

Compiler Design Lab Assessment 1

Name: Karan Sehgal

Registration number: 22BCE3939

Q1: Write a program analysing string functions.

The program includes the implementation of common string operations such as concatenation, substring extraction, case conversion, trimming, splitting. The program will analyze the performance and utility of these functions with different types and lengths of strings, providing clear output and insights on their behavior in different scenarios.

CODE:-

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void toUpperCase(char* str) {
    for (int i = 0; str[i] != '\0'; i++) {
        str[i] = toupper(str[i]);
    }
}

void stringCopy(char* dest, const char* src) {
    while ((*dest++ = *src++) != '\0');
}

int main() {
    char str1[100], str2[100], str3[200], strCopy[100];

    printf("Enter the first string (no spaces): ");
    scanf("%99s", str1);

    printf("Enter the second string (no spaces): ");
    scanf("%99s", str2);

    printf("\nLength of first string: %lu\n", strlen(str1));
    printf("Length of second string: %lu\n", strlen(str2));

    stringCopy(str3, str1);
    strcat(str3, str2);
    printf("\nConcatenated string: %s\n", str3);

    toUpperCase(str1);
    toUpperCase(str2);
    printf("\nUppercase of first string: %s\n", str1);
    printf("Uppercase of second string: %s\n", str2);

    stringCopy(strCopy, str1);
    printf("\nCopied first string: %s\n", strCopy);
```

```
return 0;}
```

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  void toUpperCase(char* str) {
6      for (int i = 0; str[i] != '\0'; i++) {
7          str[i] = toupper(str[i]);
8      }
9  }
10 void stringCopy(char* dest, const char* src) {
11     while ((*dest++ = *src++) != '\0');
12 }
13 int main() {
14     char str1[100], str2[100], str3[200], strCopy[100];
15
16     printf("Enter the first string (no spaces): ");
17     scanf("%99s", str1);
18
19     printf("Enter the second string (no spaces): ");
20     scanf("%99s", str2);
21
22     printf("\nLength of first string: %lu\n", strlen(str1));
23     printf("Length of second string: %lu\n", strlen(str2));
24
25
26     stringCopy(str3, str1);
27     strcat(str3, str2);
28     printf("\nConcatenated string: %s\n", str3);
29
30     toUpperCase(str1);
31     toUpperCase(str2);
32     printf("\nUppercase of first string: %s\n", str1);
33     printf("Uppercase of second string: %s\n", str2);
34
35     stringCopy(strCopy, str1);
36     printf("\nCopied first string: %s\n", strCopy);
37
38     return 0;
39 }
40
```

OUTPUT:

/home/karan/Documents/CPP/string1

```
Enter the first string (no spaces): Hello
Enter the second string (no spaces): Karan

Length of first string: 5
Length of second string: 5

Concatenated string: HelloKaran

Uppercase of first string: HELLO
Uppercase of second string: KARAN

Copied first string: HELLO

Process returned 0 (0x0)   execution time : 15.786 s
Press ENTER to continue.
```

/home/karan/Documents/CPP/string1

```
Enter the first string (no spaces): OffYou
Enter the second string (no spaces): GO

Length of first string: 6
Length of second string: 2

Concatenated string: OffYouGO

Uppercase of first string: OFFYOU
Uppercase of second string: GO

Copied first string: OFFYOU

Process returned 0 (0x0)   execution time : 31.335 s
Press ENTER to continue.
```

Q2: Write a code to specify the tokens in a code.

The objective of this program is to analyze a given piece of Python code and categorize its components into different types of tokens. These tokens include keywords, identifiers, operators, literals, and symbols. By breaking down the code into these elements, the program aims to provide a detailed understanding of the structure and components of the code, which can be useful for code analysis, debugging, and educational purposes.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_LENGTH 1000
#define MAX_LINES 100

enum TokenType {
    KEYWORD,
    OPERATOR,
    CONSTANT,
    PUNCTUATION,
    IDENTIFIER
};

struct Token {
    enum TokenType type;
    char value[100];
};

int isKeyword(const char* str);
int isOperator(char ch);
int isPunctuation(char ch);
void tokenize(const char* input, struct Token* tokens, int* tokenCount);
void printResults(struct Token* tokens, int tokenCount);

int main() {
    char input[MAX_LINES * MAX_LENGTH] = {0};
    char line[MAX_LENGTH];
    int lineCount = 0;

    printf("Enter a string of C code (enter an empty line to stop):\n");

    while (lineCount < MAX_LINES) {
        if (fgets(line, MAX_LENGTH, stdin) == NULL) {
            break;
        }

        line[strcspn(line, "\n")] = '\0';
```

```
if (strlen(line) == 0) {  
    break;  
}
```

```
strcat(input, line);  
strcat(input, " ");  
lineCount++;  
}
```

```
struct Token tokens[MAX_LENGTH];  
int tokenCount = 0;
```

```
tokenize(input, tokens, &tokenCount);  
printResults(tokens, tokenCount);
```

