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. 36/1

Шевякин А.А.

Проверил: профессор кафедры

Вишняков Ю. М.

Краснодар

2024

Задача: Построение программы для перевода закодированного текста исходной программы в обратную польскую запись. Программа получает на входе файл - результат лексического анализа и строит обратную польскую запись исходной программы.

№ варианта	Входной язык	Выходной язык
21	C++	PHP

Ход работы

1) Обратная польская запись (ОПЗ) – представляет собой одну из форм записи выражений и операторов, отличительной особенностью которой является расположение аргументов (операндов) перед операцией (оператором).

Например, выражение, записанное в обычной скобочной записи,

(a+d)/c+b*(e+d),

в ОПЗ имеет следующее представление:

ad+c/bed+*+.

Обратная польская запись получила широкое распространение благодаря своему основному преимуществу ОПЗ может быть вычислена за один просмотр цепочки слева направо, который часто называют проходом.

2) Исследованию формальных способов преобразования арифметических и логических выражений в ОПЗ посвящены многочисленные исследования, однако в практике системного программирования наибольшее распространение получили способы преобразования на основе алгоритма Дейкстры.

На вход алгоритма посимвольно поступает исходное выражение. Операнды исходного выражения пропускаются на выход и формируют так же посимвольно выходную строку. Операции обрабатываются по определенным правилам на основе стека.

Для реализации такой обработки известное в системном программировании понятие стека используется также в алгоритме Дейкстры для размещения в нем операций. При этом предварительно каждой операции приписывается свой приоритет на основе таблицы приоритетов, которая приведена ниже(таблица 1).

Таблица 1 – Таблица приоритетов

Входной элемент	Приоритет
(for if while [АЭМ Ф begin	0
) , ; do else]	1
=	2
||	3
&&	4
!	5
< <= != == > >=	6
+ - +=	7

-= *= /=	7
* / %	8
} void int double bool string float args return main	9

Обработка оператора цикла с предусловием while выражение do оператор;:

1. Символ while из входной строки заносится в стек. В стеке к символу while добавляется рабочая метка Mi и после этого в выходную строку записывается часть Mi:.

2. Символ DO выталкивает в выходную строку все операции из стека до ближайшего while Mi. В стеке к while Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается часть Mi+1 УПЛ.

3. Символ ‘;’ указывает на конец оператора цикла с предусловием и выталкивает из стека все символы до ближайшего while Mi Mi+1, при этом сам while уничтожается, а в выходную строку помещается Mi БП Mi+1:.

Обработка оператора цикла с счетчиком for переменная = значение1 to значение2 do оператор;:

1. for переменная из входной строки заносится в стек.

2. Символ to выталкивает в выходную строку все операции из стека до ближайшего for переменная. В стеке к for переменная добавляется рабочая метка Mi и после этого в выходную строку записывается Mi: переменная.

3. Символ do выталкивает в выходную строку все операции из стека до ближайшего for переменная Mi. В стеке к for переменная Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается <= Mi+1 УПЛ.

4. Символы ‘;’ и end указывают на конец оператора цикла с счетчиком и выталкивают из стека все символы до ближайшего for переменная Mi Mi+1, при этом сам for уничтожается, а в выходную строку помещается метка переменная переменная 1 + := Mi БП Mi+1:.

Результаты работы программы

Для наглядности в интерфейсе текст программы представлен на исходном языке, но в реальном трансляторе входной текст является результатом работы лексического анализатора. В нем уже выделены и классифицированы лексемы.

Пример кода на исходном языке №1

```
void main(){
int b = 0;
while (b < 1000){
b += 3;
}
if (b > 1000){
b = b-2/(67+12)*4;}
else b = f(b);

int arr[1000];
for(int i =0; i<b; i++){
arr[i] = i;
}
}
```

Пример кода на исходном языке №2

```
int main() {
int a = 5;
double b = 3.14;
```

```
char c = 'A';
bool d = true;

int sum = a + 10;
a = a - 5;
double product = b * 2.5;
int quotient = a / 2;
int remainder = a % 2;
double power = pow(b, 2);

std::cout << "Enter number: ";
std::cin >> a;
std::cout << "Entered: " << a << std::endl;

if (a > 0) {
std::cout << "Positive" << std::endl;
} else if (a < 0) {
std::cout << "Negative" << std::endl;
} else {
std::cout << "Zero" << std::endl;
}

int numbers[5];

for (int i = 0; i < 5; i++) {
    int temp = func(i+1);
    numbers[i] = temp
}

int i = 0;


do {
std::cout << "iteration " << i << std::endl;
i++;
} while (i < 5);
return 0;
}
```

