

# INFO-F-403 Introduction to Language Theory and Compilation

## Project Part 2: S-COBOL Parsing, Semantic Analysis and Code Generation.

DELHAYE Quentin

December 24, 2013

## 1 Introduction

The aim of this second part was to build a syntax analyzer (parser), semantic analyzer and code generator (to LLVM IR).

Firstly, the grammar had to be made LL(1) compliant, then the parser could be build.

## 2 Grammar

The table 2 present the rules of the grammar.

The following modifications were applied:

- removal of the left recursion in the various rules. Each time a left recursion occurred, the rule has been split as presented in table 1.
- Fusion of the common left parts.
- Transformations to respect the priorities of the operators.
- Addition of the rules 4 and 6 so that **WORD** can be an **INTEGER**.
- Removal of the right recursion in the rules derived from the variable **<EXP\_EQUAL\_LR>**. Indeed, having a language accepting semantics like **a < b > c = d** makes no sense.

A	→	Aa
	becomes	
A	→	aA_LR
A_LR	→	aA_LR
	→	ε

Table 1: Example of the removal of the left recursion in rule A.

[1]	<PROGRAM>	→	<IDENT><ENV><DATA><PROC>
[2]	<IDENT>	→	identification division <END_INST> program-id. ID<END_INST> author. <WORDS><END_INST> date-written. <WORDS><END_INST>
[3]	<WORDS>	→	ID <WORDS_LR>
[4]		→	INTEGER <WORDS_LR>

[5]	<WORDS_LR>	→ ID <WORDS_LR>
[6]		→ INTEGER <WORDS_LR>
[7]		→ $\varepsilon$
[8]	<END_INST>	→ $\cdot \backslash n$
[9]	<ENV>	→ environment division<END_INST> configuration section<END_INST> source-computer. <WORDS><END_INST> object-computer. <WORDS><END_INST>
[10]	<DATA>	→ data division<END_INST> working-storage section<END_INST> <VAR_LIST>
[11]	<VAR_LIST>	→ <VAR_DECL> <VAR_LIST>
[12]		→ $\varepsilon$
[13]	<VAR_DECL>	→ <LEVEL> ID pic IMAGE <VAR_DECL_TAIL>
[14]	<VAR_DECL_TAIL>	→ value INTEGER<END_INST>
[15]		→ <END_INST>
[16]	<LEVEL>	→ INTEGER
[17]	<PROC>	→ procedure division<END_INST> ID section<END_INST> <LABELS> end program ID.
[18]	<LABELS>	→ <LABEL><END_INST> <INSTRUCTION_LIST> <LABELS_LR>
[19]	<LABELS_LR>	→ <LABEL><END_INST> <INSTRUCTION_LIST> <LABELS_LR>
[20]		→ $\varepsilon$
[21]	<LABEL>	→ ID
[22]	<INSTRUCTION_LIST>	→ <INSTRUCTION> <INSTRUCTION_LIST>
[23]		→ $\varepsilon$
[24]	<INSTRUCTION>	→ <ASSIGNATION>
[25]		→ <IF>
[26]		→ <CALL>
[27]		→ <READ>
[28]		→ <WRITE>
[29]		→ stop run<END_INST>
[30]	<ASSIGNATION>	→ move <EXPRESSION> to ID<END_INST>
[31]		→ compute ID = <EXPRESSION><END_INST>
[32]		→ add <EXPRESSION> to ID<END_INST>
[33]		→ subtract <EXPRESSION> from ID<END_INST>
[34]		→ multiply <ASSIGN_END><END_INST>
[35]		→ divide <ASSIGN_END><END_INST>
[36]	<ASSIGN_END>	→ <EXPRESSION>,<EXPRESSION> giving ID
[37]	<EXPRESSION>	→ <EXP_AND> <EXPRESSION_LR>
[38]	<EXPRESSION_LR>	→ or <EXP_AND> <EXPRESSION_LR>
[39]		→ $\varepsilon$
[40]	<EXP_AND>	→ <EXP_EQUAL> <EXP_AND_LR>
[41]	<EXP_AND_LR>	→ and <EXP_EQUAL> <EXP_AND_LR>
[42]		→ $\varepsilon$
[43]	<EXP_EQUAL>	→ <EXP_ADD> <EXP_EQUAL_LR>
[44]	<EXP_EQUAL_LR>	→ = <EXP_ADD>
[45]		→ < <EXP_ADD>

