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 №3**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Перевод ОПЗ исходного выражения в текст на выходном языке. Генерация машинного кода»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. А. Корнилов

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](#_Toc163559050)

[2 Базовые понятия 3](#_Toc163559051)

[3 Правила генерации машинного кода 4](#_Toc163559052)

[3 Результаты экспериментов 7](#_Toc163559053)

[ПРИЛОЖЕНИЕ А Листинг программы и комментарии к нему 10](#_Toc163559054)

**1 Вариант задания**

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

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 47 | PHP | R |

Разработать программу для формирования по обратной польской записи текста на выходном языке.

Программа получает на входе файл, содержащий ОПЗ исходной программы, и строит текст программы на машинном язык в соответствии с заданием.

Отчет по работе должен содержать описание правил записи перечисленных выше элементов заданного выходного языка, алгоритм работы МП-автомата и описание семантических процедур, листинг программы и комментарии к нему, пример.

**2 Базовые понятия**

Перевод ОПЗ в текст на выходном (машинном) языке представляет собой следующий этап трансляции исходной программы в машинные коды. Для реализации этой процедуры также используется автомат с магазинной памятью (МП-автомат).

Эту процедуру можно схематично представить следующим образом:



Рисунок 1 – Схематичное представление процедуры

Чтение символов операций из ОПЗ инициирует семантические процедуры, которые генерируют соответствующие заготовки машинного кода. Так же обрабатываются собранные в стеке операнды данной. Например, в случае обработки переменных и констант извлекается соответствующая им информация из таблиц идентификаторов и констант, которая используется для образования правильных адресных частей соответствующих машинных команд.

**3 Правила генерации машинного кода**

Пусть в качестве машинного языка выступает язык программирования Бейсик. Его особенностью является обязательная нумерация строк и отсутствие символьных меток. Поэтому в операторах перехода в качестве меток используют номера строк, на которые нужно передать управление.

Для построения МП-автомата по переводу ОПЗ в машинные коды и его семантических процедур введем ряд внутренних переменных:

1. P – счетчик вспомогательных переменных;
2. STR – счетчик строк.

Кроме того, организуем таблицу меток, которая реализует отображение символьных меток исходного языка в номера строк машинного языка:

Таблица 2 – Таблица меток

|  |  |
| --- | --- |
| Метка | Номер строки |

Эта таблица потребуется в дальнейшем для замены символьных меток на номера строк, что также является особенностью Бейсика как выходного языка.

Рассмотрим работу МП-автомата.

1. Если элемент входной строки – идентификатор или константа, то он заносится в стек (в исходном виде, т.е. не условное обозначение, а имя из таблицы идентификаторов или константа из таблицы констант); вспомогательные переменные и константы переносятся без изменения.

2. Для каждой операции и оператора определяется арность, т.е. количество операндов, и соответствующая семантическая процедура.

3. После выполнения каждой семантической процедуры в выходную строку заносится символ <ВК>, счетчик строк STR наращивается на единицу и заносится в начало новой строки.

Семантические процедуры для операторов и операций приведены в таблице 3.

Таблица 3 – Семантические процедуры для операторов и операций

|  |  |
| --- | --- |
| Лексема | Действия |
| НП | Извлечь из стека два элемента, занести в выходную строку текст *"REM Начало процедуры арг2, арг1"* |
| КП | Занести в выходную строку текст  *"REM Конец процедуры"* |
| DFD  BFD  DFT  CHAR | Извлечь из стека *арг1* – число переменных k; извлечь из стека k аргументов; занести в выходную строку текст  *"REM Вещественные переменные арг1, арг2, …, аргk"* |
| КО | Извлечь из стека два аргумента |
| УПЛ | Извлечь из стека два аргумента, занести в выходную строку текст  *" IF NOT(арг2) THEN GOTO арг1"* |

Продолжение таблицы 3

|  |  |
| --- | --- |
| БП | Извлечь из стека один аргумент, занести в выходную строку текст  *"GOTO арг1"* |
| : | Извлечь из стека один аргумент, занести в таблицу меток *арг1* и значение счетчика STR |
| +  \*  >  < | Извлечь из стека два аргумента, нарастить счетчик вспомогательных переменных Р, занести в выходную строку текст  *"Rp = арг2 <операция> арг1",*  занести в стек Rp. |
| = | Извлечь из стека 2 аргумента и занести в выходную строку текст *"арг2 = арг1"* |

В стеке *арг1* – это элемент, находящийся в вершине стека. Увеличение номера аргумента показывает его удаление от вершины стека и обратно порядку занесения элементов в стек.

Для арифметических выражений в целях уменьшения количества операторов присваивания и временных переменных возможен вариант формирования строки "(арг2 <операция> арг1)" и занесение ее в стек как единого аргумента для последующих операций и операторов. Недостатком такого подхода является избыточность круглых скобок в выражениях.

