Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы № 2

по дисциплине «Методы разработки трансляторов»

Тема: «Перевод исходной программы в

обратную польскую запись»

Выполнил: ст. гр. 39/1

Козлов Э.Д.

Проверил: Преподаватель

Усов П.Е

Краснодар

2025

Ход работы:

Постановка задачи: Требуется разработать программу для перевода текста программы в ОПЗ.

**Основные понятия**

**Обратная польская запись (ОПЗ)** – способ записи выражений, при котором операнды записываются перед операторами. Например, стандартное выражение

a + b \* (a - b)

преобразуется в ОПЗ как

a b a b - \* +.

**Преимущества ОПЗ** заключаются в том, что порядок вычислений определяется структурой записи, а скобки становятся избыточными. Это упрощает алгоритмы вычисления и генерацию машинного кода.

**Алгоритм Дейкстры**

Преобразование инфиксного выражения в ОПЗ основано на алгоритме Дейкстры, который реализует так называемую «сортировочную станцию». Основные этапы алгоритма:

1. **Обработка операндов:** При встрече операндов (идентификаторов, чисел, констант) они немедленно добавляются в выходную строку.

2. **Обработка операторов:**

• Если стек пуст или оператор, находящийся на вершине, имеет меньший приоритет, текущий оператор помещается в стек.

• Если приоритет оператора на вершине стека выше или равен приоритету текущего оператора, производится извлечение операторов из стека в выходную строку до тех пор, пока это условие выполняется, после чего текущий оператор заносится в стек.

3. **Обработка скобок:**

• При встрече открывающей скобки она всегда помещается в стек.

• При встрече закрывающей скобки из стека извлекаются операторы до первой открывающей скобки, которая удаляется. Скобки при этом не попадают в итоговую запись.

4. **Выталкивание оставшихся операторов:** После завершения прохода по выражению оставшиеся операторы извлекаются из стека и добавляются к выходной записи.

В предыдущей лабораторной работе была разработана программа, которая выполняет лексический анализ исходного кода и формирует файл в формате .txt, содержащий лексемы (каждая лексема представлена парой «тип-значение»). На основе этого файла была создана программа для перевода инфиксных выражений в обратную польскую запись (ОПЗ) с использованием алгоритма Дейкстры. Программа читает файл с лексемами, обрабатывает операторы, операнды и скобки с учетом приоритетов, а затем записывает результат – лексемы, представленные в виде ОПЗ – в новый файл .txt.

Приведем тесты программы.

Исходный код на ЯП С:

if ((a - b) > 8) {  
 while ((a + b) < 20) {  
 x = arr[i] + 3;  
 }  
 for (i = 0; i < 10; i = i + 1) {  
 y = b[i] \* 2;  
 }  
}

Ожидаемый результат:

a b - 8 > УПЛ

a b + 20 < УЦ

x arr i 3 + =

i 0 =

i 10 < УЦ

y b i 2 \* =

i i 1 + =

УЦ

Фактический результат:



Вывод: В ходе работы была разработана программа перевода в ОПЗ. Разработанная программа успешно преобразует файл с лексемами в обратную польскую запись, корректно обрабатывая операторы и операнды.

Листинг программы

import re  
  
*# Приоритет операторов*OPERATOR\_PRIORITY = {  
 "=": 1,  
 "||": 2, "&&": 3,  
 "<": 4, "<=": 4, ">": 4, ">=": 4, "==": 4, "!=": 4,  
 "+": 5, "-": 5,  
 "\*": 6, "/": 6, "%": 6,  
 "(": 0, ")": 0, "[": -1, "]": -1 *# Учитываем скобки массивов*}  
  
*# Алгоритм Дейкстры*def to\_opz(expression\_tokens):  
 output = []  
 stack = []  
  
 for token in expression\_tokens:  
 if token.isalnum():  
 output.append(token)  
 elif token in ["(", "["]:  
 stack.append(token)  
 elif token in [")", "]"]:  
 while stack and stack[-1] not in ["(", "["]:  
 output.append(stack.pop())  
 if stack and stack[-1] in ["(", "["]:  
 stack.pop()  
 elif token in OPERATOR\_PRIORITY:  
 while stack and stack[-1] not in ["(", "["] and OPERATOR\_PRIORITY[stack[-1]] >= OPERATOR\_PRIORITY[token]:  
 output.append(stack.pop())  
 stack.append(token)  
  
 while stack: *# Выталкиваем остатки из стека* output.append(stack.pop())  
  
 return output  
  
  
*# Функция обработки `if`*def process\_if\_statement(expression):  
 expression = expression.replace("(", " ( ").replace(")", " ) ")  
 tokens = expression.split()  
 opz\_condition = to\_opz(tokens) *# Переводим в ОПЗ* return " ".join(opz\_condition) + " УПЛ"  
  
