

Práctica 3

compilador de lenguaje imperativo (curso 2016-2017)

Puri Arenas / Fernando Rosa
Facultad de Informática
UCM

[1]

Sintaxis de los programas imperativos

- Admiten **asignación**, **while**, e **if-then** como instrucciones.
- Asignación:
 - `Variable = Term` (Term es una variable o un número)
 - `Variable = Term ArithmeticOper Term` (ArithmeticOper = {+, -, *, /})
- While:

```
while BooleandCond    // cada instrucción en una línea
    instruccion_1
    ...
    instruccion_n
endwhile
```
- If-Then:

```
if BooleandCond        // cada instrucción en una línea
    instruccion_1
    ...
    instruccion_n
endif
```

Muy importantes los blancos!!!!

[3]

Descripción de la Práctica 3

- Implementación de un compilador para la TPMV, que permite compilar programas imperativos a código de bytes.
- Los programas se leen de ficheros de texto.
- El tratamiento de errores se hace utilizando excepciones.

Los comandos disponibles en esta práctica son:

-HALT, QUIT, RUN y REPLACE N: Como en la práctica 2.

-LOAD FICH: Que carga un programa almacenado en el fichero FICH y lo almacena en un programa fuente.

-COMPILE: Compila el programa fuente generando el bytecode asociado, y posteriormente lo ejecuta a través de la TPMV.

[2]

Ejemplos

- Factorial de 5

```
x = 5
r = 1
while 0 < x
    r = r * x
    x = x - 1
endwhile
write r
return
end
```

- $5! + 4! + 3! + 2! + 1!$

```
x = 5
r = 0
while 0 < x
    y = x
    s = 1
    while 0 < y
        s = s * y
        y = y - 1
    endwhile
    write s
    r = r + s
    x = x - 1
endwhile
return
end
```

[4]

Clase SourceProgram

```
public class SourceProgram {  
    private String[] sProgram;  
    ...  
}
```

(1) Programas fuentes. Una línea del programa fuente en cada posición del array

[5]

Clase ParsedProgram

```
public class ParsedProgram {  
    private Instruction[] pProgram;  
    ...  
}
```

(1) Programas parseados. Mismo papel que ByteCodeProgram respecto de los ByteCode

[6]

Interfaz Instruction

```
public interface Instruction {  
    Instruction lexParse(String[] words, LexicalParser lexParser);  
    void compile(Compiler compiler) throws ArrayException;  
}
```

(1) Implementado por las clases que definen cada uno de los tipos de instrucciones: SimpleAssignement, CompoundAssignment, IfThen, While, Return, Write

(2) Al igual que para los comandos o bytecodes, hace falta la clase InstructionParser, que se implementa de forma similar.

[7]

Clase Engine

```
public class Engine {  
    private SourceProgram sProgram;  
    private ParsedProgram pProgram;  
    private ByteCodeProgram bcProgram;  
    ...  
    public void compile() throws ... {  
        try {  
            this.lexicalAnalysis();  
            this.generateByteCode();  
        }  
        catch .....  
    }  
    private void lexicalAnalysis() throws LexicalAnalysisException {...}  
    private void generateByteCode() throws ArrayException {...}  
    ...  
}
```

[8]

ANÁLISIS LÉXICO

[9]

Análisis Léxico

Se encarga la clase LexicalParser, a través de su método

```
public void lexicalParser(ParsedProgram pProgram, String stopKey)...
```

que modifica su parámetro pProgram para que contenga el programa parseado (que acaba en stopKey)

LexicalParser tiene atributos:

- SourceProgram sProgram: programa fuente que tiene que parsear
- int programCounter: posición del programa fuente a partir de la cual tiene que parsear

```
sProgram[0] = "x = 5"
sProgram[1] = "r = 1"
sProgram[2] = "while 0 < x"
sProgram[3] = "    r = r * x"
sProgram[4] = "    x = x - 1"
sProgram[5] = "endwhile"
sProgram[6] = "write r"
sProgram[7] = "return"
sProgram[8] = "end"
```

Ejemplo de uso:

```
ParsedProgram wBody = new ParsedProgram();
//suponiendo lexParser.programCounter == 3
lexParser.lexicalParser(wBody, "ENDWHILE");

//wBody contiene las dos asignaciones simples
del cuerpo del while

// ¿Y dónde deja this.programCounter?
```

[10]

Clase LexicalParser

```
public class LexicalParser {

    private SourceProgram sProgram;
    private int programCounter;

    public void lexicalParser(ParsedProgram pProgram, String stopKey)
        throws LexicalAnalysisException {
        .....
        while (this.programCounter < sProgram.getNumeroInstrucciones() && ...) {
            String instr = sProgram.getInstruction(this.programCounter);
            if (instr.equalsIgnoreCase(stopKey)) {
                stop = true;
            }
            else {
                Instruction instruction = ParserInstruction.parse(instr, this);
                //Añadir instruccion a pProgram
            }
        }
        .....
    }

    public void increaseProgramCounter(){
        this.programCounter++;
    }
    .....
}
```

[11]

Interfaz Instruction

```
public interface Instruction {

    Instruction lexParse(String[] words, LexicalParser lexParser);
    void compile(Compiler compiler) throws ArrayException;

}
```

- (1) Implementado por las clases que definen cada uno de los tipos de instrucciones: SimpleAssignment, CompoundAssignment, IfThen, While, Return, Write
- (2) Al igual que para los comandos o bytecodes, hace falta la clase InstructionParser, que se implementa de forma similar.

