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 IIO-2019»

Выполнил студент Истомин Илья Олегович

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

Заведующий кафедрой доц. Пацей Наталья Владимировна

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

(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 2019

Содержание

[Глава 1. Спецификация языка программирования 6](#_Toc27401724)

[1.1 Характеристика языка программирования 6](#_Toc27401725)

[1.2 Применяемые сепараторы 6](#_Toc27401726)

[1.3 Применяемые кодировки 7](#_Toc27401727)

[1.4 Преобразование типов данных 8](#_Toc27401728)

[1.5 Идентификаторы 8](#_Toc27401729)

[1.6 Литералы 8](#_Toc27401730)

[1.7 Объявления данных и область видимости 9](#_Toc27401731)

[1.8 Операции языка 10](#_Toc27401732)

[1.9 Выражения и их вычисления 10](#_Toc27401733)

[1.10 Программные конструкции языка 10](#_Toc27401734)

[1.11 Область видимости идентификаторов 10](#_Toc27401735)

[1.12 Семантические проверки 11](#_Toc27401736)

[1.13 Стандартная библиотека и её состав 11](#_Toc27401737)

[1.14 Ввод и вывод данных 12](#_Toc27401738)

[1.15 Точка входа 12](#_Toc27401739)

[1.16 Препроцессор 12](#_Toc27401740)

[1.17 Соглашения о вызовах 12](#_Toc27401741)

[1.18 Классификация сообщений транслятора 12](#_Toc27401742)

[1.19 Контрольный пример 13](#_Toc27401743)

[Глава 2. Структура транслятора 14](#_Toc27401744)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc27401745)

[2.2 Перечень входных параметров транслятора 15](#_Toc27401746)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc27401747)

[Глава 3. Разработка лексического анализатора 16](#_Toc27401748)

[3.1 Структура лексического анализатора 16](#_Toc27401749)

[3.2 Контроль входных символов 16](#_Toc27401750)

[3.3 Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем 17](#_Toc27401751)

[3.6 Принцип обработки ошибок 18](#_Toc27401752)

[3.7 Структура и перечень сообщений лексического анализатора 18](#_Toc27401753)

[Глава 4. Разработка синтаксического анализатора 20](#_Toc27401754)

[4.1 Структура синтаксического анализатора. 20](#_Toc27401755)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 20](#_Toc27401756)

[4.3 Построение конечного магазинного автомата 21](#_Toc27401757)

[4.4 Основные структуры данных 22](#_Toc27401758)

[4.5 Описание алгоритма синтаксического разбора 22](#_Toc27401759)

[4.6 Структура и перечень сообщений синтаксического анализатора 22](#_Toc27401760)

[4.7 Параметры синтаксического анализатора и режимы его работы 23](#_Toc27401761)

[4.8 Принцип обработки ошибок 23](#_Toc27401762)

[4.9 Контрольный пример 23](#_Toc27401763)

[Глава 5. Разработка семантического анализатора 24](#_Toc27401764)

[5.1 Структура семантического анализатора 24](#_Toc27401766)

[5.2 Структура и перечень сообщений семантического анализатора 24](#_Toc27401767)

[5.3 Принцип обработки ошибок 25](#_Toc27401768)

[Глава 6. Преобразование выражений 26](#_Toc27401769)

[6.1 Выражения, допускаемые языком 26](#_Toc27401770)

[6.2 Польская запись и принцип ее построения 26](#_Toc27401771)

[6.3 Программная реализация обработки выражений 27](#_Toc27401772)

[6.4 Контрольный пример 27](#_Toc27401773)

[Глава 7. Генерация кода 28](#_Toc27401774)

[7.1 Структура генератора кода 28](#_Toc27401775)

[7.2 Представление типов данных в памяти 28](#_Toc27401776)

[7.3 Статическая библиотека 29](#_Toc27401777)

[7.4 Особенности алгоритма генерации кода 29](#_Toc27401778)

[7.5 Контрольный пример 30](#_Toc27401779)

[Глава 8. Тестирование транслятора 31](#_Toc27401780)

[Заключение 31](#_Toc27401781)

[Приложение А 33](#_Toc27401782)

[Приложение Б 34](#_Toc27401783)

[Приложение Г 37](#_Toc27401784)

[Приложение Д 39](#_Toc27401785)

[Приложение Е 41](#_Toc27401786)

[Литература 45](#_Toc27401787)

**Введение**

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования.

Классический транслятор состоит из следующих частей:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор кода, или интерпретатор.

Все части транслятора, взаимодействуя между собой, обрабатывают входной текст и строят для него эквивалентный текст на понятном компьютеру языке программирования.

# Глава 1. Спецификация языка программирования

* 1. **Характеристика языка программирования**

Язык IIO-2019 ­– это строго типизированный, процедурный, компилируемый язык. Он является транслируемым, не объектно-ориентированным. В IIO-2019 используется два типа данных: целочисленный тип данных num (размер 1 байт), строковыйтип данных str.

**Алфавит языка**

В основе алфавита IIO-2019 лежит таблица символов ACSII. Исходный код IIO-2019 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах.

