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 SES-2020»

Выполнил студент Шумский Евгений Сергеевич

(Ф.И.О.)

Руководитель проекта пр.ст. Наркевич Аделина Сергеевна

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

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

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

Консультант пр.ст. Наркевич Аделина Сергеевна

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

Нормоконтролер пр.ст. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

**Введение**

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования – SES-2020. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор SES-2020 – это программа, задачей которого является перевод программы, написанной на языке программирования SES-2020 в программу на язык ассемблера.

Транслятор SES-2020 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода в язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач буду приведены в соответствующих главах курсового проекта.

Глава 1.  **Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык программирования SES-2020 является процедурным, универсальным, строго типизированным, компилируемым и не объектно-ориентированным языком.

**1.2 Алфавит языка**

Алфавит языка SES-2020 основан на таблице символов ASCII, представленная на рис.1.1.

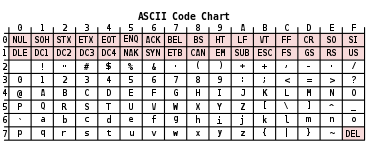


Рисунок 1.1 – Алфавит входных символов

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], символы пробела, табуляции и перевода строки, спецсимволы: [ ] ( ) , ; + - > < !.

**1.3 Символы-cепараторы**

Символы-сепараторы служат с целью разделения на токены цепочек языка. Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор(ы) | Назначение |
| ‘ ’ | Разделитель цепочек. Допускается везде кроме идентификаторов и ключевых слов |
| **[** … **]** | Блок цикла |
| **(** … **)** | Блок параметров функции или блок условия цикла |

|  |  |
| --- | --- |
| Сепаратор(ы) | Назначение |
| **,** | Разделитель параметров функций |
| **+ -** | Арифметические операции |
| **> < <= >= != ==** | Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции. |
| **;** | Разделитель программных конструкций |
| {} | Блок функции |
| **=** | Оператор присваивания |

**1.4 Применяемые кодировки**

Для написания исходного кода на языке SES-2020 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как [ ] ( ) , ; = + - > < !{}.

**1.5 Типы данных**

В языке SES-2020 реализованы два типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка SES-2020

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных *int* | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 4 байт. Максимальное значение: 2,147,483,647.  Минимальное значение:2,147,483,648.  Инициализация по умолчанию: значение 0. |
| Строковый тип данных *string* | Фундаментальный тип данных. Используется для работы с символами, каждый символ в памяти занимает 4 байтa.  Максимальное количество символов: 255.  Инициализация по умолчанию: строка нулевой длины “”. |

**1.6 Преобразование типов данных**

Преобразование типов данных в языке SES-2020 не поддерживается, т.е. язык является строго типизированным.

**1.7 Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита нижнего регистра. Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Максимальная длина имени идентификатора - 10 символов. Идентификаторы, объявленные внутри функционального блока, получают суффикс, идентичный имени функции, внутри которой они объявлены. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, имя параметра функции. Имена идентификаторов-функций могут совпадать с именами команд ассемблера.

**1.8 Литералы**

С помощью литералов осуществляется инициализация переменных. Все литералы являются *rvalue*. Типы литералов языка SES-2020 представлены в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Целочисленные литералы в десятичном представлении | Последовательность цифр [0…9] |
| Строковые литералы | Набор символов, заключённых в двойные кавычки |

Ограничения на строковые литералы языка SES-2020: внутри литерала не допускается использование символов кириллицы, а также одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль;

**1.9 Объявление данных**

Для объявления переменной используется ключевое слово *poo*, после которого указывается тип данных и имя идентификатора. Инициализация при объявлении не допускается.

Пример объявления числового типа с инициализацией:

*now number* value1;

Пример объявления переменной символьного типа с инициализацией:

*now string* str1;

Для объявления функций используется ключевое слово *func*, перед которым указывается тип функции (если функция возвращает значение), а после –имя функции либо процедуры. Далее обязателен список параметров и тело функции.

**1.10 Инициализация данных**

При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа *number* и строка длины 0 (“”) для типа *string*.

