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 1 АЭМ 3 + =

i 0 =

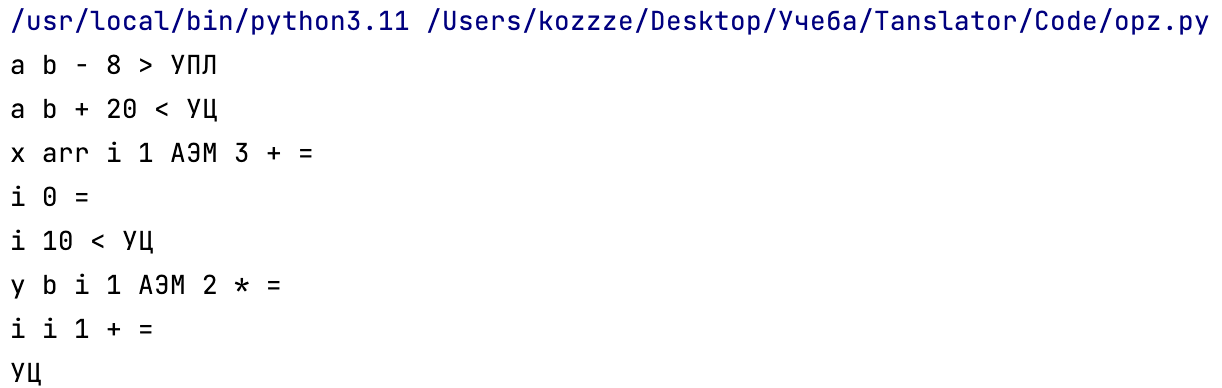
i 10 < УЦ

y b i 1 АЭМ 2 \* =

i i 1 + =

УЦ

Фактический результат:



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

Листинг программы

*import re  
  
OPERATOR\_PRIORITY = {  
 "=": 1,  
 "||": 2, "&&": 3,  
 "<": 4, "<=": 4, ">": 4, ">=": 4, "==": 4, "!=": 4,  
 "+": 5, "-": 5,  
 "\*": 6, "/": 6, "%": 6,  
 "(": 0, ")": 0, "[": -1, "]": -1  
}  
  
def to\_opz(expression\_tokens):  
 output = []  
 stack = []  
  
 for token in expression\_tokens:  
 if token.isalnum() or "АЭМ" in token:  
 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:  
 output.append(stack.pop())  
  
 return output  
  
def process\_array\_access(expression):  
 tokens = expression.replace("[", " [ ").replace("]", " ] ").split()  
 output = []  
 i = 0  
  
 while i < len(tokens):  
 if i < len(tokens) - 2 and tokens[i + 1] == "[":  
 array\_name = tokens[i]  
 index\_expr = [tokens[i + 2]]  
 index\_opz = to\_opz(index\_expr)  
 array\_access\_opz = [array\_name] + index\_opz + ["1 АЭМ"]  
 output.extend(array\_access\_opz) # Добавляем массив как цельный операнд  
 i += 3  
 elif tokens[i] != "]":  
 output.append(tokens[i])  
 i += 1  
  
 return output  
  
def process\_if\_statement(expression):  
 return " ".join(to\_opz(expression.split())) + " УПЛ"  
  
def process\_while\_statement(expression):  
 return " ".join(to\_opz(expression.split())) + " УЦ"  
  
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  
  
 # Обрабатываем `if`  
 if line.startswith("if"):  
 condition = line[line.index("(") + 1: line.rindex(")")]  
 tokens = condition.replace("(", " ( ").replace(")", " ) ").split()  
 result.append(" ".join(to\_opz(tokens)) + " УПЛ")  
 i += 1  
 continue  
  
 # Обрабатываем `while`  
 if line.startswith("while"):  
 condition = line[line.index("(") + 1: line.rindex(")")]  
 tokens = condition.replace("(", " ( ").replace(")", " ) ").split()  
 result.append(" ".join(to\_opz(tokens)) + " УЦ")  
 i += 1  
 continue  
  
 # Обрабатываем `for`  
 if line.startswith("for"):  
 condition = line[line.index("(") + 1: line.rindex(")")]  
 tokens = condition.split(";")  
  
 if len(tokens) != 3:  
 result.append("Ошибка в синтаксисе for")  
 i += 1  
 continue  
  
 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.append(f"{' '.join(init\_part)}")  
 result.append(f"{' '.join(condition\_part)} УЦ")  
  
 loop\_body = []  
 i += 1  
  
 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)  
 result.extend(processed\_body)  
  
 result.append(f"{' '.join(increment\_part)}")  
 result.append("УЦ")  
  
 i += 1  
 continue  
  
 # Обрабатываем присваивание  
 if "=" in line:  
 if line.count("=") > 1 and "==" not in line:  
 var\_name, expr = line.split("=", 1) # Разделяем только по первому `=`  
 else:  
 var\_name, expr = line.split("=")  
  
 var\_name = var\_name.strip()  
 expr = expr.strip().rstrip(";")  
  
 if "[" in var\_name and "]" in var\_name:  
 var\_name = " ".join(process\_array\_access(var\_name)) # Обрабатываем массив  
  
 expr\_tokens = process\_array\_access(expr) # Получаем список токенов  
 expr\_tokens = to\_opz(expr\_tokens) # Пропускаем через ОПЗ  
  
 result.append(f"{var\_name} {' '.join(expr\_tokens)} =")  
 i += 1  
 continue  
  
 # Обрабатываем обращение к массиву без присваивания  
 if "[" in line and "]" in line:  
 array\_tokens = process\_array\_access(line)  
 result.append(" ".join(array\_tokens))  
 i += 1  
 continue  
  
 i += 1  
  
 return result  
  
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;",  
 " }",  
 "}"  
]  
  
opz\_result = convert\_to\_opz\_plain(code\_example)  
  
for line in opz\_result:  
 print(line)*