CS335[Introduction to Compiler Design]
Specifications for designing the Oberon Compiler

Group Number : 14

Mohd. Dawood 10404 Mridul Verma 10415 Rabi Shanker Guha 10551 Shikhar Sharma 10682

Professor : Dr. Sanjeev K Aggarwal
Mentor : Ajay Kumar

January 19, 2013

Contents

1 Project Description

We will be developing a compiler for the source language Oberon.

1.1 Language

The implementation would be done in C using Lex and Yacc. The target language would be MIPS Assembly code.

1.2 Implementation

There are four different phases of a compiler(apart from the code optimization part) :

1.2.1 Lexical Analysis or Scanning

This phase reads the source code and creates meaningful sequences called lexemes. Then each lexeme is used to produce a token of the form

<token-name, attribute-value>

to pass it on for syntax analysis. Token-name is an abstract symbol used in syntax analysis while attribute-value points to an entry in symbol table for this token.

1.2.2 Syntax Analysis or Parsing

In this stage a syntax tree is constructed using the tokens obtained from lexical analyzer. In this tree, each internal node represents an operation and the children of the node represent the arguements of the operation. The subsequent phases use this structure to analyze the source program and generate the target program.

1.2.3 Semantic Analysis

This phase checks the source code for semantic consistency using the syntax tree and information in symbol table. One of the important part of semantic analysis is type checking.

1.2.4 Intermediate Code Generation

In the overall processs of translating source code into target program, one or more intermediate representations maybe generated.After syntax and semantic analysis of the source code, a low-level or machine-like intermediate code is generated. It should be easy to produce and easy to translate into the target machine.

1.2.5 Code Generation

In this phase, the intermediate representation generated in last step is mapped to target language. Since the target language is a machine code in our case, we will have to select registers or memory locations for each variable in the program. We then translate the intermediate instructions into a sequence of machine instructions that perform the same task.

2 Basic Things to be Implemented

2.1 Identifiers

Identifiers are combination of letters and digits such that the first character is always a letter.

_ID :- [a-zA-Z][a-zA-Z0-9]*_

2.2 Keywords

Keywords are identidiers with a predefined meaning.Following is a list of some examples of keywords in Oberon.

1. ABS
2. ASR
3. ASSERT
4. BOOLEAN
5. CHAR
6. CHR
7. COPY
8. DEC
9. EXCL
10. FLOOR
11. FLT
12. INC
13. INCL
14. INTEGER
15. LEN
16. LSL
17. LONG
18. LONGREAL
19. NEW
20. ODD
21. ORD
22. PACK
23. REAL
24. ROR
25. SET
26. SHORT
27. UNPK

2.3 Variables

Variable is an identifier containing known or unknown values. During lexical analysis, variables have to be sorted out according to the types they belong to.

2.4 Comments

Comments are arbitrary character sequences inserted netween any two symbols.They do not affect the meaning of the program and are opened and closed by (* and *) respectively. **Nested Comments** can also be implemented by writing comments inside other comments.**Eg:** (*This is a (*nested *) comment *).

2.5 Data Types

Our Compiler will allow the following data types.

2.5.1 Basic Data Types

BOOLEAN	value is TRUE or FALSE	TRUE | FALSE
CHAR	Belongs to a certain character set	[.\n]*
INTEGER	integers	[-0-9][0-9]*
REAL	real numbers	[0-9]+”.”[0-9]*
LONG REAL	real numbers with relatively higher number of digits	[0-9]+”.”[0-9]*

2.5.2 Array Data Type

An array consists of fixed number of elements of same type. Our compiler will support arrays of upto three dimensions of the basic data types described above. Arrays are declared as -:

$$\text{ARRAY } N_0 N_1 N_2......N_k \text{ OF T}$$

By means of lexical and syntax analysis we will make the following sense out of the above declaration : *An array of dimension $N_0 by N_1 by N_2 by......by N_k$ of the type T.*

2.5.3 Records Data Type

Records also consist of fixed number of elements but they may be of different types.In the declaratiion of record we specify the type of each element (field) and an identifier to represent each field. As the element type can be anything, it can be record itself also and hence nested records also exist.

2.5.4 Pointers

Variables of a Pointer Type P is meant to have pointer values, pointing to a certain type T.

PointerType = POINTER TO type.

2.6 Operators

Following is the list of operators that would be implemented.

Relational Operators	Equal	=
	unequal	#
	Less	<
	Less or Equal	<=
	Greater or Equal	>=
	Set Membership	IN
	Type Test	IS
	Relation	
Arithmetic Operators	Sum	+
	Difference	-
	Product	*
	Quotient	/
	Integer Quotient	DIV
	Modulus	MOD
Set Operators	Union	+
	Complement for single set	-
	Difference for more than one set	-
	Intersubsection	*
	Symmetric Set Difference	/
Logical Operators	Logical Disjunction	OR
	Logical Conjunction	&
	Negation	~
Assignment Operator	Assignment	:=

2.7 Loops

Loops are for performing a task repititively.The following loops will be implemented :

2.7.1 repeat-until

Statements are executed until a condition is satisfied.

RepeatStatement = REPEAT StatementSequence UNTIL expression.

2.7.2 for

Statements are repeated iteratively, with three defined conditions,beginning, ending and the control variable condition.

FOR v := beg TO end BY inc DO S END

here the control variable is v and the statements are S.

2.7.3 while

Statements in while loop are executed as long as the boolean guard is true.

```
WhileStatement = WHILE expression DO StatementSequenceEND
          {ELSIF expression DO StatementSequence} END
```

2.8 Control Structures

Control Structures are used to execute some statements under certain specified conditions only.The following will be implemented:

2.8.1 if-else

If-Else is used to execute the enclosed statements only when the boolean expression acting as guard is true.

```
IfStatement = IF expression THEN StatementSequence
    {ELSIF expression THEN StatementSequence}
    [ELSE StatementSequence]
     END
```

2.8.2 case (for char and integer data type only)

In Case statements, the choice of which statement sequence will be executed is dependent on the value of a variable. In our implementation that variable is limited to only integer and char data type.

```
CaseStatement = CASE expression OF case {"|" case} END
case = ValueOfk : CorrespondingStatementSequence
```

2.9 Basic string operations

We will also be implementing some basic string operations such as :

Length	Calculating the length of the string
Concatenation	To concatenate two given strings
String Copy	To copy one string into other
Reverse	To reverse the characters in a given string
Compare	To compare two strings for equality

2.10 Procedures with formal parameters

Procedure declarations consist of procedure heading to specify the procedure identifier, the formal parameters and the result type(if there is any) and a procedure body that contains declarations and statements.Formal Parameters are identifiers to denote the actual parameters in a procedure declaration. Formal parameters are local to the procedure, i.e. their scope is the procedure declaration.

3 Features to be implemented if time permits

- Modules
  - Module In
  - Module Out
- Code Optimization

4 References

- The Programming Language Oberon, Revision 22.9.2011 by Niklaus Wirth
- Compilers Principles, Tools and Techniques By Aho, Lam, Sethi and Ullman