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 TNS-2022»

Выполнил студент Тараканов Никита Сергеевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 5](#_Toc58695193)

[1. Спецификация языка программирования 6](#_Toc58695194)

[1.1 Характеристика языка программирования 6](#_Toc58695195)

[1.2 Определение алфавита языка программирования 6](#_Toc58695196)

[1.3 Применяемые сепараторы 6](#_Toc58695197)

[1.4 Применяемые кодировки 7](#_Toc58695198)

[1.5 Типы данных 7](#_Toc58695199)

[1.6 Преобразование типов данных 8](#_Toc58695200)

[1.7 Идентификаторы 9](#_Toc58695201)

[1.8 Литералы 9](#_Toc58695202)

[1.9 Объявление данных 9](#_Toc58695203)

[1.10 Инициализация данных 10](#_Toc58695204)

[1.11 Инструкции языка 10](#_Toc58695205)

[1.12 Операции языка 11](#_Toc58695205)

[1.13 Выражения и их вычисления 11](#_Toc58695206)

[1.14 Конструкции языка 12](#_Toc58695207)

[1.15 Область видимости идентификаторов 13](#_Toc58695208)

[1.16 Семантические проверки 13](#_Toc58695209)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc58695210)

[1.18 Стандартная библиотека и её состав 14](#_Toc58695211)

[1.19 Ввод и вывод данных 14](#_Toc58695212)

[1.20 Точка входа 15](#_Toc58695213)

[1.21 Препроцессор 15](#_Toc58695214)

[1.22 Соглашения о вызовах 15](#_Toc58695215)

[1.23 Объектный код 15](#_Toc58695216)

[1.24 Классификация сообщений транслятора 15](#_Toc58695217)

[1.25 Контрольный пример 15](#_Toc58695218)

[2. Структура транслятора 16](#_Toc58695219)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 16](#_Toc58695220)

[2.2 Перечень входных параметров транслятора 17](#_Toc58695221)

[2.3 Протоколы, формируемые транслятором 17](#_Toc58695222)

[3. Разработка лексического анализатора 18](#_Toc58695223)

[3.1 Структура лексического анализатора 18](#_Toc58695224)

[3.2 Контроль входных символов 18](#_Toc58695225)

[3.3 Удаление избыточных символов 19](#_Toc58695226)

[3.4 Перечень ключевых слов 20](#_Toc58695227)

[3.5 Основные структуры данных 22](#_Toc58695228)

[3.6 Структура и перечень сообщений лексического анализатора 23](#_Toc58695229)

[3.7 Принцип обработки ошибок 23](#_Toc58695230)

[3.8 Параметры лексического анализатора 24](#_Toc58695231)

[3.9 Алгоритм лексического анализа 24](#_Toc58695232)

[3.10 Контрольный пример 24](#_Toc58695233)

[4. Разработка синтаксического анализатора 25](#_Toc58695234)

[4.1 Структура синтаксического анализатора 25](#_Toc58695235)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 25](#_Toc58695236)

[4.3 Построение конечного магазинного автомата 28](#_Toc58695237)

[4.4 Основные структуры данных 29](#_Toc58695238)

[4.5 Описание алгоритма синтаксического разбора 29](#_Toc58695239)

[4.6 Структура и перечень сообщений синтаксического анализатора 29](#_Toc58695240)

[4.7 Параметры синтаксического анализатора и режимы его работы 29](#_Toc58695241)

[4.8 Принцип обработки ошибок 30](#_Toc58695242)

[4.9 Контрольный пример 30](#_Toc58695243)

[5. Разработка семантического анализатора 31](#_Toc58695244)

[5.1 Структура семантического анализатора 31](#_Toc58695245)

[5.2 Функции семантического анализатора 31](#_Toc58695246)

[5.3 Структура и перечень сообщений семантического анализатора 31](#_Toc58695247)

[5.4 Принцип обработки ошибок 32](#_Toc58695248)

[5.5 Контрольный пример 32](#_Toc58695249)

6. Преобразование выражений 33

[6.1 Выражения, допускаемые языком 33](#_Toc58695250)

[6.2 Польская запись 33](#_Toc58695251)

[6.3 Программная реализация обработки выражений 33](#_Toc58695252)

[6.4 Контрольный пример 34](#_Toc58695253)

[7. Генерация кода 35](#_Toc58695254)

[7.1 Структура генератора кода 35](#_Toc58695255)

[7.2 Представление типов данных в оперативной памяти 35](#_Toc58695256)

[7.3 Статическая библиотека 36](#_Toc58695257)

[7.4 Особенности алгоритма генерации кода 36](#_Toc58695258)

7.5 Входные параметры, управляющие генерацией кода 36

7.6 Контрольный пример 37

[8. Тестирование транслятора 38](#_Toc58695259)

[8.1 Общие положения 38](#_Toc58695260)

[8.2 Результаты тестирования 38](#_Toc58695261)

[Заключение 39](#_Toc58695264)

[Список использованных источников 39](#_Toc58695265)

[ПРИЛОЖЕНИЕ А 42](#_Toc58695266)

[ПРИЛОЖЕНИЕ Б 44](#_Toc58695267)

[ПРИЛОЖЕНИЕ В 47](#_Toc58695268)

[ПРИЛОЖЕНИЕ Г 51](#_Toc58695269)

[ПРИЛОЖЕНИЕ Д 54](#_Toc58695270)

**Введение**

Информационные технологии давно стали неотъемлемой частью жизни любого человека. Вычислительные машины оперируют битами и регистрами, а люди изъясняются на естественном языке или пользуются математическими обозначениями. Упрощенное написание компьютерных программ и стало причиной создания трансляторов для перевода программ, написанных на одних языках, в другие, в том числе в машинный код.

Целью курсового проекта является создание своего языка программирования и транслятора для его обработки. Язык был назван TNS-2022. Разработка языка программирования будет вестись на языке программирования C++, а код на разработанном языке будет транслироваться в ассемблерный.

Основные задачи курсового проекта:

- разработка спецификации языка программирования TNS-2022;

- разработка лексического анализатора;

- разработка синтаксического анализатора;

- разработка семантического анализатора;

- разбор арифметических выражений;

- разработка генератора кода в Assembler;

- тестирование транслятора.

**1. Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык программирования TNS-2022 является компилируемым, не объектно-ориентрованным, строго типизированным, процедурным.

**1.2 Определение алфавита языка программирования**

При написании программ на языке программирования TNS-2022 используется кодировка Windows-1251, таблица символов, поддерживающихся данной кодировкой представлена на рисунке 1.1.

На этапе исполнения используются следующие множества символов: {a, b, ..., z, A, B, ..., Z}, {0, 1, ..., 9}, пробел, символы табуляции и перевода строки, а также специальные символы {:>>, :<<, (, ), [, ], ;, /, \*, +, -, =, ==, <, >}. При описании строковых или символьных литералов можно также использовать множество символов русского алфавита {А, Б, …, Я, а, б, …, я}.

**1.3 Применяемые сепараторы**

Во время обработки исходного текста программы, записанной на языке программирования, символы сепараторы служат в качестве разделителей цепочек языка с целью разделения исходного текста на токены. Перечень используемых сепараторов представлен в таблице 1.1.

Таблица 1.1 – Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы)-сепаратор(ы) | Назначение |
| пробел | Разделитель цепочек. Допускается везде, кроме названий идентификаторов и ключевых слов |
| {} | Блок функции или условной конструкции/цикла |
| , | Разделитель параметров в функции |
| () | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| [] | Блок условного выражения для цикла/условной конструкции |
| > < == | Логические операции сравнения (операции больше, меньше, равно), используемые в условном выражении цикла/условной конструкции |
| \* + - / | Арифметические операции |
| ; | Разделитель программных инструкций |
| :>> :<< | Операторы сдвигов |
| = | Оператор присваивания |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| Символ(ы)-сепаратор(ы) | Назначение |
| “” | Строковый литерал |
| ‘’ | Символьный литерал |

**1.4 Применяемые кодировки**

Для написания исходного текста программ на языке TNS-2022 используется кодировка Windows-1251. Символы, поддерживающиеся в кодировке Windows-1251 представлены на рисунке 1.1.



