Sir Syed University of Engineering and Technology

Compiler Construction Project

(Lexical Analyzer, Parse Tree, Grammar)

Group Members: Munib-Ul-Hassan (2019-CS-037)

Maaz Mohiuddin (2019-CS-042)

Section 5 A (Beta)

LEXICAL ANALYSER

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
bool isValidDelimiter(char ch) {
if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
ch == '[' || ch == ']' || ch == '{' || ch == '}')
return (true);
return (false);
}
bool isValidOperator(char ch){
if (ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == '>' || ch == '<' ||
ch == '=')
return (true);
return (false);
}
// Returns 'true' if the string is a VALID IDENTIFIER.
bool isvalidIdentifier(char* str){
if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
str[0] == '3' || str[0] == '4' || str[0] == '5' ||
str[0] == '6' || str[0] == '7' || str[0] == '8' ||
str[0] == '9' || isValidDelimiter(str[0]) == true)
return (false);
return (true);
}
bool isValidKeyword(char* str) {
if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") || !strcmp(str, "do") || !strcmp(str, "break") || !strcmp(str, "continue") || !strcmp(str, "int")
|| !strcmp(str, "double") || !strcmp(str, "float") || !strcmp(str, "return") || !strcmp(str, "char") || !strcmp(str, "case") || !strcmp(str, "char")
|| !strcmp(str, "sizeof") || !strcmp(str, "long") || !strcmp(str, "short") || !strcmp(str, "typedef") || !strcmp(str, "switch") || !strcmp(str, "unsigned")
|| !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") || !strcmp(str, "goto"))
return (true);
return (false);
}
bool isValidInteger(char* str) {
int i, len = strlen(str);
if (len == 0)
return (false);
for (i = 0; i < len; i++) {
if (str[i] != '0' && str[i] != '1' && str[i] != '2'&& str[i] != '3' && str[i] != '4' && str[i] != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9' || (str[i] == '-' && i > 0))
return (false);
}
```

```
46. return (true);
47. }
48. bool isRealNumber(char* str) {
49. int i, len = strlen(str);
50. bool hasDecimal = false;
51. if (len == 0)
52. return (false);
53. for (i = 0; i < len; i++) {
54. if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3' && str[i] != '4' && str[i] != '5' && str[i] != '6' &&
    str[i] != '7' && str[i] != '8'
55. && str[i] != '9' && str[i] != '.' || (str[i] == '-' && i > 0))
56. return (false);
57. if (str[i] == '.')
58. hasDecimal = true;
59. }
60. return (hasDecimal);
61. }
62. char* subString(char* str, int left, int right) {
63. int i;
64. char* subStr = (char*)malloc( sizeof(char) * (right - left + 2));
65. for (i = left; i <= right; i++)
66. subStr[i - left] = str[i];
67. subStr[right - left + 1] = '\0';
68. return (subStr);
69. }
70. void detectTokens(char* str) {
71. int left = 0, right = 0;
72. int length = strlen(str);
73. while (right <= length && left <= right) {
74. if (isValidDelimiter(str[right]) == false)
75. right++;
76. if (isValidDelimiter(str[right]) == true && left == right) {
77. if (isValidOperator(str[right]) == true)
78. printf("Valid operator : '%c'\n", str[right]);
79. right++;
80. left = right;
81. } else if (isValidDelimiter(str[right]) == true && left != right || (right == length && left != right)) {
82. char* subStr = subString(str, left, right - 1);
83. if (isValidKeyword(subStr) == true)
84. printf("Valid keyword : '%s'\n", subStr);
85. else if (isValidInteger(subStr) == true)
86. printf("Valid Integer : '%s'\n", subStr);
87. else if (isRealNumber(subStr) == true)
88. printf("Real Number : '%s'\n", subStr);
89. else if (isvalidIdentifier(subStr) == true
90. && isValidDelimiter(str[right - 1]) == false)
91. printf("Valid Identifier : '%s'\n", subStr);
92. else if (isvalidIdentifier(subStr) == false
93. && isValidDelimiter(str[right - 1]) == false)
94. printf("Invalid Identifier : '%s'\n", subStr);
95. left = right;
```

```
96. }
97. }
98. return;
99. }
100.  int main(){
101.  char str[100] = " void mul(int a,int b){int product = a*b; return  product”}
102.  printf("The Program is : '%s' \n", str);
103.  printf("All Tokens are : \n");
104.  detectTokens(str);
105.  return (0);
106.  }
```

Output

```
The Program is : 'void mul (int a,int b){int product product = a*b; return (product);}'
All Tokens are :
Valid keyword : 'void'
Valid Identifier : 'mul'
Valid keyword : 'int'
Valid Identifier : 'a'
Valid keyword : 'int'
Valid Identifier : 'b'
Valid keyword : 'int'
Valid Identifier : 'product'
Valid Identifier : 'product'
Valid operator : '='
Valid Identifier : 'a'
Valid operator : '*'
Valid Identifier : 'b'
Valid keyword : 'return'
Valid Identifier : 'product'
```

CODE

