

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 SIV-2023»

Выполнил студент Слесарев Иван Витальевич
(Ф.И.О.)

Руководитель проекта асс. Мущук Артур Николаевич
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

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

Консультант асс. Мущук Артур Николаевич
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Мущук Артур Николаевич
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Содержание

| | |
|--|----|
| Введение | 5 |
| 1 Спецификация языка программирования | 6 |
| 1.1 Характеристика языка программирования | 6 |
| 1.2 Определение алфавита языка программирования | 6 |
| 1.3 Применяемые сепараторы | 6 |
| 1.4 Применяемые кодировки..... | 7 |
| 1.5 Типы данных..... | 7 |
| 1.6 Преобразование типов данных | 8 |
| 1.7 Идентификаторы | 8 |
| 1.8 Литералы | 8 |
| 1.9 Объявление данных..... | 9 |
| 1.10 Инициализация данных | 9 |
| 1.11 Инструкции языка | 10 |
| 1.12 Операции языка | 10 |
| 1.13 Выражения и их вычисления | 11 |
| 1.14 Конструкции языка | 11 |
| 1.15 Область видимости идентификатора | 12 |
| 1.16 Семантические проверки..... | 12 |
| 1.17 Распределение оперативной памяти на этапе выполнения..... | 12 |
| 1.18 Стандартная библиотека и её состав..... | 13 |
| 1.19 Ввод и вывод данных | 13 |
| 1.20 Точка входа | 13 |
| 1.21 Препроцессор..... | 13 |
| 1.22 Соглашение о вызовах | 13 |
| 1.23 Объектный код..... | 14 |
| 1.24 Классификация сообщений транслятора | 14 |
| 1.25 Контрольный пример | 14 |
| 2 Структура транслятора..... | 15 |
| 2.1 Компоненты транслятора их назначение и принципы взаимодействия | 15 |
| 2.2 Перечень входных параметров транслятора | 16 |

| | | |
|------|--|----|
| 2.3 | Протоколы, формируемые транслятором | 16 |
| 3 | Разработка лексического анализатора | 17 |
| 3.1 | Структура лексического анализатора | 17 |
| 3.2 | Входные и выходные данные лексического анализатора | 17 |
| 3.3 | Параметры лексического анализатора | 17 |
| 3.4 | Алгоритм лексического анализа | 17 |
| 3.5 | Контроль входных символов | 18 |
| 3.6 | Удаление избыточных символов | 18 |
| 3.7 | Перечень ключевых слов | 18 |
| 3.8 | Основные структуры данных | 21 |
| 3.9 | Структура и перечень сообщений лексического анализатора | 22 |
| 3.10 | Принцип обработки ошибок | 23 |
| 3.11 | Контрольный пример | 23 |
| 4 | Разработка синтаксического анализатора | 24 |
| 4.1 | Структура синтаксического анализатора | 24 |
| 4.2 | Контекстно-свободная грамматика, описывающая синтаксис языка | 24 |
| 4.3 | Построение конечного магазинного автомата | 27 |
| 4.4 | Основные структуры данных | 28 |
| 4.5 | Описание алгоритма синтаксического разбора | 28 |
| 4.6 | Параметры синтаксического анализатора | 29 |
| 4.7 | Структура и перечень сообщений синтаксического анализатора | 29 |
| 4.8 | Принцип обработки ошибок | 29 |
| 4.9 | Контрольный пример | 29 |
| 5 | Разработка семантического анализатора | 30 |
| 5.1 | Структура семантического анализатора | 30 |
| 5.2 | Функции семантического анализатора | 30 |
| 5.3 | Структура и перечень сообщений семантического анализатора | 31 |
| 5.4 | Принцип обработки ошибок | 32 |
| 5.5 | Контрольный пример | 32 |
| 6 | Вычисление выражений | 33 |
| 6.1 | Выражение, допускаемые языком | 33 |
| 6.2 | Польская запись и принцип её построения | 33 |
| 6.3 | Программная реализация обработки выражений | 34 |

| | | |
|-----|---|----|
| 6.4 | Контрольный пример | 35 |
| 7 | Генерация кода..... | 36 |
| 7.1 | Структура генератора кода..... | 36 |
| 7.2 | Представление типов данных в оперативной памяти | 36 |
| 7.3 | Статическая библиотека | 37 |
| 7.4 | Особенности алгоритма генерации кода | 37 |
| 7.5 | Параметры, управляющие генерацией кода | 38 |
| 7.6 | Контрольный пример | 38 |
| 8 | Тестирование транслятора..... | 39 |
| 8.1 | Общие положения | 39 |
| 8.2 | Результаты тестирования..... | 39 |
| | Заключение | 42 |
| | Список использованных литературных источников..... | 43 |
| | Приложение А | 44 |
| | Приложение Б..... | 47 |
| | Приложение В | 48 |
| | Приложение Г | 56 |
| | Приложение Д | 57 |
| | Приложение Е..... | 59 |

Введение

Целью выполнения курсового проекта по дисциплине «Конструирование программного обеспечения» является написание спецификации и разработка компилятора для собственного языка программирования.

Название языка, для которого разрабатывается компилятор, – SIV-2023. Компиляция будет производиться в язык ассемблера.

Этапы разработки компилятора для языка SIV-2023:

- Написание спецификации языка программирования;
- Разработка лексического анализатора;
- Разработка синтаксического анализатора;
- Разработка семантического анализатора;
- Преобразование арифметических выражений;
- Генерация кода;
- Тестирование транслятора.

Информация о каждом этапе разработки компилятора приведена в соответствующих разделах пояснительной записки.

В первом разделе приведена спецификация языка – точное формализованное описание набора правил, определяющих лексику, синтаксис и семантику языка.

Во втором разделе описана структура компилятора.

В третьем разделе описаны принцип работы и этапы разработки лексического анализатора, определены разрешенные символы и ключевые слова языка программирования.

В четвертом разделе описан принцип работы синтаксического анализатора, формальная грамматика определена и приведена в нормальную форму Грейбах для выполнения синтаксического разбора.

В пятом разделе описаны принцип работы и основные функции семантического анализатора.

В шестом разделе описаны выражения, допускаемые языком, форма, принципы построения и вычисления выражений.

В седьмом разделе описан процесс генерации кода.

В восьмом разделе приведены примеры тестирования транслятора.

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык SIV-2023 является компилируемым, строго типизированным, процедурным, поддерживающим парадигму структурного программирования.

1.2 Определение алфавита языка программирования

Алфавит языка SIV-2023 основан на кодировке Windows-1251, изображенной на рисунке 1.1.

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
|----|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|------------|-------------|-------------|------------|------------|------------|-------------|
| 00 | NUL 0000 | STX 0001 | SOT 0002 | ETX 0003 | EOT 0004 | ENQ 0005 | ACK 0006 | BEL 0007 | BS 0008 | HT 0009 | LF 000A | VT 000B | FF 000C | CR 000D | SO 000E | SI 000F |
| 10 | DLE 0010 | DC1 0011 | DC2 0012 | DC3 0013 | DC4 0014 | NAK 0015 | SYN 0016 | ETB 0017 | CAN 0018 | EM 0019 | SUB 001A | ESC 001B | FS 001C | GS 001D | RS 001E | US 001F |
| 20 | SP 0020 | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 30 | 0 0030 | 1 0031 | 2 0032 | 3 0033 | 4 0034 | 5 0035 | 6 0036 | 7 0037 | 8 0038 | 9 0039 | : | ; | < | = | > | ? |
| 40 | @ 0040 | A 0041 | B 0042 | C 0043 | D 0044 | E 0045 | F 0046 | G 0047 | H 0048 | I 0049 | J 004A | K 004B | L 004C | M 004D | N 004E | O 004F |
| 50 | P 0050 | Q 0051 | R 0052 | S 0053 | T 0054 | U 0055 | V 0056 | W 0057 | X 0058 | Y 0059 | Z 005A | [005B | \ 005C |] 005D | ^ 005E | _ 005F |
| 60 | ` 0060 | a 0061 | b 0062 | c 0063 | d 0064 | e 0065 | f 0066 | g 0067 | h 0068 | i 0069 | j 006A | k 006B | l 006C | m 006D | n 006E | o 006F |
| 70 | p 0070 | q 0071 | r 0072 | s 0073 | t 0074 | u 0075 | v 0076 | w 0077 | x 0078 | y 0079 | z 007A | { 007B | 007C | } 007D | ~ 007E | DEL 007F |
| 80 | Ђ 0402 | Ѓ 0403 | Ѕ 201A | Ї 0453 | Љ 201E | Њ 2026 | Ћ 2020 | Ќ 2021 | Ў 20AC | а 2030 | в 0409 | г 2039 | д 040A | е 040C | ж 040B | з 040F |
| 90 | Љ 0452 | Њ 2018 | Ћ 2019 | Ќ 201C | Ѝ 201D | Ў 2022 | а 2013 | в 2014 | г 2122 | д 0459 | е 203A | ж 045A | з 045C | и 045B | й 045E | к 045F |
| A0 | Њ 00A0 | Ћ 040E | Ќ 045E | Ѝ 0408 | Ў 00A4 | а 0490 | в 00A6 | г 00A7 | д 0401 | е 00A9 | ж 0404 | з 00AB | и 00AC | й 00AD | к 00AE | л 0407 |
| B0 | ° 00B0 | ± 00B1 | І 0406 | і 0456 | ґ 0491 | μ 00B5 | ¶ 00B6 | · 00B7 | ё 0451 | № 2116 | е 0454 | » 00BB | ј 0458 | ѕ 0405 | ѕ 0455 | ї 0457 |
| C0 | А 0410 | В 0411 | В 0412 | Г 0413 | Д 0414 | Е 0415 | Ж 0416 | З 0417 | И 0418 | Й 0419 | К 041A | Л 041B | М 041C | Н 041D | О 041E | П 041F |
| D0 | Р 0420 | С 0421 | Т 0422 | У 0423 | Ф 0424 | Х 0425 | Ц 0426 | Ч 0427 | Ш 0428 | Щ 0429 | Ъ 042A | Ы 042B | Ь 042C | Э 042D | Ю 042E | Я 042F |
| E0 | а 0430 | б 0431 | в 0432 | г 0433 | д 0434 | е 0435 | ж 0436 | з 0437 | и 0438 | й 0439 | к 043A | л 043B | м 043C | н 043D | о 043E | п 043F |
| F0 | р 0440 | с 0441 | т 0442 | у 0443 | ф 0444 | х 0445 | ц 0446 | ч 0447 | ш 0448 | щ 0449 | ъ 044A | ы 044B | ь 044C | э 044D | ю 044E | я 044F |

Рисунок 1.1 – Алфавит входных символов языка SIV-2023

1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей конструкций языка во время обработки исходного текста программы с целью разделения на токены.

Сепараторы, применяемые в языке SIV-2023, приведены в таблице 1.1.

Таблица 1.1 – Применяемые сепараторы

| Разделители | Назначение |
|--|--|
| ‘пробел’, ‘табуляция’, ‘переход на новую строку’ | Разделяют входные лексемы |
| +, -, *, /, % | Арифметические операторы. Используются в арифметических операциях |
| = | Оператор присваивания. Используется для присваивания значения переменной |
| <, >, <=, >=, !=, == | Условные операторы. Используются для сравнения переменных и литералов |
| () | Блок параметров функции, так же указывает приоритет в арифметических операциях |
| , | Разделяет параметры функции |
| { } | Ограничивают программные конструкции |
| ; | Признак конца инструкции языка |

1.4 Применяемые кодировки

Для написания кода на языке программирования SIV-2023 используется кодировка Windows-1251.

1.5 Типы данных

В языке SIV-2023 поддерживается 3 типа данных: целочисленный, строковый и логический. Подробная описание типов данных приведено в таблице 1.2.

Таблица 1.2 – Типы данных языка SIV-2023

| Тип данных | Характеристика |
|---------------------|---|
| Целочисленный (int) | <p>В памяти занимает 2 байта. Максимальное значение: 32767. Минимальное значение: -32768. Принцип размещения в памяти: Последний бит числа отведен под знак, оставшиеся 15 бит предназначены для хранения значения числа. Значение по умолчанию: 0. В арифметических выражениях и условных конструкциях к целочисленным переменным и литералам применимы все арифметические и условные операции соответственно, поддерживаемые языком SIV-2023.</p> |

Окончание таблицы 1.2

| Тип данных | Характеристика |
|----------------------|--|
| Строковый (str) | <p>В памяти занимает $n + 1$ байт, где n – количество символов в строке + символ конца строки.</p> <p>Максимальное количество символов в строке: 255.</p> <p>Принцип размещения в памяти:</p> <p>Каждый символ строки занимает 1 байт. В конце строки располагается NULL символ (признак конца строки).</p> <p>К строковым переменным и литералам операции не применяются.</p> |
| Логический (bool) | <p>В памяти занимает 1 байт.</p> <p>Может принимать одно из двух значений: true или false.</p> <p>Принцип размещения в памяти:</p> <p>В зависимости от значения 1 бит числа установлен true: 1, false: 0.</p> <p>К булевым переменным и литералам применимы все условные операции, поддерживаемые языком SIV-2023.</p> |

1.6 Преобразование типов данных

Преобразования типов данных в языке программирования SIV-2023 не поддерживаются.

