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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. М. Нагалевский

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](#_Toc104196070)

[2 Базовые понятия 3](#_Toc104196071)

[3 Правила генерации машинного кода 4](#_Toc104196072)

[3 Результаты экспериментов 7](#_Toc104196073)

[Приложение А Описание правил записи элементов заданного выходного языка 9](#_Toc104196074)

[Приложение Б Листинг программы и комментарии к нему 13](#_Toc104196075)

**1 Вариант задания**

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

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 42 | Java | C++ |

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: на рисунке 2 представлен скриншот файла, содержащего текст на входном языке программирования, на рисунке 3 – скриншоты результата работы программы.

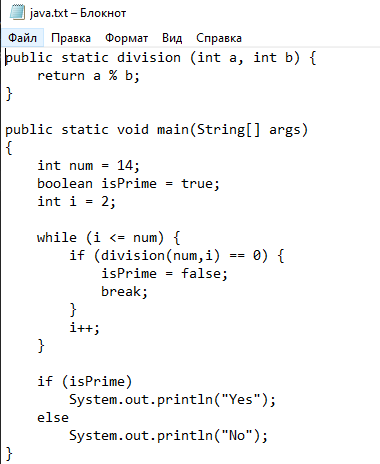


Рисунок 2 – Скриншот файла, содержащего текст на входном языке программирования

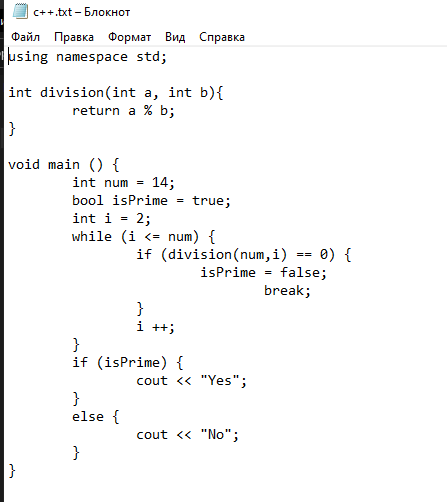


Рисунок 3 – Результат работы программы

**ПРИЛОЖЕНИЕ А   
Описание правил записи элементов заданного выходного языка**

***Язык программирования С++***

**Идентификаторы**

Произвольная последовательность букв и цифр, начинающаяся с буквы. Может включать символы подчеркивания и начинаться с них.

**Числовые константы целого типа**

Произвольная последовательность цифр без знака.

**Числовые константы вещественного типа, представленные с фиксированной точкой**

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

**Числовые константы вещественного типа, представленные с плавающей точкой**

Последовательность, включающая цифры, десятичную точку (необязательную), символ «е» или «Е», а также знак «+» или «-» вида (необязательный):

1.23е-25

1.23Е-25

1.23е+25

1.23Е+25

1.23е2

1.23Е2

**Символьные (строковые) константы**

Символьная константа – один символ, заключенный в апострофы:

‘a’

Строковая константа – последовательность символов, заключенная в кавычки, расположенная в пределах одной строки, вида:

“acb 12\_& ?tu”

**Переменные с индексами (массивы и элементы массивов)**

Идентификатор, после которого в квадратных скобках перечислены выражения-индексы, вида:

Abc[12][I][i-6]

C[1+i]

**Комментарии (строчные и блочные)**

Блочные – последовательность символов, начинающаяся с «/\*» и оканчивающаяся «\*/», возможно содержащая несколько строк:

/\* Это комментарий,

Который содержит 2 строки\*/

Строчные – от символов «//» до конца строки.

i=i+1; // это инкремент

**Обращения к функциям пользователя**

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Скобки могут быть пустыми в случае отсутствия аргументов:

F(12, 4, i)

f(av-6)

g()

**Арифметические операции**

Сложение +

Вычитание -

Умножение \*

Деление /

**Операции сравнения**

Меньше <

Больше >

Равно ==

Не равно !=

Меньше или равно <=

Больше или равно >=

**Оператор присваивания**

Имеет вид «=». Слева стоит идентификатор или элемент массива, а справа – выражение. Заканчивается символом «;», например:

a=b+с;

b[2][i-9]=12;

**Операторы блока**

{ – начало блока

…

} - конец блока

**Структура программы**

Программа начинается операторами описания данных. Затем могут идти описания данных и функций, а затем основная функция программы void main () и ее тело, заключенное в операторы блока, оканчивающееся точкой.