**1.11 Инструкции языка**

Инструкции языка SES-2020 представлены в таблице 1.4

Таблица 1.4 – ИнструкцииязыкаSES-2020

|  |  |
| --- | --- |
| Инструкция | Запись на языке SES-2020 |
| Объявление переменной | *poo* <тип данных> <идентификатор>; |
| Возврат из функции или процедуры | Для функций, возвращающих значение:  *return* <идентификатор/литерал>; |
| Вывод данных | *out*<идентификатор/литерал>; |
| Вызов функции или процедуры | <идентификатор функции> (<список параметров>);  Список параметров может быть пустым. |
| Присваивание | <идентификатор> **=** <выражение>; |

**1.12 Операции языка**

В языке SES-2020 предусмотрены следующие операции с данными. Приоритетность операции сложения равна приоритетности операции вычитания. Операции языка представлены в таблице 1.5.

Таблица 1.5 – Операции языка SES-2020

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | 1. **+ –** сложение 2. - – вычитание 3. = – присваивание |
| Строковые | 1. **=** – присваивание |
| Логические | 1. **>** – больше  2. **<** – меньше  3. ! – проверка на неравенство |
| Сдвиговые | 1. } – cдвиг вправо  2. { – сдвиг влево |

**1.13 Выражения и их вычисление**

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Выражение записывается в строку без переносов;
2. Использование двух подряд идущих операторов не допускается;
3. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера.

**1.14 Программные конструкции языка**

Программа на языке SES-2020 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка SES-2020

|  |  |
| --- | --- |
| Конструкция | Запись на языке SES-2020 |
| Главная функция(точка входа) | *main*  {  …  } |
| Цикл | Twirl( <идентификатор1> <оператор> <идентификатор2> **)**  *{*  *…*  *}* |
| Внешняя функция | <тип данных> *func* <идентификатор> **(**<тип> <идентификатор>, ...**)**  {…  *return* <идентификатор/литерал>;  } |

**1.15 Области видимости идентификаторов**

В языке SES-2020 все переменные являются локальными. Они обязаны находится внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Каждая переменная или параметр функции получают суффикс – название функции, внутри которой они находятся.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Параметры видны только внутри функции, в которой объявлены.

**1.16 Семантические проверки**

В языке программирования SVY-2020 выполняются следующие семантические проверки приведённых в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Проверка соответствия типа функции и возвращаемого параметра |
| 2 | Правильность передаваемых в функцию параметров: количество, типы |
| 3 | Превышение размера строковых и числовых литералов |
| 4 | Правильность составленного условия цикла |

**1.17 Распределение оперативной памяти на этапе выполнения**

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода, несмотря на то, что в оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

**1.18 Стандартная библиотека и её состав**

В языке SVY-2020 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |
| --- | --- |
| Функция | Описание |
| *int mod(int a, int b)* | Целочисленная функция. Возвращает остаток от деления числа a на число b. |
| *int square*(*int* a, int b); | Целочисленная функция. Возвращает число а возведенное в степень b. |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор *out*. Эти функции представлены в таблице 1.10.

Таблица 1.10 – Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void printi(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void prints(char\* line) | Функция для вывода в стандартный поток значения строкового идентификатора/литерала. |

**1.19 Ввод и вывод данных**

В языке SVY-2020 реализованы средства вывода данных с помощью оператора *out*. Допускается использование оператора *out* с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда *out*  в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

**1.20 Точка входа**

В языке SVY-2020 каждая программа должна содержать главную функцию *main*, т.е. точку входа, с которой начнется последовательное выполнение программы.

**1.21 Препроцессор**

Препроцессор в языке программирования SVY-2020 отсутствует.

**1.22 Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

**1.23 Объектный код**

Язык SVY-2020 транслируется в язык ассемблера, а затем - в объектный код.

**1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке SVY-2020 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| **Сообщение** | **Описание** |
| 0-99 | ошибка системы |
| 100-199 | Ошибка входного кода |
| 200-299 | Ошибка на этапе лексического анализа |
| 300-399 | Ошибка на этапе синтаксического анализа |
| 400-499 | Ошибка на этапе семантического анализа |

**1.25 Контрольный пример**

Контрольный пример представлен в главе Приложения A.

**2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке SVY-2020 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера.

Принцип работы представлен на рисунке 2.1.

Рисунок 2.1 – Структура транслятора языка программирования SVY-2020

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора

Семантический анализ в свою очередь является проверкой исходной программы SVY-2020 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке SVY-2020, прошедший все предыдущие этапы, в код на языке Ассемблера.

