CS424 Compiler Construction

Assignment # 2
Report

Name: Muhammad Hamza Azeem
Reg No: 202029

Overview:

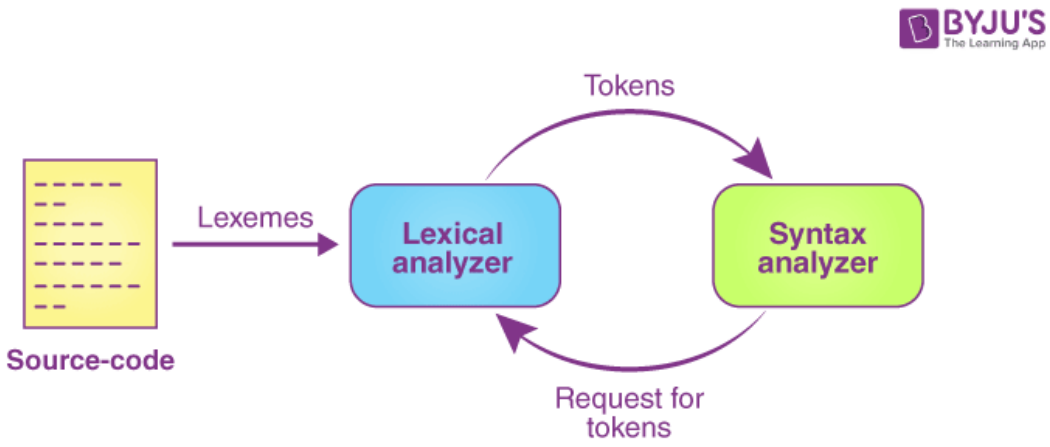
The evaluator is designed to handle a basic expression language, encompassing essential arithmetic functions, variables, and parentheses. It consists of two primary modules: Lexical analysis, implemented through Lex code, and Syntax analysis, implemented using Yacc code.

2. Lexical Analysis (Lex)

2.1 Token Specification

The Lex code defines the rules for tokenizing the input source code, covering numbers, variables, and operators.

```
[0-9]+ { yylval.num = atoi(yytext); return NUMERIC; }
[a-zA-Z]+ { yylval.str = strdup(yytext); return VAR; }
[ \t\n] ; // Ignore spaces
[-+*()] { return yytext[0]; }
. { printf("Error: Unidentified character %s\n", yytext); }
```



2. Syntax Analysis (Yacc)

3.1 Grammar Definition

The Yacc code establishes the grammatical structures for the expression language, specifying the organization of expressions, terms, and factors.

```
expr: term extExpr
;
extExpr: '+' term extExpr { printf("+ "); }
| '-' term extExpr { printf("- "); }
```

```
| /* void / { printf("\n"); }
;
term: factor extTerm
;
extTerm: '' factor extTerm { printf("* "); }
| /* void */ { }
;
factor: NUMERIC { printf("Num= %d ", $1); }
| VAR { printf("Var = %s ", $1); free($1); }
| '(' expr ')' { }
;
```

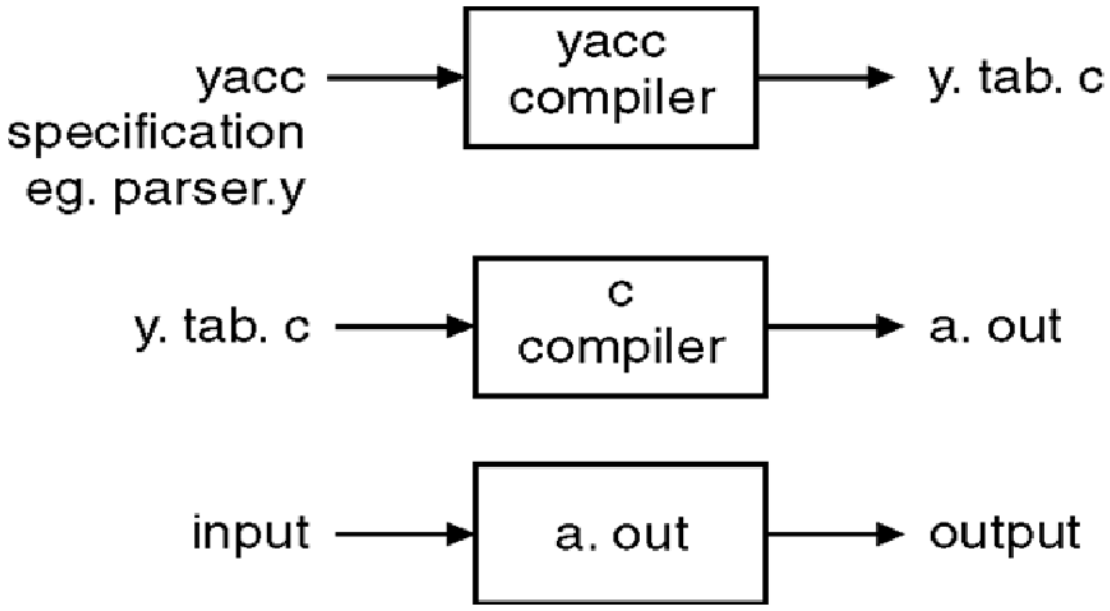
3.2 Concept of Abstract Syntax Tree (AST)

Although not explicitly stated in the provided code, the Yacc actions can be extended to construct an Abstract Syntax Tree (AST). This AST could be useful in subsequent processes or code generation.

Integration

The main function combines the Lex and Yacc components, initiating the parsing process.

```
int main() {
yyparse(); // Start parsing
return 0;
}
```



Summary:

The interpreter proficiently dissects and interprets expressions, generating output mirroring the input's structure. Future enhancements might include expanding the grammar for added functionalities, refining error-handling mechanisms, or integrating the generated AST for more advanced applications. Ensure this document stays current as the interpreter evolves and new features are incorporated.

Examples:

```
mx + b
Variable = mx Variable = b
```

```
a + b + 6 + 2
Variable = a Variable = b Number= 6 Number= 2
```