**Содержание**

[Введение 5](#_Toc122372122)

[Раздел 1. Спецификация языка программирования 6](#_Toc122372123)

[1.1 Характеристика языка программирования 6](#_Toc122372124)

[1.2 Алфавит языка программирования 6](#_Toc122372125)

[1.3 Применяемые сепараторы 6](#_Toc122372126)

[1.4 Применяемые кодировки 6](#_Toc122372127)

[1.5 Типы данных 7](#_Toc122372128)

[1.6 Преобразование типов данных 8](#_Toc122372129)

[1.7 Идентификаторы 8](#_Toc122372130)

[1.8 Литералы 8](#_Toc122372131)

[1.9 Объявление данных 9](#_Toc122372132)

[1.10 Инициализация данных 9](#_Toc122372133)

[1.11 Инструкции языка 9](#_Toc122372134)

[1.12 Операции языка 9](#_Toc122372135)

[1.13 Вычисление выражений 10](#_Toc122372136)

[1.14 Конструкции языка 10](#_Toc122372137)

[1.15 Области видимости идентификаторов 11](#_Toc122372138)

[1.16 Семантические проверки 11](#_Toc122372139)

[1.17 Распределение памяти 11](#_Toc122372140)

[1.18 Стандартная библиотека 11](#_Toc122372141)

[1.19 Ввод и вывод данных 12](#_Toc122372142)

[1.20 Точка входа 12](#_Toc122372143)

[1.21 Препроцессор 12](#_Toc122372144)

[1.22 Соглашение о вызовах 12](#_Toc122372145)

[1.23 Объектный код 12](#_Toc122372146)

[1.24 Классификация сообщений транслятора 12](#_Toc122372147)

[1.25 Контрольный пример 13](#_Toc122372148)

[Раздел 2. Структура транслятора 14](#_Toc122372149)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc122372150)

[2.2 Перечень входных параметров транслятора 14](#_Toc122372151)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc122372152)

[Раздел 3. Разработка лексического анализатора 16](#_Toc122372153)

[3.1 Структура лексического анализатора 16](#_Toc122372154)

[3.2 Контроль входных символов 16](#_Toc122372155)

[3.3 Удаление избыточных символов 17](#_Toc122372156)

[3.4 Перечень ключевых слов 17](#_Toc122372157)

[3.5 Основные структуры данных 18](#_Toc122372158)

[3.6 Принцип обработки ошибок 18](#_Toc122372159)

[3.7 Структура и перечень сообщений лексического анализа 18](#_Toc122372160)

[3.8 Параметры лексического анализатора 18](#_Toc122372161)

[3.9 Алгоритм лексического анализа 18](#_Toc122372162)

[3.10 Контрольный пример 19](#_Toc122372163)

[Раздел 4. Разработка синтаксического анализатора 20](#_Toc122372164)

[4.1 Структура синтаксического анализатора 20](#_Toc122372165)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис 20](#_Toc122372166)

[4.3 Построение конечного магазинного автомата 22](#_Toc122372167)

[4.4 Основные структуры данных 23](#_Toc122372168)

[4.5 Описание алгоритма синтаксического разбора 23](#_Toc122372169)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc122372170)

[4.7. Параметры синтаксического анализатора и режимы его работы 24](#_Toc122372171)

[4.8. Принцип обработки ошибок 24](#_Toc122372172)

[4.9. Контрольный пример 24](#_Toc122372173)

[Раздел 5. Разработка семантического анализатора 25](#_Toc122372174)

[5.1 Структура семантического анализатора 25](#_Toc122372175)

[5.2 Функции семантического анализатора 25](#_Toc122372176)

[5.3 Структура и перечень семантических ошибок 25](#_Toc122372177)

[5.4 Принцип обработки ошибок 25](#_Toc122372178)

[5.5 Контрольный пример 25](#_Toc122372179)

[Раздел 6. Вычисление выражений 27](#_Toc122372180)

[6.1 Выражения, допускаемые языком 27](#_Toc122372181)

[6.2 Польская запись и принцип ее построения 27](#_Toc122372182)

[6.3 Программная реализация обработки выражений 27](#_Toc122372183)

[6.4 Контрольный пример 27](#_Toc122372184)

[Раздел 7. Генерация кода 29](#_Toc122372185)

[7.1 Структура генератора кода 29](#_Toc122372186)

[7.2 Представление типов данных в оперативной памяти 29](#_Toc122372187)

[7.3 Статическая библиотека 29](#_Toc122372188)

[7.4 Особенности алгоритма генерации кода 30](#_Toc122372189)

[7.5 Входные параметры генератора кода 30](#_Toc122372190)

[7.6 Контрольный пример 30](#_Toc122372191)

[Раздел 8. Тестирование транслятора 31](#_Toc122372192)

[8.1 Общие положения 31](#_Toc122372193)

[8.2 Результаты тестирования 31](#_Toc122372194)

[Заключение 33](#_Toc122372195)

[Список использованных источников 34](#_Toc122372196)

[Приложение А 35](#_Toc122372197)

[Приложение Б 36](#_Toc122372198)

[Приложение В 39](#_Toc122372199)

[Приложение Г 43](#_Toc122372200)

[Приложение Д 46](#_Toc122372201)

# Введение

Основной целью данной курсовой работы является разработка транслятора для языка программирования YNS-2022. Главная задача транслятора заключается в переводе исходного кода на языке YNS-2022 в код на языке Assembler. Язык программирования YNS-2022 предназначен для выполнения простейших арифметический действий и операций над строками.

Классический транслятор состоит из следующих частей:

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

Все части транслятора, взаимодействуя с собой, обрабатывают входной текст и строят эквивалентный ему текст на понятном компьютеру языке.

# Раздел 1. Спецификация языка программирования

## 1.1 Характеристика языка программирования

YNS-2022 является компилируемым процедурным языком высокого уровня. Он компилируемый, не объектно-ориентируемый. В языке поддерживается 2 типа данных: строковый (line) и целочисленный (shint). В стандартной библиотеке языка содержится 2 функция для работы со строками: CopyLine (копирование строк) и LengthLine (вычисление длины строки).

## 1.2 Алфавит языка программирования

Символы, которые используются на этапе выполнения: [a…z], [A…Z], [0…9], [a…я], символ пробела, табуляции, перевода строки и спецсимволы: () [] , ; : # + - / \* % < > !.

## 1.3 Применяемые сепараторы

Символы-сепараторы необходимы для эффективного разбиения исходного текста программы на лексемы на этапе лексического анализа. В таблице 1.1 приведены сепараторы и их назначение.

Таблица 1.1 Символы - сепараторы

|  |  |
| --- | --- |
| Символы | Назначение |
| […] | Ограничение тела функции или цикла. |
| (…) | Повышение приоритета арифметических операций, ограничение блока параметров функции. |
| «пробел» | Разделитель цепочек языка, не допустим и именах идентификаторов и ключевых словах языка. |
| , | Разделитель параметров функции. |
| = | Оператор присваивания. |
| + - \* / mod | Арифметические операции. |
| ; | Разделитель программных конструкций. |
| < > ! | Логические операторы (Операции сравнения: меньше, больше, не равно). |
| # | Символ, отделяющий цикл. |

## 1.4 Применяемые кодировки

Для написания программ на языке YNS-2022 применяется кодировка Windows-1251, которая является стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10 версии.

Таблица кодировки представлена на рисунке 1.1.

