора мы получили дерево разбора, которое в дальнейшем будет использовано для генерации кода.

**Глава 5. Разработка семантического анализатора**

**5.1 Структура семантического** **анализатора**

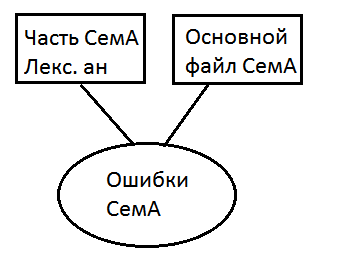


Рис. 14. Структура семантического анализатора

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

**5.2 Функции семантического анализатора**

Таблица 14. Таблица семантических правил

|  |  |
| --- | --- |
| № | Правило |
| 1 | СМА: Наличие функции main |
| 2 | Усечение слишком длинных идентификаторов до 5 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 |
| 11 | ЛА: Округление слишком большого значения int –литерала до INT\_MAX |

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

Перечень сообщений семантического анализатора можно увидеть на рисунке 15.

??РИСУНОК

**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".

**Глава 6. Вычисление выражений**

**6.1 Общие позиции**

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

**6.2 Алгоритм преобразования выражения**

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

**Глава 7. Генерация кода**

**7.1 Общие положения**

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

**7.2 Представление типов данных в оперативной памяти**

Типу данных **INT** соответствует тип данных языка ассемблера **SDWORD**, где хранится целочисленное значение

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

**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. Возврат идет через регистр eax.