Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №2**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Перевод исходной программы в обратную польскую запись»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. С. Прозоров

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

**СОДЕРЖАНИЕ**

[1 Вариант задания 3](#_Toc103025612)

[2 Понятие обратной польской записи 3](#_Toc103025613)

[3 Алгоритм Дейкстры 4](#_Toc103025614)

[4 Перевод операторов цикла в ОПЗ 7](#_Toc103025615)

[5 Результаты экспериментов 9](#_Toc103025616)

[Приложение А Листинг программы 13](#_Toc103025617)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 43 | Java | C# |

Разработать программу 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.

Отчет по работе должен содержать полное описание алгоритма Дейкстры: таблицу приоритетов операторов и операций, а также алгоритм работы со стеком. Листинг программы и комментарии к нему, пример.

**2 Понятие обратной польской записи**

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.

**3 Алгоритм Дейкстры**

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 2).

Таблица 2 – Таблица приоритетов

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

Продолжение таблицы 2

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| <  <=  !=  =  >  >= | 6 |
| +  -  +=  -=  \*=  /= | 7 |
| \*  /  % | 8 |
| }  public  static  void  procedure  int  double  boolean  string  float  args  return  system | 9 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| out  println  main | 9 |

**4 Перевод операторов цикла в ОПЗ**

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 постусловием REPEAT оператор UNTIL выражение; можно заменить последовательностью операторов

1. Символ ‘;’ указывает на конец оператора цикла с постусловием и выталкивают из стека все символы до ближайшего REPEAT Mi, при этом сам REPEAT уничтожается, а в выходную строку помещаются Mi+1 УПЛ 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:.

**5 Результаты экспериментов**

Примеры работы программы для программы для тестирования.

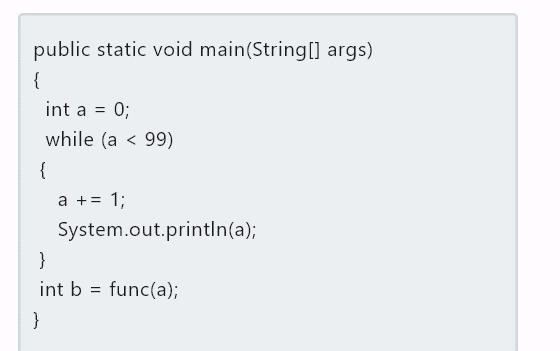


Рисунок 1 – Скриншот файла №1, содержащего текст на входном языке программирования

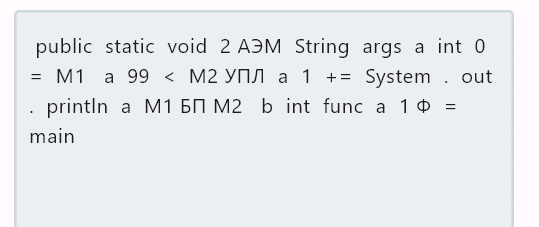


Рисунок 2 – Результат работы программы №1

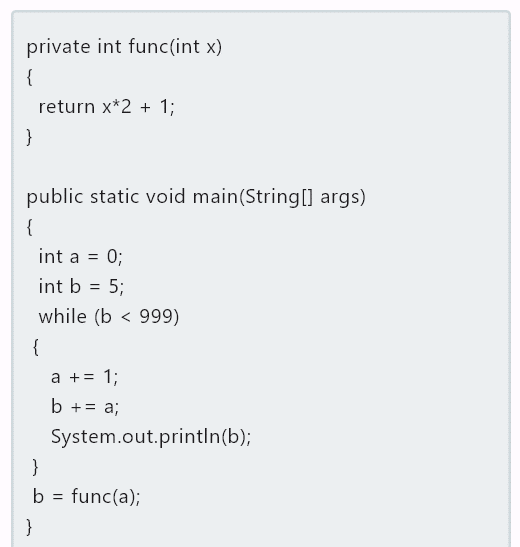


Рисунок 3 – Скриншот файла №2, содержащего текст на входном языке программирования

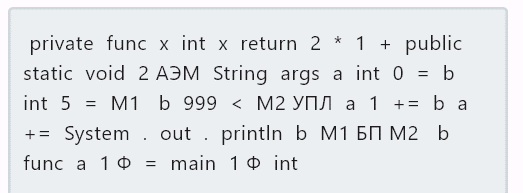


Рисунок 4 – Результат работы программы №2

**ПРИЛОЖЕНИЕ А   
Листинг программы**

import '../lexical\_anallyzer/models/lexical\_analyzer\_output.dart';

import '../lexical\_anallyzer/tokens/token.dart';

final class ReversePolishEntryOutput {

final List<String> result;

ReversePolishEntryOutput({

required this.result,

});

String convertToText() {

return result.fold(

'',

(previousValue, element) => '$previousValue $element',

);

}

}

class ReversePolishEntry {

ReversePolishEntryOutput execute(LexicalAnalyzerOutput input) {

List<String> identifiers =

input.identifiers.map((e) => e.value.toString()).toList();