  
*# Функция обработки `while`*def process\_while\_statement(expression):  
 expression = expression.replace("(", " ( ").replace(")", " ) ")  
 tokens = expression.split()  
 opz\_condition = to\_opz(tokens) *# Переводим в ОПЗ* return " ".join(opz\_condition) + " УЦ" *# УЦ - Условный Цикл  
  
  
# Функция обработки `for`*def process\_for\_statement(expression, loop\_body):  
 expression = expression.replace("(", " ( ").replace(")", " ) ")  
 tokens = expression.split(";")  
  
 if len(tokens) != 3:  
 return "Ошибка в синтаксисе for"  
  
 init\_part = to\_opz(tokens[0].split()) *# `i = 0`* condition\_part = to\_opz(tokens[1].split()) *# `i < 10`* increment\_part = to\_opz(tokens[2].split()) *# `i = i + 1`* result = [  
 " ".join(init\_part), *# Инициализация (выполняется один раз)* " ".join(condition\_part) + " УЦ" *# Проверка условия перед каждой итерацией* ]  
  
 *# Добавляем тело цикла перед инкрементом* result.extend(loop\_body)  
  
 *# Добавляем инкремент перед возвратом к проверке условия* result.append(" ".join(increment\_part))  
  
 *# Условный переход к проверке условия* result.append("УЦ")  
  
 return "\n".join(result)  
*# Основная функция обработки кода*def convert\_to\_opz\_plain(code\_lines):  
 result = []  
 i = 0  
 while i < len(code\_lines):  
 line = code\_lines[i].strip()  
  
 if line in ["{", "}"]: *# Игнорируем фигурные скобки* i += 1  
 continue  
  
 if line.startswith("if"):  
 condition = line[line.index("(") + 1: line.rindex(")")]  
 result.append(process\_if\_statement(condition))  
 i += 1  
 continue  
  
 if line.startswith("while"):  
 condition = line[line.index("(") + 1: line.rindex(")")]  
 result.append(process\_while\_statement(condition))  
 i += 1  
 continue  
  
 if line.startswith("for"):  
 condition = line[line.index("(") + 1: line.rindex(")")]  
 loop\_body = []  
 i += 1  
  
 *# Собираем тело цикла* while i < len(code\_lines) and code\_lines[i].strip() != "}":  
 loop\_body.append(code\_lines[i].strip())  
 i += 1  
  
 processed\_body = convert\_to\_opz\_plain(loop\_body) *# Преобразуем тело цикла* result.append(process\_for\_statement(condition, processed\_body))  
 i += 1  
 continue  
  
 if "=" in line:  
 var\_name, expr = line.split("=")  
 var\_name = var\_name.strip()  
 expr = expr.strip().rstrip(";")  
 expr = expr.replace("(", " ( ").replace(")", " ) ").replace("[", " [ ").replace("]", " ] ")  
 expr\_tokens = expr.split()  
 opz\_expr = to\_opz(expr\_tokens)  
 result.append(f"{var\_name} {' '.join(opz\_expr)} =")  
 i += 1  
 continue  
  
 if "[" in line and "]" in line:  
 array\_access = line.replace("[", " [ ").replace("]", " ] ").split()  
 opz\_expr = to\_opz(array\_access)  
 result.append(" ".join(opz\_expr))  
 i += 1  
 continue  
  
 i += 1  
  
 return result  
  
*# Пример кода*code\_example = [  
 "if ((a - b) > 8) {",  
 " while ((a + b) < 20) {",  
 " x = arr[i] + 3;",  
 " }",  
 " for (i = 0; i < 10; i = i + 1) {",  
 " y = b[i] \* 2;",  
 " }",  
 "}"  
]  
  
*# Выполняем обработку кода*opz\_result = convert\_to\_opz\_plain(code\_example)  
  
*# Вывод результата*for line in opz\_result:  
 print(line)