Lab#:	17
Topics :	Shift-Reduce Parsing
Objectives:	

The main goal of this lab is to study design issues to be considered for developing a lexical analyzer application such as the following:

Write a program to the end of the operation of the shift-reduce parser there can be traced in reverse the rightmost derivation of the input string according to the grammar. The grammar used in this program is

E->E+E

E->E*E

E->(E)

E->id

This program works for all possible input strings. Let's take the input string (a*b) + c or anything.

Tasks:

- Start the program.
- Get the input string from the user.
- Push $ onto top of the stack.
- Set ip to point to the first input symbol.
- If there is any production which can be used to reduce the input symbol reduce the string otherwise push it to the top of the stack.
- Set ip to point to next input symbol.
- Repeat the above steps until the top of the stack contains the $ and the starting symbol. If so, then the string is valid, otherwise the string is invalid, return an error message.
- Stop the program.

Program:

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
boolisValidDelimiter(char ch)
{
```

```
if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||   ch == '/' || ch == ',' || ch == ';' || ch
==
'>' ||ch == '<' || ch == '=' || ch == '(' || ch == ')' ||ch == '[' || ch == ']' || ch == '{' ||
ch == '}')
return (true);
return (false);
}
boolisValidOperator(char ch)
{
if (ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == '>' || ch == '<' ||
ch == '=')
return (true);
return (false);
}

boolisvalidIdentifier(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);
}
boolisValidKeyword(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);
}
boolisValidInteger(char* str)
{
inti, 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);
        }
return (true);
}
boolisRealNumber(char* str)
{
inti, len = strlen(str);
boolhasDecimal = false;
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]
!= '.' || (str[i] == '-' &&i> 0))
return (false);
if (str[i] == '.')
hasDecimal = true;
  }
return (hasDecimal);
}
char* subString(char* str, int left, int right)
{
inti;
char* subStr = (char*)malloc( sizeof(char) * (right - left + 2));
for (i = left; i<= right; i++)
subStr[i - left] = str[i];
subStr[right - left + 1] = '\0';
return (subStr);
}
voiddetectTokens(char* str)
{
int left = 0, right = 0;
int length = strlen(str);
while (right <= length && left <= right)
{
if (isValidDelimiter(str[right]) == false)
right++;
if (isValidDelimiter(str[right]) == true && left == right) {
if (isValidOperator(str[right]) == true)
printf("Valid operator : '%c'\n", str[right]);
right++;
left = right;
}
else if (isValidDelimiter(str[right]) == true && left != right || (right == length &&
left !=       right))
{
char* subStr = subString(str, left, right - 1);
if (isValidKeyword(subStr) == true)
printf("Valid keyword : '%s'\n", subStr);
else if (isValidInteger(subStr) == true)
printf("Valid Integer : '%s'\n", subStr);
else if (isRealNumber(subStr) == true)
printf("Real Number : '%s'\n", subStr);
else if (isvalidIdentifier(subStr) == true
&&isValidDelimiter(str[right - 1]) == false)
printf("Valid Identifier : '%s'\n", subStr);
else if (isvalidIdentifier(subStr) == false
&&isValidDelimiter(str[right - 1]) == false)
printf("Invalid Identifier : '%s'\n", subStr);
left = right;
    }
  }
return;
}


int main(){
charstr[100];
printf("Enter the String: ");
gets(str);
printf("The Program is : '%s' \n", str);
printf("All Tokens are : \n");
detectTokens(str);
return (0);
}
```

Input:

```
Enter the String: (a*b) + c
```

Output:

```
The Program is          : '(a*b) + c'
All Tokens are:
Valid Identifier        : 'a'
Valid operator          : '*'
Valid Identifier        : 'b'
Valid operator          : '+'
Valid Identifier        : 'c'
```