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 CES-2020»

Выполнил студент Черник Евгений Сергеевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультант ст.пр. Наркевич Аделина Сергеевна

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[Введение 4](#_Toc58885999)

[1. Спецификация языка программирования CES-2020 5](#_Toc58886000)

[1.1 Характеристика языка программирования 5](#_Toc58886001)

[1.2 Определение алфавита языка программирования 5](#_Toc58886002)

[1.3 Применяемые сепараторы 5](#_Toc58886003)

[1.4 Применяемые кодировки 5](#_Toc58886004)

[1.5 Типы данных 6](#_Toc58886005)

[1.6 Преобразование типов данных 7](#_Toc58886006)

[1.7 Идентификаторы 7](#_Toc58886007)

[1.8 Литералы 7](#_Toc58886008)

[1.9 Объявление данных 8](#_Toc58886009)

[1.10 Инициализация данных 8](#_Toc58886010)

[1.11 Инструкции языка 8](#_Toc58886011)

[1.12 Операции языка 8](#_Toc58886012)

[1.13 Выражения и их вычисление 9](#_Toc58886013)

[1.14 Конструкции языка 10](#_Toc58886014)

[1.15 Область видимости идентификаторов 10](#_Toc58886015)

[1.16 Семантические проверки 10](#_Toc58886016)

[1.17 Распределение оперативной памяти на этапе выполнения. 11](#_Toc58886017)

[1.18 Стандартная библиотека и ее состав 11](#_Toc58886018)

[1.19 Ввод и вывод данных 11](#_Toc58886019)

[1.20 Точка входа 11](#_Toc58886020)

[1.21 Препроцессор 11](#_Toc58886021)

[1.22 Соглашения о вызовах 12](#_Toc58886022)

[1.23 Объектный код 12](#_Toc58886023)

[1.24 Классификация сообщений транслятора 12](#_Toc58886024)

[1.25 Контрольный пример 12](#_Toc58886025)

[2. Структура транслятора 13](#_Toc58886026)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 13](#_Toc58886027)

[2.2 Перечень входных параметров транслятора 14](#_Toc58886028)

[2.3 Протоколы, формируемые транслятором 14](#_Toc58886029)

[3. Разработка лексического анализатора 16](#_Toc58886030)

[3.1 Структура лексического анализатора 16](#_Toc58886031)

[3.2 Контроль входных символов 16](#_Toc58886032)

[3.3 Удаление избыточных символов 17](#_Toc58886033)

[3.4 Перечень ключевых слов 17](#_Toc58886034)

[3.5 Основные структуры данных 18](#_Toc58886035)

[3.6. Структура и перечень сообщений лексического анализатора 18](#_Toc58886036)

[3.7. Принцип обработки ошибок 18](#_Toc58886037)

[3.8. Параметры лексического анализатора 19](#_Toc58886038)

[3.9. Алгоритм лексического анализа 19](#_Toc58886039)

[3.10. Контрольный пример 19](#_Toc58886040)

[4. Разработка синтаксического анализатора 20](#_Toc58886041)

[4.1. Структура синтаксического анализатора 20](#_Toc58886042)

[4.2. Контекстно-свободная грамматика, описывающая синтаксис языка 20](#_Toc58886043)

[4.3. Построение конечного магазинного автомата 21](#_Toc58886044)

[4.4. Основные структуры данных 23](#_Toc58886045)

[4.5. Описание алгоритма синтаксического разбора 23](#_Toc58886046)

[4.6. Структура и перечень сообщений синтаксического анализатора 23](#_Toc58886047)

[4.7. Параметры синтаксического анализатора и режимы его работы 24](#_Toc58886048)

[4.8. Принцип обработки ошибок 24](#_Toc58886049)

[4.9. Контрольный пример 24](#_Toc58886050)

[5. Разработка семантического анализатора 25](#_Toc58886051)

[5.1. Структура семантического анализатора 25](#_Toc58886052)

[5.2. Функции семантического анализатора 25](#_Toc58886053)

[5.3. Структура и перечень сообщений семантического анализатора 25](#_Toc58886054)

[5.4. Принцип обработки ошибок 26](#_Toc58886055)

[5.5. Контрольный пример 26](#_Toc58886056)

[6. Вычисление выражений 27](#_Toc58886057)

[6.1. Выражения, допускаемые языком 27](#_Toc58886058)

[6.2. Польская запись и принцип ее построения 27](#_Toc58886059)

[6.3. Программная реализация обработки выражений 28](#_Toc58886060)

[6.4. Контрольный пример 28](#_Toc58886061)

[7. Генерация кода 29](#_Toc58886062)

[7.1 Структура генератора кода 29](#_Toc58886063)

[7.2 Представление типов данных в оперативной памяти 29](#_Toc58886064)

[7.3 Статическая библиотека 30](#_Toc58886065)

[7.4 Особенности алгоритма генерации кода 30](#_Toc58886066)

[7.5 Входные параметры генератора кода 30](#_Toc58886067)

[7.6 Контрольный пример 30](#_Toc58886068)

[8. Тестирование транслятора 31](#_Toc58886069)

[8.1 Общие положения 31](#_Toc58886070)

[8.2 Тестирование проверки на допустимость символов 31](#_Toc58886071)

[8.3 Тестирование лексического анализатора 31](#_Toc58886072)

[8.4 Тестирование синтаксического анализатора 32](#_Toc58886073)

[8.5 Тестирование семантического анализатора 32](#_Toc58886074)

[Заключение 36](#_Toc58886075)

[Список использованных источников 37](#_Toc58886076)

[Приложение А 38](#_Toc58886077)

[Приложение Б 48](#_Toc58886078)

[Приложение В 49](#_Toc58886079)

[Приложение Г 52](#_Toc58886080)

[Приложение Д 54](#_Toc58886081)

[Приложение Е 58](#_Toc58886082)

# Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – CES-2020. Написание транслятора будет осуществляться на языке C++, при этом код на языке CES-2020 будет транслироваться в язык ассемблера.

Транслятор CES-2020 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Задание на курсовой проект можно разделить на следующие задачи:

– разработка спецификации языка CES-2020;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– разбор арифметических выражений;

– разработка генератора кода;

– тестирование транслятора.

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

# 1. Спецификация языка программирования CES-2020

## 1.1 Характеристика языка программирования

Язык CES-2020 является компилируемым, процедурным, универсальным, высокоуровневым, со строгой типизацией, не объектно-ориентированным языком программирования.

## 1.2 Определение алфавита языка программирования

В языке CES-2020 разрешено использовать латинские символы [A- Z, a- z], арабские цифры [0-9], а также сепараторы и непечатные символы пробела, перевода строки и табуляции. Символы русского языка разрешены только в строковых литералах.

## 1.3 Применяемые сепараторы

В языке СES-2020 применяются сепараторы, представленные в таблице 1.1.

Таблица 1.1 – Символы-сепараторы языка CES-2020

|  |  |  |
| --- | --- | --- |
| Символ | Символ в таблице Windows-1251 | Описание |
| Пробел | 0x20 | Допускается везде, кроме идентификаторов и ключевых слов. |
| . | 0x2E | Разделитель конструкций. |
| ( ) | 0x28 ; 0x29 | Параметры. |
| ( ) | 0x28 ; 0x29 | Приоритетность операций. |
| ` ` | 0x60 | Строковый литерал. |
| , | 0x2C | Разделитель параметров функции. |
| ‘’ | (0x29) | Символьный литерал. |

## 1.4 Применяемые кодировки

Набор символов — таблица, задающая кодировку конечного множества символов алфавита. Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита.

В языке CES-2020 используется 8-битовая кодировка Windows-1251, представлена на рисунке 1.1.

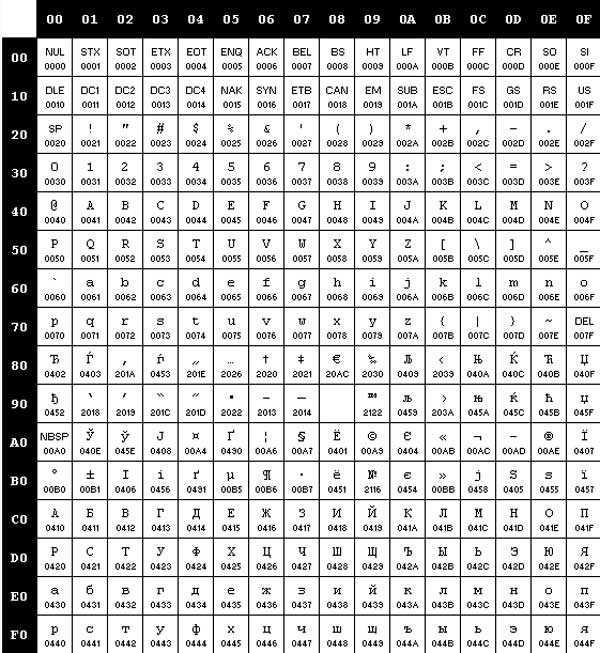


Рисунок 1.1 – Базовые символы языка CES-2020

## 1.5 Типы данных

В языке CES-2020 разрешены типы данных, представленные в таблице 1.2.

Таблица 1.2 – Типы данных языка CES-2020

|  |  |  |  |
| --- | --- | --- | --- |
| Название типа | Описание | Диапазон значений | Значение по умолчанию |
| usint | Целочисленный беззнаковый тип данных (1 байт). Только rvalue. | от 0 до 255. | 0 |
| string | Строковый тип данных, максимальная длина: 255 (плюс символ конца строки) символов. Только rvalue. | От 1 до 255 разрешенных символов. Строка оканчивается терминальным нулём (0x00). | Пустая строка `` |

Продолжение таблицы 1.2

|  |  |  |  |
| --- | --- | --- | --- |
| symbol | Беззнаковый символьный тип данных (1 байт).  Только rvalue. | От 0 до 255. | ‘0’ |
| boolean | Логический тип данных.  Только rvalue. | true либо false. | false |

## 1.6 Преобразование типов данных

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

## 1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Запрещается использовать идентификаторы длиной более 10 символов. Идентификаторы большей длины усекаются. Идентификаторы, объявленные внутри функционального блока, получают префикс, равный имени функции, в которой они объявлены. Идентификатор не должен совпадать с ключевыми словами языка CES-2020. Зарезервированные идентификаторы, используемые в стандартной библиотеке: DATE, TIME.

Регулярное выражение для идентификатора: [a-z]+[0-9 | a-z]\*

Правильные идентификаторы: a, abc, b3.

Неправильные идентификаторы: 1a, Ab, &a1Bc.

## 1.8 Литералы

В языке CES-2020 разрешены 4 вида литералов, представленные в таблице 1.3.

Таблица 1.3 – Литералы в языке CES-2020

|  |  |  |
| --- | --- | --- |
| Тип литерала | Описание | Примечание |
| Целочисленный | Задаются в двоичном, восьмеричном, либо десятичном представлениях. | Двоичный литерал:  [0-1]+b  Восьмеричный литерал:  [0-7]+o  Десятичный литерал:  [0-9]+[d] |
| Строковый | Набор символов (от 1 до 255). | Заключается в обратные кавычки ` `  (0x60) |
| Логический | Применяется в условном операторе. | Может быть либо true, либо false. |
| Символьный | Один разрешенный символ. | Заключается в одинарные кавычки ‘’  (0x29) |

## 1.9 Объявление данных

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

Синтаксис объявления:

new <тип данных> <идентификатор>

Пример:

new usint i1.

Для объявления функций используется ключевое слово DEF, перед которым указывается тип возвращаемого значения.

Пример объявления функции:

usint DEF fun(usint a, usint b)

OPEN

RET a +b.

CLOSE

## 1.10 Инициализация данных

В языке CES-2020 инициализация переменной происходит после её объявления значением по умолчанию. Инициализация переменной в момент объявления не предусмотрена.

