AIM: DFA and Algorithm for below language.

Valid operations:

1. x=p+q Concatenates the p with q and stores result in x
2. x=p-q Removes the last “q” symbols from p and stores answer back in x.
3. x=p<->q Replaces all occurrence of p variable with q variable in x string.
4. pos=x?p Find and return the first position in X where q string was found

● Variables can be of type: “string” or “char”

● Variables can be assigned value using “=” symbol.

● Variable names should start with “small case character” & can’t contain integer.

● The maximum size of name can be 3.

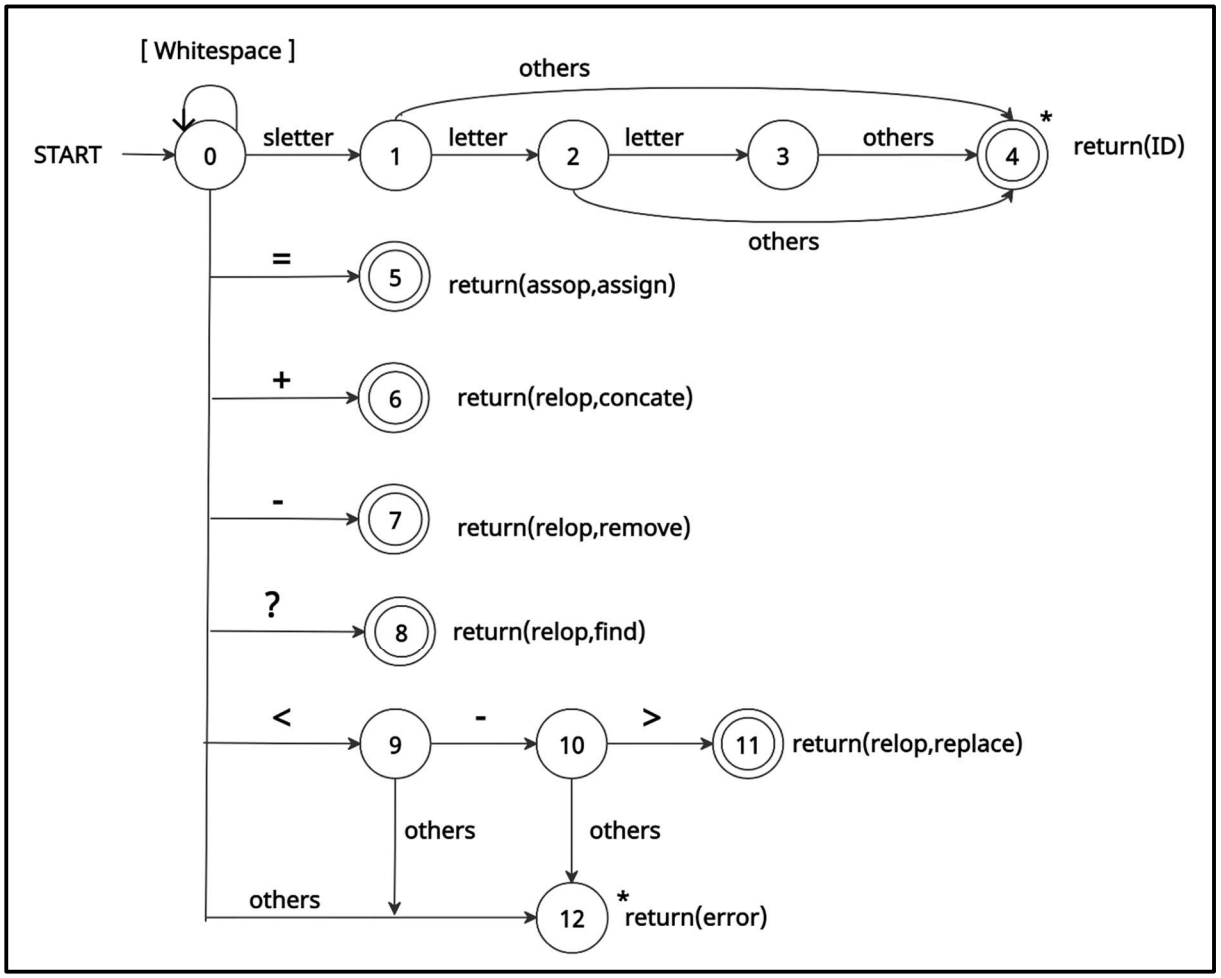
Regular Expressions:

letter	-->	[a-z A-Z]
sletter	-->	[a-z]
assop	-->	=
relop	-->	+ | - | <-> | ?
string	-->	string
char	-->	char
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	-
=	assop	assign
+	relop	concate
-	relop	remove
<->	relop	replace
?	relop	find

DFA:



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);
      }
   }
}
```