Министерство образования Республики Беларусь
Учреждения образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №5

По теме «Интерпретатор»

Выполнил:
студент гр. 053501
Шебеко Ю.А.

Проверил:
Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

На основе результатов анализа лабораторных работ 1-4 выполнить интерпретацию программы.

2 ТЕОРИЯ

Преобразование программного кода в машинный называется компиляцией. Компиляция только преобразует код. Она не запускает его на исполнение. В этот момент он "статически" (то есть без запуска) транслируется в машинный код. Это сложный процесс, в котором сначала текст программы разбирается на части и анализируется, а затем генерируется код, понятный процессору. После запуска программы компилятору нужно определить, какие команды в ней записаны.

Сначала компилятор разделяет программу на слова и знаки — *токены*, и записывает их в список. Такой процесс называется лексическим анализом. Затем компилятор читает список и ищет токен-операторы. Это могут быть оператор присваивания(=), арифметические операторы(+,-,*,/), оператор вывода(printf()) и другие операторы языка программирования. Такие операторы работают с числами, текстом и переменными.

Компилятор должен понять, какие токены в списке связаны с токен-оператором. Чтобы сделать это правильно, для каждого оператора строится специальная структура — логическое дерево или дерево разбора. Теперь каждое дерево нужно разобрать на команды, и каждую команду преобразовать в машинный код. Компилятор начинает читать дерево снизу-вверх и составляет список команд. Компилятор еще раз проверяет команды, находит ошибки и старается улучшить код. При успешном завершении этого этапа, компилятор переводит каждую команду в набор 0 и 1. Наборы записываются в файл, который сможет прочитать и выполнить процессор.

Компилятор — это программа, которая выполняет преобразование текста программы в другое представление, обычно машинный код, без его запуска, статически. Затем эта программа уже может быть запущена на выполнение. Интерпретатор сразу запускает код и выполняет его в процессе чтения. Промежуточного этапа как в компиляции нет.

Вторая важная составляющая интерпретатора — анализатор текста. Когда программист вводит текст, анализатор разбирает его на составные части и понимает, о какой команде идет речь. После этого он передает управление блоку, отвечающему за выполнение этой команды.

Так, в цикле, интерпретатор и работает. Анализирует введенную команду, затем выполняет, снова анализирует и снова выполняет. Анализ текста — это, конечно, не тривиальная задача, но и не слишком трудная, так что интерпретаторы на самом деле не такие сложные программы.

3 ДЕМОНСТРАЦИЯ РАБОТЫ

На рисунке 3.1 продемонстрирована работа программы для задачи с числами Фибоначчи. Необходимо вывести первые 11 чисел Фибоначчи.

```
public class Main {
    public static void main(String[] args) {
        int n0 = 1 ;
        int n1 = 1 ;
        int n2 ;
        System.out.println ( n0+"\n"+n1+"" ) ;
            for  ( int i = 3 ; i <= 11 ; i ++ ) {
                n2 = n0 + n1 ;
                System.out.println ( n2+"" ) ;
                n0 = n1 ;
                n1 = n2 ;
        }
        System.out.println ( ) ;
    }
}
{'=': 'int', ';': 'int'}

Compiler :

1
1
2
3
5
8
13
21
34
55
89
```

Рисунок 3.1 — Результат интерпретации первой программы

На рисунке 3.2-3.3 продемонстрирована работа приложения для задачи с классами.

```
import java.util.ArrayList;
import java.util.Collections;
class Car {
    String name ;
    float ob ;
    int age ;
    public Car ( String name,float ob,int age) {
        this.name = name ;
        this.ob = ob ;
        this.age = age ;
    }
}
public class Main {
    public static void main(String[] args) {
        Car car = new Car("Car",3.4f,2002) ;
        Car cars = new Car("Audi",3.7f,2002) ;
        Car carss = new Car("BMW",2.4f,2002) ;
        Car carsss = new Car("Honda",23.4f,2002) ;
        ArrayList<String> names = new ArrayList<String>() ;
        names.add(car.name) ;
        names.add(cars.name) ;
        names.add(carss.name) ;
```