1.7 Идентификаторы

Идентификатор – это имя, используемое для переменных, функций, параметров. Идентификаторы могут состоять как из одного, так и из нескольких символов. Первым символом должна быть маленькая буква латинского алфавита, а за ним могут стоять маленькие буквы латинского алфавита или цифры. Идентификаторы не могут совпадать с ключевыми словами.

Пример корректных идентификаторов: str1, abc161 и т.п.

Пример некорректных идентификаторов: 14stroka, Stroka1, _stroka1 и т.п.

1.8 Литералы

Литерал – это запись в исходном коде программы, представляющая собой фиксированное значение. В языке программирования SIV-2023 предусмотрены следующие типы литералов: строковый, логический и целочисленный. Целочисленные литералы представлены в 4 системах счисления: двоичная, восьмеричная, десятичная, шестнадцатеричная.

Описание литералов приведено в таблице 1.3.

Таблица 1.3 – Литералы языка SIV-2023

| Тип литерала | Характеристика |
|---------------|---|
| Целочисленный | <p>Десятичный: Последовательность десятичных цифр 0..9 с предшествующим знаком минус или без него.</p> <p>Двоичный: Последовательность двоичных цифр 0 и 1 с предшествующим знаком минус или без него, в конце которой стоит символ 'В' (признак двоичного целочисленного литерала).</p> <p>Восьмеричный: Последовательность восьмеричных цифр 0..7 с предшествующим знаком минус или без него, в конце которой стоит символ 'О' (признак восьмеричного целочисленного литерала).</p> <p>Шестнадцатеричный: Последовательность шестнадцатеричных чисел 0..F с предшествующим знаком минус или без него, в конце которой стоит символ 'Н' (признак шестнадцатеричного целочисленного литерала).</p> <p>Допустимый диапазон значений: От -32768 до 32767 в десятичной системе исчисления.</p> |
| Строковый | <p>Набор, состоящий из символов русского и латинского алфавитов, десятичных цифр и специальных символов, заключенный в двойные кавычки.</p> <p>Допустимый диапазон значений: От 0 до 255 символов.</p> |
| Логический | Допустимые значения: true или false, где true: логическая единица, false: логический ноль. |

Пример корректных литералов: 56, -3, 6D4H, -110B, 43O, true, "string" и т.п.

Примеры некорректных литералов: -7A, 'stroka', fals, 9O и т.п.

1.9 Объявление данных

Для объявления переменной используется ключевое слово new, после которого указывается тип переменной и имя идентификатора. Так же при объявлении допускается инициализация переменной.

new <тип> <имя идентификатора>;

new <тип> <имя идентификатора> = <литерал>;

Переменную можно объявить в блоке main, в блоке функции или в условном блоке if-else. Область видимости идентификаторов определяется блоком кода, который заключен в { }.

1.10 Инициализация данных

В языке SIV-2023 присутствует 2 вида инициализации:

- Инициализация в месте объявления
new <тип> <имя идентификатора> = <литерал>;
- Инициализация после объявления
<имя идентификатора> = <литерал>;

Так же в языке присутствует инициализация по умолчанию. Переменные целочисленного типа по умолчанию инициализируются значением 0. Переменные строкового типа по умолчанию инициализируются пустой строкой. Переменные логического типа по умолчанию инициализируются значением false.

1.11 Инструкции языка

Инструкции языка SIV-2023 приведены в таблице 1.4.

Таблица 1.4 – Инструкции языка SIV-2023

| Инструкция | Форма записи |
|--|---|
| Объявление переменной | new <тип данных> <идентификатор>. |
| Объявление переменной с явной инициализацией | new <тип данных> <идентификатор> = <значение> <выражение>; Значение – литерал, идентификатор, вызов функции соответствующего типа данных |
| Объявление функции | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, ...) { / тело функции / return <идентификатор/литерал>. }; |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | <идентификатор> = <значение>; |
| Вывод данных | print <идентификатор/литерал>; |
| Возвращаемое значение | return <литерал/идентификатор>. |

1.12 Операции языка

В языке SIV-2023 существует два типа операций: арифметические и логические.

Наибольшая приоритетность у операций умножения, деления и деления с остатком, затем идут операции сложения и вычитания. Можно задать самый высокий приоритет, поместив операции в скобки.

Все логические операции имеют равный приоритет.

Описание операций языка SIV-2023 приведено в таблице 1.5.

Таблица 1.5 – Операции языка SIV-2023

| Тип оператора | Оператор |
|-------------------------|--|
| Арифметические операции | + – сложение - – вычитание * – умножение / – деление % – остаток от деления |
| Логические операции | > – больше < – меньше >= – больше или равно <= – меньше или равно == – проверка на равенство != – проверка на неравенство |

1.13 Выражения и их вычисления

Выражение языка программирования SIV-2023 представляет собой совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка.

Правила составления выражений:

- Выражения записываются в одну строку;
- В выражении могут присутствовать только операнды одинакового типа;
- В выражении могут использоваться функции. Как стандартные, так и пользовательские;
- В выражении не могут идти подряд два оператора;
- Допускается использование круглых скобок для смены приоритета операций.

В арифметических выражениях допускаются только операнды целочисленного типа. В выражениях сравнения допускаются операнды булевого и целочисленного типов.

Перед генерацией кода выражения приводятся к ПОЛИЗ для более удобного вычисления на языке ассемблера.

1.14 Конструкции языка

Конструкции языка SIV-2023 приведены в таблице 1.6.

Таблица 1.6 – Конструкции языка SIV-2023

| Конструкция | Описание |
|-----------------|--------------------------|
| Главная функция | <pre>main { ... };</pre> |

Окончание таблицы 1.6

| Конструкция | Описание |
|--------------------------|---|
| Пользовательская функция | <code><тип возвращаемого значения> function(<тип параметра> <имя параметра>, ...)</code> <code>{</code> <code>...</code> <code>return <имя переменной/литерал>;</code> <code>};</code> Максимальное количество параметров: 8. |
| Условная конструкция | <code>if(<имя переменной/литерал><условный оператор><имя переменной/литерал>)</code> <code>{</code> <code>...</code> <code>}</code> <code>else</code> <code>{</code> <code>...</code> <code>}</code> (блок else необязателен) |

1.15 Область видимости идентификатора

Каждой конструкции языка SIV-2023 соответствует своя область видимости. Причем функции имеют глобальную область видимости.

Глобальные переменные отсутствуют, поэтому объявление переменных вне функций невозможно.

1.16 Семантические проверки

Семантическим анализатором языка SIV-2023 предусмотрены следующие проверки:

- Наличие блока `main`, точки входа в программу;
- Единственная точка входа в программу;
- Использование идентификаторов до их объявления;
- Переопределение идентификаторов;
- Соответствие параметров, передаваемых в функцию, с параметрами в объявлении функции;
- Соответствие типа возвращаемого значения с типом функции;
- Соответствие типов в выражениях;
- Превышение размера целочисленных и строковых литералов;
- Превышение длины лексемы;
- Соответствие операторов типам данных, для работы с которыми они предназначены.

1.17 Распределение оперативной памяти на этапе выполнения

Для запоминания промежуточных результатов в вычислении выражения используется стек. В сегмент констант записываются все литералы языка. В сегмент данных записываются все имена переменных.

1.18 Стандартная библиотека и её состав

В языке SIV-2023 предусмотрена стандартная библиотека, которая включает в себя набор стандартных функций, а также функций вывода в консоль. Функции, входящие в состав стандартной библиотеки приведены в таблице 1.7.

Таблица 1.7 – Функции стандартной библиотеки языка SIV-2023

| Прототип функции | Описание |
|----------------------------------|--|
| <code>_pow(int a, int b);</code> | Возводит число <code>a</code> в степень <code>b</code> и возвращает результат. |
| <code>_abs(int a);</code> | Берет абсолютное значение числа <code>a</code> и возвращает результат. |
| <code>noutl(int value);</code> | Выводит целочисленный идентификатор или литерал на консоль. |
| <code>soutl(str value);</code> | Выводит строковый идентификатор или литерал на консоль. |

Стандартная библиотека написана на языке C++, подключается на этапе компоновки. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

1.19 Ввод и вывод данных

В языке программирования SIV-2023 ввод данных не поддерживается.

Вывод данных на консоль осуществляется за счет оператора **print**. Использование данного оператора допускается только с идентификаторами или литералами.

Функции, управляющие выводом данных на консоль, реализованы на языке C++. На этапе генерации кода операторы вывода языка SIV-2023 заменяются на встроенные функции, находящиеся в стандартной библиотеке.

1.20 Точка входа

В языке SIV-2023 точкой входа в программу является функция **main**.

1.21 Преппроцессор

Преппроцессор в языке программирования SIV-2023 не предусмотрен.

1.22 Соглашение о вызовах

В языке SIV-2023 вызов функций происходит по стандартному соглашению о вызовах `stdcall`. Данное соглашение имеет следующие особенности:

- Все параметры функции передаются через стек;
- Освобождением памяти занимается вызываемый код;
- Параметры в стек заносятся справа налево.

1.23 Объектный код

Язык программирования SIV-2023 транслируется в язык ассемблера.

1.24 Классификация сообщений транслятора

Описание и классификация сообщений компилятора об ошибках приведено в таблице 1.8.

Таблица 1.8 – Описание ошибок транслятора языка SIV-2023

| Интервал кодов | Описание |
|----------------|-------------------------------------|
| 0-9 | Системные ошибки. |
| 10-19 | Ошибки параметров. |
| 20-29 | Ошибки файлов. |
| 110-129 | Ошибки лексического анализатора. |
| 130-149 | Ошибки семантического анализатора. |
| 600-609 | Ошибки синтаксического анализатора. |

1.25 Контрольный пример

Контрольный пример языка SIV-2023 представлен в приложении А.

2 Структура транслятора

2.1 Компоненты транслятора их назначение и принципы взаимодействия

Транслятор – это программа преобразующая исходный код на одном языке программирования в исходный код на другом языке.

Схема, поясняющая принцип работы транслятора, изображена на рисунке 2.1.

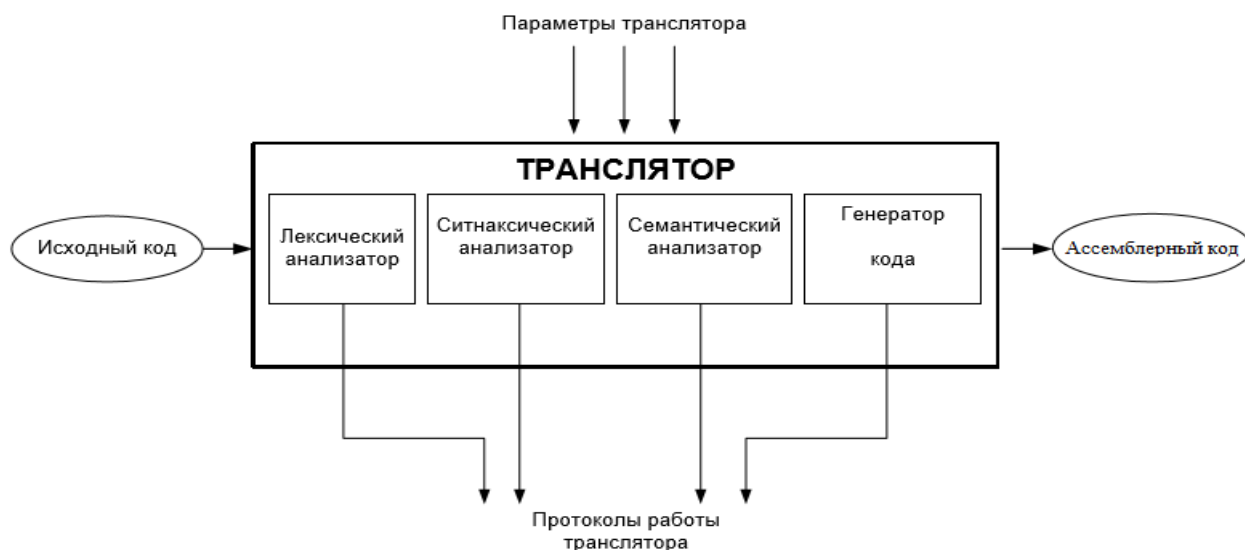


Рисунок 2.1 – Структура транслятора языка SIV-2023

Трансляция исходного кода в язык ассемблера разделена на 4 этапа:

- Лексический анализ
- Синтаксический анализ
- Семантический анализ
- Генерация кода

Этапы выполняются последовательно. У каждого этапа есть входные и выходные данные, которые последовательно передаются следующему компоненту транслятора.

Первой частью трансляции является лексический анализ. На вход лексического анализатора подается исходный код программы. В свою очередь лексический анализатор производит деление исходного кода программы на токены, которые затем идентифицируются и заменяются на лексемы.

На выходе лексического анализатора мы имеем две таблицы: таблицу лексем и таблицу идентификаторов.

