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 № 3

по дисциплине «Методы разработки трансляторов»

Тема: «Перевод ОПЗ исходного

выражения в текст на выходном языке. Генерация машинного

кода»

Выполнил: ст. гр. 39/1

Козлов Э.Д.

Проверил: Преподаватель

Усов П.Е

Краснодар

2025

Ход работы:

Постановка задачи: Разработать интерпретатор обратной польской записи (ОПЗ), способный восстановить исходный программный код на языке C++ из ОПЗ-представления, полученного в предыдущей лабораторной работе.

**Общая идея:**

В этой лабораторной работе была реализована стековая интерпретация ОПЗ. Программа читает строки, содержащие команды на ОПЗ, и восстанавливает соответствующий C++ код, учитывая вложенные управляющие конструкции: if, while, for, присваивания и обращения к массивам.

Приведем тесты программы.

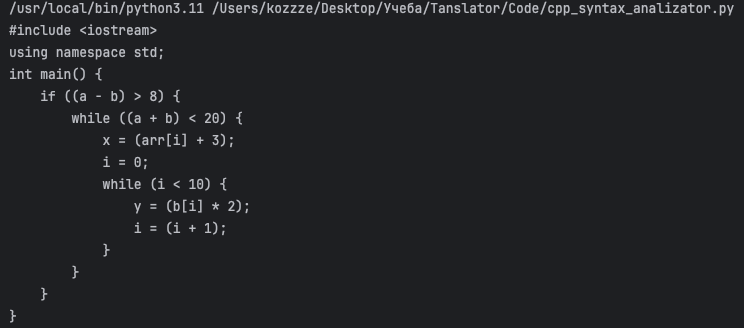
Исходный код на ЯП С:

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

Результат перевода в ОПЗ:

a b - 8 > УПЛ  
a b + 20 < УЦ  
x arr i АЭМ 3 + =  
i 0 =  
i 10 < УЦ  
y b i АЭМ 2 \* =  
i i 1 + =  
УЦ

**Восстановленный код на языке C++:**



Можно заметить, что цикл for переводится с использованием цикла while. Цикл for — это синтаксический сахар (удобная форма записи), которая на этапе компиляции преобразуется в конструкцию с использованием while. Это означает, что с точки зрения логики выполнения и семантики программы, цикл for полностью эквивалентен циклу while, при условии правильного переноса инициализации и итерации.

Вывод: В результате выполнения лабораторной работы была успешно реализована программа, восстанавливающая исходный код на C++ из представления в ОПЗ. Полученный результат корректно отражает структуру начального кода, включая вложенные условные и циклические операторы. Программа читает ОПЗ и восстанавливает полную структуру C++-программы с правильными отступами и синтаксисом.

Листинг программы

*import re  
  
# Операции и ключевые слова  
binary\_ops = {'+': '+', '-': '-', '\*': '\*', '/': '/', '<': '<', '>': '>', '==': '=='}  
commands = {'АЭМ': 'ARRAY\_ACCESS', '=': 'ASSIGN', 'УПЛ': 'IF\_COND', 'УЦ': 'CTRL\_FLOW'}  
  
# Простейший стековый интерпретатор OPЗ для восстановления C++ кода  
class OPZParser:  
 def \_\_init\_\_(self, tokens):  
 self.tokens = tokens  
 self.pos = 0  
 self.output = []  
 self.stack = []  
 self.indent\_level = 0  
 self.loop\_context = []  
  
 def indent(self):  
 return ' ' \* self.indent\_level  
  
 def parse(self):  
 # Добавим стандартные заголовки C++  
 self.output.append("#include <iostream>")  
 self.output.append("using namespace std;")  
 self.output.append("int main() {")  
 self.indent\_level += 1  
  
 while self.pos < len(self.tokens):  
 token = self.tokens[self.pos]  
 if token in binary\_ops:  
 b = self.stack.pop()  
 a = self.stack.pop()  
 self.stack.append(f'({a} {binary\_ops[token]} {b})')  
 elif token == 'АЭМ':  
 index = self.stack.pop()  
 array = self.stack.pop()  
 self.stack.append(f'{array}[{index}]')  
 elif token == '=':  
 value = self.stack.pop()  
 var = self.stack.pop()  
 self.output.append(f'{self.indent()}{var} = {value};')  
 elif token == 'УПЛ':  
 condition = self.stack.pop()  
 self.output.append(f'{self.indent()}if {condition} {{')  
 self.indent\_level += 1  
 elif token == 'УЦ':  
 # Разные случаи: while, for или конец блока  
 if self.loop\_context and self.loop\_context[-1] == 'for':  
 self.indent\_level -= 1  
 self.output.append(f'{self.indent()}}}')  
 self.loop\_context.pop()  
 elif self.stack:  
 condition = self.stack.pop()  
 self.output.append(f'{self.indent()}while {condition} {{')  
 self.indent\_level += 1  
 else:  
 self.indent\_level -= 1  
 self.output.append(f'{self.indent()}}}')  
 else:  
 self.stack.append(token)  
 self.pos += 1  
  
 # Закрываем все открытые блоки  
 while self.indent\_level > 0:  
 self.indent\_level -= 1  
 self.output.append(f'{self.indent()}}}')  
  
 return self.output  
  
  
def parse\_opz(opz\_lines):  
 # Очистка и разбиение токенов по пробелам, пропуская заголовки  
 tokens = []  
 for line in opz\_lines:  
 line = line.strip()  
 if not (line.startswith("#include") or line.startswith("using") or line.startswith("int main")):  
 tokens.extend(line.split())  
 parser = OPZParser(tokens)  
 return parser.parse()  
  
  
# Пример использования  
opz\_input = [  
 "a b - 8 > УПЛ",  
 "a b + 20 < УЦ",  
 "x arr i АЭМ 3 + =",  
 "i 0 =",  
 "i 10 < УЦ",  
 "y b i АЭМ 2 \* =",  
 "i i 1 + =",  
 "УЦ"  
]  
  
cpp\_lines = parse\_opz(opz\_input)  
for line in cpp\_lines:  
 print(line)*