TALHA MUDASSAR L1F22BSCS0379

Compiler Construction Project Phase 2

Language Overview:
PUNJ++ is a high-level programming language designed to bridge the gap between human thought and machine logic for Punjabi speakers. It replaces traditional English syntax with Roman Punjabi expressions, making the logic intuitive and conversational. While the syntax is localized, the structure maintains backward compatibility with C++ logic (procedural style) to ensure a smooth transition for students. It uses distinct blocking styles to manage scope visually.
Purpose:
To simplify the learning curve for first-time programmers by removing the language barrier, allowing them to focus on logic rather than memorizing foreign syntax.
Syntax Style:
Procedural, Case-Sensitive, and Block-Structured. It uses ::: to start blocks and :::; to close them, creating a clear visual rhythm.
Reason for Choosing Keywords:
The keywords are chosen from everyday Punjabi conversation. For example, fher (then/if) implies a condition naturally, while likh (write) and dass (tell) map directly to input and output actions.

Keywords

These keywords are Punjabi-inspired replacements for C++ words.

KEYWORDS	MEANING	EQUIVALENT C++
Fher	Conditional check	If
Nahi Ta	Other wise	Else
Jad Tak	Loop Until fail	While
Likh	Take input	Cin
Dass	Print output	Cout
Morjaa	Return Value	Return
Kaam	Loop	For

Chakkar	Repeat Loop	Do/while
Rok	Break	Break
Jaari	Continuous	Continue
Nava	New	New
Class	Class decleration	Class
Dekh	Switch-Like structure	Switch
Halat	Case Condition	Case
Mukao	Default case	Default

Reasoning: These words are chosen because they are easy for Punjabi speakers to understand and make programming feel natural and intuitive.

Operators:

Operators	Meaning
<+>	Add and assign
<->	Equal comparsion
<!>	Not equal
++>	Increment

Punctuations:

Symbol	Description
:::	Start of custom block
:::;	End of custom block
~>	Custom end marker
<>	Custom separator

Part 1: Documentation Content :

Terminals:

Class, int, float, char,
Fher, nahiTa, jadTak, likh, dass,
IDENTIFIER, NUMBER,
::: , :::; , ~>, <>
<+>, <->, <!>

1.Grammar (CFG):

1. Program → KW_class IDENTIFIER PUNC_LBLOCK StmtList PUNC_RBLOCK

2. StmtList → Stmt StmtList | ε

3. Stmt → Declaration | Assignment | IfStmt | Loop | IOStmt | BlockStmt

4. Declaration → Type IDENTIFIER PUNC_END

5. Type → KW_in | KW_float | KW_char

6. Assignment → IDENTIFIER OP_ADDASSIGN Expression PUNC_END
| IDENTIFIER OP_ASSIGN_PUNJ Expression PUNC_END

7. IfStmt → KW_fher LPAREN Condition RPAREN BlockStmt ElsePart

8. ElsePart → KW_nahiTa BlockStmt | ε

9. Loop → KW_jadTak LPAREN Condition RPAREN BlockStmt

10. IOStmt → KW_likh OP_SHIFT_RIGHT IDENTIFIER PUNC_END
| KW_dass OP_SHIFT_LEFT Expression PUNC_END

11. BlockStmt → PUNC_LBLOCK StmtList PUNC_RBLOCK

12.Condition → Expression OP_EQUAL Expression
| Expression OP_NOTEQ Expression

| Expression OP_LT Expression
| Expression OP_GT Expression

13.Expression → Expression OP_PLUS Expression
| Expression OP_MINUS Expression
| Expression OP_MULT Expression
| Expression OP_DIV Expression
| Expression OP_MOD Expression
| IDENTIFIER
| NUMBER_INTEGER
| NUMBER_FLOAT
| STRING_LITERAL

2. FIRST and FOLLOW Sets:

Assignment → IDENTIFIER OP_ADDASSIGN Expression PUNC_END
| IDENTIFIER OP_ASSIGN_PUNJ Expression PUNC_END

First	Follow
{ IDENTIFIER }	{ KW_int, KW_float, KW_char, IDENTIFIER, KW_fher, KW_jadTak, KW_likh, KW_dass, PUNC_RBLOCK (:::;) }

IfStmt → KW_fher LPAREN Condition RPAREN BlockStmt ElsePart

First	Follow
{ KW_fher }	{ KW_int, KW_float, KW_char, IDENTIFIER, KW_fher, KW_jadTak, KW_likh, KW_dass, PUNC_RBLOCK (:::;) }

Declaration → Type IDENTIFIER PUNC_END

First	Follow
{ KW_int, KW_float, KW_char }	{ KW_int, KW_float, KW_char, IDENTIFIER, KW_fher, KW_jadTak, KW_likh, KW_dass, PUNC_RBLOCK }

3.Parse Tree + Explaination:

