ASSIGNMENT

**Write a c program Create a simple analyzer reading a input string, identify tokens like keywords, identifiers, operators etc categorizing them and then outputing the list of tokens along with their category.**

**SOLUTION:**

|  |
| --- |
| CODE:-  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <ctype.h>  #define MAX\_LENGTH 100  #define MAX\_KEYWORDS 10  // Function declarations  void tokenize(char \*input);  int isKeyword(char \*word);  int isIdentifier(char \*word);  int isOperator(char ch);  void printToken(char \*token, char \*type);  // Keywords list  const char \*keywords[MAX\_KEYWORDS] = {  "int", "char", "if", "else", "while", "return", "for", "void", "main", "break"  };  int main() {  char input[MAX\_LENGTH];  printf("Enter the C code (end with a single line with 'exit'):\n");  while (1) {  char line[MAX\_LENGTH];  fgets(line, sizeof(line), stdin);  if (strcmp(line, "exit\n") == 0) {  break;  }  strcat(input, line);  }  printf("\nTokenizing the input...\n");  tokenize(input);  return 0;  }  // Function to tokenize input  void tokenize(char \*input) {  char \*token = strtok(input, " \t\n();,");  while (token != NULL) {  // Check for keywords  if (isKeyword(token)) {  printToken(token, "Keyword");  }  // Check for identifiers  else if (isIdentifier(token)) {  printToken(token, "Identifier");  }  // Check for operators  else if (isOperator(token[0])) {  printToken(token, "Operator");  }  // Check for literals (numbers)  else if (isdigit(token[0])) {  printToken(token, "Literal");  }  // Not recognized  else {  printToken(token, "Unknown");  }  token = strtok(NULL, " \t\n();,");  }  }  // Function to check if the token is a keyword  int isKeyword(char \*word) {  int i;  for ( i = 0; i < MAX\_KEYWORDS; i++) {  if (strcmp(word, keywords[i]) == 0) {  return 1; // Found keyword  }  }  return 0; // Not a keyword  }  // Function to check if the token is a valid identifier  int isIdentifier(char \*word) {  if (!isalpha(word[0]) && word[0] != '\_') {  return 0; // Must start with a letter or underscore  }  int i;  for ( i = 1; word[i] != '\0'; i++) {  // Check if the character is not a valid part of an identifier  if (!isalnum(word[i]) && word[i] != '\_') {  return 0; // Invalid character  }  }  return 1; // Valid identifier  }  // Function to check if the character is an operator  int isOperator(char ch) {  switch (ch) {  case '+':  case '-':  case '\*':  case '/':  case '=':  case '<':  case '>':  case '&':  case '|':  return 1; // It's an operator  }  return 0; // Not an operator  }  // Function to print the token and its type  void printToken(char \*token, char \*type) {  printf("Token: %-15s Type: %s\n", token, type);  }  Output:- |