List<String> t = input.tokens

.map(

(e) => e is ValToken ? e.value.toString() : e.lexeme,

)

.toList()

..removeWhere((element) => element == ' ');

List<String> stack = [], result = [];

int aemCount = 1, procLevel = 1, operandCount = 1;

int funcCount = 0,

tagCount = 0,

procNum = 0,

ifCount = 0,

whileCount = 0,

beginCount = 0,

endCount = 0,

bracketCount = 0;

int i = 0;

bool isIf = false, isWhile = false, isDescriptionVar = false;

while (i < t.length) {

int p = \_getPriority(t[i]);

if (p == -1) {

if (t[i] != '\n' && t[i] != '\t') {

result.add('${t[i]} ');

}

} else {

if (t[i] == '[') {

aemCount += 1;

stack.add('$aemCount АЭМ');

} else if (t[i] == ']') {

while (!(RegExp(r'^\d+ АЭМ$').hasMatch(stack.last))) {

result.add('${stack.removeLast()} ');

}

result.add('${stack.removeLast()} ');

aemCount = 1;

} else if (t[i] == '(') {

if (identifiers.contains(t[i - 1])) {

if (t[i + 1] != ')') {

funcCount += 1;

}

stack.add('$funcCount Ф');

} else {

stack.add(t[i]);

}

bracketCount += 1;

} else if (t[i] == ')') {

while (

stack.last != '(' && !(RegExp(r'^\d+ Ф$').hasMatch(stack.last))) {

result.add('${stack.removeLast()} ');

}

if (RegExp(r'^\d+ Ф$').hasMatch(stack.last)) {

stack.add('${funcCount + 1} Ф');

funcCount = 0;

}

stack.removeLast();

bracketCount -= 1;

if (bracketCount == 0) {

if (isIf) {

while (stack.last != 'if') {

result.add('${stack.removeLast()} ');

}

tagCount += 1;

stack.last += ' М$tagCount';

result.add('М$tagCount УПЛ ');

isIf = false;

}

if (isWhile) {

while (!(RegExp(r'^while М\d+$').hasMatch(stack.last))) {

result.add('${stack.removeLast()} ');

}

tagCount += 1;

result.add('М$tagCount УПЛ ');

stack.last += ' М$tagCount';

isWhile = false;

}

}

} else if (t[i] == ',') {

while (!(RegExp(r'^\d+ АЭМ$').hasMatch(stack.last)) &&

!(RegExp(r'^\d+ Ф$').hasMatch(stack.last)) &&

!(RegExp(r'^var').hasMatch(stack.last))) {

result.add('${stack.removeLast()} ');

}

if (RegExp(r'^\d+ АЭМ$').hasMatch(stack.last)) {

aemCount += 1;

stack.add('$aemCount АЭМ');

}

if (RegExp(r'^\d+ Ф$').hasMatch(stack.last)) {

funcCount += 1;

stack.add('$funcCount Ф');

}

} else if (t[i] == 'if') {

stack.add(t[i]);

ifCount += 1;

bracketCount = 0;

isIf = true;

} else if (t[i] == 'else') {

while (!(RegExp(r'^if М\d+$').hasMatch(stack.last))) {

result.add('${stack.removeLast()} ');

}

stack.removeLast();

tagCount += 1;

stack.add('if М$tagCount');

result.add('М$tagCount БП М${tagCount - 1} ');

} else if (t[i] == 'while') {

tagCount += 1;

stack.add('${t[i]} М$tagCount');

result.add('М$tagCount ');

whileCount += 1;

bracketCount = 0;

isWhile = true;

} else if (t[i] == 'for') {

int j = i + 2;

bracketCount = 1;

List<String> a = [];

while (t[j] != ';') {

a.add(t[j]);

j += 1;

if (t[j] == '(') {

bracketCount += 1;

} else if (t[j] == ')') {

bracketCount -= 1;

}

}

j += 1;

List<String> b = [];

while (t[j] != ';') {

b.add(t[j]);

j += 1;

if (t[j] == '(') {

bracketCount += 1;

} else if (t[j] == ')') {

bracketCount -= 1;

}

}

j += 1;

List<String> c = [];

while (bracketCount != 0) {

c.add(t[j]);

j += 1;

if (t[j] == '(') {

bracketCount += 1;

} else if (t[j] == ')') {

bracketCount -= 1;

}

}

j += 1;

List<String> d = [];

while (t[j] != ';' && t[j] != '{') {

d.add(t[j]);

j += 1;

}

if (t[j] == '{') {

j += 1;

bracketCount = 1;

d = ['{'];

while (bracketCount != 0) {

d.add(t[j]);

j += 1;

if (t[j] == '{') {

bracketCount += 1;

} else if (t[j] == '}') {

bracketCount -= 1;

}

}

d.add('}');

}

j += 1;

//варнинг

t = t.take(14).toList() +

a +

[';', '\n', 'while', '('] +

b +

[')', '{', '\n'] +

d +

['\n'] +

c +

[';', '\n', '}'] +

t.skip(46).toList();

i -= 1;