```
return 0;  
}
```

```
int isKeyword(const char* str) {
```

```
    const char* keywords[] = {  
        "auto", "break", "case", "char", "const", "continue", "default",  
        "do", "double", "else", "enum", "extern", "float", "for", "goto",  
        "if", "inline", "int", "long", "register", "restrict", "return",  
        "short", "signed", "sizeof", "static", "struct", "switch", "typedef",  
        "union", "unsigned", "void", "volatile", "while", "_Alignas",  
        "_Alignof", "_Atomic", "_Bool", "_Complex", "_Decimal128",  
        "_Decimal32", "_Decimal64", "_Generic", "_Imaginary", "_Noreturn",  
        "_Static_assert", "_Thread_local"  
    };
```

```
const int numKeywords = sizeof(keywords) / sizeof(keywords[0]);
```

```
for (int i = 0; i < numKeywords; ++i) {  
    if (strcmp(str, keywords[i]) == 0) {  
        return 1;  
    }  
}  
return 0;  
}
```

```
int isOperator(char ch) {  
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '=' || ch == '<' || ch == '>');  
}
```

```
int isPunctuation(char ch) {  
    return (ch == '(' || ch == ')' || ch == '{' || ch == '}' || ch == ';' || ch == ',' || ch == '.');
```

```
}
```

```
void tokenize(const char* input, struct Token* tokens, int* tokenCount) {  
    char buffer[100];  
    int bufferIndex = 0;  
    *tokenCount = 0;
```

```
    for (int i = 0; input[i] != '\0'; i++) {  
        if (isalnum(input[i]) || input[i] == '_') {  
            buffer[bufferIndex++] = input[i];  
        } else {  
            if (bufferIndex > 0) {  
                buffer[bufferIndex] = '\0';  
                if (isKeyword(buffer)) {  
                    tokens[*tokenCount].type = KEYWORD;  
                } else if (isdigit(buffer[0])) {  
                    tokens[*tokenCount].type = CONSTANT;  
                } else {  
                    tokens[*tokenCount].type = IDENTIFIER;  
                }  
                strcpy(tokens[*tokenCount].value, buffer);  
                (*tokenCount)++;  
                bufferIndex = 0;  
            }  
        }  
    }
```

```
    if (isOperator(input[i])) {  
        tokens[*tokenCount].type = OPERATOR;  
        tokens[*tokenCount].value[0] = input[i];  
        tokens[*tokenCount].value[1] = '\0';  
        (*tokenCount)++;  
    } else if (isPunctuation(input[i])) {  
        tokens[*tokenCount].type = PUNCTUATION;  
        tokens[*tokenCount].value[0] = input[i];  
        tokens[*tokenCount].value[1] = '\0';  
        (*tokenCount)++;  
    } else if (input[i] == '"') {  
        int j = i + 1;  
        while (input[j] != '"' && input[j] != '\0') {  
            j++;  
        }  
        if (input[j] == '"') {  
            tokens[*tokenCount].type = IDENTIFIER;  
            tokens[*tokenCount].value[0] = '"';  
            strncpy(tokens[*tokenCount].value + 1, input + i + 1, j - i - 1);  
            tokens[*tokenCount].value[j - i] = '"';  
            tokens[*tokenCount].value[j - i + 1] = '\0';  
            (*tokenCount)++;  
            i = j;  
        }  
    }  
}
```

```
}  
}  
}
```

```
void printResults(struct Token* tokens, int tokenCount) {  
    int keywordCount = 0, operatorCount = 0, constantCount = 0, punctuationCount = 0,  
    identifierCount = 0;
```

```
    char seenKeywords[MAX_LENGTH][100] = {0};  
    char seenOperators[MAX_LENGTH][100] = {0};  
    char seenConstants[MAX_LENGTH][100] = {0};  
    char seenPunctuations[MAX_LENGTH][100] = {0};  
    char seenIdentifiers[MAX_LENGTH][100] = {0};
```