Синтаксический анализ является второй частью работы транслятора. Синтаксический анализатор выполняет синтаксический анализ. Входом для синтаксического анализатора является таблица лексем и таблица идентификаторов. Выходом – дерево разбора.

Затем выполняется семантический анализ. Задача семантического анализатора: проверка соблюдения в исходной программе семантических правил

входного языка. Входом для семантического анализатора является таблица идентификаторов, таблица лексем и дерево разбора.

Последним этапом трансляции является генерация кода. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с кодом на языке ассемблера.

2.2 Перечень входных параметров транслятора

Входные параметры необходимы для формирования файлов с результатами работы транслятора. Входные параметры транслятора языка программирования SIV-2023 приведены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка SIV-2023

| Ключ и входной параметр | Описание | Значение по умолчанию |
|-------------------------|--|-----------------------|
| -in:<путь к файлу> | Текстовый файл с исходным кодом на языке SIV-2023. | Отсутствует |
| -out:<путь к файлу> | Выходной файл, содержащий исходный код на языке ассемблера. | Отсутствует |
| -log:<путь к файлу> | Файл с протоколом работы транслятора. | <имя файла in>.log |
| -an:<путь к файлу> | Файл с таблицей лексем, таблицей идентификаторов, деревом разбора синтаксического анализатора. | <имя файла>.an.txt |

2.3 Протоколы, формируемые транслятором

Протоколы, формируемые транслятором языка SIV-2023 приведены в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка SIV-2023

| Протокол | Описание |
|-----------------------------------|--|
| Файл, заданный параметром “-log:” | Содержит общую информацию о ходе выполнения трансляции: перечисление входных параметров, количество символов и строк, успех или ошибку по каждому этапу трансляции. В случае возникновения ошибки, в файл будет выведена информация об ошибке. |
| Файл, заданный параметром “-an:” | Содержит таблицу лексем, итог работы лексического анализа; таблицу идентификаторов, итог работы лексического анализа; дерево разбора, итог работы синтаксического анализатора. |

3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Лексический анализатор – это программа, преобразующая исходный текст программы, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Структурная схема лексического анализатора изображена на рисунке 3.1.

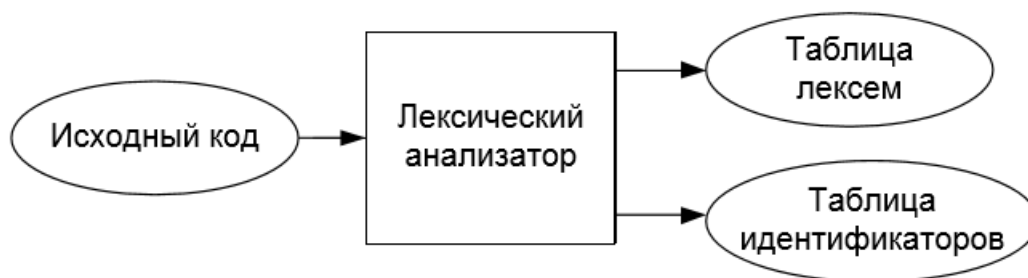


Рисунок 3.1 – Структурная схема лексического анализатора

Лексический анализ в языке SIV-2023 происходит в два этапа:

- Разбиение исходного кода программы на токены.
- Идентификация токенов и последующая их замена на лексемы.

Заполнение таблиц лексем и идентификаторов.

Входные данные: исходный код.

Результат работы: Таблица лексем и таблица идентификаторов.

3.2 Входные и выходные данные лексического анализатора

На вход лексического анализатора поступает исходный код программы на языке SIV-2023. Результатом работы лексического анализатора является таблица лексем и таблица идентификаторов.

3.3 Параметры лексического анализатора

Входным параметром лексического анализа является структура, полученная после чтения входного файла на этапе проверки исходного кода на допустимость символов.

3.4 Алгоритм лексического анализа

Алгоритм лексического анализа языка SIV-2023 заключается в следующем:

- Считываем исходный текст программы и делим его на токены, формируя структуру таблицы токенов;
- Слова из таблицы токенов пропускаем через графы, определяя тип лексем;

- Составляем таблицу лексем и идентификаторов.

Программный код, реализующий данный алгоритм, представлен в листинге 3.1.

```
struct LEX
{
    IT::IdTable idtable;
    LT::LexTable lextable;
    LEX();
    LEX(int lexTableSize, int idTableSize);
};
// Определяем тип лексемы
char LexType(Tokens::Token token);
// Заполнение таблиц
LEX FillingInTables(Tokens::TokenTable tokenTable);
// Поиск id в таблице
int SearchID(stack<int> areaOfVisibility, string id,
IT::IdTable& idTable);
// Поиск id функции в таблице
int SearchGlobalFunctionID(int globalAreaOfVisibility, string
id, IT::IdTable& idTable);
```

Листинг 3.1 – Программный код, реализующий лексический анализ

3.5 Контроль входных символов

Для проверки входных символов на допустимость была написана таблица входных символов, которая дублирует таблицу кодировки Windows-1251. Разрешенные символы в таблице помечены символом Т, запрещенные – F. Таблица контрольных символов языка SIV-2023 представлена в листинге приложения Б.

3.6 Удаление избыточных символов

Избыточный символ – это символ, отсутствие которого никак не влияет на исходный текст программы. В языке SIV-2023 символ пробела и табуляции являются избыточными символами. Их удаление предусмотрено на этапе разбиения исходного кода программы на токены.

Алгоритм удаления избыточных символов

Пока есть символы для чтения:

- Читаем очередной символ;
- Если символ является пробелом или табуляцией:
- Если идёт запись слова, пробел или табуляция являются символом сепаратором, сигнализирующем о начале или конце записи токена;
- Иначе символ пробела или табуляции пропускаются.

3.7 Перечень ключевых слов

Все ключевые слова языка SIV-2023, символы операций, сепараторы, соответствующие им лексемы и регулярные выражения приведены в таблице 3.1.

Таблица 3.1 – Все ключевые слова, сепараторы и т.д. языка SIV-2023

| Слово (токен) | Лексема | Описание |
|---------------|---------|--|
| int | d | Целочисленный тип. |
| str | s | Строковый тип. |
| bool | b | Логический тип. |
| идентификатор | i | Идентификатор любого типа языка. |
| литерал | l | Литерал любого типа языка. |
| if | c | Условный оператор и истинная ветвь условного оператора. |
| else | e | Ложная ветвь условного оператора. |
| function | f | Начало объявления функции. |
| new | n | Объявление переменной. |
| return | r | Выходи из функции и возврат значения. |
| print | o | Вывод данных в консоль. |
| entry | m | Главная функция (точка входа в программу). |
| pow | p | Стандартная функция, возведения в степень целочисленного литерала, языка. |
| abs | a | Стандартная функция, взятия абсолютного значения целочисленного литерала, языка. |
| , | , | Разделитель параметров функции. |
| ; | ; | Признак конца инструкции. |
| { | { | Начало тела функции. |
| } | } | Конец тела функции. |
| (| (| Начало перечислений параметров у функции, приоритет операций в выражениях. |
|) |) | Конец перечислений параметров у функции, приоритет операций в выражениях. |
| + | + | Арифметический оператор(сложение). |
| - | - | Арифметический оператор(вычитание). |
| * | * | Арифметический оператор(умножение). |
| / | / | Арифметический оператор(деление). |
| % | % | Арифметический оператор(остаток от деления). |
| > | < | Оператор сравнения(больше). |
| < | > | Оператор сравнения(меньше). |
| <= | [| Оператор сравнения(меньше или равно). |
| >= |] | Оператор сравнения(больше или равно). |
| == | & | Оператор сравнения(равенство). |
| != | ! | Оператор сравнения(неравенство). |
| = | = | Арифметический оператор(присваивание). |

Пример графов переходов конечных автоматов, соответствующих регулярным выражений представлены в листинге 3.2 и 3.3.

```

#define GRAPH_NEW \
    4, \
    FST::NODE(1, FST::RELATION('n'
, 1)), \
    FST::NODE(1, FST::RELATION('e'
, 2)), \
    FST::NODE(1, FST::RELATION('w'
, 3)), FST::NODE()
#define GRAPH_FUNCTION \
    9, \
    FST::NODE(1, FST::RELATION('f'
, 1)), \
    FST::NODE(1, FST::RELATION('u'
, 2)), \
    FST::NODE(1, FST::RELATION('n'
, 3)), \
    FST::NODE(1, FST::RELATION('c'
, 4)), \
    FST::NODE(1, FST::RELATION('t'
, 5)), \
    FST::NODE(1, FST::RELATION('i'
, 6)), \
    FST::NODE(1, FST::RELATION('o'
, 7)), \
    FST::NODE(1, FST::RELATION('n'
, 8)), FST::NODE()

#define GRAPH_RETURN \
    7, \
    FST::NODE(1, FST::RELATION('r'
, 1)), \
    FST::NODE(1, FST::RELATION('e'
, 2)), \
    FST::NODE(1, FST::RELATION('t'
, 3)), \
    FST::NODE(1, FST::RELATION('u'
, 4)), \
    FST::NODE(1, FST::RELATION('r'
, 5)), \
    FST::NODE(1, FST::RELATION('n'
, 6)), FST::NODE()
#define GRAPH_IF \
    3, \
    FST::NODE(1, FST::RELATION('i'
, 1)), \
    FST::NODE(1, FST::RELATION('f'
, 2)), FST::NODE()
#define GRAPH_ELSE \
    5, \
    FST::NODE(1, FST::RELATION('e'
, 1)), \
    FST::NODE(1, FST::RELATION('l'
, 2)), \
    FST::NODE(1, FST::RELATION('s'
, 3)), \
    FST::NODE(1, FST::RELATION('e'
, 4)), FST::NODE()

```

Листинг 3.2 и 3.3 – Фрагменты графов переходов языка SIV-2023

В листинге 3.4 представлен фрагмент кода функции на языке C++, реализующей алгоритм разбора входной цепочки в соответствии с графами переходов языка SIV-2023.

```

bool execute(FST& fst)
{
    short* rstates = new short[fst.nstates];
    memset(rstates, 0xff, sizeof(short) * fst.nstates);
    short lstring = strlen(fst.string);
    bool rc = true;
    for (short i = 0; i < lstring && rc; i++)
    {
        fst.position++; rc = step(fst, rstates);
    }
    delete[] rstates;
    return (rc?(fst.rstates[fst.nstates-1]==lstring):rc);
};

```

Листинг 3.4 – Функция разбора входной цепочки на языке SIV-2023

3.8 Основные структуры данных

Основные структуры данных лексического анализатора: таблица токенов, таблица лексем и таблица идентификаторов.

В листинге 3.5 представлена структура токена и таблицы токенов.

```
// Структура токена
struct Token
{
    char token[258];
    int length;
    int line;
    int linePosition;
};
// Структура таблицы токенов
struct TokenTable
{
    int maxsize;
    int size;
    Token* table;
};
```

Листинг 3.5 – Структура токена и таблицы токенов языка SIV-2023

Структура TokenTable представляет таблицу токенов, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, а table – указатель на строку таблицы.

Структура Token представляет строку таблицы TokenTable, где в массив token записывается слово, length – длина слова, line – номер строки в исходном тексте программы, а linePosition – позицию в строке.

Реализация структуры таблицы лексем представлена в листинге 3.6.

```
struct Entry
{
    char lexema;
    int sn;
    int idxTI;
};

struct LexTable
{
    int maxsize;
    int size;
    Entry* table;
};
```

Листинг 3.6 – Структура таблицы лексем языка SIV-2023

Структура LexTable представляет таблицу лексем, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, а table – указатель на строку таблицы.

Структура `Entry` представляет строку таблицы `LexTable`, где `lexeme` представляет лексему, `sn` – номер строки в исходном тексте программы, а `idxTI` – номер в таблице идентификаторов.

Реализация структуры таблицы идентификаторов представлена в листинге 3.7.

```
struct Entry
{
    int          idxfirstLE;
    std::string id;
    IDDATATYPE  idDataType;
    IDTYPE      idType;
    struct
    {
        int vshort;
        struct
        {
            int len;
            std::string str;
        } vstr;
    } value;
};

struct IdTable
{
    int maxsize;
    int size;
    Entry* table;
};
```

Листинг 3.7 – Структура таблицы идентификаторов языка SIV-2023

Структура `IdTable` представляет собой таблицу идентификаторов, где `maxsize` – число, равное максимальному размеру таблицы, `size` – текущий размер таблицы, а `table` – указатель на строку таблицы.

Структура `Entry` представляет строку таблицы `IdTable`, где переменная `idxfirstLE` – индекс первого вхождения в таблицу лексем, `id` – идентификатор, `idDataType` – тип данных, `idType` – тип идентификатора, `vshort` – целочисленное значение, `len` – длина строку, `str` – строка.

3.9 Структура и перечень сообщений лексического анализатора

При возникновении ошибки в лексическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте, позиция в строке.