Таблица 1.1

|  |  |
| --- | --- |
| Название подгруппы | Символы подгруппы |
| Символы латинского алфавита | [a-z] && [A-Z] |
| Символы русского алфавита | [а-я] && [А-Я] |
| Знаковые символы и числовые символы | [‘ - > ] && [ { - } ] && [ [ - ] ] && символ  “ “ ( пробел ) |

* 1. **Применяемые сепараторы**

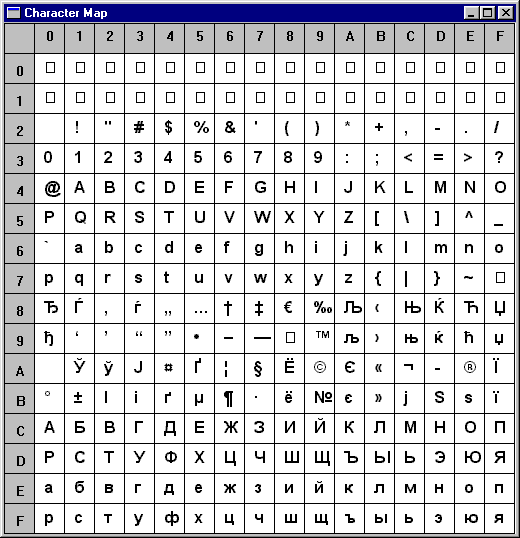
Символы-сепараторы – символы, используемые для разделения отдельных лексических единиц или функциональных элементов в исходном коде программы. Символы, которые являются сепараторами представлены в таблице 1.2.

Таблица 1.2 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ;  “ “ (пробел)  , | Символы-сепараторы для разделения инструкций, параметров функции |
| {  } | Символы-сепараторы указывающие на программный блок |
| (  ) | Символы-сепараторы указывающие на параметры/приоритетность операций(в выражениях) |

* 1. **Применяемые кодировки**

При написании исходного кода на языке программирования используется кодировка ACSII, представленная на рисунке ниже.



**Типы данных**

Пользовательские типы данных не поддерживаются. Допускается использование фундаментальных типов данных определенных в таблице 1.3.

Таблица 1.3 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| num | Является целочисленным типом данных. Этот тип данных занимает 1 байт. Предназначен для арифметических и побитовых операций над числами.  Предусмотрены следующие арифметические и побитовые операции:  + - бинарная операция суммирования (num + num);  - - бинарная операция вычитания (num - num);  \* - бинарная операция умножения (num \* num);  A – побитовая операция И (num A num)  O – побитовая операция ИЛИ (num O num)  N – побитовая операция НЕ (num N num) |
| str | Является строковым типом данных. Предназначен для работы с символами, каждый символ занимает 1 байт.  Операции над данными строкового типа: возможно присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции. Длину строки возможно вычислить с помощью функции стандартной библиотеки strln, а также преобразовать строку в число с помощью функции strton. |

* 1. **Преобразование типов данных**

Преобразование типов в языке IIO-2019 поддерживается. Возможно преобразование строки в число с помощью функции стандартной библиотеки strton.

* 1. **Идентификаторы**

Для именования функций, параметров и переменных используются идентификаторы. Не предусмотрены зарезервированные идентификаторы. Имя идентификатора составляется по следующим образом:

* состоит из символов латинского алфавита [a..z].
* длина имени не менее двух символов
  1. **Литералы**

Предусмотрены числовые (num) и строковые (str) литералы. Правила записи приведены в таблице 1.4.

Таблица 1.4 - Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Числовые | Максимально допустимое значение равно . Минимально допустимое значение равно . Могут состоять только из цифр [0..9]. При выходе за пределы допустимости будет выведена ошибка. Для инициализации отрицательного целочисленного литерала используется префикс 0, который ставится в начале знака -. |
| Строковые | Состоит из символов, заключенных в "…" (двойные кавычки). Максимальное число которых не может превышать . В случае превышения длины литерала работа транслятора прекращается. Объявление строчного литерала реализуется строго в двойных кавычках “ “. |

* 1. **Объявления данных и область видимости**

В языке программирования IIO-2019 переменная должны быть объявлена до ее использования. Областью видимости переменной является блок функции, в которой она определена. Не допустимо объявление глобальных переменных. Область видимости схожа с областью видимости C++, то есть сверху вниз.

**Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. Для объявления отрицательных целочисленных литералов требуется до его записи поставить префикс 0, который ставится в начале знака -. Для инициализации целочисленных литералов, представленных в восьмеричной системе счисления, предусмотрен префикс 0, который ставится после знака равенства.

**Инструкции языка**

В языке программирования IIO-2019 применяются инструкции, представ-ленные в таблице 1.5.

Таблица 1.5 - Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Главная функция | start  {  < <инструкции языка>  } |
| Вызов функций | <идентификатор функции>(<идентификатор / литерал>,…) |
| Возврат из функции | comeback <идентификатор / литерал> |
| Объявление функции | func <тип данных> <идентификатор функции>(<идентификатор / литерал>,…)  {  < <инструкции языка>  } |
| Объявление переменной | rc <тип данных> <идентификатор> |
| Присваивание | num и str:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Вывод данных | read <идентификатор / литерал> |
| Инструкция цикла | cycle (<литерал>)  {  <тело цикла>  }; |

* 1. **Операции языка**

Наибольшую приоритетность арифметических операций имеют операция произведения, а сложение, И, ИЛИ, НЕ, вычитание меньшую. При одинаковом приоритете первой выполнится операция расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

Операции в языке программирования IIO-2019 применимые к целочисленным типам данных приведены в таблице 1.6.

Таблица 1.6 - Операции языка

|  |  |
| --- | --- |
| Тип | Операторы |
| Арифметические | () – приоритетность операций  + ̶ сложение  -  ̶ вычитание  \* ̶ умножение |
| Побитовые | A – И  O – ИЛИ  N – НЕ |

* 1. **Выражения и их вычисления**

Предусмотрены следующие правила составления выражений:

* + выражения читаются слева направо и записываются в одну строку;
  + реализация выражений происходит с помощью обратной польской записи;
  + для изменения приоритета операция используются круглые скобки.
  1. **Программные конструкции языка**

Программные конструкции представлены в таблице 1.7.

Таблица 1.7 - Программные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | start  {  <инструкции языка>  } |
| Функция | func <тип данных> <идентификатор функции>(<идентификатор / литерал>,…)  { <инструкции языка> } |

* 1. **Область видимости идентификаторов**

Все идентификаторы обязаны быть объявленными внутри функции. Глобальных переменных нет, только локальные. Параметры видны только внутри функции, в которой объявлены.