## 1.11 Инструкции языка

Инструкция — наименьшая автономная часть языка программирования, команда или набор команд. Программа обычно представляет собой последовательность инструкций.

В языке CES-2020 предусмотрены инструкции, представленные в таблице 1.4.

Таблица 1.4 – Инструкции языка CES-2020

|  |  |
| --- | --- |
| Инструкция | Описание |
| new <тип данных> <идентификатор>. | Объявление переменных. |
| out<идентификатор>|<литерал>. | Вывод в стандартный поток вывода. |
| <идентификатор> (<литерал>|<идентификатор>, ...) | Вызов функции. |
| <идентификатор> : <литерал>|<идентификатор>. | Присваивание значение переменной. |

## 1.12 Операции языка

Операция — конструкция в языках программирования, аналогичная по записи математическим операциям, то есть специальный способ записи некоторых действий.

В языке CES-2020 предусмотрены операции, представленные в табл. 1.5. Приоритетность операций определяется с помощью круглых скобок (). Выполнение операций с разными типами данных не допускается.

Таблица 1.5 – Операции языка CES-2020

|  |  |  |
| --- | --- | --- |
| Тип операции | Операция | Описание |
| Побитовые | <~  (usint <~ usint). | Бинарный, сдвиг влево |
| ~>  (usint <~ usint). | Бинарный, сдвиг вправо |
| Арифметические | +  (usint + usint). | Бинарный, сложение |
| -  (usint - usint). | Бинарный, вычитание |
| \*  (usint \* usint). | Бинарный, умножение |
| /  (usint / usint). | Бинарный, деление |
| Логические | =  (usint = usint). | Бинарный, проверка на равенство |
| !  (usint ! usint). | Бинарный, проверка на неравенство |
| <  (usint < usint). | Бинарный, меньше |
| >  (usint > usint). | Бинарный, больше |

## 1.13 Выражения и их вычисление

Выражением называется совокупность переменных, констант, знаков операций, имен функций, скобок, которая может быть вычислена в соответствии с синтаксисом языка программирования.

Вычисление выражений в языке CES-2020 осуществляется по следующим правилам:

* Для удобочитаемости кода следует записывать выражения в одну строку;
* Допускается использовать скобки для смены приоритета;
* Выражение может содержать вызов функции;
* Использование двух операторов подряд не допускается.
* В одном выражении могут участвовать только операнды одного и того же типа данных.
* Составные выражения с логическими операциями не допускаются.

## 1.14 Конструкции языка

В языке CES-2020 предусмотрены конструкции, представленные в таблице 1.6.

Таблица 1.6 – Конструкции языка CES-2020

|  |  |
| --- | --- |
| Конструкция | Описание |
| HEAD  OPEN  <тело программы>  CLOSE | Главная функция. |
| <тип данных>DEF<идентификатор>(<тип данных><идентификатор>, …)  OPEN  <тело функции>.  RET <идентификатор> | <литерал>.  CLOSE | Описание функции. Максимальное количество параметров в функции: 5. Максимальное количество функций в программе: 5. |
| WHEN (<условие>)  <выражение>.  [  OTHERWISE  <Выражение>.  ]  ENDW | Условный оператор.  Если условие истинно, то выполняются инструкции блока WHEN, иначе – блока OTHERWISE (при наличии).  Внутри блоков WHEN и OTHWERWISE должны находиться однотипные. Объявление переменных внутри не допускается. |

## 1.15 Область видимости идентификаторов

В языке CES-2020 область видимости идентификаторов локальная, сверху вниз, ограниченная блоками OPEN-CLOSE. Каждая переменная или параметр функции получают префикс – название функции, внутри которой они находятся. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены. Объявление пользовательских областей видимости не предусмотрено.

## 1.16 Семантические проверки

В языке CES-2020 осуществляются следующие семантические проверки:

* Наличие одной функции HEAD;
* Переопределение идентификаторов;
* Использование идентификатора до объявления или без него;
* Проверка соответствия типа функции и возвращаемого параметра;
* Проверка на допустимые значения литералов, а также максимальную длину строкового литерала;
* Проверка на инициализацию переменной без её использования.
* Проверка на то, что пользовательская функция вызывается в точке входа.
* Проверка на соответствие количества передаваемых в функцию параметров и их типов.
* Проверка на допустимое количество пользовательских функций в программе.
* Проверка на допустимое количество параметров в пользовательских функциях.
* Деление на 0 в выражении.

## 1.17 Распределение оперативной памяти на этапе выполнения.

В языке CES-2020 для хранения 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 ее состав

В стандартной библиотеке языка CES-2020 содержатся функции, представленные в таблице 1.7.

Таблица 1.7 – Стандартная библиотека языка CES-2020

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Количество параметров |
| string DATE() | Возвращает строку с текущей датой в формате ДД.ММ.ГГ. | 0 |
| string TIME() | Возвращает строку с текущим временем в формате ЧЧ:ММ:СС. | 0 |

## 1.19 Ввод и вывод данных

Ввод данных в языке CES-2020 не предусмотрен.

Вывод данных осуществляется с помощью инструкции out, синтаксис приведён в таблице 1.4.

## 1.20 Точка входа

В языке CES-2020 может быть только одна точка входа и определяется наличием функции HEAD**.** При инициализации более одной или менее одной – выдаст ошибку лексического анализатора.

## 1.21 Препроцессор

Препроцессор — программа для обработки текста. Может быть отдельной программой, или интегрированной в компилятор.

В языке CES-2020 препроцессор не предусмотрен.

## 1.22 Соглашения о вызовах

Соглашение о вызове — описание технических особенностей вызова подпрограмм, определяющее: способы передачи параметров подпрограммам; способы вызова подпрограмм; способы передачи результатов вычислений, выполненных подпрограммами, в точку вызова; способы возврата из подпрограмм в точку вызова.

В языке CES-2020 используется соглашение о вызовах stdcall.

Особенности stdcall:

* Параметры заносятся в стек справа налево;
* Очистка стека выполняется вызываемой функцией.

## 1.23 Объектный код

Объектный код – подлежащая исполнению форма подходящего представления одного или более процессов (текст программы или язык программы), которая компилируется программирующей системой.

Язык CES-2020 транслируется в ассемблер, а затем в объектный код.

## 1.24 Классификация сообщений транслятора

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

Таблица 1.8 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Диапазон ошибок | Описание |
| 0-99 | Системные ошибки. |
| 110-119 | Ошибки чтения и открытия файлов. |
| 120-199 | Ошибки на лексического анализа. |
| 200-299 | Ошибки синтаксического анализа. |
| 300-399 | Ошибки семантического анализа. |
| 400-999 | Зарезервированные ошибки. |

## 1.25 Контрольный пример

Контрольный пример, написанный на языке CES-2020, 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.

Трансляция исходного кода в язык ассемблера разделена на 4 этапа: Лексический анализ, синтаксический анализ, семантический анализ и генерация кода.

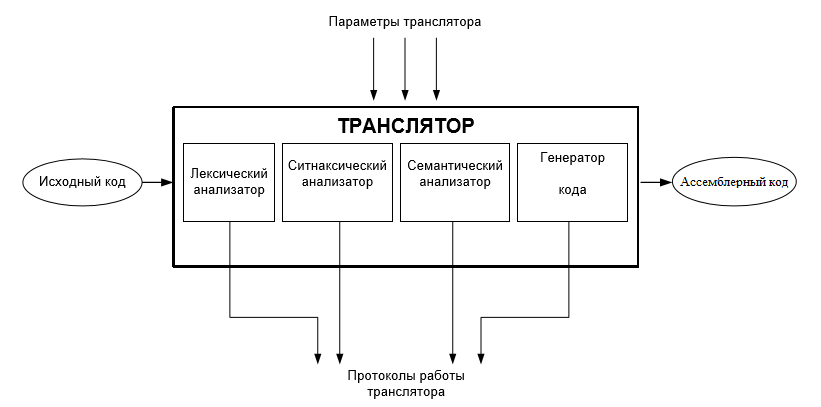


Рисунок 2.1 – Структура транслятора языка CES-2020

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 анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

## 2.2 Перечень входных параметров транслятора

Входные параметры, которые можно передать транслятору языка CES-2020, представлены в таблице 2.1.

Таблица 2.1 – Параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Описание параметра | Значение по умолчанию |
| /in:<путь\_к\_файлу> | Предназначен для определения местонахождения файла с исходным кодом, для проверки и разбора его на лексическом анализаторе.  Указание данного параметра обязательно. | Не предусмотрено |
| /out:<путь\_к\_файлу> | Предназначен для указания имени файла, в который будет производиться генерация исходного кода. | <имя\_файла in>.out |
| /log:<путь\_к\_файлу> | Содержит краткую информацию об исходном коде на языке CES-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла in>.log |
| /an:<путь\_к\_файлу> | Содержит информацию, собранную на лексическом и синтаксическом анализах, а именно таблицу лексем, таблицу идентификаторов, дерево разбора, а также обновленные таблицу лексем и таблицу идентификаторов после преобразования выражений в обратную польскую запись. | <имя\_файла\_in>.an |

## 2.3 Протоколы, формируемые транслятором

Протоколы, формируемые транслятором языка CES-2020 приведены в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка CES-2020

|  |  |
| --- | --- |
| 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 выведен номер ошибки. |
| Файл, заданный параметром “/an:” | Содержит таблицу лексем, таблицу идентификаторов, итог работы лексического анализа, также обновлённые таблицу лексем и идентификаторов после преобразования выражений в обратную польскую запись. |
| Файл, заданный параметров “/out:” | Содержит сгенерированный код на языке Ассемблера. |

# 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.

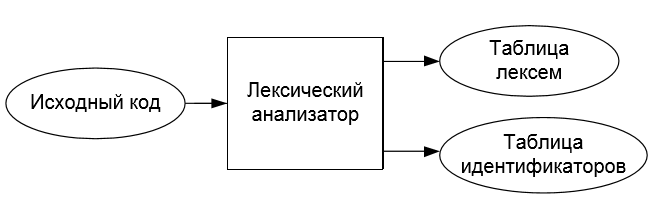


Рисунок 3.1 – Структурная схема лексического анализатора

Лексический анализ в языке CES-2020 состоит из двух частей:

1. Разбиение исходного кода программы на слова (токены).
2. Идентификация слов (токенов) и последующая их замена на лексемы (внутренне представление). Заполнение таблиц лексем и идентификаторов.

Входные данные: исходный код.

Результат работы: Таблица лексем и таблица идентификаторов.

