(Assignment #02)
Compiler Construction

Name:- M. Awais Tariq

Reg:- L1-F22-BSCS-1176

Date:- 31/Dec/2025

Section:- G-01

Task #01 (Language overview)

Mavlang is a custom case sensitive mini programming language inspired by Maverick (Top Gun). It uses aviation themed keywords and syntax to represent common programming constructs such as conditions, loops, functions variables and Input & output.

Style of Syntax:-

The syntax is keyword driven, readable, and Block structured using braces { }. It follows a C-like structure but uses aviation themed keywords for clarity & uniqueness.

Reason for choosing keywords:-

Keywords are inspired by aviation terminology to make the language distinctive, easy to remember, and clearly differentiated from existing programming languages.

Keywords from Phase-01:-

① Launch — Program Start.
② lock — Conditional Statement.
③ loopback — loop.
④ signal — output.
⑤ target — integer datatype.

Operators:-

1. assign — assignment operator.
2. ^^ — increment.
3. VV — decrement.

Punctuations:-

1. ; — Statement terminator
2. { } — Block delimeters

Task #02 (Grammer Definition (CFG)).

Non Terminals:-

<Program>, <Block>, <StmtList>, <Stmt>, <Decl>
<Assign>, <If>, <Loop>, <Output>, <Expr>

Terminals:-

Keywords, Identifiers, Operators,
Punctuations from Phase-01.

Task #03 (Sample Production Rules)

```
<Program>  →  launch <Block>
<Block>    →  { <StmtList> }
<StmtList> →  <Stmt> <StmtList>
<StmtList> →  ε
<Stmt>     →  <Decl>
<Stmt>     →  <Assign>
<Stmt>     →  <If>
<Stmt>     →  <Loop>
<Stmt>     →  <Output>
<Decl>     →  target identifier ;
<Assign>   →  identifier assign <Expr> ;
<If>       →  lock (<Expr>) <Block> .
<Loop>     →  loopback (<Expr>) <Block>
<Output>   →  signal identifier ;
<Expr>     →  identifier | integer
```

Task # 04 (First() & Follow() Sets)

1st Non-Terminals = $\boxed{\text{<Stmt>}}$

Productions:-

<Stmt> ⟶ <Decl> | <Assign> | <If> | <Loop> | <Output>

First (<stmt>)

{target, identifier, lock, loopback, signal}

2nd Non-Terminal = $\boxed{\text{<StmtList>}}$

Productions:-

<StmtList> ⟶ <Stmt> <StmtList> | ε

First()	Follow()
{target, identifier, lock, loopback, signal}	{ "}" }

Task #05 (Ambiguity Check)

Q. Is the Grammer Ambiguous?

No! for the current construct.

Reason:-

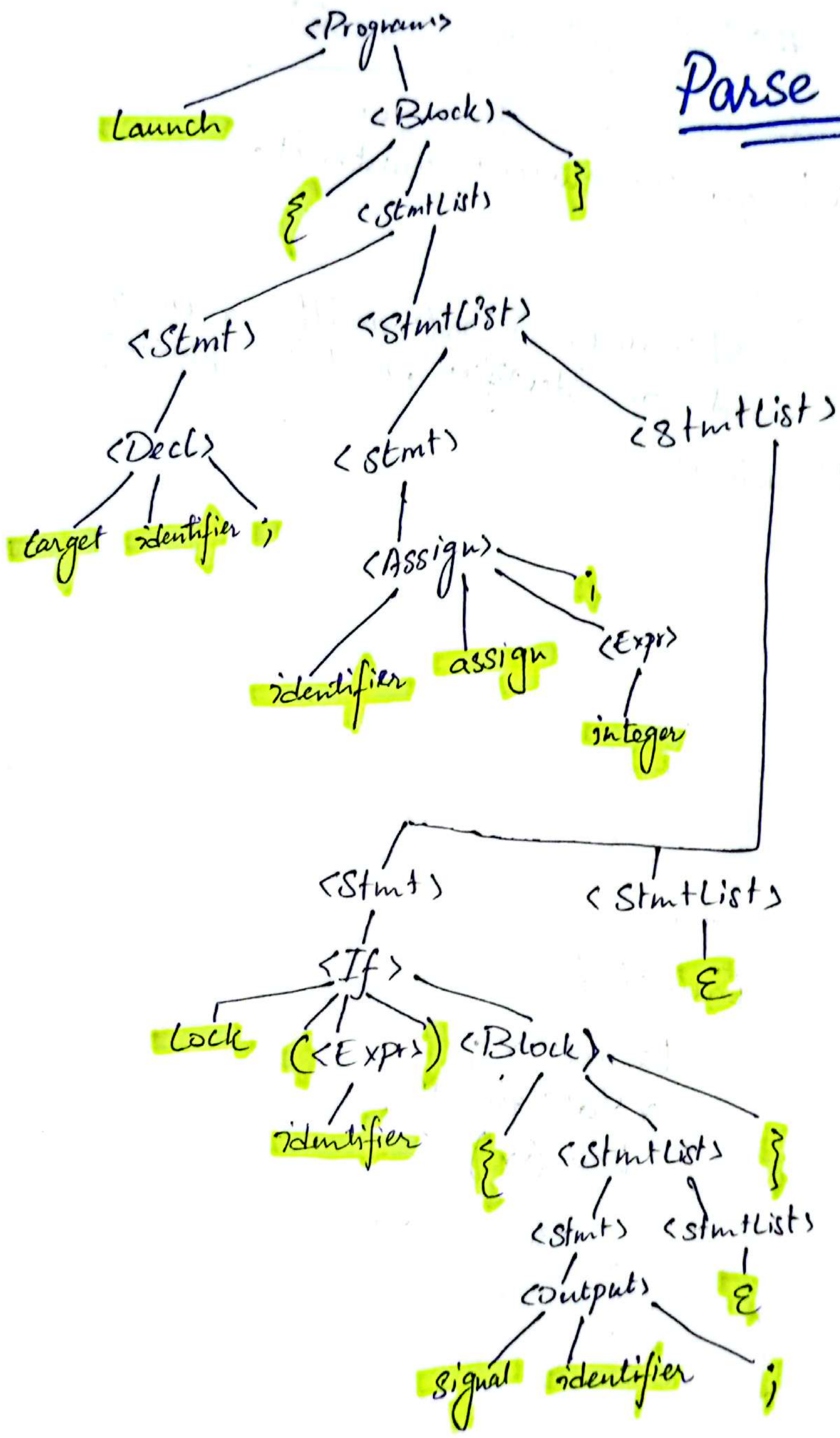
Each statement starts with a unique keyword or identifier, making parsing deterministic.

Task #06 (Parse Tree Construction)

Program:-

```
launch{
    target x;
    x assign 10;
    lock (x) {
        signal x;
    }
}
```

Parse Tree



<Program>
Launch
<Block>
{
<StmtList>
}
<Stmt>
<StmtList>
<Decl>
target
identifier
;
<stmt>
<StmtList>
<Assign>
identifier
assign
<Expr>
;
integer
<Stmt>
<StmtList>
ε
<If>
Lock
(<Expr>)
<Block>
identifier
{
<StmtList>
}
<Stmt>
<stmtList>
ε
<Output>
signal
identifier
;

Task #07 (Error Scenarios)

(1) Error Snippet :-

3: target 123x;

Line : 3
Error: Invalid Identifier
Violated Rule: <Decl> → target idendifier;
Expected Token: identifier.

(2) Error Snippet :-

5: x assign;

Line: 5
Error: Missing Expression
Violated Rule: <Assign> → identifier assign <Expr>;
Expected Token: integer or identifier.