Рисунок 1.1 – Таблица кодировки Windows-1251

**1.5 Типы данных**

В языке программирования TNS-2022 реализованы 4 фундаментальных типа данных: символьный, строковый, целочисленный без знака, логический. Описание фундаментальных типов приведено в таблице 1.2.

Таблица 1.2 – Типы данных TNS-2022

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный беззнаковый тип данных unsint32 | Фундаментальный тип данных. Используется для работы с целочисленными беззнаковыми значениями. В памяти занимает 4 байта. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Характеристика |
|  | Максимальное значение: 4294967295.  Минимальное значение: 0.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления нацело;  **=** (бинарный) – оператор присваивания;  > (бинарный) – оператор сравнения(больше);  < (бинарный) – оператор сравнения(меньше);  == (бинарный) – оператор сравнения(равенство);  :>> (бинарный) – оператор сдвига вправо;  :<< (бинарный) – оператор сдвига влево; |
| Символьный тип данных symbol | Фундаментальный тип данных. Используется для работы с символьными значениями. В памяти занимает 1 байт.  Инициализация по умолчанию: значение ‘\0’.  Поддерживаемые операции:  **=** (бинарный) – оператор присваивания. |
| Логический тип данных logical | Фундаментальный тип данных. Используется для работы с 2 значениями: true(истина), false(ложь). В памяти занимает 1 байт.  Инициализация по умолчанию: значение false(0).  Поддерживаемые операции:  **=** (бинарный) – оператор присваивания. |
| Строковый тип данных line | Фундаментальный тип данных. Используется для работы с  последовательностью символьных значений. В памяти каждый символ занимает 1 байт, в следствие чего и будет строиться размер всей строки.  Максимальная длина строки: 255 символов.  Инициализация по умолчанию: значение ‘\0’.  Поддерживаемые операции:  **=** (бинарный) – оператор присваивания. |

**1.6 Преобразование типов данных**

Преобразование типов данных в языке программирования TNS-2022 не предусмотрено. Язык является строго типизированным.

**1.7 Идентификаторы**

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Имена идентификаторов могут содержать символы латинского алфавита нижнего и верхнего регистра, и цифры. Имя идентификатора должно начинать с латинской буквы, и не должно совпадать с ключевыми словами. Максимальная длина имени идентификатора равна 15 символам, при вводе большего количество символов имя идентификатора будет усекаться до 15 символов. Типы идентификаторов: имя переменной, имя функции/процедуры, параметр функции. Регулярное выражение для разбора имени идентификатора: [A, ..., Z]\* [a, ..., z] [0, ..., 9]\* [A, ..., Z]\* [0, ..., 9]\* [a, …, z]\*.

**1.8 Литералы**

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Типы литералов языка TNS-2022 представлены в таблице 1.3.

Таблица 1.3 – Литералы языка TNS-2022

|  |  |  |
| --- | --- | --- |
| Литералы | Регулярное выражение | Описание |
| Беззнаковый целочисленный литерал | [1, ..., 9]+ [0, ..., 9]\* | Последовательность цифр 0…9. |
| Символьный литерал | [a, ..., z | A, ..., Z | 0, ..., 9 | + | - | = | / | \* | ! | ?] | Символ, заключённый в  одинарные кавычки |
| Строковый литерал | [a, ..., z | A, ..., Z | 0, ..., 9 | + | - | = | / | \* | ! | ? | =]\* | Набор символов (от 1 до 255), заключённых в двойные кавычки |

**1.9 Объявление данных**

Объявление данных в языке программирования TNS-2022 выглядит следующим образом:

<тип данных> <имя идентификатора> [= <значение>];

Для объявления функций используется ключевое слово function, после которого указывается тип функции (если функция возвращает значение), или ключевое слово procedure, если функция ничего не возвращает, а после ключевого слова или типа данных идёт имя функции или процедуры. Далее обязателен список параметров и тело функции, заключённое в фигурные скобки.

Объявление функции выглядит следующим образом:

function <тип данных> <имя идентификатора> (<список параметров>)

{

<тело функции>

return <литерал | идентификатор>;

}

Объявление процедуры выглядит следующим образом:

procedure <имя идентификатора> (<список параметров>)

{

<тело функции>

return;

}

**1.10 Инициализация данных**

Инициализация данных подразумевает под собой присвоение переменной значения в месте её объявления. Для инициализации переменной в месте объявления после знака ‘=’(присвоение) требует указать литерал, соответствующий типу данных переменной, либо же имя переменной, объявленной ранее. При объявлении без инициализации переменной будет присвоено значение по умолчанию.

Примеры:

unsint32 variable; (присвоится значение по умолчанию для типа данных unsint32(0)),

symbol symb22 = ‘a’; (присвоится значение литерала после знака присваивания).

**1.11 Инструкции языка**

Инструкции языка программирования TNS-2022 представлены в таблице 1.4.

Таблица 1.4 – Инструкции языка программирования TNS-2022

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменой | <тип данных> <идентификатор>; |
| Объявление функции | function <тип данных> <идентификатор>(<параметры>) {<блок кода>} |
| Инициализация переменной | <тип данных> <идентификатор> = <значение>;  Под значением подразумевается литерал данного типа данных, либо же переменная данного типа данных. |
| Присвоение переменной значения | <идентификатор> = <значение>;  Под значением подразумевается литерал данного типа данных, либо же переменная данного типа данных. |
| Возврат значения из функции | return <идентификатор | литерал>; |
| Выход из процедуры | return; |
| Объявление процедуры | procedure <идентификатор>(<параметры>) {<блок кода>}  Блок кода процедуры должен оканчиваться оператором return. |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Вывод данных в стандартный поток | writeln <литерал | идентификатор> (вывод значения с переходом на новую строку);  write <литерал | идентификатор> (вывод значения без перехода на новую строку) |
| Условная конструкция | when[<условие>] {<блок кода>} otherwise {<блок кода>} |
| Вызов функции или процедуры | <имя функции | процедуры>(<параметры>);  Список параметров может быть пустым |
| Цикл | iteration[<условие>]{<блок кода>} |

**1.12 Операции языка**

Все возможные операции языка программирования TNS-2022 представлены в таблице 1.5. Приоритетность арифметических операций умножения и деления выше приоритетности операций сложения и вычитания. Наивысший приоритет имеет операция ().

Таблица 1.5 Операции языка TNS-2022

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметический | + - сложение  - - вычитание  \* - умножение  / - деление нацело |
| Логический | > - больше  < - меньше  == – проверка на равенство |
| Сдвиговый | :>> - сдвиг вправо  :<< - сдвиг влево |
| Присваивание | = - присвоение значения |

**1.13 Выражения и их вычисления**

Выражения - это фундаментальный инструмент для записи алгоритмов, который является одним из важнейших задач языков программирования. Для вычисления выражений формируются определенные правила. Правила вычисления выражений языка программирования TNS-2022:

1. Приоритет операций умножения и деления выше, чем приоритет операций сложения, вычитания и сдвиговых.

2. Операция круглые скобки имеет наивысший приоритет и служит для смены приоритета операций.

3. Выражение записывается в одну строку, без переноса.

4. Использование двух подряд идущих операторов не поддерживается.

5. Допускается одним из операндов выражения использовать вызов функции, которой возвращает значение, в соответствии с типом данных.

Перед генерацией кода выражение преобразуется в форму обратной польской записи, для вычисления его на языке ассемблера.

