Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧЕТ**

по лабораторной работе №6

по дисциплине «Трансляторы и компиляторы»

## на тему «Разработка транслятора для построения дерева вывода»

Выполнили студенты группы 22ВВП1

Демин М.С.

Беляев Д. И.

Приняли:

Дубинин В.Н.

Карамышева Н.С.

Пенза 2024

**Название**

Разработка транслятора для построения дерева вывода

**Цель работы**

Разработать транслятора для построения дерева вывода

**Задание**

На основе предыдущей лаб. работы «Разработка синтаксического анализатора методом рекурсивного спуска» разработать программу-транслятор предложений некоторого языка в дерево вывода. Язык описывается той же самой грамматикой, что и в предыдущей работе. Дерево вывода представляется в виде списка дуг. Транслятор должен записывать дерево вывода в текстовый файл в определенном формате.

**Листинг**

**main.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_RULES 100

int Rules[MAX\_RULES];

int currentNode = 0; // Счетчик текущей вершины

typedef struct {

int sourceNode;

char sourceType;

int targetNode;

char targetType;

} Edge;

Edge edges[MAX\_RULES];

int edgeCount = 0;

char\* SymbolS(char\* szInput, FILE\* output);

char\* SymbolA(char\* szInput, FILE\* output);

char\* SymbolB(char\* szInput, FILE\* output);

char\* SymbolC(char\* szInput, FILE\* output);

void AddRuleToOutput(int nRule, FILE\* output, char sourceType, char targetType) {

static int i = 0;

if (i == MAX\_RULES) {

printf("Input chain is too long\n");

exit(0);

}

Rules[i++] = nRule;

// Запись правила в массив дуг

edges[edgeCount].sourceNode = currentNode;

edges[edgeCount].sourceType = sourceType;

edges[edgeCount].targetNode = currentNode + 1;

edges[edgeCount].targetType = targetType;

edgeCount++;

currentNode++;

}

int main() {

char szInput[256];

char\* szOutput;

int i = 0;

memset(Rules, 0, MAX\_RULES \* sizeof(int));

memset(edges, 0, MAX\_RULES \* sizeof(Edge));

printf("Enter chain> ");

scanf("%s", szInput);

FILE\* output = fopen("output\_tree.txt", "w");

if (!output) {

printf("Error opening file for writing.\n");

return 1;

}

szOutput = SymbolS(szInput, output);

if (!szOutput) {

printf("Parse error\n");

fclose(output);

return 1;

}

if (\*szOutput) {

printf("Unexpected symbol %c\n", \*szOutput);

fclose(output);

return 2;

}

printf("Output tree for chain: ");

while (Rules[i]) {

printf("%d ", Rules[i++]);

}

printf("\n");

// Запись дерева в файл

for (i = 0; i < edgeCount; i++) {

fprintf(output, "%d:%c->%d:%c\n", edges[i].sourceNode, edges[i].sourceType, edges[i].targetNode, edges[i].targetType);

}

fclose(output);

return 0;

}

char\* SymbolS(char\* szInput, FILE\* output) {

// S -> aB | dC | b

switch (\*szInput) {

case '\0':

return szInput;

case 'a':

AddRuleToOutput(1, output, 'S', 'B'); // S -> aB

return SymbolB(szInput + 1, output);

case 'd':

AddRuleToOutput(2, output, 'S', 'C'); // S -> dC

return SymbolC(szInput + 1, output);

case 'b':

AddRuleToOutput(3, output, 'S', 'b'); // S -> b

return szInput + 1;

default:

return NULL;

}

}

char\* SymbolA(char\* szInput, FILE\* output) {

// A -> aC | bB | dC

switch (\*szInput) {

case '\0':

return szInput;

case 'a':

AddRuleToOutput(4, output, 'A', 'C'); // A -> aC

return SymbolC(szInput + 1, output);

case 'b':

