Computer Science and Engineering

Compiler Design Laboratory

Course No : CSE-3212

Project : KothaCompiler, A Banglish-Themed Programming Language Compiler

Name : MD Tahsinur Rahman

Roll : 2007091

Section : B

Date : 14/01/2025

Introduction:

KothaCompiler is an innovative programming language compiler designed to facilitate the development and execution of scripts in a user-friendly and efficient manner. The name "Kotha," which means "word" or "speech" in Bengali, reflects the compiler's core mission: to empower users to express their ideas and logic through a structured language that is both intuitive and powerful.

Built using Flex and Bison, KothaCompiler leverages the strengths of lexical analysis and parsing to transform high-level code into executable instructions. It supports a variety of features, including:

Rich Syntax: KothaCompiler introduces a unique syntax that combines familiar programming constructs with culturally relevant keywords, making it accessible to a broader audience, especially those familiar with Bengali.

Mathematical Operations: The compiler includes built-in support for mathematical functions and operations, allowing users to perform complex calculations seamlessly.

Control Structures: With support for conditional statements, loops, and functions, KothaCompiler enables users to write dynamic and efficient code.

Error Handling: KothaCompiler is equipped with robust error detection and reporting mechanisms, ensuring that users receive meaningful feedback during the development process.

Symbol Table Management: The compiler maintains a comprehensive symbol table to track variable declarations, types, and values, facilitating effective memory management and type checking.

KothaCompiler aims to bridge the gap between traditional programming languages and the needs of local developers, providing a platform that encourages creativity and innovation. Whether you are a beginner looking to learn programming concepts or an experienced developer seeking a new tool for rapid development, KothaCompiler offers a versatile and engaging environment to bring your ideas to life..

Flex Code:

```
digits [0-9]

Datatype "purno"|"vogno"|"faka"|"string"

Identifiers [a-zA-Z][a-zA-Z0-9]*

S_comment \/\/[^\n]*

M_comment \/\([^]|\[^\/])\*\/


%{
        #include "2007091.tab.h"
        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        int varindex(char *var);
        extern int yylex();
        extern int yyparse();
        extern FILE *yyin;
        extern FILE *yyout;
        int yyerror(char *s);
        //int lineNo = 1;
%}


%%
```

```
{S_comment} { printf("\nSingle Line Comment\n"); }

{M_comment} { printf("\nMultiple Line Comment\n"); }


"purno" { return PURNO; }

"vogno"          { return VOGNO; }

"string" { return STRING; }


"(" { return '('; }

")"       { return ')'; }

"<" { return '<'; }

">"       { return '>'; }

"{"       { return '{'; }

"}"       { return '}'; }

";" { return SESH; }

"," { return ','; }

"="       { return '='; }

":" { return ':'; }


"+"       { return '+'; }

"-"       { return '-'; }

""        { return ''; }

"/"       { return '/'; }

"^"       { return '^'; }

"vag" { return VAG; }
```

```
"choto" { return CHOTO; }

"boro" { return BORO; }

"soman"          { return SOMAN; }

"boro_soman" { return BORO_SOMAN; }

"choto_soman" { return CHOTO_SOMAN; }

"soman_na" { return SOMAN_NA; }


"+++" { return BARAO; }

"---" { return KOMAO; }

"!"       { return NA; }


"sin" { return SIN; }

"cos" { return COS; }

"tan" { return TAN; }

"ln" { return LN; }

"log" { return LOG; }

"jor_bijor" { return JOR_BIJOR; }

"factorial"       { return FACTORIAL; }

"boro_man" { return BORO_MAN; }

"choto_man" { return CHOTO_MAN; }

"moulik" { return MOULIK; }


"dekhao" { return DEKHAO; }
```

```
"jodi" { return JODI; }
"nahole_jodi" { return NAHOLE_JODI; }
"nahole" { return NAHOLE; }

"jotokhon" { return JOTOKHON; }
"barao" { return BARAO_FOR; }
"komao" { return KOMAO_FOR; }
"jokhon" { return JOKHON; }

"khetre" { return KHETRE; }
"bikolpo" { return BIKOLPO; }
"onnothai" { return ONNOTHAI; }

"-"?{digits}+ {
        yylval.string = strdup(yytext);
        return NUMBER;
}

"-"?({digits}+)?"."{digits}+ {
        yylval.string = strdup(yytext);
        return NUMBER;
}
```

```
"\""[^"]*"\"" {
        yylval.string = strdup(yytext);
        return STR;
}

{Datatype}[ ]+"main" { return MAIN; }
"#include" { return IMPORT; }
{Identifiers}".h" { return HEADER; }

"function" { return FUNCTION; }

{Identifiers} {
        yylval.string = strdup(yytext);
        return VARIABLE;
}

[ \t\n]*

. {yyerror("Unknown Character.\n");}

%%
```