```
    printf("Keywords : [");  
    for (int i = 0; i < tokenCount; i++) {  
        if (tokens[i].type == KEYWORD) {  
            int isNew = 1;  
            for (int j = 0; j < keywordCount; j++) {  
                if (strcmp(seenKeywords[j], tokens[i].value) == 0) {  
                    isNew = 0;  
                    break;  
                }  
            }  
            if (isNew) {  
                if (keywordCount > 0) {  
                    printf(", ");  
                }  
                printf("%s", tokens[i].value);  
                strcpy(seenKeywords[keywordCount++], tokens[i].value);  
            }  
        }  
    }  
    printf("]");
```

```
    printf("\n-----\n");  
    printf("Operators : [");  
    for (int i = 0; i < tokenCount; i++) {  
        if (tokens[i].type == OPERATOR) {  
            int isNew = 1;  
            for (int j = 0; j < operatorCount; j++) {  
                if (strcmp(seenOperators[j], tokens[i].value) == 0) {  
                    isNew = 0;  
                    break;  
                }  
            }  
            if (isNew) {  
                if (operatorCount > 0) {  
                    printf(", ");  
                }  
            }  
        }  
    }
```

```
printf("%s", tokens[i].value);
strcpy(seenOperators[operatorCount++], tokens[i].value);
}
}
printf("]");
```

```
printf("\n-----\n");
printf("Constants : [");
for (int i = 0; i < tokenCount; i++) {
    if (tokens[i].type == CONSTANT) {
        int isNew = 1;
        for (int j = 0; j < constantCount; j++) {
            if (strcmp(seenConstants[j], tokens[i].value) == 0) {
                isNew = 0;
                break;
            }
        }
        if (isNew) {
            if (constantCount > 0) {
                printf(", ");
            }
            printf("%s", tokens[i].value);
            strcpy(seenConstants[constantCount++], tokens[i].value);
        }
    }
}
printf("]");
```

```
printf("\n-----\n");
printf("Punctuations : [");
for (int i = 0; i < tokenCount; i++) {
    if (tokens[i].type == PUNCTUATION) {
        int isNew = 1;
        for (int j = 0; j < punctuationCount; j++) {
            if (strcmp(seenPunctuations[j], tokens[i].value) == 0) {
                isNew = 0;
                break;
            }
        }
        if (isNew) {
            if (punctuationCount > 0) {
                printf(", ");
            }
            printf("%s", tokens[i].value);
            strcpy(seenPunctuations[punctuationCount++], tokens[i].value);
        }
    }
}
printf("]");
```



```

printf("\n-----\n");
printf("Identifiers : [");
for (int i = 0; i < tokenCount; i++) {
    if (tokens[i].type == IDENTIFIER) {
        int isNew = 1;
        for (int j = 0; j < identifierCount; j++) {
            if (strcmp(seenIdentifiers[j], tokens[i].value) == 0) {
                isNew = 0;
                break;
            }
        }
        if (isNew) {
            if (identifierCount > 0) {
                printf(", ");
            }
            printf("%s", tokens[i].value);
            strcpy(seenIdentifiers[identifierCount++], tokens[i].value);
        }
    }
}
printf("]");

printf("\n-----\n");
printf("Total number of tokens are %d\n", tokenCount);
}

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX_LENGTH 1000
7  #define MAX_LINES 100
8
9  enum TokenType {
10     KEYWORD,
11     OPERATOR,
12     CONSTANT,
13     PUNCTUATION,
14     IDENTIFIER
15 };
16
17 struct Token {
18     enum TokenType type;
19     char value[100];
20 };
21
22 int isKeyword(const char* str);
23 int isOperator(char ch);
24 int isPunctuation(char ch);
25 void tokenize(const char* input, struct Token* tokens, int* tokenCount);
26 void printResults(struct Token* tokens, int tokenCount);
27
28 int main() {
29     char input[MAX_LINES * MAX_LENGTH] = {0};
30     char line[MAX_LENGTH];
31     int lineCount = 0;
32
33     printf("Enter a string of C code (enter an empty line to stop):\n");
34
35     while (lineCount < MAX_LINES) {
36         if (fgets(line, MAX_LENGTH, stdin) == NULL) {
37             break;
38         }
39
40         line[strcspn(line, "\n")] = '\0';
41
42         if (strlen(line) == 0) {
43             break;
44         }
45
46         strcat(input, line);
47         strcat(input, " ");
48         lineCount++;
49     }
50
51     struct Token tokens[MAX_LENGTH];
52     int tokenCount = 0;

```