## 3.2 Контроль входных символов

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

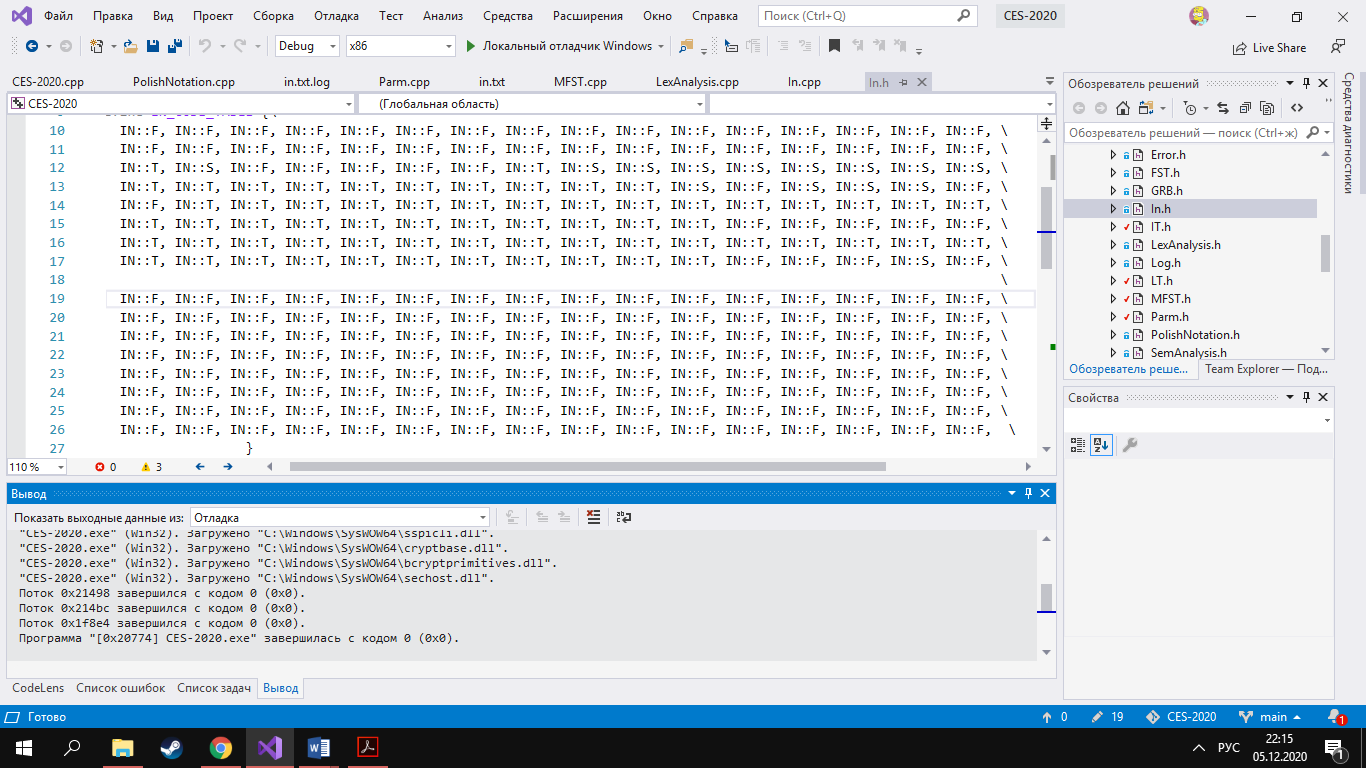
Таблица для проверки входных символов представлена на рисунке 3.2.

Рисунок 3.2 – Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице Windows-1251.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – сепаратор.

## 3.3 Удаление избыточных символов

Избыточный символ – это символ, отсутствие которого никоем образом не влияет на исходный текст программы. В языке CES-2020 символы пробела (0x20) и табуляции (0x09) являются избыточными. Они игнорируются при считывании из файла исходного кода на языке CES-2020.

## 3.4 Перечень ключевых слов

Ключевые слова языка CES-2020, сепараторы, символы операций, соответствующие им лексемы и регулярные выражения приведены в таблице 3.1.

Таблица 3.1 – Все ключевые слова, сепараторы и т.д. языка CES-2020

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Описание |
| usint, string, boolean, symbol | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – не более 10 символов. |
| Литерал | l | Литерал любого доступного типа. |
| Функция статической библиотеки | x | Функции, расположенные в стандартной библиотеке CESlib. |
| DEF | f | Объявление функции. |
| RET | r | Выход из функции/процедуры. |
| HEAD | m | Главная функция. |
| new | n | Объявление переменной. |
| out | o | Оператор вывода. |
| WHEN | w | Условная конструкция. |
| OTHWERWISE | e |
| ENDW | ] | Конец условного блока.` |
| OPEN | { | Начало блока функции. |
| CLOSE | } | Конец блока функции. |
| ( | ) | Открывающая скобка. |
| ) | ) | Закрывающая скобка. |
| +, -, /, \*, ~>, <~ | v | Арифметические и побитовые операторы. |
| =, !, <, > | z | Логические операторы. |
| , | , | Разделитель параметров функции. |
| : | : | Оператор присваивания. |
| . | . | Конец инструкции. |

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.

Таблица лексем и таблица идентификаторов контрольного примера языка CES-2020, а также детерминированные конечные автоматы представлены в приложении А.

## 3.5 Основные структуры данных

Основными структурами данных лексического анализатора языка CES-2020 являются таблица лексем и таблица идентификаторов. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки и позиции в исходном коде, индекс в таблице идентификаторов, а также приоритет операции и её знак, если лексема является операцией. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, а также его значение.

## 3.6. Структура и перечень сообщений лексического анализатора

В языке CES-2020 для обработки ошибок лексический анализатор использует таблицу с сообщениями, структура которой представлена в Приложении Б. Структура сообщений содержит номер ошибки, вид ошибки, её сообщение, а также номер строки и позиции возникшей ошибки.

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-199. Текст ошибки содержит в себе префикс [LexA]. Перечень сообщений лексического анализатора представлен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 120 | Цепочка символов не разобрана. |
| 121 | Переполнение таблицы лексем. |
| 122 | Переполнение таблицы идентификаторов. |

## 3.7. Принцип обработки ошибок

В языке CES-2020 при обнаружении ошибки в исходном коде программы лексический анализатор формирует сообщение об ошибке и выводит его в файл с протоколом работы, заданный параметром /log, а также в консоль.

## 3.8. Параметры лексического анализатора

В ходе своей работы лексический анализатор использует входные параметры транслятора /in и /log, которые описаны в таблице 2.1.

## 3.9. Алгоритм лексического анализа

Лексический анализ – первая и наиболее простая фаза трансляции.

Алгоритм работы лексического анализатора заключается распознавании и разборе цепочек исходного кода. Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно представить в виде графа.

Пример графа конечного автомата представлен на рисунке 3.3. На рисунке осуществляется разбор двоичного целочисленного литерала, где S0 – начальное состояние, а S2 – конечное.

0, 1

b

0, 1

Рисунок 3.3 – Граф переходов для двоичного целочисленного литерала.

## 3.10. Контрольный пример

Результат работы лексического анализатора, полученный при выполнении контрольного примера, представлен в приложении А.

# 4. **Разработка синтаксического анализатора**

## 4.1. Структура синтаксического анализатора

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходом - дерево разбора.

В языке CES-2020 синтаксический анализ выполняется после завершения работы лексического анализатора.

Структура синтаксического анализатора представлена на рисунке 4.1.

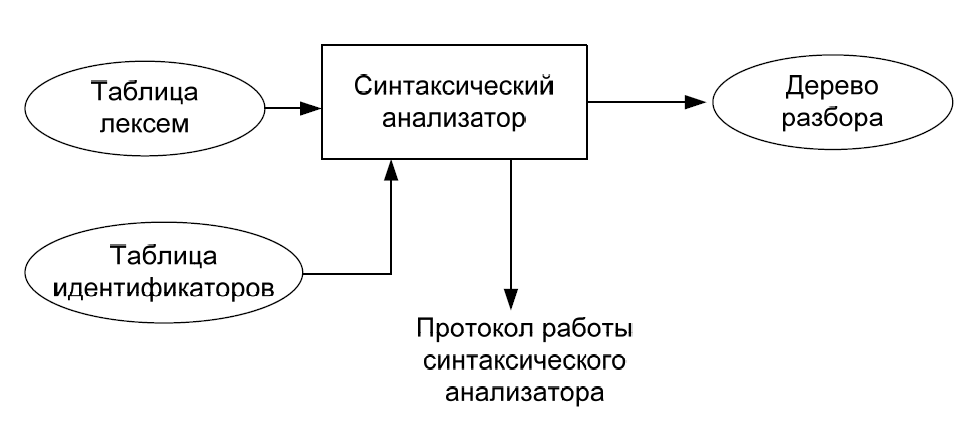


Рисунок 4.1 – Структура синтаксического анализатора

## 4.2. Контекстно-свободная грамматика, описывающая синтаксис языка

Синтаксис языка CES-2020 описывается при помощи грамматики типа 2 иерархии Хомского (Контекстно-свободной грамматики).

Контекстно-свободная грамматика – грамматика типа 2 по иерархии Хомского. Данная грамматика имеет вид , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил переходов,

S – стартовый символ.

В контекстно-свободной грамматике правила имеют вид:

, где

,

,

 - словарь грамматики .

Грамматика языка CES-2020 представлена в таблице 4.1 в виде структуры на языке С++. Терминальные символы помечены символами TS. Нетерминальные символы помечены символами NS.

Табл. 4.1 – Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминал | Соответствующее правило | Описание |
| S | m{N} | tfi(F){NrO.}S | Стартовый символ. Правила, порождающие главную и пользовательскую функции. |
| N | nti; | nti;N |  rO. | rO.N |  i:E. | i:E.N |  oO. | oO.N |  w(B)C] | w(B)C]N | Правила, порождающие конструкции в функциях. |
| C | i:E.ei:E. | i:E.  oO.eoO. | oO. | Правила, порождающие тело условной конструкции. |
| B | i | l |  izO | lzO | Правила порождающие выражения в условных конструкциях. |
| E | i | iM  l | lM  (E) | (E)M  i(W) | i(W)M  x() | x()M | Правила, порождающие выражения. |
| F | ti | ti,F | Правила, порождающие формальные параметры функции |
| W | i | i,W  l | l,W | Правила, порождающие принимаемые параметры функции |
| M | vE | vEM | zO | Правила, порождающие арифметические и логические действия. |
| O | i | l | Правила, порождающие литералы и идентификаторы в операторе вывода. |

## 4.3. Построение конечного магазинного автомата

В языке CES-2020 Конечный автомат с магазинной памятью представляет собой семерку вида

, где

Q – множество состояний автомата,

V – алфавит входных символов,

Z – алфавит специальных магазинных символов,

 - функция переходов автомата,

 - начальное состояние автомата,  - начальное состояние магазина, F – множество конечных состояний.

Схема конечного автомата с магазинной памятью представлена на рисунке 4.2.

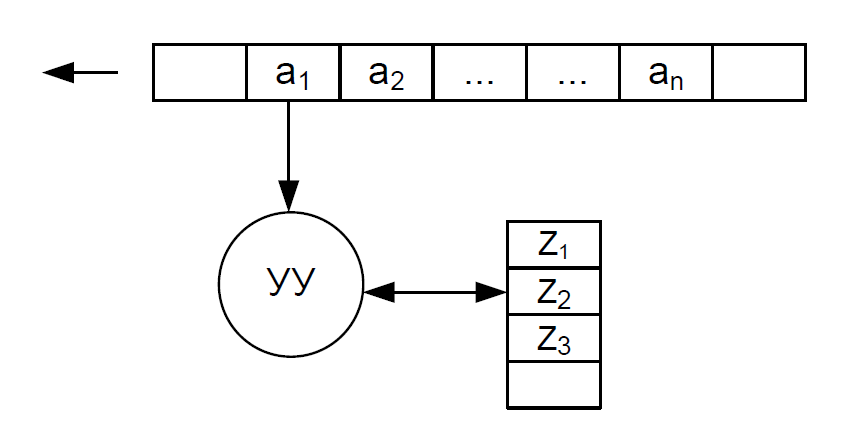


Рисунок 4.2 – Схема конечного автомата с магазинной памятью

Описание компонентов магазинного автомата представлено в таблице 4.2.

|  |  |  |
| --- | --- | --- |
| Таблица 4.2 – Описание компонентов магазинного автомата | | |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Структура, содержащая позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Множество терминальных и нетерминальных символов, описание которых содержится в таблицах 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 | Представляет из себя символ маркера дна стека $ |
|  | Множество конечных состояний | Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## 4.4. Основные структуры данных

Основными структуры данных синтаксического анализатора языка CES-2020 являются автомат с магазинной памятью и структура грамматики Грейбах, описывающей правила языка CES-2020. Данные структуры представлены в приложении В.

