Project 1: Lexical and Syntax Analyzer

CS215
Shanghai Jiao Tong University

October 12, 2014

1 Introduction

C and some *C-liked* languages are the dominant programming languages. In this project, you are required to design and implement a simplified compiler, including a lexical analyzer and a syntax analyzer, for a given programming language, namely SMALL C , which is a simplified C-liked language containing only the core part of C language. You will learn how to incrementally design and implement the successive phases of the compilation processes using off-the-shelf generators. In this project, Linux environment (e.g., Ubuntu or CentOS) is required.

2 Lexical Analyzer

In this section, you are going to write a lexical analyser. The lexical analyser reads the source codes of SMALL C and separates them into tokens.

2.1 Tokens

INT	⇒	/* integer [1]*/
ID	⇒	/* identifier[2]*/
SEMI	⇒	;
COMMA	⇒	,
BINARYOP	⇒	/* binary operators[3] */

[1] A sequence of digits or digits followed by "0x(0X)" or "0" without spaces. In addition, the value should be in the range of $(-2^{31}, 2^{31})$

[2] A character string consisting of alphabetic characters, digits and the underscore. In addition, digits can't be the first character.

[3] See section 2.2.

UNARYOP	⇒	/* unary operators[4] */
TYPE	⇒	int
LP	⇒	(
RP	⇒	)
LB	⇒	[
RB	⇒	]
LC	⇒	{
RC	⇒	}
STRUCT	⇒	struct
RETURN	⇒	return
IF	⇒	if
ELSE	⇒	else
BREAK	⇒	break
CONT	⇒	continue
FOR	⇒	for

2.2 Operators

Operators in SMALLC are shown below.

Precedence	Operator	Associativity	Description
1	()	Left-to-right	Function call or parenthesis
	[]		Array subscripting
	.		Structure element selection by reference
2	−	Right-to-left	Unary minus
	!		Logical NOT
	++		Prefix increment
	−−		Prefix decrement
	∼		Bit NOT
3	*	Left-to-right	Product
	/		Division
	%		Modulus
4	+		Plus
	−		Binary minus
5	<<		Shift left
	>>		Shift right
6	>		Greater than
	>=		Not less than
	<		Less than
	<=		Not greater than

[4] See section 2.2.

7	==		Equal to
	!=		Not equal to
8	&		Bit AND
9	^		Bit XOR
10	|		Bit OR
11	&&		Logical AND
12	||		Logical OR
13	=	Right-to-left	Assign
	+=		+ and assign
	-=		- and assign
	*=		* and assign
	/=		/ and assign
	&=		& and assign
	^=		^ and assign
	|=		| and assign
	<<=		<< and assign
	>>=		>> and assign

2.3 Flex

Flex is short for *fast lexical analyzer generator*, which is a free version of lex written in C. In this project, you can use flex to generate the lexical analyzer.

Here are some references about Flex:

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, Compilers: Principles, Techniques, and Tools, Second Edition. Chapter 3.5.
- http://en.wikipedia.org/wiki/Flex_lexical_analyser.
- http://flex.sourceforge.net/manual/.

3 Syntax Analyzer

In this step, you are going to perform syntax analysis using Yacc.