```

95 void tokenize(const char* input, struct Token* tokens, int* tokenCount) {
96     int bufferIndex = 0;
97     *tokenCount = 0;
98
99     for (int i = 0; input[i] != '\0'; i++) {
100         if (isalnum(input[i]) || input[i] == '_') {
101             buffer[bufferIndex++] = input[i];
102         } else {
103             if (bufferIndex > 0) {
104                 buffer[bufferIndex] = '\0';
105                 if (isKeyword(buffer)) {
106                     tokens[*tokenCount].type = KEYWORD;
107                 } else if (isdigit(buffer[0])) {
108                     tokens[*tokenCount].type = CONSTANT;
109                 } else {
110                     tokens[*tokenCount].type = IDENTIFIER;
111                 }
112                 strcpy(tokens[*tokenCount].value, buffer);
113                 (*tokenCount)++;
114                 bufferIndex = 0;
115             }
116
117             if (isOperator(input[i])) {
118                 tokens[*tokenCount].type = OPERATOR;
119                 tokens[*tokenCount].value[0] = input[i];
120                 tokens[*tokenCount].value[1] = '\0';
121                 (*tokenCount)++;
122             } else if (isPunctuation(input[i])) {
123                 tokens[*tokenCount].type = PUNCTUATION;
124                 tokens[*tokenCount].value[0] = input[i];
125                 tokens[*tokenCount].value[1] = '\0';
126                 (*tokenCount)++;
127             } else if (input[i] == '"') {
128                 int j = i + 1;
129                 while (input[j] != '"' && input[j] != '\0') {
130                     j++;
131                 }
132                 if (input[j] == '"') {
133                     tokens[*tokenCount].type = IDENTIFIER;
134                     tokens[*tokenCount].value[0] = '"';
135                     strncpy(tokens[*tokenCount].value + 1, input + i + 1, j - i - 1);
136                     tokens[*tokenCount].value[j - i] = '"';
137                     tokens[*tokenCount].value[j - i + 1] = '\0';
138                     (*tokenCount)++;
139                     i = j;
140                 }
141             }
142         }
143     }
144 }
145
146

```



```

146
147 void printResults(struct Token* tokens, int tokenCount) {
148     int keywordCount = 0, operatorCount = 0, constantCount = 0, punctuationCount = 0, identifierCount = 0;
149
150     char seenKeywords[MAX_LENGTH][100] = {0};
151     char seenOperators[MAX_LENGTH][100] = {0};
152     char seenConstants[MAX_LENGTH][100] = {0};
153     char seenPunctuations[MAX_LENGTH][100] = {0};
154     char seenIdentifiers[MAX_LENGTH][100] = {0};
155
156     printf("Keywords : [");
157     for (int i = 0; i < tokenCount; i++) {
158         if (tokens[i].type == KEYWORD) {
159             int isNew = 1;
160             for (int j = 0; j < keywordCount; j++) {
161                 if (strcmp(seenKeywords[j], tokens[i].value) == 0) {
162                     isNew = 0;
163                     break;
164                 }
165             }
166             if (isNew) {
167                 if (keywordCount > 0) {
168                     printf(", ");
169                 }
170                 printf("%s", tokens[i].value);
171                 strcpy(seenKeywords[keywordCount++], tokens[i].value);
172             }
173         }
174     }
175     printf("]");
176
177     printf("\n-----\n");
178     printf("Operators : [");
179     for (int i = 0; i < tokenCount; i++) {
180         if (tokens[i].type == OPERATOR) {
181             int isNew = 1;
182             for (int j = 0; j < operatorCount; j++) {
183                 if (strcmp(seenOperators[j], tokens[i].value) == 0) {
184                     isNew = 0;
185                     break;
186                 }
187             }
188             if (isNew) {
189                 if (operatorCount > 0) {
190                     printf(", ");
191                 }
192                 printf("%s", tokens[i].value);
193                 strcpy(seenOperators[operatorCount++], tokens[i].value);
194             }
195         }
196     }
197     printf("]");
198
199     printf("\n-----\n");

```