**3 Результаты экспериментов**

Пример работы программы для программы для тестирования.

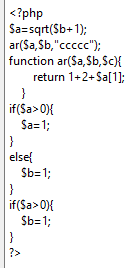


Рисунок 2 – Скриншот файла, содержащего текст 1 на входном языке программирования

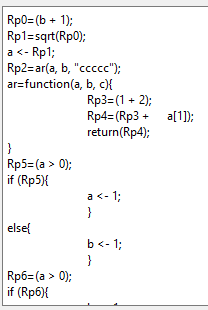


Рисунок 3 – Результат работы программы 1

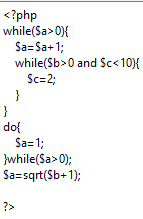


Рисунок 4 – Скриншот файла, содержащего текст 2 на входном языке программирования

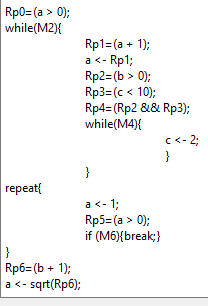


Рисунок 5 – Результат работы программы 2

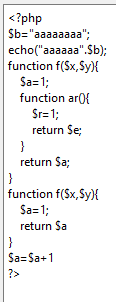


Рисунок 6 – Скриншот файла, содержащего текст 3 на входном языке программирования

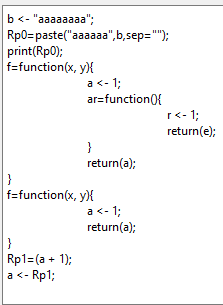


Рисунок 7 – Результат работы программы 3

**ПРИЛОЖЕНИЕ А   
Листинг программы и комментарии к нему**

def translate\_to\_R(self):  
 CLASSES\_OF\_TOKENS = ['W', 'I', 'O', 'R', 'N', 'C']  
 self.tokens = {}  
 *# файлы, содержащие все таблицы лексем* for token\_class in CLASSES\_OF\_TOKENS:  
 with open('./files/%s.json' % token\_class, 'r') as read\_file:  
 data = json.load(read\_file)  
 self.tokens.update(data)  
 if token\_class == 'C':  
 for k in data.keys():  
 data[k] = re.sub(r"'([^']\*)'", r'"\1"', data[k])  
 self.tokens.update(data)  
 *# лексемы (значение-код)* inverse\_tokens = {val: key for key, val in self.tokens.items()}  
 replace = {'echo': 'print', '=': '<-', '!=': '!=',  
 '==': '==', '/': '/', '%': '%%','>':'>','>=':'>=','<':'<','<=':'<=',  
 '$':'','.':'paste(','+':'+','-':'-','\*':'\*','\*\*':'\*\*',  
 '^':'bitwXor(','&':'bitwAnd(','|':'bitwOr(','~':'bitwNot(','<<':'bitwShiftL(','>>':'bitwShiftR(',  
 'and':'&&','or':'||','xor':'xor(','!': '!',  
 'true':'TRUE','false':'FALSE','null':'NULL',  
 'do':'repeat','array':'c'  
 }  
 *# файл, содержащий обратную польскую запись* f = open('./files/reverse\_polsk.txt', 'r')  
 inp\_seq = f.read()  
 f.close()  
  
 t = re.findall(r'(?:\'[^\']\*\')|(?:"[^"]\*")|(?:[^ ]+)', inp\_seq)  
  
 def is\_identifier(token):  
 return ((token in inverse\_tokens) and re.match(r'^I\d+$', inverse\_tokens[token]))  
  
 def is\_constant(token):  
 return ((token in inverse\_tokens) and re.match(r'^C\d+$', inverse\_tokens[token])) or (  
 (token in inverse\_tokens) and re.match(r'^N\d+$',inverse\_tokens[token]))  
  
 def is\_operation(token):  
 return (token in inverse\_tokens) and re.match(r'^O\d+$', inverse\_tokens[token])  
  
 i = 0  
 stack = []  
 out\_seq = ''  
 is\_func = False  
 variable = 0  
  
 t=[self.tokens[i] if i in self.tokens.keys() else i for i in t]  
 tub\_num=0  
 markers = []  
  