## 4.5. Описание алгоритма синтаксического разбора

Алгоритм работы синтаксического анализатора в языке CES-2020:

1. В стек магазинного автомата кладётся маркер дна и стартовый символ.
2. На основе полученной ранее таблицы лексем формируется входная лента.
3. Автомат запускается.
4. Выбирается цепочка по первому символу, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.
5. Если терминалы в стеке и в ленте совпадают, то данный терминал выталкивается из стека магазинного автомата и лента сдвигается на одну позицию вправо. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другое правило нетерминала.
6. Если в правиле встретился нетерминал - переходим к пункту 4.
7. Если в вершине стека лежит маркер дна и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе мы получаем ошибку синтаксического анализа.

## 4.6. Структура и перечень сообщений синтаксического анализатора

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 200-299. Текст ошибки содержит в себе префикс [SynA]. Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Перечень сообщений синтаксического анализатора языка CES-2020 представлен в таблице 4.3.

Табл. 4.3 - Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 200 | Неверная структура программы. |
| 201 | Ошибочный сепаратор. |
| 202 | Ошибка в выражении. |
| 203 | Ошибка в параметрах функции. |
| 204 | Ошибка в параметрах вызываемой функции. |
| 205 | Ошибка в условном операторе. |
| 206 | Синтаксический анализ завершен с ошибками. |

## 4.7. Параметры синтаксического анализатора и режимы его работы

В языке CES-2020 специальные параметры для управления режимом работы синтаксического анализатора не предусмотрены. Результат работы лексического анализатора выводится в файл, указанный параметром /an.

## 4.8. Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Все ошибки записываются в общую структуру ошибок.
4. В случае нахождения ошибки в протокол будет выведено диагностическое сообщение.

## 4.9. Контрольный пример

Результатом работы синтаксического анализатора, является дерево разбора контрольного примера, представленного в приложении А. Протокол разбора и дерево разбора исходного кода представлены в приложении Г, а также графическое представление дерева разбора в Графической работе №1.

Протокол разбора представляет собой вывод результата пошаговой работы конечного автомата с магазинной памятью.

В файл, указанный параметром /an выводится номер текущего шага, текущее рассматриваемое правило, входная лента и содержимое стека.

# 5. Разработка семантического анализатора

## 5.1. Структура семантического анализатора

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Структура семантического анализатора языка CES-2020 представлена на рисунке 5.1.

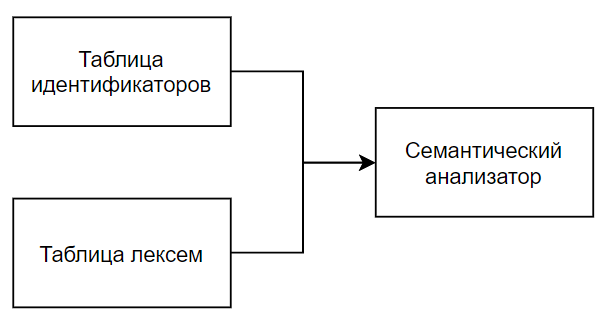


Рисунок 5.1 – Структура семантического анализатора языка CES-2020

## 5.2. Функции семантического анализатора

Семантический анализатор языка CES-2020 выполняет проверку на основные правила языка (семантики языка), описанные в разделе 1.16.

## 5.3. Структура и перечень сообщений семантического анализатора

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 300-399. Текст ошибки содержит в себе префикс [SemA]. Перечень сообщений семантического анализатора представлен в таблице 5.1

Перечень сообщений, формируемых семантическим анализатором, представлены в таблице 5.1.

Таблица 5.1 – Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 300 | Переопределение идентификатора. |
| 301 | Несоответствие возвращаемого значения функции с её типом. |
| 302 | Переданы неверные типы параметров. |
| 303 | Деление на 0 в выражении. |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| 304 | Вызов функции не в точке входа. |
| 305 | Превышено максимальное количество функций в программе. |
| 306 | Необъявленный идентификатор |
| 307 | Несоответствие типов в операторе присваивания. |
| 308 | Более одной главной функции HEAD. |
| 309 | Имя пользовательской функции не должно совпадать с именем функций CESlib. |
| 310 | Отсутствие главной функции HEAD. |
| 311 | Объявление идентификатора без его использования. |
| 312 | Применение разных типов операторов в одном выражении. |
| 313 | Превышено максимальное количество параметров в функции. |
| 314 | Значение целочисленного литерала не входит в диапазон допустимых значений. |
| 315 | Превышена максимальная длина строкового литерала. |
| 316 | Несоответствие операндов с операцией. |
| 317 | Неверное выражение в условной конструкции. |
| 318 | Передано неверное количество параметров. |

## 5.4. Принцип обработки ошибок

Семантический анализатор языка CES-2020 выполняет проверки в соответствии с пунктом 1.16. Работа транслятора, в случае возникновения ошибки, прекращается, а сообщение об ошибке выводится в файл, заданный параметром /log.

## 5.5. Контрольный пример

Результат работы контрольного примера представлен в приложении А, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

# 6. **Вычисление выражений**

## 6.1. Выражения, допускаемые языком

В языке CES-2020 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений, а также допускаются односоставные логические выражения с переменными или литералами целочисленного типа. Применение логических и арифметических операций в одном выражении не допускается. Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке CES-2020

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| , | 1 |
| ( | 4, если это вызов функции, иначе 0. |
| ) | 4, если это вызов функции, иначе 0. |
| + | 2 |
| - | 2 |
| \* | 3 |
| / | 3 |
| ~> | 3 |
| <~ | 3 |

## 6.2. Польская запись и принцип ее построения

Обратная польская запись – форма записи выражений, в которой операнды расположены перед знаками операций.

Язык CES-2020 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 преобразования выражений в польскую запись:

* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку в порядке их следования;
* операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* открывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку;
* если идентификатор является именем функции, то он заменяется на специальный символ «@».

## 6.3. Программная реализация обработки выражений

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

## 6.4. Контрольный пример

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

# 7. **Генерация кода**

## 7.1 Структура генератора кода

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

Выходным будет являться файл, генерируемый во время компиляции, который будет содержать в себе транслированный исходный код.

Структура генератора кода CES-2020 представлена на рисунке 7.1.

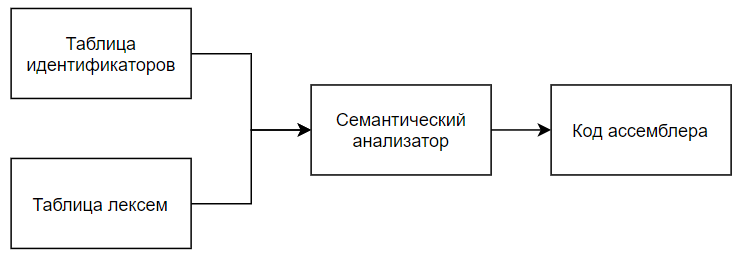


Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

В языке CES-2020 элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – сегменте данных и сегменте констант. Идентификаторы языка размещены в сегменте данных. Литералы – в сегменте констант. Соответствия между типами данных идентификаторов на языке CES-2020 и на языке ассемблера представлены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка CES-2020 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке CES-2020 | Тип идентификатора на языке ассемблера | Описание |
| usint | sdword | Хранит целочисленный тип данных. |
| string | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |
| boolean | dword | Хранит логический тип данных. |
| symbol | dword | Хранит символьный тип данных. |
| L[0-1]+ | byte  dword | Литералы: byte – для строковых, dword – для целочисленных, символьных и логических. |

## 7.3 Статическая библиотека

Функции стандартной библиотеки языка CES-2020, представлены в таблице 1.7.

Статическая библиотека написана на языке C++. Подключение статической библиотеки производится на этапе генерации кода.

## 7.4 Особенности алгоритма генерации кода

Генерация кода в языке CES-2020 производится на основе таблиц лексем и таблиц идентификаторов.

Генерация кода в языке CES-2020 производится на основе таблиц лексем и таблиц идентификаторов.

Алгоритм генерации исходного кода языка CES-2020:

* в файл, in.txt.asm записывается информация о модели памяти, используемом соглашении о вызовах, подключаются необходимые библиотеки и объявляются прототипы функций;
* проходим по таблице идентификаторов и в сегмент констант записываем литералы;
* проходим по таблице лексем и ищем объявление переменных, заполняем сегмент данных этими переменными;
* снова проходим по таблице лексем, заполняя уже сегмент кода.

## 7.5 Входные параметры генератора кода

Входными параметрами генератора в языке CES-2020 являются таблица идентификаторов и таблица лексем, которые предназначены для генерации кода ассемблера. Результат работы генератора кода выводится в файл, указанный параметром /out.

## 7.6 Контрольный пример

Контрольный пример языка CES-2020, сгенерированный в язык ассемблера, представлен в приложении Е.

# 8. Тестирование транслятора

## 8.1 Общие положения

В языке CES-2020 исходный код проходит проверку на наличие соответствующих ошибок на всех этапах трансляции. При обнаружении ошибки работа транслятора прекращается и выводится соответствующее сообщение об ошибке на консоль и в файл, указанный параметром /log.

## 8.2 Тестирование проверки на допустимость символов

В языке CES-2020 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа представлен в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| HEAD  OPEN  new symbol ь.  CLOSE | Ошибка 111: [IN] Недопустимый символ в исходном файле (/in), строка 3, позиция 11. |

## 8.3 Тестирование лексического анализатора

На этапе лексического анализа в языке CES-2020 могут возникнуть ошибки, описанные в пункте 3.6. Результаты тестирования лексического анализатора представлены в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| HEAD  OPEN  new using a.  a:8o.  CLOSE | Ошибка 120: [LexA] Цепочка символов не разобрана, строка 4, позиция 2 |

Также возможны ещё две семантические ошибки:

1. Если количество строк в таблице лексем превышает 4096, то компилятор выдаст ошибку вида: «Ошибка 121: [LexA] Переполнение таблицы лексем, строка -1, позиция -1;
2. Если количество строк в таблице идентификаторов превышает 4096, то компилятор выдаст ошибку вида: «Ошибка 122: [LexA] Переполнение таблицы идентификаторов, строка -1, позиция -1;

## 8.4 Тестирование синтаксического анализатора

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора представлены в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| HEAD CLOSE | Ошибка 206: [SynA] Синтаксический анализ завершён с ошибками(подробности в log), строка -1, позиция -1. |

## 8.5 Тестирование семантического анализатора

На этапе семантического анализа могут возникнуть ошибки, описанные в пункте 5.3. Результаты тестирования синтаксического анализатора представлены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| HEAD OPEN  new symbol c.  new symbol c. CLOSE | Ошибка 300: [SemA] Переопределение идентификатора, строка 4, позиция 11. |
| usint DEF mul(usint p1, usint p2)  OPEN  new usint ans.  ans:p1\*p2.  RET 'a'.  CLOSE | Ошибка 301: [SemA] Несоответствие возвращаемого значения функции с её типом, строка 5, позиция 4. |
| usint DEF mul(usint p1, usint p2)  OPEN  new usint ans.  ans:p1\*p2.  RET ans.  CLOSE  HEAD OPEN  mul(5,`hello`). CLOSE | Ошибка 302: [SemA] Переданы неверные типы параметров, строка 9, позиция 13. |
| HEAD  OPEN  new usint a.  a:6/0.  CLOSE | Ошибка 303: [SemA] деление на 0 в выражении, строка 4, позиция 3. |