[46]		→	> <EXP_ADD>
[47]		→	<= <EXP_ADD>
[48]		→	>= <EXP_ADD>
[49]		→	$\varepsilon$
[50]	<EXP_ADD>	→	<EXP_MULT> <EXP_ADD_LR>
[51]	<EXP_ADD_LR>	→	+ <EXP_MULT> <EXP_ADD_LR>
[52]		→	- <EXP_MULT> <EXP_ADD_LR>
[53]		→	$\varepsilon$
[54]	<EXP_MULT>	→	<EXP_NOT> <EXP_MULT_LR>
[55]	<EXP_MULT_LR>	→	* <EXP_NOT> <EXP_MULT_LR>
[56]		→	/ <EXP_NOT> <EXP_MULT_LR>
[57]		→	$\varepsilon$
[58]	<EXP_NOT>	→	-<EXP_NOT>
[59]		→	not <EXP_NOT>
[60]		→	<EXP_PARENTHESIS>
[61]	<EXP_PARENTHESIS>	→	(<EXPRESSION>)
[62]		→	<EXP_TERM>
[63]	<EXP_TERM>	→	ID
[64]		→	INTEGER
[65]		→	true
[66]		→	false
[67]	<IF>	→	if <EXPRESSION> then <INSTRUCTION_LIST>
			<IF_END>
[68]	<IF_END>	→	else <INSTRUCTION_LIST> end-if
[69]		→	end-if
[70]	<CALL>	→	perform ID <CALL_TAIL>
[71]	<CALL_TAIL>	→	until <EXPRESSION><END_INST>
[72]		→	<END_INST>
[73]	<READ>	→	accept ID<END_INST>
[74]	<WRITE>	→	display <WRITE_TAIL>
[75]	<WRITE_TAIL>	→	<EXPRESSION><END_INST>
[76]		→	STRING<END_INST>

Table 2: LL(1) grammar of the S-COBOL language.

Variable	First <sup>1</sup>	Follow <sup>1</sup>
<PROGRAM>	identification	
<IDENT>	identification	environment
<WORDS>	ID, INTEGER	.
<WORDS_LR>	ID, INTEGER, $\varepsilon$	.
<END_INST>	.	program-id, date-written, environment, configuration, source-computer, object-computer, data, working-storage, INTEGER, $\varepsilon$ , ID, move, compute, add, subtract, multiply, divide, if, perform, accept, display, stop
<ENV>	environment	data

<DATA>	data	procedure
<VAR_LIST>	INTEGER, $\varepsilon$	procedure
<VAR_DECL>	INTEGER	INTEGER, $\varepsilon$
<VAR_DECL_TAIL>	value, .	INTEGER, $\varepsilon$
<LEVEL>	INTEGER	ID
<PROC>	procedure	
<LABELS>	ID	end
<LABELS_LR>	ID, $\varepsilon$	end
<LABEL>	ID	.
<INSTRUCTION_LIST>	move, compute, add, subtract, multiply, divide, if, perform, accept, display, stop, $\varepsilon$	ID, $\varepsilon$
<INSTRUCTION>	move, compute, add, subtract, multiply, divide, if, perform, accept, display, stop	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$
<ASSIGNATION>	move, compute, add, subtract, multiply, divide	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$
<ASSIGN_END>	-, not, (, ID, INTEGER, true, false	.
<EXPRESSION>	-, not, (, ID, INTEGER, true, false	to, ., from, , , giving, ), then
<EXPRESSION_LR>	or, $\varepsilon$	to, ., from, , , giving, ), then
<EXP_AND>	-, not, (, ID, INTEGER, true, false	or, $\varepsilon$
<EXP_AND_LR>	and, $\varepsilon$	or, $\varepsilon$
<EXP_EQUAL>	-, not, (, ID, INTEGER, true, false	and, $\varepsilon$
<EXP_EQUAL_LR>	=, <, >, <=, >=, $\varepsilon$	and, $\varepsilon$
<EXP_ADD>	-, not, (, ID, INTEGER, true, false	=, <, >, <=, >=, $\varepsilon$
<EXP_ADD_LR>	+, -, $\varepsilon$	=, <, >, <=, >=, $\varepsilon$
<EXP_MULT>	-, not, (, ID, INTEGER, true, false	+, -, $\varepsilon$
<EXP_MULT_LR>	*, /, $\varepsilon$	+, -, $\varepsilon$
<EXP_NOT>	-, not, (, ID, INTEGER, true, false	*, /, $\varepsilon$
<EXP_PARENTHESIS>	(, ID, INTEGER, true, false	*, /, $\varepsilon$
<EXP_TERM>	ID, INTEGER, true, false	*, /, $\varepsilon$
<IF>	if	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$
<IF_END>	else, end-if	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$
<CALL>	perform	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$
<CALL_TAIL>	until, .	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$
<READ>	accept	move, compute, add, subtract, mul- tiply, divide, if, perform, accept, dis- play, stop, $\varepsilon$