**2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка SVY-2020

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на SVY-2020. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке SVY-2020. | <имя\_файла>.log |
| -out:<имя\_файла> | Файл содержит в себе выходной код на языке ассемблера. | <имя\_файла>.asm |
| -d | Ключ выключает режим отладки. Выводит информацию о лексическом и синтаксическом анализе, а также результат польской записи | По умолчанию отключен |
| -lex | Ключ для вывода результата лексического анализа | По умолчанию отключен |

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

Таблица с перечнем протоколов, формируемых транслятором языка SVY-2020 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка SVY-2020

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, заданный параметром "-log:" | Файл содержит в себе краткую информацию об исходном коде на языке SVY-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. |
| Файл таблицы лексем, с расширением ".lex" | Файл содержит в себе часть результата работы лексического анализатора: таблицу лексем. |
| Файл таблицы идентификаторов, с расширением ".id" | Файл содержит в себе часть результата работы лексического анализатора: таблицу идентификаторов. |
| Выходной файл, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

**3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

− удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 – Структура лексического анализатора

**3.2. Контроль входных символов**

Исходный код на языке программирования SVY-2020 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена на рисунке 3.2.,

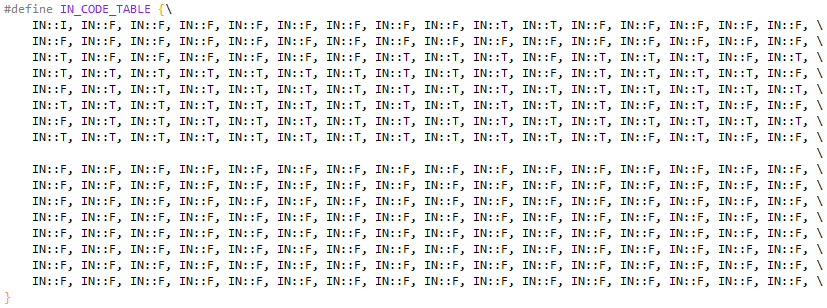
категории входных символов представлены в таблице 3.1.

Рисунок 3.2. – Таблица контроля входных символов

Таблица 3.1 – Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |

**3.3 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования SVY-2020 разбивается на токены, которые записываются в очередь.

Описание алгоритма удаления избыточных символов:

* + - 1. Посимвольно считываем файл с исходным кодом программы;
      2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора;
      3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

**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 3.2.

Таблица 3.2 – Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| main | m | Главная функция. |
| poo | d | Объявление переменной. |
| func | f | Объявление функции. |
| out | o | Ввод данных. |
| istrue | r | Истинная ветвь условного оператора. |
| int, string | t | Названия типов данных языка. |
| Идентификатор | i | Идентификатор переменной, функции либо параметра функции. |
| Литерал | l | Литерал любого доступного типа. |
| return | r | Выход из функции/процедуры. |
| twirl | w | Указывает начало цикла |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| +  - | +  - | Знаки операций. |
| >  <  <=  >=  != | c | Знаки логических операторов |
| [ | [ | Открытие блока цикла. |
| ] | ] | Закрытие блока цикла. |
| } | } | Открытие блока функции. |
| } | } | Закрытие блока функции. |
| ( | ( | Передача параметров в функцию. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров. |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.



Рисунок 3.3 – Структура конечного автомата

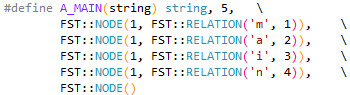


Рисунок 3.4 – Пример реализации графа конечного автомата для токена main**(**точки входа)

**3.5 Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы лексем представлен на рисунке 3.3. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

