CS 4031
Compiler Construction
Lecture 6

Mahzaib Younas

Lecturer, Department of Computer Science

FAST, NUCES CFD

Context Free Grammar

A **context-free grammar (CFG)** is a formal system used to describe a class of languages known as **context-free languages (CFLs).**

- **Components of CFG**
  - **Terminals**
  - **Non-terminals**
  - **Production Rules**
  - **Start Symbol**

Lets consider the following C Code

```
#include <stdio.h>
int main() {
 int a = 5, b = 10, temp;
printf("Before swapping.\n");
printf("a = %d, b = %d\n", a, b);
// Swapping process
temp = a;
a = b;
b = temp;
printf("After swapping.\n");
printf("a = %d, b = %d\n", a, b);
return 0;
}
```

Lex Code

- "#include" {printf("%s is include keyowrd\n",yytext); }
- "<stdio.h>" {printf("%s is a library \n",yytext);}
- "int" {printf("%s is a int data type\n",yytext);}
- "main" {printf("%s Main \n",yytext);}
- "return" {printf("%s RETURN \n",yytext);}
- "printf" {printf("%s PRINTF n",yytext);}
- "(" {printf("%s LEFT_PAREN",yytext); }
- ")" {printf("%s RIGHT_PAREN \n",yytext); }
- "{" {printf("%s LEFT_BRACE\n",yytext);}
- "}" {printf("%s RIGHT_BRACE",yytext);}

Lex Code

- ";" {printf("%s SEMICOLON",yytext);}
- "," {printf("%s COMMA \n",yytext);}
- "=" {printf("%s ASSIGN\n",yytext);}
- "==" {printf("%s EQ",yytext);}
- [a-zA-Z_][a-zA-Z0-9_]* {printf("%s IDENTIFIER",yytext); }
- [0-9]+ {printf("%s NUMBER\n",yytext);}
- "\\n" {printf("%s NEWLINE\n",yytext);}
- “\[^\"]*\" {printf("%s STRING_LITERAL",yytext);}
- [\t]+ { /* ignore whitespace */ }
- . { PRINTF(“%s”, yytext[0]; }

CFG for above Code

- Program → function
- Function → type MAIN LEFT_PAREN RIGHT_PAREN block
- type → int
- block → LEFT_BRACE statements RIGHT_BRACE

CFG for above Code

- Statements → statement | statement statements
- statement → declaration SEMICOLON | assignment SEMICOLON | output SEMICOLON | RETURN NUMBER SEMICOLON
- declaration → type IDENTIFIER ASSIGN NUMBER |
  type IDENTIFIER COMMA INDENTIFIER ASSIGN NUMBER
- assignment → IDENTIFIER ASSIGN expression

CFG for above Code

- expression -> IDENTIFIER | NUMBER

- output -> PRINTF LEFT_PAREN STRING_LITERAL COMMA arguments RIGHT_PAREN |

  PRINTF LEFT_PAREN STRING_LITERAL RIGHT_PAREN

- arguments -> IDENTIFIER | IDENTIFIER COMMA arguments