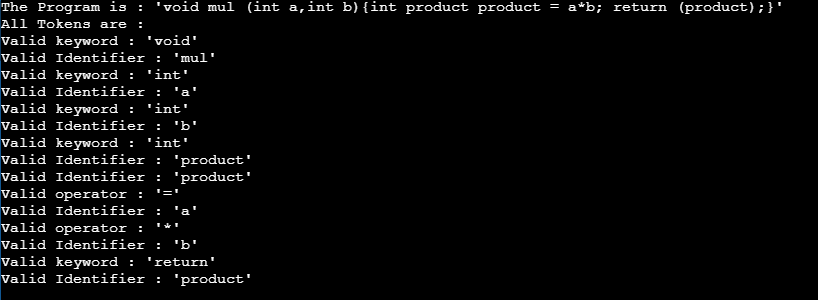
LEXICAL ANALYSER

1. #include <stdbool.h>
2. #include <stdio.h>
3. #include <string.h>
4. #include <stdlib.h>
5. bool isValidDelimiter(char ch) {
6. if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' ||
7. ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
8. ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
9. ch == '[' || ch == ']' || ch == '{' || ch == '}')
10. return (true);
11. return (false);
12. }
13. bool isValidOperator(char ch){
14. if (ch == '+' || ch == '-' || ch == '\*' ||
15. ch == '/' || ch == '>' || ch == '<' ||
16. ch == '=')
17. return (true);
18. return (false);
19. }
20. // Returns 'true' if the string is a VALID IDENTIFIER.
21. bool isvalidIdentifier(char\* str){
22. if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
23. str[0] == '3' || str[0] == '4' || str[0] == '5' ||
24. str[0] == '6' || str[0] == '7' || str[0] == '8' ||
25. str[0] == '9' || isValidDelimiter(str[0]) == true)
26. return (false);
27. return (true);
28. }
29. bool isValidKeyword(char\* str) {
30. if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") || !strcmp(str, "do") ||    !strcmp(str, "break") || !strcmp(str, "continue") || !strcmp(str, "int")
31. || !strcmp(str, "double") || !strcmp(str, "float") || !strcmp(str, "return") || !strcmp(str,    "char") || !strcmp(str, "case") || !strcmp(str, "char")
32. || !strcmp(str, "sizeof") || !strcmp(str, "long") || !strcmp(str, "short") || !strcmp(str, "typedef") || !strcmp(str, "switch") || !strcmp(str, "unsigned")
33. || !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") || !strcmp(str, "goto"))
34. return (true);
35. return (false);
36. }
37. bool isValidInteger(char\* str) {
38. int i, len = strlen(str);
39. if (len == 0)
40. return (false);
41. for (i = 0; i < len; i++) {
42. if (str[i] != '0' && str[i] != '1' && str[i] != '2'&& str[i] != '3' && str[i] != '4' && str[i] != '5'
43. && str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9' || (str[i] == '-' && i > 0))
44. return (false);
45. }
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



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

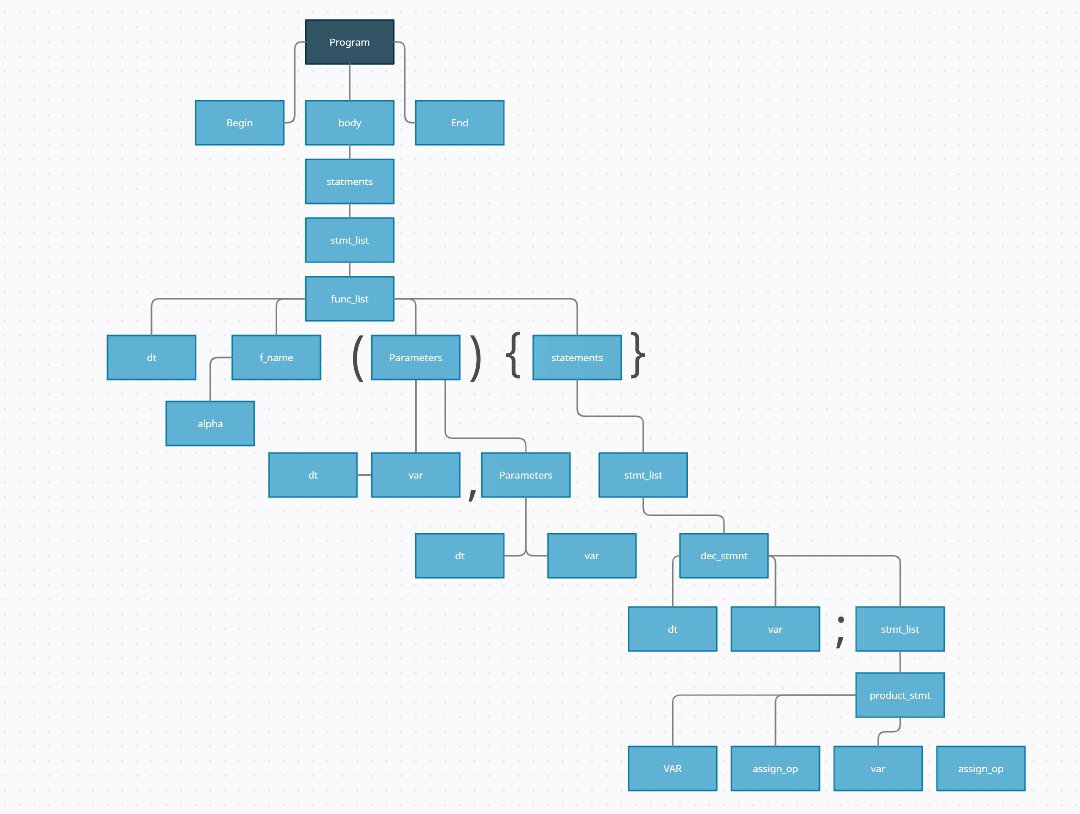
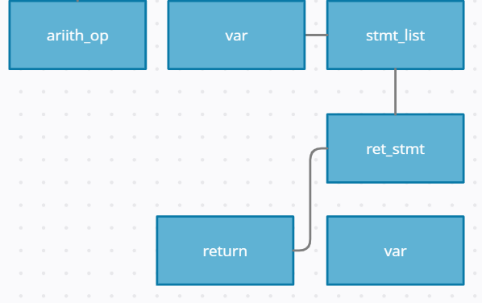
var -> letters (letter|digits)

assign-op -> =

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

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

PARSE TREE



product

return

\*

b

a

=

product

product

int

b

int

a

int

mul\_function

int