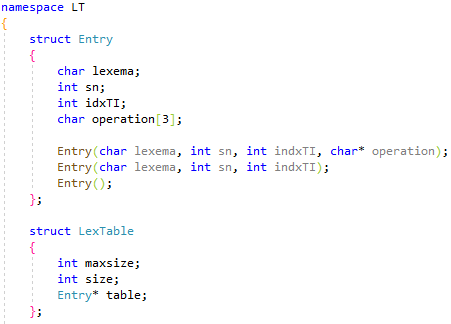


Рисунок 3.3 – Структура таблицы лексем

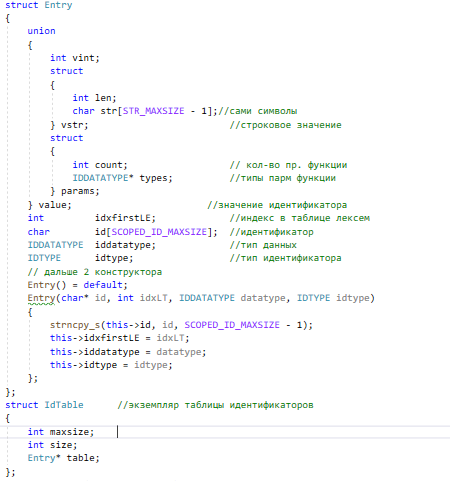


Рисунок 3.4 – Структура таблицы идентификаторов

**3.6 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, дальнейшая обработка прекращается. Перечень сообщений представлен на рисунке 3.5.

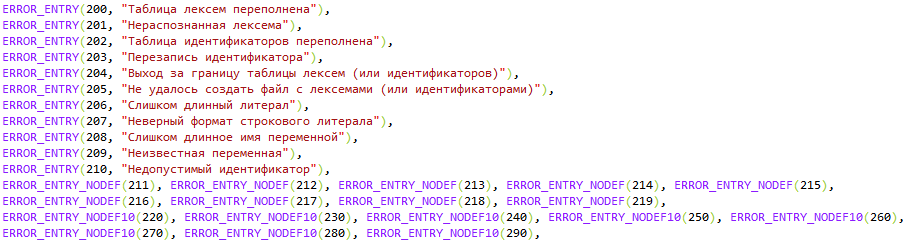


Рисунок 3.5 – Сообщения лексического анализатора

**3.7 Структура и перечень сообщений лексического анализатора**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

**3.8 Параметры лексического анализатора**

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов могут выводиться как в файл, так и в командную строку.

* 1. **Алгоритм лексического анализа**
     + 1. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
       2. для выделенной части входного потока выполняется функция распознавания лексемы;
       3. при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
       4. при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «*string*» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

s t r i n g

Рисунок 3.2 – Граф переходов для цепочки *“string*”

**3.10 Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении Б.

**4. Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

Рисунок 4.1 – Структура синтаксического анализатора.

**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка**

В синтаксическом анализаторе транслятора языка SVY-2020 используется контекстно-свободная грамматика , где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или );
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

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 |
| T | T->[eV;]  T->[KeV;] | Правила для тела функций |
| G | G->[e;]  G->[Ke;] | Правила для тела процедур |
| E | E->ti,E  E->ti | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вывозов функций(в т.ч. 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/условного оператора |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла/условного оператора |
| W | W->l  W->i  W->(W)  W->(W)AW  W->iF  W->iAW  W->lAW  W->iFAW | Правила для сложных выражений |
| A | A->+  A->-  A->\*  A->/  A->{  A->} | Правила для арифметических и свдиговых операторов |

Окончание таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| V | V->l  V->i  V->q | Правила для простых выражений |
| Y | Y->[X] | Правила для тела цикла/условного выражения |
| L | L-><  L->>  L->! | Правила для логических операторов |
| 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/условного оператора |

**4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики |
|  | Начальное состояние магазина автомата | Символ маркера дна стека $ |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

**4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка SVY-2020. Данные структуры в приложении В.

**4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. Если в магазине встретился нетерминал, переходим к пункту 4;
7. Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

**4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

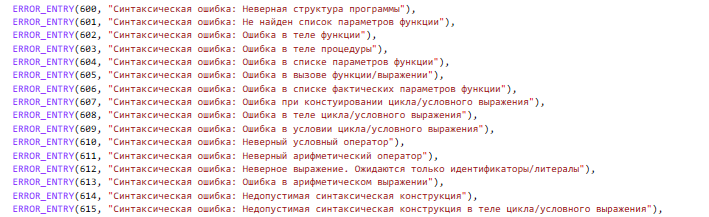


Рисунок 4.3 – Сообщения синтаксического анализатора

**4.7. Параметры синтаксического анализатора и режимы его работы**

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

**4.8. Принцип обработки ошибок**

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

**4.9. Контрольный пример**

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.