
# University of Central Punjab
## Faculty of Information Technology

# Compiler Construction
## Assignment 02

**Total Marks: 10** ***Due Date: 1 January 2026***

***Instructions:***

1. ***This is your individual handwritten assignment.***
2. *Your assignment should be* ***plagiarism free****. In case of plagiarism, straight Zero would be rewarded.*
3. *Your assignment is projection of your project phase 02.*
4. *No late submissions would be entertained in any case.*

---

**Name: Nayab Irfan**

**Roll No: L1F22BSCS0402**

**Section: G5**

**Assignment 2**

## Task 1 : Purpose :

Avian is a minimalist educational Programing language designed to demonstrate Compiler Construction principle while maintaing C++ familiarity. The word 'avian' comes from 'birds' or "flight" aiming it to be flight of compiler construction.

## Style of Syntax :

Avian adopts a hybrid syntax style. Combines C++ structure mostly. I used underscores before and after of usual C++ style of syntax with avn_ i.e., avn_if , avn_for , avn_int and so on. Operators were given underscores as well for simplicity and readibility. i.e., _op_ eg: _+_ , _-_ , _*_ ---- and so on.

## Reasons for Keywords :

The syntax and name conventions were chosen for following reasons :

→ Namespace Isolation , prevents collisions between keywords and identifiers.

→ enchanced readibility.

→ Simplified Parsing

→ Educational Demonstration.

→ Avian self identity ("Avian" → avn_)

## Language Elements from Phase 1:

| Keyword | Purpose | Example |
|---|---|---|
| • avn_int | integer variable | avn_int n = 10; |
| • avn_if | conditional branch | avn_if (x_>_0){...} |
| • avn_main | program entry | avn_int avn_main() |
| • avn_cin | input stream | avn_cin_>>_n; |
| • avn_break | loop break | avn_break; |
| • avn_for | for loop | avn_for (i=0; i<5; i++) {....} |

| Operators | Description | Example |
|---|---|---|
| • _+_ | addition | sum = a_+_b; |
| • = | equal (assignment) | x = 10; |
| • >> | exsersion output | avn_cout << x; |

| Punctuations | Purpose | Example |
|---|---|---|
| , | Comma | avn_int add(a, b) |
| ; | Statement Terminator | x = 5; |
| ( ) | Paranthesces | avn_if (x=5){...} |

# Task 2 : CFG

// Start

P

P → GL FL

GL → D GL | ε

FL → F FL | ε

// function definition

F → avn_func T id (para) B

F → avn_func T avn_main () B

// DType

T → avn_int | avn_float | avn_double | avn_char

T → avn_void

// Parameters

Para → PL | ε

PL → PR PL'

PL → , PR PL' | ε

PR → T id

// Bracket Blocks

B → { SL }

SL → S SL | ε

// Statement Expression

S → D | A | IF | WH | FOR | OUT | IN

S → RET | ES | avn_end ;

// Variable Delaration

D → T id DI ;

DI → = E | ε

// Assigment operator

A → id = E ;

// Conditional Statement

IF → avn_if (C) B ELSE

ELSE → avn_else B | ε

// Loops

WH → avn_while (C) B

FOR → avn_for (Afor, Copt, Upot) B

Afor → id = E | ε

Copt → C | ε

Upt → U | ε

// Operators (Unary)

U → id_++_ | id_--_ | _--_id | _++_id

// Input / output

OUT → avn_cout << OL ;

OL → OI OL'

OL' → , OI OL' | ε

OI → str | id | num

// Input

IN → avn_cin >> IO ;

IO → id IO'

IO' → id IO' | ε

// Return

RET → avn_return Eopt ;

Eopt → E | ε

// Expression

ES → CALL | V ;

CALL → id (ALopt) .

ALopt → AL | ε

AL → E AL'

AL' → , E AL' | ε

// Conditions + operators

C → C&&R | R

R → E RO E | E

RO → < | > | == | != 

// Arithmetic Operators

E → E AO T | T

AO → _+_ | _-_

T → T MO F | F

MO → _*_ | _/_

F → (E) | id | num | ch | str | CALL

# Task 3:

## Production Rules

P → F

F → avn_func avn_int id() B

B → {SL}

SL → S SL | ε

S → D | A | IF | WH | OUT

D → avn_int id ;

A → id = num ;

IF → avn_if (num) B

WH → avn_while (num) B

OUT → avn_cout << id ;

# Task 4: FIRST & FOLLOWS

| Non Terminal | First | Follow |
|---|---|---|
| P (start) | { avn_func } | { $ } |
| S | { avn_int, id, avn_if avn_while, avn_cout, ε } | { avn_int, id, avn_if avn_while, avn_cout $} |
| SL | {avn_int, id, avn_if, avn_while, avn_cout, ε} | { [illegible] }, $ } |
| IF | { avn_if } | { avn_int, avn_if, avn_while, avn_cout, $} |

# Task 5

## Ambiguity Check

The ambguity occurs in the if-else statement.

IF → avn_if (num) B | avn_if (num) B avn_else B

This creates a problem clasically "dangling else problem" where it is unclear which avn_if and avn_else belongs to when nested if statements are used.

example :

```
avn_if (num)
    avn_if (num) B
avn_else B
```

Here 'avn_else' can be assosiated with either inner if and outer if.

