SCII/00825/2019
MICHAEL ORINA

COMPILER DESIGN AND CONSTRUCTION

CAT 2 – 22th SEP, 2022

1.
a) Write pseudocode to check for a valid identifier [3 Marks]

```
check first letter of an identifier
check if the identifier is five letters or less
if first letter is small letter and identifier is 5 letters or less
        Print “Valid Identifier”
else
        Print “Invalid Identifier”
```

b) Write pseudocode to check for a valid statement [3 Marks]

```
check keyword of the statement
check valid identifier
check operator
check costant
if keyword is int, boolean, load and identifier is valid and operator is valid
and constant is digit
        Print “Valid Statement”
else
        Print “Invalid Statement”
```

c) Implement a checker such that given a statement, your program output is either valid or invalid [10 Marks]

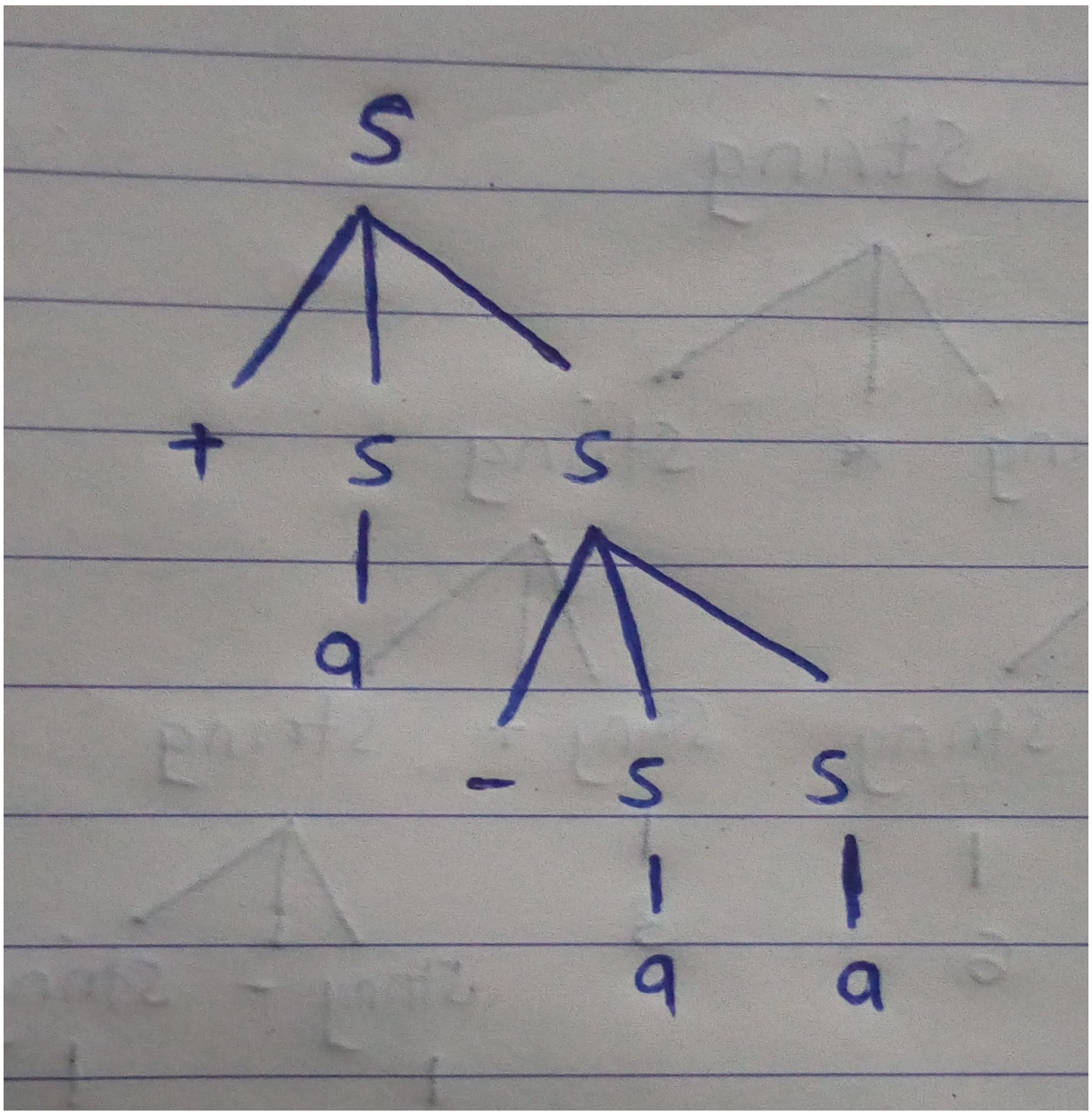
Code implemented in checker.java

2. Compare and contrast:
i) Recursive Descent parsing ii) Top-Down parsing iii) Predictive parsing [6 Marks]

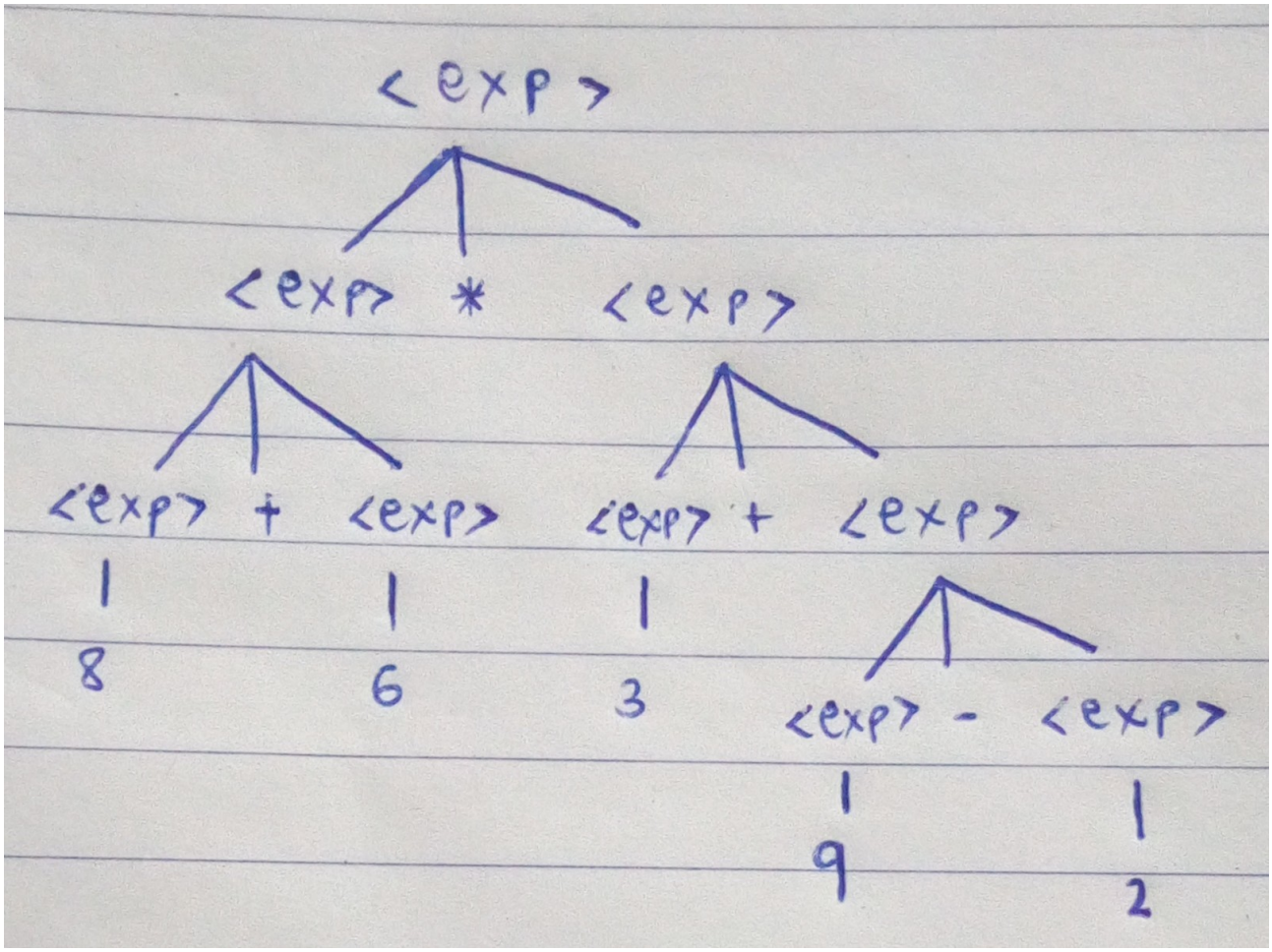
A top-down parser builds the parse tree from the top to down, starting with the start non-terminal.
A *Predictive Parser* is a special case of Recursive Descent Parser, where no Back Tracking is required.
Recursive Descent pursing is a top-down method of syntax analysis in which we execute a set of recursive procedures to process the input, a procedure which is associated with each non-terminal of a grammar.

Predictive parsing looks ahead symbol unambiguously and determines the procedure selected for each non-terminal.
The sequence of procedures called in processing the input implicitly defines a parse tree for the input.

3. Construct a recursive-descent parser, starting with the grammar: S -> + SS | -SS | a [6 Marks]



4. Given some form of computation for which the expression 8+6*3+9-2 yields 140, you are required to:
a. draw the parse tree, whose yield results in the value of the expression being 140. [4 Marks]