Перечень сообщений лексического анализатора приведен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора языка SIV-2023

| Номер ошибки | Пояснительный текст |
|--------------|---|
| 110 | Недопустимый символ в исходном файле (-in:) |
| 111 | Превышена емкость таблицы лексем |

Окончание таблицы 3.2

| | |
|-----|---|
| 112 | Превышено количество строк в таблице лексем |
| 113 | В таблице лексем отсутствует строка с заданным номером |
| 114 | Превышена емкость таблицы идентификаторов |
| 115 | Превышено количество строк в таблице идентификаторов |
| 116 | В таблице идентификаторов отсутствует строка с заданным номером |
| 117 | Не удалось определить тип лексемы |
| 118 | Превышена емкость таблицы токенов |
| 119 | Превышено количество токенов в таблице токенов |
| 120 | Ошибка с разбиением исходного текста на токены |
| 121 | Ошибка с разбором строкового литерала |

3.10 Принцип обработки ошибок

При обнаружении ошибки в исходном коде программы лексический анализатор формирует сообщение об ошибке и выводит его в консоль и записывает в файл с протоколом работы, заданный параметром `-log:`.

3.11 Контрольный пример

Результат работы лексического анализатора, полученный при выполнении контрольного примера, а именно таблица лексем и таблица идентификаторов, представлен в приложении В.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализ языка SIV-2023 выполняется после завершения работы лексического анализатора. Синтаксический анализатор предназначен для сопоставления последовательности лексем языка SIV-2023 с его формальной грамматикой.

Входными данными являются: таблица лексем и таблица идентификаторов. Результатом работы является дерево разбора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе языка SIV-2023 используется грамматика типа 2 иерархии Хомского (Контекстно-свободная грамматика) $G = \{N, T, P, S\}$, где

- N – конечный алфавит нетерминальных символов, приведенный в первом столбце таблицы 4.1;
- T – конечный алфавит терминальных символов;
- P – конечное множество правил порождения;
- S – начальный нетерминал грамматики G .

Контекстно-свободная грамматика G имеет нормальную форму Грейбах, если она не является леворекурсивной и правила P имеют вид:

- $A \rightarrow a\alpha$, где $a \in T$, $\alpha \in N^*$;
- $S \rightarrow \lambda$, где $S \in N$ – начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Алгоритм преобразования грамматик в нормальную форму Грейбах:

- Исключить недостижимые символы из грамматики;
- Исключить лямбда-правила из грамматики;
- Исключить цепные правила.

Перечень и описание терминальных, нетерминальных символов и правил языка приведен в таблице 4.1.

Таблица 4.1 – Описание правил, составляющих грамматику языка SIV-2023

| Нетерминальный символ | Цепочки правил | Описание |
|-----------------------|--|---|
| S | $m\{N\};$ $m\{N\};S$ $d\{f\}\{N\};S$ $d\{f\}\{N\};$ $b\{f\}\{N\};S$ $b\{f\}\{N\};$ $s\{f\}\{N\};S$ $s\{f\}\{N\};$ | Правила, порождающие главную функцию main и глобальные функции. |

Продолжение таблицы 4.1

| Нетерминальный символ | Цепочки правил | Описание |
|-----------------------|---|--|
| | $d\text{fi}(F)\{N\};$ $b\text{fi}(F)\{N\};S$ $b\text{fi}(F)\{N\};$ $s\text{fi}(F)\{N\};S$ $s\text{fi}(F)\{N\};$ | |
| N | $n\text{Ti};N$ $n\text{Ti}=E;N$ $n\text{Ti};$ $n\text{Ti}=E;$ $i=E;$ $i=E;N$ $oL;N$ $oL;$ $p(W);N$ $p(W);$ $a(W);N$ $a(W);$ $i(W);N$ $i(W);$ $i();N$ $i();$ $cQ\{N\}N$ $cQ\{N\}$ $cQ\{N\}e\{N\}$ $cQ\{N\}e\{N\}N$ $rL;N$ $rL;$ | Правила, порождающие конструкции в функциях. |
| F | d_i s_i b_i d_i, F s_i, F b_i, F | Правила, порождающие параметры объявления функции. |
| E | i l (E) $i(W)$ $i()$ $p(W)$ $a(W)$ iM lM | Правила, порождающие выражения. |

Продолжение таблицы 4.1

| Нетерминальный символ | Цепочки правил | Описание |
|-----------------------|---|---|
| | (E)M i(W)M i()M p(W)M a(W)M | |
| W | i l i,W l,W i() i(),W i(W) i(W),W p(W) p(W),W a(W) a(W),W | Правила, порождающие параметры вызываемой функции. |
| M | +E +EM -E -EM /E /EM *E *EM %E %EM !L &L >L <L]L [L | Правила, порождающие операторы. |
| Q | (L<L) (L>L) (L!L) (L&L) (L]L) (L[L) | Правила, порождающие условия в условных конструкциях. |
| L | L i | Правила, порождающие литерал и идентификатор |

Окончание таблицы 4.1

| Нетерминальный символ | Цепочки правил | Описание |
|-----------------------|----------------|-----------------------------------|
| T | d s b | Правила, порождающие типы данных. |

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку, описание которой приведено ниже.

$M = \{Q, V, Z, \delta, q_0, z_0, F\}$, где

- Q – множество состояний;
- V – алфавит входных символов;
- Z – специальный алфавит магазинных символов;
- δ – функция переходов автомата;
- $q_0 \in Q$ – начальное состояние автомата;
- $z_0 \in Z$ – начальное состояние магазина (маркер дна);
- $F \subseteq Q$ – Множество конечных состояний.

Схема конечного автомата с магазинной памятью изображена на рисунке 4.1.

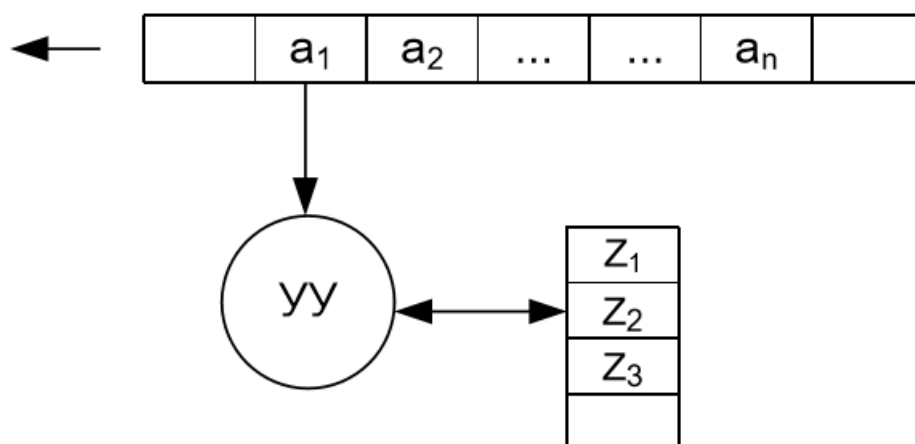


Рисунок 4.1 – Схема конечного автомата с магазинной памятью

Алгоритм работы конечного автомата с магазинной памятью:

- Состояние автомата $(q, \alpha, z\beta)$;
- Читает символ a находящийся под головкой (сдвигает ленту);
- Не читает ничего (читает λ , не сдвигает ленту);
- Из δ определяет новое состояние q' , если $(q', \gamma) \in \delta(q, a, z)$ или $(q', \gamma) \in \delta(q, \lambda, z)$;
- Читает верхний (в стеке) символ z и записывает цепочку γ т.к. $(q', \gamma) \in \delta(q, a, z)$, при этом, если $\gamma = \lambda$, то верхний символ магазина просто удаляется;
- Работа автомата заканчивается (q, λ, λ) .

Результат, демонстрирующий успешный разбор цепочки из контрольного примера, приведен в приложении Г.

4.4 Основные структуры данных

Программный код основных структур данных на языке C++, описывающих контекстно-свободную грамматику, представлен в приложении Д.

4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора языка SIV-2023:

- 1) В магазин заносится стартовый символ;
- 2) Формируется входная лента, полученная из таблицы лексем;
- 3) Нетерминальный символ раскрывается, согласно правилам, и записывается в магазин;
- 4) Если терминал на вершине стека и в начале ленты совпадают, то данный терминал удаляется из входной ленты. Иначе возвращается в предыдущее состояние и выбирает другую цепочку нетерминала;
- 5) Если в магазине встречается нетерминал, переход к пункту 3;
- 6) Если достигнуто дно стека и входная цепочка пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

Обобщенная блок-схема алгоритма синтаксического анализа изображена на рисунке 4.2.

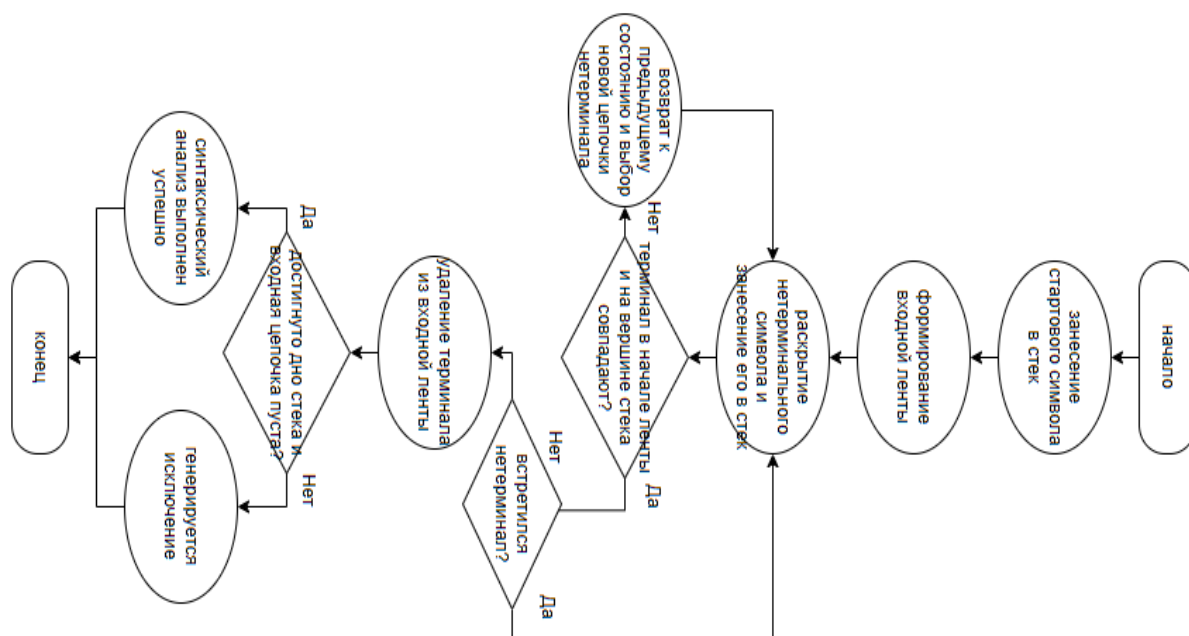


Рисунок 4.2 – Блок-схема алгоритма синтаксического анализа

4.6 Параметры синтаксического анализатора

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются правила разбора, которые выводятся на консоль.

4.7 Структура и перечень сообщений синтаксического анализатора

При возникновении ошибки в синтаксическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте.

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 – Перечень сообщений синтаксического анализатора языка SIV-2023

| Номер ошибки | Пояснительный текст |
|--------------|--|
| 600 | Неверная структура программы |
| 601 | Ошибка в конструкции блока кода |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах функции |
| 604 | Ошибка в параметрах вызываемой функции |
| 605 | Ошибочный оператор |
| 606 | Ошибка в условной конструкции |
| 607 | Ошибка типа |

4.8 Принцип обработки ошибок

Обработка ошибок происходит по следующему алгоритму:

- Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
- При невозможности подбора подходящей цепочки, то генерируется соответствующая ошибка.
- Ошибки записываются в общую структуру ошибок.
- В случае нахождения ошибки, после всей процедуры трассировки в протокол и на консоль будет выведено диагностическое сообщение в файл протокола .log.

4.9 Контрольный пример

Результат работы синтаксического анализатора, полученный при выполнении контрольного примера, а именно дерево разбора, представлен в приложении Г.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализ языка SIV-2023 выполняется после выполнения лексического и синтаксического анализа. Несмотря на это, некоторые семантические проверки выполняются на этапе лексического анализа. На вход семантического анализатора подаются таблица лексем и таблица идентификаторов. Схема семантического анализатора представлена на рисунке 5.1.

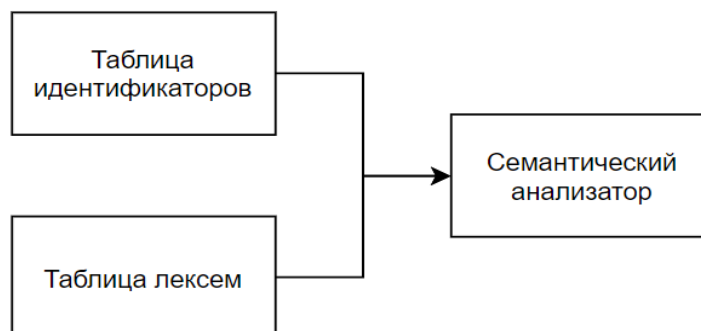


Рисунок 5.1 – Схема семантического анализатора языка SIV-2023

5.2 Функции семантического анализатора

Семантические проверки языка SIV-2023 с указанием фаз их выполнения приведены в таблице 5.1.