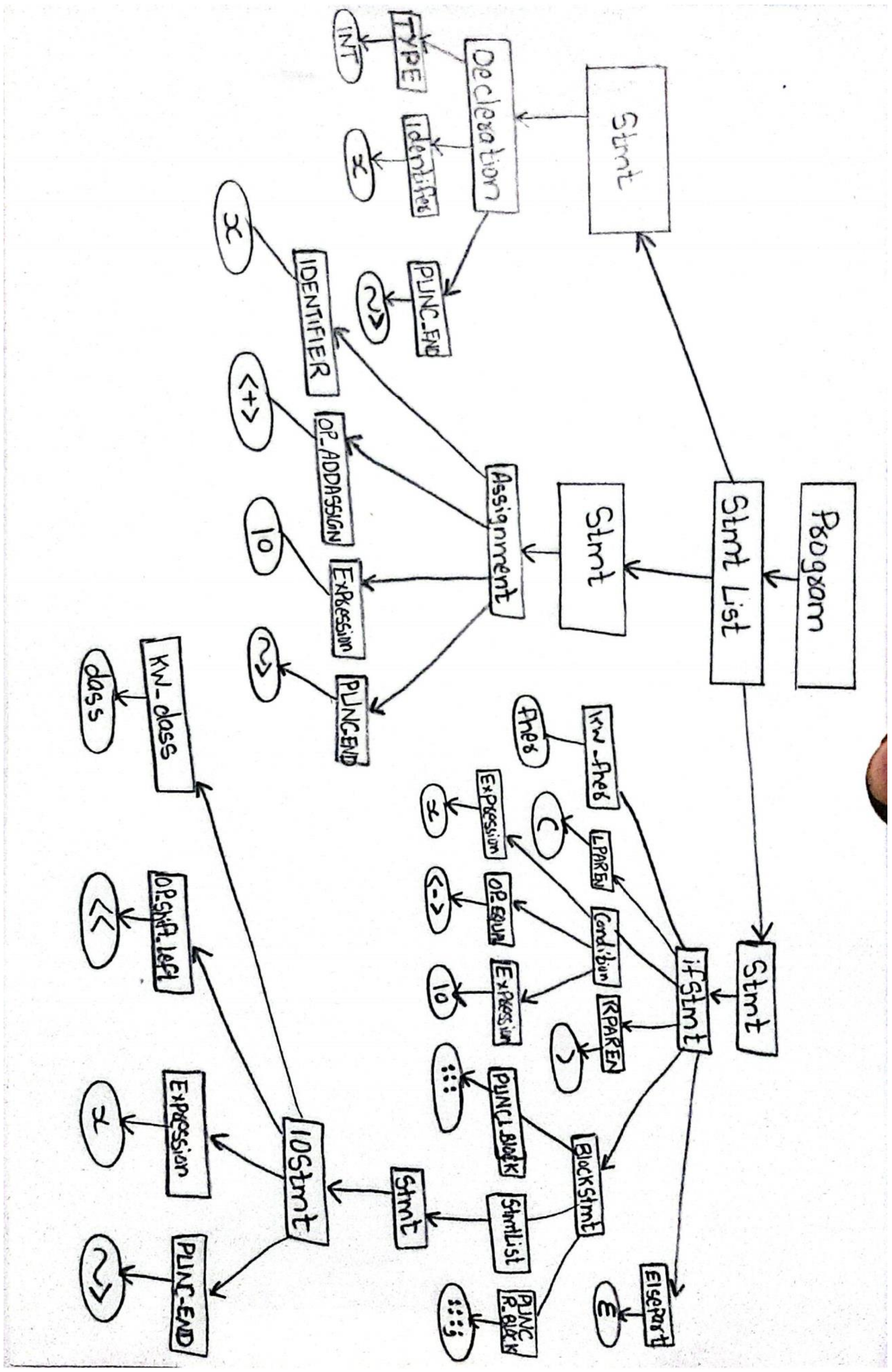
Valid Program Fragment:

OUR LANGUAGE	**C++**

int x ~>
x <+> 10 ~>
 	
fher (x <-> 10) :::
    dass << x ~>
:::;
	int x;
	x= x+10;
	
	if(x = 10){
	cout << x;
	};

PARSE TREE:

TALHA MUDASSAR L1F22BSCS0379



. Parse Tree Explanation:

the parse tree starts at `Program`. It branches into the class definition. The `StmtList` recursively expands.

1. **Declaration:** `int x ~>` is parsed where `Type` matches `int` and `PUNC_END` matches `~>`.
2. **Assignment:** `x <+> 10 ~>` is parsed. `x` is the ID, `<+>` is the operator.
3. **IfStmt:** `fher (x <-> 10)` is checked. The condition matches. The block `:::` opens, contains an `IOStmt` (`dass << x`), and closes `:::;`.