**1.14 Конструкции языка**

Исходный текст программы на языке программирования TNS-2022 оформляется в виде главной функции, точки входа в программу, также может содержать пользовательские функции или процедуры для упрощения алгоритма программы. При составлении функций рекомендуется выделять блоки и фрагменты отступами для лучшей читаемости кода.

Конструкции языка TNS-2022 представлены в таблице 1.6.

Таблица 1.6 – Конструкции языка программирования TNS-2022

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция(точка входа в программу) | MAIN  {  <блок кода>  } |
| Пользовательская функция | function <тип данных> <имя идентификатора> (<параметры>)  {  <блок кода>  return <идентификатор | литерал>;  }  Тип возвращаемого значения должен соответствовать типу функции. |
| Пользовательская процедура | procedure <имя идентификатора> (<параметры>)  {  <блок кода>  return;  } |
| Условная конструкция | when[<условие>]  {  <блок кода>  }  otherwise  { |

Продолжение таблицы 1.6

|  |  |
| --- | --- |
| Конструкция | Реализация |
|  | <блок кода>  {  Если условие указанное после оператора when вернёт true(1), то выполниться блок кода, указанный в фигурный скобках после when, иначе выполниться блок кода указанный в фигурных скобках после otherwise. |
| Цикл | iteration[<условие>]  {  <блок кода>  } |

**1.15 Область видимости идентификаторов**

Область видимости реализована также, как и в языке программирования C++, то есть сверху вниз. Переменные объявленные в одной функции, недоступны в другой.

Все идентификаторы являются локальными и обязаны быть в блоке кода соответствующей функции. Глобальной области видимости нет. Параметры, передаваемые в функцию, доступны только в функции, в которой они объявлены.

**1.16 Семантические проверки**

В языке программирования TNS-2022 выполняются следующие семантические проверки:

* точка входа в программу должна быть единственной, то есть должно

быть только одно объявление функции MAIN;

* не допускается использование идентификаторов до их объявления;
* проверка на превышение допустимого размера типов данных;
* проверка правильности составления условных конструкций, циклов, процедур и функций;
* проверка на соответствие типа функции к типу возвращаемого значения;
* проверка правильности передаваемых в функцию или процедуру параметров;
* не допускается повторное объявление идентификатора;
* проверка соответствия операции к допустимым операциям для данного типа данных.

**1.17 Распределение оперативной памяти на этапе выполнения**

Транслированный код на языке программирования TNS-2022 использует 2 допустимые области памяти: сегмент констант и сегмент данных. В сегмент констант заносятся все литералы, а в сегмент данных заносятся все идентификаторы.

**1.18 Стандартная библиотека и ее состав**

В языке программирования TNS-2022 присутствует стандартная библиотека, которая подключается к исходному коду программы на этапе трансляции в язык ассемблера. Перечень функций стандартной библиотеки приведен в таблице 1.7.

Таблица 1.7 – Функции стандартной библиотеки языка программирования TNS-2022

|  |  |
| --- | --- |
| Функция | Описание |
| line getdata() | Функция, возвращающая текущую дату в формате строки, имеющей вид “YY.MM.DD” |
| line gettime() | Функция, возвращающая текущее время в формате строки, имеющей вид “HH:MM:SS” |

Стандартная библиотека написана на языке программирования С++ и подключается к исходному коду на этапе генерации кода. Вызов стандартной функции доступен в том же формате, что и вызов пользовательской функции.

Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Эти функции представлены в таблице 1.8.

Таблица 1.8 – Дополнительные функции стандартной библиотеки языка программирования TNS-2022

|  |  |
| --- | --- |
| Функция | Описание |
| void write() | Функция для вывода значения в стандартный поток |
| void writeln() | Функция для вывода значения в стандартный поток с переводом строки |

**1.19 Ввод и вывод данных**

Вывод данных осуществляется с помощью операторов writeln и write, допускается использование операторов с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке программирования С++ и вызываются из транслированного кода, конечному пользователю недоступны. Операторы writeln и write в транслированном коде будут заменены вызовом библиотечных функций. Библиотека, содержащая нужные функции, подключается на этапе генерации кода.

Ввод данных не поддерживается языком.

**1.20 Точка входа**

Программа, написанная на языке программирования TNS-2022 должна содержать точку входа в программу, главную функцию MAIN, с первой инструкции которой начнется выполнение программы.

**1.21 Препроцессор**

Препроцессор для языка программирования TNS-2022 отсутствует.

**1.22 Соглашение о вызовах**

В языке используется соглашение о вызовах под названием stdcall – это значит, что в программе все параметры функции будут передаваться через стек справа налево и память будет освобождаться вызываем кодом.

**1.23 Объектный код**

Программа, исходный код которой написан на языке программирования TNS-2022, будет транслироваться в ассемблерный код, а после из ассемблерного кода в объектный код**.**

**1.24 Классификация сообщений транслятора**

Сообщения, генерируемые транслятором, должны давать подробную информацию об возникшей ошибке и месте ошибки в программе, чтобы пользователь мог с легкостью решить возникшую проблему. Сообщения транслятора и их описание приведены в таблице 1.9.

Таблица 1.9 – Классификация сообщений транслятора

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

**1.25 Контрольный пример**

Контрольный пример демонстрирует все возможности языка, а именно: объявление переменных, объявление функции, выражения с операциями, инициализация переменных, конструкции языка и прочее. Исходный код приведён в приложении А.

**2 Структура транслятора**

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

В языке программирования TNS-2022 исходный код транслируется в ассемблерный код. Транслятор языка разделен на взаимосвязанные части, которые выполняют свои определенные функции. Графическое отображение структуры транслятора представлено на рисунке 2.1.



Рисунок 2.1 – Структура транслятора языка программирования TNS-2022

На первом этапе исходный код поступает на вход лексическому анализатору, который в соответствии с набором допустимых цепочек языка проверяет правильность написания исходного кода, и в результате генерирует 2 таблицы: таблицу лексем(все слова исходного языка преобразуются в токены) и таблицу идентификаторов. Каждой лексеме сопоставляется ее запись в таблице идентификаторов. Таблица идентификаторов хранит дополнительную информацию некоторых лексем. Таблица лексем и таблица идентификаторов подаются на вход следующему этапу транслятора: синтаксическому анализатору.

Синтаксический анализатор – этап трансляции исходного кода, выполняющий синтаксический анализ(проверка грамматики языка). На вход синтаксическому анализатору поступают таблица лексем и таблица идентификаторов, и при отсутствии ошибок на данном этапе, выходными данными синтаксического анализатора является дерево разбора.

Семантический анализатор – этап трансляции исходного кода, выполняющий семантический анализ исходного кода, для нахождения ошибок, которые не смогли отследить 2 прошлых этапа. Входными данными семантического анализатора являются таблица лексем и таблица идентификаторов.

Генерация кода – заключительный этап трансляции исходного кода, выполняющий генерацию кода в язык ассемблера на основе полученных данных после предыдущих этапов трансляции. На вход генератору кода поступает таблица лексем и таблица идентификаторов, на основе чего, генератор формирует код на языке ассемблера.

**2.2 Перечень входных параметров транслятора**

Для формирования файлов с информацией о результате работы транслятора используются входные параметры, приведенные в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка TNS-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке TNS-2022, имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | <имя in-файла>.log |

**2.3 Протоколы, формируемые транслятором**

В ходе работы транслятора, на каждом из его этапов формируется протокол работы. В таблице 2.2 приведен перечень протоколов, формируемых транслятором и их содержимое.

Таблица 2.2 – Протоколы, формируемые транслятором языка TNS-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования TNS-2022. 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. При возникновении ошибок на одном из этапов, содержимое ошибки будет выведено в файл с протоколом. |

**3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первый этап трансляции исходного кода называется лексическим анализом, а программа выполняющая анализ – лексический анализатор. На вход лексическому анализатору подается исходный текст программы, записанный на языке программирования. Лексический анализатор в свою очередь проверяет лексемы на соответствие к доступным лексемам языка, и преобразует исходный текст в массив токенов.

