Министерство образования и науки Российской Федерации

Северо-Кавказский горно-металлургический институт (государственный технологический университет)

Кафедра информатики и вычислительной техники

**ОТЧЕТ**

**по лабораторной работе № 8**

**по дисциплине:**

**«лингвистическое и программное обеспечение САПР»**

**Выполнил ст.гр. ИВб-22-2**:

Степанян Константин Александрович

**Проверил**:

Преподаватель: Фомин Андрей Николаевич

Владикавказ 2025

ВАРИАНТ 1

Задание 2: Для выполнения лабораторной работы требуется написать программу, которая на основании дерева синтаксического разбора порождает объектный код и выполняет затем его оптимизацию методом свертки объектного кода и методом исключения лишних операций. В качестве исходного дерева синтаксического разбора рекомендуется взять дерево, которое порождает программа, построенная по заданию лабораторной работы №6.

Программу рекомендуется построить из трех основных частей:

первая часть — порождение дерева синтаксического разбора,

вторая часть — реализация алгоритма порождения объектного кода по дереву разбора,

третья часть — оптимизация порожденного объектного кода (если в результирующей программе присутствуют линейные участки кода).

Результатом работы должна быть построенная на основе заданного предложения грамматики программа на объектном языке или построенная последовательность триад (по согласованию с преподавателем выбирается форма представления конечного результата).

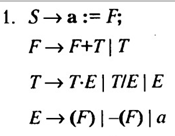
Все встречающиеся в исходной программе идентификаторы считать простыми скалярными переменными, не требующими выполнения преобразования типов.

Задание 6-й лабораторной работы:

«Разработать программу синтаксического анализатора исходного текста согласно своему варианту задания. Программа получает на входе файл - результат лексического анализа и выполняет синтаксический анализ исходной программы. Результатом работы должно быть сообщение о корректности программы или сообщение об первой обнаруженной ошибке с указанием строки и конструкции языка, при разборе которых обнаружена ошибка.

Далее приведены варианты грамматик. Во всех вариантах символ S является начальным символом грамматики; S, F, Т и Е обозначают нетерминальные символы. Терминальные символы выделены жирным шрифтом. Вместо символа а должны подставляться лексемы.

Вариант 1 (6-й лабораторной):

»

Код:

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**IO**;**

class Program

**{**

static List**<**string**>** tokens**;** // список лексем

static int pos **=** 0**;** // текущая позиция

// Для генерации и оптимизации кода

static List**<**string**>** objectCode **=** **new** List**<**string**>();**

static void Main**()**

**{**

**try**

**{**

tokens **=** **new** List**<**string**>(**File**.**ReadAllLines**(**"вход.txt"**));**

Console**.**WriteLine**(**"Начат синтаксический анализ..."**);**

Console**.**WriteLine**(**"Лексемы: " **+** string**.**Join**(**" "**,** tokens**.**ToArray**())** **+** "\n"**);**

S**();** // стартовый символ грамматики

**if** **(**pos **<** tokens**.**Count**)**

**throw** **new** Exception**(**"Лишние лексемы после конца программы"**);**

Console**.**WriteLine**(**"\nПрограмма корректна"**);**

Console**.**WriteLine**(**"\n[Генерация объектного кода]"**);**

foreach **(**var line in objectCode**)**

Console**.**WriteLine**(**line**);**

Console**.**WriteLine**(**"\n[Оптимизация кода]"**);**

var optimized **=** OptimizeCode**(**objectCode**);**

foreach **(**var line in optimized**)**

Console**.**WriteLine**(**line**);**

**}**

**catch** **(**Exception ex**)**

**{**

Console**.**WriteLine**(**"\nОШИБКА: " **+** ex**.**Message**);**

**}**

Console**.**WriteLine**(**"\nНажмите любую клавишу для выхода..."**);**

Console**.**ReadKey**();**

**}**

// S → a := F ;

static void S**()**

**{**

Match**(**"a"**);**

Match**(**":="**);**

string result **=** F**();**

Match**(**";"**);**

objectCode**.**Add**(**$"STORE {result}"**);** // Сохраняем результат в переменную a

