**Custom Language Compiler using Flex and Bison - Documentation**

**Overview**

This project implements a simple compiler for a custom-designed language using **Flex** (Lex) and **Bison** (Yacc). The language supports variable assignment, arithmetic operations, conditional statements (if-else), loops (while), and output statements (print). The compiler converts high-level code into **Three Address Code (TAC)** and stores it in an output file.

**Custom Language Keywords**

| **Keyword** | **Meaning** | **Equivalent in C/C++** |
| --- | --- | --- |
| shuru | Start of program | int main() / { |
| bolo | Output/Print | printf() |
| jabtak | While loop | while |
| agar | If condition | if |
| nahi | Else clause | else |

**Grammar Rules (Bison - yacc.y)**

**Start Symbol**

program: SHURU statements

**Statements**

statements:

statements statement

| statement

statement:

ID ASSIGN expression ';'

| BOL ID ';'

| JABTAK '(' condition ')' '{' statements '}'

| AGAR '(' condition ')' '{' statements '}'

| AGAR '(' condition ')' '{' statements '}' NAHI '{' statements '}'

**Expressions**

expression:

expression '+' expression

| expression '-' expression

| expression '\*' expression

| expression '/' expression

| ID

| NUMBER

**Conditions**

condition:

expression LT expression

| expression GT expression

| expression LE expression

| expression GE expression

| expression EQ expression

| expression NE expression

**Lexical Rules (Flex - lex.l)**

**Keywords**

"shuru" { return SHURU; }

"bolo" { return BOL; }

"jabtak" { return JABTAK; }

"agar" { return AGAR; }

"nahi" { return NAHI; }

**Operators and Symbols**

"+" { return PLUS; }

"-" { return MINUS; }

"\*" { return MUL; }

"/" { return DIV; }

"=" { return ASSIGN; }

"==" { return EQ; }

"!=" { return NE; }

"<" { return LT; }

">" { return GT; }

"<=" { return LE; }

">=" { return GE; }

";" { return SEMICOLON; }

"(" { return LPAREN; }

")" { return RPAREN; }

"{" { return LBRACE; }

"}" { return RBRACE; }

**Identifiers and Numbers**

[a-zA-Z\_][a-zA-Z0-9\_]\* { yylval.str = strdup(yytext); return ID; }

[0-9]+ { yylval.str = strdup(yytext); return NUMBER; }

**Sample Input with All Functionalities**

shuru

x = 10;

y = 5;

z = x + y;

agar (z > 10) {

bolo z;

} nahi {

bolo y;

}

i = 0;

sum = 0;

jabtak (i < 3) {

sum = sum + i;

i = i + 1;

}

bolo sum;

**Corresponding Three Address Code Output**

t1 = 10

x = t1

t2 = 5

y = t2

t3 = x + y

z = t3

t4 = 10

t5 = z > t4

if t5 goto L1

goto L2

L1:

print z

goto L3

L2:

print y

L3:

t6 = 0

i = t6

t7 = 0

sum = t7

L4:

t8 = 3

t9 = i < t8

if t9 goto L5

goto L6

L5:

t10 = sum + i

sum = t10

t11 = 1

t12 = i + t11

i = t12

goto L4

L6:

print sum

**Files in the Project**

| **File** | **Purpose** |
| --- | --- |
| lex.l | Lexical analyzer: Tokenizes input code |
| yacc.y | Parser and code generator for TAC |
| input.txt | Contains source code in custom language |
| output.txt | Stores the generated TAC |

**Contributors**

* Project Author: [Harshil Brahmani]
* Subject: Compiler Design (B.Tech CSE, Semester 6)

**Conclusion**

This project demonstrates a working compiler for a custom language using Flex and Bison. It transforms high-level structured code into low-level three-address representation, forming the foundation for further compilation or interpretation stages.