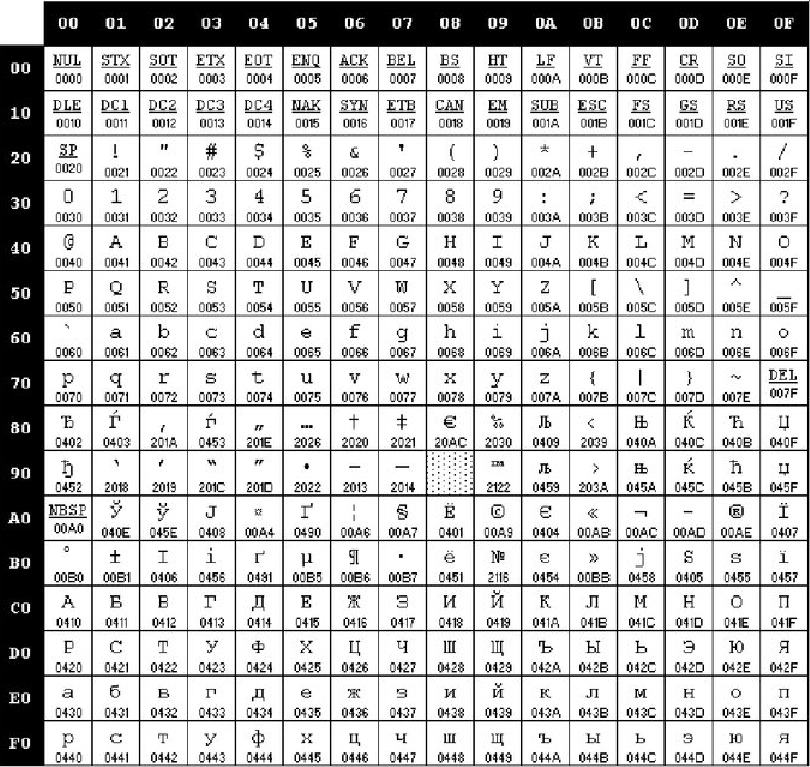


Рисунок 1.1 Алфавит входных символов

## 1.5 Типы данных

В языке YNS-2022 реализованы 2 фундаментальных типа данных: целочисленный и строковый. Их описание приведено в таблице 1.2.

Таблица 1.2. Описание типов данных языка YNS-2022

|  |  |
| --- | --- |
| Тип данных | Описание |
| Целочисленный (shint) | Фундаментальный тип данных, занимающий в памяти 4 байта. Используется для работы с целыми числами. По умолчанию имеет значение: 0  Поддерживаемые операции:  + (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  / (бинарный) – оператор деления;  = (бинарный) – оператор присваивания;  % (бинарный) – оператор деления по модулю; |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Описание |
|  | В качестве операторов условия цикла поддерживаются следующие операторы:  > (бинарный) – оператор “больше”;  < (бинарный) – оператор “меньше”;  ! (бинарный) – оператор “не равно”; |
| Строковый (line) | Фундаментальный тип данных, каждый символ занимает 1 байт в памяти. Максимальная длина строки – 255 символов. Используется для работы со строками. По умолчанию имеет значение: ‘’ (строка нулевой длины).  Операции над данными строкового типа:  = (бинарный) – присваивание значения. |

## 1.6 Преобразование типов данных

В языке программирования YNS-2022 преобразование типов данных не поддерживается, поскольку язык YNS-2022 строго типизированный.

## 1.7 Идентификаторы

Общее число идентификаторов не должно превышать максимальный размер таблицы идентификаторов (4096). Длина идентификатора не должна превышать 10 символов. В именах идентификаторов допустимо использование символов нижнего регистра. Идентификаторы не должны совпадать с ключевыми словами языка. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Идентификаторами являются: имя переменной, имя функции, имя параметра.

## 1.8 Литералы

Литералы позволяют производить инициализацию переменных. Для целочисленных литералов предусмотрено десятичное и шестнадцатеричное представление. Подробное описание литералов языка YNS-2022 представлены в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не отделяется пробелом). Не могут начинаться с 0, если их значение не 0. Если литерал отрицательный, то после минуса не может идти 0. |
| Целочисленные литералы в шестнадцатеричном представлении | Последовательность цифр 0…9 и символов ‘A’, ‘B’, ‘C’, ‘D’, ‘E’ с предшествующим символом “s” |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Строковые литералы | Набор символов алфавита языка, заключенных в одинарные кавычки. |

## 1.9 Объявление данных

Для объявления переменной используется слово new, далее указывается тип данных и имя идентификатора.

Объявление литерала:

new <тип> <имя идентификатора>;

Для объявления функции используется ключевое слово function перед которым указывается тип функции. Далее идёт список параметров и тело функции.

## 1.10 Инициализация данных

При инициализации данных переменной присваивается значение, находящееся справа от знака равенства. Инициализаторами могут быть идентификаторы и литералы. Инициализация допускается при объявлении переменной. При объявлении переменные инициализируются значением по умолчанию, у shint это 0, у line это ‘’.

## 1.11 Инструкции языка

Инструкции языка YNS-2022 приведены в таблице 1.4.

Таблица 1.4. Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Описание |
| Объявление переменной | new <тип данных><идентификатор>;  Пример: new line a; |
| Возврат значения функции | return <идентификатор|литерал>;  Пример: return a; |
| Вывод данных | out <идентификатор|литерал>;  Пример: out a; |
| Объявление функции | <тип> function <имя функции> (<список параметров>) [ тело функции ];  Пример: shint function func (shint a) [ return a; ] |
| Вызов функции | <идентификатор>(<список параметров>);  Пример: func(a); |
| Присваивание | <идентификатор>=<литерал|идентификатор|выражение>;  Пример: a = 5; |

## 1.12 Операции языка

В языке YNS-2022 предусмотрены два вида операций: арифметические и логические. Каждая из операций обладает приоритетом. Для повышения приоритета операции используются круглые скобки. Операции приведены в таблице 1.5.

Таблица 1.5 – Операции языка YNS-2022

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | **+ -** сложение (приоритет 3)  - **-** разность (приоритет 3)  **\* -** умножение (приоритет 2)  **/ -** деление (приоритет 2)  % - остаток от деления (приоритет 2) |
| Логические | **>** - больше (приоритет 4)  **<** - меньше (приоритет 4)  **!** – не равно (приоритет 5) |

## 1.13 Вычисление выражений

Вычислить выражение – значит выполнить все арифметические действия, указанные в нём, по определённым правилам:

* Выражение записывается в одну строку, без переносов
* Недопустимо использование двух арифметических операторов подряд;
* Разрешено использовать в качестве операнда вызов функции;
* Разрешено изменять приоритет операций, используя оператор «()»;
* Присваивать идентификатору результат выражения разрешено только если их типы данных совпадают;

## 1.14 Конструкции языка

Код программы на YNS- 2022 обязательно включает в себя главную функцию, но также может содержать и функции, разработанные пользователем. Конструкции языка представлены в таблице 1.6.

Таблица 1.6. Конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция | Main  [  …  ] |
| Внешняя функция | <тип> function <идентификатор>(<тип><идентификатор>,…)[ … return <идентификатор>; ] |
| Цикл | for: <идентификатор1> <логические оператор> <идентификатор2> #  perform […  ] # |

## 1.15 Области видимости идентификаторов

Область видимости идентификаторов: сверху вниз. Переменные, которые были объявлены в одной функции не доступны в другой. Объявления и операции происходят внутри тела функции. Переменные и параметры функции получают префикс: название функции, в которой они находятся.

Идентификаторы локальны, должны быть объявлены внутри функции. Глобальные переменные не поддерживаются. Параметры функции доступны только функции, в которой они объявлены.

## 1.16 Семантические проверки

В языке YNS-2022 будут осуществляться следующие проверки:

* Проверка наличия главной функции Main – точки входа в программу.
* Проверка количества точек входа в программу (не должно превышать 1).
* Проверка на наличие повторного объявления идентификатора.
* Проверка совпадения типа функции и типа возвращаемого ею значения.
* Проверка на использование идентификатора до его объявления.
* Проверка соответствия типа идентификатора и типа присваиваемого ему значения.
* Проверка значений целочисленных литералов на допустимость (входит ли в разрешенный диапазон).
* Проверка на правильность передаваемых в функцию параметров: количество и типы.

## 1.17 Распределение памяти

При трансляции кода используется две области памяти: сегмент констант и сегмент данных. Они заполняются с помощью таблицы лексем и таблицы идентификаторов, сформированных на этапе лексического анализа. В сегмент констант заносятся строковые и целочисленные литералы. В сегмент данных – переменные и параметры функций.

