CSM2010 – Compiling Techniques
Course Assignment 2009 – 2010

Department of Computer Science. University of MALTA.

Sandro Spina / Gordon Mangion

This is the description for the assignment of unit CSA2010, Compiling Techniques. This assignment is worth 15% of the total mark for this unit. The assignment has to be carried out on an individual basis. Under no circumstances should code be shared among students. Please remember that plagiarism will not be tolerated; the final submission must be entirely your work.

Deliverables

You will submit your project source code, executables and a PDF file containing the project documentation/report on optical medium. Note that assignment binaries must run straight from CD/DVD-ROM without the need for any installation unless specified in the accompanying document, in which case a detailed installation guide must also be provided.

Description

In this assignment you are to develop a compiler which will translate source files of a simple language called VSL (Very Simple Language) into Java programs. Below is the definition in Extended Backus-Naur Form (EBNF) of the VSL language. The starting point of the grammar is the "*program*" non-terminal at the bottom of the definition.

```
Letter              ::=   ["a"-"z" "A"-"z"]
Digit               ::=   ["0"-"9"]
Type                ::=   "Numeric" | "Boolean"
Literal             ::=   BooleanLiteral | NumericLiteral |
                          StringLiteral
BooleanLiteral      ::=   "true" | "false"
NumericLiteral      ::=   { ["0"-"9"] } [ "." ["0"-"9"]+ ]
StringLiteral       ::=   """ (~["","\r","\n"])* """
Identifier          ::=   ( _ | Letter ) { _ | Letter | Digit }
RelationOp          ::=   "<" | ">" | "==" | "!=" | "<=" | ">="
AdditiveOp          ::=   "+" | "-" | "or"
MultiplicativeOp    ::=   "*" | "/" | "and"
AssignmentOp        ::=   "="
```

```
Factor                  ::=     "(" Expression ")"
                                | "(" Type ")" Expression
                                | Literal
                                | Identifier

Term                    ::=     Factor [ MultiplicativeOp Factor ]

SimpleExpression        ::=     Term { AdditiveOp Term }

Expression              ::=     SimpleExpression RelationOp
                                SimpleExpression

DeclarationStatement    ::=     "var" Identifier ":" Type [ "=" Literal ]
                                ";"

AssignmentStatement     ::=     Identifier AssignmentOp Expression ";"

IfStatement             ::=     "if" "(" Expression ")" StatementBlock [
                                "else" StatementBlock ]

LabelStatement          ::=     Identifier ":"

GotoStatement           ::=     "goto" Identifier ";"

ReadStatement           ::=     "read" Identifier ";"

WriteStatement          ::=     "write" [StringLiteral] [Identifier] ";"

HaltStatement           ::=     "halt" ";"

Statement                       AssignmentStatement
                                | LabelStatement
                                | DeclarationStatement
                                | IfStatement
                                | GotoStatement
                                | ReadStatement
                                | WriteStatement
                                | HaltStatement

StatementBlock          ::=     "{" { Statement } "}"

Header                  ::=     "Script" StringLiteral ";" [ "Author"
                                StringLiteral ";" ]

Program                 ::=     Header StatementBlock
```

Task Breakdown

The assignment is broken down into four tasks. Below is a description of each task accompanied with the assigned mark.

Task 1 - Create Javacc grammar file

In this first task you are to create the Javacc grammar file for the VSL definition given above. You are free to modify the production rules as long as the changes are documented in the report and that the source language (VSL) remains unaltered. Should you prefer to use an alternative to JJTree pre-processor in order to build the parse tree please go ahead.

[Marks: 25%]

Task 2 - Parse Tree Generation

You should enhance the parser developed in Task 1 to output a textual (or graphical if you prefer) representation of the generated parse tree.

[Marks: 10%]

Task 3 - Semantic Analysis and Java class Generation

In this task you are to use the visitor design pattern (or any method you deem suitable) to traverse the parse tree to perform type checking and java code generation.

[Marks: 20%]

Task 4 Sample programs

Together with the above, you are to design and implement short sample source programs to test the outcome of your compiler. In your report, state what you are testing for, insert the programs' parse tree, the resulting Java code and the outcome of your test.

[Marks: 20%]

The Report

In addition to the source and class files, you are to deliver a report. In your report include any deviations from the original EBNF, the salient points on how you developed the compiler (and reasons behind any decisions you took) including semantic rules and code generation, and any sample VSL programs you developed for testing.

[Marks: 25%]

Final Notes

As an example, the VSL source script below:

```
Script "SampleProgram";
Author "Gordon Mangion";
{
	var n : number = 0;
	write "Please enter a number:";
	read n;

	if( n > 10 )
	{
		write "Your number is greater than 10";
	}
	else
	{
		write "Your number is less than or equal to 10";
	}
	halt;
}
```

Should generate a Java class along the lines of the code below:

```
//Generated Java Class for VSL script!
//Generated by VSLJavaGen Visitor!
public class SampleProgram
{
public static void main(String[] args)
{double n = 0;
System.out.print("Please enter a number:");
n = Double.parseDouble(((new java.io.BufferedReader(new
java.io.InputStreamReader(System.in))).readLine()));
if (n>10)
{System.out.print("Your number is greater than 10");
}
else
{
System.out.print("Your number is less than or equal to 10");
}
System.exit(0);
}
}
```

Remember that we are not after pretty printed versions of the code so do not waste time on the production of indented code. As an aid in code generation, below is a table of equivalent-construct mapping between the source and target languages (please feel free to make use of others if you prefer):

VSL	Java
script “Program1”	public class Program1 ...
var n : number;	double n;
Halt;	System.exit(0);
goto label1;	break label1;
read n; //n being a number	n = Double.parseDouble(((new java.io.BufferedReader(new java.io.InputStreamReader(System.in))).readLine()));
write “Text Here”;	System.out.print(“Test Here”);
and	&&
or	||