b. Give example production rules that could guide the parse tree you have generated using the following symbols: expr, +, *, - and digits 0 to 9 [4 Marks]

<exp> → <exp> * <exp> | <exp> + <exp> | <exp> - <exp> | digits
digits → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

5. What is Syntax directed translation? Describe the two concepts associated with syntax directed translation [5 Marks]

Syntax directed translation is an augmented type of grammar rule that facilitate semantic analysis. Along with the grammar, we associate some informal notations.
Synthesized Attributes are such attributes that depend only on the attribute values of children nodes. Thus [E -> E+T { E.val = E.val + T.val }] has a synthesized attribute val corresponding to node E. If all the semantic attributes in an augmented grammar are synthesized, one depth-first search traversal in any order is sufficient for the semantic analysis phase.
Inherited Attributes are such attributes that depend on parent and/or sibling's attributes. Thus [Ep -> E+T { Ep.val = E.val + T.val, T.val = Ep.val }], where E & Ep are same production symbols annotated to

differentiate between parent and child, has an inherited attribute val corresponding to node T

6. In designing (or re-designing) a new compiler, what are 4 considerations that one must deal with?[4Marks]

i. Production compilers must respond to concerns about **performance** and **usability** code. **Speed** must be paramount.
ii. The **architecture** must respond to the usage profile of its **environment** compatible with its associated software development environment. This might, for example, be batch or interactive.
iii. **Instruction selection**. Nature of instruction set of the target machine should be complete and uniform.
iv. **Register allocation**. Register can be accessed faster than memory. The instructions involving operands in register are shorter and faster than those involving in memory operand.