<WRITE>	display	move, compute, add, subtract, multiply, divide, if, perform, accept, display, stop, $\varepsilon$
<WRITE_TAIL>	STRING, -, not, (, ID, INTEGER, true, false	move, compute, add, subtract, multiply, divide, if, perform, accept, display, stop, $\varepsilon$

Table 3: First and Follow table.

	identification	ID	.	environment	data	INTEGER	value
<PROGRAM>	1						
<IDENT>	2						
<WORDS>		3					
<WORDS_LR>		4	7				
<END_INST>			8				
<ENV>				9			
<DATA>					10		
<VAR_LIST>					11		12
<VAR_DECL>					11		
<VAR_DECL_TAIL>		15					14
<LEVEL>						16	
<PROC>							
<LABELS>		18					
<LABELS_LR>		19					
<LABEL>		21					
<INSTRUCTION_LIST>		23					
<INSTRUCTION>							
<ASSIGNATION>							
<ASSIGN_END>		36				36	
<EXPRESSION>		37				37	
<EXPRESSION_LR>			39				
<EXP_AND>		40				40	
<EXP_AND_LR>							
<EXP_EQUAL>		43				43	
<EXP_EQUAL_LR>							
<EXP_ADD>		50				50	
<EXP_ADD_LR>							
<EXP_MULT>		54				54	
<EXP_MULT_LR>							
2<EXP_NOT>		60				60	
<EXP_PARENTHESIS>		61				61	
<EXP_TERM>		63				64	
<IF>							
<IF_END>							
<CALL>							
<CALL_TAIL>							
<READ>							
<WRITE>							
<WRITE_TAIL>		75				75	

Table 4: Action table (1/5).

	procedure	move	compute	add	subtract	multiply	divide
<PROGRAM>							
<IDENT>							
<WORDS>							
<WORDS_LR>							
<END_INST>							
<ENV>							
<DATA>							
<VAR_LIST>	12						
<VAR_DECL>							
<VAR_DECL_TAIL>							
<LEVEL>							
<PROC>	15						
<LABELS>							
<LABELS_LR>							
<LABEL>							
<INSTRUCTION_LIST>		22	22	22	22	22	22
<INSTRUCTION>		24	24	24	24	24	24
<ASSIGNATION>		30	31	32	33	34	35
<ASSIGN_END>							
<EXPRESSION>							
<EXPRESSION_LR>							
<EXP_AND>							
<EXP_AND_LR>							
<EXP_EQUAL>							
<EXP_EQUAL_LR>							
<EXP_ADD>							
<EXP_ADD_LR>							
<EXP_MULT>							
<EXP_MULT_LR>							
<EXP_NOT>							
<EXP_PARENTHESIS>							
<EXP_TERM>							
<IF>							
<IF_END>							
<CALL>							
<CALL_TAIL>							
<READ>							
<WRITE>							
<WRITE_TAIL>							

Table 5: Action table (2/5).

	perform	accept	display	stop	-	not	(	true	false
<PROGRAM>									
<IDENT>									
<WORDS>									
<WORDS_LR>									
<END_INST>									
<ENV>									
<DATA>									
<VAR_LIST>									
<VAR_DECL>									
<VAR_DECL_TAIL>									
<LEVEL>									
<PROC>									
<LABELS>									
<LABELS_LR>									
<LABEL>									
<INSTRUCTION_LIST>	22	22	22	22					
<INSTRUCTION>	26	27	28	29					
<ASSIGNATION>									
<ASSIGN_END>						36	36	36	36
<EXPRESSION>						37	37	37	37
<EXPRESSION_LR>									
<EXP_AND>						40	40	40	40
<EXP_AND_LR>									
<EXP_EQUAL>						43	43	43	43
<EXP_EQUAL_LR>									
<EXP_ADD>						50	50	50	50
<EXP_ADD_LR>						52			
<EXP_MULT>						54	54	54	54
<EXP_MULT_LR>						57			
<EXP_NOT>						58	59	60	60
<EXP_PARENTHESIS>								61	62
<EXP_TERM>								65	66
<IF>									
<IF_END>									
<CALL>	70								
<CALL_TAIL>									
<READ>		72							
<WRITE>			73						
<WRITE_TAIL>						75	75	75	75