Продолжение таблицы 8.4

|  |  |
| --- | --- |
| string DEF test(string p1)  OPEN  p1:test(`error`).  RET `a`.  CLOSE  HEAD  OPEN  new string s.  s:test(`jjj`).  CLOSE | Ошибка 304: [SemA] Вызов функции не в точке входа, строка 3, позиция 2 |
| HEAD OPEN  out a.  CLOSE | Ошибка 306: [SemA] Необъявленный идентификатор, строка 3, позиция 4. |
| HEAD  OPEN  new symbol a.  a:2.  CLOSE | Ошибка 307: [SemA] Несоответствие типов в операторе присваивания, строка 4, позиция 2 |
| HEAD  OPEN  out 1.  CLOSE  HEAD  OPEN  out 2.  CLOSE | Ошибка 308: [SemA] Более одной главной функции HEAD, строка -1, позиция -1 |
| string DEF DATE(string a)  OPEN  new string ans.  ans:a.  RET ans.  CLOSE  HEAD  OPEN  CLOSE | Ошибка 309: [SemA] Имя пользовательской функции не должно совпадать с именем функций CESlib, строка 1, позиция 11 |
| usint DEF test(usint a)  OPEN  new usint ans.  ans:a.  RET ans.  CLOSE | Ошибка 310: [SemA] Отсутствие главной функции HEAD, строка -1, позиция -1. |

Продолжение таблицы 8.4

|  |  |
| --- | --- |
| HEAD OPEN  new usint useless.  CLOSE | Ошибка 311: [SemA] Объявление идентификатора без его использования, строка 3, позиция 0. |
| HEAD OPEN  new usint var.  var: 2+1<2.  CLOSE | Ошибка 312: [SemA] Применение разных типов операторов в одном выражении, строка 4, позиция 5. |
| usint DEF test(usint a, usint b, usint c, usint d, usint e, usint f)  OPEN  new usint ans.  ans:0.  out ans.  CLOSE  HEAD  OPEN  new usint ans.  ans:test(1, 2, 3, 4, 5, 6).  CLOSE | Ошибка 313: [SemA] Превышено максимальное количество параметров в функции, строка 1, позиция 10. |
| HEADOPEN  out 401o.  CLOSE | Ошибка 314: [SemA] Значение целочисленного литерала не входит в диапазон допустимых значений, строка 3, позиция 4. |
| HEAD OPEN  new string str.  str:`a`+`b`.  CLOSE | Ошибка 316: [SemA] Несоответствие операндов с операцией, строка 4, позиция 7. |
| HEAD OPEN  WHEN(`error`)  out `test`.  ENDW  CLOSE | Ошибка 317: [SemA] Неверное выражение в условной конструкции, строка 3, позиция 5. |
| usint DEF test(usint a, usint b)  OPEN  new usint ans.  ans:a+b.  RET ans.  CLOSE  HEAD  OPEN  new usint t.  t:test(2,5,8).  CLOSE | Ошибка 318: [SemA] Передано неверное количество параметров, строка 10, позиция 12. |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор языка CES-2020 в язык ассемблера и написана пояснительная записка со спецификацией языка.

Таким образом, были выполнены основные задачи данной курсовой работы:

* + Сформулирована спецификация языка CES-2020;
  + Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
  + Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
  + Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
  + Осуществлена программная реализация синтаксического анализатора;
  + Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
  + Разработан транслятор кода на язык ассемблера;
  + Проведено тестирование всех вышеперечисленных компонентов.

Количественные и качественные характеристики реализации транслятора:

* Количество типов данных: 4;
* Количество инструкция языка: 4;
* Количество лексем: 21;
* Правил грамматики: 9;
* Наличие стандартной библиотеки;
* Количество строк на языке реализации: 2833.

По итогу выполнения курсового проекта по дисциплине «Языки программирования» были приобретены навыки разработки системы программирования (трансляторов, интерпретаторов), изучены основы теории формальных грамматик и основы общей теории компиляторов.

# Список использованных источников

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

# Приложение А

Листинг 1 – Исходный код программы на языке CES-2020

usint DEF mul(usint p1, usint p2)

OPEN

new usint ans.

ans: p1\*p2.

RET ans.

CLOSE

usint DEF max(usint p1, usint p2)

OPEN

new boolean a.

new usint ans.

a: p1 > p2.

WHEN(a)

ans:p1.

OTHERWISE

ans:p2.

ENDW

RET ans.

CLOSE

HEAD

OPEN

new symbol c.

c: 'a'.

out c.

new string s.

s: `CES-2020`.

out s.

WHEN(6!2)

s: DATE().

OTHERWISE

s: TIME().

ENDW

out s.

new usint q.

new usint w.

q: max(11o,5o) + max(1011b,1111b).

w: (3d<~2d) / (2 + q).

e: mul(q, w).

out e.

CLOSE

Листинг 2 – Таблица лексем и таблица идентификаторов контрольного примера

1: tfi(ti,ti)

2: {

3: nti.

4: i:ivi.

5: ri.

6: }

7: tfi(ti,ti)

8: {

9: nti.

10: nti.

11: i:izi.

12: w(i)

13: i:i.

14: e

15: i:i.

16: ]

17: ri.

18: }

19: m

20: {

21: nti.

22: i:l.

23: oi.

24: nti.

25: i:l.

26: oi.

27: w(lzl)

28: i:x().

29: e

30: i:x().

31: ]

32: oi.

33: nti.

34: nti.

35: i:i(l,l)vi(l,l).

36: i:(lvl)v(lvi).

37: nti.

38: i:i(i,i).

39: oi.

40: }

---------------------------------------------------------------------------------

№ id | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение

---------------------------------------------------------------------------------

0 | mul | usint | функция | 2 | null

1 | mulp1 | usint | параметр | 5 | null

2 | mulp2 | usint | параметр | 8 | null

3 | mulans | usint | переменная | 13 | 0

4 | max | usint | функция | 27 | null

5 | maxp1 | usint | параметр | 30 | null

6 | maxp2 | usint | параметр | 33 | null

7 | maxa | boolean | переменная | 38 | false

8 | maxans | usint | переменная | 42 | 0

9 | HEADc | symbol | переменная | 72 | '0'

10 | L0 | symbol | литерал | 76 | 'a'

11 | HEADs | string | переменная | 83 | ``(0)

12 | L1 | string | литерал | 87 | `CES-2020`(8)

13 | L2 | usint | литерал | 94 | 6

14 | L3 | usint | литерал | 96 | 2

15 | DATE | string | функция | 100 | null

16 | TIME | string | функция | 107 | null

17 | HEADq | usint | переменная | 117 | 0

18 | HEADw | usint | переменная | 121 | 0

19 | L4 | usint | литерал | 127 | 9

20 | L5 | usint | литерал | 129 | 5

21 | L6 | usint | литерал | 134 | 11

22 | L7 | usint | литерал | 136 | 15

23 | L8 | usint | литерал | 173 | 3

24 | HEADe | usint | переменная | 186 | 0

Листинг 3 – Детерминированные конечные автоматы языка CES-2020

FST fstOperators(LEX\_OPERATOR, OPERATION, "", 2,

NODE(4, RELATION('+', 1), RELATION('-', 1),

RELATION('\*', 1), RELATION('/', 1)),

NODE()

);

FST fstLogicOperators(LEX\_LOG\_OPERATOR, LOG\_OPERATION, "", 2,

NODE(4, RELATION('<', 1), RELATION('>', 1),

RELATION('!', 1), RELATION('=', 1)),

NODE()

);

FST fstLeftHesis(LEX\_LEFTHESIS, NEED\_PRIORITY, "", 2,

NODE(1, RELATION('(', 1)),

NODE()

);

FST fstRightHesis(LEX\_RIGHTHESIS, NEED\_PRIORITY, "", 2,

NODE(1, RELATION(')', 1)),

NODE()

);

FST fstAssign(LEX\_ASSIGN, DEF\_LEX, "", 2,

NODE(1, RELATION(':', 1)),

NODE()

);

FST fstSemicolon(LEX\_DOT, DEF\_LEX, "", 2,

NODE(1, RELATION('.', 1)),

NODE()

);

FST fstComma(LEX\_COMMA, NEED\_PRIORITY, "", 2,

NODE(1, RELATION(',', 1)),

NODE()

);

FST fstUsintDecLit(LEX\_USINT, LITERAL, "", 2,

NODE(30, RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),

RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),

RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),

RELATION('9', 1)), NODE()

);

FST fstId(LEX\_ID, ID, "", 2,

NODE(26,

RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),

RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),

RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),

RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),

RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),

RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),

RELATION('y', 1), RELATION('z', 1)),NODE());

FST fstUsint(LEX\_USINT, DEF\_LEX, "", 6,

NODE(1, RELATION('u', 1)),

NODE(1, RELATION('s', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('n', 4)),

NODE(1, RELATION('t', 5)),

NODE()

);

FST fstString(LEX\_STRING, DEF\_LEX, "", 7,

NODE(1, RELATION('s', 1)),

NODE(1, RELATION('t', 2)),

NODE(1, RELATION('r', 3)),

NODE(1, RELATION('i', 4)),

NODE(1, RELATION('n', 5)),

NODE(1, RELATION('g', 6)),

NODE()

);

FST fstBoolean(LEX\_BOOLEAN, DEF\_LEX, "", 8,

NODE(1, RELATION('b', 1)),

NODE(1, RELATION('o', 2)),

NODE(1, RELATION('o', 3)),

NODE(1, RELATION('l', 4)),

NODE(1, RELATION('e', 5)),

NODE(1, RELATION('a', 6)),

NODE(1, RELATION('n', 7)),

NODE()

);

FST fstSymbol(LEX\_SYMBOL, DEF\_LEX, "", 7,

NODE(1, RELATION('s', 1)),

NODE(1, RELATION('y', 2)),

NODE(1, RELATION('m', 3)),

NODE(1, RELATION('b', 4)),

NODE(1, RELATION('o', 5)),

NODE(1, RELATION('l', 6)),

NODE()

);

FST fstFunc(LEX\_DEF, DEF\_LEX, "", 4,

NODE(1, RELATION('D', 1)),

NODE(1, RELATION('E', 2)),

NODE(1, RELATION('F', 3)),

NODE()

);

FST fstVariable(LEX\_NEW, DEF\_LEX, "", 4,

NODE(1, RELATION('n', 1)),

NODE(1, RELATION('e', 2)),

NODE(1, RELATION('w', 3)),

NODE()

);

FST fstHead(LEX\_HEAD, DEF\_LEX, "", 5,

NODE(1, RELATION('H', 1)),

NODE(1, RELATION('E', 2)),

NODE(1, RELATION('A', 3)),

NODE(1, RELATION('D', 4)),

NODE()

);

FST fstLeftBrace(LEX\_OPEN, DEF\_LEX, "", 5,

NODE(1, RELATION('O', 1)),

NODE(1, RELATION('P', 2)),

NODE(1, RELATION('E', 3)),

NODE(1, RELATION('N', 4)),

NODE()

);

FST fstRightBrace(LEX\_CLOSE, DEF\_LEX, "", 6,

NODE(1, RELATION('C', 1)),

NODE(1, RELATION('L', 2)),

NODE(1, RELATION('O', 3)),

NODE(1, RELATION('S', 4)),

NODE(1, RELATION('E', 5)),

NODE()

);

FST fstOut(LEX\_OUT, DEF\_LEX, "", 4,

NODE(1, RELATION('o', 1)),

NODE(1, RELATION('u', 2)),

NODE(1, RELATION('t', 3)),

NODE()

);

FST fstRet(LEX\_RET, DEF\_LEX, "", 4,

NODE(1, RELATION('R', 1)),

NODE(1, RELATION('E', 2)),

NODE(1, RELATION('T', 3)),

NODE()

);

