Date: 07/08/2024

Experiment 1.1

AIM

To develop lexical analyzer using C

ALGORITHM

1. Start
2. Define token type and names. For C, we are considering Keywords, Identifiers, Integer constants, String literals, Operators, Punctuators, Compiler directives and token representing end of input.
3. Define arrays containing list of operators, keywords, and punctuators.
4. Define helper function isInteger to verify whether a string is a valid integer.
5. Define helper function isPunctuator to verify whether a character is a punctuator.
6. Define helper function isKeyword to verify whether a string is a keyword.
7. Define helper function isOperator to verify whether a string is an operator.
8. Use the above functions to define helper function identifierParse to determine if a given buffer contains a keyword, integer constant, a valid identifier, or is unknown.
9. Define function getToken to extract token from stdin as follows:
    1. Reset the buffer.
    2. Read characters one by one using getchar and do the following:
        1. If current caracter is EOF:
            1. If buffer is empty, return END token.
            2. Else, parse buffer and return token using identifierParse function.
        2. If the input is at a newline starting from #, this indicates a compiler directive.
            1. Read the whole line and store in buffer.
            2. Return DIRECTIVE token.
        3. If the current character is a punctuator:
            1. If buffer is empty, store punctuator in buffer and return PUNCTUATOR token.
            2. Else, move input pointer backward and parse and return current contents of the buffer using identifierParse.

4. If current character is white space:
    1. If buffer is empty, ignore the character and go to next iteration of the loop.
    2. Else, parse and return current contents of the buffer using identifierParse.
5. If the current character is a forward slash:
    1. Look ahead one character to see if this is beginning of a comment.
    2. If next character is a ‘/’, then read till newline and skip the characters.
    3. If next character is a ‘*’, read till “*/” is encountered and skip the characters.
    4. If no comment is found, move input pointer backward so the next character can be processed as usual. Otherwise, go to next iteration of the loop.
6. Add current character to buffer.
7. If buffer contains a valid operator:
    1. Read into buffer toll it contains at least 3 characters (The maximum length of an operator)
    2. Remove last character and move input pointer backward until the buffer has a valid operator.
    3. This ensures that larger operators are considered first.
8. If the current character is a double quote:
    1. Read characters and store in buffer till another double quote character is encountered.
    2. Return token STRING_LITERAL.
10. Create main function to continuously get tokens until TOKEN_END is returned and print the type and value of each token.
11. Stop

RESULT

Successfully performed lexical analysis of given C program.