SER 502 Design Document

Team 4
Aneesh Dalvi
Janice Abraham
Jinal Patel
Viraj Talaty

Table of Contents

INTRODUCTION

DESI - Our compiler is based on some words of Hindi dialect, so we have given our compiler name 'Desi'. We are excited to inculcate some Hindi keywords in our programming language to spread our culture with it. Our language follows imperative paradigm.

Our language allows working with integer and boolean data types and perform basic mathematical arithmetic operations. We have also implemented control abstraction with the help of rules defined loop(while) and conditional statement(if-else).

DATA TYPES

Following Data Types are supported and can be defined/declared:

ID	Data Type	Description
1	**int**	This data type will have natural numbers but will limit to 4 bytes.
2	**bool**	This data type will have 'true' or 'false' as the values

GRAMMAR RULES

1. Indentation is not mandatory in the language.
2. Precedence is incorporated and supports different arithmetic operations with multiple operators.
3. Every statement must end with a semicolon.
4. Any program block is enclosed between 'shuru' and 'khatam' keywords which are reserved for our language.
5. Our grammar is unambiguous and free of issue of left recursion.
6. All Declarations are allowed at the beginning of the program and assignment is allowed anywhere within program block.

GRAMMAR

program → *shuru* block *khatam*
block → declaration; command | command

declaration → datatype identifier|
datatype identifier = digit |
datatype identifier = boolean |
datatype identifier; declaration |
datatype identifier = digit; declaration |
datatype identifier = boolean; declaration .

datatype → int | bool
command → expression; |
shuru block *khatam* |
jabtak (boolean) *shuru* block *khatam tabtak* |
agar (boolean) *shuru* block *khatam nahitoh shuru* block *khatam bas* |
expression > expression |
expression < expression |
expression == expression |
expression <= expression |
expression >= expression

expression → identifier = expression | expression1.
expression1 → term + expression | term - expression | term .
term → factor * term |factor / term|factor.
factor → (expression) | unit.
unit → identifier|digit.
boolean → true | false | expression = expression | !boolean
identifier → [a-z]+
digit → [0-9]+

OPERATORS

Our programming language supports the following operators -

Id	Operator	Meaning
1	+	Addition
2	-	Subtraction
3	*	Multiplication

4	/	Division
5	==	Comparison
6	<	Less Than
7	>	Greater Than
8	<=	Less Than Equal To
9	>=	Greater Than Equal To
10	=	Assignment

KEYWORDS

Our grammar supports the following keywords -

Id	Keywords	Description
1	**shuru**	block starting point.
2	**khatam**	block ending point.
3	**int**	indicates integer data type.
4	**bool**	indicates boolean data type.
5	**agar**	it is followed by a condition and execution takes place if that condition is true. - same as ‘if’ condition
6	**nahitoh**	it is followed by a condition and execution takes place if the previous if condition is false and the current condition evaluates to true. - same as ‘else’ condition
7	**bas**	end of the ‘agar’ statement. Same as ‘endif’ statement
8	**jabtak**	It is followed by a condition and execution takes place until that condition holds true.
9	**tabtak**	end of the while loop.

TOOLS

ANTLR 4.7.1 (Another Tool for Language Recognition) to generate a parser to build the parse trees and verify our grammar rules.
SourceCode.desi to generate the parse tree for the grammar.
ImmdCode.vdesi to run intermediate assembly code to generate program output.

PARSING TECHNIQUE

We decided to use ANTLR as we are going to use a top down approach.

INTERPRETER

We have decided to use Three Access Code method for generating the intermediate code for the given input.

PARADIGM OF THE LANGUAGE

Imperative Paradigm

DATA STRUCTURES

1. Hash Map
2. Arrays

ADDITIONAL DETAILS

We have verified our grammar by using prolog DCG to test sample runs for the designed grammar.