* 1. **Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведен в таблице 1.8.

Таблица 1.8 - Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Проверка на правильность типа параметра comeback |
| 2 | Проверка на дублирование функции main |
| 3 | Проверка на количество параметров объявляемых функций и ограничения применяемые к ним |
| 4 | Проверка на соответствие левой и правой части выражения |
| 5 | Проверка правильности использования функции strln |
| 6 | Проверка правильности использования функции strton |

* 1. **Стандартная библиотека и её состав**

В языке IIO-2019 присутствует стандартная библиотека, которая автоматически подключается при трансляции исходного кода в язык ассемблера. У каждого типа данных есть свои функции реализующие различные команды. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 - Стандартная библиотека языка IIO-2019

|  |  |
| --- | --- |
| Функция | Описание |
| readr(num x) | Выводит на консоль переменную типа num. |
| readw(str x) | Выводит на консоль переменную типа str. |
| strton(str x) | Функция для перевода строки в число. |
| strln(str x) | Функция для определения длины строки x. |

* 1. **Ввод и вывод данных**

Ввод данных не предусмотрен. Вывод данных осуществляется с помощью ключевого слова read. В качестве аргумента принимаются числовые и строковые идентификаторы.

* 1. **Точка входа**

В языке IIO-2019 точкой входа является ключевое слово “start”. Точка входа не может отсутствовать, также не может быть переопределена. В программе может быть только одна точка входа, иначе будет выведена ошибка.

* 1. **Препроцессор**

В языке IIO-2019 препроцессоры не предусмотрены.

* 1. **Соглашения о вызовах**

При генерации кода используется соглашение stdcall. Особенности stdcall:

1. все параметры функций передаются через стек;
2. память высвобождает вызываемый код;
3. занесение в стек параметров идет справа налево.

**Объектный код**

Исходный код языка транслируется в язык ассемблера.

* 1. **Классификация сообщений транслятора**

Транслятор генерирует сообщения о ошибках пользователю. В соответствии с префиксами будут различаться сообщения, представленные в таблице 1.10.

Таблица 1.10 - Классификация сообщений транслятора

|  |  |
| --- | --- |
| Префикс ошибки | Описание ошибки |
| |LA|### | ### - код ошибки. Сообщение, генерируемое на этапе лексического анализа. |
| |SA|### | ### - код ошибки. Сообщение, генерируемое на этапе синтаксического анализа. |
| |SMA|### | ### - код ошибки. Сообщение, генерируемое на этапе семантического анализа. |
| |IN|### | ### - код ошибки. Сообщение, генерируемое при критической ошибке системы. |

* 1. **Контрольный пример**

Контрольный пример, написанный на языке IIO-2019, представлен в приложении А.

**Глава 2. Структура транслятора**

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

Исходный код, написанный на языке программирования IIO-2019, является для транслятора входными данными.

Как выходные данные, используется объектный код и протоколы работы транслятора.

Компоненты транслятора приведены на рисунке 2.1.



Рис 2.1 - Структура транслятора IIO-2019

Первоначально на вход лексического анализатора передается исходный код. Анализатором проверяется исходный текст на недопустимые символы, выделяет литералы, идентификаторы и ключевые слова, а также формирует таблицы лексем и идентификаторов.

Далее наступает черед синтаксического анализатора, к нему на вход поступа-ет таблица лексем, полученная на этапе лексического анализа. Если программа по-строена синтаксически правильно, то осуществляется переход к этапу трансляции стоящему далее, при ином раскладе работа транслятора останавливается.

Наборы функций, проверяющие правила на разных этапах работы транслято-ра представлены в семантическом анализаторе. Продолжение или остановка работы транслятора всецело зависит от критичности возникающих ошибок.

Генерация кода реализуется посредством чистой интерпретации, без создания промежуточного представления кода. В финале происходит генерация кода, во время исполнения которого формируется объектный код.

**2.2 Перечень входных параметров транслятора**

В таблице 2.1 представлены входные параметры, которые могут использоваться для представления работы транслятора.

Таблица 2.1 - Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Назначение | Тип |
| -in: | Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением \*.txt | Обязательный |
| -out: | Указывает имя протокола. Если не указан явно, то имя протокола не формируется. | Не обязательный |
| -log: | Указывает имя протокола. Если не указан явно, то имя протокола формируется конкатенацией имени файла исходного кода и постфикса «.log» | Не обязательный |

**2.3 Перечень протоколов, формируемых транслятором и их содержимое**

По итогам своей работы транслятор формирует протокол, согласно заданным входным параметрам. –log файл- в этом файле находятся информация о

времени выполнения компиляции, количестве символов и выходных параметрах.

–out файл - в этом файле находятся содержимое таблицы лексем, содержимое таблицы идентификаторов в соответствии с её первоначальным представлением. Остальные компоненты (работа синтаксического анализатора, дерево разбора) выводятся напрямую на консоль.

# Глава 3. Разработка лексического анализатора

**3.1 Структура лексического анализатора**

Структура лексического анализатора представлена на рисунке 3.1.



Рис 3.1 - Структура лексического анализатора

Исходный код на языке IIO-2019 является входными данными;

Таблицы лексем и идентификаторов являются выходными данными;

**3.2 Контроль входных символов**

Таблица допустимости представлена на рисунке 3.2.