FST fstSymbolLit(LEX\_SYMBOL, LITERAL, "", 4,

NODE(1, RELATION('\'', 1)),

NODE(138,

RELATION('a', 2), RELATION('b', 2), RELATION('c', 2), RELATION('d', 2),

RELATION('e', 2), RELATION('f', 2), RELATION('g', 2), RELATION('h', 2),

RELATION('i', 2), RELATION('j', 2), RELATION('k', 2), RELATION('l', 2),

RELATION('m', 2), RELATION('n', 2), RELATION('o', 2), RELATION('p', 2),

RELATION('q', 2), RELATION('r', 2), RELATION('s', 2), RELATION('t', 2),

RELATION('u', 2), RELATION('v', 2), RELATION('w', 2), RELATION('x', 2),

RELATION('y', 2), RELATION('z', 2), RELATION('1', 2), RELATION('2', 2),

RELATION('3', 2), RELATION('4', 2), RELATION('5', 2), RELATION('6', 2),

RELATION('7', 2), RELATION('8', 2), RELATION('9', 2), RELATION('0', 2),

RELATION('A', 2), RELATION('B', 2), RELATION('C', 2), RELATION('D', 2),

RELATION('E', 2), RELATION('F', 2), RELATION('G', 2), RELATION('H', 2),

RELATION('I', 2), RELATION('J', 2), RELATION('K', 2), RELATION('L', 2),

RELATION('M', 2), RELATION('N', 2), RELATION('O', 2), RELATION('P', 2),

RELATION('Q', 2), RELATION('R', 2), RELATION('S', 2), RELATION('T', 2),

RELATION('U', 2), RELATION('V', 2), RELATION('W', 2), RELATION('X', 2),

RELATION('Y', 2), RELATION('Z', 2),

RELATION('а', 2), RELATION('б', 2), RELATION('в', 2), RELATION('г', 2),

RELATION('д', 2), RELATION('е', 2), RELATION('ё', 2), RELATION('ж', 2),

RELATION('з', 2), RELATION('и', 2), RELATION('й', 2), RELATION('к', 2),

RELATION('л', 2), RELATION('м', 2), RELATION('н', 2), RELATION('о', 2),

RELATION('п', 2), RELATION('р', 2), RELATION('с', 2), RELATION('т', 2),

RELATION('у', 2), RELATION('ф', 2), RELATION('х', 2), RELATION('ц', 2),

RELATION('ч', 2), RELATION('ш', 2), RELATION('щ', 2), RELATION('ъ', 2),

RELATION('ы', 2), RELATION('ь', 2), RELATION('э', 2), RELATION('ю', 2),

RELATION('я', 2), RELATION('А', 2), RELATION('Б', 2), RELATION('В', 2),

RELATION('Г', 2), RELATION('Д', 2), RELATION('Е', 2), RELATION('Ё', 2),

RELATION('Ж', 2), RELATION('З', 2), RELATION('И', 2), RELATION('Й', 2),

RELATION('К', 2), RELATION('Л', 2), RELATION('М', 2), RELATION('Н', 2),

RELATION('О', 2), RELATION('П', 2), RELATION('Р', 2), RELATION('С', 2),

RELATION('Т', 2), RELATION('У', 2), RELATION('Ф', 2), RELATION('Х', 2),

RELATION('Ц', 2), RELATION('Ч', 2), RELATION('Ш', 2), RELATION('Щ', 2),

RELATION('Ъ', 2), RELATION('Ы', 2), RELATION('Ь', 2), RELATION('Э', 2),

RELATION('Ю', 2), RELATION('Я', 2), RELATION(' ', 2), RELATION('.', 2),

RELATION(',', 2), RELATION('?', 2), RELATION('!', 2), RELATION(';', 2),

RELATION(':', 2), RELATION('-', 2), RELATION(')', 2), RELATION('(', 2)),

NODE(1, RELATION('\'', 3)),

NODE()

);

FST fstUsintDecLit(LEX\_USINT, LITERAL, "", 3,

NODE(30,

RELATION('0', 0), RELATION('1', 0), RELATION('2', 0),

RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),

RELATION('6', 0), RELATION('7', 0), RELATION('8', 0),

RELATION('9', 0),

RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),

RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),

RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),

RELATION('9', 1),

RELATION('0', 2), RELATION('1', 2), RELATION('2', 2),

RELATION('3', 2), RELATION('4', 2), RELATION('5', 2),

RELATION('6', 2), RELATION('7', 2), RELATION('8', 2),

RELATION('9', 2)),

NODE(1, RELATION('d', 2)),

NODE()

);

FST fstUsintOctLit(LEX\_USINT, LITERAL, "", 3,

NODE(16,

RELATION('0', 0), RELATION('1', 0), RELATION('2', 0),

RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),

RELATION('6', 0), RELATION('7', 0),

RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),

RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),

RELATION('6', 1), RELATION('7', 1)),

NODE(1, RELATION('o', 2)),

NODE()

);

FST fstUsintBinLit(LEX\_USINT, LITERAL, "", 3,

NODE(4,

RELATION('0', 0), RELATION('1', 0),

RELATION('0', 1), RELATION('1', 1)),

NODE(1, RELATION('b', 2)),

NODE()

);

FST fstBooleanLit(LEX\_BOOLEAN, LITERAL, "", 9,

NODE(2, RELATION('t', 1), RELATION('f', 4)),

NODE(1, RELATION('r', 2)),

NODE(1, RELATION('u', 3)),

NODE(1, RELATION('e', 8)),

NODE(1, RELATION('a', 5)),

NODE(1, RELATION('l', 6)),

NODE(1, RELATION('s', 7)),

NODE(1, RELATION('e', 8)),

NODE()

);

FST fstId(LEX\_ID, ID, "", 3, // [a-z][a-z | 0-9]\*

NODE(26,

RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),

RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),

RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),

RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),

RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),

RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),

RELATION('y', 1), RELATION('z', 1)),

NODE(72,

RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),

RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),

RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),

RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),

RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),

RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),

RELATION('y', 1), RELATION('z', 1), RELATION('0', 1), RELATION('1', 1),

RELATION('2', 1), RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),

RELATION('6', 1), RELATION('7', 1), RELATION('8', 1), RELATION('9', 1),

RELATION('a', 2), RELATION('b', 2), RELATION('c', 2), RELATION('d', 2),

RELATION('e', 2), RELATION('f', 2), RELATION('g', 2), RELATION('h', 2),

RELATION('i', 2), RELATION('j', 2), RELATION('k', 2), RELATION('l', 2),

RELATION('m', 2), RELATION('n', 2), RELATION('o', 2), RELATION('p', 2),

RELATION('q', 2), RELATION('r', 2), RELATION('s', 2), RELATION('t', 2),

RELATION('u', 2), RELATION('v', 2), RELATION('w', 2), RELATION('x', 2),

RELATION('y', 2), RELATION('z', 2), RELATION('0', 2), RELATION('1', 2),

RELATION('2', 2), RELATION('3', 2), RELATION('4', 2), RELATION('5', 2),

RELATION('6', 2), RELATION('7', 2), RELATION('8', 2), RELATION('9', 2)),

NODE()

);