Таблица 5.1 – Семантические проверки языка SIV-2023

| Семантическая проверка | Фаза выполнения |
|--|----------------------|
| Превышение длины строки | Лексический анализ |
| Превышение целочисленного значения | Лексический анализ |
| Повторная реализация функции main | Лексический анализ |
| Превышение длины лексемы | Лексический анализ |
| Наличие точки входа в программу | Лексический анализ |
| Объявление идентификатора перед использованием | Семантический анализ |
| Повторное объявление идентификатора | Семантический анализ |
| Соответствие типов в выражении | Семантический анализ |
| Количество параметров функции | Семантический анализ |
| Соответствие параметров объявленной и вызываемой функции | Семантический анализ |
| Соответствие параметров встроенной функции | Семантический анализ |

Окончание таблицы 5.1

| Семантическая проверка | Фаза выполнения |
|---|----------------------|
| Проверка левосторонних выражений | Семантический анализ |
| Повторная реализация функции | Семантический анализ |
| Соответствие типа возвращаемого значения типу функции | Семантический анализ |
| Перегрузка оператора для работы со строками | Семантический анализ |

5.3 Структура и перечень сообщений семантического анализатора

Структура и текст сообщений семантического анализатора приведены в рисунке 5.2.

Таблица 5.2 – Перечень сообщений семантического анализатора языка SIV-2023

| Номер ошибки | Пояснительный текст |
|--------------|---|
| 130 | Превышена длина строки в 255 символов |
| 131 | Функция main уже имеет реализацию |
| 132 | Превышена длина лексемы |
| 133 | Превышено значение целочисленного литерала (2 byte) |
| 134 | Не найдена точка входа в программу (main) |
| 135 | Идентификатор с таким именем не найден |
| 136 | Повторное объявление идентификатора |
| 137 | Несоответствие типов в выражении |
| 138 | Слишком много параметров в функции |
| 139 | Превышено количество функций |
| 140 | Несоответствие параметров объявленной и вызываемой функций |
| 141 | Несоответствие параметров встроенной функции |
| 142 | Левостороннее выражение не является идентификатором и не должно являться функцией |
| 143 | Данная функция уже имеет реализацию |
| 144 | В вызове функции отсутствуют () |
| 145 | Тип возвращаемого значения не соответствует типу функции |
| 146 | Оператор не предназначен для работы со строками |
| 147 | В функции отсутствует возвращаемое значение |
| 148 | Деление на 0 недопустимо |
| 149 | Недопустимый строковый литерал |

5.4 Принцип обработки ошибок

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибке и выводит его на консоль и в файл с протоколом работы, заданный параметром `-log:`.

5.5 Контрольный пример

Контрольный пример для демонстрации ошибок, диагностируемых семантическим анализатором вместе с отчетом выданных сообщений представлен в приложении А.

6 Вычисление выражений

6.1 Выражение, допускаемые языком

В языке SIV-2023 допускаются выражения, применимые к целочисленным типам данных. Допускается использование функций, возвращающих целочисленное значение, в выражениях. Операции и их приоритетность приведены в таблице 6.1.

Таблица 6.1 – Операции и их приоритетность языка SIV-2023

| Приоритет | Операция |
|-----------|----------|
| 0 | (|
| 0 |) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | * |
| 3 | / |
| 3 | % |

Примеры выражений из контрольного примера:

5*(pow(1011В,2)-В2Н)+23О и т.п.

6.2 Польская запись и принцип её построения

Язык SIV-2023 транслируется в язык ассемблера, в котором все вычисления производятся через стек. Преобразование исходных выражений в обратную польскую запись, упрощает генерацию кода вычисления выражений в язык ассемблера. Алгоритм построения польской записи приведен ниже:

Пока есть символы для чтения:

- Читаем очередной символ;
- Если символ является числом, добавляем его к выходной строке;
- Если символ является функцией, помещаем его в стек;
- Если символ является открывающей скобкой, помещаем его в стек;
- Если символ является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется.

– Если символ является операцией:

- Пока операция на вершине стека приоритетнее или пока на вершине стека функция;
- Помещаем операцию в стек.

Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку.

Пример построения обратной польской записи: `lllpl-*l+`.

6.3 Программная реализация обработки выражений

Фрагмент кода, реализующего преобразование выражений в обратный польский формат представлен в листинге 6.1.

```

case LEX_ID:
{
    if(idtable.table[lextable.table[lextable_pos].idxTI].idType
== IT::IDTYPE::F)
    {
        stack.push(lextable.table[lextable_pos]);
        continue;
    }
    queue.push(lextable.table[lextable_pos]);
    continue;
}
case LEX_LITERAL:
{
    queue.push(lextable.table[lextable_pos]);
    continue;
}
case LEX_LEFTHESIS:
{
    stack.push(lextable.table[lextable_pos]);
    continue;
}
case LEX_RIGHTHESIS:
{
    if (!stack.empty())
    {
        while (stack.top().lexema != LEX_LEFTHESIS)
        {
            queue.push(stack.top());
            stack.pop();
        }
        stack.pop();
    }
    continue;
}

```

Листинг 6.1 – Фрагмент кода, реализующего преобразование выражений

При встрече лексемы идентификатора, идет проверка на функцию. Если идентификатор является функцией, идентификатор помещается в стек. Иначе идентификатор помещается в выходную строку.

При встрече литерала, литерал помещается в выходную строку.

При встрече открывающей скобки, скобка кладется в стек.

При встрече закрывающей скобки, идет извлечение элементов из стека, пока не будет достигнута открывающая скобка.

6.4 Контрольный пример

В приложении В представлена обновленная таблица лексем, с выражениями, приведенными к обратной польской записи.

7 Генерация кода

7.1 Структура генератора кода

Трансляция с языка SIV-2023 производится в язык ассемблера. Структура генератора кода изображена на рисунке 7.1.

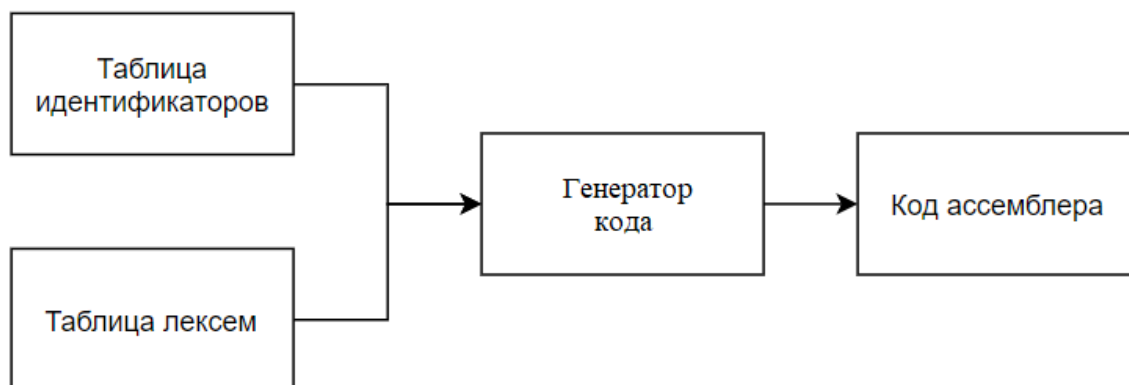


Рисунок 7.1 – Структура генератора кода

Генератор кода последовательно проходит таблицу лексем, при необходимости обращаясь к таблице идентификаторов. В зависимости от пройденных лексем выполняется генерация кода ассемблера.

7.2 Представление типов данных в оперативной памяти

Плоская модель памяти (flat): приложению для кода и данных предоставляется один непрерывный сегмент. Данный сегмент в свою очередь разбит на области:

- .STACK – стек;
- .CONST – константы;
- .DATA – переменные;
- .CODE – код.

Соответствие типов данных в исходном языке программирования SIV-2023 типам целевого языка приведены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка SIV-2023 и языка ассемблера

| Тип идентификатора языка SIV-2023 | Тип идентификатора ассемблера | Пояснение |
|-----------------------------------|-------------------------------|------------------------------------|
| int | SDWORD | Хранит знаковый целочисленный тип. |
| str | DWORD | Хранит указатель на начало строки. |
| bool | DWORD | Хранит логическое значение. |

7.3 Статическая библиотека

Функции, входящие в состав статической библиотеки языка SIV-2023, приведены в таблице 1.7.

Статическая библиотека написана на языке C++. Путь к статической библиотеке указан в Свойства проекта > Компонент > Командная строка. Библиотека подключается на этапе компоновки.

Таблица 1.7 – Функции стандартной библиотеки языка SIV-2023

| Прототип функции | Описание |
|----------------------------------|--|
| <code>_pow(int a, int b);</code> | Возводит число <code>a</code> в степень <code>b</code> и возвращает результат. |
| <code>_abs(int a);</code> | Берет абсолютное значение числа <code>a</code> и возвращает результат. |
| <code>noutl(int value);</code> | Выводит целочисленный идентификатор или литерал на консоль. |
| <code>soutl(str value);</code> | Выводит строковый идентификатор или литерал на консоль. |

7.4 Особенности алгоритма генерации кода

Обобщенная блок-схема алгоритма генерации кода языка ассемблера изображена на рисунке 7.2.

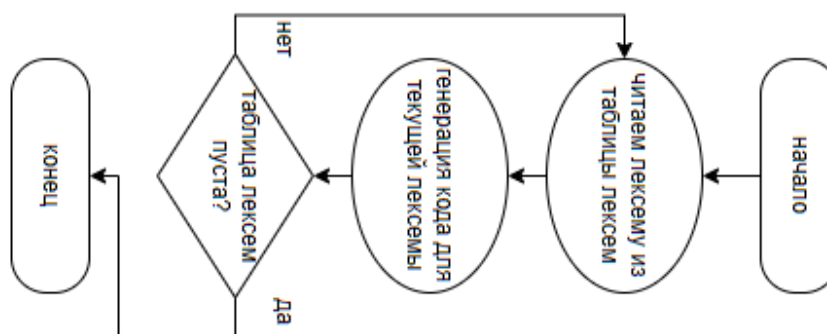


Рисунок 7.2 – Блок-схема алгоритма генерации кода в язык ассемблера

Пока таблица лексем не пуста, читаем лексему:

- Если для лексемы есть код генерации, генерируем код;
- Иначе читаем следующую лексему, пока таблица лексем не будет пуста.

Генерация кода строится с помощью блока макросов и функции `void CodeGeneration(LA::LEX lex, wchar_t outfile[])`.

В листинге 7.1 представлен блок макросов, используемый при генерации кода.

```

#define START \
".586\n"\
".model flat, stdcall\n"\
"includelib libucrt.lib\n"\
"includelib kernel32.lib\n"

#define PROTOTYPES \
"\nExitProcess PROTO:DWORD "\
"\nSYSPAUSE PROTO "\
"\nsoutl PROTO : BYTE "\
"\nnoutl PROTO : SDWORD "\
"\n_pow PROTO : SDWORD, : SDWORD "\
"\n_abs PROTO : SDWORD "\
"\n\n.STACK 4096\n\n"

#define FINISH \
"\tcall SYSPAUSE"\
"\n\tpush 0"\
"\n\tcall ExitProcess"\
"\nSOMETHINGWRONG::"\
"\n\tpush offset null_division"\
"\n\tcall soutl"\
"\njmp THEEND"\
"\noverflow::"\
"\n\tpush offset OVER_FLOW"\
"\n\tcall soutl"\
"\nTHEEND::"\
"\n\tcall SYSPAUSE"\
"\n\tpush -1"\
"\n\tcall ExitProcess"\
"\nmain ENDP\nend main"

```

Листинг 7.1 – Использование макросов при генерации

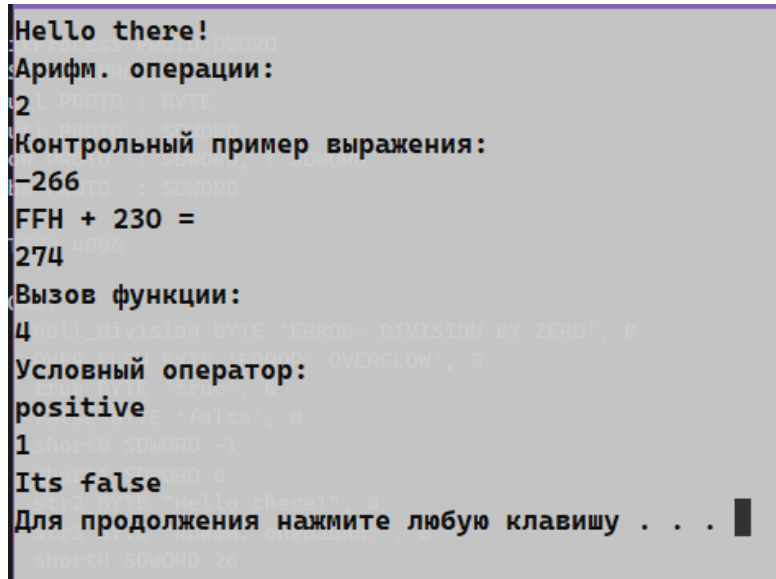
7.5 Параметры, управляющие генерацией кода

На вход генератору кода поступают таблицы лексем и идентификаторов. Результат работы генератора кода выводится в файл с расширением .asm

7.6 Контрольный пример

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

На рисунке 7.3 приведен результат работы контрольного примера.



```

Hello there!
Арифм. операции:
2
Контрольный пример выражения:
-266
FFH + 230 =
274
Вызов функции:
4
Условный оператор:
positive
1
Its false
Для продолжения нажмите любую клавишу . . .

