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, sub string 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.

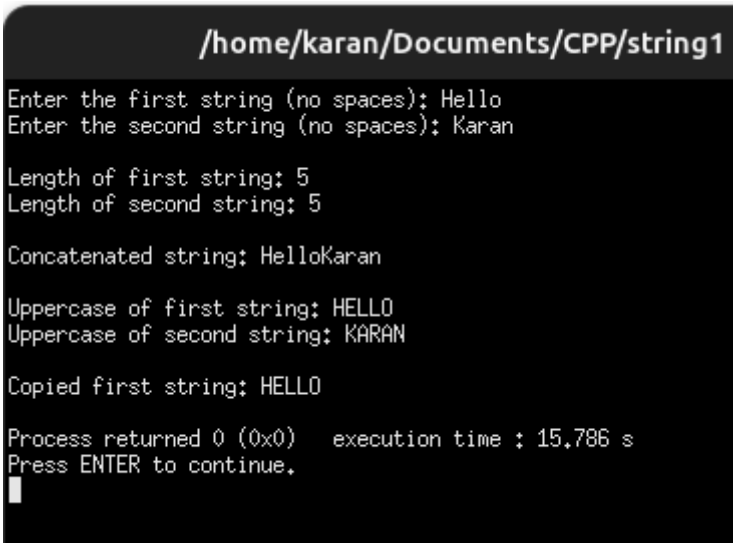
CODE:-

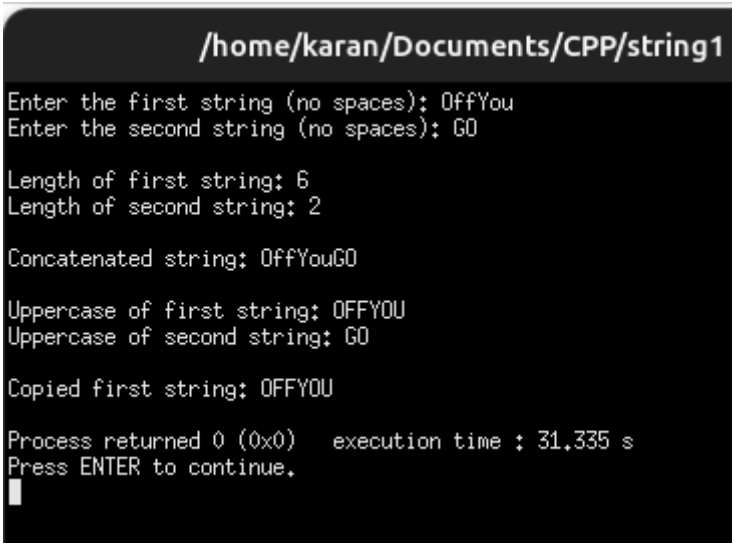
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

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;}

```c
#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;
}

```

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);
}
```

```c
#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;
```

```c

    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;
```

```c
void tokenize(const char* input, struct Token* tokens, int* tokenCount) {
    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;
                }
            }
        }
    }
}
```