Для примера можно выделить следующие лексические единицы: идентификаторы, литералы, операторы языка и другие. Каждая лексема заносится в таблицу лексем, а также лексемы, о которых нужна дополнительная информация, заносятся в таблицу идентификаторов.

Можно выделить следующие функции лексического анализатора:

* удаление пустых символов(пробелов/знаков табуляций/символов

перехода на новую строку) и комментариев;

* распознавание идентификаторов и ключевых слов;
* распознавание разделителей и операторов языка.

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 правильного разбиения исходного текста на слова, все символы, опираясь на код символа в кодировке Windows-1251, разделяются по категориям.

Таблица входных символов представлена на рисунке 3.2, категории входных символов представлены в таблице 3.1.

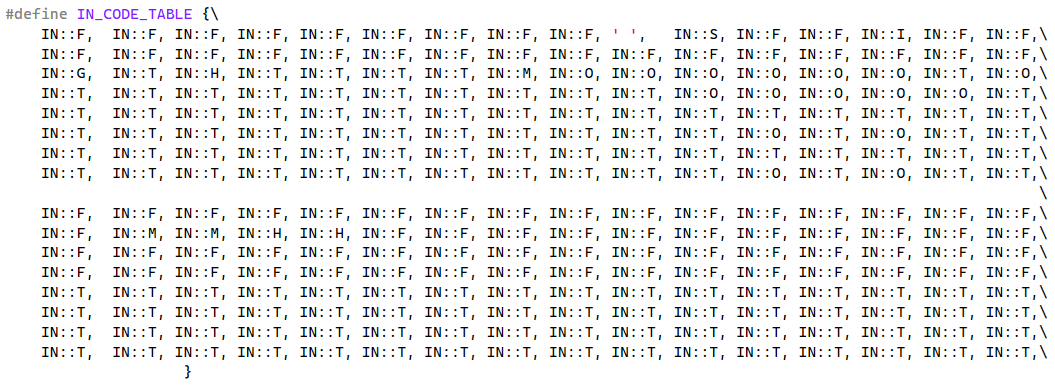


Рисунок 3.2. Таблица контроля входных символов

Таблица 3.1 – Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Арифметические операции | O |
| Перевод строки | S |
| Пробел | G |
| Строковый литерал | H |
| Символьный литерал | M |

**3.3. Удаление избыточных символов**

Избыточными символами исходного кода являются пробелы, знаки табуляции и перевод на новую строку. Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Алгоритм удаления избыточных символов:

* посимвольно считываем файл с исходным кодом программы;
* встреча знака пробела, табуляции или перевода на новую строку

подразумевает под собой встречу символа сепаратора;

* в отличие от других лексем, избыточные символы мы игнорируем.

**3.4. Перечень ключевых слов**

Соответствие ключевых слов и лексем приведено в таблице 3.2.

Таблица 3.2 – Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| unsint32, line, symbol, logical | t | Названия типов данных языка. |
| Идентификатор | i | Длина имени идентификатора – 15 символов. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| procedure | p | Ключевое слово для процедур – функций, не возвращающих значения. |
| return | e | Выход из функции/процедуры. |
| MAIN | m | Главная функция. |
| write | k | Вывод данных. |
| writeln | n | Вывод данных с переходом на новую строку.. |
| when | ? | Указывает начало условного оператора. |
| otherwise | ! | Ложная ветвь условного оператора. |
| iteration | с | Цикл |
| ; | ; | Разделение инструкций языка. |
| , | , | Разделение параметров функций. |
| { | { | Начало тела функции/процедуры, либо же условного оператора/цикла. |
| } | } | Закрытие тела функции/процедуры, либо же условного оператора/цикла. |
| [ | [ | Начало условного выражения. |
| ] | ] | Конец условного выражения. |
| :<< | q | Знак сдвига влево. |
| :>> | w | Знак сдвига вправо. |

Продолжения таблицы 3.2

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| + | + | Знак сложения. |
| - | - | Знак вычитания. |
| / | / | Знак деления. |
| \* | \* | Знак умножения. |
| > | > | Знак больше. |
| < | < | Знак меньше. |
| == | e | Знак равно |

Пример реализации таблицы лексем представлен в приложении Б.

Каждой цепочке символов соответствует свой конечный автомат, по которому происходит разбор входной цепочки. На каждый автомат подается цепочка символов, пока не встретится автомат, способный разобрать ее. В случае успешного разбора цепочки, она записывается в таблицу лексем. Если лексеме требуется дополнительная информация, то она так же заносится в таблицу идентификаторов.

Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.

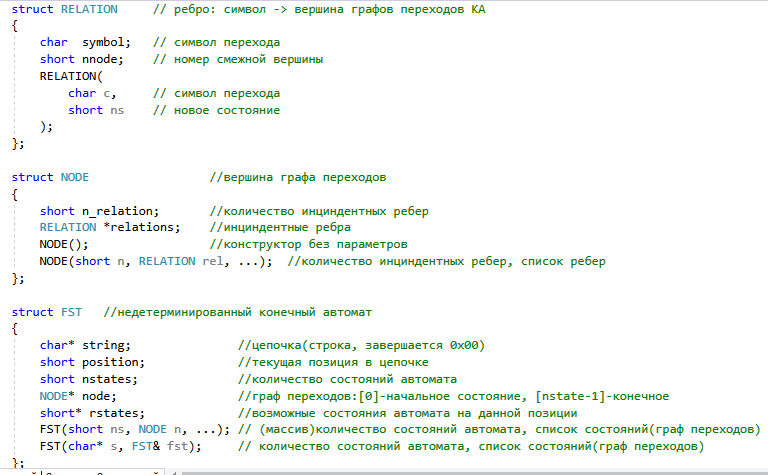


Рисунок 3.3 – Структура конечного автомата

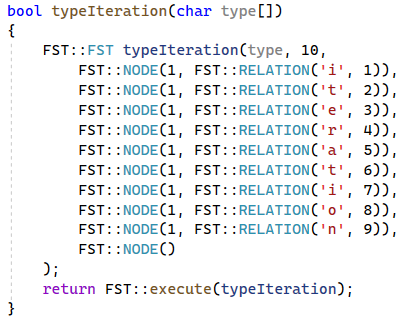


Рисунок 3.4 – Пример реализации графа конечного автомата для iteration

**3.5. Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов.

Таблица лексем содержит номер лексемы, лексему (lexemax), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов(idxTI), если лексема является идентификатором, если лексема не является идентификатором, то idxTI будет равняться -1. Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и его значение (value).

Код C++ со структурой таблицы лексем представлен на рисунке 3.5. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.6.

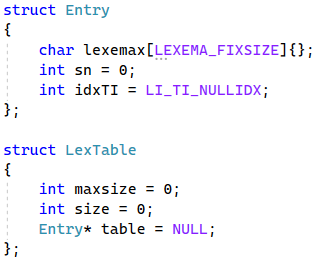


Рисунок 3.5 – Структура таблицы лексем

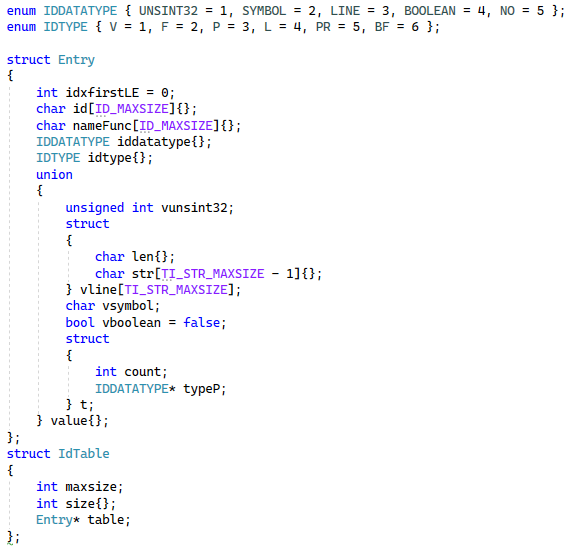


Рисунок 3.6 – Структура таблицы идентификаторов

**3.6. Структура и перечень сообщений лексического анализатора**

Ошибки, возникающие на этапе лексического анализатора, фиксируются в протокол, заданный входным параметров –log. В случае возникновения ошибки производится ее запись в протокол, с точным описанием ошибки и позиции, где она возникла. Если в процессе анализа находится более 3-ёх ошибок, то анализ останавливается, также анализ может остановится при нахождении критической ошибки.