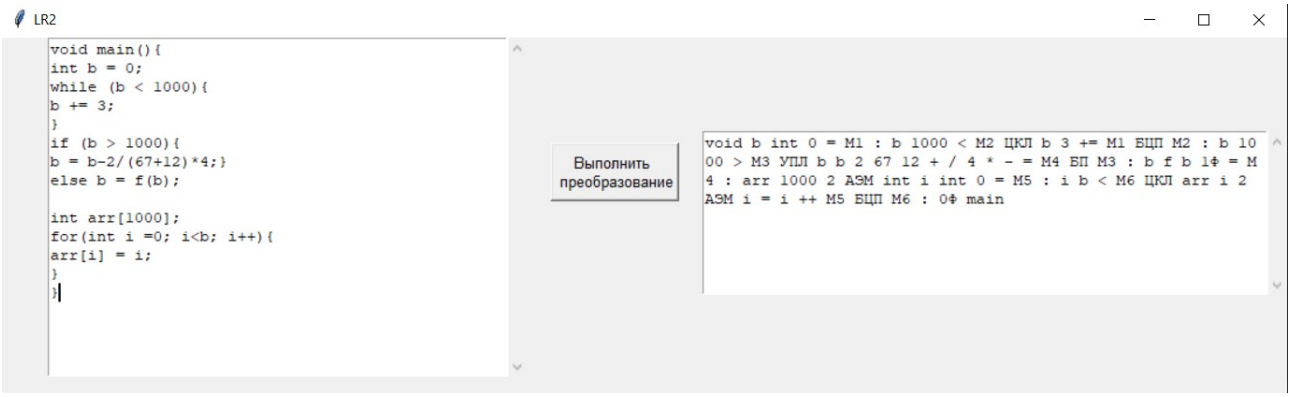

Рисунок 1 — интерфейс с результатом перевода примера №1

Результат перевода в ОПЗ: void b int 0 = M1 : b 1000 < M2 ЦКЛ b 3 += M1 БЦП M2 : b 1000 > M3 УПЛ b b 2 67 12 + / 4 * - = M4 БП M3 : b f b 1Ф = M4 : arr 1000 2 АЭМ int i int 0 = M5 : i b < M6 ЦКЛ arr i 2 АЭМ i = i ++ M5 БЦП M6 : 0Ф main

Рисунок 2 — интерфейс с результатом перевода примера №2

Результат перевода в ОПЗ: int a int 5 = b double 3.14 = char c 'A' = d bool true = sum int a 10 + = a a 5 - = product double b 2.5 * = quotient int a std : : cout < "Enter number: " < = std : : cin > a > std : : cout < "Entered: " < < a < < std : : endl < a 0 > M1 УПЛ std : : cout < "Positive" < < std : : endl < M2 БП M1 : a 0 < M3 УПЛ std : : cout < "Negative" < < std : : endl < M4 БП M3 : std : : cout < "Zero" < < std : : endl < M4 : numbers 5 2 АЭМ int M2 : i int 0 = M5 : i 5 < M6 ЦКЛ temp int func i 1 + 1Ф = numbers i 2 АЭМ temp = i ++ M5 БЦП M6 : i int 0 = std : : cout < "iteration " < < i < < std : : endl < i ++ M7 : i 5 < M8 ЦКЛ M7 БЦП M8 : 0 return while M7 M8 do 0Ф main