```

147 void printResults(struct Token* tokens, int tokenCount) {
200     printf("Constants : [");
201     for (int i = 0; i < tokenCount; i++) {
202         if (tokens[i].type == CONSTANT) {
203             int isNew = 1;
204             for (int j = 0; j < constantCount; j++) {
205                 if (strcmp(seenConstants[j], tokens[i].value) == 0) {
206                     isNew = 0;
207                     break;
208                 }
209             }
210             if (isNew) {
211                 if (constantCount > 0) {
212                     printf(", ");
213                 }
214                 printf("%s", tokens[i].value);
215                 strcpy(seenConstants[constantCount++], tokens[i].value);
216             }
217         }
218     }
219     printf("]");
220
221     printf("\n-----\n");
222     printf("Punctuations : [");
223     for (int i = 0; i < tokenCount; i++) {
224         if (tokens[i].type == PUNCTUATION) {
225             int isNew = 1;
226             for (int j = 0; j < punctuationCount; j++) {
227                 if (strcmp(seenPunctuations[j], tokens[i].value) == 0) {
228                     isNew = 0;
229                     break;
230                 }
231             }
232             if (isNew) {
233                 if (punctuationCount > 0) {
234                     printf(", ");
235                 }
236                 printf("%s", tokens[i].value);
237                 strcpy(seenPunctuations[punctuationCount++], tokens[i].value);
238             }
239         }
240     }
241     printf("]");
242

```

```

242
243     printf("\n-----\n");
244     printf("Identifiers : []");
245     for (int i = 0; i < tokenCount; i++) {
246         if (tokens[i].type == IDENTIFIER) {
247             int isNew = 1;
248             for (int j = 0; j < identifierCount; j++) {
249                 if (strcmp(seenIdentifiers[j], tokens[i].value) == 0) {
250                     isNew = 0;
251                     break;
252                 }
253             }
254             if (isNew) {
255                 if (identifierCount > 0) {
256                     printf(", ");
257                 }
258                 printf("%s", tokens[i].value);
259                 strcpy(seenIdentifiers[identifierCount++], tokens[i].value);
260             }
261         }
262     }
263     printf("]");
264
265     printf("\n-----\n");
266     printf("Total number of tokens are %d\n", tokenCount);
267 }

```

OUTPUT:

1) Print Statement:

```

/home/karan/Documents/CPP/string2
Enter a string of C code (enter an empty line to stop):
printf("Hello World");

Keywords : []
-----
Operators : []
-----
Constants : []
-----
Punctuations : [(, ), ;]
-----
Identifiers : [printf, "Hello World"]
-----
Total number of tokens are 5

Process returned 0 (0x0)   execution time : 26.148 s
Press ENTER to continue.
^A

```

2) Array initialization:

```
/home/karan/Documents/CPP/string2
Enter a string of C code (enter an empty line to stop):
int arr[]={1,2,3,4,5};

Keywords : [int]
-----
Operators : [=]
-----
Constants : [1, 2, 3, 4, 5]
-----
Punctuations : [{, ,, }, ;]
-----
Identifiers : [arr]
-----
Total number of tokens are 15

Process returned 0 (0x0)   execution time : 30,816 s
Press ENTER to continue.
█
```

3) Complex arithmetic statement:

```
/home/karan/Documents/CPP/string2
Enter a string of C code (enter an empty line to stop):
int result = a+b-c/d;

Keywords : [int]
-----
Operators : [=, +, -, /]
-----
Constants : []
-----
Punctuations : [;]
-----
Identifiers : [result, a, b, c, d]
-----
Total number of tokens are 11

Process returned 0 (0x0)   execution time : 37,514 s
Press ENTER to continue.
█
```

4) For loop initialization:

```
/home/karan/Documents/CPP/string2
Enter a string of C code (enter an empty line to stop):
for(int i=0; i<6; i++)

Keywords : [for, int]
-----
Operators : [=, <, +]
-----
Constants : [0, 6]
-----
Punctuations : [(, ;, )]
-----
Identifiers : [i]
-----
Total number of tokens are 15

Process returned 0 (0x0)   execution time : 37,451 s
Press ENTER to continue.
█
```


5) Sum with for loops:

/home/karan/Documents/CPP/string2