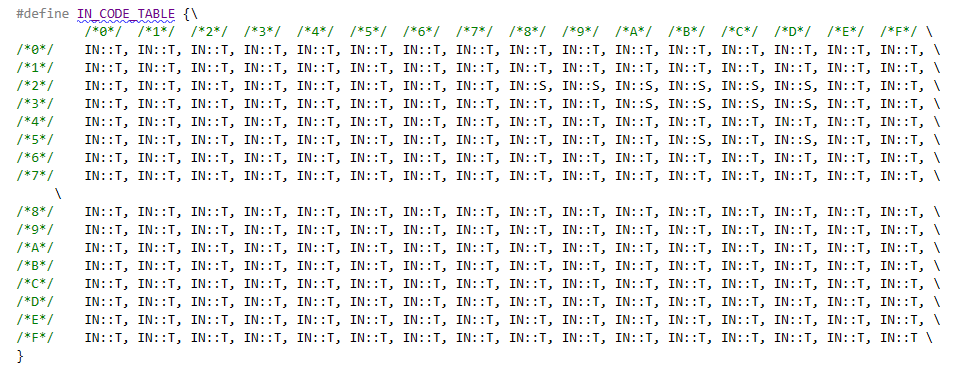


Рисунок 3.2 - Таблица допустимости входных символов

Таблица допустимости была сформирована на основе кодировки ASCII. Таблица необходима для проверки входных символов на допустимость. Символы могут быть разрешенными, запрещенными, игнорируемыми.

«T» - разрешенные алфавитом символы;

«F» - запрещенные алфавитом символы;

«I» - символы, которые игнорируются;

«S» - сепараторы.

## 3.3 Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем

Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем представлен в таблице 3.1.

Таблица 3.1 - Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| num | n |
| str | s |
| fun | f |
| read | w |
| rc | r |
| start | m |
| comeback | c |
| + | v |
| - | v |
| \* | v |
| A | v |
| O | v |
| N | ~ |
| / | v |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ; | ; |
| идентификатор | i |
| числовой литерал | l |
| строковый литерал | l |
| strton | j |
| strln | k |
| cycle | b |

**3.4 Основные структуры данных**

В приложении Б представлены основные структуры данных на этапе синтаксического анализа.

**3.6 Принцип обработки ошибок**

В случае обнаружения критической ошибки, которая не позволяет работать анализаторам или генератору правильно функционировать, транслятор прекращает свою работу и в log-файл записывается ошибка. Подсчет количества ошибок не ведется.

## 3.7 Структура и перечень сообщений лексического анализатора

Префикс сообщений “|LA|”. Перечень сообщений, генерируемых на этапе лексического анализа, представлен в таблице 3.2.

Таблица 3.2 - Сообщения лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 120 | Цепочка символов не разобрана |
| 121 | Таблица лексем переполнена |
| 122 | Таблица идентификаторов переполнена |
| 123 | Дублирование идентификатора |
| 124 | Дублирование арифметических операций |
| 125 | Превышение допустимого значения типа |

**3.8 Параметры лексического анализатора и режим его работы**

Текст кода на языке IIO-2019 подается на вход. Параметры не определяют режим работы лексического анализатора.

**3.9 Контрольный пример**

На вход лексического анализатора подается программа на языке IIO-2019. Результат работы лексического анализатора представлен в приложении В.

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора.

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 - Структура синтаксического анализатора

Таблицы лексем и идентификаторов являются входными данными.

Дерево разбора является выходными данными.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика, описывающая язык IIO-2019 представлена в таблице 4.1.

Таблица 4.1 - грамматика языка IIO-2019

|  |  |
| --- | --- |
| Нетереминалы | Описание |
| S –> | Правила, описывающие общую структуру программы |
| N –> | Порождает правила, описывающие инструкции языка |
| M –> | Порождает правила, описывающие выражения |
| F –> | Порождает правила, описывающие формальные параметры функции |
| E –> | Порождает правила, описывающие подвыражения |
| W –> | Порождает правила, описывающие параметры вызываемой функции |

## 4.3 Построение конечного магазинного автомата

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.



Рис. 4.2 - МП-автомат

Формальное описание МП-автомата:



 - множество состояний;

 - алфавит входных символов;

 - специальный алфавит магазинных символов;

-функция переходов автомата , где  - множество подмножеств ;

 - начальное состояние автомата;

- начальное состояние магазина (маркер дна);

- множество конечных состояний.

Конфигурация (текущее состояние автомата) описывается тройкой , где - текущее состояние автомата, - остаток цепочки,  - цепочка-содержимое магазина.

Начальное состояние , - начальное состояние автомата,  - входная цепочка, - маркер дна магазина.

Цепочка  является допустимой (распознается) автоматом , если  и .

Работа автомата 

1. состояние автомата 
2. читает символ  находящийся под головкой (сдвигает ленту);
3. не читает ничего (читает , не сдвигает ленту);
4. из  определяет новое состояние , если  или .
5. читает верхний (в стеке) символ  и записывает цепочку  т.к. , при этом, если , то верхний символ магазина просто удаляется.

работа автомата заканчивается 

## 4.4 Основные структуры данных

В приложении Г представлены основные структуры данных и правила пере-хода, используемые на этапе синтаксического анализа.

## 4.5 Описание алгоритма синтаксического разбора

Входные символы и лексемы в форме Грейбах находятся в ленте на входе конечного автомата.

1) Если лента не пустая, переходим далее следующему пункту, иначе переходим к пункту 5.

2) Если на верхушке магазина нетерминальный символ.

2.1) Если есть такое правило, то переходим к следующему пункту.

2.1.1) Если цепочка есть, возвращаем NS\_OK. Переходим к пункту 4.

2.1.2) Иначе восстанавливаем состояние. Переходим к пункту 4.

2.2) Иначе возвращаем ошибку. Переход к пункту 4.

3) Если на верхушке терминал и он совпадает с символом на ленте, то удаляем его из стека и продвигаем ленту. Переход к пункту 4.

4) Повторяем шаг, переходим к пункту 1.

5) Конец работы.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Префикс сообщений “|SA|”. Перечень сообщений, генерируемых на этапе синтаксического анализа, представлен в таблице 4.3.