Таблица 3.3 – Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Номер сообщения | Текст сообщения |
| 200 | Превышен максимальный размер таблицы идентификаторов |
| 201 | Превышено максимальное количество идентификаторов |
| 202 | Идентификатор не определен |
| 203 | Превышен максимальный размер таблицы лексем |
| 204 | Превышено максимальное количество лексем |
| 205 | Лексема не определена |
| 206 | Нераспознанная лексема |

**3.7. Принцип обработки ошибок**

Для обработки ошибок, лексический анализатор использует структуру ERROR\_ENTRY для вывода в файл протокола ошибки, возникшей на этапе лексического анализа. При возникновении ошибки, лексический анализ работает дальше, как будто прошлой ошибки не было.

**3.8. Параметры лексического анализатора**

Входным параметром лексического анализатора является исходный текст программы, написанный на языке TNS-2022.

Результат работы лексического анализатора, а именно таблица лексем и таблица идентификаторов, выводится в файл журнала, и хранится в самой программе, поскольку он еще пригодится для следующих этапов трансляции.

**3.9. Алгоритм лексического анализа**

Алгоритм работы лексического анализатора можно описать следующим образом:

* проверяет символы исходного кода на допустимость в данном языке, удаляя лишние пробелы, знаки табуляции и добавляя сепаратор для вычисление текущей строки;
* для отдельной цепочки выполняется функция ее распознавания;
* при успешном распознавании информация о данной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к 1 этапу;
* формирует протокол работы лексического анализатора;
* при неудачном распознавании цепочки, выдается диагностическое сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «symbol» представлен на рисунке 3.7, где S0 – начальное, а S6 – конечное состояние автомата.

s y m b o l

Рисунок 3.7 – Пример графа переходов для цепочки symbol

**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 является 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 TNS-2022 используется контекстно-свободная грамматика, где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

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.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

NS – нетерминальные символы, представленные несколькими заглавными

буквами латинского алфавита.

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 – Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->ftiPTS  S->ftiPT;S  S->piPJS  S->m{K}  S->m{K}; | Стартовые правила, описывающее общую структуру программы |
| P | P->(L)  P->() | Правила для параметров объявляемых функций |
| L | L->ti  L->ti,L | Правила для списка параметров функции |
| T | T->{rD;}  T->{rD;};  T->{KrD;}  T->{KrD;}; | Правила для тела функции |
| D | D->i  D->l | Правила случаев требования только литерала или переменной |
| K | K->ti=E;K  K->ti;K  K->i=E;K  K->iC;K  K->ti=E;  K->ti;  K->i=E;  K->iC;  K->?ZRMK  K->?ZR  K->?ZRK  K->?ZRM  K->cZR  K->kD;  K->Nd;  K->kD;K  K->nD;K | Правила для конструкции тела функции |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| E | E->i  E->l  E->(E)  E->(E)AE  E->iAe  E->iCAE  E->iC  E->lAE | Правила для выражений языка |
| A | A->+  A->-  A->\*  A->/  A->q  A->w | Правила арифметических операций |
| C | С->(N)  C->() | Правила для передачи параметров при вызове функции |
| N | N->i  N->l  N->i,N  N->l,N | Правила для передаваемых параметров |
| J | J->{r;};  J->{r;}  J->{Kr;};  J->{Kr;} | Правила для конструкции тела процедуры |
| Z | Z->[iOi]  Z->[lOi]  Z->[lOl]  Z->[iOl]  Z->[D] | Правила для условного выражения |
| O | O->>  O-><  0->e | Правила для операторов сравнения |
| R | R->{X}  R-><X}; | Правила для тела условной конструкции/цикла |
| X | X->i=E;X  X->iC;X  X->i=E;  X->iC;  X->kD;  X->nD; | Правила для конструкции тела условной конструкции/цикла |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
|  | X->kD;X  X->nD;X |  |
| M | M->!R | Программные конструкции для otherwise |

**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. |
|  | 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. |
|  | 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 TNS-2022. Данные структуры представлены 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 4.2.

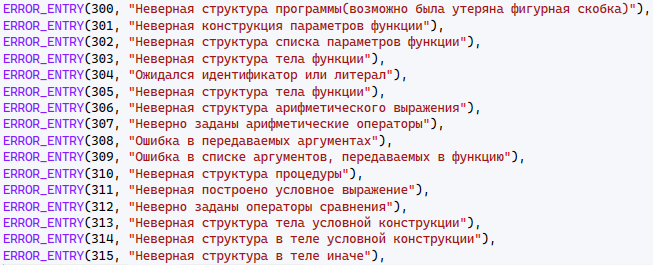


Рисунок 4.2 – Сообщения синтаксического анализатора

**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 синтаксического 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 исходнойпоследовательности 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 синтаксического 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.

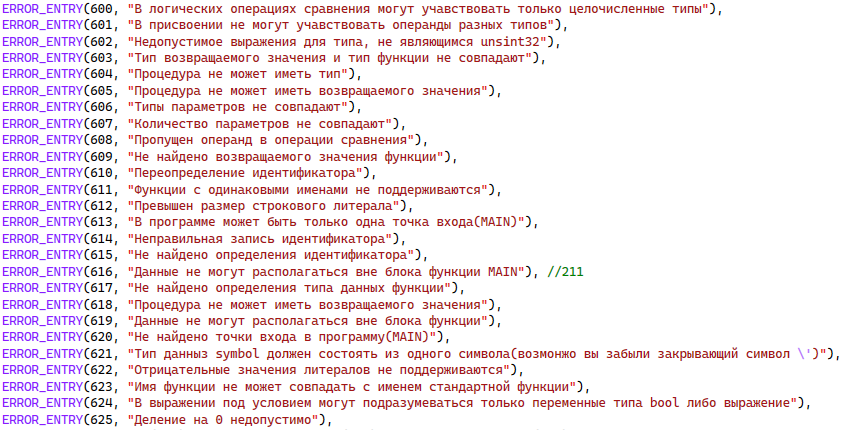


Рисунок 5.2 – Перечень сообщений семантического анализатора

**5.4. Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметром. 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  {  write 2;  }  MAIN  {  write 4;  } | Ошибка 613, В программе может быть только одна точка входа(MAIN) |
| MAIN  {  a = 5;  unsint32 a;  } | Ошибка 615, Не найдено определения идентификатора |

**6. Преобразование выражений**

**6.1. Выражения, допускаемые языком**

В языке TNS-2022 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 3 |
| \* | 2 |
| / | 2 |
| :>> | 1 |
| :<< | 1 |
| + | 1 |
| - | 1 |

**6.2. Польская запись и принцип ее построения**

Все выражения языка TNS-2022 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов. Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

**6.3. Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

**6.4. Контрольный пример**

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду.