FST fstStringLit(LEX\_STRING, LITERAL, "", 4,

NODE(2,

RELATION('`', 1),

RELATION('`', 2)),

NODE(276,

RELATION('A', 1), RELATION('B', 1), RELATION('C', 1), RELATION('D', 1),

RELATION('E', 1), RELATION('F', 1), RELATION('G', 1), RELATION('H', 1),

RELATION('I', 1), RELATION('J', 1), RELATION('K', 1), RELATION('L', 1),

RELATION('M', 1), RELATION('N', 1), RELATION('O', 1), RELATION('P', 1),

RELATION('Q', 1), RELATION('R', 1), RELATION('S', 1), RELATION('T', 1),

RELATION('U', 1), RELATION('V', 1), RELATION('W', 1), RELATION('X', 1),

RELATION('Y', 1), RELATION('Z', 1), RELATION('a', 1), RELATION('b', 1),

RELATION('c', 1), RELATION('d', 1), RELATION('e', 1), RELATION('f', 1),

RELATION('g', 1), RELATION('h', 1), RELATION('i', 1), RELATION('j', 1),

RELATION('k', 1), RELATION('l', 1), RELATION('m', 1), RELATION('n', 1),

RELATION('o', 1), RELATION('p', 1), RELATION('q', 1), RELATION('r', 1),

RELATION('s', 1), RELATION('t', 1), RELATION('u', 1), RELATION('v', 1),

RELATION('w', 1), RELATION('x', 1), RELATION('y', 1), RELATION('z', 1),

RELATION('1', 1), RELATION('2', 1), RELATION('3', 1), RELATION('4', 1),

RELATION('5', 1), RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),

RELATION('9', 1), RELATION('0', 1), RELATION('А', 1), RELATION('Б', 1),

RELATION('В', 1), RELATION('Г', 1), RELATION('Д', 1), RELATION('Е', 1),

RELATION('Ё', 1), RELATION('Ж', 1), RELATION('З', 1), RELATION('И', 1),

RELATION('Й', 1), RELATION('К', 1), RELATION('Л', 1), RELATION('М', 1),

RELATION('Н', 1), RELATION('О', 1), RELATION('П', 1), RELATION('Р', 1),

RELATION('С', 1), RELATION('Т', 1), RELATION('У', 1), RELATION('Ф', 1),

RELATION('Х', 1), RELATION('Ц', 1), RELATION('Ч', 1), RELATION('Ш', 1),

RELATION('Щ', 1), RELATION('Ъ', 1), RELATION('Ы', 1), RELATION('Ь', 1),

RELATION('Э', 1), RELATION('Ю', 1), RELATION('Я', 1),

RELATION('а', 1), RELATION('б', 1), RELATION('в', 1), RELATION('г', 1),

RELATION('д', 1), RELATION('е', 1), RELATION('ё', 1), RELATION('ж', 1),

RELATION('з', 1), RELATION('и', 1), RELATION('й', 1), RELATION('к', 1),

RELATION('л', 1), RELATION('м', 1), RELATION('н', 1), RELATION('о', 1),

RELATION('п', 1), RELATION('р', 1), RELATION('с', 1), RELATION('т', 1),

RELATION('у', 1), RELATION('ф', 1), RELATION('х', 1), RELATION('ц', 1),

RELATION('ч', 1), RELATION('ш', 1), RELATION('щ', 1), RELATION('ъ', 1),

RELATION('ы', 1), RELATION('ь', 1), RELATION('э', 1), RELATION('ю', 1),

RELATION('я', 1), RELATION(' ', 1), RELATION('.', 1), RELATION(',', 1),

RELATION('?', 1), RELATION('!', 1), RELATION(';', 1), RELATION(':', 1),

RELATION('-', 1), RELATION(')', 1), RELATION('(', 1), RELATION('A', 2),

RELATION('B', 2), RELATION('C', 2), RELATION('D', 2), RELATION('E', 2),

RELATION('F', 2), RELATION('G', 2), RELATION('H', 2), RELATION('I', 2),

RELATION('J', 2), RELATION('K', 2), RELATION('L', 2), RELATION('M', 2),

RELATION('N', 2), RELATION('O', 2), RELATION('P', 2), RELATION('Q', 2),

RELATION('R', 2), RELATION('S', 2), RELATION('T', 2), RELATION('U', 2),

RELATION('V', 2), RELATION('W', 2), RELATION('X', 2), RELATION('Y', 2),

RELATION('Z', 2), RELATION('a', 2), RELATION('b', 2), RELATION('c', 2),

RELATION('d', 2), RELATION('e', 2), RELATION('f', 2), RELATION('g', 2),

RELATION('h', 2), RELATION('i', 2), RELATION('j', 2), RELATION('k', 2),

RELATION('l', 2), RELATION('m', 2), RELATION('n', 2), RELATION('o', 2),

RELATION('p', 2), RELATION('q', 2), RELATION('r', 2), RELATION('s', 2),

RELATION('t', 2), RELATION('u', 2), RELATION('v', 2), RELATION('w', 2),

RELATION('x', 2), RELATION('y', 2), RELATION('z', 2), RELATION('1', 2),

RELATION('2', 2), RELATION('3', 2), RELATION('4', 2), RELATION('5', 2),

RELATION('6', 2), RELATION('7', 2), RELATION('8', 2), RELATION('9', 2),

RELATION('0', 2), RELATION('а', 2), RELATION('б', 2), RELATION('в', 2),

RELATION('г', 2), RELATION('д', 2), RELATION('е', 2), RELATION('ё', 2),

RELATION('ж', 2), RELATION('з', 2), RELATION('и', 2), RELATION('й', 2),

RELATION('к', 2), RELATION('л', 2), RELATION('м', 2), RELATION('н', 2),

RELATION('о', 2), RELATION('п', 2), RELATION('р', 2), RELATION('с', 2),

RELATION('т', 2), RELATION('у', 2), RELATION('ф', 2), RELATION('х', 2),

RELATION('ц', 2), RELATION('ч', 2), RELATION('ш', 2), RELATION('щ', 2),

RELATION('ъ', 2), RELATION('ы', 2), RELATION('ь', 2), RELATION('э', 2),

RELATION('ю', 2), RELATION('я', 2), RELATION('А', 2), RELATION('Б', 2),

RELATION('В', 2), RELATION('Г', 2), RELATION('Д', 2), RELATION('Е', 2),

RELATION('Ё', 2), RELATION('Ж', 2), RELATION('З', 2), RELATION('И', 2),

RELATION('Й', 2), RELATION('К', 2), RELATION('Л', 2), RELATION('М', 2),

RELATION('Н', 2), RELATION('О', 2), RELATION('П', 2), RELATION('Р', 2),

RELATION('С', 2), RELATION('Т', 2), RELATION('У', 2), RELATION('Ф', 2),

RELATION('Х', 2), RELATION('Ц', 2), RELATION('Ч', 2), RELATION('Ш', 2),

RELATION('Щ', 2), RELATION('Ъ', 2), RELATION('Ы', 2), RELATION('Ь', 2),

RELATION('Э', 2), RELATION('Ю', 2), RELATION('Я', 2),

RELATION(' ', 2), RELATION('.', 2), RELATION(',', 2), RELATION('?', 2),

RELATION('!', 2), RELATION(';', 2), RELATION(':', 2), RELATION('-', 2),

RELATION(')', 2), RELATION('(', 2)),

NODE(1, RELATION('`', 3)),

NODE()

);

FST fstDate(LEX\_LIB\_FUNC, STATIC\_LIB, "", 5,

NODE(1, RELATION('D', 1)),

NODE(1, RELATION('A', 2)),

NODE(1, RELATION('T', 3)),

NODE(1, RELATION('E', 4)),

NODE()

);

FST fstTime(LEX\_LIB\_FUNC, STATIC\_LIB, "", 5,

NODE(1, RELATION('T', 1)),

NODE(1, RELATION('I', 2)),

NODE(1, RELATION('M', 3)),

NODE(1, RELATION('E', 4)),

NODE()

);

FST fstShiftOp(LEX\_OPERATOR, OPERATION, "", 4,

NODE(2, RELATION('<', 1), RELATION('~', 2)),

NODE(1, RELATION('~', 3)),

NODE(1, RELATION('>', 3)),

NODE()

);

FST fstWhen(LEX\_WHEN, DEF\_LEX, "", 5,

NODE(1, RELATION('W', 1)),

NODE(1, RELATION('H', 2)),

NODE(1, RELATION('E', 3)),

NODE(1, RELATION('N', 4)),

NODE()

);

FST fstOtherwise(LEX\_OTHERWISE, DEF\_LEX, "", 10,

NODE(1, RELATION('O', 1)),

NODE(1, RELATION('T', 2)),

NODE(1, RELATION('H', 3)),

NODE(1, RELATION('E', 4)),

NODE(1, RELATION('R', 5)),

NODE(1, RELATION('W', 6)),

NODE(1, RELATION('I', 7)),

NODE(1, RELATION('S', 8)),

NODE(1, RELATION('E', 9)),

NODE()

);

FST fstEndw(LEX\_ENDW, DEF\_LEX, "", 5,

NODE(1, RELATION('E', 1)),

NODE(1, RELATION('N', 2)),

NODE(1, RELATION('D', 3)),

NODE(1, RELATION('W', 4)),

NODE()

);

# Приложение Б

Листинг 4 – Структура сообщений об ошибках языка CES-2020

struct ERROR

{

int id; //код ошибки

char message[ERROR\_MAXSIZE\_MESSAGE]; //сообщение об ошибке

struct IN //расширение для ошибок при обработке входных данных

{

short line; //номер строки

short col; //номер позиции

} inext;

};

Листинг 5 – Обновлённая таблица лексем

1: tfi(ti,ti)

2: {

3: nti.

4: i:iiv.

5: ri.

6: }

7: tfi(ti,ti)

8: {

9: nti.

10: nti.

11: i:iiz.

12: w(i)

13: i:i.

14: e

15: i:i.

16: ]

17: ri.

18: }

19: m

20: {

21: nti.

22: i:l.

23: oi.

24: nti.

25: i:l.

26: oi.

27: w(lzl)

28: i:@.

29: e

30: i:@.

31: ]

32: oi.

33: nti.

34: nti.

35: i:ll@ll@v.

36: i:llvlivv.

37: nti.

38: i:ii@.

39: oi.

40: }

# Приложение В

Листинг 6 – Структура автомата с магазинной памятью

struct MfstState // состояние автомата (для сохранения)

{

short lenta\_position; // позиция на ленте

short nrule; // номер текущего правила

short nrulechain; // номер текущей цепочки, текущего правил

MFSTSTSTACK st; // стек автомата

MfstState();

MfstState(

short pposition, // позиция на ленте

MFSTSTSTACK pst, // стек автомата

short pnrulechain // номер текущей цепочки, текущего правила

);

MfstState(

short pposition, // позиция на ленте

MFSTSTSTACK pst, // стек автомата

short pnrule,

short pnrulechain // номер текущей цепочки, текущего правила

);

};

struct Mfst // магазинный автомат

{

enum RC\_STEP { // код возврата функции step

NS\_OK,

NS\_NORULE,

NS\_NORULECHAIN,

NS\_ERROR,

TS\_OK, TS\_NOK,

LENTA\_END,

SURPRISE };

struct MfstDiagnosis // диагностика

{

short lenta\_position; // позиция на ленте

RC\_STEP rc\_step; // код завершения шага

short nrule; // номер правила

short nrule\_chain; // номер цепочки правила

MfstDiagnosis();

MfstDiagnosis(

short plenta\_position, // позиция на ленте

RC\_STEP prc\_step, // код завершения шага

short pnrule, // номер правила

short pnrule\_chain // номер цепочки правила

);

}diagnosis[MFST\_DIAGN\_NUMBER];//последние самые глубокие сообщения

GRBALPHABET\* lenta;

short lenta\_position; // текущая позиция на ленте

short nrule; // номер текущего правила

short nrulechain; // номер текущей цепочки, текущего правила

short lenta\_size; // размер ленты

GRB::Greibach greibach; // грамматика Грейбах

LT::LexTable lex; // результат работы лексич. анализатора

MFSTSTSTACK st; // стек автомата

std::vector<MfstState> storestate;// стек для сохранения состояний

Mfst();

Mfst(

LT::LexTable plex, // результат работы лексич. анализатора

GRB::Greibach pgreibach // грамматика Грйбах

);

char\* getCSt(char\* buf); // получить содержимое стека

char\* getCLenta(char\* buf, short pos, short n = 25);

char\* getDiagnosis(short n, char\* buf);

bool saveState(Log::LOG log);

bool restState(Log::LOG log);

bool push\_chain( // поместить цепочку правила в стеку

GRB::Rule::Chain chain // цепочка правила

);

RC\_STEP step(Log::LOG log); // выполинть шаг автомата

bool start(Log::LOG &log, Parm::PARM& parm);

bool saveDiagnosis(

RC\_STEP pprc\_step // код завершения шага

);

void printRules(Log::LOG& log);

struct Deducation // вывод

{

short size; // кол-во шагов в выводе

short\* nrules; // номера правил грамматики

short\* nrulechains; // номера цепочек правил грамматики

Deducation() { size = 0; nrules = 0; nrulechains = 0; };

}deducation;

bool saveDeducation(); // сохранить дерево вывода

};

Листинг 7 – Структура грамматики Грейбах

struct Rule // правило в грамматике Грейбах

{

GRBALPHABET nn; // нетерминал(левый символ правила) < 0

int iderror; // индетификатор диагностического сообщения

short size; // кол-во цепочек - правых частей правила

struct Chain // цепочка(правая часть правила)

{

short size; // длина цепочки

GRBALPHABET\* nt; // цепочка терминалов и нетерминалов

Chain() { size = 0; nt = 0; };

Chain(

short psize, // кол-во символов в цепочке

GRBALPHABET s, ... // символы(терминал и нетерминал)

);

char\* getCChain(char\* b); // получить правую сторону правила

static GRBALPHABET T(char t) { return GRBALPHABET(t); };

static GRBALPHABET N(char n) { return -GRBALPHABET(n); };

static bool isT(GRBALPHABET s) { return s > 0; };

static bool isN(GRBALPHABET s) { return !isT(s); };

static char alphabet\_to\_char(GRBALPHABET s) { return isT(s) ? char(s) : char(-s); };

}\* chains;

Rule() { nn = 0x00; size = 0; iderror = 0; chains = nullptr; }

Rule(

GRBALPHABET pnn, // нетерминал

int iderror, // идент. диагностического сообщения (Error)

short psize, // кол-во цепочек - правых частей правила

Chain c, ...// множество цепочек- правых частей правила

);

char\* getCRule(

char\* b, // буфер

short nchain // номер цепочки (правой части в правиле

);

short getNextChain(

GRBALPHABET t, // первый символ цепочки

Rule::Chain& pchain, // возвращаемая цепочка

short j // номер цепочки

);

};

struct Greibach // грамматика Грейбах

{

short size; // кол-во правил

GRBALPHABET startN; // стартовый символ

GRBALPHABET stbottomT; // дно стека

Rule\* rules; // множество правил

Greibach() { size = 0; startN = 0; stbottomT = 0; rules = 0; };

Greibach(

GRBALPHABET pstartN, // стартовый символ

GRBALPHABET pstBottomT, // дно стека

short psize, // кол-во правил

Rule r, ... // правила

);

short getRule(

GRBALPHABET pnn, // левый символ правила

Rule& prule

);

Rule getRule(short n); // получить правило по номеру

};

Greibach getGreibach(); //получить грамматику

# Приложение Г

Листинг 8 – Разбор исходного кода синтаксическим анализатором

Шаг : Правило Входная лента Стек

0 : S->tfi(F){NrO.}S tfi(ti,ti){nti.i:ivi.ri.} S$

0 : SAVESTATE: 1

0 : tfi(ti,ti){nti.i:ivi.ri.} tfi(F){NrO.}S$

1 : fi(ti,ti){nti.i:ivi.ri.}t fi(F){NrO.}S$

2 : i(ti,ti){nti.i:ivi.ri.}tf i(F){NrO.}S$

3 : (ti,ti){nti.i:ivi.ri.}tfi (F){NrO.}S$

4 : ti,ti){nti.i:ivi.ri.}tfi( F){NrO.}S$

5 : F->ti ti,ti){nti.i:ivi.ri.}tfi( F){NrO.}S$

5 : SAVESTATE: 2

5 : ti,ti){nti.i:ivi.ri.}tfi( ti){NrO.}S$

6 : i,ti){nti.i:ivi.ri.}tfi(t i){NrO.}S$

7 : ,ti){nti.i:ivi.ri.}tfi(ti ){NrO.}S$

8 : TS\_NOK/NS\_NORULECHAIN

8 : RESSTATE

8 : ti,ti){nti.i:ivi.ri.}tfi( F){NrO.}S$

9 : F->ti,F ti,ti){nti.i:ivi.ri.}tfi( F){NrO.}S$

9 : SAVESTATE: 2

Листинг 8 (прод.) – Разбор исходного кода синтаксическим анализатором

940 : SAVESTATE: 67

940 : i,i).oi.} i,W).N}$

941 : ,i).oi.} ,W).N}$

942 : i).oi.} W).N}$

943 : W->i i).oi.} W).N}$

943 : SAVESTATE: 68

943 : i).oi.} i).N}$

944 : ).oi.} ).N}$

945 : .oi.} .N}$

946 : oi.} N}$

947 : N->oO. oi.} N}$

947 : SAVESTATE: 69

947 : oi.} oO.}$

948 : i.} O.}$

949 : O->i i.} O.}$

949 : SAVESTATE: 70

949 : i.} i.}$

950 : .} .}$

951 : } }$

952 : $

953 : LENTA\_END

954 : ------>LENTA\_END

Листинг 9 – Дерево разбора

0 : S->tfi(F){NrO.}S

4 : F->ti,F

7 : F->ti

11 : N->nti.N

15 : N->i:E.