Описания

void main()

{

…

}

**Операторы описания данных (идентификаторов и массивов)**

Начинается с ключевого слова типа и содержит перечисление идентификаторов через запятую. Оканчивается знаком «;»

<тип> <список элементов>;

Типы переменных: int (целый), float (вещественный), char (символьный)

Элементом списка может быть массив, для которого указывается идентификатор и размерности:

int a, b, c;

float d[3][4], c[78];

**Операторы описания функций**

Функции имеют заголовок вида

<тип> <идентификатор> (<список формальных параметров>);

и тело – список операторов, заключенный в операторы блока

{ … };

Например:

int abc (float r) {

float r1, r2;

return sin(r1)/cos(r2)\*tan(r);

}

В теле функции может присутствовать оператор

return (<значение>);

Оператор безусловного перехода и метки

goto <метка>;

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

a: str=’ujhti’;

**Оператор условного перехода**

Начинается с ключевого слова «if», имеет полный и неполный формат:

if (логическое выражение) оператор\_1 else оператор\_2;

if (логическое выражение) оператор\_1;

Вместо отдельных операторов могут использоваться блоки операторов:

if (логическое выражение)

{операторы\_1}

else

{операторы\_2}

**Цикл for**

Цикл for имеет следующее формальное определение:

for (выражение\_1; выражение\_2; выражение\_3) {

// тело цикла

}

выражение\_1 выполняется один раз при начале выполнения цикла и представляет установку начальных условий, как правило, это инициализация счетчиков - специальных переменных, которые используются для контроля за циклом.

выражение\_2 представляет условие, при соблюдении которого выполняется цикл. Как правило, в качестве условия используется операция сравнения, и если она возвращает ненулевое значение (то есть условие истинно), то выполняется тело цикла, а затем вычисляется выражение\_3.

выражение\_3 задает изменение параметров цикла, нередко здесь происходит увеличение счетчиков цикла на единицу.

**Цикл while**

В отличие от цикла do цикл while сразу проверяет истинность некоторого условия, и если условие истинно, то есть не равно 0, то код цикла выполняется:

Using namespace std;

int main(void) {

int i = 6;

while (i > 0) {

cout << i;

i--;

}

return 0;

}

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

**import lab2**

**import json**

**import re**

**CLASSES\_OF\_TOKENS = ['W', 'I', 'O', 'R', 'N', 'C']**

**def is\_identifier(token):**

**return ((token in inverse\_tokens) and re.match(r'^I\d+$', inverse\_tokens[token])) or re.match(r'^M\d+$', token) or token in ['String[]', 'args', 'System.out.println', 'System.out.print', 'int', 'double', 'boolean','float']**

**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])) or token.isdigit() or token in ["in.nextInt()", "division(num,i)","No","Yes"]**

**def is\_operation(token):**

**return (token in inverse\_tokens) and re.match(r'^O\d+$', inverse\_tokens[token])**

**# 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)**

**if token\_class == 'C':**

**for k in data.keys():**

**data[k] = re.sub(r"'([^']\*)'", r'"\1"', data[k])**

**tokens.update(data)**

**# лексемы (значение-код)**

**inverse\_tokens = {val: key for key, val in tokens.items()}**

**replace = {'in.nextInt()': 'cin >> ', 'System.out.println': 'cout << ', 'System.out.print': 'cout << ', 'int': 'int', 'double': 'double','boolean':'bool', '=': '=', '||': '||', '&&': '&&', '!=': '!=', '==': '==', '/': '/', '%': '%', '!': '!', '++': '+= 1'}**