Листинг программы

```
import json
import re
from tkinter import *
import tkinter.scrolledtext as st
import re

def prog():

    CLASSES_OF_TOKENS = ['W', 'I', 'O', 'R', 'N', 'C']

    def is_identifier(token):
        return re.match(r'^I\d+$', inverse_tokens[token])

    def get_priority(token):
        if token in ['(', 'for', 'if', 'while', '[', 'АЭМ', 'Ф', '{']:
            return 0
        if token in [')', ',', ';', 'do', 'else', ']']:
            return 1
        if token == '=':
            return 2
        if token == '||':
            return 3
        if token == '&&':
            return 4
        if token == '!':
            return 5
        if token in ['<', '<=', '!=', '=', '>', '>=']:
            return 6
        if token in ['+', '-', '+=', '-=', '*=', '/=']:
            return 7
        if token in ['*', '/', '%']:
            return 8
        if token in ['}', 'void','int', 'double', 'bool', 'string', 'float', 'args','return','cout', 'main']:
            return 9
        return -1

    # лексемы (код-значение)
    tokens = {}

    # файлы, содержащие все таблицы лексем
    for token_class in CLASSES_OF_TOKENS:
        with open('%s.json' % token_class, 'r') as read_file:
            data = json.load(read_file)
            tokens.update(data)

    # лексемы (значение-код)
    inverse_tokens = {val: key for key, val in tokens.items()}

    # файл, содержащий последовательность кодов лексем входной программы
    f = open('tokens.txt', 'r')
    inp_seq = f.read()
    f.close()

    regexp = '[' + '|'.join(CLASSES_OF_TOKENS) + ']' + '\d+'
    match = re.findall(regexp, inp_seq)

    t = [tokens[i] for i in match]
```

```
i = 0
stack = []
out_seq = ''
aem_count = proc_num = proc_level = operand_count = 1
func_count = tag_count = proc_num = if_count = while_count = \
        begin_count = end_count = bracket_count = 0
is_if = is_while = is_description_var = False
while i < len(t):
    p = get_priority(t[i])
    if p == -1:
        if t[i] != '\n' and t[i] != '\t':
            out_seq += t[i] + ' '
    else:
        if t[i] == '[':
            aem_count += 1
            stack.append(str(aem_count) + ' АЭМ')
        elif t[i] == ']':
            while not(re.match(r'^\d+ АЭМ$', stack[-1])):
                out_seq += stack.pop() + ' '
            out_seq += stack.pop() + ' '
            aem_count = 1
        elif t[i] == '(':
            if is_identifier(t[i - 1]):
                if t[i + 1] != ')':
                    func_count += 1
                stack.append(str(func_count) + ' Ф')
            else:
                stack.append(t[i])
            bracket_count += 1
        elif t[i] == ')':
            while stack[-1] != '(' and not(re.match(r'^\d+ Ф$', stack[-1])):
                out_seq += stack.pop() + ' '
            if re.match(r'^\d+ Ф$', stack[-1]):
                stack.append(str(func_count + 1) + ' Ф')
                func_count = 0
            stack.pop()
            bracket_count -= 1
            if bracket_count == 0:
                if is_if:
                    while stack[-1] != 'if':
                        out_seq += stack.pop() + ' '
                    tag_count += 1
                    stack[-1] += ' М' + str(tag_count)
                    out_seq += 'М' + str(tag_count) + ' УПЛ '
                    is_if = False
                if is_while:
                    while not (re.match(r'^while М\d+$', stack[-1])):
                        out_seq += stack.pop() + ' '
                    tag_count += 1
                    out_seq += 'М' + str(tag_count) + ' ЦКЛ '
                    stack[-1] += ' М' + str(tag_count)
                    is_while = False

        elif t[i] == ',':
            while not(re.match(r'^\d+ АЭМ$', stack[-1])) and \
                not(re.match(r'^\d+ Ф$', stack[-1])) and \
                not(re.match(r'^var', stack[-1])):
                out_seq += stack.pop() + ' '
            if re.match(r'^\d+ АЭМ$', stack[-1]):
                aem_count += 1
                stack.append(str(aem_count) + ' АЭМ')
```

```
            if re.match(r'^\d+ Ф$', stack[-1]):
                func_count += 1
                stack.append(str(func_count) + ' Ф')
        elif t[i] == 'if':
            stack.append(t[i])
            if_count += 1
            bracket_count = 0
            is_if = True
        elif t[i] == 'else':
            while not(re.match(r'^if M\d+$', stack[-1])):
                out_seq += stack.pop() + ' '
            stack.pop()
            tag_count += 1
            stack.append('if M' + str(tag_count))
            out_seq += 'M' + str(tag_count) + ' БП M' + str(tag_count - 1) + ' : '
        elif t[i] == 'while':
            tag_count += 1
            stack.append(t[i] + ' M' + str(tag_count))
            out_seq += 'M' + str(tag_count) + ' : '
            while_count += 1
            bracket_count = 0
            is_while = True


        elif t[i] == 'for':
            j = i + 2
            bracket_count = 1
            a = []
            while t[j] != ';':
                a.append(t[j])
                j += 1
                if t[j] == '(':
                    bracket_count += 1
                elif t[j] == ')':
                    bracket_count -= 1
            j += 1
            b = []
            while t[j] != ';':
                b.append(t[j])