```
Enter a string of C code (enter an empty line to stop):
int main(){
int sum=0;
for(int i=0; i<10; i++){
count = count + i;
}
printf("%d",count);
return 0;
}
```

Keywords : [int, for, return]

Operators : [=, <, +]

Constants : [0, 10]

Punctuations : [(,), {, }, ,, ,]

Identifiers : [main, sum, i, count, printf, "%d"]

Total number of tokens are 44

Process returned 0 (0x0) execution time : 84.990 s
Press ENTER to continue.

Q3: Write a code to count the total number of tokens in a code.

The objective of the program is to analyze a given piece of C code and count the total number of tokens it contains. Tokens include keywords, identifiers, operators, literals, and symbols. The program should accurately parse the input code, identify all the tokens, and provide a count of these tokens. This analysis can be useful for understanding code complexity, estimating effort for code review, and providing insights into code structure and composition.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_LENGTH 1000
#define MAX_LINES 100

enum TokenType {
    KEYWORD,
    OPERATOR,
    CONSTANT,
    PUNCTUATION,
    IDENTIFIER
};

struct Token {
    enum TokenType type;
    char value[100];
};

int isKeyword(const char* str);
int isOperator(char ch);
int isPunctuation(char ch);
void tokenize(const char* input, struct Token* tokens, int* tokenCount);
void printResults(struct Token* tokens, int tokenCount);

int main() {
    char input[MAX_LINES * MAX_LENGTH] = {0};
    char line[MAX_LENGTH];
    int lineCount = 0;

    printf("Enter a string of C code (enter an empty line to stop):\n");

    // Read multiple lines of input
    while (lineCount < MAX_LINES) {
        if (fgets(line, MAX_LENGTH, stdin) == NULL) {
            break; // Stop if there's an input error
        }

        // Remove the newline character at the end of the line, if present
```

```
line[strcspn(line, "\n")] = '\0';
```

```
// Check if the line is empty
```

```
if (strlen(line) == 0) {  
    break; // Stop if an empty line is entered  
}
```

```
// Concatenate the line into the input string
```

```
strcat(input, line);  
strcat(input, " "); // Add a space to separate lines  
lineCount++;  
}
```

```
struct Token tokens[MAX_LENGTH];
```

```
int tokenCount = 0;
```

```
tokenize(input, tokens, &tokenCount);  
printResults(tokens, tokenCount);
```

```
return 0;  
}
```

```
int isKeyword(const char* str) {
```

```
// List of C keywords
```

```
const char* keywords[] = {  
    "auto", "break", "case", "char", "const", "continue", "default",  
    "do", "double", "else", "enum", "extern", "float", "for", "goto",  
    "if", "inline", "int", "long", "register", "restrict", "return",  
    "short", "signed", "sizeof", "static", "struct", "switch", "typedef",  
    "union", "unsigned", "void", "volatile", "while", "_Alignas",  
    "_Alignof", "_Atomic", "_Bool", "_Complex", "_Decimal128",  
    "_Decimal32", "_Decimal64", "_Generic", "_Imaginary", "_Noreturn",  
    "_Static_assert", "_Thread_local"  
};
```

```
// Number of keywords
```

```
const int numKeywords = sizeof(keywords) / sizeof(keywords[0]);
```

```
// Check if the given string matches any of the keywords
```

```
for (int i = 0; i < numKeywords; ++i) {  
    if (strcmp(str, keywords[i]) == 0) {  
        return 1; // It is a keyword  
    }  
}  
return 0; // Not a keyword  
}
```

```
int isOperator(char ch) {
```

```
return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '=' || ch == '<' || ch == '>');  
}
```

```
int isPunctuation(char ch) {
return (ch == '(' || ch == ')' || ch == '{' || ch == '}' || ch == ';' || ch == ',' || ch == '.');
}
```

```
void tokenize(const char* input, struct Token* tokens, int* tokenCount) {
char buffer[100];
int bufferIndex = 0;
*tokenCount = 0;
```

```
for (int i = 0; input[i] != '\0'; i++) {
if (isalnum(input[i]) || input[i] == '_' || input[i] == '-') {
buffer[bufferIndex++] = input[i];
} else {
if (bufferIndex > 0) {
buffer[bufferIndex] = '\0';
if (isKeyword(buffer)) {
tokens[*tokenCount].type = KEYWORD;
} else if (isdigit(buffer[0])) {
tokens[*tokenCount].type = CONSTANT;
} else {
tokens[*tokenCount].type = IDENTIFIER;
}
strcpy(tokens[*tokenCount].value, buffer);
(*tokenCount)++;
bufferIndex = 0;
}
```

```
if (isOperator(input[i])) {
tokens[*tokenCount].type = OPERATOR;
tokens[*tokenCount].value[0] = input[i];
tokens[*tokenCount].value[1] = '\0';
(*tokenCount)++;
} else if (isPunctuation(input[i])) {
tokens[*tokenCount].type = PUNCTUATION;
tokens[*tokenCount].value[0] = input[i];
tokens[*tokenCount].value[1] = '\0';
(*tokenCount)++;
} else if (input[i] == '"') {
int j = i + 1;
while (input[j] != '"' && input[j] != '\0') {
j++;
}
if (input[j] == '"') {
tokens[*tokenCount].type = IDENTIFIER;
tokens[*tokenCount].value[0] = '"';
strncpy(tokens[*tokenCount].value + 1, input + i + 1, j - i - 1);
tokens[*tokenCount].value[j - i] = '"';
tokens[*tokenCount].value[j - i + 1] = '\0';
(*tokenCount)++;
}
```

```
i = j;  
}  
}  
}  
}  
}
```

```
void printResults(struct Token* tokens, int tokenCount) {  
    int keywordCount = 0, operatorCount = 0, constantCount = 0, punctuationCount = 0,  
    identifierCount = 0;
```

```
    // Count each type of token  
    for (int i = 0; i < tokenCount; i++) {  
        switch (tokens[i].type) {  
            case KEYWORD:  
                keywordCount++;  
                break;  
            case OPERATOR:  
                operatorCount++;  
                break;  
            case CONSTANT:  
                constantCount++;  
                break;  
            case PUNCTUATION:  
                punctuationCount++;  
                break;  
            case IDENTIFIER:  
                identifierCount++;  
                break;  
        }  
    }
```

```
    // Print counts  
    printf("Keywords: %d\n", keywordCount);  
    printf("Operators: %d\n", operatorCount);  
    printf("Constants: %d\n", constantCount);  
    printf("Punctuations: %d\n", punctuationCount);  
    printf("Identifiers: %d\n", identifierCount);  
    printf("Total number of tokens: %d\n", tokenCount);  
}
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX_LENGTH 1000
7  #define MAX_LINES 100
8
9  enum TokenType {
10     KEYWORD,
11     OPERATOR,
12     CONSTANT,
13     PUNCTUATION,
14     IDENTIFIER
15 };
16
17 struct Token {
18     enum TokenType type;
19     char value[100];
20 };
21
22 int isKeyword(const char* str);
23 int isOperator(char ch);
24 int isPunctuation(char ch);
25 void tokenize(const char* input, struct Token* tokens, int* tokenCount);
26 void printResults(struct Token* tokens, int tokenCount);
27
28 int main() {
29     char input[MAX_LINES * MAX_LENGTH] = {0};
30     char line[MAX_LENGTH];
31     int lineCount = 0;
32
33     printf("Enter a string of C code (enter an empty line to stop):\n");
34
35     // Read multiple lines of input
36     while (lineCount < MAX_LINES) {
37         if (fgets(line, MAX_LENGTH, stdin) == NULL) {
38             break; // Stop if there's an input error
39         }
40
41         // Remove the newline character at the end of the line, if present
42         line[strcspn(line, "\n")] = '\0';
43
44         // Check if the line is empty
45         if (strlen(line) == 0) {
46             break; // Stop if an empty line is entered
47         }
48
49         // Concatenate the line into the input string
50         strcat(input, line);
51         strcat(input, " "); // Add a space to separate lines
52         lineCount++;
53     }