**# файл, содержащий обратную польскую запись**

**f = open('reverse\_polish\_entry.txt', 'r')**

**f.close()**

**t = re.findall(r'(?:\'[^\']\*\')|(?:"[^"]\*")|(?:[^ ]+)', inp\_seq)**

**i = 0**

**stack = []**

**out\_seq = ''**

**is\_func = False**

**variable = {}**

**while i < len(t):**

**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]):**

**stack.append(replace[t[i]] if t[i] in replace else t[i])**

**elif t[i] == 'НП':**

**stack.pop()**

**stack.pop()**

**out\_seq += 'void main ()'**

**is\_func = True**

**elif t[i] == 'КП':**

**out\_seq += '}\n'**

**elif t[i] == 'КО':**

**stack.pop()**

**stack.pop()**

**elif t[i] == 'УПЛ':**

**arg1 = stack.pop()**

**arg2 = stack.pop()**

**out\_seq += f'if (!({arg2})) goto {arg1};\n'**

**elif t[i] == 'БП':**

**arg1 = stack.pop()**

**out\_seq += f'goto {arg1};\n'**

**elif t[i] == ':':**

**arg1 = stack.pop()**

**out\_seq += f'{arg1}: '**

**elif is\_operation(t[i]):**

**if t[i] == '=' and len(stack) == 2:**

**arg1 = stack.pop()**

**arg2 = stack.pop()**

**out\_seq += f'{arg2} = {arg1};\n'**

**elif t[i] == '=' and len(stack) >= 3:**

**arg1 = stack.pop()**

**arg0 = stack.pop()**

**arg2 = stack.pop()**

**out\_seq += f'{arg0} {arg2} = {arg1};\n'**

**elif t[i] in ['+=', '-=', '\*=', '/=', '++']:**

**op = t[i][0]**

**arg1 = stack.pop()**

**arg2 = stack.pop()**

**out\_seq += f'{arg2} {op}= {arg1};\n'**

**else:**

**operation = replace[t[i]] if t[i] in replace else t[i]**

**arg1 = stack.pop()**

**if t[i] != '!':**

**arg2 = stack.pop()**

**stack.append(f'({arg2} {operation} {arg1})')**

**else:**

**stack.append(f'({operation}{arg1})')**

**elif t[i] == 'АЭМ':**

**k = int(stack.pop())**

**a = []**

**while k != 0:**

**a.append(stack.pop())**

**k -= 1**

**a.reverse()**

**out\_seq += 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 += f'\t{arg0};\n'**

**elif t[i] == 'Ф':**

**k = int(stack.pop()) + 1**

**a = []**

**while k != 0:**

**a.append(stack.pop())**

**k -= 1**

**a.reverse()**

**if a[0] == 'cin >>':**

**b = []**

**out\_seq += a[0] + '("' + ' '.join(b) + '", ' + ', '.join(map(lambda x: '&' + x, a[1:])) + ');\n'**

**else:**

**out\_seq += a[0] + ', '.join(a[1:]) + ';\n'**

**i += 1**

**#while**

**out\_seq = re.sub(r'(M\d+): if \(!\((.\*)\)\) goto (M\d+);(?:\n|\n((?:.|\n)+)\n)goto \1;\n\3: ', r'while \2 {\n\4\n}\n', out\_seq)**

**# if else**

**out\_seq = re.sub(r'if\s\*\((.\*)\s\*<\s\*(.\*)\)\s\*{\s\*(.\*?)\s\*}\s\*else\s\*{\s\*(.\*?)\s\*}\s\*', r'if (\1 < \2)\n{\n\3\n}\nelse\n{\n\4\n}\n', out\_seq)**

**#if**

**out\_seq = re.sub(r"if\s\*\(\s\*!\s\*\(\s\*(.\*)\s\*\)\s\*\)\s\*goto\s+(M\d+)\s\*;\s\*(\n(?:.|\n)+?)\s\*\2:\s\*", r"if \1 {\3\n}\n", out\_seq)**

**# Правильная расстановка табуляции**

**def indent\_cpp\_code(code):**

**indented\_code = ""**

**indentation = 0**

**for line in code.split("\n"):**

**if "{" in line:**

**indented\_code += ("\t" \* indentation) + line + "\n"**

**indentation += 1**

**elif "}" in line:**

**indentation -= 1**

**indented\_code += ("\t" \* indentation) + line + "\n"**

**else:**

**indented\_code += ("\t" \* indentation) + line + "\n"**

**return indented\_code**

**stack.clear()**

**# файл, содержащий текст на выходном языке программирования**

**f = open('c++.txt', 'w')**

**f.write(out\_seq)**

**f.close()**