## 1.18 Стандартная библиотека

В YNS-2022 имеется стандартная библиотека, включающая в себя две функции: копирование строк и вычисление длины строки. Она подключается автоматически при трансляции в язык ассемблера.

Содержимое библиотеки и его описание в таблице 1.7.

Таблица 1.7. Содержимое библиотеки

|  |  |
| --- | --- |
| Функция стандартной библиотеки | Описание |
| CopyLine(line a, line b) | Является процедурой, которая выводит строку, полученную в результате копирования второй строки в первую |
| LengthLine(line a) | Численная функция, принимающая на вход строку типа line. Считает количество символов строки. Возвращает значение равное количеству символов. |

## 1.19 Ввод и вывод данных

Вывод данных осуществляется с помощью ключевого слова out. Использование оператора out допускается с литералами и идентификаторами. Для перевода строки существует специальный оператор newline.

Функция, управляющая выводом данных, реализована на языке C++. Вызов команды out в транслированном коде заменяется на вызов необходимой библиотечной функции. Библиотека подключается на этапе генерации кода автоматически.

## 1.20 Точка входа

Точкой входа является функция Main. Точка входа в приложение не может отсутствовать, а также не может быть переопределена.

## 1.21 Препроцессор

В языке YNS-2022 препроцессор отсутствует.

## 1.22 Соглашение о вызовах

При генерации кода используется соглашение о вызове stdcall. Параметры функций передаются через стек справа налево. Память освобождает вызываемый код.

## 1.23 Объектный код

Исходный код языка YNS-2022 транслируется в язык ассемблера.

## 1.24 Классификация сообщений транслятора

Сообщения, генерируемые транслятором, дают подробную информацию, чтобы пользователь понял, где и какие имеются ошибки в написанном коде. Сообщения об ошибках имеют префикс, который показывает на каком этапе обнаружена ошибка.

Сообщения транслятора и их характеристики приведены в таблице 1.8.

Таблица 1.8. Сообщения транслятора и их характеристики

|  |  |
| --- | --- |
| Префикс | Характеристика |
| [Lex]# | Ошибка обнаружена на этапе лексического анализа |
| [Sem]# | Ошибка обнаружена на этапе семантического анализа |
| [Syn]# | Ошибка обнаружена на этапе синтаксического анализа |

#### Сообщения без префикса означают что ошибка системная.

## 1.25 Контрольный пример

Контрольный пример демонстрирует основные особенности языка YNS-2022. В нём представлены типы данных, использование функций стандартной библиотеки, использование циклов, вывод данных, создание функций. Исходный код приведён в приложении А.

# Раздел 2. Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке YNS-2022 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора. Структура транслятора языка YNS-2022 приведена на рисунке 2.1.



Рисунок 2.1 Структура транслятора

Первым этапом трансляции является лексический анализ, поэтому сначала исходный код попадает в лексический анализатор. Лексический анализатор проверяет входные символы на допустимость и разбивает текст на токены. После этого формируется таблица лексем и таблица идентификаторов, которые, попадают в синтаксический анализатор. Если исходный код написан синтаксически верно (сохранена правильность всех конструкций языка), начинается следующая фаза трансляции – семантический анализ. В противном случае работа транслятора останавливается. Семантический анализатор представляет собой набор функций, которые проверяют соблюдение неформальных правил языка – объявление переменной до ее использования, соответствие фактических и формальных параметров функции и так далее. Конечным этапом является генерация кода, во время исполнения которого формируется ассемблерный код, созданный на основе входных данных полученных в результате работы предыдущих этапов.

## 2.2 Перечень входных параметров транслятора

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Описание | Значение по умолчанию |
| -in:<путь к файлу> | Файл с исходным кодом на языке YNS-2022 | Не предусмотрено |
| -log:<путь к файлу> | Файл для вывода протоколов работы программы | <имя файла>.log |

## 

## 2.3 Перечень протоколов, формируемых транслятором и их содержимое

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, содержащие в себе перечень протоколов работы.

В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка YNS-2022

|  |  |
| --- | --- |
| Протокол | Характеристика |
| Файл журнала работы программы (-log:) | Содержит таблицу лексем, таблицу идентификаторов, протокол работы синтаксического анализатора, дерево разбора, результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл с расширением \*.asm | Содержит код на языке ассемблера, сгенерированный из исходного кода. |

# Раздел 3. Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Первая стадия работы транслятора, это лексический анализ. Программа, которая его реализует, называется лексическим анализатором. Он выделяет из исходного кода простейшие конструкции языка, преобразует массив текстовых символов в массив токенов.

К функциям лексического анализатора относятся:

* удаление «пустых» символов. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними;
* распознавание идентификаторов и ключевых слов;
* распознавание констант;
* распознавание разделителей и знаков операций.

Структура лексического анализатора представлена на рисунке 3.1.

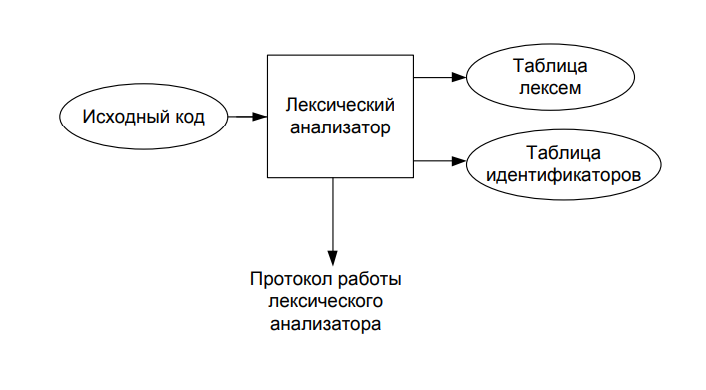


Рисунок 3.1 Структура лексического анализатора

Входными данными является исходный код на языке YNS-2022. Результатом работы программы являются таблица лексем и таблица идентификаторов, которые формируются в результате лексического анализа.

## 3.2 Контроль входных символов

Контроль входных символов осуществляется при помощи таблицы входных символов. Принцип работы таблицы заключается в соответствии значения каждого её элемента значению в таблице ASCII.

F – запрещенные символы, Т – разрешенные символы, I – игнорируемый символ, S – символ-разделитель, SPC – пробельные символы, LX –символы арифметических операций, скобки и запятые.

Таблица контроля входных символов представлена в приложение Б.

## 3.3 Удаление избыточных символов

Избыточными символами являются пробелы, символы табуляции. Они удаляются на этапе разбиения исходного кода на лексемы.

Алгоритм удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом;
2. При нахождении избыточного символа пропускаем его, не записывая в результирующий массив;
3. Продолжаем посимвольное считывание файла до встречи с символом, который не является избыточным.

## 3.4 Перечень ключевых слов

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.1.

Таблица 3.1 – Соответствие токенов и лексем в языке YNS-2022

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Описание |
| shint, line | t | Названия типов данных языка |
| Идентификатор | i | Максимальная длина идентификатора – 10 символов |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| return | r | Выход из функции. |
| Main | m | Главная функция (точка входа). |
| new | n | Объявление переменной. |
| out | p | Вывод данных. |
| for: | ? | Указывает на начало цикла. |
| perform | c | Указывает на начало тела цикла. |
| newline | ^ | Оператор вывода символа перевода строки. |
| # | # | Разделение конструкций в цикле. |
| ; | ; | Разделение конструкций языка. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока/тела функции. |
| ] | ] | Закрытие блока/тела функции. |
| ( | ( | Открытие блока передачи параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Оператор присваивания. |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Описание |
| +  -  \*  /  % | +  -  \*  /  % | Операторы арифметических операций. |
| >  <  ! | > < ! | Операторы логических операций. |

В приложении Б находится пример конечного автомата, используемый для разбора цепочки символов.

## 3.5 Основные структуры данных

Структуры таблиц лексем и идентификаторов данных языка YNS-2022, которые используются для хранения, представлены в приложении Б.