```

```

55     struct Token tokens[MAX_LENGTH];
56     int tokenCount = 0;
57
58     tokenize(input, tokens, &tokenCount);
59     printResults(tokens, tokenCount);
60
61     return 0;
62 }
63
64 int isKeyword(const char* str) {
65     // List of C keywords
66     const char* keywords[] = {
67         "auto", "break", "case", "char", "const", "continue", "default",
68         "do", "double", "else", "enum", "extern", "float", "for", "goto",
69         "if", "inline", "int", "long", "register", "restrict", "return",
70         "short", "signed", "sizeof", "static", "struct", "switch", "typedef",
71         "union", "unsigned", "void", "volatile", "while", "_Alignas",
72         "_Alignof", "_Atomic", "_Bool", "_Complex", "_Decimal128",
73         "_Decimal32", "_Decimal64", "_Generic", "_Imaginary", "_Noreturn",
74         "_Static_assert", "_Thread_local"
75     };
76
77     // Number of keywords
78     const int numKeywords = sizeof(keywords) / sizeof(keywords[0]);
79
80     // Check if the given string matches any of the keywords
81     for (int i = 0; i < numKeywords; ++i) {
82         if (strcmp(str, keywords[i]) == 0) {
83             return 1; // It is a keyword
84         }
85     }
86     return 0; // Not a keyword
87 }
88
89 int isOperator(char ch) {
90     return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '=' || ch == '<' || ch == '>');
91 }
92
93 int isPunctuation(char ch) {
94     return (ch == '(' || ch == ')' || ch == '{' || ch == '}' || ch == ';' || ch == ',' || ch == '.');
95 }