 print(self.else\_marks)  
 print(self.if\_marks)  
 *# print(self.while\_start\_marks)  
 # print(self.while\_end\_marks)  
 # print(self.do\_start\_marks)  
 # print(self.do\_end\_marks)* while i < len(t):  
 *# print(out\_seq)* print(stack)  
 *# print(markers)  
 # print(t[i])  
 # print('-------')* if is\_func == True and not (is\_identifier(t[i])):  
 out\_seq += '{\n'  
 is\_func = False  
 if is\_identifier(t[i]) or is\_constant(t[i]):  
 if t[i] in inverse\_tokens and re.match(r'^C\d+$', inverse\_tokens[t[i]]):  
 stack.append(f'"{t[i]}"')  
 else:  
 stack.append(replace[t[i]] if t[i] in replace else t[i])  
 elif t[i] == 'НП':  
 stack.pop()  
 stack.pop()  
 func\_name = stack.pop()  
 out\_seq += '\t'\*tub\_num + func\_name.split('(')[0]+'=function('+func\_name.split('(')[1]  
 tub\_num = tub\_num + 1  
 is\_func = True  
 elif t[i] == 'КП':  
 tub\_num-=1  
 out\_seq += '\t'\*tub\_num+ '}\n'  
 elif t[i]=='return':  
 result = stack.pop()  
 out\_seq+='\t'\*tub\_num+'return('+result+');\n'  
 elif t[i] == 'УПЛ':  
 if(t[i-1] in self.while\_end\_marks):  
 arg1 = stack.pop()  
 out\_seq += '\t'\*tub\_num+f'while({arg1})' + '{\n'  
 tub\_num += 1  
 elif(t[i-1] in self.else\_marks):  
 stack.pop()  
 arg1 = stack.pop()  
 out\_seq += '\t'\*tub\_num+f'if ({arg1})' + '{\n'  
 tub\_num += 1  
 elif (t[i + 1] in self.do\_start\_marks):  
 arg1 = stack.pop()  
 out\_seq += '\t'\*tub\_num+f'if ({arg1})' + '{break;}\n}\n'  
 tub\_num -= 1  
 elif t[i] == 'БП':  
 if (t[i - 1] in self.while\_start\_marks):  
 out\_seq += '\t'\*tub\_num+'}\n'  
 tub\_num -= 1  
 elif (t[i - 1] in self.if\_marks):  
 out\_seq += '\t'\*tub\_num+'}\nelse{\n'  
 out\_seq = out\_seq  
 elif t[i] == ':':  
 if(t[i-1] in self.end\_marks):  
 out\_seq += '\t' \* tub\_num + '}\n'  
 tub\_num -= 1  
 elif(t[i-1] in self.if\_marks):  
 out\_seq+='\t'\*tub\_num+'}\n'  
 tub\_num-=1  
 elif(t[i-1] in self.do\_start\_marks):  
 out\_seq+='\t'\*tub\_num+'repeat{\n'  
 tub\_num+=1  
 elif t[i]=='echo':  
 arg1=stack.pop()  
 out\_seq+='\t'\*tub\_num+f'print({arg1});\n'  
 elif is\_operation(t[i]):  
 if t[i]=='$':  
 out\_seq = out\_seq  
 elif t[i]=='.' and len(stack)>=2:  
 arg1=stack.pop()  
 arg2=stack.pop()  
 out\_seq += '\t'\*tub\_num+f'Rp{variable}='+f'paste({arg2},{arg1},sep="");\n'  
 stack.append(f'Rp{variable}')  
 variable+=1  
 elif t[i] == '=' and len(stack) >= 2:  
 arg1 = stack.pop()  
 arg2 = stack.pop()  
 out\_seq += '\t'\*tub\_num+ f'{arg2} <- {arg1};\n'  
 else:  
 operation = replace[t[i]] if t[i] in replace else t[i]  
 arg1 = stack.pop()  
 if t[i] != '!':  
 arg2 = stack.pop()  
 out\_seq+='\t'\*tub\_num+f'Rp{variable}='+f'({arg2} {operation} {arg1});\n'  
 stack.append(f'Rp{variable}')  
 variable+=1  
 else:  
 stack.append(f'({operation}{arg1})')  
 elif re.match("[0-9]+АЭМ",t[i]):  
 k = int(t[i].split('АЭМ')[0])  
 a = []  
 while k != 0:  
 a.append(stack.pop())  
 k -= 1  
 a.reverse()  
 stack.append('\t'\*tub\_num + a[0] + '[' + ','.join(a[1:]) + ']')  
 elif t[i] in ['break', 'continue']:  
 stack.append(replace[t[i]] if t[i] in replace else t[i])  
 arg0 = stack.pop()  
 out\_seq += '\t'\*tub\_num + f'\t{arg0};\n'  
  
 elif re.match(r"[0-9]+Ф",t[i]):  
 k = int(t[i].split('Ф')[0])  
 a = []  
 while k != 0:  
 a.append(stack.pop())  
 k -= 1  
 a.reverse()  
 if(i<len(t)-3 and t[i+3]!='НП'):  
 out\_seq+='\t'\*tub\_num+f"Rp{variable}="+a[0]+'(' + ', '.join(a[1:]) + ');\n'  
 stack.append(f"Rp{variable}")  
 variable+=1  
 else:  
 stack.append(a[0]+'(' + ', '.join(a[1:]) + ')')  
 else:  
 stack.append(t[i])  
 i += 1  
  
 stack.clear()  
 return out\_seq