```

Рисунок 7.3 – Результат работы программы на языке SIV-2023

8 Тестирование транслятора

8.1 Общие положения

Тесты предназначены для выявления ошибок в работе компилятора и последующего их устранения. Ошибки были выявлены как на раннем этапе разработки компилятора, так и на позднем.

После возникновения ошибки работа транслятора прекращается, поскольку ошибка на одном этапе трансляции может вызвать ошибки на последующих этапах (за исключением синтаксического анализатора). Текст с номером и сообщением об ошибке будет выведен в файл протокола и консоли.

8.2 Результаты тестирования

Описание тестовых наборов, демонстрирующих проверки на разных этапах трансляции, приведено в таблице 8.1.

Таблица 8.1 – Описание тестовых наборов языка SIV-2023

| Фрагмент исходного кода | Диагностическое сообщение |
|---|--|
| Допустимость символов | |
| <code>new int gf#;</code> | Ошибка 110: Ошибка лексического анализатора: Недопустимый символ в исходном файле (-in:), строка 3, позиция 12 |
| Лексический анализ | |
| <code>24rgt str stroka;</code> | Ошибка 117: Ошибка лексического анализатора: Не удалось определить тип лексемы, строка 3, позиция 1 |
| Лексический анализ | |
| <code>main{ new str str2 = "qwerty; };</code> | Ошибка 121: Ошибка лексического анализатора: Ошибка с разбором строкового литерала, строка 2, позиция 1 |
| Синтаксический анализ | |
| <code>new main { }</code> | Ошибка 600: Ошибка синтаксического анализа: Неверная структура программы, строка 1, позиция 1 |
| <code>if (5 > 6) {</code> | Ошибка 601: Ошибка синтаксического анализа: Ошибка в конструкции блока кода, строка 3, позиция 1 |
| <code>new int x = 5 ++ 4;</code> | Ошибка 602: Ошибка синтаксического анализа: Ошибка в выражении, строка 2, позиция 1 |
| <code>str function str1(str x,) { };</code> | Ошибка 603: Ошибка синтаксического анализа: Ошибка в параметрах функции, строка 1, позиция 1 |

Продолжение таблицы 8.1

| Фрагмент исходного кода | Диагностическое сообщение |
|--|--|
| Синтаксический анализ | |
| <code>new int x1 = pow(2,);</code> | Ошибка 604: Ошибка синтаксического анализа: Ошибка в параметрах вызываемой функции, строка 2, позиция 1 |
| <code>new int x = (54 – 5;</code> | Ошибка 605: Ошибка синтаксического анализа: Ошибочный оператор, строка 2, позиция 1 |
| <code>if (5 > d + 2)</code> | Ошибка 606: Ошибка синтаксического анализа: Ошибка в условной конструкции, строка 3, позиция 1 |
| <code>main{ new ind; };</code> | Ошибка 607: Ошибка синтаксического анализа: Ошибка типа, строка 2, позиция 1 |
| Семантический анализ | |
| <code>main{new str s;}; main{new int i;};</code> | Ошибка 131: Ошибка семантического анализа: Функция main уже имеет реализацию, строка 2, позиция 1 |
| <code>new int k = 32800;</code> | Ошибка 133: Ошибка семантического анализа: Превышено значение целочисленного литерала (2 byte), строка 2, позиция 14 |
| <code>int function f1(){return 0;};</code> | Ошибка 134: Ошибка семантического анализа: Не найдена точка входа в программу (main), строка -1, позиция -1 |
| <code>main{ x = 5; };</code> | Ошибка 135: Ошибка семантического анализа: Идентификатор с таким именем не найден, строка 2, позиция 2 |
| <code>main{ new int r; new int r; };</code> | Ошибка 136: Ошибка семантического анализа: Повторное объявление идентификатора, строка 3, позиция 13 |
| <code>main{ new int a = 5; new str b = “stroka”; a = a + b; };</code> | Ошибка 137: Ошибка семантического анализа: Несоответствие типов в выражении, строка 4, позиция 1 |
| <code>int function f1(int x){ return x; }; entry{ f1("str"); };</code> | Ошибка 140: Ошибка семантического анализа: Несоответствие параметров объявленной и вызываемой функций, строка 6, позиция 1 |
| <code>main{ new int i = pow(2,3,4); };</code> | Ошибка 141: Ошибка семантического анализа: Несоответствие параметров встроенной функции, строка 2, позиция 1 |

Окончание таблицы 8.1

| Фрагмент исходного кода | Диагностическое сообщение |
|---|---|
| Семантический анализ | |
| <pre>int function f2(str x){...}; main{ f2 = 5; };</pre> | Ошибка 142: Ошибка семантического анализа: Левостороннее выражение не является идентификатором и не должно являться функцией, строка 7, позиция 1 |
| <pre>int function f3(int a){ ... }; str function f3(str b){ ... };</pre> | Ошибка 143: Ошибка семантического анализа: Данная функция уже имеет реализацию, строка 5, позиция 14 |
| <pre>int function f4(int v){ ... }; main{ new int c = f4; };</pre> | Ошибка 144: Ошибка семантического анализа: В вызове функции отсутствуют (), строка 15, позиция 1 |
| <pre>int function f5(int v){ new str res = "qwerty"; return res; };</pre> | Ошибка 145: Ошибка семантического анализа: Тип возвращаемого значения не соответствует типу функции, строка 4, позиция 1 |
| <pre>main{ new str s = "qwe" + "asd"; };</pre> | Ошибка 146: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 3, позиция 1 |
| <pre>int function f6(){ new int s; };</pre> | Ошибка 147: Ошибка семантического анализа: В функции отсутствует возвращаемое значение, строка 1, позиция 1 |
| <pre>main{ new int a = 3/0; };</pre> | Ошибка 148: Ошибка семантического анализа: Деление на 0 недопустимо, строка 3, позиция 1 |
| <pre>main{ new str s = ""; };</pre> | Ошибка 149: Ошибка семантического анализа: Недопустимый строковый литерал, строка 3, позиция 13 |

Заключение

В ходе выполнения курсовой работы был разработан компилятор для языка SIV-2023. Выполнены минимальные требования к курсовому проекту, а также ряд дополнений.

Язык SIV-2023 включает в себя:

- 3 типа данных;
- оператор вывода данных в консоль;
- вызов функций из стандартной библиотеки;
- 5 арифметических операторов для вычисления выражений;
- 6 операций сравнения;
- условный оператор if-else;

Работа по разработке компилятора позволила получить необходимое представление о структурах и процессах, использующихся при построении компиляторов, также были изучены основы теории формальных грамматик и основы общей теории компиляторов.

Список использованных литературных источников

- 1) Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 с.
- 2) Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – М.: Вильямс, 2005. – 912с.
- 3) Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс, 2003. – 768с.
- 4) Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с
- 5) Курс лекций по КПО / Наркевич А.С

Приложение А

Контрольный пример

```

int function func(int a, int b){
    new int res = -1;
    if(a<0)
    {
        res = abs(a);
    }
    else
    {
        res = pow(a, b);
    }
    return res;
};

str function setstring(str s){
    return s;
};

main
{
    // There are comments here
    new str str1 = setstring("Hello there!");
    print str1;

    print "Арифм. операции:";
    new int i = 26/13;
    print i;
    print "Контрольный пример выражения: ";
    i = 5*(pow(1011B,2)-B2H)+230;          // 5*(pow(11,2)-178)+19 = -266
    print i;
    i = FFH + 230;
    print "FFH + 230 = ";
    print i;

    print "Вызов функции:";
    new int num = func(-4, 3);
    print num;

    print "Условный оператор:";
    new int c;
    if(i >= 0)
    {
        print "positive";
        c = 1;
    }
    else
    {
        print "negative";
        c = -1;
    }
    print c;
}

```

```

new bool flag;
if(flag == true)
{
    print "Its true";
}
else
{
    print "Its false";
}
};

```

Контрольный пример, содержащий 3 семантические ошибки

```

int function func(int a, int b){
    new int res = -1;
    if(a<0)
    {
        res = abs(a);
    }
    else
    {
        res = pow(a, b);
    }
    return res;
};

str function func(str s){
    return s;
};

main
{
    // There are comments here
    new str str1 = setstring("Hello there!");
    new str str1;
    print str1;

    new str str2 = "a"+5;

    print "Арифм. операции:";
    new int i = 26/13;
    print i;
    print "Контрольный пример выражения: ";
    i = 5*(pow(1011B,2)-B2H)+230;          // 5*(pow(11,2)-178)+19 = -266
    print i;
    i = FFH + 230;
    print "FFH + 230 = ";
    print i;

    print "Вызов функции:";
    new int num = func(-4, 3);
    print num;

```

```

print "Условный оператор:";
new int c;
if(i >= 0)
{
    print "positive";
    c = 1;
}
else
{
    print "negative";
    c = -1;
}
print c;

new bool flag;
if(flag == true)
{
    print "Its true";
}
else
{
    print "Its false";
}
};

```

Допущенные семантические ошибки:

- Ошибка 143: Ошибка семантического анализа: Данная функция уже имеет реализацию, строка 14, позиция 14
- Ошибка 136: Ошибка семантического анализа: Повторное объявление идентификатора, строка 21, позиция 10
- Ошибка 146: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 21, позиция 1

Приложение Б

Таблица допустимых и запрещенных символов

```

#define IN_CODE_TABLE {\
    /*00*/    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*16*/    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*32*/    IN::T, IN::T, IN::T, IN::F, IN::F, IN::T, IN::F,
IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*48*/    IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*64*/    IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*80*/    IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::I, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*96*/    IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*112*/   IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::T, IN::F, IN::F, \
    \
    /*128*/   IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*144*/   IN::F, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*160*/   IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*176*/   IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,
IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    /*192*/   IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*208*/   IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*224*/   IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
    /*240*/   IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,
IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, \
}

```

Приложение В

Таблица идентификаторов

| № | Name | Type | Data type | First in LT | Value |
|----|------------|-----------|-----------|-------------|----------------------------------|
| 0 | func1 | Function | Int | 2 | - |
| 1 | a1 | Parameter | Int | 5 | - |
| 2 | b1 | Parameter | Int | 8 | - |
| 3 | res10 | Variable | Int | 13 | - |
| 4 | short0 | Literal | Int | 15 | -1 |
| 5 | short1 | Literal | Int | 21 | 0 |
| 6 | setstring2 | Function | Str | 51 | - |
| 7 | s2 | Parameter | Str | 54 | - |
| 8 | stroka63 | Variable | Str | 66 | - |
| 9 | str163 | Variable | Str | 70 | - |
| 10 | str2 | Literal | Str | 74 | "Hello there!" |
| 11 | str3 | Literal | Str | 81 | "Арифм. операции:" |
| 12 | i63 | Variable | Int | 85 | - |
| 13 | short4 | Literal | Int | 87 | 26 |
| 14 | short5 | Literal | Int | 89 | 13 |
| 15 | str6 | Literal | Str | 95 | "Контрольный пример выражения: " |
| 16 | short7 | Literal | Int | 99 | 5 |
| 17 | short8 | Literal | Int | 104 | 11 |
| 18 | short9 | Literal | Int | 106 | 2 |
| 19 | short10 | Literal | Int | 109 | 178 |
| 20 | short11 | Literal | Int | 112 | 19 |
| 21 | short12 | Literal | Int | 119 | 255 |
| 22 | short13 | Literal | Int | 121 | 19 |
| 23 | str14 | Literal | Str | 124 | "FFH + 23O = " |
| 24 | str15 | Literal | Str | 130 | "Вызов функции:" |
| 25 | num63 | Variable | Int | 134 | - |
| 26 | short16 | Literal | Int | 138 | -4 |
| 27 | short17 | Literal | Int | 140 | 3 |
| 28 | str18 | Literal | Str | 147 | "Условный оператор:" |
| 29 | c63 | Variable | Int | 151 | - |
| 30 | short19 | Literal | Int | 157 | 0 |
| 31 | str20 | Literal | Str | 161 | "positive" |
| 32 | short21 | Literal | Int | 165 | 1 |
| 33 | str22 | Literal | Str | 171 | "negative" |
| 34 | short23 | Literal | Int | 175 | -1 |
| 35 | flag63 | Variable | Bool | 183 | - |

| | | | | | | |
|----|--------|---------|------|-----|-------------|--|
| 36 | bool24 | Literal | Bool | 189 | true | |
| 37 | str25 | Literal | Str | 193 | "Its true" | |
| 38 | str26 | Literal | Str | 199 | "Its false" | |