                j += 1
                if t[j] == '(':
                    bracket_count += 1
                elif t[j] == ')':
                    bracket_count -= 1
            j += 1
            c = []
            while bracket_count != 0:
                c.append(t[j])
                j += 1
                if t[j] == '(':
                    bracket_count += 1
                elif t[j] == ')':
                    bracket_count -= 1
            j += 1
            d = []
            while t[j] != ';' and t[j] != '{':
                d.append(t[j])
                j += 1
            if t[j] == '{':
                j += 1
                bracket_count = 1
                d = ['{']
                while bracket_count != 0:
```

```
                d.append(t[j])
                j += 1
                if t[j] == '{':
                    bracket_count += 1
                elif t[j] == '}':
                    bracket_count -= 1
            d.append('}')
        j += 1
        t = t[:i] + a + [';', '\n', 'while', '('] + b + [')', '{', '\n'] + d + \
            ['\n'] + c + [';', '\n', '}'] + t[j:]
        i -= 1


    elif t[i] == 'sub':
        proc_num += 1
        stack.append('PROC ' + str(proc_num) + ' ' + str(proc_level))
    elif t[i] == '{':
        if len(stack) > 0 and re.match(r'^PROC', stack[-1]):
            num = re.findall(r'\d+', stack[-1])
            stack.pop()
            out_seq += '0 Ф ' + str(num[0]) + ' ' + str(num[1]) + ' НП '
            stack.append('PROC ' + str(proc_num) + ' ' + str(proc_level))
        begin_count += 1
        proc_level = begin_count - end_count + 1
        stack.append(t[i])
    elif t[i] == '}':
        end_count += 1
        proc_level = begin_count - end_count + 1
        while stack[-1] != '{':
            out_seq += stack.pop() + ' '
        stack.pop()
        if len(stack) > 0 and re.match(r'^PROC', stack[-1]):
            stack.pop()
            out_seq += 'КП '
        if if_count > 0 and re.match(r'^if M\d+$', stack[-1]):
            tag = re.search('M\d+', stack[-1]).group(0)
            j = i + 1
            while j < len(t) and t[j] == '\n':
                j += 1
            if j >= len(t) or t[j] != 'else':
                stack.pop()
                out_seq += tag + ' : '
                if_count -= 1
        if while_count > 0 and re.match(r'^while M\d+ M\d+$', stack[-1]):
            tag = re.findall('M\d+', stack[-1])
            stack.pop()
            out_seq += tag[0] + ' БЦП ' + tag[1] + ' : '
            while_count -= 1
    elif t[i] == ';':
        if len(stack) > 0 and re.match(r'^PROC', stack[-1]):
            num = re.findall(r'\d+', stack[-1])
            stack.pop()
            out_seq += str(num[0]) + ' ' + str(num[1]) + ' НП '
        elif len(stack) > 0 and stack[-1] == 'end':
            stack.pop()
            out_seq += 'КП '
        elif is_description_var:
            proc_num, proc_level = re.findall('\d+', stack[-1])
            stack.pop()
            out_seq += str(operand_count) + ' ' + proc_num + ' ' + proc_level + \
                   ' КО '
            is_description_var = False
        elif if_count > 0 or while_count > 0:
```

```
                while not(len(stack) > 0 and stack[-1] == '{') and \
                    not(if_count > 0 and re.match(r'^if M\d+$', stack[-1])) and \
                  not(while_count > 0 and re.match(r'^while M\d+ M\d+$', stack[-1])):
                    out_seq += stack.pop() + ' '
                if if_count > 0 and re.match(r'^if M\d+$', stack[-1]):
                    tag = re.search('M\d+', stack[-1]).group(0)
                    j = i + 1
                    while t[j] == '\n':
                        j += 1
                    if t[j] != 'else':
                        stack.pop()
                    out_seq += tag + ' : '
                    if_count -= 1
                if while_count > 0 and re.match(r'^while M\d+ M\d+$', stack[-1]):
                    tag = re.findall('M\d+', stack[-1])
                    out_seq += tag[0] + ' БЦП ' + tag[1] + ' : '
                    while_count -= 1
            else:
                while len(stack) > 0 and stack[-1] != '{':
                    out_seq += stack.pop() + ' '
        else:
            while len(stack) > 0 and get_priority(stack[-1]) >= p:
                out_seq += stack.pop() + ' '
            stack.append(t[i])
    i += 1

while len(stack) > 0:
    out_seq += stack.pop() + ' '

out_seq = re.sub(r'(\d) Ф', r'\1Ф', out_seq)

# файл, содержащий обратную польскую запись
f = open('reverse_polish_entry.txt', 'w')
f.write(out_seq)
f.close()

def write_txt(data):
    with open('cpp2.txt','w') as file:
        file.write(data)

def clicked():
    write_txt(codetxt.get("1.0","end"))

    opzstext.delete("1.0",END)

    prog()

    f1 = open('reverse_polish_entry.txt','r')
    text = f1.read()
    opzstext.insert("1.0",text)
    f1.close()

window=Tk()
window.title("LR2")

window.geometry('1100x500')

codetxt=st.ScrolledText(window)
```

```
codetxt.place(x=40,y=0,width=410,height=290)


opzstext=st.ScrolledText(window)
opzstext.place(x=600,y=80,width=500,height=140)


btngo=Button(window,text="Выполнить \n преобразование",command=clicked,font=("Arial", 10))
btngo.place(x=470,y=90,width=110,height=50)


window.mainloop()
```