Table 6: Action table (3/5).



	or	and	=	<	>	<=	>=	*	/	if	else	end-if	+	until
<PROGRAM>														
<IDENT>														
<WORDS>														
<WORDS_LR>														
<END_INST>														
<ENV>														
<DATA>														
<VAR_LIST>														
<VAR_DECL>														
<VAR_DECL_TAIL>														
<LEVEL>														
<PROC>														
<LABELS>														
<LABELS_LR>														
<LABEL>														
<INSTRUCTION_LIST>												22		
<INSTRUCTION>												25		
<ASSIGNATION>														
<ASSIGN_END>														
<EXPRESSION>														
<EXPRESSION_LR>	38													
<EXP_AND>														
<EXP_AND_LR>	42	41												
<EXP_EQUAL>														
<EXP_EQUAL_LR>			49	44	45	46	47	48						
<EXP_ADD>														
<EXP_ADD_LR>				53	53	53	53	53						51
<EXP_MULT>														
<EXP_MULT_LR>									55	56				57
<EXP_NOT>														
<EXP_PARENTHESIS>														
<EXP_TERM>														
<IF>											67			
<IF_END>												68	69	
<CALL>														
<CALL_TAIL>														71
<READ>														
<WRITE>														
<WRITE_TAIL>														

Table 7: Action table (4/5).

<PROGRAM>	until	STRING	,	to	from	giving	)	then
<IDENT>								
<WORDS>								
<WORDS_LR>								
<END_INST>								
<ENV>								
<DATA>								
<VAR_LIST>								
<VAR_DECL>								
<VAR_DECL_TAIL>								
<LEVEL>								
<PROC>								
<LABELS>								
<LABELS_LR>								
<LABEL>								
<INSTRUCTION_LIST>								
<INSTRUCTION>								
<ASSIGNATION>								
<ASSIGN_END>								
<EXPRESSION>								
<EXPRESSION_LR>			39	39	39	39	39	39
<EXP_AND>								
<EXP_AND_LR>								
<EXP_EQUAL>								
<EXP_EQUAL_LR>								
<EXP_ADD>								
<EXP_ADD_LR>								
<EXP_MULT>								
<EXP_MULT_LR>								
<EXP_NOT>								
<EXP_PARENTHESIS>								
<EXP_TERM>								
<IF>								
<IF_END>								
<CALL>								
<CALL_TAIL>		71						
<READ>								
<WRITE>								
<WRITE_TAIL>						76		

Table 8: Action table (5/5).

## 3 Implementation

### 3.1 Parser Class

This class mainly handles the parsing and the semantic analysis.

Each rule of the grammar has been implemented in recursive methods. The parser can be accessed from the outside via the `parse(String)` method, which takes as argument the String to parse (typically the source code).

### 3.2 RaiseError Class

Whenever a problematic situation is encountered, one of those object is created. Its main purpose is to write on the error output the error message.

### 3.3 WarningError Class

Works just like `RaiseError`, but throw a warning instead of an error.

### 3.4 LLVMGenerator Class

This class has a method for every fonctionnality of the S-COBOL language that may be translated into LLVM IR.

It writes the code in a String, and finally write it into a file upon external solicitation of `toFile(String)` (the argument being the filename).

## 4 Semantic analysis

The following particularities are verified:

- every variable can not be assigned with a number longer than what its declaration authorizes (the length of a number being the number of digits composing it). To serve that purpose, the types of the variables are transmitted to the parent nodes in the recursive tree.
- The call list is checked at the end of the parsing. If a section is called without having been declared, an error is raised. If a section is declared without having been called, a warning is called.

## 5 Limitation of the current implementation

This section acts as a conclusion and presents what is left to be improved.

- The `OR` and `AND` keywords are not generated in the LLVM IR code.
- The order of some variables in the LLVM IR generated code are in reverse order.
- If an error is raised during the parsing, the execution is not stoped.