**}**

// F → F + T | T

static string F**()**

**{**

string left **=** T**();**

**while** **(**MatchOptional**(**"+"**))**

**{**

string right **=** T**();**

string temp **=** NewTemp**();**

objectCode**.**Add**(**$"ADD {left} {right} -> {temp}"**);**

left **=** temp**;**

**}**

**return** left**;**

**}**

// T → T : E | E

static string T**()**

**{**

string left **=** E**();**

**while** **(**MatchOptional**(**":"**))**

**{**

string right **=** E**();**

string temp **=** NewTemp**();**

objectCode**.**Add**(**$"CONCAT {left} {right} -> {temp}"**);** // Используем CONCAT вместо :

left **=** temp**;**

**}**

**return** left**;**

**}**

// E → ( F ) | - ( F ) | a

static string E**()**

**{**

**if** **(**MatchOptional**(**"("**))**

**{**

string val **=** F**();**

Match**(**")"**);**

**return** val**;**

**}**

**else** **if** **(**MatchOptional**(**"-"**))**

**{**

Match**(**"("**);**

string val **=** F**();**

Match**(**")"**);**

string temp **=** NewTemp**();**

objectCode**.**Add**(**$"NEG {val} -> {temp}"**);**

**return** temp**;**

**}**

**else**

**{**

Match**(**"a"**);**

**return** "a"**;**

**}**

**}**

// --- Генерация временных переменных ---

static int tempCount **=** 0**;**

static string NewTemp**()**

**{**

tempCount**++;**

**return** $"t{tempCount}"**;**

**}**

// --- Оптимизация кода ---

static List**<**string**>** OptimizeCode**(**List**<**string**>** code**)**

**{**

List**<**string**>** optimized **=** **new** List**<**string**>();**

foreach **(**string line in code**)**

**{**

**if** **(**line**.**StartsWith**(**"ADD"**))**

**{**

var parts **=** line**.**Split**(new[]** **{** ' ' **},** StringSplitOptions**.**RemoveEmptyEntries**);**

**if** **(**int**.**TryParse**(**parts**[**1**],** out int x**)** **&&** int**.**TryParse**(**parts**[**2**],** out int y**))**

**{**

int sum **=** x **+** y**;**

optimized**.**Add**(**$"LOAD {sum} -> {parts[4]}"**);**

**continue;**

**}**

**}**

**else** **if** **(**line**.**StartsWith**(**"NEG"**))**

**{**

var parts **=** line**.**Split**(new[]** **{** ' ' **},** StringSplitOptions**.**RemoveEmptyEntries**);**

**if** **(**int**.**TryParse**(**parts**[**1**],** out int val**))**

**{**

int neg **=** **-**val**;**

optimized**.**Add**(**$"LOAD {neg} -> {parts[3]}"**);**

**continue;**

**}**

**}**

optimized**.**Add**(**line**);**

**}**

**return** optimized**;**

**}**

// --- Сопоставление обязательной и необязательной лексемы ---

static void Match**(**string expected**)**

**{**

**if** **(**pos **>=** tokens**.**Count**)**

**throw** **new** Exception**(**"Ожидалась '" **+** expected **+** "', но достигнут конец ввода"**);**

**if** **(**tokens**[**pos**]** **!=** expected**)**

**throw** **new** Exception**(**"Ожидалась '" **+** expected **+** "', но получено '" **+** tokens**[**pos**]** **+** "' на позиции " **+** **(**pos **+** 1**));**

pos**++;**

**}**

static bool MatchOptional**(**string token**)** **{**

**if** **(**pos **<** tokens**.**Count **&&** tokens**[**pos**]** **==** token**)** **{**

pos**++;**

**return** **true;** **}**

**return** **false;** **} }**

Результат:

