Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Преподаватель _____ А.С. Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ19–04–1М, 031943354 _____ Т.В. Радионов
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2020

1 Цель и задачи

Цель: дизучение методов генерации промежуточного кода с их программной реализацией.

Задачи:

- изучение теоретического материала по организации генерации промежуточного кода компиляторов простых языков программирования;
- составление формального описания синтаксически-управляемого транслятора с действиями по генерации промежуточного кода;
- программная реализация компилятора в промежуточный код по формальному описанию.

2 Описание языка промежуточного кода

В качестве промежуточного кода используется трехадресный код. Трехадресный код – это последовательность операторов вида x := y op z, где x, y и z – имена, константы или сгенерированные компилятором временные объекты. Язык, используемый в качестве промежуточного кода, включает в себя следующие конструкции:

- идентификаторы – имя переменной, состоящее из букв латинского алфавита, цифр и символа нижнего подчеркивания;
- временные переменные, создаваемые компилятором, имеют вид: $E<номер>, где вместо <номер> указывается номер текущей переменной;
- метки (указатели), которые являются уникальными числами (позициями при считывании кода);
- оператор goto для перехода по меткам;
- ifFalse и ifTrue для обозначения ложного и истинного условия.

Для генерации промежуточного кода функций, операторов условия и цикла был добавлен класс IntermediateCode.java и вызов его методов непосредственно из класса Parser.java во время его выполнения. Организация хранения меток – стековая.

3 Отслеживаемые ошибки

Лексические – LexicalException. Ошибки написания языковых конструкций.

Синтаксические – SyntaxException. Ошибки последовательности языковых конструкций.

Семантические – SemanticException. Логические ошибки языковых конструкций.

4 Описание семантики основных конструкций

Функции: являются именованным контейнером, в которых объявляются операторы или не объявляется ничего, также может вызываться внутренняя функция. Функция объявляется с ключевого слова “def”, далее задается название функции (именование соответствует правилам именования переменных), затем в скобках перечисляются аргументы функции, далее в фигурных скобках описываются операторы.

Операторы: логические конструкции, использующиеся для действий с выражениями и также вызывающие функции. Есть несколько типов: условные (if…else), цикличные (do…while), возвращаемые (return), выводящие (print), присваивающие (=), вызывающие функции (имя функции и передаваемые аргументы).

Выражения: представляют из себя арифметические и логические операции, а также операции над строками.

Приоритеты функций, операций и выражений представлены в таблице 1 в порядке возрастания сверху-вниз и слева-направо.

Таблица 1 – Формальное описание языка CatCode

Порождающее	Порождаемое
F	def VAR (VAR*) { S* }
S	VAR=E | VAR(VAR*) | print (E) | if (E) { S+ } | if (E) { S+ } else { S+ } | while (E) { S+ } | do { S+ } while (E) | return E
E	E||E | E&&E | E>E | E<E | E>=E| E<=E | E==E| E+E | E-E | E*E | E/E | (E) | INT | BOOL | STRING | VAR
VAR	[a-zA-Z]+[0-9]*
INT	[0-9]+
BOOL	true | false
STRING	“[^“]*”

5 Исходный текст компилятора

```
package app.classes;

import java.util.ArrayList;
import java.util.Map;

public class IntermediateCode {
    // Свойства
    private static String iCode = "";
    private static Boolean stop = false;
    // Поля
    private static String tab1 = "  ";

    /**
     * Получить промежуточный код
```

```
     *
     * @return код (string)
     */
    public static String getICode() {
        return iCode;
    }

    /**
     * Остановить запись промежуточного кода
     * @param stop логическая команда
     */
    public static void setStop(Boolean stop) {
        IntermediateCode.stop = stop;
    }

    /**
     * Сбросить промежуточный код
     */
    public static void resetICode() {
        iCode = "";
    }

    /**
     * Задать промежуточный код функции - начало
     * @param funcVar имя функции
     * @param funcArgs аргументы функции
     */
    public static void setFunction_Start(String funcVar, Map<String, Expression> fu
ncArgs) {
        iCode += funcVar + ":\n";
        for (String a : funcArgs.keySet()) {
            iCode += tab1 + "pop " + a + "\n";
        }
    }
    /**
     * Задать промежуточный код функции - конец
     */
    public static void setFunction_End() {
        iCode += "return" + "\n";
    }

    /**
     * Задать промежуточный код функции - вызов
     * @param funcVar имя функции
     * @param funcArgs аргументы функции
     */
    public static void setFunction_Call(String funcVar, Map<String, Expression> fun
cArgs) {
        ArrayList<Expression> funcArgsExprList = new ArrayList<Expression>();
```

```
        for (Expression e : funcArgs.values()) {
            funcArgsExprList.add(0, e);
        }
        for (Expression e : funcArgsExprList) {
            setExpression(e);
            iCode += tab1 + "push $" + e.getName() + "\n";
        }
        iCode += tab1 + "call " + funcVar + " " + funcArgs.size() + "\n";
    }

    /**
     * Задать промежуточный код присвоения
     * @param var переменная
     * @param expr присваиваемое выражение
     */
    public static void setAssign(String var, Expression expr) {
        if (stop)
            return;
        for (Expression e : expr.getExpressions()) {
            setExpression(e);
        }
        iCode += tab1 + expr.getICode() + "\n";
        iCode += tab1 + var + "=$" + expr.getName() + "\n";
    }

    /**
     * Задать промежуточный код операции
     */
    public static void setOperation(String operation, Expression expr) {
        if (stop)
            return;
        for (Expression e : expr.getExpressions()) {
            setExpression(e);
        }
        iCode += tab1 + operation + " $" + expr.getName() + "\n";
    }

    /**
     * Задать промежуточный код условия "Если" - начало
     * @param expr выражение условия
     * @param pointer указатель
     */
    public static void setCondition_IfStart(Expression expr, int pointer) {
        if (stop)
            return;
        for (Expression e : expr.getExpressions()) {
            setExpression(e);
        }
        iCode += tab1 + expr.getICode() + "\n";
```

```
        iCode += tab1 + "ifFalse $" + expr.getName() + " goto " + pointer + "\n";
    }

    /**
     * Задать промежуточный код условия "Если" - конец
     * @param pointer указатель
     */
    public static void setCondition_IfEnd(int pointer) {
        iCode += tab1 + "goto " + pointer + "\n";
    }

    /**
     * Задать промежуточный код условия "Иначе" - начало
     * @param pointer указатель
     */
    public static void setCondition_ElseStart(int pointer) {
        iCode += pointer + ":\n";
    }

    /**
     * Задать промежуточный код условия "Иначе" - конец
     * @param pointer указатель
     */
    public static void setCondition_ElseEnd(int pointer) {
        iCode += pointer + ":\n";
    }

    /**
     * Задать промежуточный код условия "Иначе" - пропустить
     */
    public static void setCondition_ElsePass(int pointer) {
        iCode += pointer + ":\n";
        iCode += tab1 + "goto " + (pointer - 1) + "\n";
        iCode += (pointer - 1) + ":\n";
    }

    /**
     * Задать промежуточный код цикла
     * @param expr выражение цикла
     */
    public static void setCycle(Expression expr) {
        if (stop)
            return;
        iCode += tab1 + expr.getICode() + "\n";
    }

    /**
     * Задать промежуточный код выражения
     * @param expr выражение
```

```
     */
    private static void setExpression(Expression expr) {
        iCode += tab1 + expr.getICode() + "\n";
    }
}
```

Пример составления промежуточного кода для выражения:

```
        iCode = "$" + name + "=" + expr; // $E=33
```

6 Тестовые примеры

```
def test00(x y z)
{
    // test function
    x = x + 1
    y = 1
    return y
}
def test0()
{
    x = 2 * 2
    z = 3
    test00(10 9 8)
    return 5 + x
}
```

```
test00:
  pop x
  pop y
  pop z
  $E0=x
  $E3=1
  $E4=$E0+$E3
  x=$E4
  $E5=1
  y=$E5
  return $E5
return
test0:
  $E6=2
  $E7=2
  $E8=$E6*$E7
  x=$E8
  $E9=3
  z=$E9
  $E12=8
  push $E12
  $E11=9
  push $E11
```

```
  $E10=10
  push $E10
  call test00 3
  $E16=5
  $E6=2
  $E7=2
  $E8=$E6*$E7
  return $E17
return

def test1()
{
    x = 2 + 3 * (4 - 5) - (2 + 3)
}

test1:
  $E0=2
  $E1=3
  $E2=4
  $E3=5
  $E4=$E2-$E3
  $E5=E4
  $E6=$E1*$E5
  $E7=$E0+$E6
  $E8=2
  $E9=3
  $E10=$E8+$E9
  $E11=E10
  $E12=$E7-$E11
  x=$E12
return

def test2()
{
    a = (2 > 3) && (3 < 1) || true || !false
}

test2:
  $E0=2
  $E1=3
  $E2=$E0>$E1
  $E3=E2
  $E4=3
  $E5=1
  $E6=$E4<$E5
  $E7=E6
  $E8=$E3&&$E7
  $E9=true
  $E10=$E8||$E9
```

```
  $E11=false
  $E12=!E11
  $E13=$E10||$E12
  a=$E13
return

def test3()
{
    x = (2 + 2) * 2
    y = 7
    if (x == y)
    {
        if (2 * 4 > y)
        {
            print("y < 8")
            y = y + 1
        }
        else
        {
            x = -5
        }
    }
}

test3:
  $E0=2
  $E1=2
  $E2=$E0+$E1
  $E3=E2
  $E4=2
  $E5=$E3*$E4
  x=$E5
  $E6=7
  y=$E6
  $E0=2
  $E1=2
  $E2=$E0+$E1
  $E3=E2
  $E4=2
  $E5=$E3*$E4
  $E6=7
  $E7=$E5==$E6
  ifFalse $E7 goto 18
  $E8=2
  $E9=4
  $E10=$E8*$E9
  $E6=7
  $E11=$E10>$E6
  ifFalse $E11 goto 25
```

```
  print $E12
  $E6=7
  $E13=1
  $E14=$E6+$E13
  y=$E14
  goto 26
25:
  $E15=5
  $E16=-E15
  x=$E16
26:
  goto 19
19:
  goto 18
18:
return


def test4()
{
    w = 1
    while (w < 3)
    {
        w = w + 1
    }
}

test4:
  $E0=1
  w=$E0
  $E2=$E0<$E1
  $E0=1
  $E3=1
  $E4=$E0+$E3
  w=$E4
  $E0=1
  $E3=1
  $E4=$E0+$E3
  $E7=1
  $E8=$E4+$E7
  w=$E8
  $E0=1
  $E3=1
  $E4=$E0+$E3
  $E7=1
  $E8=$E4+$E7
  $E11=1
  $E12=$E8+$E11
  w=$E12
```

```
return

def test5()
{
    y = 3
    do
    {
        y = y - 1
    } while (y > 1)
}

test5:
  $E0=3
  y=$E0
  $E0=3
  $E1=1
  $E2=$E0-$E1
  y=$E2
  $E4=$E2>$E3
  $E0=3
  $E1=1
  $E2=$E0-$E1
  $E5=1
  $E6=$E2-$E5
  y=$E6
return

def test6() {

}

test6:
return

def test8()
{
    x = 1
}

test8:
  $E0=1
  x=$E0
return

def test9()
{
    if(true)
    {
        x = 1
```

```
    }
}

test9:
  $E0=true
  ifFalse $E0 goto 6
  $E1=1
  x=$E1
  goto 7
7:
  goto 6
6:
return

def test10()
{
    x = 0
    if(x > 1)
    {
        x = x + 1
    }
}

test10:
  $E0=0
  x=$E0
  $E0=0
  $E1=1
  $E2=$E0>$E1
  ifFalse $E2 goto 9
  $E0=0
  $E3=1
  $E4=$E0+$E3
  x=$E4
  goto 10
10:
  goto 9
9:
return

def test11()
{
    x = 5
    if (x - 3 == 2)
    {
        print("x = 2")
    }
}
```

```
test11:
  $E0=5
  x=$E0
  $E0=5
  $E1=3
  $E2=$E0-$E1
  $E3=2
  $E4=$E2==$E3
  ifFalse $E4 goto 9
  print $E5
  goto 10
10:
  goto 9
9:
return

def test12()
{
    x = true
    if (x)
    {
        x = false
    }
    else
    {
        x = true
    }
}

test12:
  $E0=true
  x=$E0
  $E0=true
  ifFalse $E0 goto 9
  $E1=false
  x=$E1
  goto 10
9:
  $E2=true
  x=$E2
10:
return

def test13()
{
    x = 5 / 1
}

test13:
```

```
  $E0=5
  $E1=1
  $E2=$E0/$E1
  x=$E2
return

def test13()
{
    x = 5 / 1
}

test13:
  $E0=5
  $E1=1
  $E2=$E0/$E1
  x=$E2
return

def test14()
{
    x = true
    x = false
}

test14:
  $E0=true
  x=$E0
  $E1=false
  x=$E1
return
```