Таблица лексем (до построения ПОЛИЗ)

| =====LEX | | | | | | | | | |
|---------------------------|------|--------|------------|--|-----|----|---|----|--|
| TABLE===== | | | | | | | | | |
| +-----+-----+-----+-----+ | | | | | | | | | |
| № | Line | Lexema | ID from IT | | | | | | |
| 0 | 0 | d | - | | 35 | 8 | = | - | |
| 1 | 0 | f | - | | 36 | 8 | p | - | |
| 2 | 0 | i | 0 | | 37 | 8 | (| - | |
| 3 | 0 | (| - | | 38 | 8 | i | 1 | |
| 4 | 0 | d | - | | 39 | 8 | , | - | |
| 5 | 0 | i | 1 | | 40 | 8 | i | 2 | |
| 6 | 0 | , | - | | 41 | 8 |) | - | |
| 7 | 0 | d | - | | 42 | 8 | ; | - | |
| 8 | 0 | i | 2 | | 43 | 9 | } | - | |
| 9 | 0 |) | - | | 44 | 10 | r | - | |
| 10 | 0 | { | - | | 45 | 10 | i | 3 | |
| 11 | 1 | n | - | | 46 | 10 | ; | - | |
| 12 | 1 | d | - | | 47 | 11 | } | - | |
| 13 | 1 | i | 3 | | 48 | 11 | ; | - | |
| 14 | 1 | = | - | | 49 | 13 | s | - | |
| 15 | 1 | l | 4 | | 50 | 13 | f | - | |
| 16 | 1 | ; | - | | 51 | 13 | i | 6 | |
| 17 | 2 | c | - | | 52 | 13 | (| - | |
| 18 | 2 | (| - | | 53 | 13 | s | - | |
| 19 | 2 | i | 1 | | 54 | 13 | i | 7 | |
| 20 | 2 | < | - | | 55 | 13 |) | - | |
| 21 | 2 | l | 5 | | 56 | 13 | { | - | |
| 22 | 2 |) | - | | 57 | 14 | r | - | |
| 23 | 3 | { | - | | 58 | 14 | i | 7 | |
| 24 | 4 | i | 3 | | 59 | 14 | ; | - | |
| 25 | 4 | = | - | | 60 | 15 | } | - | |
| 26 | 4 | a | - | | 61 | 15 | ; | - | |
| 27 | 4 | (| - | | 62 | 17 | m | - | |
| 28 | 4 | i | 1 | | 63 | 18 | { | - | |
| 29 | 4 |) | - | | 64 | 19 | n | - | |
| 30 | 4 | ; | - | | 65 | 19 | s | - | |
| 31 | 5 | } | - | | 66 | 19 | i | 8 | |
| 32 | 6 | e | - | | 67 | 19 | ; | - | |
| 33 | 7 | { | - | | 68 | 21 | n | - | |
| 34 | 8 | i | 3 | | 69 | 21 | s | - | |
| 75 | 21 |) | - | | 70 | 21 | i | 9 | |
| 76 | 21 | ; | - | | 71 | 21 | = | - | |
| | | | | | 72 | 21 | i | 6 | |
| | | | | | 73 | 21 | (| - | |
| | | | | | 74 | 21 | l | 10 | |
| | | | | | 120 | 30 | + | - | |
| | | | | | 121 | 30 | l | 22 | |

| | | | | | | | | | |
|-----|----|---|----|--|-----|----|---|----|--|
| 77 | 22 | o | - | | 122 | 30 | ; | - | |
| 78 | 22 | i | 9 | | 123 | 31 | o | - | |
| 79 | 22 | ; | - | | 124 | 31 | l | 23 | |
| 80 | 24 | o | - | | 125 | 31 | ; | - | |
| 81 | 24 | l | 11 | | 126 | 32 | o | - | |
| 82 | 24 | ; | - | | 127 | 32 | i | 12 | |
| 83 | 25 | n | - | | 128 | 32 | ; | - | |
| 84 | 25 | d | - | | 129 | 34 | o | - | |
| 85 | 25 | i | 12 | | 130 | 34 | l | 24 | |
| 86 | 25 | = | - | | 131 | 34 | ; | - | |
| 87 | 25 | l | 13 | | 132 | 35 | n | - | |
| 88 | 25 | / | - | | 133 | 35 | d | - | |
| 89 | 25 | l | 14 | | 134 | 35 | i | 25 | |
| 90 | 25 | ; | - | | 135 | 35 | = | - | |
| 91 | 26 | o | - | | 136 | 35 | i | 0 | |
| 92 | 26 | i | 12 | | 137 | 35 | (| - | |
| 93 | 26 | ; | - | | 138 | 35 | l | 26 | |
| 94 | 27 | o | - | | 139 | 35 | , | - | |
| 95 | 27 | l | 15 | | 140 | 35 | l | 27 | |
| 96 | 27 | ; | - | | 141 | 35 |) | - | |
| 97 | 28 | i | 12 | | 142 | 35 | ; | - | |
| 98 | 28 | = | - | | 143 | 36 | o | - | |
| 99 | 28 | l | 16 | | 144 | 36 | i | 25 | |
| 100 | 28 | * | - | | 145 | 36 | ; | - | |
| 101 | 28 | (| - | | 146 | 38 | o | - | |
| 102 | 28 | p | - | | 147 | 38 | l | 28 | |
| 103 | 28 | (| - | | 148 | 38 | ; | - | |
| 104 | 28 | l | 17 | | 149 | 39 | n | - | |
| 105 | 28 | , | - | | 150 | 39 | d | - | |
| 106 | 28 | l | 18 | | 151 | 39 | i | 29 | |
| 107 | 28 |) | - | | 152 | 39 | ; | - | |
| 108 | 28 | - | - | | 153 | 40 | c | - | |
| 109 | 28 | l | 19 | | 154 | 40 | (| - | |
| 110 | 28 |) | - | | 155 | 40 | i | 12 | |
| 111 | 28 | + | - | | 156 | 40 |] | - | |
| 112 | 28 | l | 20 | | 157 | 40 | l | 30 | |
| 113 | 28 | ; | - | | 158 | 40 |) | - | |
| 114 | 29 | o | - | | 159 | 41 | { | - | |
| 115 | 29 | i | 12 | | 160 | 42 | o | - | |
| 116 | 29 | ; | - | | 161 | 42 | l | 31 | |
| 117 | 30 | i | 12 | | 162 | 42 | ; | - | |
| 118 | 30 | = | - | | 163 | 43 | i | 29 | |
| 119 | 30 | l | 21 | | 164 | 43 | = | - | |
| 165 | 43 | l | 32 | | | | | | |
| 166 | 43 | ; | - | | | | | | |

| | | | | |
|-----|----|---|----|--|
| 167 | 44 | } | - | |
| 168 | 45 | e | - | |
| 169 | 46 | { | - | |
| 170 | 47 | o | - | |
| 171 | 47 | l | 33 | |
| 172 | 47 | ; | - | |
| 173 | 48 | i | 29 | |
| 174 | 48 | = | - | |
| 175 | 48 | l | 34 | |
| 176 | 48 | ; | - | |
| 177 | 49 | } | - | |
| 178 | 50 | o | - | |
| 179 | 50 | i | 29 | |
| 180 | 50 | ; | - | |
| 181 | 52 | n | - | |
| 182 | 52 | b | - | |
| 183 | 52 | i | 35 | |
| 184 | 52 | ; | - | |
| 185 | 53 | c | - | |
| 186 | 53 | (| - | |
| 187 | 53 | i | 35 | |
| 188 | 53 | & | - | |
| 189 | 53 | l | 36 | |
| 190 | 53 |) | - | |
| 191 | 54 | { | - | |
| 192 | 55 | o | - | |
| 193 | 55 | l | 37 | |
| 194 | 55 | ; | - | |
| 195 | 56 | } | - | |
| 196 | 57 | e | - | |
| 197 | 58 | { | - | |
| 198 | 59 | o | - | |
| 199 | 59 | l | 38 | |
| 200 | 59 | ; | - | |
| 201 | 60 | } | - | |
| 202 | 61 | } | - | |
| 203 | 61 | ; | - | |

Таблица лексем (после построения ПОЛИЗ)

| =====LEX | | | | | | | | | |
|---------------------------|------|--------|------------|---|--|----|----|---|----|
| TABLE===== | | | | | | | | | |
| +-----+-----+-----+-----+ | | | | | | | | | |
| № | Line | Lexema | ID from IT | | | | | | |
| | 0 | 0 | d | - | | 39 | -1 | # | - |
| | 1 | 0 | f | - | | 40 | -1 | # | - |
| | 2 | 0 | i | 0 | | 41 | -1 | # | - |
| | 3 | 0 | (| - | | 42 | 8 | ; | - |
| | 4 | 0 | d | - | | 43 | 9 | } | - |
| | 5 | 0 | i | 1 | | 44 | 10 | r | - |
| | 6 | 0 | , | - | | 45 | 10 | i | 3 |
| | 7 | 0 | d | - | | 46 | 10 | ; | - |
| | 8 | 0 | i | 2 | | 47 | 11 | } | - |
| | 9 | 0 |) | - | | 48 | 11 | ; | - |
| | 10 | 0 | { | - | | 49 | 13 | s | - |
| | 11 | 1 | n | - | | 50 | 13 | f | - |
| | 12 | 1 | d | - | | 51 | 13 | i | 6 |
| | 13 | 1 | i | 3 | | 52 | 13 | (| - |
| | 14 | 1 | = | - | | 53 | 13 | s | - |
| | 15 | 1 | l | 4 | | 54 | 13 | i | 7 |
| | 16 | 1 | ; | - | | 55 | 13 |) | - |
| | 17 | 2 | c | - | | 56 | 13 | { | - |
| | 18 | 2 | (| - | | 57 | 14 | r | - |
| | 19 | 2 | i | 1 | | 58 | 14 | i | 7 |
| | 20 | 2 | < | - | | 59 | 14 | ; | - |
| | 21 | 2 | l | 5 | | 60 | 15 | } | - |
| | 22 | 2 |) | - | | 61 | 15 | ; | - |
| | 23 | 3 | { | - | | 62 | 17 | m | - |
| | 24 | 4 | i | 3 | | 63 | 18 | { | - |
| | 25 | 4 | = | - | | 64 | 19 | n | - |
| | 26 | 4 | i | 1 | | 65 | 19 | s | - |
| | 27 | 4 | a | - | | 66 | 19 | i | 8 |
| | 28 | -1 | # | - | | 67 | 19 | ; | - |
| | 29 | -1 | # | - | | 68 | 21 | n | - |
| | 30 | 4 | ; | - | | 69 | 21 | s | - |
| | 31 | 5 | } | - | | 70 | 21 | i | 9 |
| | 32 | 6 | e | - | | 71 | 21 | = | - |
| | 33 | 7 | { | - | | 72 | 21 | l | 10 |
| | 34 | 8 | i | 3 | | 73 | 21 | i | 6 |
| | 35 | 8 | = | - | | 74 | -1 | # | - |
| | 36 | 8 | i | 1 | | 75 | -1 | # | - |
| | 37 | 8 | i | 2 | | 76 | 21 | ; | - |
| | 38 | 8 | p | - | | 77 | 22 | o | - |
| | | | | | | 78 | 22 | i | 9 |
| | | | | | | 79 | 22 | ; | - |
| | | | | | | 80 | 24 | o | - |
| | | | | | | 81 | 24 | l | 11 |
| | | | | | | 82 | 24 | ; | - |