AddRuleToOutput(5, output, 'A', 'B'); // A -> bB

return SymbolB(szInput + 1, output);

case 'd':

AddRuleToOutput(6, output, 'A', 'C'); // A -> dC

return SymbolC(szInput + 1, output);

default:

return NULL;

}

}

char\* SymbolB(char\* szInput, FILE\* output) {

// B -> aC | dC | a

switch (\*szInput) {

case '\0':

return szInput;

case 'a':

if (\*(szInput + 1) == 'a' || \*(szInput + 1) == 'b' || \*(szInput + 1) == 'd') {

AddRuleToOutput(7, output, 'B', 'C'); // B -> aC

return SymbolC(szInput + 1, output);

}

else {

AddRuleToOutput(9, output, 'B', 'a'); // B -> a (final)

return szInput + 1;

}

case 'd':

AddRuleToOutput(8, output, 'B', 'C'); // B -> dC

return SymbolC(szInput + 1, output);

default:

return NULL;

}

}

char\* SymbolC(char\* szInput, FILE\* output) {

// C -> b | aA | d

switch (\*szInput) {

case '\0':

return szInput;

case 'b':

AddRuleToOutput(10, output, 'C', 'b'); // C -> b

return szInput + 1;

case 'a':

AddRuleToOutput(11, output, 'C', 'A'); // C -> aA

return SymbolA(szInput + 1, output);

case 'd':

AddRuleToOutput(12, output, 'C', 'd'); // C -> d

return szInput + 1;

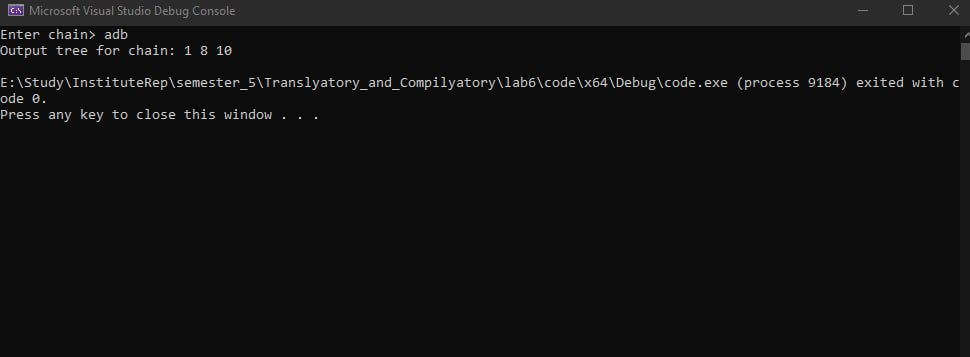
default:

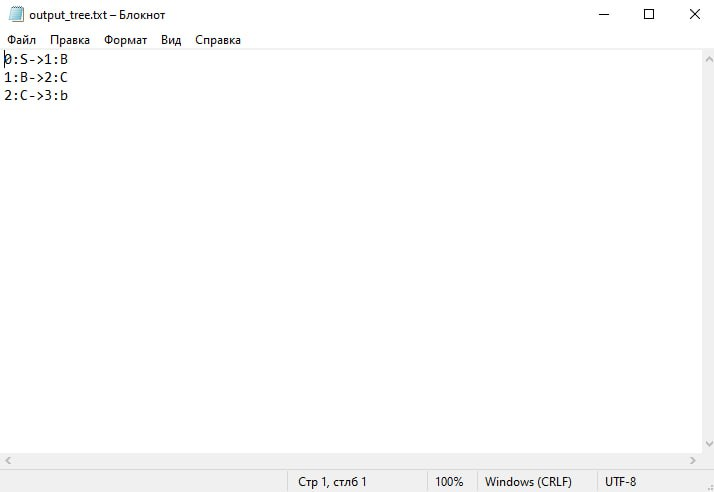
return NULL;

}

}

**Результат работы**





**Вывод**

Разработали транслятор для построения дерева вывода