[12]

Interfaz Term

```
public interface Term {  
    Term parse(String term);  
    ByteCode compile(Compiler compiler);  
}
```

(1) Implementado por las clases

- Variable
- Number

(2) Al igual que para los comandos o bytecodes, hace falta la clase TermParser, que se implementa de forma similar.

[13]

Class Variable

```
public class Variable implements Term {  
    private String varName;  
  
    @Override  
    public Term parse(String term) {  
        if (term.length!=1) return null;  
        else {  
            char name = term.charAt(0);  
            if ('a' <= name && name <= 'z') return new Variable(term);  
            else return null;  
        }  
  
    @Override  
    public ByteCode compile(Compiler compiler){ ... }  
  
    public String toString(){  
        return this.varName;  
    }  
  
}
```

[14]

Class SimpleAssignment

```
public class SimpleAssignment implements Instruction {  
    private String varName;  
    private Term rhs;  
  
    @Override  
    public Instruction lexParse(String[] words, LexicalParser lexer){  
        if (words.length != 3) return null;  
        else {  
            .....  
            Term term = TermParser.parse(words[2]);  
            lexer.increaseProgramCounter();  
            .....  
        }  
  
    @Override  
    public void compile(Compiler compiler){ ... }  
  
}
```

[15]

Class CompoundAssignment

```
public class CompoundAssignment implements Instruction {  
    private String varName;  
    private String operator;  
    private Term t1;  
    private Term t2;  
  
    @Override  
    public Instruction lexParse(String[] words, LexicalParser lexer){  
        .....  
    }  
  
    @Override  
    public void compile(Compiler compiler){ ... }  
  
}
```

[16]

Clase abstracta Condition

```
public abstract class Condition {  
  
    private Term t1;  
    private Term t2;  
    private ConditionalJumps condition; //para la compilación  
  
    public Condition parse(String t1, String op, String t2,  
                           LexicalParser parser){...}  
  
    public void compile(Compiler compiler) throws ArrayException{...}  
  
    //Otros métodos abstractos  
}
```

- (1) De Condition heredan los 4 tipos de condiciones: Equal, Less, LessEq, NotEqual.
- (2) Al igual que para los comandos o bytecodes, hace falta la clase ConditionParser, que se implementa de forma similar.

[17]

Clase While

```
public class While implements Instruction {  
  
    private Condition condition;  
    private ParsedProgram whileBody;  
  
    public Instruction lexParse(String[] words, LexicalParser lexer){  
        ...  
        ParsedProgram wBody = new ParsedProgram();  
        lexer.lexicalParser(wBody, "ENDWHILE");  
        lexer.increaseProgramCounter();  
        ...  
    }  
  
    public void compile(Compiler compiler) throws ArrayException{...}  
  
}
```

[19]

GENERACIÓN DEL BYTECODE

Generación del ByteCode

Se encarga la clase Compiler, a través de su método

```
public void compile(ParsedProgram pProgram) throws ...
```

que añade a su atributo `this.bcProgram` la compilación del programa pProgram

Más concretamente, Compiler necesita atributos:

- String[] `varTable`: tabla de variables (posición que ocupa cada variable en memoria)
- int `numVars`: número de variables almacenadas en `this.varTable`
- ByteCodeProgram `bcProgram`: ByteCodeProgram generado por el método compile

[20]

[21]

Clase Compiler

```
public class Compiler {  
  
    private ByteCodeProgram bcProgram;  
    private String[] varTable;  
    private int numVars;  
  
    public void compile(ParsedProgram pProgram) throws ... {  
        int i = 0;  
        try {  
            while (i < pProgram.getNumeroInstrucciones()){  
                Instruction inst = pProgram.getInstruction(i);  
                instr.compile(this);  
                i++;  
            }  
        }  
        catch ...  
    }  
  
    public void addByteCode(ByteCode b) throws ... {...}  
  
    public int getIndex(String varName) {...}  
}
```

[22]

Clase Write

```
public class Write implements Instruction {  
  
    private String varName;  
  
    public Instruction lexParse(String[] words, LexicalParser lexParser){  
        .....  
    }  
  
    public void compile(Compiler compiler) throws ArrayException{  
        int index = compiler.getIndex(this.varName);  
        compiler.addByteCode(new Load(index));  
        compiler.addByteCode(new Out());  
    }  
}
```

[23]

Clase IfThen

```
public class IfThen implements Instruction {  
  
    private Condition condition;  
    private ParsedProgram ifBody;  
  
    public Instruction lexParse(String[] words, LexicalParser lexParser){  
        .....  
    }  
  
    public void compile(Compiler compiler) throws ArrayException{  
        this.condition.compile(compiler);  
        compiler.compile(this.ifBody);  
        this.condition.setJump(compiler.getProgramCounter());  
    }  
}
```

[24]

Excepciones

* **ArrayException**: posiciones incorrectas de un array (p. ej., al añadir una instrucción a un ParsedProgram "lleno")

* **BadFormatByteCode**: sintaxis incorrecta en bytecodes

* **ExecutionError**: errores al ejecutar un programa de bytecodes

* **DivisionByZero**: error de ejecución al dividir por cero

* **StackException**: error de ejecución al superar el tamaño permitido para la pila

* **LexicalAnalysisException**: producida al parsear programas fuentes incorrectos

[25]