| | | | | | | | | | |
|-----|----|---|----|--|-----|----|---|----|--|
| 83 | 25 | n | - | | 128 | 32 | ; | - | |
| 84 | 25 | d | - | | 129 | 34 | o | - | |
| 85 | 25 | i | 12 | | 130 | 34 | l | 24 | |
| 86 | 25 | = | - | | 131 | 34 | ; | - | |
| 87 | 25 | l | 13 | | 132 | 35 | n | - | |
| 88 | 25 | l | 14 | | 133 | 35 | d | - | |
| 89 | 25 | / | - | | 134 | 35 | i | 25 | |
| 90 | 25 | ; | - | | 135 | 35 | = | - | |
| 91 | 26 | o | - | | 136 | 35 | l | 26 | |
| 92 | 26 | i | 12 | | 137 | 35 | l | 27 | |
| 93 | 26 | ; | - | | 138 | 35 | i | 0 | |
| 94 | 27 | o | - | | 139 | -1 | # | - | |
| 95 | 27 | l | 15 | | 140 | -1 | # | - | |
| 96 | 27 | ; | - | | 141 | -1 | # | - | |
| 97 | 28 | i | 12 | | 142 | 35 | ; | - | |
| 98 | 28 | = | - | | 143 | 36 | o | - | |
| 99 | 28 | l | 16 | | 144 | 36 | i | 25 | |
| 100 | 28 | l | 17 | | 145 | 36 | ; | - | |
| 101 | 28 | l | 18 | | 146 | 38 | o | - | |
| 102 | 28 | p | - | | 147 | 38 | l | 28 | |
| 103 | 28 | l | 19 | | 148 | 38 | ; | - | |
| 104 | 28 | - | - | | 149 | 39 | n | - | |
| 105 | 28 | * | - | | 150 | 39 | d | - | |
| 106 | 28 | l | 20 | | 151 | 39 | i | 29 | |
| 107 | 28 | + | - | | 152 | 39 | ; | - | |
| 108 | -1 | # | - | | 153 | 40 | c | - | |
| 109 | -1 | # | - | | 154 | 40 | (| - | |
| 110 | -1 | # | - | | 155 | 40 | i | 12 | |
| 111 | -1 | # | - | | 156 | 40 |] | - | |
| 112 | -1 | # | - | | 157 | 40 | l | 30 | |
| 113 | 28 | ; | - | | 158 | 40 |) | - | |
| 114 | 29 | o | - | | 159 | 41 | { | - | |
| 115 | 29 | i | 12 | | 160 | 42 | o | - | |
| 116 | 29 | ; | - | | 161 | 42 | l | 31 | |
| 117 | 30 | i | 12 | | 162 | 42 | ; | - | |
| 118 | 30 | = | - | | 163 | 43 | i | 29 | |
| 119 | 30 | l | 21 | | 164 | 43 | = | - | |
| 120 | 30 | l | 22 | | 165 | 43 | l | 32 | |
| 121 | 30 | + | - | | 166 | 43 | ; | - | |
| 122 | 30 | ; | - | | 167 | 44 | } | - | |
| 123 | 31 | o | - | | 168 | 45 | e | - | |
| 124 | 31 | l | 23 | | 169 | 46 | { | - | |
| 125 | 31 | ; | - | | 170 | 47 | o | - | |
| 126 | 32 | o | - | | 171 | 47 | l | 33 | |
| 127 | 32 | i | 12 | | 172 | 47 | ; | - | |

| | | | | |
|-----|----|---|----|--|
| 173 | 48 | i | 29 | |
| 174 | 48 | = | - | |
| 175 | 48 | l | 34 | |
| 176 | 48 | ; | - | |
| 177 | 49 | } | - | |
| 178 | 50 | o | - | |
| 179 | 50 | i | 29 | |
| 180 | 50 | ; | - | |
| 181 | 52 | n | - | |
| 182 | 52 | b | - | |
| 183 | 52 | i | 35 | |
| 184 | 52 | ; | - | |
| 185 | 53 | c | - | |
| 186 | 53 | (| - | |
| 187 | 53 | i | 35 | |
| 188 | 53 | & | - | |
| 189 | 53 | l | 36 | |
| 190 | 53 |) | - | |
| 191 | 54 | { | - | |
| 192 | 55 | o | - | |
| 193 | 55 | l | 37 | |
| 194 | 55 | ; | - | |
| 195 | 56 | } | - | |
| 196 | 57 | e | - | |
| 197 | 58 | { | - | |
| 198 | 59 | o | - | |
| 199 | 59 | l | 38 | |
| 200 | 59 | ; | - | |
| 201 | 60 | } | - | |
| 202 | 61 | } | - | |
| 203 | 61 | ; | - | |

Приложение Г

Последовательность правил грамматики

| | | |
|----------------------|---------------------|--------------------|
| 0 : S->d fi(F){N};S | 92 : L->i | 157 : L->l |
| 4 : F->di,F | 94 : N->oL;N | 160 : N->oL;N |
| 7 : F->di | 95 : L->l | 161 : L->l |
| 11 : N->nTi=E;N | 97 : N->i=E;N | 163 : N->i=E; |
| 12 : T->d | 99 : E->lM | 165 : E->l |
| 15 : E->l | 100 : M->*E | 170 : N->oL;N |
| 17 : N->cQ{N}e{N}N | 101 : E->(E)M | 171 : L->l |
| 18 : Q->(L<L) | 102 : E->p(W)M | 173 : N->i=E; |
| 19 : L->i | 104 : W->l,W | 175 : E->l |
| 21 : L->l | 106 : W->l | 178 : N->oL;N |
| 24 : N->i=E; | 108 : M->-E | 179 : L->i |
| 26 : E->a(W) | 109 : E->l | 181 : N->nTi;N |
| 28 : W->i | 111 : M->+E | 182 : T->b |
| 34 : N->i=E; | 112 : E->l | 185 : N->cQ{N}e{N} |
| 36 : E->p(W) | 114 : N->oL;N | 186 : Q->(L&L) |
| 38 : W->i,W | 115 : L->i | 187 : L->i |
| 40 : W->i | 117 : N->i=E;N | 189 : L->l |
| 44 : N->rL; | 119 : E->lM | 192 : N->oL; |
| 45 : L->i | 120 : M->+E | 193 : L->l |
| 49 : S->s fi(F){N};S | 121 : E->l | 198 : N->oL; |
| 53 : F->si | 123 : N->oL;N | 199 : L->l |
| 57 : N->rL; | 124 : L->l | |
| 58 : L->i | 126 : N->oL;N | |
| 62 : S->m{N}; | 127 : L->i | |
| 64 : N->nTi;N | 129 : N->oL;N | |
| 65 : T->s | 130 : L->l | |
| 68 : N->nTi=E;N | 132 : N->nTi=E;N | |
| 69 : T->s | 133 : T->d | |
| 72 : E->i(W) | 136 : E->i(W) | |
| 74 : W->l | 138 : W->l,W | |
| 77 : N->oL;N | 140 : W->l | |
| 78 : L->i | 143 : N->oL;N | |
| 80 : N->oL;N | 144 : L->i | |
| 81 : L->l | 146 : N->oL;N | |
| 83 : N->nTi=E;N | 147 : L->l | |
| 84 : T->d | 149 : N->nTi;N | |
| 87 : E->lM | 150 : T->d | |
| 88 : M->/E | 153 : N->cQ{N}e{N}N | |
| 89 : E->l | 154 : Q->(L]L) | |
| 91 : N->oL;N | 155 : L->i | |

Приложение Д

Структур данных, описывающие контекстно-свободную грамматику

```

struct Rule // правило в грамматике Грейбах
{
    GRBALPHABET nn; // нетерминал (левый символ правила) < 0
    int iderror; // идентификатор диагностического сообщения
    short size; // количество цепочек - правых частей правила
    struct Chain // цепочка (правая часть правила)
    {
        short size; // длина цепочки
        GRBALPHABET* nt; // цепочка терминалов (> 0) и
// нетерминалов (< 0)
        Chain() { size = 0; nt = 0; };
        Chain(
            short psize, // количество символов в цепочке
            GRBALPHABET s, ... // символы (терминал
// или нетерминал)
        );
        char* getCChain(char* b); // получить правую
// сторону правила
        static GRBALPHABET T(char t) { return GRBALPHABET(t); };
// терминал
        static GRBALPHABET N(char n) { return -GRBALPHABET(n); };
// нетерминал
        static bool isT(GRBALPHABET s) { return s > 0; };
// терминал?
        static bool isN(GRBALPHABET s) { return !isT(s); };
// нетерминал?
        static char alphabet_to_char(GRBALPHABET s) { return
isT(s) ? char(s) : char(-s); }; // GRBALPHABET->char
    }*chains; // массив цепочек - правых частей правила
    Rule() { nn = 0x00; size = 0; };
    Rule(
        GRBALPHABET pnn, // нетерминал (< 0)
        int iderror, // идентификатор диагностического
// сообщения (Error)
        short psize, // количество цепочек - правых
// частей правила
        Chain c, ... // множество цепочек - правых
// частей правила
    );
    char* getCRule( // получить правило в виде N-
// цепочки (для распечатки)
        char* b, // буфер
        short nchain // номер цепочки (правой части) в правиле
    );
    short getNextChain( // получить следующую за j
// подходящую цепочку, вернуть её номер или -1
        GRBALPHABET t, // первый символ цепочки
        Rule::Chain& pchain, // возвращаемая цепочка
        short j // номер цепочки
    );
};

```

```

    );
};

struct Greibach                                // грамматика Грейбах
{
    short size;                                // количество правил
    GRBALPHABET startN;                        // стартовый символ
    GRBALPHABET stbottomT;                     // дно стека
    Rule* rules;                               // множество правил
    Greibach() { short size = 0; startN = 0; stbottomT = 0;
rules = 0; };
    Greibach(
        GRBALPHABET pstartN,                  // стартовый символ
        GRBALPHABET pstbottomT,               // дно стека
        short psize,                           // количество правил
        Rule r, ...                            // правила
    );
    short getRule(                              // получить правило,
возвращающая номер правила или -1
        GRBALPHABET pnn,                      // левый символ
        Rule& prule                           // возвращаемое правило грамматики
    );
    Rule getRule(short n);                      // получить правило по номеру
};
    Greibach getGreibach();                    // получить грамматику

```

Приложение Е

Результат генерации кода

```

.586
.model flat, stdcall
includelib libcrt.lib
includelib kernel32.lib

ExitProcess PROTO:DWORD
SYSPAUSE PROTO
soutl PROTO : BYTE
noutl PROTO : SDWORD
_pow PROTO : SDWORD, : SDWORD
_abs PROTO : SDWORD

.STACK 4096

.CONST
    null_division BYTE 'ERROR:
DIVISION BY ZERO', 0
    OVER_FLOW BYTE 'ERROR:
OVERFLOW', 0
    true BYTE 'true', 0
    false BYTE 'false', 0
    short0 SDWORD -1
    short1 SDWORD 0
    str2 BYTE "Hello there!", 0
    str3 BYTE "Арифм. операции:",
0
    short4 SDWORD 26
    short5 SDWORD 13
    str6 BYTE "Контрольный пример
выражения: ", 0
    short7 SDWORD 5
    short8 SDWORD 11
    short9 SDWORD 2
    short10 SDWORD 178
    short11 SDWORD 19
    short12 SDWORD 255
    short13 SDWORD 19
    str14 BYTE "FFH + 230 = ", 0
    str15 BYTE "Вызов функции:",
0
    short16 SDWORD -4
    short17 SDWORD 3
    str18 BYTE "Условный
оператор:", 0
    short19 SDWORD 0
    str20 BYTE "positive", 0
    short21 SDWORD 1
    str22 BYTE "negative", 0
    mov res10, eax

short23 SDWORD -1
bool24 DWORD 1
str25 BYTE "Its true", 0
str26 BYTE "Its false", 0

.DATA
    res10 SDWORD 0
    str163 DWORD ?
    i63 SDWORD 0
    num63 SDWORD 0
    c63 SDWORD 0
    flag63 DWORD 0

.CODE

func1 PROC b1 : SDWORD, a1 :
SDWORD
    push short0
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov res10, eax
    mov eax, a1
    cmp eax, short1
    jl ifi1
    jge else1
ifi1:
    push a1
    call _abs
    push eax
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov res10, eax
    jmp ifEnd1
else1:
    push a1
    push b1
    call _pow
    push eax
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    push eax

```

```

ifEnd1:
    push res10
    jmp local0
local0:
    pop eax
    ret
func1 ENDP

setstring2 PROC s2 : DWORD
    push s2
    jmp local1
local1:
    pop eax
    ret
setstring2 ENDP

main PROC
    push offset str2
    call setstring2
    push eax
    pop str163

    push str163
    call sout1

    push offset str3
    call sout1
    push short4
    push short5
    pop ebx
    pop eax
    cmp ebx,0
    je SOMETHINGWRONG
    cdq
    idiv ebx
    push eax
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov i63, eax

    push i63
    call nout1

    push offset str6
    call sout1
    push short7
    push short8
    push short9
    call _pow
    mov num63, eax

```

```

    push short10
    pop ebx
    pop eax
    sub eax, ebx
    push eax
    pop eax
    pop ebx
    mul ebx
    push eax
    push short11
    pop eax
    pop ebx
    add eax, ebx
    push eax
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov i63, eax

    push i63
    call nout1
    push short12
    push short13
    pop eax
    pop ebx
    add eax, ebx
    push eax
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov i63, eax

    push offset str14
    call sout1

    push i63
    call nout1

    push offset str15
    call sout1
    push short16
    push short17
    call func1
    push eax
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
ifEnd3:

```

```

push num63
call noutl

push offset str18
call soutl
    mov eax, i63
    cmp eax, short19
    jge ifi2
    jl else2
ifi2:

push offset str20
call soutl
    push short21
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov c63, eax
    jmp ifEnd2
else2:

push offset str22
call soutl
    push short23
    pop eax
    cmp eax, 32767
    jg overflow
    cmp eax, -32768
    jl overflow
    mov c63, eax
ifEnd2:

push c63
call noutl
    mov eax, flag63
    cmp eax, bool24
    jz ifi3
    jnz else3
ifi3:

push offset str25
call soutl
    jmp ifEnd3
else3:

push offset str26
call soutl
    call SYSPAUSE
    push 0
    call ExitProcess
SOMETHINGWRONG::
    push offset null_division
    call soutl
    jmp THEEND
overflow::
    push offset OVER_FLOW
    call soutl
THEEND:
    call SYSPAUSE
    push -1
    call ExitProcess
main ENDP
end main

```