Таблица 4.3 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибочный оператор |
| 602 | Неверное выражение |
| 603 | Ошибка в параметрах функции или операторе объявления |
| 604 | Ошибка в параметре вызываемой функции strton стандартной библиотеки |
| 605 | Блок содержит неверное подвыражение |
| 606 | Ошибка в параметре вызываемой функции strl стандартной библиотеки |
| 607 | Ошибка в параметрах вызываемой функции |
| 609 | Неверная структура программы. В языке IIO-2019 не предусмотрено объявление переменной вне функции |
| 613 | Точка входа в программу не задана |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Таблицы идентификаторов и лексем являются входными параметрами для синтаксического анализатора. Эти таблицы мы получаем в ходе лексического анализа. В конце, после разбора формируется дерево разбора, которое выводится на консоль.

## 4.8 Принцип обработки ошибок

При обнаружении ошибки в цепочке какого-либо правила, синтаксического анализатора идет вверх по дереву разбора, пока не найдет верный вариант. Иначе запоминается самая глубокая ошибка, которая выводится в протокол работы.

## 4.9 Контрольный пример

Результатом работы синтаксического анализатора является дерево разбора. Результат работы синтаксического анализатора представлен в приложении Д.

# Глава 5. Разработка семантического анализатора

1. 1. **Структура семантического анализатора**

Семантический анализатор состоит из набора функций для проверки правильности исходной программы. Функции анализатора выполняются на различных этапах работы транслятора. Структура семантического анализатора представлена на рисунке 5.1.

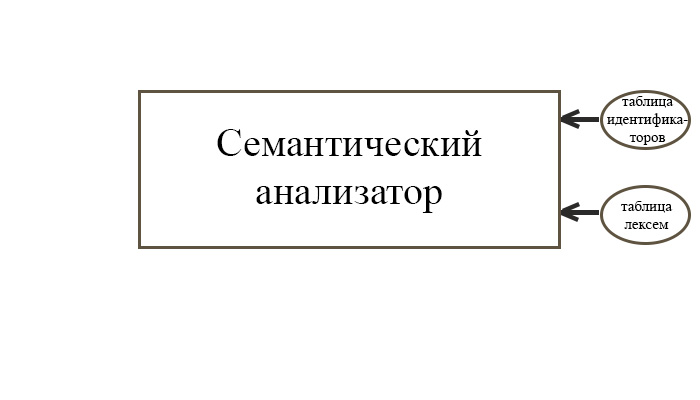


Рисунок 5.1 - Структура семантического анализатора

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

Префикс сообщений “|SMA|”. Сообщения, генерируемые при выполнении семантических проверок, представлены в таблице 5.1.

Таблица 5.1 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 700 | Повторное объявление идентификатора |
| 701 | Ошибка в типе идентификатора |
| 702 | Ошибка в передаваемых значениях в функцию |
| 703 | В функцию не переданы параметры |
| 704 | Тип данных результата выражения не соответствует присваиваемому идентификатору |
| 705 | Ошибка в параметре вызываемой функции strton стандартной библиотеки |
| 706 | Необъявленный идентификатор |
| 707 | Несоответствие типов в операторе присваивания |
| 708 | Неверная структура программы |
| 709 | Превышен максимальный размер идентификатора |
| 710 | Повторное объявление точки входа |

* 1. **Принцип обработки ошибок**

Все семантические ошибки являются критическими, из-за чего транслятор прекращает свою работу и в протокол работы транслятора выводится соответствующее сообщение об ошибке.

**Глава 6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

В языке IIO-2019 допускаются выражения с использованием числовых идентификаторов и литералов. Также предусмотрены следующие арифметические операции:

* сложения: «+» ;
* вычитания: «-» ;
* умножения: «\*» ;
* И: «A»;
* ИЛИ: «O»;
* НЕ: «N».

Приоритетность операций представлена в таблице 6.1. Чем ниже число, тем выше и приоритет.

Таблица 6.1 - Приоритетность операций

|  |  |
| --- | --- |
| Операция | Приоритет |
| «\*» | 1 |
| «+», «-», | 2 |
| «A» | 10 |
| «N» | 11 |
| «O» | 12 |

**6.2 Польская запись и принцип ее построения**

Польская запись -форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Приоритетность операций приведена в таблице 6.1. Известен следующий принцип 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**

После этапов лексического и синтаксического анализа происходит преобразование в польскую запись, во время этапа генерации кода на язык ассемблера.

**6.4 Контрольный пример**

Контрольный пример разбора выражения содержится в таблице 6.2.

Таблица 6.2 - Разбор выражения

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| 10+4\*2/(1-5) |  | 10 |
| +4\*2/(1-5) | + | 10 |
| 4\*2/(1-5) | + | 104 |
| \*2/(1-5) | +\* | 104 |
| 2/(1-5) | +\* | 1042 |
| /(1-5) | +/ | 1042\* |
| (1-5) | +/( | 1042\* |
| 1-5) | +/( | 1042\*1 |
| -5) | +/(- | 1042\*1 |
| 5) | +/(- | 1042\*15 |
| ) | +/ | 1042\*15- |

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Заключительным этапом трансляции языка IIO-2019 является генерация кода. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. Результатом работы транслятора будет выходной файл на языке ассемблера. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода IIO-2019 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в памяти

Элементы таблицы идентификаторов расположены в сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке IIO-2019 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка IIO-2019 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке IIO-2019 | Тип идентификатора на языке ассемблера | Пояснение |
| num | DWORD | Хранит целочисленный тип данных. |
| str | BYTE | Хранит строковый тип данных |

## 

## 7.3 Статическая библиотека

В языке IIO-2019 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Эти функции представлены в таблице 1.9.