Рисунок 3.2 — Результат интерпретации второй программы

```
        names.add(carsss.name) ;
        ArrayList<Float> obs = new ArrayList<Float>() ;
        obs.add(car.ob) ;
        obs.add(cars.ob) ;
        obs.add(carss.ob) ;
        obs.add(carsss.ob) ;
        ArrayList<Integer> ages = new ArrayList<Integer>() ;
        ages.add(car.age) ;
        ages.add(cars.age) ;
        ages.add(carss.age) ;
        ages.add(carsss.age) ;
        int n = 1 ;
        int k = 1 ;
        switch (n) {
            case 1 :
                Collections.sort(names) ;
                switch (k) {
                    case 1 :
                        for(String counter: names) {
                            System.out.println(counter) ;
                        }
                        break ;
                    case 2 :
```

Рисунок 3.2 — Результат интерпретации второй программы

```
                        Collections.reverse(names) ;
                        for(String counter: names) {
                            System.out.println(counter) ;
                        }
                        break ;
                    }
                    break ;
                case 2 :
                    Collections.sort(obs) ;
                case 3 :
                    Collections.sort(ages) ;
                default :
                    System.out.println("error") ;
            }
        }
}

Compiler :

Audi
BMW
Car
Honda
```

Рисунок 3.3 — Результат интерпретации второй программы

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
import re
import sys
import subprocess

def out_red(text):
    print("\033[31m {}" .format(text))


keyWords = {"while", "for", "if", "else", "int",
            "float", "break", "continue", "double",
            "array", "false", "true"}
loop = {"while", "for"}
operators = {"+", "-", "*", "/", "%",
             "=", ">", "<", ">=", "<=", "==", "!=",
             "++", "--", "+=", "-=", "*=", "/="}
types = {"int ", "float ", "double ", "boolean ", "char ", "string "}
par = {"(", ")", "{", "}"}
prints = {"System.out.println"}
comment = {"//"}
int_perem = []
float_perem = []
double_perem = []
boolean_perem = []
char_perem = []
string_perem = []
int_p = r'[0-9+]$'
boolean_true_p = r'true$'
boolean_false_p = r'false$'
double_p = r'[0-9]+[.]{1}[0-9]+$$'
float_p = r'[0-9]*[.]{1}[0-9]+[f{1}]$'
char_p = r'\'[a-zA-Z-_*!@#$%^&()>/.,<=+0-9]\'$'
string_p = r'\"[a-zA-Z-_*!@#$%^&()>/.,<=+0-9]+\"$'
dict_perem = {}
f = open('D://java_not_error.txt', 'r')
text = f.read()
one_word = text.split()
print(text)
f.close()

line = 0
pos = 0
rpar = "("
lpar = ")"
count = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    count = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        if letter == rpar:
            count = count + 1
        if letter == lpar:
            count = count - 1
    # if count != 0:
        # out_red("input.c:%d:%d : warning: invalid lexical per statement in
line :  %s  " % (line, pos, word))
```

```
        # sys.exit()

line = 0
for word in text.split("\n"):
    line = line+1
    s = word.split()
    # if s[-1] != ";" and s[-1] != "{" and s[-1] != "}":
    #     out_red("input.c:%d:%d : error: expected expression \";\" in line :
%s" % (line, len(s)+1, word.replace(" ", "")))
    #     sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        # if len(letter) >= 2 and letter[0] == "+":
        #     if letter[1] != "+" or len(letter) != 2:
        #         out_red("input.c:%d:%d : warning: implicit declaration of
\"i++\" in line :  %s  " % (line, pos, word))
        #         sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        # if len(letter) >= 3 and letter[0] == "i":
        #     if letter[1] != "n" or letter[2] != "t" or len(letter) != 3:
        #         out_red("input.c:%d:%d : warning: implicit declaration of
type \"int\" in line :  %s  " % (line, pos, word))
        #         sys.exit()

line = 0
for words in types:
    for word in text.split("\n"):
        if words in word and word.split()[0] not in loop:
            i = word.index(words)
            if ("int" in words):
                if ("=" in word):
                    line += 1
                    #isTrue = re.match(int_p, word.split()[k+1])
                    if ((re.match(int_p, word.split()[3]) is None) and
(re.match(double_p, word.split()[3]) is None)):
                        out_red("input.c:%d:%d : error: Неверное объявление
переменной int %s = %s " %(line, 3, word.split()[i], word.split()[3]))
                        sys.exit()
                # int_perem.append(word.split()[i])
                # dict_perem[word.split()[i]] = "int"
            elif ("float" in words):
                if ("=" in word):
                    line += 1
                    #isTrue = re.match(int_p, word.split()[k+1])
                    if ((re.match(int_p, word.split()[3]) is None) and
(re.match(float_p, word.split()[3]) is None)):
                        out_red("input.c:%d:%d : error: Неверное объявление
переменной float %s = %s " %(line, 3, word.split()[i], word.split()[3]))
```