```

```

97 void tokenize(const char* input, struct Token* tokens, int* tokenCount) {
98     char buffer[100];
99     int bufferIndex = 0;
100     *tokenCount = 0;
101
102     for (int i = 0; input[i] != '\0'; i++) {
103         if (isalnum(input[i]) || input[i] == '_') {
104             buffer[bufferIndex++] = input[i];
105         } else {
106             if (bufferIndex > 0) {
107                 buffer[bufferIndex] = '\0';
108                 if (isKeyword(buffer)) {
109                     tokens[*tokenCount].type = KEYWORD;
110                 } else if (isdigit(buffer[0])) {
111                     tokens[*tokenCount].type = CONSTANT;
112                 } else {
113                     tokens[*tokenCount].type = IDENTIFIER;
114                 }
115                 strcpy(tokens[*tokenCount].value, buffer);
116                 (*tokenCount)++;
117                 bufferIndex = 0;
118             }
119         }
120     }

```



```

120         if (isOperator(input[i])) {
121             tokens[*tokenCount].type = OPERATOR;
122             tokens[*tokenCount].value[0] = input[i];
123             tokens[*tokenCount].value[1] = '\0';
124             (*tokenCount)++;
125         } else if (isPunctuation(input[i])) {
126             tokens[*tokenCount].type = PUNCTUATION;
127             tokens[*tokenCount].value[0] = input[i];
128             tokens[*tokenCount].value[1] = '\0';
129             (*tokenCount)++;
130         } else if (input[i] == '"') {
131             int j = i + 1;
132             while (input[j] != '"' && input[j] != '\0') {
133                 j++;
134             }
135             if (input[j] == '"') {
136                 tokens[*tokenCount].type = IDENTIFIER;
137                 tokens[*tokenCount].value[0] = '"';
138                 strncpy(tokens[*tokenCount].value + 1, input + i + 1, j - i - 1);
139                 tokens[*tokenCount].value[j - i] = '"';
140                 tokens[*tokenCount].value[j - i + 1] = '\0';
141                 (*tokenCount)++;
142                 i = j;
143             }
144         }
145     }
146 }
147 }
148

```

```

149 void printResults(struct Token* tokens, int tokenCount) {
150     int keywordCount = 0, operatorCount = 0, constantCount = 0, punctuationCount = 0, identifierCount = 0;
151
152     // Count each type of token
153     for (int i = 0; i < tokenCount; i++) {
154         switch (tokens[i].type) {
155             case KEYWORD:
156                 keywordCount++;
157                 break;
158             case OPERATOR:
159                 operatorCount++;
160                 break;
161             case CONSTANT:
162                 constantCount++;
163                 break;
164             case PUNCTUATION:
165                 punctuationCount++;
166                 break;
167             case IDENTIFIER:
168                 identifierCount++;
169                 break;
170         }
171     }
172
173     // Print counts
174     printf("Keywords: %d\n", keywordCount);
175     printf("Operators: %d\n", operatorCount);
176     printf("Constants: %d\n", constantCount);
177     printf("Punctuations: %d\n", punctuationCount);
178     printf("Identifiers: %d\n", identifierCount);
179     printf("Total number of tokens: %d\n", tokenCount);
180 }

```


OUTPUT:

1) C program to find sum of first 10 natural numbers:

```
/home/karan/Documents/CPP/string3
Enter a string of C code (enter an empty line to stop):
#include <stdio.h>
int main(){
int count=0;
for(int i=0;i<10;i++){
count = count + i;
}
printf("%d",count);
return 0;
}

Keywords: 5
Operators: 9
Constants: 4
Punctuations: 18
Identifiers: 14
Total number of tokens: 50

Process returned 0 (0x0)   execution time : 116.802 s
Press ENTER to continue.
█
```

2) Function Definition

```
/home/karan/Documents/CPP/string3
Enter a string of C code (enter an empty line to stop):
void my(int a, int b){
int sum = a + b;
printf("sum is: %d",sum);
}

Keywords: 4
Operators: 2
Constants: 0
Punctuations: 10
Identifiers: 9
Total number of tokens: 25

Process returned 0 (0x0)   execution time : 70.852 s
Press ENTER to continue.
█
```

3) Struct Initialization

/home/karan/Documents/CPP/string3

Enter a string of C code (enter an empty line to stop):

```
struct Point {  
int x;  
int y;  
};  
struct Point p = {10,20};
```

Keywords: 4

Operators: 1

Constants: 2

Punctuations: 9

Identifiers: 5

Total number of tokens: 21

Process returned 0 (0x0) execution time : 43.264 s

Press ENTER to continue.

■