В таблице лексем содержатся сами лексемы, строка для каждой лексемы, в которой она была замечена. Так же размер самой таблицы лексем. В таблице идентификаторов содержится имя идентификатора, его номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение, а также имя родительской функции.

## 3.6 Принцип обработки ошибок

В случае возникновения ошибок во время трансляции программы, они фиксируются в протокол, который задаётся входными параметрами. Протоколирование происходит с номером ошибки и диагностическим сообщением.

## 3.7 Структура и перечень сообщений лексического анализа

Перечень сообщений представлен в приложении Б.

Сообщения об ошибках данной стадии имеют префикс [Lex]# что дает пользователю понять, на каком этапе возникла ошибка.

## 3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся в файл журнала.

## 3.9 Алгоритм лексического анализа

Aлгоритм работы лексического анализатора представлен ниже:

* Разделение текста на лексемы.
* Распознавание каждой строки с помощью автоматов.
* При удачном прохождении информация заносится в таблицу лексем и идентификаторов. Возвращение к шагу 2.
* Формирование протокола работы.
* При невозможности обработать строку выводится сообщение об ошибке.
* Конец работы лексического анализатора

## 3.10 Контрольный пример

Таблицы лексем и идентификаторов представлены в приложении Б.

# Раздел 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

Синтаксический анализатор, это часть компилятора, которая выполняет синтаксический анализ, то есть проверяет исходный код на соответствие всем правилам грамматики. На вход он принимает таблицу лексем и таблицу идентификаторов. В результате выдаёт дерево разбора.

Структура синтаксического анализатора представлена на рисунке 4.1

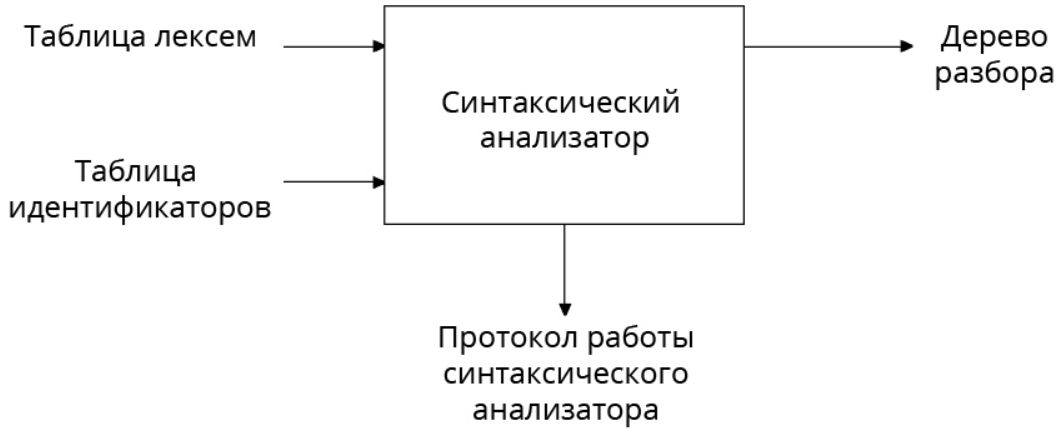


Рисунок 4.1 – Структура синтаксического анализатора

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис

В синтаксическом анализаторе транслятора языка YNS-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, поскольку она не леворекурсивная 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. Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 – Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->tfiPTS  S->pfiPGS  S->m[K] | Стартовые правила, описывающее общую структуру программы |
| P | P->(E) P->() | Правила для параметров объявляемых функций |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| T | T->[eV;]  T->[KeV;] | Правила для тела функций |
| E | E->ti,E  E->ti | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вывозов функций (в том числе и в выражениях) |
| N | N->i  N>l  N->l,N  N->I,N | Правила для параметров вызываемых функций |
| R | R->rY#  R>wY#  R>cY#  R->rYwY#  R->wYrY# | Правила составления цикла |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла |
| L | L->!  L->>  L->< | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->% | Правила для арифметических операторов |
| V | V->l  V->i | Правила для простых выражений |
| Y | Y->[X] | Правила для тела цикла |
| W | W->l  W->i  W->(W)  W->(W)AW  W->iF  W->iAW  W->lAW  W->iFAW | Правила для сложных выражений |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| K | K->nti=V;K  K->nti;K  K->i=W;K  K->oV;K  K->^;K  K->&Z#RK  K->iF;K  K->nti=V;  K->nti;  K->i=W;  K->oV;  K->^;  K->&Z#R  K->iF; | Программные конструкции |
| X | X->i=W;X  X->oV;X  X->^;X  X->iF;X  X->i=W;  X->oV;  X->^;  X->iF; | Программные конструкции внутри цикла |

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку.

Описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонент | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила, 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. |
|  | 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 состояния вынуждают автомат завершить 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 представлены в виде структуры магазинного конечного автомата, который выполняет разбор входной ленты, и структуры грамматики Грейбах, которая описывает синтаксические правила языка. Структуры представлены в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора:

1. Запись стартового символа в магазин;
2. Формирование входной ленты на основе полученных ранее таблиц;
3. Запуск автомата;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. В случае совпадения терминалов в стеке и в ленте, 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

Сообщения, который генерируются синтаксическим анализатором представлены в приложении В.

## 4.7. Параметры синтаксического анализатора и режимы его работы

Входными данными для синтаксического анализатора являются таблица лексем и идентификаторов. Помимо этого, используется описание грамматики в форме Грейбах. В результате лексического разбора, дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

## 4.8. Принцип обработки ошибок

В случае, если лексический анализатор во время разбора исходной последовательности найдёт ошибку, то анализ останавливается и выводится сообщение об ошибке. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## 4.9. Контрольный пример

Дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

# Раздел 5. Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Входными данными семантического анализатора являются результаты работ лексического и синтаксического анализаторов, а именно таблицы лексем, таблица идентификаторов и дерево разбора. Семантический анализатор 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 семантического анализатора представлена на рисунке 5.1.

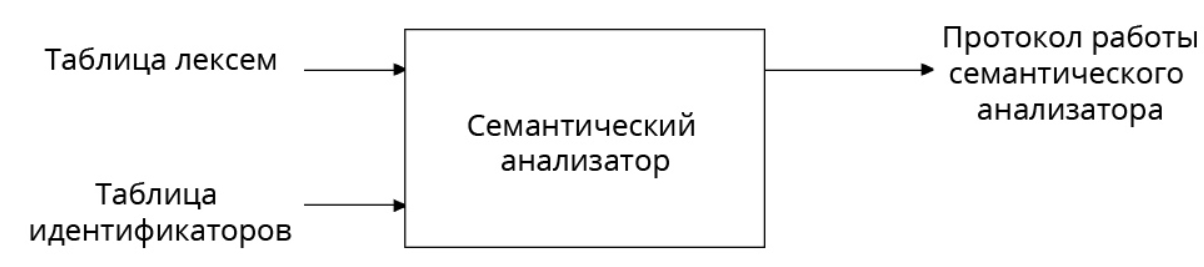


Рисунок 5.1. Структура семантического анализатора

## 5.2 Функции семантического анализатора

В функции семантического анализатора входит правильность составления программных конструкций. Если подобрать правило перехода невозможно, будет выведен код ошибки. Информация об ошибках выводится в консоль и в протокол работы.

## 5.3 Структура и перечень семантических ошибок

Сообщения, которые формируются семантическим анализатором, представлены в приложении Г.

## 5.4 Принцип обработки ошибок

Возникающие ошибки в процессе трансляции, заносятся в протокол. В случае появления ошибки, происходит её протоколирование с её номером и диагностическим сообщением.