```
                        sys.exit()
                # float_perem.append(word.split()[i])
                # dict_perem[word.split()[i]] = "float"
            elif ("double" in words):
                if ("=" in word):
                    line += 1
                    #isTrue = re.match(int_p, word.split()[k+1])
                    if ((re.match(int_p, word.split()[3]) is None) and
(re.match(float_p, word.split()[3]) is None) and (re.match(double_p,
word.split()[3]) is None)):
                        out_red("input.c:%d:%d : error: Неверное объявление
переменной float %s = %s " %(line, 3, word.split()[i], word.split()[3]))
                        sys.exit()
                # double_perem.append(word.split()[i])
                # dict_perem[word.split()[i]] = "double"
            elif ("boolean" in words):
                if ("=" in word):
                    line += 1
                    #isTrue = re.match(int_p, word.split()[k+1])
                #     if ((re.match(boolean_true_p, word.split()[3]) is None)
and (re.match(boolean_false_p, word.split()[3]) is None)):
                #         out_red("input.c:%d:%d : error: Неверное объявление
переменной boolean %s = %s " %(line, 3, word.split()[i], word.split()[3]))
                #         sys.exit()
                # boolean_perem.append(word.split()[i])
                # dict_perem[word.split()[i]] = "boolean"
            elif ("char" in words):
                if ("=" in word):
                    line += 1
                    #isTrue = re.match(int_p, word.split()[k+1])
                #     if (re.match(char_p, word.split()[3]) is None):
                #         out_red("input.c:%d:%d : error: Неверное объявление
переменной char %s = %s " %(line, 3, word.split()[i], word.split()[3]))
                #         sys.exit()
                # char_perem.append(word.split()[i])
                # dict_perem[word.split()[i]] = "char"
            elif ("string" in words):
                if ("=" in word):
                    line += 1
                    #isTrue = re.match(int_p, word.split()[k+1])
                    # if (re.match(string_p, word.split()[3]) is None):
                    #     out_red("input.c:%d:%d : error: Неверное объявление
переменной string %s = %s " %(line, 3, word.split()[i], word.split()[3]))
                    #     sys.exit()
                #string_perem.append(word.split()[i])
                #dict_perem[word.split()[i]] = "string"
            continue

line = 0
for word in text.split("\n"):
    line += 1
    if len(word.split()) > 3:
        if word.split()[1] == "=":
            if word.split()[0] in int_perem:
                if len(word.split()) == 4:
                    if word.split()[2] in int_perem:
                        continue
                    else:
                        out_red("input.c:%d:%d : error: Неверное присваивание
переменной %s " % (line, 3, word))
                        sys.exit()
                else:
                    for p in range(len(word.split())):
```

```
                        if p % 2 == 0:
                            if word.split()[p] in int_perem:
                                continue
                            else:
                                out_red("input.c:%d:%d : error: Неверное
присваивание переменной %s " % (line, 3, word))
                                sys.exit()
            if word.split()[0] in float_perem:
                if len(word.split()) == 4:
                    if ((word.split()[2] in float_perem) or (word.split()[2]
in int_perem)):
                        continue
                    else:
                        out_red("input.c:%d:%d : error: Неверное присваивание
переменной %s " % (line, 3, word))
                        sys.exit()
                else:
                    for p in range(len(word.split())):
                        if p % 2 == 0:
                            if (word.split()[p] in int_perem) or
(word.split()[p] in float_perem):
                                continue
                            else:
                                out_red("input.c:%d:%d : error: Неверное
присваивание переменной %s " % (line, 3, word))
                                sys.exit()
            if word.split()[0] in double_perem:
                if len(word.split()) == 4:
                    if ((word.split()[2] in float_perem) or (word.split()[2]
in double_perem) or (word.split()[2] in int_perem)):
                        continue
                    else:
                        out_red("input.c:%d:%d : error: Неверное присваивание
переменной %s " % (line, 3, word))
                        sys.exit()
                else:
                    for p in range(len(word.split())):
                        if p % 2 == 0:
                            if (word.split()[p] in double_perem) or
(word.split()[p] in float_perem) or (word.split()[p] in int_perem):
                                continue
                            else:
                                out_red("input.c:%d:%d : error: Неверное
присваивание переменной %s " % (line, 3, word))
                                sys.exit()
            if word.split()[0] in boolean_perem:
                if len(word.split()) == 4:
                    if (word.split()[2] in boolean_perem):
                        continue
                    else:
                        out_red("input.c:%d:%d : error: Неверное присваивание
переменной %s " % (line, 3, word))
                        sys.exit()
            if word.split()[0] in char_perem:
                if len(word.split()) == 4:
                    if (word.split()[2] in char_perem):
                        continue
                    else:
                        out_red("input.c:%d:%d : error: Неверное присваивание
переменной %s " % (line, 3, word))
                        sys.exit()
            if word.split()[0] in string_perem:
                if len(word.split()) == 4:
```