} else if (t[i] == 'sub') {

procNum += 1;

stack.add('PROC $procNum $procLevel');

} else if (t[i] == '{') {

if (stack.isNotEmpty && RegExp(r'^PROC').hasMatch(stack.last)) {

final number = RegExp(r'\d+').allMatches(stack.last).toList();

stack.removeLast();

result.add('0 Ф ${number[0]} ${number[1]} НП ');

stack.add('PROC $procNum $procLevel');

}

beginCount += 1;

procLevel = beginCount - endCount + 1;

stack.add(t[i]);

} else if (t[i] == '}') {

endCount += 1;

procLevel = beginCount - endCount + 1;

while (stack.last != '{') {

result.add('${stack.removeLast()} ');

}

stack.removeLast();

if (stack.isNotEmpty && RegExp(r'^PROC').hasMatch(stack.last)) {

stack.removeLast();

result.add('КП ');

}

if (ifCount > 0 && RegExp(r'^if М\d+$').hasMatch(stack.last)) {

String tag = RegExp(r'М\d+').firstMatch(stack.last)!.group(0)!;

int j = i + 1;

while (j < t.length && t[j] == '\n') {

j += 1;

}

if (j >= t.length || t[j] != 'else') {

stack.removeLast();

result.add('$tag ');

ifCount -= 1;

}

}

if (whileCount > 0 &&

RegExp(r'^while М\d+ М\d+$').hasMatch(stack.last)) {

final tag = RegExp(r'М\d+').allMatches(stack.last).toList();

stack.removeLast();

result.add('${tag[0].group(0)} БП ${tag[1].group(0)} ');

whileCount -= 1;

}

} else if (t[i] == ';') {

if (stack.isNotEmpty && RegExp(r'^PROC').hasMatch(stack.last)) {

// toList ??

final number = RegExp(r'\d+').allMatches(stack.last).toList();

stack.removeLast();

result.add('${number[0]} ${number[1]} НП ');

} else if (stack.isNotEmpty && stack.last == 'end') {

stack.removeLast();

result.add('КП ');

} else if (isDescriptionVar) {

//??

final tag = RegExp(r'\d+').allMatches(stack.last).toList();

procNum = int.parse(tag[0].group(0)!);

procLevel = int.parse(tag[1].group(0)!);

stack.removeLast();

result.add('$operandCount $procNum $procLevel КО ');

isDescriptionVar = false;

} else if (ifCount > 0 || whileCount > 0) {

while (!(stack.isNotEmpty && stack.last == '{') &&

!(ifCount > 0 && RegExp(r'^if М\d+$').hasMatch(stack.last)) &&

!(whileCount > 0 &&

RegExp(r'^while М\d+ М\d+$').hasMatch(stack.last))) {

result.add('${stack.removeLast()} ');

}

if (ifCount > 0 && RegExp(r'^if М\d+$').hasMatch(stack.last)) {

String tag = RegExp(r'М\d+').firstMatch(stack.last)!.group(0)!;

int j = i + 1;

while (t[j] == '\n') {

j += 1;

}

if (t[j] != 'else') {

stack.removeLast();

}

result.add('$tag ');

ifCount -= 1;

}

if (whileCount > 0 &&

RegExp(r'^while М\d+ М\d+$').hasMatch(stack.last)) {

final tag = RegExp(r'М\d+').allMatches(stack.last);

String tag1 = tag.first.group(0)!;

String tag2 = tag.skip(1).first.group(0)!;

result.add('$tag1 БП $tag2 ');

whileCount -= 1;

}

} else {

while (stack.isNotEmpty && stack.last != '{') {

result.add('${stack.removeLast()} ');

}

}

} else {

while (stack.isNotEmpty && \_getPriority(stack.last) >= p) {

result.add('${stack.removeLast()} ');

}

stack.add(t[i]);

}

}

i += 1;

}

while (stack.isNotEmpty) {

result.add('${stack.removeLast()} ');

}

return ReversePolishEntryOutput(result: result);

}

int \_getPriority(String token) {

if (['(', 'for', 'if', 'while', '[', 'АЭМ', 'Ф', '{'].contains(token)) {

return 0;

}

if ([')', ',', ';', 'do', 'else', ']'].contains(token)) {

return 1;

}

if (token == '=') {

return 2;

}

if (token == '||') {

return 3;

}

if (token == '&&') {

return 4;

}

if (token == '!') {

return 5;

}

if (['<', '<=', '!=', '=', '>', '>='].contains(token)) {

return 6;

}

if (['+', '-', '+=', '-=', '\*=', '/='].contains(token)) {

return 7;

}

if (['\*', '/', '%'].contains(token)) {

return 8;

}

if ([

'}',

'public.static.void',

'procedure',

'int',

'double',

'boolean',

'String',

'float',

'args',

'return',

'System.out.println',

'main'

].contains(token)) {

return 9;

}

return -1;

}

}