## 5.5 Контрольный пример

Соответствие 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 [  new shint a = 5;  c = a;  ] | Ошибка 300: [Sem]# Необъявленный идентификатор  Строка 4 позиция -1 |
| Main[  new shint x = 9;  new line y =x;  ] | Ошибка 314: [Sem]# Типы данных в выражении не совпадают  Строка 4 позиция -1 |
| Main[  new shint x = 9;  ]  Main[  new line y = "qwerty";  ] | Ошибка 302: [Sem]# Обнаружено несколько точек входа main  Строка 6 позиция -1 |

# Раздел 6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

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

Таблица 6.1 – Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 0 |
| \* | 3 |
| / | 3 |
| + | 2 |
| - | 2 |
| % | 3 |

## 6.2 Польская запись и принцип ее построения

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

Польская запись — это альтернативный способ записи арифметических выражений, преимуществом которого является отсутствие скобок. Алгоритм построения польской записи:

* Входная 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;

## 6.3 Программная реализация обработки выражений

Реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

## 6.4 Контрольный пример

Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов при последующей обработке таблицы лексем.

Пример преобразования выражения к польской записи представлен в таблице 6.2.

Таблица 6.2 – Преобразование выражений к ПОЛИЗ

|  |  |  |
| --- | --- | --- |
| Исходное выражение | Стек | Результирующая строка |
| ( i - (l \* i)) - l |  |  |
| i - (l \* i)) - l | ( |  |
| - (l \* i)) - l | ( | i |
| (l \* i)) - l | -( | i |
| l \* i)) - l | (-( | i |
| \* i)) - l | (-( | il |
| i)) - l | \*(-( | il |
| )) - l | \*(-( | ili |
| ) - l | -( | ili\* |
| - l |  | ili\*- |
| l | - | ili\*- |
|  |  | ili\*-l- |

В приложении Г приведена изменённая таблица лексем, отображающая результаты преобразования выражений в формат польской записи.

# Раздел 7. Генерация кода

## 7.1 Структура генератора кода

В языке YNS-2022 генерация кода, это последний этап трансляции. Генератор принимает на вход таблицы лексем и идентификаторов. В соответствии с таблицей лексем строится выходной файл на языке ассемблера. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода YNS-2022 представлена на рисунке 7.1.

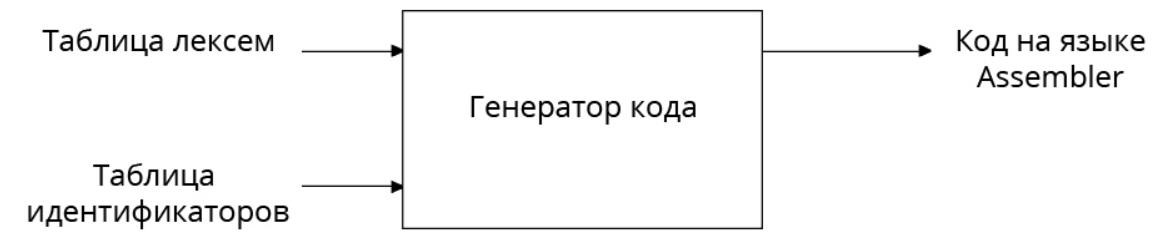


Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .DATA и .CONST языка ассемблера. Соответствия между типами данных идентификаторов на языке YNS-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке YNS-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| shint | sdword | Хранит целочисленный тип данных. |
| line | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## 

## 7.3 Статическая библиотека

В языке YNS-2022 есть статическая библиотека. В ней содержатся функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Стандартная библиотека подключается автоматически при генерации кода. Путь к библиотеке генерируется автоматически на стадии генерации кода.

Функции статической библиотеки приведены в таблице 7.3.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void outstr(char\* str) | Вывод на консоль строки line |

Продолжение таблицы 7.3

|  |  |
| --- | --- |
| Функция | Назначение |
| void outnum(int num) | Вывод на консоль целочисленной переменной shint |
| int LineLen(char\* str) | Возвращает длину строки |
| void CopyLine(char\* str1, char\* str2) | Вывод в консоль результат копирования строки 2 в конец строки 1 |

## 7.4 Особенности алгоритма генерации кода

Для генерации используются векторы и строки. Отдельные сегменты сначала записываются в строки, а после отправляются в вектор. В конце работы весь вектор последовательно выводится в файл.

## 7.5 Входные параметры генератора кода

Входными данными генератора являются таблицы лексем и идентификаторов исходного код программы на языке YNS-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 в приложении Д.

# Раздел 8. Тестирование транслятора

## 8.1 Общие положения

В языке YNS-2022 при возникновении ошибки на одном из этапов она обрабатывается в главной функции. Затем код ошибки и сообщение выводится в консольное окно и записывается в протокол работы.

## 8.2 Результаты тестирования

В таблице 8.1 приведены ошибки, возникающие при считывании из файла, а также на стадии лексического, синтаксического и семантического анализа.

Таблица 8.1 – Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Main  new line a = 'hello';  ] | 600: строка 1, [Syn]# Неверная структура программы |
| Main  [  new shint x = 1;  for: x < 15  perform [  x = x + 1;  ] #  ] | 610: строка 4, [Syn]# Неверный условный оператор |
| Main  [  new shint x = 1;  for: x < 15 #  perform [  x = x + 1;  ]  ] | 615: строка 7, [Syn]# Недопустимая синтаксическая конструкция в теле цикла |
| Main  [  new shint x = 2;  new line k = 'hello';  x = x + k;  ] | Ошибка 314: [Sem]# Типы данных в выражении не совпадают |
| [  ] | Ошибка 301: [Sem]# Отсутствует точка входа main  Строка -1 позиция -1 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Main  [  new shint a;  new shint b = 1;  a = LineLen(b);  ] | Ошибка 309: [Sem]# Несовпадение типов передаваемых параметров  Строка 5 позиция -1 |
| Main  [  вввввввввв;  new shint a;  out a;  ] | Ошибка 200: [Lex]# Невозможно распознать цепочку символов  Строка 3 позиция -1 |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования YNS-2022. Были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка YNS-2022;
* Разработаны конечные автоматы для работы лексического анализатора;
* Реализован лексический анализатор, распознающий допустимые цепочки спроектированного языка;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* Реализован синтаксический анализатор;
* Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка YNS-2022 включает:

* 2 типа данных;
* Поддержка операторов вывода и перевода строки;
* Наличие 5 арифметических операторов для вычисления выражений;
* Наличие 3 логических операторов для использования в условиях цикла;
* Поддержка функций и операторов цикла;
* Наличие библиотеки стандартных функций языка;
* Структурированная и классифицированная система для обработки ошибок пользователя.

В ходе данной работы было получено необходимое представление о структурах и процессах, использующихся при построении трансляторов и основные различия и преимущества тех или иных средств трансляции.

# Список использованных источников

1. Курс лекций по КПО Наркевич А.С.

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

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

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

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

# Приложение А

Исходный код программы на языке YNS-2022

|  |
| --- |
| shint function odd(shint t, shint n)  [  new shint i;  for: t < n #  perform [  i = i + 1;  out t; out ' ';  t = t + 2;  ] #  return i;  ]  Main  [  new shint a = 1;  new shint b = 10;  new shint q = sA;  out 'Вывод числа из шестнадцатеричного вида в десятичный: '; out q; newline;  new line t = 'Функция выбирающая нечетные числа';  out t; newline;  out 'Нижний предел: '; out a; newline;  out 'Верхний предел: '; out b; newline;  out 'Нечeтные числа: '; a = odd(a, b) + 5; newline;  out 'Количество чисел плюс 5: '; out a; newline;  new line s = 'Никита Яшный';  out 'Строка: '; out s; newline;  new shint l;  l = LineLen(s);  out 'Длина строки: '; out l; newline;  new line n = 'Nikita'; out n; newline;  new line y = 'Yashny'; out y; newline;  out 'Копирование одной строки в другую: ';  CopyLine(n, y); newline;  ] |

