**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ**

**“ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ ЕВФРОСИНИИ ПОЛОЦКОЙ”**

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по Лабораторной работе №4**

**«Генерация и оптимизация объектного кода»**

Выполнил студент 2 курса, группа 21-ИТ-1 Макеёнок Д.И.

Проверил Сыцевич Д.Н.

Полоцк, 2022 г.

**Цель работы:** изучение основных принципов генерации компилятором

объектного кода, ознакомление с методами оптимизации результирующего

объектного кода для линейного участка программы с помощью свертки и

исключения лишних операций.

**Результат работы:**

#include<iostream>

#include<string>

#include<list>

using namespace std;

enum tok\_names { ident, num, math };

string names[3];

struct token

{

enum tok\_names token\_name;

string token\_names[3] = { "Ident", "Number", "Math" };

string token\_value;

};

list<token> lexeme\_table;

token add\_token(tok\_names a, string b) {

token tok;

tok.token\_name = a;

tok.token\_value = b;

return tok;

}

list<token> lexer(string str)

{

list<token> lexeme\_table;

int i = 0; int value = 0;

while (i < str.size())

{

if (str[i] == '+') { lexeme\_table.push\_back(add\_token(math, "+")); }

if (str[i] == '-') { lexeme\_table.push\_back(add\_token(math, "-")); }

if (str[i] == '\*') { lexeme\_table.push\_back(add\_token(math, "\*")); }

if (str[i] == '/') { lexeme\_table.push\_back(add\_token(math, "/")); }

if ((str[i] >= 'a' and str[i] <= 'z') or (str[i] >= 'A' and str[i] <= 'Z')) {

string var = "";

while ((str[i] >= 'a' and str[i] <= 'z') or (str[i] >= 'A' and str[i] <= 'Z') or (str[i] >= '0' and str[i] <= '9'))

{

var += str[i];

i++;

}

i--;

lexeme\_table.push\_back(add\_token(ident, var));

}

if ((str[i] >= '0' and str[i] <= '9')) {

string number = "";

while (str[i] >= '0' and str[i] <= '9')

{

number += str[i];

i++;

}

i--;

lexeme\_table.push\_back(add\_token(num, number));

}

i++;

}

return lexeme\_table;

}

////////tree

struct node

{

token token; //Информационное поле

node\* l, \* r; //Левая и Правая часть дерева

};

node\* tree = NULL; //Объявляем переменную, тип которой структура Дерево

void pushTree(token token, node\*\* t) //Add

{

if ((\*t) == NULL) //Если дерева не существует

{

(\*t) = new node; //Выделяем память

(\*t)->token = token; //Кладем в выделенное место аргумент a

(\*t)->l = (\*t)->r = NULL; //Очищаем память для следующего роста

return; //Заложили семечко, выходим

}

if (token.token\_name != num and token.token\_name != ident) {

pushTree(token, &(\*t)->r);

return;

}

if ((token.token\_name == num or token.token\_name == ident) and (\*t)->l == NULL) {

pushTree(token, &(\*t)->l);

return;

}

else {

pushTree(token, &(\*t)->r);

return;

}

}

void printTriad(node\*\* t, int u)

{

if ((\*t)->r->token.token\_name == math)

{

printTriad(&(\*t)->r, ++u);

cout << u << ". ";

cout << (\*t)->token.token\_value;

cout << '(';

if ((\*t)->l->token.token\_name != math) cout << u + 1 << '^' << ',' << (\*t)->l->token.token\_value;

else cout << u + 1 << "^," << u << '^';

cout << ')' << endl;

if ((\*t)->l->token.token\_name == math)

{

cout << u << ". ";

cout << (\*t)->l->token.token\_value;

cout << '(';

cout << (\*t)->l->r->token.token\_value << ',' << (\*t)->l->l->token.token\_value;

cout << ')' << endl;

}

}

else

{

u++;

cout << u << ". ";

cout << (\*t)->token.token\_value;

cout << '(';

cout << (\*t)->r->token.token\_value << ',';

cout << (\*t)->l->token.token\_value;

cout << ')' << endl;

return;

}

}

void printTree(node\* t, int u, bool Direction) //Input

{

if (t == NULL) return; //Если дерево пустое, то отображать нечего, выходим

else

{

printTree(t->l, ++u, 1); //С помощью рекурсивного посещаем левое поддерево

cout << u;

cout << ' ';

if (u != 1) Direction == 1 ? cout << "l " : cout << "r ";

cout << "\t" << t->token.token\_value << endl;

u--;

}

printTree(t->r, ++u, 0); //С помощью рекурсии посещаем правое поддерево

}

int main()

{

token tok;

string str = "24 + 5 - A \* D + B + C + 1";

list<token> lexeme\_table = lexer(str);

list<token> temp = lexeme\_table;

while (temp.empty() == 0) {

tok = temp.front();

cout << "name:" << tok.token\_names[tok.token\_name] << ", " << "value:" << tok.token\_value << endl;

temp.pop\_front();

}

cout << endl;

list<token> znak;

list<token> identif;

temp = lexeme\_table;

while (temp.size() > 0)

{

if (temp.front().token\_name == num or temp.front().token\_name == ident) {

identif.push\_back(temp.front());

temp.pop\_front();

}

else {

znak.push\_back(temp.front());

temp.pop\_front();

}

}

node\* sosna = NULL;

while (znak.size() > 0 or identif.size() > 0)

{

if (znak.size() > 0)

{

pushTree(znak.front(), &sosna);

znak.pop\_front();

}

pushTree(identif.front(), &sosna);

identif.pop\_front();

}

printTree(sosna, 0, 1);

cout << endl;

printTriad(&sosna, 0);

}



Рисунок 1 – Результат работы программы

**Вывод:** В результате выполнения написанного программного кода было построено дерево и оптимизирован результирующий объектный код для линейного участка программы с помощью свертки и исключения лишних операций.