Таблица 1.9 - Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на ассемблерном языке | Описание |
| readr PROC | Функции для вывода на консоль результата выполнения программы типа num. |
| readw PROC | Функции для вывода на консоль результата выполнения программы типа str. |
| strln PROC | Функция для вывода длины строки |
| strton PROC | Функция для преобразования строки в число |

## 7.4 Особенности алгоритма генерации кода

В языке IIO-2019 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

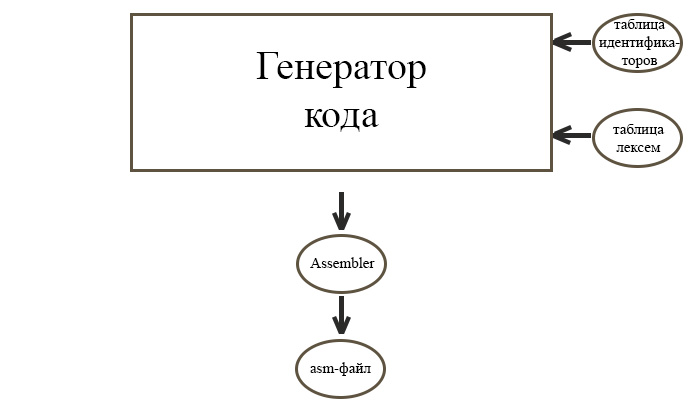


Рисунок 7.2 - Структура генератора кода

## 7.5 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Е.

# Глава 8. Тестирование транслятора

В данной главе описаны возможные ошибки, возникающие на различных этапах работы транслятора. Результат тестирования представлен в таблице 8.1.

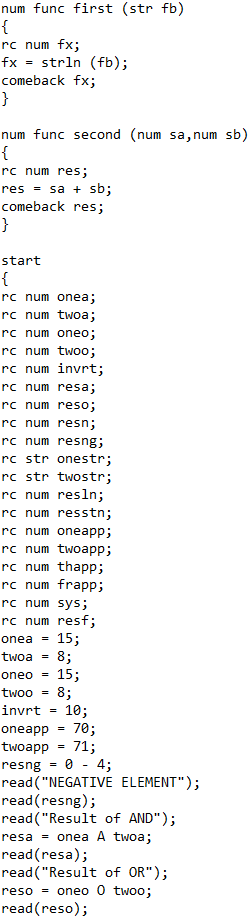
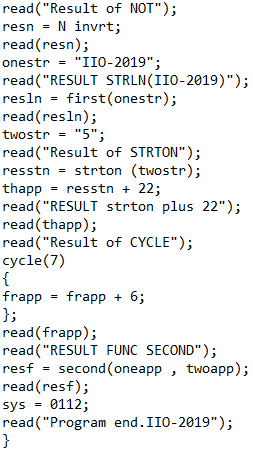
Таблица 8.1 - Тестирование транслятора

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| num func first(num sa)  {  …  }  start  {  …  }  start  {  } | Ошибка 710: |SMA|: Повторное объявление точки входа |
| num func first(num sa)  {  …  } | Ошибка 613: |SA|: Точка входа в программу не задана |
| num func first(num sa)  {  …  }  num func first(num sa)  {  …  } | Ошибка 700: |SMA|: Повторное объявление идентификатора |
| num func first(num sa)  {  rc str se;  comeback se;  } | Ошибка 701: |SMA|: Ошибка в типе идентификатора |

# Заключение

* Реализованы 6 арифметических оператора для вычисления выражений;
* Реализованы два типа данных;
* Поддерживается оператор вывода;
* Присутствует подключаемая стандартная библиотека;
* Обрабатывается более 20 возможных исключительных ситуаций;
* Реализованы две функции работы со строкой
* Реализован оператор цикла
* Присутствует сложение целочисленных литералов, представленных в разных системах счисления

# Приложение А



# Приложение Б

namespace LT

{ // таблица лексем

struct Entry

{

char lexema; // лексема

int lineNo; // номер строки в исходном коде

int indexTI; // индекс в таблице идентификаторов

short priority; // приоритет для операций

};

struct LexTable

{

int maxize; // емкость таблицы лексем

int size; // текущий размер таблицы лексем

Entry\* table; // массив строк табилцы лексем

};

LexTable Create(int size);

Entry GetEntry(LexTable& lextable, int n);

Entry Add(LexTable& lextable, Entry entry);

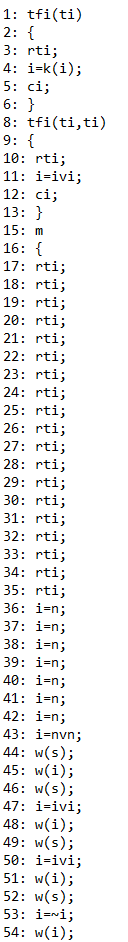
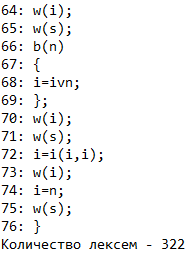
void ShowTable(LexTable& lexTable);

void Write(Log::LOG log, LexTable& lexTable);