```
void mul (int a,int b)

{

int product

product = a*b;

return (product);

}
```

GRAMMAR

Program -> Begin body end

Body -> Stmts

Statements -> Stmt- list

Stmt-list -> Func-stmt|dec-stmt|prod-stmt |ret-stmt

Func-stmt -> dt fun.name ((parameters)*){statements}

dt-> int|float|string

fun.name=alpha

parameters ->dt var(,)*

dec-stmt ->dt var;

prod-stmt -> var assign-op var arith-op var

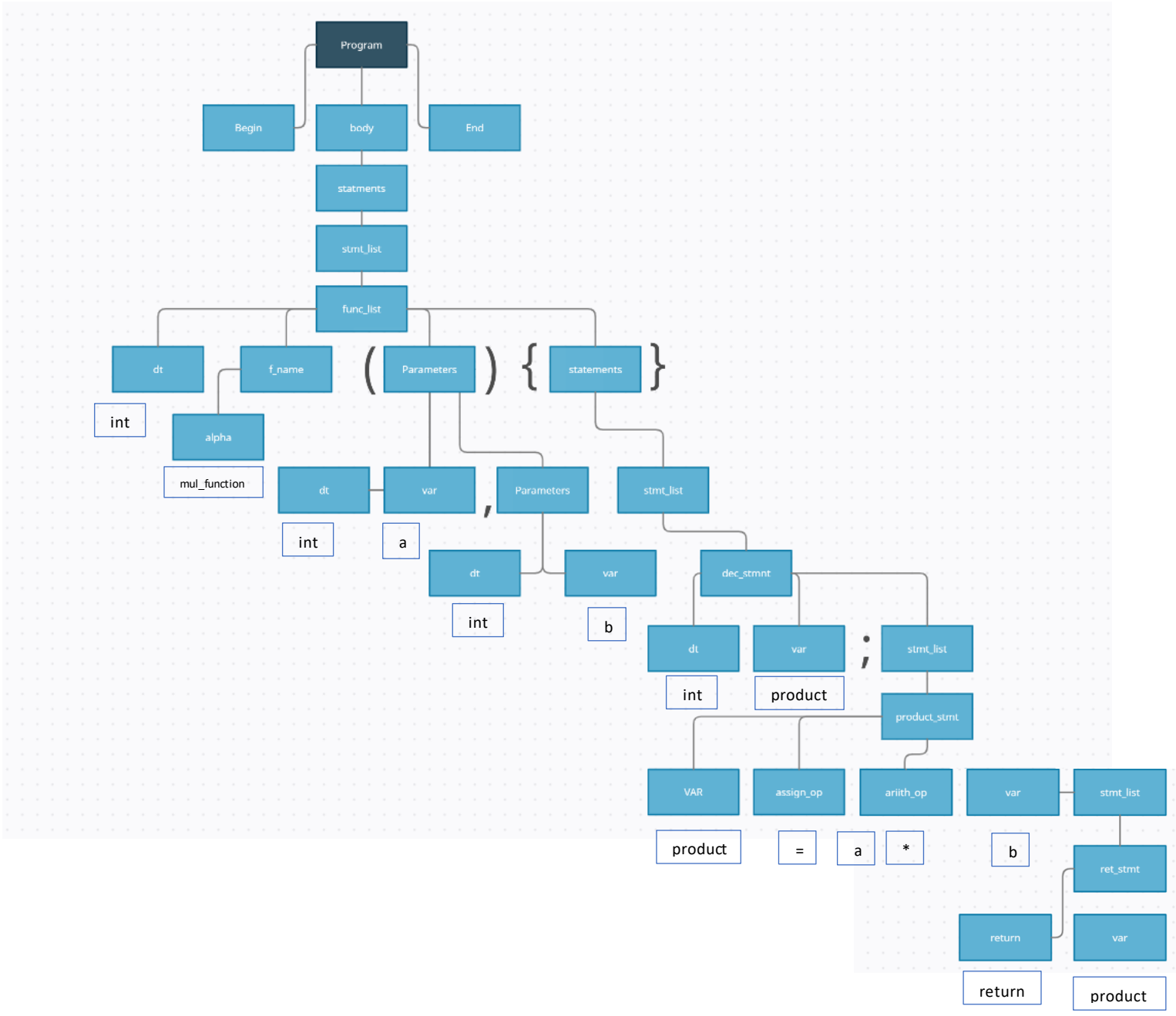
var -> letters (letter|digits)

assign-op -> =

arith-op -> +|-|/|*

ret-stmt ->return (0|var);

PARSE TREE


Program
Begin
body
End
statments
stmt_list
func_list
dt
f_name
(
Parameters
)
{
statements
}
int
alpha
mul_function
dt
var
,
Parameters
stmt_list
int
a
dt
var
dec_stmnt
int
b
dt
var
;
stmt_list
int
product
product_stmt
VAR
assign_op
ariith_op
var
stmt_list
product
=
a
*
b
ret_stmt
return
var
return
product