3.1 Grammar[5]

[5]Input and output are not included, which will be provided in **??**

PROGRAM	$\rightarrow$	EXTDEFS
EXTDEFS	$\rightarrow$	EXTDEF EXTDEFS
	$\mid$	ϵ
EXTDEF	$\rightarrow$	TYPE EXTVARS SEMI
	$\mid$	STSPEC SEXTVARS SEMI
	$\mid$	TYPE FUNC STMTBLOCK
SEXTVARS	$\rightarrow$	ID
	$\mid$	ID COMMA SEXTVARS
	$\mid$	ϵ
EXTVARS	$\rightarrow$	VAR
	$\mid$	VAR ASSIGN INIT
	$\mid$	VAR COMMA EXTVARS
	$\mid$	VAR ASSIGN INIT COMMA EXTVARS
	$\mid$	ϵ
STSPEC	$\rightarrow$	STRUCT ID LC SDEFS RC
	$\mid$	STRUCT LC SDEFS RC
	$\mid$	STRUCT ID
FUNC	$\rightarrow$	ID LP PARAS RP
PARAS	$\rightarrow$	TYPE ID COMMA PARAS
	$\mid$	TYPE ID
	$\mid$	ϵ
STMTBLOCK	$\rightarrow$	LC DEFS STMTS RC
STMTS	$\rightarrow$	STMT STMTS
	$\mid$	ϵ
STMT	$\rightarrow$	EXP SEMI
	$\mid$	STMTBLOCK
	$\mid$	RETURN EXP SEMI
	$\mid$	IF LP EXP RP STMT
	$\mid$	IF LP EXP RP STMT ELSE STMT
	$\mid$	FOR LP EXP SEMI EXP SEMI EXP RP STMT
	$\mid$	CONT SEMI
	$\mid$	BREAK SEMI
DEFS	$\rightarrow$	TYPE DECS SEMI DEFS
	$\mid$	STSPEC SDECS SEMI DEFS
	$\mid$	ϵ
SDEFS	$\rightarrow$	TYPE SDECS SEMI SDEFS
	$\mid$	ϵ
SDECS	$\rightarrow$	ID COMMA SDECS
	$\mid$	ID
DECS	$\rightarrow$	VAR
	$\mid$	VAR COMMA DECS

	|	VAR ASSIGN INIT COMMA DECS
	|	VAR ASSIGN INIT
VAR	→	ID
	|	VAR LB INT RB
INIT	→	EXP
	|	LC ARGS RC
EXP	→	EXPS
	|	ϵ
EXPS	→	EXPS BINARYOP EXPS
	|	UNARYOP EXPS
	|	LP EXPS RP
	|	ID LP ARGS RP
	|	ID ARRS
	|	ID DOT ID
	|	INT
ARRS	→	LB EXP RB ARRS
	|	ϵ
ARGS	→	EXP COMMA ARGS
	|	EXP

3.2 Notes and Hints

- The meanings of statements in SMALL-C is based on the meanings in *C*.
- The grammar given above may induce one or two reduce-reduce/shift-reduce conflicts, you need to assign precedence of some expressions manually to eliminate these conflicts. For example, "IF LP EXP RP STMT" should have lower precedence than "IF LP EXP RP STMT ELSE STMT".
- A number starts with '0x' or '0X' is a hexadecimal number, while a number starts with '0' is a octal number.
- Only integers and 1-dimensional array can be initialized.
- The dimension of arrays will be no more than 2.
- *Struct* can only contain *int* variables.
- The return type of a function can only be *int*.
- There are no "strange" statements such as $a - - - b$.

3.3 Yacc

Yacc is an LALR parser generator, which stands for *yet another compiler-compiler*. In this project, we can use yacc/bison (bison is another version of yacc) to generate a parser.

Here are some references about Yacc:

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, Compilers: Principles, Techniques, and Tools, Second Edition. Chapter 4.9.
- http://en.wikipedia.org/wiki/Yacc.
- http://www.gnu.org/software/bison/manual/.

Submission Requirements

1. Pack the source files into a file named **StudentID-prj1.tar**. Please DO use "tar" command to compress your source files. Your source files should include:

File name	Description
smallc.l	Lex program
smallc.y	Yacc program
makefile	makefile
StudentID-report.pdf	project report

Table 4: File List.

2. Your analyzer will be tested by the following command:
   ./program "Source file name"
   Your analyzer is expected to read source codes from a source file, and output the lexical and syntax analysis result.

3. Please state clearly the purpose of each program at the head of the source codes, and comment your programs if necessary.

4. Send your **StudentID-prj1.tar** file to cs215.sjtu@gmail.com.

5. Due date: Midnight, Nov. 6, 2014.