LANGUAGE TRANSLATOR

Scanner phase implementation of assigned project in "C" language.

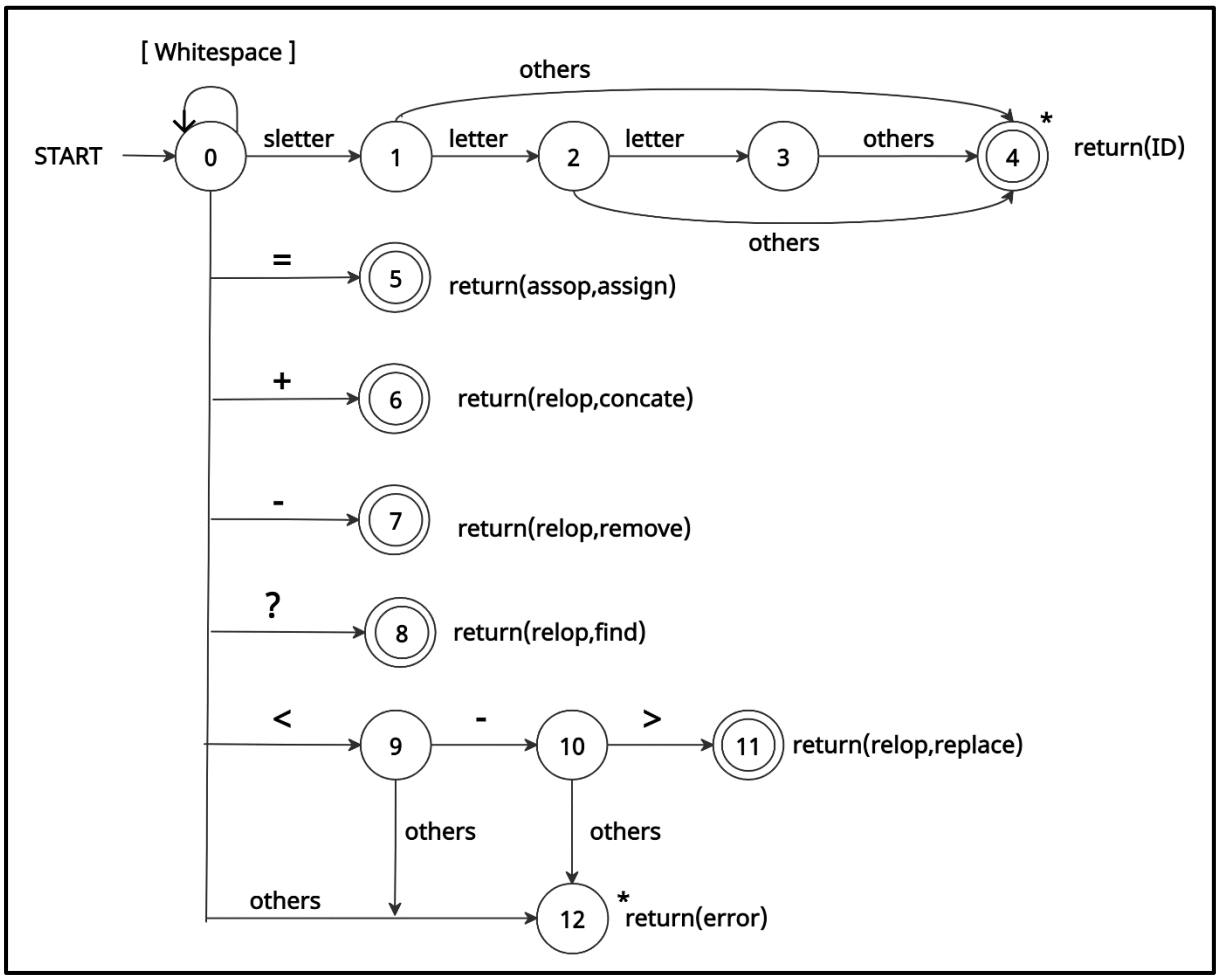
Regular Expressions:

letter --> [a-z A-Z]
sletter --> [a-z]
assop --> =
relop --> + | - | <-> | ?
singlequote --> ‘
doublequote--> “
string --> string
char --> char
int --> int
id --> sletter(letter)?(letter)?
whitespace --> (space|tab|newline)*

Token Table:

Regular Expression	TOKEN	Attribute-Value
ws	-	-
id	id	pointer to table entry
string	string	-
char	char	-
int	Int	-
=	assop	assign
+	relop	concate
-	relop	remove
<->	relop	replace
?	relop	find
‘	singlequote	quotes
“	doublequote	quotes

DFA:


[Whitespace]
others
START
0
sletter
1
letter
2
letter
3
others
4
*
return(ID)
others
=
5
return(assop,assign)
+
6
return(relop,concate)
-
7
return(relop,remove)
?
8
return(relop,find)
<
9
-
10
>
11
return(relop,replace)
others
others
others
12
*
return(error)

Algorithm:

```
Lexer()
{
   input(c);
   state=0;
   while(c!=EOF)
   {
      switch(state)
      {
         case 0: if(c==' ' || c=='\t' || c=='\n')    state=0;
                 else if(c == sletter) state = 1;
                 else if(c == '=') state = 5;
                 else if(c == '+') state = 6;
                 else if(c == '-') state = 7;
                 else if(c == '?') state = 8;
                 else if(c == '<') state = 9;
                 else state = 12;
                 break;

         case 1: input(c);
                 if(c == letter)  state = 2;
                 else state = 4;
                 break;

         case 2: input(c);
                 if(c == letter)     state=3;
                 else state = 4;
                 break;

         case 3: input(c);
                 if(c == other)  state=4;
                 break;
```

```
        case 4: state=0;
                unput(c);
                return(ID);

        case 5: state = 0;
                return(assop,assign);

        case 6: state = 0;
                return(relop,concate);

        case 7: state = 0;
                return(relop,remove);

        case 8: state = 0;
                return(relop,find);

        case 9: input(c);
                if(c == '-')            state = 10;
                else state=12;
                break;

        case 10: input(c);
                 if(c == '>')           state = 11;
                 else state=12;
                 break;

        case 11: state = 0;
                 return(relop,replace);

        case 12: unput(c);
                 return(error);
      }
   }
}
```