```c

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");
```

```c
void printResults(struct Token* tokens, int tokenCount) {
    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);
}
```

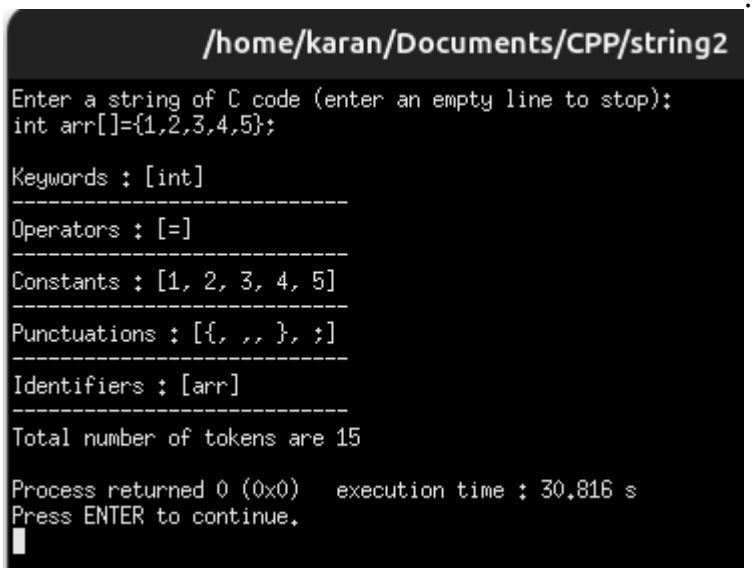
OUTPUT:

1) Print Statement:

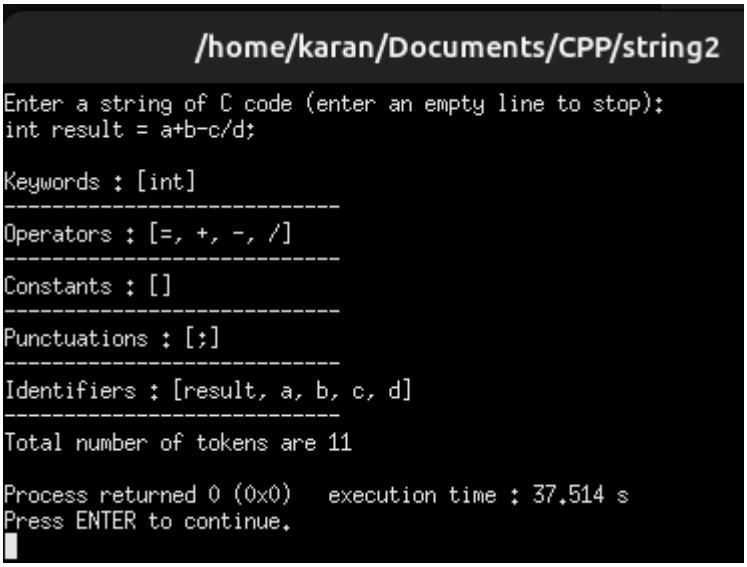
```
/home/karan/Documents/CPP/string2
Enter a string of C code (enter an empty line to stop):
printf("Hello World");

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:

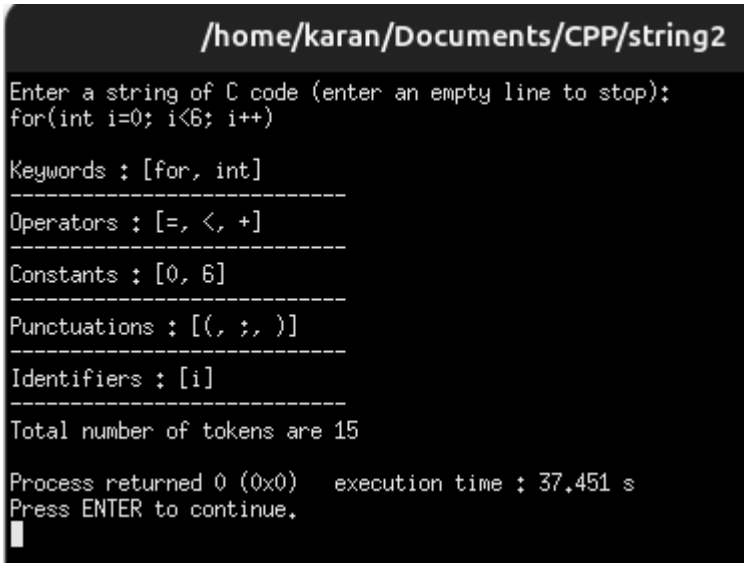
:



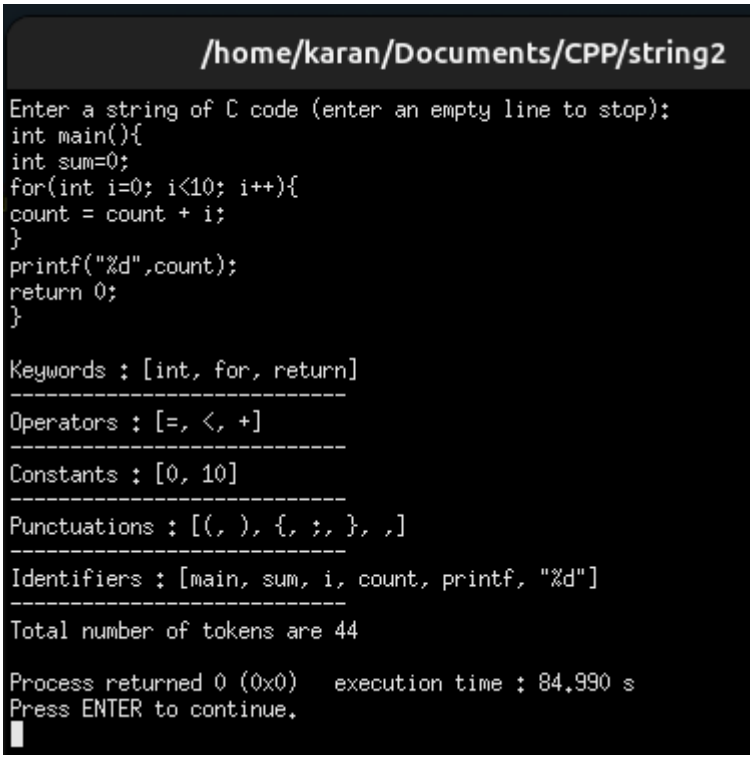
3) Complex arithmetic statement:



4) For loop initialization:



5) Sum with for loops:



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] == '_') {
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);
}
```

```c
#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++;
    }
```

```c
    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 == '.');
}
```

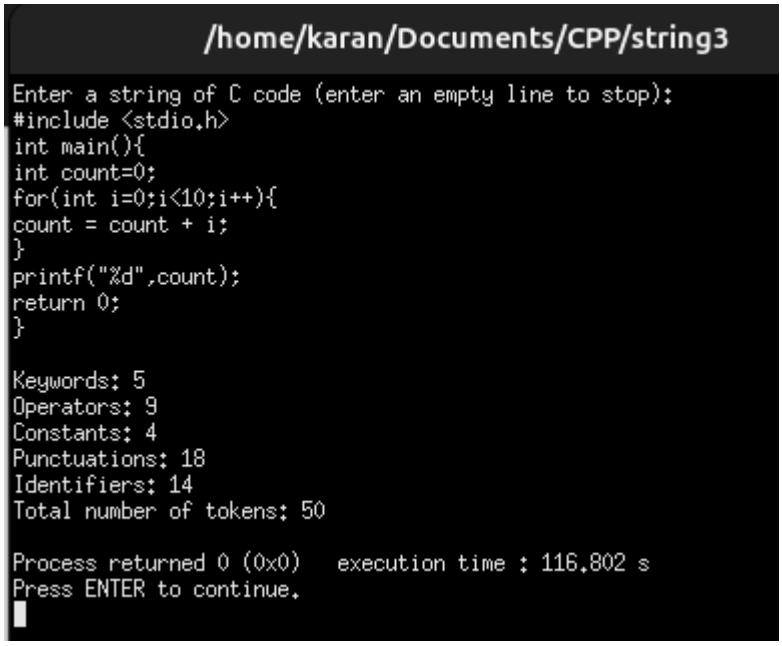
```c
void tokenize(const char* input, struct Token* tokens, int* tokenCount) {
    char buffer[100];
    int bufferIndex = 0;
    *tokenCount = 0;

    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;
                }
            }
        }
    }
}
```

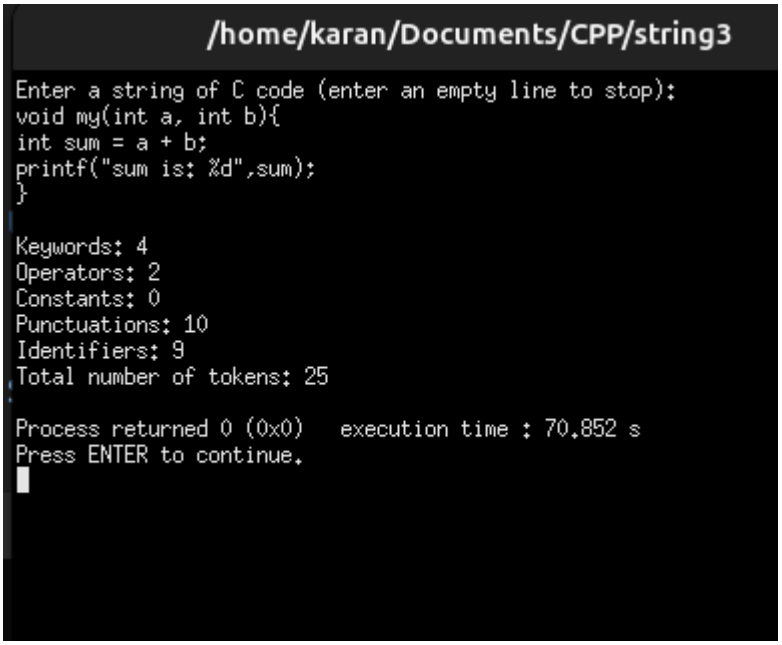
```
void printResults(struct Token* tokens, int tokenCount) {
    int keywordCount = 0, operatorCount = 0, constantCount = 0, punctuationCount = 0, identifierCount = 0;

    // 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);
}
```

OUTPUT:

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



2) Function Definition



3) Struct Initialization