# Приложение Б

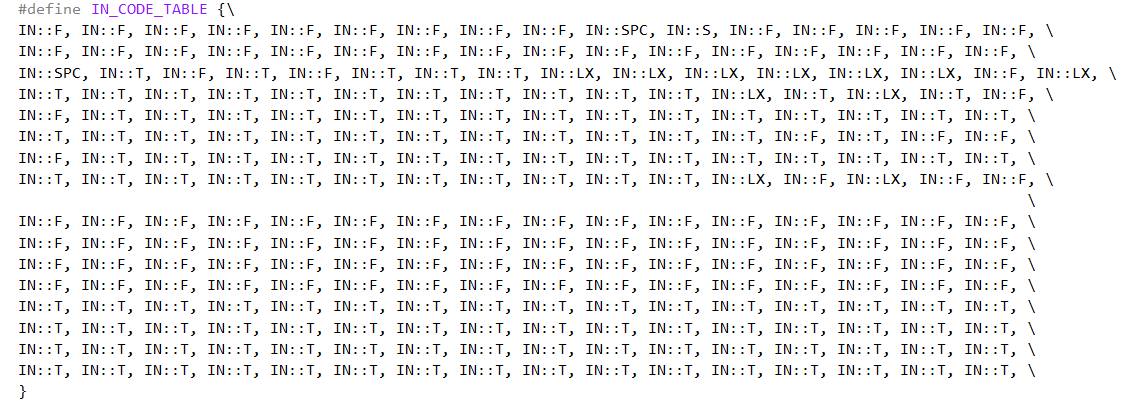


Рисунок 1 – Таблица контроля входных символов

Пример конечного автомата

|  |
| --- |
| #define FST\_PRINT 4, \  FST::NODE(1, FST::RELATION('o', 1)),\  FST::NODE(1, FST::RELATION('u, 2)),\  FST::NODE(1, FST::RELATION('t', 3)),\  FST::NODE() |

Структура таблицы лексем

|  |
| --- |
| struct Entry {  char lexema;  int sn;  int idxTI;  int priority;  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;  }; |

Структура таблицы идетификаторов

|  |
| --- |
| struct Entry  {  int idxfirstLE;  char visibility[ID\_MAXSIZE];  char id[ID\_MAXSIZE];  IDDATATYPE iddatatype;  IDTYPE idtype;  int countOfPar = 0;  IDDATATYPE\* types;  union  {  int vint;  bool vbool;  struct  {  int len;  char str[TI\_STR\_MAXSIZE - 1];  } vstr;  } value;  };  struct IdTable  {  int maxsize;  int size;  Entry\* table;  }; |

Сообщения об ошибках стадии лексического анализа

|  |
| --- |
| ERROR\_ENTRY(120, "[Lex]# Превышен максимльный размер таблицы лексем"),  ERROR\_ENTRY(121, "[Lex]# Таблица лексем переполнена"),  ERROR\_ENTRY(160, "[Lex]# Превышен максимльный размер таблицы идентификаторов"),  ERROR\_ENTRY(161, "[Lex]# Таблица идентификаторов переполнена"),  ERROR\_ENTRY(162, "[Lex]# Невозможно разобрать цепочку символов"), |

Таблица идентификаторов контрольного примера

|  |
| --- |
| № | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение  ---------------------------------------------------------------------------------------  0000 | odd | shint | функция | 2 | -  0001 | oddt | shint | параметр | 5 | -  0002 | oddn | shint | параметр | 8 | -  0003 | oddi | shint | переменная | 13 | 0  0004 | L1 | shint | литерал | 26 | 1  0005 | L2 | line | литерал | 32 | [1]" "  0006 | L3 | shint | литерал | 38 | 2  0007 | Maina | shint | переменная | 50 | 0  0008 | Mainb | shint | переменная | 56 | 0  0009 | L4 | shint | литерал | 58 | 10  0010 | Mainq | shint | переменная | 62 | 0  0011 | L5 | line | литерал | 67 | [53]"Вывод числа из шестнадцатиричного вида в десятичный: "  0012 | Maint | line | переменная | 76 | [0]""  0013 | L6 | line | литерал | 78 | [33]"Функция выбирающая нечетные числа"  0014 | L7 | line | литерал | 86 | [15]"Нижний предел: "  0015 | L8 | line | литерал | 94 | [16]"Верхний предел: "  0016 | L9 | line | литерал | 102 | [16]"Нечeтные числа: "  0017 | L10 | shint | литерал | 113 | 5  0018 | L11 | line | литерал | 118 | [25]"Количество чисел плюс 5: "  0019 | Mains | line | переменная | 127 | [0]""  0020 | L12 | line | литерал | 129 | [12]"Никита Яшный"  0021 | L13 | line | литерал | 132 | [8]"Строка: "  0022 | Mainl | shint | переменная | 141 | 0  0023 | LineLen | shint | стандартная функция | 145 | -  0024 | L14 | line | литерал | 151 | [14]"Длина строки: "  0025 | Mainn | line | переменная | 160 | [0]""  0026 | L15 | line | литерал | 162 | [6]"Nikita"  0027 | Mainy | line | переменная | 171 | [0]""  0028 | L16 | line | литерал | 173 | [6]"Yashny"  0029 | L17 | line | литерал | 181 | [35]"Копирование одной строки в другую: "  0030 | CopyLine | void | стандартная функция | 183 | - |

Таблица лексем контрольного примера

|  |  |
| --- | --- |
| tfi(ti,ti)[nti;  ?i<i#  c[i=i+l;  pi;  pl;  i=i+l;  ]#  ri;  ]m[nti=l;  nti=l;  nti=l;  pl;  pi;  ^;  nti=l;  pi;  ^;  pl;  pi;  ^;  pl;  pi;  ^;  pl; | i=i(i,i)+l;  ^;  pl;  pi;  ^;  nti=l;  pl;  pi;  ^;  nti;  i=i(i);  pl;  pi;  ^;  nti=l;  pi;  ^;  nti=l;  pi;  ^;  pl;  i(i,i);  ^;  nti; |