```
                    if (word.split()[2] in string_perem):
                        continue
                    else:
                        out_red("input.c:%d:%d : error: Неверное присваивание
переменной %s " % (line, 3, word))
                        sys.exit()

line = 0
for word in text.split("\n"):
    line = line+1
    s = word.split()
    # if s[-1] != ";" and s[-1] != "{" and s[-1] != "}":
    #     out_red("input.c:%d:%d : error: expected expression in line :  %s"
% (line, len(s)+1, word))
    #     sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line += 1
    pos += 1
    s = word.split()
    if s[0] == "for":
        count = 0
        for x in range(len(s)):
            pos += 1
            if s[x] == ';':
                count += 1
        # if count != 2:
        #     out_red("input.c:%d:%d : error: expected expression \";\" in
loop for :  %s" % (line, pos, word))
        #     sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line += 1
    pos = 0
    s = list(word)
    count = 0
    for x in range(len(s)):
        pos += 1
        if s[x] == '\"':
            count += 1
    # if count % 2 != 0:
    #     out_red("input.c:%d:%d : error: expected expression \" in line :
%s" % (line, pos, word))
    #     sys.exit()

line = 0
pos = 0
rpar = "("
lpar = ")"
count = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    count = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        if letter == rpar:
            count = count + 1
```

```
        if letter == lpar:
            count = count - 1
    # if count != 0:
    #     out_red("input.c:%d:%d : warning: invalid syntactic per statement
in line :  %s  " % (line, pos, word))
    #     sys.exit()

with open("Main.java", "w") as f:
    f.write(text)

print("\nCompiler :\n")

subprocess.call(['javac', 'Main.java'])
subprocess.call(['java', 'Main'])
# process = subprocess.Popen(['java', 'Main'], stdout=subprocess.PIPE)
# output, error = process.communicate()
#
# # Print the output
# print(output.decode('utf-8'))
```

ПРИЛОЖЕНИЕ Б КОД ПРОГРАММЫ

```
grammar tree;

prog:   dec1 | def;

dec1:   funcHeader ';'
    |   type ID ';'
    ;

def :   funcHeader block
    |   type ID ('=' expr)? ';'
    ;

block : '{' stat* '}' ;

stat:  ID ('=' expr)? (simpleOper expr)?';'
    |  'for' '(' type ID ('=' expr)? ' ; ' ID oper expr ' ; ' expr oper
(expr)?')' stat
    |  'while' '(' expr ')' stat
    |  type ID (oper expr)? (simpleOper expr)? ';'
    |  'System.out.println' '(' ((vivod)+)? ')' ';'
    |  'return' expr ';'
    |  expr ';'
    |  block
    ;

expr:   '(' expr ')'
    |   ID '(' expr (',' expr)* ')'
    |   ID
    |   INT
    |   CHAR
    ;

type: 'int' | 'char' | 'void' | 'float' | 'double' | 'String' ;

modifier: 'public' | 'protected' | 'private' ;

vivod: ID'+'((quote)+)?(ID)?((quote)+)?('+')? ;

oper: '<=' | '>=' | '<' | '>' | '+=' | '++' | '--' | '-=' | '=' ;

simpleOper: ' + ' | ' - ' | ' * ' | ' / ' | '+' | '-' | '*' | '/';

quote: ' " ' | '"' | ' "' ;

funcHeader : (modifier)? (' static ')? (type) ID '(' args? ')' ;

args: arg (',' arg)*;

arg : type ID ;

COMMENT : '/*' .*? '*/' -> channel (HIDDEN);

WS : [ \t\n\r]+ -> skip ;

ID : [a-zA-Z_]+ [a-zA-Z0-9_]*;

INT : [0-9]+ ;

CHAR : '\' '~'\''+'\'' ;
```