}

Рис. 9 - Структуры данных, используемые на фазе лексического анализа

**Приложение В**



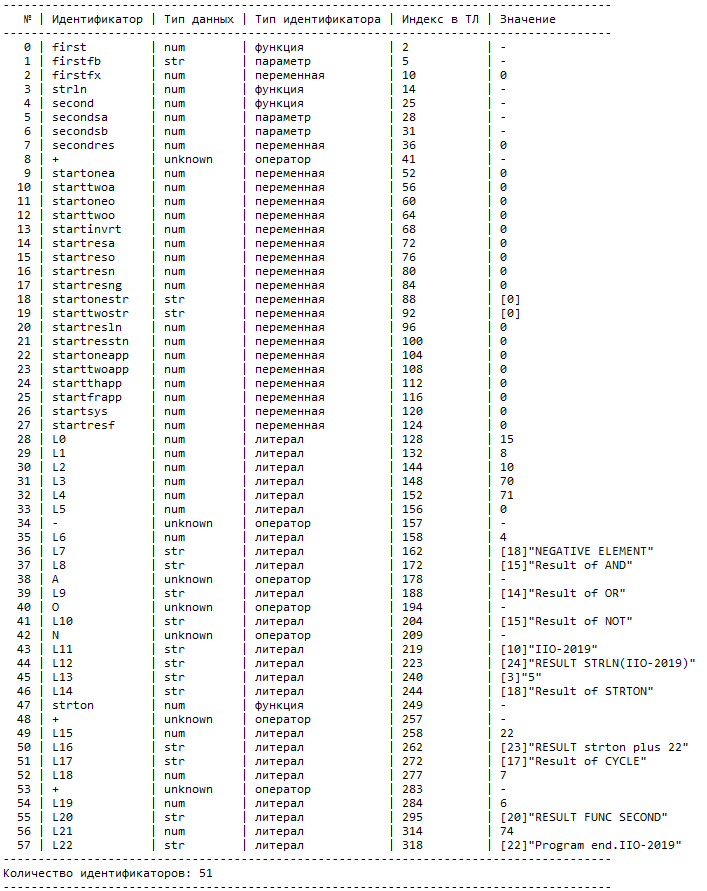


Рис. 11 - Таблица идентификаторов

# Приложение Г

Greibach greibach(

NS('S'), TS('$'), // стартовый символ, дно стека NS-нетерминал(большие буквы),TS-терминал

6, // количество правил

Rule(

NS('S'), GRB\_ERROR\_SERIES + 0, // неверная структура программы

4,

Rule::Chain(10, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}'), NS('S')),

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}')),

Rule::Chain(9, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}')),

Rule::Chain(5, TS('m'), TS('{'), NS('N'), TS('}'), NS('S'))

),

Rule(

NS('N'), GRB\_ERROR\_SERIES + 1, // ошибочный оператор

13,

Rule::Chain(5, TS('r'), TS('t'), TS('i'), TS(';'), NS('N')),

Rule::Chain(4, TS('r'), TS('t'), TS('i'), TS(';')),

Rule::Chain(3, TS('c'), NS('E'), TS(';')),

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(6, TS('w'), TS('('), TS('i'), TS(')'), TS(';'), NS('N')),

Rule::Chain(6, TS('w'), TS('('), TS('s'), TS(')'), TS(';'), NS('N')),

Rule::Chain(6, TS('w'), TS('('), TS('n'), TS(')'), TS(';'), NS('N')),

Rule::Chain(5, TS('w'), TS('('), TS('s'), TS(')'), TS(';')),

Rule::Chain(5, TS('w'), TS('('), TS('i'), TS(')'), TS(';')),

Rule::Chain(5, TS('w'), TS('('), TS('n'), TS(')'), TS(';')),

Rule::Chain(9, TS('b'), TS('('), TS('n'), TS(')'), TS('{'), NS('N'), TS('}'),TS(';'), NS('N')),

Rule::Chain(9, TS('b'), TS('('), TS('i'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N'))

),

Rule(

NS('E'), GRB\_ERROR\_SERIES + 2, // ошибка в выражении

16,

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('s')),

Rule::Chain(1, TS('n')),

Rule::Chain(2,TS('~'),TS('n')),

Rule::Chain(2, TS('~'), TS('i')),

Rule::Chain(3, TS('('), NS('E'), TS(')')),

Rule::Chain(2, TS('n'), NS('M')),

Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('r'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(4, TS('j'), TS('('), TS('i'), TS(')')), // strton

Rule::Chain(4, TS('j'), TS('('), TS('s'), TS(')')),

Rule::Chain(4, TS('k'), TS('('), TS('i'), TS(')')), //strln

Rule::Chain(4, TS('k'), TS('('), TS('s'), TS(')'))

),

Rule(

NS('F'), GRB\_ERROR\_SERIES + 3, // ошибка в параметрах функции

2,

Rule::Chain(2, TS('t'), TS('i')),

Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))

),

Rule(

NS('W'), GRB\_ERROR\_SERIES + 4, // ошибка в параметрах вызываемой функции

6,

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('s')),

Rule::Chain(1, TS('n')),

Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('s'), TS(','), NS('W')),

Rule::Chain(3, TS('n'), TS(','), NS('W'))

),

Rule(

NS('M'), GRB\_ERROR\_SERIES + 5, // оператор

3,

Rule::Chain(1, TS('v')),

Rule::Chain(2, TS('v'), NS('E')),

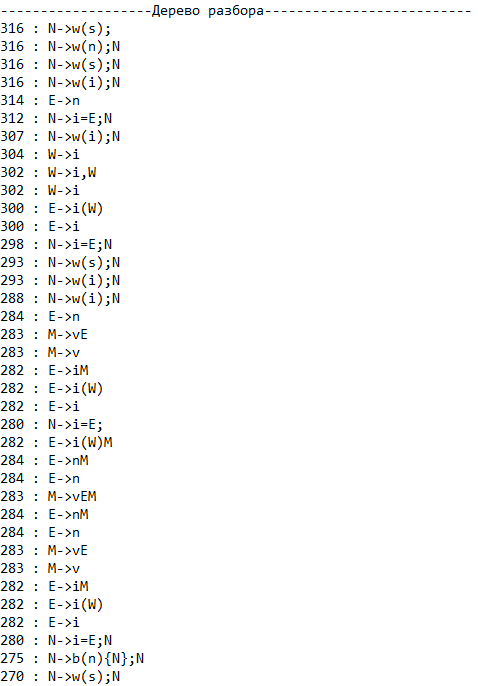
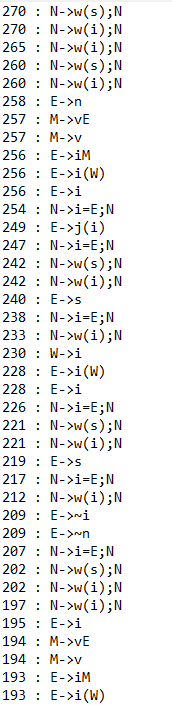
Rule::Chain(3, TS('v'), NS('E'), NS('M'))