Grammar Rules:

/* Program Structure */

program ::= import 'main' '(' ')' '{' statements '}'

| ε

/* Basic Components */

VARIABLE ::= [a-zA-Z][a-zA-Z0-9]*

NUMBER ::= [0-9]+(\.[0-9]+)?

STR ::= ".*"

HEADER ::= [a-zA-Z]+\.h

/* Types */

type ::= PURNO /* Integer */

| VOGNO /* Float */

| STRING /* String */

/* Operators */

Arithmetic_Operators ::=

'+' /* Addition */

| '-' /* Subtraction */

| '*' /* Multiplication */

| '/' /* Division */

```
    | '^'   /* Power */
    | VAG   /* Modulus */

Relational_Operators ::=
    CHOTO      /* Less than (<) */
    | BORO     /* Greater than (>) */
    | SOMAN    /* Equal to (=) */
    | BORO_SOMAN /* Greater than or equal to (>=) */
    | CHOTO_SOMAN /* Less than or equal to (<=) */
    | SOMAN_NA   /* Not equal to (!=) */

Increment_Decrement ::=
    BARAO   /* Increment (++) */
    | KOMAO  /* Decrement (--) */

Logical_Operators ::=
    NA     /* NOT */

/* Control Structures */
if_structure ::=
    JODI '(' expression ')'          /* If */
    | NAHOLE_JODI '(' expression ')'  /* Else if */
    | NAHOLE                         /* Else */
```

```
loop_structure ::=
    JOTOKHON    /* For loop */
    | JOKHON    /* While loop */

loop_operators ::=
    BARAO_FOR   /* Increment in for loop */
    | KOMAO_FOR  /* Decrement in for loop */

switch_structure ::=
    BIKOLPO     /* Switch */
    | KHETRE    /* Case */
    | ONNOTHAI  /* Default */

/* Functions */
Built_in_Functions ::=
    DEKHAO '(' VARIABLE ')'   /* Print function */
    | SIN '(' expression ')'  /* Sine */
    | COS '(' expression ')'  /* Cosine */
    | TAN '(' expression ')'  /* Tangent */
    | LOG '(' expression ')'  /* Log base 10 */
    | LN '(' expression ')'   /* Natural log */
    | JOR_BIJOR '(' expression ')'  /* Even/Odd check */
```

```
    | FACTORIAL '(' expression ')'   /* Factorial */
    | BORO_MAN '(' expression ',' expression ')'  /* Maximum */
    | CHOTO_MAN '(' expression ',' expression ')' /* Minimum */
    | MOULIK '(' expression ')'      /* Prime check */

User_Function ::=
    FUNCTION VARIABLE '(' param ')'   /* Function declaration */

/* Other Keywords */
Keywords ::=
    MAIN       /* Main function */
    | IMPORT    /* Import statement */
    | SESH      /* Statement terminator (like semicolon) */

/* Parameter and Variable Declaration */
param ::= param ',' type pid
        | type pid

declare ::= type id SESH

id ::= id ',' VARIABLE
     | id ',' VARIABLE '=' expression
     | VARIABLE
```

```
    | VARIABLE '=' expression


/* Statements */
statements ::= statements cstatement
            | ε


cstatement ::= SESH
            | declare
            | expression SESH
            | VARIABLE '=' expression SESH
            | function_call SESH
            | DEKHAO '(' VARIABLE ')' SESH
            | if_condition '{' statements '}'
            | else_if_condition '{' statements '}'
            | else_condition '{' statements '}'
            | for_start '(' for_loop ')' '{' statements '}'
            | while_start '(' while_loop ')' '{' statements '}'
            | switch_start '(' switch_exp ')' '{' switch_statement '}'


/* Special Characters */
Delimiters ::=
   '('    /* Opening parenthesis */
   | ')'   /* Closing parenthesis */
```

```
| '{'   /* Opening brace */

| '}'   /* Closing brace */

| ','   /* Comma */

| '='   /* Assignment */

| '<'   /* For import statements */

| '>'   /* For import statements */

| ':'   /* For switch cases */
```