17 : E->iM

18 : M->vE

19 : E->i

22 : O->i

25 : S->tfi(F){NrO.}S

29 : F->ti,F

32 : F->ti

36 : N->nti.N

40 : N->nti.N

44 : N->i:E.N

46 : E->iM

47 : M->zO

48 : O->i

50 : N->w(B)C]

52 : B->i

54 : C->i:E.ei:E.

56 : E->i

61 : E->i

65 : O->i

68 : S->m{N}

70 : N->nti.N

74 : N->i:E.N

76 : E->l

78 : N->oO.N

79 : O->i

81 : N->nti.N

85 : N->i:E.N

87 : E->l

89 : N->oO.N

90 : O->i

92 : N->w(B)C]N

94 : B->lzO

96 : O->l

98 : C->i:E.ei:E.

100 : E->x()

107 : E->x()

112 : N->oO.N

113 : O->i

115 : N->nti.N

119 : N->nti.N

123 : N->i:E.N

125 : E->i(W)M

127 : W->l,W

129 : W->l

131 : M->vE

132 : E->i(W)

134 : W->l,W

136 : W->l

139 : N->i:E.N

141 : E->(E)M

142 : E->lM

143 : M->vE

144 : E->l

146 : M->vE

147 : E->(E)

148 : E->lM

149 : M->vE

150 : E->i

153 : N->nti.N

157 : N->i:E.N

159 : E->i(W)

161 : W->i,W

163 : W->i

166 : N->oO.

167 : O->i

# Приложение Д

Листинг 10 – Программная реализация механизма преобразования в ПОЛИЗ

void AddToResult(IT::IdTable& idTable, LT::Entry\* result, LT::Entry

elementLT, int& pos, int& flag, int lenout, int lextablePos)

{

if (TI\_EL(elementLT.idxTI).idxfirstLE >= lextablePos && TI\_EL(

elementLT.idxTI).idxfirstLE < (lextablePos + lenout))

TI\_EL(elementLT.idxTI).idxfirstLE = lextablePos + flag;

result[pos] = elementLT;

flag++;

pos++;

}

void AddToResult(std::stack<LT::Entry>& stack, IT::IdTable& idTable, LT::Entry\* result, int& pos, int& flag, int lenout, int lextablePos)

{

if (stack.top().idxTI != LT\_TI\_NULLIDX)

{

if (TI\_EL(stack.top().idxTI).idxfirstLE >= lextablePos && TI\_EL(stack.top().idxTI).idxfirstLE < (lextablePos + lenout))

TI\_EL(stack.top().idxTI).idxfirstLE = lextablePos + flag;

}

result[pos] = stack.top();

flag++;

pos++;

stack.pop();

}

bool PolishNotation(int lextablePos, LT::LexTable & lexTable, IT::IdTable & idTable)

{

std::stack<LT::Entry> stackLTelements;

LT::Entry\* elementsLT = new LT::Entry[lexTable.size];

LT::Entry tempLTEntry, bufEntry;

int ncomma, flag, waste, funcPositionTI, lenght, lenout, dotId;

bool wasLogOper, wasMathOper;

wasMathOper = wasLogOper = false;

ncomma = flag = waste = lenght = lenout = 0;

dotId = funcPositionTI = -1;

for (int i = lextablePos; LT\_EL(i).lexema != LEX\_DOT; i++)

{

lenout = i + 1;

dotId = i + 1;

}

for (int i = lextablePos; i < lenout; i++)

{

tempLTEntry = LT\_EL(i);

if (tempLTEntry.lexema == LEX\_ID || tempLTEntry.lexema == LEX\_LITERAL || tempLTEntry.lexema == LEX\_LIB\_FUNC)

{

if (TI\_EL(tempLTEntry.idxTI).idtype == IT::F)

{

funcPositionTI = tempLTEntry.idxTI;

continue;

}

AddToResult(idTable, elementsLT, tempLTEntry, lenght, flag, lenout, lextablePos);

}

else

{

if (tempLTEntry.lexema == LEX\_OPERATOR || tempLTEntry.lexema == LEX\_LOG\_OPERATOR)

{

if (wasLogOper && tempLTEntry.lexema == LEX\_OPERATOR

|| wasMathOper && tempLTEntry.lexema == LEX\_LOG\_OPERATOR)

throw ERROR\_THROW\_IN(312, LT\_EL(lextablePos).sn, LT\_EL(lextablePos).pn);

// Проверка на корректность выражения

if (tempLTEntry.lexema == LEX\_OPERATOR) wasMathOper = true;

else if (tempLTEntry.lexema == LEX\_LOG\_OPERATOR) wasLogOper = true;

while (!stackLTelements.empty() &&

stackLTelements.top().priority >= tempLTEntry.priority)

{

if (stackLTelements.top().lexema != LEX\_LEFTHESIS)

AddToResult(stackLTelements, idTable, elementsLT, lenght, flag, lenout, lextablePos);

else

break;

}

stackLTelements.push(tempLTEntry);

}

if (tempLTEntry.lexema == LEX\_COMMA)

{

ncomma++;

while (stackLTelements.top().lexema == LEX\_OPERATOR)

{

AddToResult(stackLTelements, idTable, elementsLT, lenght, flag, lenout, lextablePos);

}

}

else if (tempLTEntry.lexema != LEX\_RIGHTHESIS)

{

if (stackLTelements.empty() || stackLTelements.top().lexema == LEX\_LEFTHESIS || tempLTEntry.lexema == LEX\_LEFTHESIS)

stackLTelements.push(tempLTEntry);

}

if (tempLTEntry.lexema == LEX\_RIGHTHESIS && tempLTEntry.priority != 4)

{

waste += 2;

while (stackLTelements.top().lexema != LEX\_LEFTHESIS)

{

AddToResult(stackLTelements, idTable, elementsLT, lenght, flag, lenout, lextablePos);

}

stackLTelements.pop();

}

if (tempLTEntry.lexema == LEX\_RIGHTHESIS && tempLTEntry.priority == 4)

{

bufEntry.lexema = LEX\_SUBST;

bufEntry.idxTI = funcPositionTI;

bufEntry.sn = tempLTEntry.sn;

bufEntry.pn = tempLTEntry.pn;

// количество параметров в функции

bufEntry.priority = ncomma + 1;

elementsLT[lenght] = bufEntry;

lenght++;

if (ncomma != 0)

{

waste += ncomma;

ncomma = 0;

}

while (stackLTelements.top().lexema != LEX\_LEFTHESIS)

{

AddToResult(stackLTelements, idTable, elementsLT, lenght, flag, lenout, lextablePos);

}

stackLTelements.pop();

waste += 2;

}

}

}

while (!stackLTelements.empty())

{

AddToResult(stackLTelements, idTable, elementsLT, lenght, flag, lenout, lextablePos);

}

for (int i = lextablePos, k = 0; i < lextablePos + lenght; i++, k++)

{

// Запись в ТЛ

LT\_EL(i) = elementsLT[k];

}

// Вставка элемента с точкой

LT\_EL(lextablePos + lenght) = LT\_EL(dotId);

for (int i = 0; i < waste; i++)

{

lexTable.size--;

// сдвиг таблицы лексем, т.к после преобразования она может быть меньше

for (int j = lextablePos + lenght + 1; j < lexTable.size; j++)

{

LT\_EL(j) = LT\_EL(j + 1);

if (LT\_EL(j).idxTI != LT\_TI\_NULLIDX && TI\_EL(LT\_EL(j).idxTI).idxfirstLE == (j + 1))

TI\_EL(LT\_EL(j).idxTI).idxfirstLE--;

}

}

return true;

}

void FindExpressions(LT::LexTable& lexTable, IT::IdTable& idTable)

{

for (int i = 0; i < lexTable.size; i++)

{

if (LT\_EL(i).lexema == LEX\_ASSIGN)

PolishNotation(++i, lexTable, idTable);

}

}

# Приложение Е

Листинг 11 – Контрольный пример языка CES-2020, сгенерированный в язык ассемблера

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

ExitProcess PROTO : DWORD

EXTRN outUs : proc

EXTRN outStr : proc

EXTRN outBool : proc

EXTRN outSymb : proc

EXTRN TIME : proc

EXTRN DATE : proc

.stack 4096

.const

L0 DWORD 'a'

L1 BYTE 'CES-2020', 0

L2 DWORD 6

L3 DWORD 2

L4 DWORD 9

L5 DWORD 5

L6 DWORD 11

L7 DWORD 15

L8 DWORD 3

.data

mulans3 DWORD ?

maxa7 DWORD ?

maxans8 DWORD ?

HEADc9 DWORD ?

HEADs11 DWORD ?

HEADq17 DWORD ?

HEADw18 DWORD ?

HEADe24 DWORD ?

.code

mul0 PROC mulp11 : SDWORD, mulp22 : SDWORD

push mulp11

push mulp22

pop ebx

pop eax

mul ebx

cmp eax, 256

jl OK0

push 7

push 4

push eax

call outUs

push 0

call ExitProcess

OK0:

push eax

pop mulans3

mov eax, mulans3

ret

mul0 ENDP

max4 PROC maxp15 : SDWORD, maxp26 : SDWORD

push maxp15

push maxp26

pop ebx

pop eax

cmp eax, ebx

jle M1

push 1

jmp M2

M1:

push 0

M2:

pop maxa7

mov eax, maxa7

cmp eax, 1

jne W0

push maxp15

pop maxans8

jmp W1

W0:

push maxp26

pop maxans8

W1:

mov eax, maxans8

ret

max4 ENDP

main PROC

push L0

pop HEADc9

push HEADc9

call outSymb

push offset L1

pop HEADs11

push HEADs11

call outStr

mov eax, L2

cmp eax, L3

je W2

pop edx

push offset L1

call DATE

push eax

pop HEADs11

jmp W3

W2:

pop edx

push L0

call TIME

push eax

pop HEADs11

W3:

push HEADs11

call outStr

push L4

push L5

pop edx

pop edx

push L5

push L4

call max4

push eax

push L6

push L7

pop edx

pop edx

push L7

push L6

call max4

push eax

pop ebx

pop eax

add eax, ebx

cmp eax, 256

jl OK1

push 15

push 35

push eax

call outUs

jmp ERROR

OK1:

push eax

pop HEADq17

push L8

push L3

pop ebx

pop eax

cmp eax, ebx

jge M9

push 1

jmp M10

M9:

push 0

M10:

push L3

push HEADq17

pop ebx

pop eax

add eax, ebx

cmp eax, 256

jl OK2

push 17

push 36

push eax

call outUs

jmp ERROR

OK2:

push eax

pop ebx

pop eax

cdq

cmp ebx, 0

jne OK3

push 12

push 36

push -2147483648

call outUs

jmp ERROR

OK3:

div ebx

push eax

pop HEADw18

push HEADq17

push HEADw18

pop edx

pop edx

push HEADw18

push HEADq17

call mul0

push eax

pop HEADe24

push HEADe24

call outUs

ERROR:

push 0

call ExitProcess

main ENDP

end main