Таблица 6.2 – Преобразование выражения к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i=i+i/5-i\*5ql | i=iil/+il\*-lq |
| i=(i+l)\*l | i=il+l\* |

**7. Генерация кода**

**7.1 Структура генератора кода**

В языке TNS-2022 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода TNS-2022 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

**7.2. Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены в сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке TNS-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка TNS-2022 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке TNS-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| unsint32 | sdword | Хранит целочисленный тип данных. |
| logical | dword | Хранит целочисленный тип данных. |

Продолжение таблицы 7.1

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке TNS-20212 | Тип идентификатора на языке ассемблера | Пояснение |
| line | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |
| symbol | dword | Хранит указатель на строку, состоящую из 1-го символа. |

**7.3. Статическая библиотека**

В языке TNS-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Список функций и их описание приведены в разделе 1.18.

**7.4. Особенности алгоритма генерации кода**

В языке TNS-2022 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

Рисунок 7.2 – Структура генератора кода

**7.5. Входные параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке TNS-2022. Результаты работы генератора кода выводятся в файл с расширением .asm.

7**.6. Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.3.

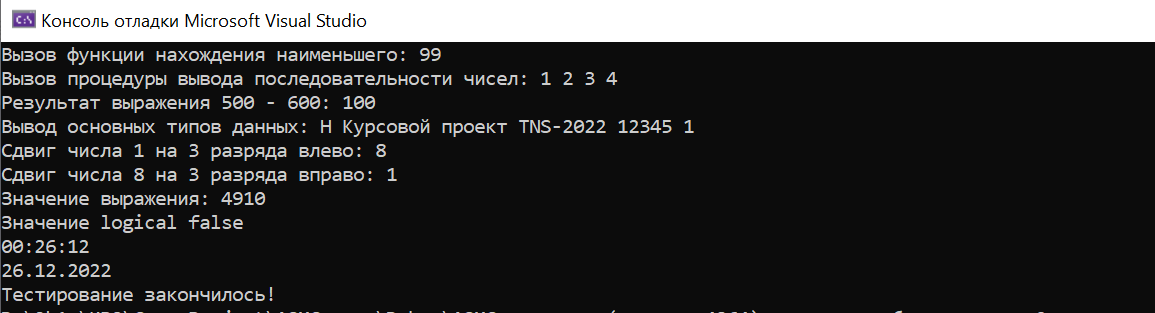


Рисунок 7.3 – Результат работы программы на языке TNS-2022

**8. Тестирование транслятора**

**8.1. Общие положения**

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программы: ошибка записывается в файл журнала построения. Если ошибок обнаружено не было, то в файле журнала будет выведена вся информация о проекте и проект скомпилируется.

**8.2 Результаты тестирования**

Результат тестирования лексического анализатора приведен в таблице 8.1.

Таблица 8.1 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| unsint32 .a; | Ошибка 206: Нераспознанная лексема |

Результаты тестирования синтаксического анализатора приведены в таблице 8.2.

Таблица 8.2 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| MAIN  {  return;  } | Ошибка 300: Неверная структура программы |
| MAIN  {  write 2 + 2;  } | Ошибка 304: Ожидался идентификатор или литерал |
| MAIN  {  unsint32 a = 2 ++2;  } | Ошибка 306: Неверная структура арифметического выражения |

Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.3.

Таблица 8.3 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| procedure a()  {  return 2;  } | Ошибка 605: Процедура не может иметь возвращаемого значения |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| unsint32 a; | Ошибка 616: Данные не могут располагаться вне блока функции MAIN |
| MAIN  {  symbol a = ‘22’;  } | Ошибка 621:Тип данных symbol должен состоять из одного символа(возможно вы забыли закрывающий символ ' |
| MAIN  {  when[“aaa”]  {  write 2;  }  } | Ошибка 624: В выражении под условием могут подразумеваться только переменные типа bool либо выражение |

**Заключение**

В ходе выполнения курсовой работы был разработан полноценный транслятор для языка программирования TNS-2022. Таким образом, были выполнены основные задачи курсовой работы:

* разработка спецификации языка программирования TNS-2022;
* разработка структуры транслятора;
* разработка лексического распознавателя;
* разработка синтаксического распознавателя;
* разработка семантического распознавателя;
* разработка преобразование выражений в форму обратной польской записи;
* разработка генератора кода в Assembler;
* выполнено тестирования работоспособности транслятора.

Окончательный стандарт языка TNS-2022 содержит в себе:

* поддержку 4 фундаментальных типов данных;
* поддержку операторов вывода;
* поддержку стандартных функций языка;
* поддержку арифметических, сдвиговых и логических операторов;
* поддержание процедур, функций, циклов и условных конструкций.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

## **Литература**

1. Курс лекций по предмету «Конструирование программного обеспечения» Наркевич А.С.

2. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

3. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. – 3‑е изд. – Москва: Вильямс, 2003. - 429 с.

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

**Приложение А**

|  |
| --- |
|  |

Листинг 1 – Контрольный пример

**Приложение Б**

|  |  |
| --- | --- |
| 1 | 1  1 |

|  |
| --- |
| 1 |

Рисунок 1 – Таблица лексем

|  |
| --- |
|  |

Рисунок 2 – Таблица идентификаторов

**Приложение В**

|  |
| --- |
| typedef short GRBALPHABET;  #define NS(n) GRB::Rule::Chain::N(n)  #define TS(n) GRB::Rule::Chain::T(n)  #define ISNS(n) GRB::Rule::Chain::isN(n)  namespace GRB  {  struct Rule  {  GRBALPHABET nn;  int iderror{};  short size{};  struct Chain  {  short size{};  GRBALPHABET\* nt{};  Chain() { size = 0, nt = 0; }  Chain(short psize, GRBALPHABET s, ...);  char\* getCChain(char\* b);  static GRBALPHABET T(char t) { return GRBALPHABET(t); };  static GRBALPHABET N(char n) { return -GRBALPHABET(n); };  static bool isT(GRBALPHABET s) { return s > 0; };  static bool isN(GRBALPHABET s) { return !isT(s); };  static char alphabet\_to\_char(GRBALPHABET s) { return isT(s) ? char(s) : char(-s); };  } \*chains{};  Rule() { nn = 0x00; size = 0; };  Rule(GRBALPHABET pnn, int iderror, short psize, Chain c, ...);  char\* getCRule(char\* b, short nchain);  short getNextChain(GRBALPHABET t, Rule::Chain& pchain, short j);  };  struct Greibach  {  short size{};  GRBALPHABET startN{};  GRBALPHABET stbottomT{};  Rule\* rules{};  Greibach() { short size = 0; startN = 0; stbottomT = 0; rules = 0; };  Greibach(GRBALPHABET pstartN, GRBALPHABET pstbottomT, short psize, Rule r, ...);  short getRule(GRBALPHABET pnn, Rule& prule);  Rule getRule(short n);  };  Greibach getGreibach();} |