Solution :

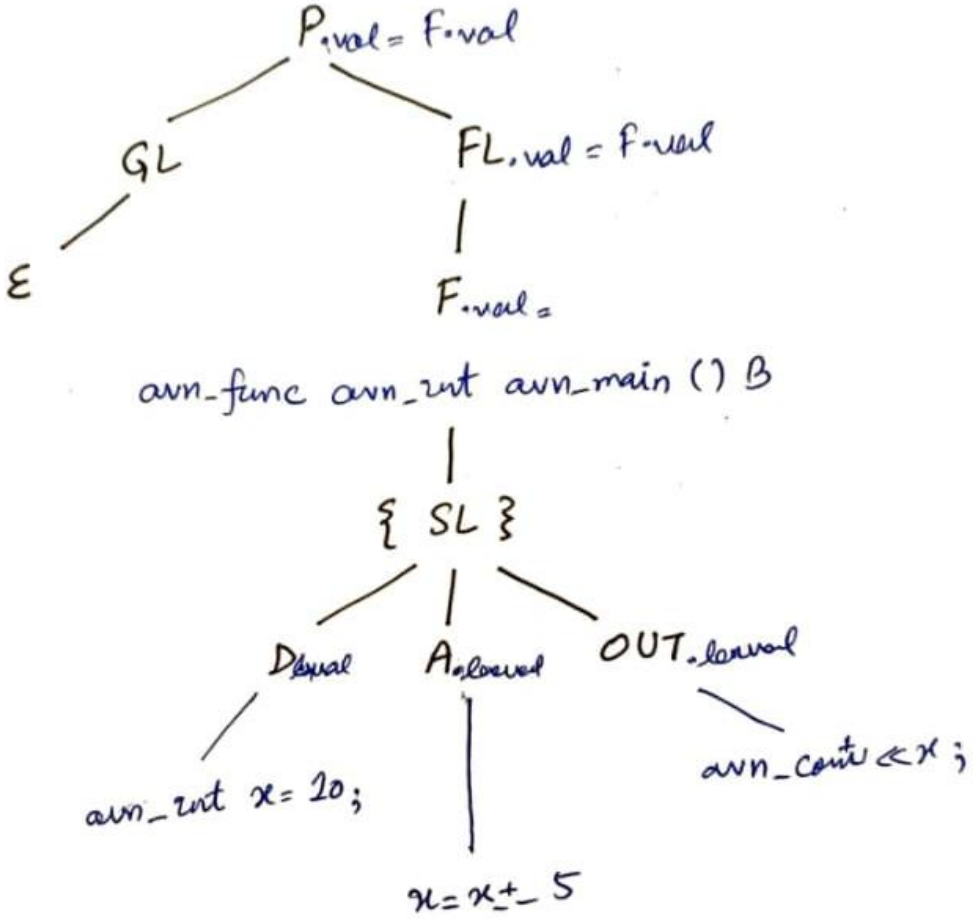
In implementation is resolved by attaching avn_else to nearest unmatched avn_if.

Parser generator like YACC/Bison handle this using procedural rules or by separating matched and unmatched if statements.

# Task 6:

## Parse Tree

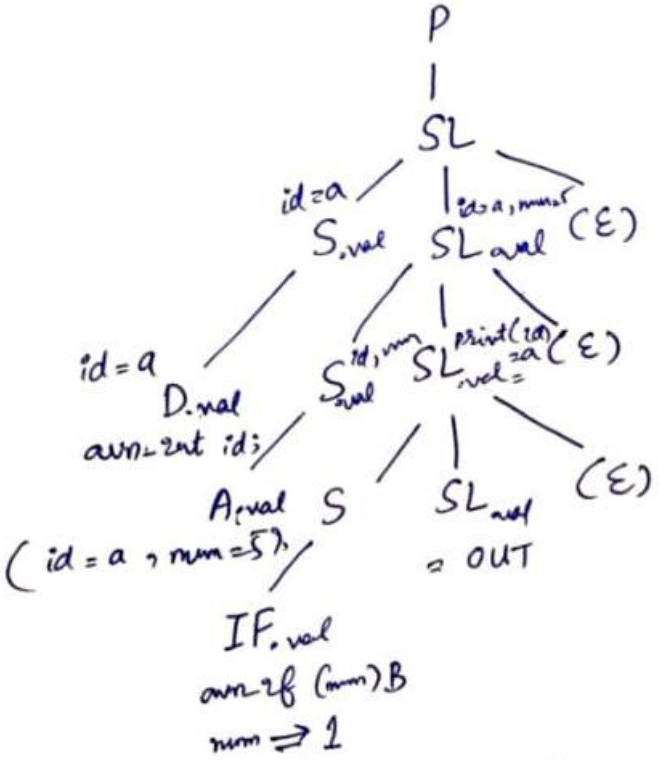
```
avn_int  x = 10 ;
x = x_+_10 ;
avn_cout << x ;
```



# Task 6 :

## Passe Tree

```
avn_int a;
a = 5;
avn_if (1) { avn_print a; }
```



# Task 7:
# Error Scenrios

## E1: Missing Semi-Colon

avn_int a

Rule violated: D → avn_int id ;

Expected token: ;

## E2: Missing closing Paranthesis

avn_if (2 { avn_cout << a ; }

Rule Violated: avn_if (num) B

Expected Token: )