# Приложение В

Грамматика языка YNS-2022

|  |
| --- |
| Greibach greibach(NS('S'), TS('$'), 16,  Rule(NS('S'), GRB\_ERROR\_SERIES, 3,  Rule::Chain(6, TS('t'), TS('f'), TS('i'), NS('P'), NS('T'), NS('S')),  Rule::Chain(6, TS('p'), TS('f'), TS('i'), NS('P'), NS('G'), NS('S')),  Rule::Chain(4, TS('m'), TS('['), NS('K'), TS(']'))  ),  Rule(NS('T'), GRB\_ERROR\_SERIES + 2, 2,  Rule::Chain(5, TS('['), TS('r'), NS('V'), TS(';'), TS(']')),  Rule::Chain(6, TS('['), NS('K'), TS('r'), NS('V'), TS(';'), TS(']'))  ),  Rule(NS('P'), GRB\_ERROR\_SERIES + 1, 2  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(2, TS('('), TS(')'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 4, 2  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('E')),  Rule::Chain(2, TS('t'), TS('i'))  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 5, 2,  Rule::Chain(3, TS('('), NS('N'), TS(')')),  Rule::Chain(2, TS('('), TS(')'))  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 6, 4  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('N')),  Rule::Chain(3, TS('l'), TS(','), NS('N'))  ),  Rule(NS('R'), GRB\_ERROR\_SERIES + 7, 5,  Rule::Chain(3, TS('u'), NS('Y'), TS('#')),  Rule::Chain(3, TS('w'), NS('Y'), TS('#')),  Rule::Chain(3, TS('c'), NS('Y'), TS('#')),  Rule::Chain(5, TS('u'), NS('Y'), TS('w'), NS('Y'), TS('#')),  Rule::Chain(5, TS('w'), NS('Y'), TS('u'), NS('Y'), TS('#'))  ),  Rule(NS('Y'), GRB\_ERROR\_SERIES + 8, 1,  Rule::Chain(3, TS('['), NS('X'), TS(']'))  ),  Rule(NS('Z'), GRB\_ERROR\_SERIES + 9, 3,  Rule::Chain(3, TS('i'), NS('L'), TS('i')),  Rule::Chain(3, TS('i'), NS('L'), TS('l')),  Rule::Chain(3, TS('l'), NS('L'), TS('i'))  ),  Rule(NS('L'), GRB\_ERROR\_SERIES + 10, 3  Rule::Chain(1, TS('<')),  Rule::Chain(1, TS('>')),  Rule::Chain(1, TS('!'))  ),  Rule(NS('A'), GRB\_ERROR\_SERIES + 11, 5  Rule::Chain(1, TS('+')),  Rule::Chain(1, TS('-')),  Rule::Chain(1, TS('\*')),  Rule::Chain(1, TS('/')),  Rule::Chain(1, TS('%'))  ),  Rule(NS('V'), GRB\_ERROR\_SERIES + 12, 2  Rule::Chain(1, TS('l')),  Rule::Chain(1, TS('i'))  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 13, 8,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('W'), TS(')')),  Rule::Chain(5, TS('('), NS('W'), TS(')'), NS('A'), NS('W')),  Rule::Chain(2, TS('i'), NS('F')),  Rule::Chain(3, TS('i'), NS('A'), NS('W')),  Rule::Chain(4, TS('i'), NS('F'), NS('A'), NS('W')),  Rule::Chain(3, TS('l'), NS('A'), NS('W'))  ),  Rule(NS('K'), GRB\_ERROR\_SERIES + 14, 14  Rule::Chain(7, TS('n'), TS('t'), TS('i'), TS('='), NS('V'), TS(';'), NS('K')),  Rule::Chain(4, TS('i'), TS('='), NS('W'), TS(';')),  Rule::Chain(5, TS('n'), TS('t'), TS('i'), TS(';'), NS('K')),  Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('K')),  Rule::Chain(4, TS('p'), NS('V'), TS(';'), NS('K')),  Rule::Chain(3, TS('^'), TS(';'), NS('K')),  Rule::Chain(5, TS('?'), NS('Z'), TS('#'), NS('R'), NS('K')),  Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('K')),  Rule::Chain(6, TS('n'), TS('t'), TS('i'), TS('='), NS('V'), TS(';')),  Rule::Chain(4, TS('n'), TS('t'), TS('i'), TS(';')),  Rule::Chain(3, TS('p'), NS('V'), TS(';')),  Rule::Chain(2, TS('^'), TS(';')),  Rule::Chain(4, TS('?'), NS('Z'), TS('#'), NS('R')),  Rule::Chain(3, TS('i'), NS('F'), TS(';'))  ),  Rule(NS('X'), GRB\_ERROR\_SERIES + 15,  Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('X'  Rule::Chain(4, TS('p'), NS('V'), TS(';'), NS('X')),  Rule::Chain(3, TS('^'), TS(';'), NS('X')),  Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('X')),  Rule::Chain(4, TS('i'), TS('='), NS('W'), TS(';')),  Rule::Chain(3, TS('p'), NS('V'), TS(';')),  Rule::Chain(2, TS('^'), TS(';')),  Rule::Chain(3, TS('i'), NS('F'), TS(';'))  )  ); |

Пример конечного автомата

|  |
| --- |
| 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];  class my\_stack\_MfstState :public std::stack<MfstState> {  public:  using std::stack<MfstState>::c;  };  GRBALPHABET\* lenta;  short lenta\_position;  short nrule;  short nrulechain;  short lenta\_size;  GRB::Greibach grebach;  LT::LexTable lex;  MFSTSTSTACK st;  my\_stack\_MfstState storestate;  Mfst();  Mfst(LT::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(const Log::LOG& log);  bool resetstate(const Log::LOG& log);  bool push\_chain(GRB::Rule::Chain chain);  RC\_STEP step(const Log::LOG& log);  bool start(const Log::LOG& log);  bool savediagnosis(RC\_STEP pprc\_step);  void printrules(const Log::LOG& log);  struct Deducation  {  short size;  short\* nrules;  short\* nrulechains;  Deducation()  {  size = 0;  nrules = 0;  nrulechains = 0;  };  }deducation;  bool savededucation();  }; |

Начало разбора исходного кода синтаксическим анализатором

|  |
| --- |
| 0 : S->tfiPTS S$  0 : SAVESTATE: 1  0 : tfiPTS$  1 : fi(ti,ti)[nti;?i<i#c[i=i+ fiPTS$  2 : i(ti,ti)[nti;?i<i#c[i=i+l iPTS$  3 : (ti,ti)[nti;?i<i#c[i=i+l; PTS$ |

Конец разбора исходного кода синтаксическим анализатором

|  |
| --- |
| 673 : K->^; ^;]]]]]]]]]]]]]]]]]]]]]]] K]$  673 : SAVESTATE: 100  673 : ^;]]]]]]]]]]]]]]]]]]]]]]] ^;]$  674 : ;]]]]]]]]]]]]]]]]]]]]]]]] ;]$  675 : ]]]]]]]]]]]]]]]]]]]]]]]]] ]$  676 : ]]]]]]]]]]]]]]]]]]]]]]]]] $  677 : 6  678 : ------>LENTA\_END |

Структура грамматики Грейбах

|  |
| --- |
| 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);  }; |

Сообщения об ошибках стадии синтаксического анализа

|  |
| --- |
| ERROR\_ENTRY(600, "[Syn]# Неверная структура программы"),  ERROR\_ENTRY(601, "[Syn]# Не найден список параметров функции"),  ERROR\_ENTRY(602, "[Syn]# Ошибка в теле функции"),  ERROR\_ENTRY(603, "[Syn]# Ошибка в теле процедуры"),  ERROR\_ENTRY(604, "[Syn]# Ошибка в списке параметров функции"),  ERROR\_ENTRY(605, "[Syn]# Ошибка в вызове функции/выражении"),  ERROR\_ENTRY(606, "[Syn]# Ошибка в списке фактических параметров функции"),  ERROR\_ENTRY(607, "[Syn]# Ошибка при констуировании цикла"),  ERROR\_ENTRY(608, "[Syn]# Ошибка в теле цикла"),  ERROR\_ENTRY(609, "[Syn]# Ошибка в условии цикла"),  ERROR\_ENTRY(610, "[Syn]# Неверный условный оператор"),  ERROR\_ENTRY(611, "[Syn]# Неверный арифметический оператор"),  ERROR\_ENTRY(612, "[Syn]# Неверное выражение. Ожидаются только идентификаторы/литералы"),  ERROR\_ENTRY(613, "[Syn]# Ошибка в арифметическом выражении"),  ERROR\_ENTRY(614, "[Syn]# Недопустимая синтаксическая конструкция"),  ERROR\_ENTRY(615, "[Syn]# Недопустимая синтаксическая конструкция в теле цикла"), |

# Приложение Г

Сообщения об ошибках стадии семантического анализа

|  |
| --- |
| ERROR\_ENTRY(300, "[Sem]# Необъявленный идентификатор"),  ERROR\_ENTRY(301, "[Sem]# Отсутствует точка входа main"),  ERROR\_ENTRY(302, "[Sem]# Обнаружено несколько точек входа main"),  ERROR\_ENTRY(303, "[Sem]# В объявлении не указан тип идентификатора"),  ERROR\_ENTRY(304, "[Sem]# В объявлении отсутствует ключевое слово new"),  ERROR\_ENTRY(305, "[Sem]# Попытка переопределения идентификатора"),  ERROR\_ENTRY(308, "[Sem]# Кол-во ожидаемых функцией и передаваемых пар-ов не совпадают"),  ERROR\_ENTRY(309, "[Sem]# Несовпадение типов передаваемых параметров"),  ERROR\_ENTRY(310, "[Sem]# Использование пустого строкового литерала недопустимо"),  ERROR\_ENTRY(311, "[Sem]# Обнаружен символ \'. Возможно, не закрыт строковый литерал"),  ERROR\_ENTRY(312, "[Sem]# Превышен размер строкового литерала"),  ERROR\_ENTRY(313, "[Sem]# Недопустимый целочисленный литерал"),  ERROR\_ENTRY(314, "[Sem]# Типы данных в выражении не совпадают"),  ERROR\_ENTRY(315, "[Sem]# Тип функции и возвращаемого значения не совпадают"),  ERROR\_ENTRY(317, "[Sem]# Неверное условное выражение"),  ERROR\_ENTRY(318, "[Sem]# Деление на ноль"), |