Листинг 1 – Структура грамматики Грейбах

|  |
| --- |
| typedef LEX::TableLaI LEXTABLE;  typedef std::stack<short> MFSTSTSTACK;  #define MFST\_DIAGN\_NUMBER 3  #define MFST\_DIAGN\_MAXSIZE 2\*ERROR\_MAXSIZE\_MESSAGE  namespace MFST  {  struct MfstState  {  short lenta\_position;  short nrule;  short nrulechain;  MFSTSTSTACK st;  MfstState();  MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);  MfstState(short pposition, MFSTSTSTACK pst, short pnrule, short pnrulechain);  };  struct Mfst  {  enum RC\_STEP  {  NS\_OK,  NS\_NORULE,  NS\_NORULECHAIN,  NS\_ERROR,  TS\_OK,  TS\_NOK,  LENTA\_END,  SURPRISE  };  struct MfstDiagnosis  {  short lenta\_position;  RC\_STEP rc\_step;  short nrule;  short nrule\_chain;  MfstDiagnosis();  MfstDiagnosis(short plenta\_position, RC\_STEP prc\_step, short pnrule, short pnrule\_chain);  } diagnosis[MFST\_DIAGN\_NUMBER];  GRBALPHABET\* lenta;  short lenta\_position;  short nrule;  short nrulechain;  short lenta\_size;  GRB::Greibach grebach;  LEXTABLE lex;  MFSTSTSTACK st;  std::stack<MfstState> storestate;  Mfst();  Mfst(LEXTABLE plex, GRB::Greibach pgrebach);  char\* getCSt(char\* buf);  char\* getCLenta(char\* buf, short pos, short n = 25);  char\* getDiagnosis(short n, char\* buf);  bool savestate(Log::LOG& log);  bool reststate(Log::LOG& log);  bool push\_chain(GRB::Rule::Chain chain);  RC\_STEP step(Log::LOG& log);  bool start(Log::LOG& log);  bool savediadnosis(RC\_STEP pprc\_step);  void printrules(Log::LOG& log);  struct Deducation  {  short size;  short\* nrules;  short\* nrulechains;  Deducation() { size = 0; nrules = 0; nrulechains = 0; }  } deducation;  bool savededucation();  };  }; |

Листинг 2 – Структура магазинного конечного автомата

|  |  |
| --- | --- |
| 3 : P->(L)  4 : L->ti,L  7 : L->ti  10 : T->{KrD;}  11 : K->ti=E;K  14 : E->i  16 : K->?ZRM  17 : Z->[iOi]  19 : O-><  22 : R->{X}  23 : X->i=E;  25 : E->i  28 : M->!R  29 : R->{X}  30 : X->i=E;  32 : E->i  36 : D->i  39 : S->piPJS  41 : P->(L)  42 : L->ti  45 : J->{Kr;}  46 : K->ti=E;K  49 : E->l  51 : K->cZR  52 : Z->[iOi]  54 : O-><  57 : R->{X}  58 : X->kD;K  59 : D->i  61 : K->kD;K  62 : D->l  64 : K->i=E;  66 : E->iAE  67 : A->+  68 : E->l  74 : S->ftiPTS  77 : P->()  79 : T->{rD;}  81 : D->l  84 : S->m{K};  86 : K->kD;K  87 : D->l  89 : K->ti;K  92 : K->i=E;K  94 : E->iC  95 : C->(N)  96 : N->l,N  98 : N->l  101 : K->nD;K  102 : D->i  104 : K->kD;K  105 : D->l  107 : K->iC;K  108 : C->(N)  109 : N->l  112 : K->nD;K  113 : D->l  115 : K->ti=E;K  118 : E->lAE  119 : A->-  120 : E->l  122 : K->kD;K  123 : D->l  125 : K->nD;K  231 : E->lAE  232 : A->q  233 : E->l  235 : K->kD;K  236 : D->l  238 : K->nD;K  239 : D->i  241 : K->i=E;K  243 : E->l  245 : K->?ZRMK  246 : Z->[D]  247 : D->i  249 : R->{X}  250 : X->nD;  251 : D->l  254 : M->!R | 126 : D->i  128 : K->ti=E;K  131 : E->l  133 : K->ti=E;K  136 : E->l  138 : K->ti=E;K  141 : E->l  143 : K->ti=E;K  146 : E->l  148 : K->kD;K  149 : D->l  151 : K->kD;K  152 : D->i  154 : K->kD;K  155 : D->l  157 : K->kD;K  158 : D->i  160 : K->kD;K  161 : D->l  163 : K->kD;K  164 : D->i  166 : K->kD;K  167 : D->l  169 : K->nD;K  170 : D->i  172 : K->ti=E;K  175 : E->l  177 : K->kD;K  178 : D->l  180 : K->kD;K  181 : D->i  183 : K->kD;K  184 : D->l  186 : K->i=E;K  188 : E->iAE  189 : A->q  190 : E->l  192 : K->nD;K  193 : D->i  195 : K->kD;K  196 : D->l  198 : K->kD;K  199 : D->i  201 : K->kD;K  202 : D->l  204 : K->i=E;K  206 : E->iAE  207 : A->w  208 : E->l  210 : K->nD;K  211 : D->i  213 : K->ti;K  216 : K->i=E;K  218 : E->iAE  219 : A->+  220 : E->iAE  221 : A->/  222 : E->lAE  223 : A->-  224 : E->iCAE  225 : C->(N)  226 : N->l,N  228 : N->l  230 : A->\*  255 : R->{X}  256 : X->nD;  257 : D->l  260 : K->ti=E;K  263 : E->iC  264 : C->()  267 : K->nD;K  268 : D->i  270 : K->i=E;K  272 : E->iC  273 : C->()  276 : K->nD;K  277 : D->i  279 : K->kD;  280 : D->l |

Листинг 3 – Дерево разбора

**Приложение Г**

|  |
| --- |
| namespace POL  {  LT::LexTable lextable2 = LT::Create(LT\_MAXSIZE - 1);  bool PolishNotation(int& lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)  {  bool polGood = true;  char\* str = new char[TI\_STR\_MAXSIZE] {};  int sizeSTR = 0;  int idFunc = 0;  std::stack<LT::Entry> stack;  std::map<char, int> prioryti = { {'(', 0}, {')', 0}, {'+', 1}, {'-', 1}, {'q', 1}, {'w', 1}, {'/', 2}, {'\*', 2} };  while (lextable.table[lextable\_pos].lexemaх[0] != ';')  {  if (lextable.table[lextable\_pos].lexemaх[0] == LEX\_ID || lextable.table[lextable\_pos].lexemaх[0] == LEX\_LITERAL)  {  if (idtable.table[lextable.table[lextable\_pos].idxTI].idtype == IT::IDTYPE::F)  {  lextable2.table[lextable2.size++] = lextable.table[lextable\_pos];  while (lextable.table[++lextable\_pos].lexemaх[0] != ')')  {  lextable2.table[lextable2.size++] = lextable.table[lextable\_pos];  }  lextable2.table[lextable2.size++] = lextable.table[lextable\_pos];  }  else  {  lextable2.table[lextable2.size++] = lextable.table[lextable\_pos];  }  if (lextable.table[lextable\_pos + 1].lexemaх[0] == ';')  {  while (!stack.empty())  {  lextable2.table[lextable2.size++] = stack.top();  stack.pop();  }  }  }  else if (lextable.table[lextable\_pos].lexemaх[0] == '+' || lextable.table[lextable\_pos].lexemaх[0] == '-' ||  lextable.table[lextable\_pos].lexemaх[0] == '\*' || lextable.table[lextable\_pos].lexemaх[0] == '/' ||  lextable.table[lextable\_pos].lexemaх[0] == '(' || lextable.table[lextable\_pos].lexemaх[0] == ')' ||  lextable.table[lextable\_pos].lexemaх[0] == 'w' || lextable.table[lextable\_pos].lexemaх[0] == 'q')  {  if (stack.empty() || stack.top().lexemaх[0] == '(' || lextable.table[lextable\_pos].lexemaх[0] == '(')  {  while (!stack.empty() && stack.top().lexemaх[0] != '(' && lextable.table[lextable\_pos].lexemaх[0] != '(')  {  if (prioryti[lextable.table[lextable\_pos].lexemaх[0]] <= prioryti[stack.\_Get\_container()[stack.size() - 1].lexemaх[0]])  {  lextable2.table[lextable2.size++] = stack.top();  stack.pop();  }  else  {  break;  }  }  stack.push(lextable.table[lextable\_pos]);  }  else if (lextable.table[lextable\_pos].lexemaх[0] == ')')  {  while (!stack.empty() && stack.top().lexemaх[0] != '(')  {  lextable2.table[lextable2.size++] = stack.top();  stack.pop();  }  stack.pop();  }  else if (lextable.table[lextable\_pos + 1].lexemaх[0] == ';')  {  while (!stack.empty())  {  lextable2.table[lextable2.size++] = stack.top();  stack.pop();  }  }  else  {  while (!stack.empty() && stack.top().lexemaх[0] != '(')  {  if (prioryti[lextable.table[lextable\_pos].lexemaх[0]] <= prioryti[stack.\_Get\_container()[stack.size() - 1].lexemaх[0]])  {  lextable2.table[lextable2.size++] = stack.top();  stack.pop();  }  else  {  break;  }  }  stack.push(lextable.table[lextable\_pos]);  }  }  lextable\_pos++;  }  while (!stack.empty())  {  lextable2.table[lextable2.size++] = stack.top();  stack.pop();  }  return true;  delete[]str;  }  bool PolishStart(LT::LexTable& lextable, IT::IdTable& idtable)  {  bool res = true;  for (int i = 0; i < lextable.size; i++)  {  if (lextable.table[i].lexemaх[0] == LEX\_GRAB)  {  lextable2.table[lextable2.size++] = lextable.table[i];  res = PolishNotation(++i, lextable, idtable);  }  lextable2.table[lextable2.size++] = lextable.table[i];  }  lextable.table = lextable2.table;  lextable.size = lextable2.size;  return res;  }  } |