)

);

Рис.11 - Структура данных грамматики Грейбах

# Приложение Д



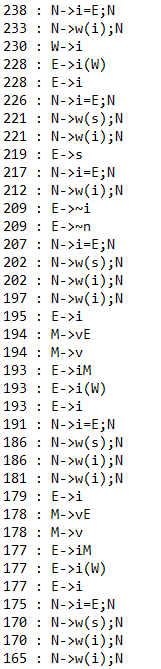
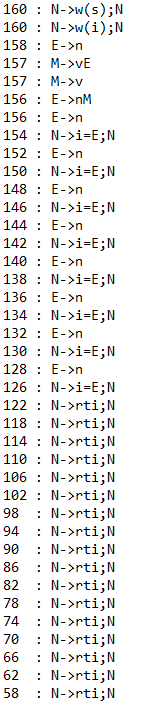
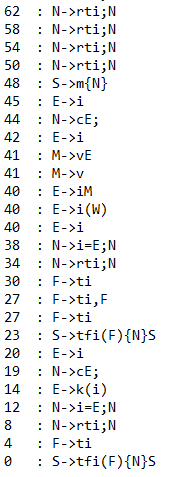


Рис. 12 - Дерево разбора

## Приложение Е

Таблица 1 – Результат генерации кода

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib "D:\Visual Studio 2019\YAP\IIO\IIO-2019\Debug\Lib.lib"  ExitProcess PROTO :DWORD  readw PROTO: DWORD  readr PROTO: DWORD  strln PROTO: DWORD  strton PROTO: BYTE  .stack 4096  .const  L0 DWORD 15  L1 DWORD 8  L2 DWORD 10  L3 DWORD 70  L4 DWORD 71  L5 DWORD 0  L6 DWORD 4  L7 BYTE "NEGATIVE ELEMENT", 0  L8 BYTE "Result of AND", 0  L9 BYTE "Result of OR", 0  L10 BYTE "Result of NOT", 0  L11 BYTE "IIO-2019", 0  L12 BYTE "RESULT STRLN(IIO-2019)", 0  L13 BYTE "5", 0  L14 BYTE "Result of STRTON", 0  L15 DWORD 22  L16 BYTE "RESULT strton plus 22", 0  L17 BYTE "Result of CYCLE", 0  L18 DWORD 7  L19 DWORD 6  L20 BYTE "RESULT FUNC SECOND", 0  L21 DWORD 74  L22 BYTE "Program end.IIO-2019", 0  .data  firstfx DWORD ?  secondres DWORD ?  startonea DWORD ?  starttwoa DWORD ?  startoneo DWORD ?  starttwoo DWORD ?  startinvrt DWORD ?  startresa DWORD ?  startreso DWORD ?  startresn DWORD ?  startresng DWORD ?  startonestr DWORD ?  starttwostr DWORD ?  startresln DWORD ?  startresstn DWORD ?  startoneapp DWORD ?  starttwoapp DWORD ?  startthapp DWORD ?  startfrapp DWORD ?  startsys DWORD ?  startresf DWORD ?  .code  first PROC firstfb : DWORD  push firstfb  pop edx  push firstfb  call strln  push eax  pop firstfx  push firstfx  ret  first ENDP  second PROC secondsa : SDWORD, secondsb : SDWORD  push secondsa  push secondsb  pop eax  pop ebx  add eax, ebx  push eax  pop secondres  push secondres  ret  second ENDP  main PROC  push L0  pop startonea  push L1  pop starttwoa  push L0  pop startoneo  push L1  pop starttwoo  push L2  pop startinvrt  push L3  pop startoneapp  push L4  pop starttwoapp  push L5  push L6  pop ebx  pop eax  sub eax, ebx  push eax  pop startresng  push offset L7  call readw  push startresng  call readr  push offset L8  call readw  push startonea  push starttwoa  pop eax  pop ebx  and eax, ebx  push eax  pop startresa  push startresa  call readr  push offset L9  call readw  push startoneo  push starttwoo  pop eax  pop ebx  or eax, ebx  push eax  pop startreso  push startreso  call readr  push offset L10  call readw  push startinvrt  pop startresn  push startresn  call readr  push offset L11  pop startonestr  push offset L12  call readw  push startonestr  pop edx  push startonestr  call first  push eax  pop startresln  push startresln  call readr  push offset L13  pop starttwostr  push offset L14  call readw  push starttwostr  pop edx  push starttwostr  call strton  push eax  pop startresstn  push startresstn  push L15  pop eax  pop ebx  add eax, ebx  push eax  pop startthapp  push offset L16  call readw  push startthapp  call readr  push offset L17  call readw  mov ecx, L18  Mark:  push startfrapp  push L19  pop eax  pop ebx  add eax, ebx  push eax  pop startfrapp  push startfrapp  loop Mark  push startfrapp  call readr  push offset L20  call readw  push startoneapp  push starttwoapp  pop edx  pop edx  push starttwoapp  push startoneapp  call second  push eax  pop startresf  push startresf  call readr  push L21  pop startsys  push offset L22  call readw  push 0  call ExitProcess  main ENDP  end main |

# Литература

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

2. Смелов, В.В. Курс лекций по предмету языки программирования – 2016

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

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

5. Малявко, А.А. Формальные языки и компиляторы. Учебное пособие / А. Малявко ­­– 2017 – 432 с.

6. Гагарина Л.Г. Введение в теорию алгоритмических языков и компиляторов / Л. Гагарина – 2009 – 176 с.