Программная реализация механизма преобразования в ПОЛИЗ

|  |
| --- |
| bool PolishNotation(int i, Lex::LEX& lex)  {  stack<LT::Entry> stack;  queue<LT::Entry> queue;  LT::Entry temp;  temp.idxTI = -1;  temp.lexema = ' ';  temp.sn = -1;  LT::Entry func;  func.lexema = '^';  LT::Entry num;  num.lexema = ' ';  int countLex = 0;  int posLex = i;  bool findFunc = false;  int buf = 0;  for (i; lex.lextable.table[i].lexema != LEX\_SEMICOLON; i++, countLex++)  {  switch (lex.lextable.table[i].lexema)  {  case LEX\_ID:  {  queue.push(lex.lextable.table[i]);  continue;  }  case LEX\_LITERAL:  {  queue.push(lex.lextable.table[i]);  continue;  }  case LEX\_COMMA:  {  if (findFunc)  stack.push(lex.lextable.table[i]);  }  case LEX\_LEFTTHESIS: // если (  {  if (lex.lextable.table[i - 1].lexema == LEX\_ID)  {  if (lex.idtable.table[lex.lextable.table[i - 1].idxTI].idtype == IT::F || lex.idtable.table[lex.lextable.table[i - 1].idxTI].idtype == IT::SF || lex.idtable.table[lex.lextable.table[i - 1].idxTI].idtype == IT::SFL) {  findFunc = true;  num.lexema = (char)lex.idtable.table[lex.lextable.table[i - 1].idxTI].countOfPar + '0';  }  }  if (findFunc)  queue.push(lex.lextable.table[i]);  else  stack.push(lex.lextable.table[i]);  continue;  }  case LEX\_RIGHTTHESIS:  {  if (findFunc)  queue.push(lex.lextable.table[i]);  else {  while (stack.top().lexema != LEX\_LEFTTHESIS)  {  queue.push(stack.top());  stack.pop();  if (stack.empty())  return false;  }  stack.pop();  }  continue;  }  case LEX\_PLUS: case LEX\_MINUS: case LEX\_STAR: case LEX\_DIRSLASH: case LEX\_MOD:  {  while (!stack.empty() && lex.lextable.table[i].priority <= stack.top().priority)  {  queue.push(stack.top()); с  stack.pop();  }  stack.push(lex.lextable.table[i]);  continue;  }  }  }  while (!stack.empty())  {  if (stack.top().lexema == LEX\_LEFTTHESIS || stack.top().lexema == LEX\_RIGHTTHESIS)  return false;  queue.push(stack.top());  stack.pop();  }  while (countLex != 0)  {  if (!queue.empty()) {  lex.lextable.table[posLex++] = queue.front();  queue.pop();  }  else  {  lex.lextable.table[posLex++] = temp;  }  countLex--;  }  for (int i = 0; i < posLex; i++)  {  if (lex.lextable.table[i].lexema == LEX\_PLUS || lex.lextable.table[i].lexema == LEX\_MINUS || lex.lextable.table[i].lexema == LEX\_STAR || lex.lextable.table[i].lexema == LEX\_DIRSLASH || lex.lextable.table[i].lexema == LEX\_LITERAL)  lex.idtable.table[lex.lextable.table[i].idxTI].idxfirstLE = i;  }  return true;  } |

Измененная таблица лексем после преобразования выражений

|  |  |  |
| --- | --- | --- |
| tfi(ti,ti)[nti;  ?i<i#  c[i=il+;  pi;  pl;  i=il+;  ]#  ri;  ]m[nti=l;  nti=l;  nti=l;  pl;  pi;  ^;  nti=l;  pi; | ^;  pl;  pi;  ^;  pl;  pi;  ^;  pl;  i=i(i,i)l+;  ^;  pl;  pi;  ^;  nti=l;  pl;  pi; | ^;  nti;  i=i(i);  pl;  pi;  ^;  nti=l;  pi;  ^;  nti=l;  pi;  ^;  pl;  i(i,i);  ^; |

# Приложение Д

Результат генерации кода контрольного примера в Ассемблер

|  |  |
| --- | --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib "C:\Users\User\Desktop\курсач\YNS-2022\Generation\Debug\GenLib.lib  ExitProcess PROTO:DWORD  .stack 4096  outnum PROTO : DWORD  outstr PROTO : DWORD  LineLen PROTO : DWORD  CopyLine PROTO : DWORD, : DWORD  .const  printline byte 13, 10, 0  L1 sdword 1  L2 byte ' ', 0  L3 sdword 2  L4 sdword 10  L5 byte 'Вывод числа из шестнадцатиричного вида в десятичный: ', 0  L6 byte 'Функция выбирающая нечетные числа', 0  L7 byte 'Нижний предел: ', 0  L8 byte 'Верхний предел: ', 0  L9 byte 'Нечeтные числа: ', 0  L10 sdword 5  L11 byte 'Количество чисел плюс 5: ', 0  L12 byte 'Никита Яшный', 0  L13 byte 'Строка: ', 0  L14 byte 'Длина строки: ', 0  L15 byte 'Nikita', 0  L16 byte 'Yashny', 0  L17 byte 'Копирование одной строки в другую: ', 0  .data  temp sdword ?  buffer byte 256 dup(0)  oddi sdword 0  Maina sdword 0  Mainb sdword 0  Mainq sdword 0  Maint dword ?  Mains dword ?  Mainl sdword 0  Mainn dword ?  Mainy dword ?  .code  ;----------- odd ------------  odd PROC,  oddt : sdword, oddn : sdword  push ebx  push edx  mov edx, oddt  cmp edx, oddn  jl repeat1  jmp repeatnext1  repeat1:  push oddi  push L1  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov oddi, ebx  push oddt  call outnum  push offset L2  call outstr  push oddt  push L3  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov oddt, ebx  mov edx, oddt  cmp edx, oddn  jl repeat1  repeatnext1:  pop edx  pop ebx  mov eax, oddi  ret  odd ENDP  ;------------------------------  ;----------- MAIN ------------  main PROC  push L1  pop ebx  mov Maina, ebx  push L4  pop ebx  mov Mainb, ebx  push L4 | pop ebx  mov Mainq, ebx  push offset L5  call outstr  push Mainq  call outnum  push offset printline  call outstr  mov Maint, offset L6  push Maint  call outstr  push offset printline  call outstr  push offset L7  call outstr  push Maina  call outnum  push offset printline  call outstr  push offset L8  call outstr  push Mainb  call outnum  push offset printline  call outstr  push offset L9  call outstr  push Mainb  push Maina  call odd  push eax  push L10  pop ebx  pop eax  add eax, ebx  push eax  pop ebx  mov Maina, ebx  push offset printline  call outstr  push offset L11  call outstr  push Maina  call outnum  push offset printline  call outstr  mov Mains, offset L12  push offset L13  call outstr  push Mains  call outstr  push offset printline  call outstr  push Mains  call LineLen  push eax  pop ebx  mov Mainl, ebx  push offset L14  call outstr  push Mainl  call outnum  push offset printline  call outstr  mov Mainn, offset L15  push Mainn  call outstr  push offset printline  call outstr  mov Mainy, offset L16  push Mainy  call outstr  push offset printline  call outstr  push offset L17  call outstr  push Mainy  push Mainn  call CopyLine  push offset printline  call outstr  push 0  call ExitProcess  main ENDP  end main |

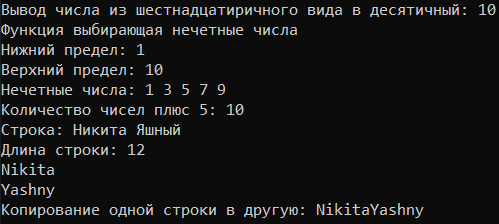


Рисунок 1 – Результат работы программы на языке YNS-2022