Free University of Bozen-Bolzano
Faculty of Computer Science
Bachelor in Computer Science and Engineering
Prof. Alessandro Artale

Formal Languages and Compilers – A.Y. 2016/2017 – 23.June.2017
`Compiler Part`
`Time: 1`^{h}`50 minutes`

STUDENT NAME:
STUDENT NUMBER:
STUDENT SIGNATURE:

`This is a closed book exam. The use of Pencils is not allowed! Write clearly, in the sense of logic, language and legibility. The clarity of your explanations affects your grade. Write your name and ID on every solution sheet.`

1 Exercise: Lexical Analyser [6 Points]

1. **Lexeme, Attribute and Token.** Describe these notions, what kind of information is associated to each of them and during what phase of the compilation we need these information.
   Make an example where you show what is a *lexeme* what is a *token* and what is the *attribute* when the tokens are *identifier* and the keyword *while*. [2 Point]

2. During the lexical analysis there are 2 kinds of conflicts:

   Case 1. The same Lexeme is recognized by two different RE's.

   Case 2. A given RE can recognize portion of a Lexeme.

   How is the first case solved? Furthermore, describe one technique that can solve the second case and show how it works in the case we need to recognize identifiers. [4 Points]

2 Exercise: LL(1) Top-Down Parsing [8 Points]

Given the following grammar with terminals $VT = \{a, b, c\}$:

$$\begin{array}{lcl} \mathtt{S} & \rightarrow & \mathtt{S\ F\ c} \mid \mathtt{G\ c} \mid \epsilon \\ \mathtt{F} & \rightarrow & \mathtt{a} \mid \mathtt{a\ F} \\ \mathtt{G} & \rightarrow & \mathtt{a} \mid \mathtt{b\ G\ a} \end{array}$$

1. Show why it is not LL(1). [2 Points]

Given the following grammar with terminals $VT = \{\mathbf{id}, , , =, : , ; , \mathbf{real}, \mathbf{int}\}$:

$$\begin{array}{rcl} \mathtt{P} & \rightarrow & \mathtt{SL} \\ \mathtt{SL} & \rightarrow & \mathtt{S}\ ;\ \mathtt{SL} \mid \epsilon \\ \mathtt{S} & \rightarrow & \mathtt{VD} \mid \mathbf{id} = \mathbf{id} \\ \mathtt{VD} & \rightarrow & \mathbf{real} : \mathtt{VL} \mid \mathbf{int} : \mathtt{VL} \\ \mathtt{VL} & \rightarrow & \mathbf{id}\ \mathtt{L} \\ \mathtt{L} & \rightarrow & , \mathbf{id}\ \mathtt{L} \mid \epsilon \end{array}$$

2. Show the value of the function `FIRST` for all the non terminal symbols. [1 Point]

3. Show the value of the function `FOLLOW` for all the non terminal symbols. [1 Point]

4. Show the parsing table for the LL(1) Top Down Parser recognizing the grammar. [2 Points]

5. Show the stack and the moves of the LL(1) parser on the input: "int : id_1;". [2 Points]

3 Exercise: Bottom-Up Parsing [12 Points]

Consider the following grammar with terminals $VT = \{\mathtt{a}, \mathtt{b}\}$:

$$\begin{aligned} \mathtt{S} &\rightarrow \mathtt{A}\ \mathbf{a} \mid \mathbf{b} \\ \mathtt{A} &\rightarrow \mathtt{B} \mid \mathbf{a} \\ \mathtt{B} &\rightarrow \mathbf{a} \mid \epsilon \end{aligned}$$

1. Prove that the above Grammar is not LR(1). Hint: It is enough to generate the starting state I_0. [3 Points]

Consider the following grammar with terminals $VT = \{;\ ,\mathtt{if}, \mathtt{then}. +, (,), \mathtt{id}, \mathtt{oth}\}$:

$$\begin{aligned} \mathtt{SL} &\rightarrow \mathtt{S}\ ;\ \mathtt{SL} \mid \mathtt{S} \\ \mathtt{S} &\rightarrow \mathbf{if}\ \mathtt{E}\ \mathbf{then}\ \mathtt{S} \mid \mathbf{oth} \\ \mathtt{E} &\rightarrow \mathtt{E} + \mathbf{id} \mid (\ \mathtt{E}\ ) \mid \mathbf{id} \end{aligned}$$

