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**

**ВСЕ ТАБЛИЦЫ ОФОРМИТЬ ОДНИМ ТИПОМ**

**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:

|  |  |
| --- | --- |
| **;** | 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:

|  |  |
| --- | --- |
| 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 букв. 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**.

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| 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>;  } |
| Условная инструкция | if (<int-идентификатор> или <bool-идентификатор> <условный оператор> <int-идентификатор> или <bool-идентификатор> или <int-литерал> или <bool-литерал> )  {…}  else {…};  (else не обязательно) |
| 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.

|  |  |
| --- | --- |
| 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

|  |
| --- |
| 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

|  |  |
| --- | --- |
| **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):

|  |
| --- |
| 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

|  |  |
| --- | --- |
| **int** strlen (**str**) | длина строки |
| **int** pow (**int, int**) | возведение в степень |
| **str** uppercase(**str**) | буквы верхнего регистра |
| **str** lowercase(**str**) | буквы нижнего регистра |

**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--;*

*s = pow(2,y);*

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

*z = s+z;*

*return z;*

*};*

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

*{*

*var int с;*

*var str func strlen (str a);*

*c = strlen (a) + b;*

*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); // show z*

*var str sa;*

*sa = ‘Hello World’;*

*sa = lowercase(sa);*

*display (sa);*

*sa = lowercase(sa);*

*display (sa);*

*var str sb;*

*sb = “13.02.1998”;*

*var str sc;*

*sc = sa+sb;*

*display (sc);*

*var bool ba;*

*ba = false;*

*var bool bb; // auto initialization* ***true***

*var bool bc;*

*bc = pl(ba,bb);*

*if (bc* **~** *true)*

*{*

*display (bc);*

*}*

*else*

*{*

*display (“bc = false”);*

*}*

*var int ll;*

*ll =5;*

*while(ll)*

*{*

*display(ll);*

*l--;*

*}*

*return 0;*

*}*

**Глава 2. Структура транслятора**

2.1 Компоненты транслятора, их назначение и принципы взаимодействия



Рис. 2 Структура транслятора языка программирования KDV-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

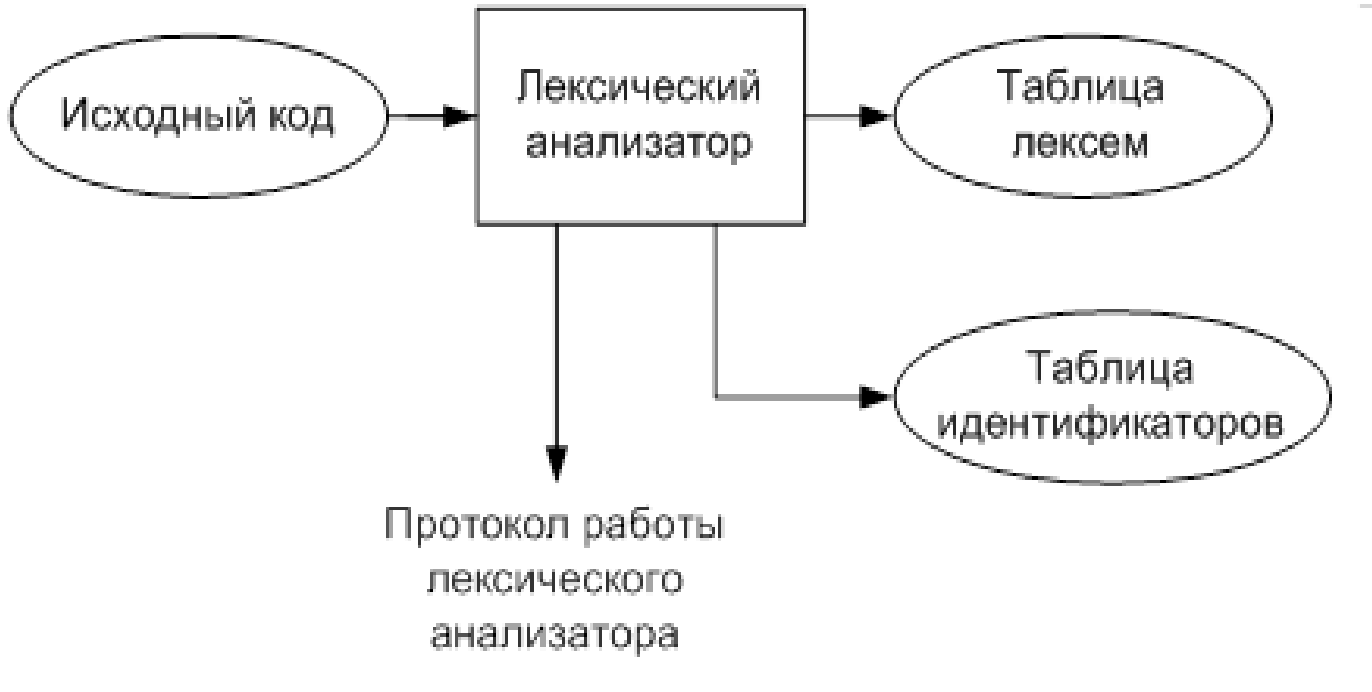
|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:\*.txt | Файл с исходным кодом на языке KDV-2016, имеющий расширение \*.txt | Не предусмотрено, должен быть задан обязательно |
| -lt | Параметр для вывода таблицы лексем в файл log | Таблица лексем 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

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| “log.txt” | Файл с протоколом работы транслятора языка программирования KDV-2016, содержащий таблицу лексем, полученную на этапе лексического анализа и дерево разбора, полученное на этапе синтаксического анализа, ошибки выполнения. |
| “out.txt” | Файл с ассемблерным кодом. |

**Глава 3. Разработка лексического анализатора**

3.1 Структура лексического анализатора



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

3.2

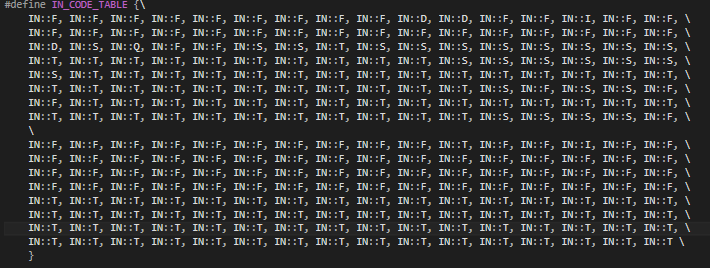


Таблица содержит массив, содержащий типы входных символов, соответствующим символам таблицы ASCII(0-255). При проверке исходного кода идет сравнение типа входного символа и типа символа, соответствующего его индексу в этом массиве.

Каждый символ имеет определённое значением:

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 слова на массив цепочек, в алгоритме которого избыточные символы игноритуются.

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

Табл. ? ???

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | лексема |
| Ключевые слова | var | v |
| str | t |
| int | t |
| bool | t |
| func | f |
| return | r |
| main | m |
| display | d |
| if | u |
| else | h |
| true | l |
| false | l |
| Сепараторы | , | , |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| ; | ; |
| Символы операций | + | + |
| - | - |
| \* | \* |
| / | / |
| ++ | ++ |
| -- | -- |
| & | & |
| | | | |
| > | > |
| < | < |
| ~ | ~ |
| = | = |

Дописать регулярки

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

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 дефолту

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. Также при нахождении ошибки записывается позиция этой ошибки в исходном файле, то есть строка. Каждая ошибка имеет префикс LA.



3.8 Алгоритм лексического анализа не имеет входных параметров

3.9 ??? ссылаемся на код из приложения

3.10 ???Распечатка таблиц