Листинг 4 – Реализация преобразования выражений к обратной польской записи

**Приложение Д**

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib D:\2k1s\KPO\CurseProject\ASMCourse\Debug\StandartLib.lib  ExitProcess PROTO:DWORD  .stack 4096  writenum PROTO : DWORD  writestr PROTO : DWORD  writelnstr PROTO : DWORD  writelnnum PROTO : DWORD  getdata PROTO  gettime PROTO  .const  newline byte 13, 10, 0  L0 sdword 1  L1 byte ' ', 0  L2 byte 'n', 0  L3 byte 'Вызов функции нахождения наименьшего: ', 0  L4 sdword 99  L5 sdword 1000  L6 byte 'Вызов процедуры вывода последовательности чисел: ', 0  L7 sdword 5  L8 sdword 500  L9 sdword 600  L10 byte 'Результат выражения 500 - 600: ', 0  L11 byte 'Н', 0  L12 byte 'Курсовой проект TNS-2022', 0  L13 sdword 12345  L14 dword 1  L15 byte 'Вывод основных типов данных: ', 0  L16 byte 'Сдвиг числа ', 0  L17 byte ' на 3 разряда влево: ', 0  L18 sdword 3  L19 byte ' на 3 разряда вправо: ', 0  L20 sdword 7  L21 byte 'Значение выражения: ', 0  L22 dword 0  L23 byte 'Значение logical true', 0  L24 byte 'Значение logical false', 0  L25 byte 'Тестирование закончилось!', 0  .data  temp sdword ?  buffer byte 256 dup(0)  resultFindMin sdword 0  iPrintNumb sdword 0  resultMAIN sdword 0  exprMAIN sdword 0  test1MAIN dword ?  test2MAIN dword ?  test3MAIN sdword 0  test4MAIN dword 0  test5MAIN sdword 0  test6MAIN sdword 0  timeNowMAIN dword ?  .code  FindMin PROC,  xFindMin : sdword, yFindMin : sdword  push ebx  push edx  push xFindMin  pop ebx  cmp ebx, 0  jnl minus0  neg ebx  minus0:  mov resultFindMin, ebx  mov edx, xFindMin  cmp edx, yFindMin  jl right1  jg wrong1  right1:  push xFindMin  pop ebx  cmp ebx, 0  jnl minus1  neg ebx  minus1:  mov resultFindMin, ebx  jmp next1  wrong1:  push yFindMin  pop ebx  cmp ebx, 0  jnl minus2  neg ebx  minus2:  mov resultFindMin, ebx  next1:  pop edx  pop ebx  mov eax, resultFindMin  ret  FindMin ENDP  PrintNumb PROC,  nPrintNumb : sdword  push ebx  push edx  push L0  pop ebx  cmp ebx, 0  jnl minus3  neg ebx  minus3:  mov iPrintNumb, ebx  mov edx, iPrintNumb  cmp edx, nPrintNumb  jl cycle1  jmp cyclenext1  cycle1:  push iPrintNumb  call writenum  push offset L1  call writestr  push iPrintNumb  push L0  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  cmp ebx, 0  jnl minus4  neg ebx  minus4:  mov iPrintNumb, ebx  mov edx, iPrintNumb  cmp edx, nPrintNumb  jl cycle1  cyclenext1:  pop edx  pop ebx  ret  PrintNumb ENDP  nik PROC  push ebx  push edx  pop edx  pop ebx  mov eax, offset L2  ret  nik ENDP  main PROC  push offset L3  call writestr  push L5  push L4  call FindMin  push eax  pop ebx  cmp ebx, 0  jnl minus5  neg ebx  minus5:  mov resultMAIN, ebx  push resultMAIN  call writelnnum  push offset L6  call writestr  push L7  call PrintNumb  push offset L1  call writelnstr  push L8  push L9  pop ebx  pop eax  sub eax, ebx  push eax  pop ebx  cmp ebx, 0  jnl minus6  neg ebx  minus6:  mov exprMAIN, ebx  push offset L10  call writestr  push exprMAIN  call writelnnum  mov test1MAIN, offset L11  mov test2MAIN, offset L12  push L13  pop ebx  cmp ebx, 0  jnl minus7  neg ebx  minus7:  mov test3MAIN, ebx  push L14  pop ebx  mov test4MAIN, ebx  push offset L15  call writestr  push test1MAIN  call writestr  push offset L1  call writestr  push test2MAIN  call writestr  push offset L1  call writestr  push test3MAIN  call writenum  push offset L1  call writestr  push test4MAIN  call writelnnum  push L0  pop ebx  cmp ebx, 0  jnl minus8  neg ebx  minus8:  mov test5MAIN, ebx  push offset L16  call writestr  push test5MAIN  call writenum  push offset L17  call writestr  push test5MAIN  push L18  pop ebx  pop eax  mov cl, bl  shl eax, cl  push eax  pop ebx  cmp ebx, 0  jnl minus9  neg ebx  minus9:  mov test5MAIN, ebx  push test5MAIN  call writelnnum  push offset L16  call writestr  push test5MAIN  call writenum  push offset L19  call writestr  push test5MAIN  push L18  pop ebx  pop eax  mov cl, bl  shr eax, cl  push eax  pop ebx  cmp ebx, 0  jnl minus10  neg ebx  minus10:  mov test5MAIN, ebx  push test5MAIN  call writelnnum  push test5MAIN  push test3MAIN  push L7  pop ebx  pop eax  cdq  idiv ebx  push eax  pop ebx  pop eax  add eax, ebx  push eax  push L20  push L18  call FindMin  push eax  push L7  pop ebx  pop eax  imul eax, ebx  push eax  pop ebx  pop eax  sub eax, ebx  push eax  push L0  pop ebx  pop eax  mov cl, bl  shl eax, cl  push eax  pop ebx  cmp ebx, 0  jnl minus11  neg ebx  minus11:  mov test6MAIN, ebx  push offset L21  call writestr  push test6MAIN  call writelnnum  push L22  pop ebx  mov test4MAIN, ebx  mov edx, test4MAIN  cmp edx, 1  jz right2  jnz wrong2  right2:  push offset L23  call writelnstr  jmp next2  wrong2:  push offset L24  call writelnstr  next2:  call gettime  mov timeNowMAIN, eax  xor eax, eax  push timeNowMAIN  call writelnstr  call getdata  mov timeNowMAIN, eax  xor eax, eax  push timeNowMAIN  call writelnstr  push offset L25  call writestr  push 0  call ExitProcess  main ENDP  end main |

Листинг 5 – Результат генерации ассемблерного кода