Show the following:

2. The canonical SLR collection. [4 Points]
3. The transition diagram describing the automaton which recognizes handles at the top of the stack. [1 Point]
4. The parsing table for the SLR parser. [2 Points]
5. The stack and the moves of the SLR parser on input: "if x then oth". [2 Points]

4 Exercise: Semantic Analysis [7 Points]

Consider the following grammar with terminals $VT = \{:, \texttt{id}, \texttt{vect}, \texttt{num}, \texttt{of}, [,], \texttt{int}, \texttt{string}\}$:

$$\begin{array}{rcl} \texttt{Decl} & \rightarrow & \texttt{T} : \texttt{VL} \\ \texttt{T} & \rightarrow & \textbf{int} \mid \textbf{real} \mid \textbf{vect}\ [\textbf{num}]\ \textbf{of}\ \texttt{T} \\ \texttt{VL} & \rightarrow & \texttt{VL},\ \textbf{id} \mid \textbf{id} \end{array}$$

which accepts vector declarations of the form: `vect[10] of int : x, y`.

1. Complete the following syntax directed definition:

Production	Semantic Rules
$\texttt{Decl} \rightarrow \texttt{T} : \texttt{VL}$	??
$\texttt{T} \rightarrow$ **int**	T.type = 'int'
$\texttt{T} \rightarrow$ **real**	T.type = 'real'
$\texttt{T} \rightarrow$ **vect** [**num**] **of** $\texttt{T}_1$	T.type = vect(num.val, ??)
$\texttt{VL} \rightarrow \texttt{VL}_1,$ **id**	$\texttt{VL}_1$.type = ??
$\texttt{VL} \rightarrow$ **id**	*addtype*(id.ptr, ??)

Where *addtype* is a function that adds to the symbol table entry id.ptr its type. [2 Points]

Given the following syntax directed definition:

Production	Semantic Rules
$Prog \rightarrow S$	$S.next := newlabel; Prog.code := S.code \parallel gen(S.next\ ':')$
$S \rightarrow S_1\ ;\ S_2$	$S_1.next := newlabel; S_2.next := S.next;$
	$S.code := S_1.code \parallel gen(S_1.next\ ':') \parallel S_2.code$
$S \rightarrow \text{while } Test \text{ do } \{S_1\}$	$Test.begin := newlabel;\ Test.true := newlabel;$
	$Test.false := S.next;\ S_1.next := Test.begin;$
	$S.code := gen(Test.begin\ ':') \parallel Test.code \parallel gen(Test.true\ ':') \parallel$
	$S_1.code \parallel gen('\text{goto}'\ Test.begin)$
$S \rightarrow \text{id} := E$	$S.code := E.code \parallel gen(\text{id}.place\ ':='\ E.place)$
$Test \rightarrow \text{id}_1 \text{ relop id}_2$	$Test.code := gen('\text{if}'\ \text{id}_1.place\ \text{relop}.op\ \text{id}_2.place\ '\text{goto}'\ Test.true) \parallel$
	$gen('\text{goto}'\ Test.false)$
$E \rightarrow E_1 + \text{id}$	$E.place := newtemp;$
	$E.code := E_1.code \parallel gen(E.place\ ':='\ E_1.place\ '+'\ \text{id}.place)$
$E \rightarrow \text{id}$	$E.place := \text{id}.place; E.code :='\ '$

Where:

- The function *newlabel* generates new symbolic labels: $l_1, l_2, \ldots$
- The function *newtemp* generates new variables names: $t_1, t_2, \ldots$

- The function *gen* generates strings such that everything in quotes is generated literally while the rest is evaluated.
- The attribute *code* produces the three-address code.
- The attribute id.*place* represents the name of the variable associated to the token id.
- The attribute relop.*op* represents the comparison operators (i.e., $<, <=, =, <>, >, >=$).
- The symbol $\|$ means string concatenation.

Given the input:

```
w := a + b;
while w > z do {
    z := z + a}
```

Show the following:

2. The annotated parse tree (without the *code* attribute) for the input together with the values of the attributes. [2 POINTS]
3. The three-address code produced by the semantic actions for the given input. [3 POINTS]