TITLE: Regular Expression Parser

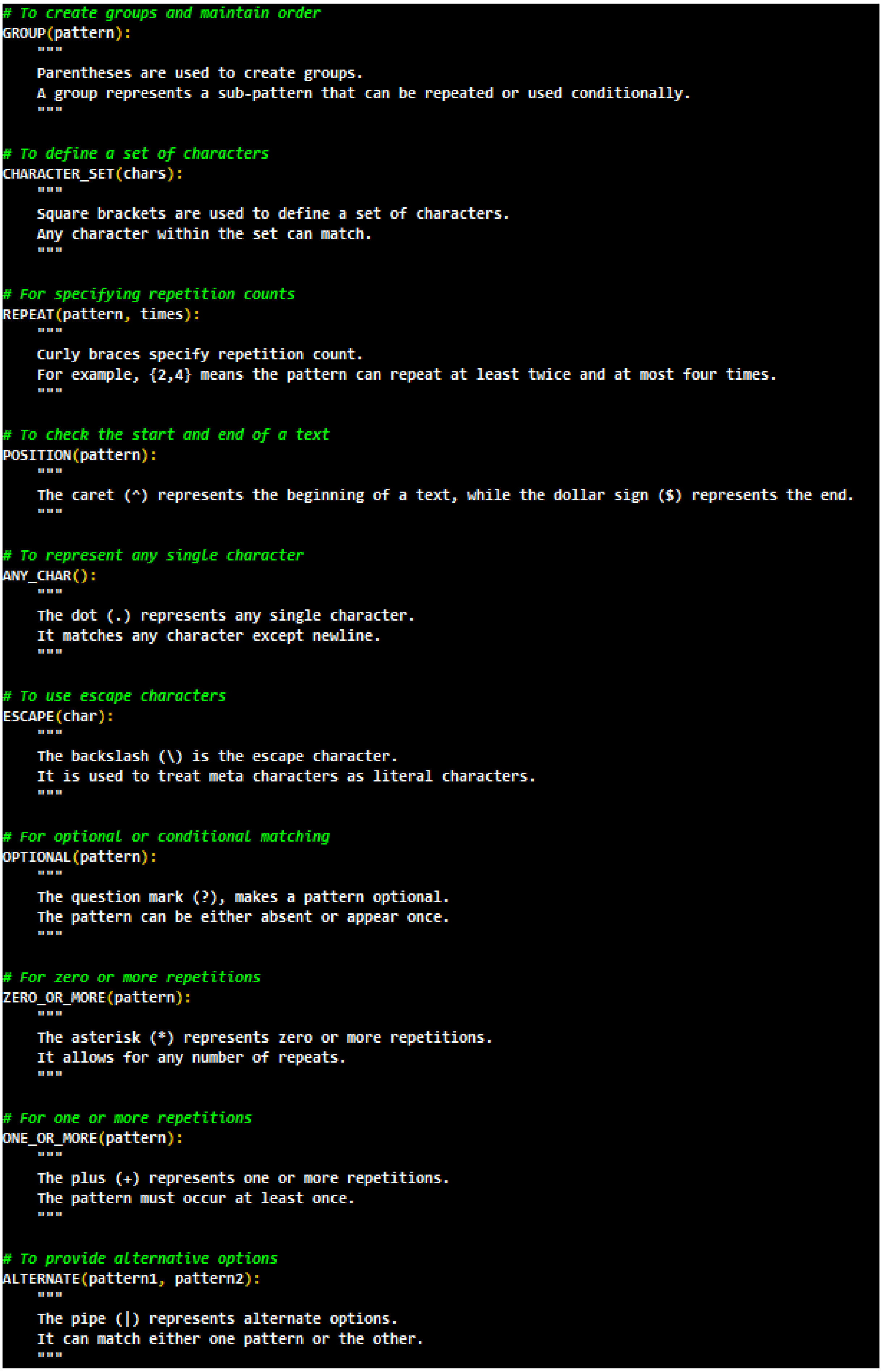
1. Introduction

- Regular Expressions (RegEx) are a powerful tool for text manipulation and pattern matching in programming. They allow developers to define search patterns that can be used to find, extract, or replace specific sequences within a text. RegEx is widely used in various applications, such as data validation, text parsing, and complex string searches.

2. Problem Definition

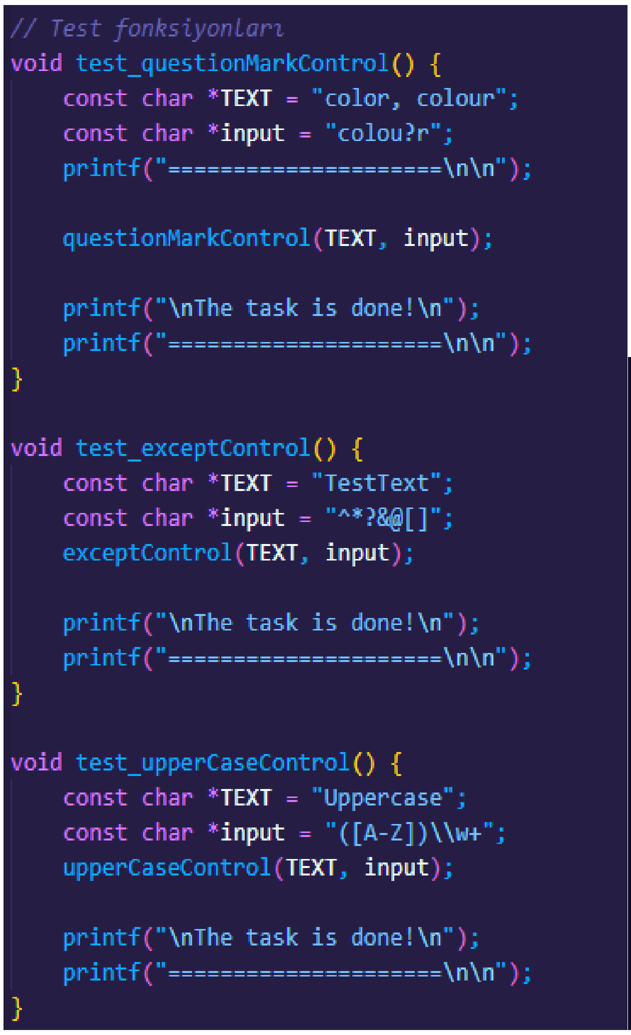
- Given a text or string, there are often scenarios where you need to find specific patterns, extract certain elements, or validate the format. For example, in email validation, file path extraction, or HTML content parsing, RegEx provides a flexible way to define patterns and match them efficiently. However, understanding and constructing these patterns requires a clear grasp of the different meta characters and their roles in RegEx. This project aims to explore these meta characters and demonstrate their usage through various examples and pseudo code explanations.

3. Basic Requirements and Corresponding Algorithm

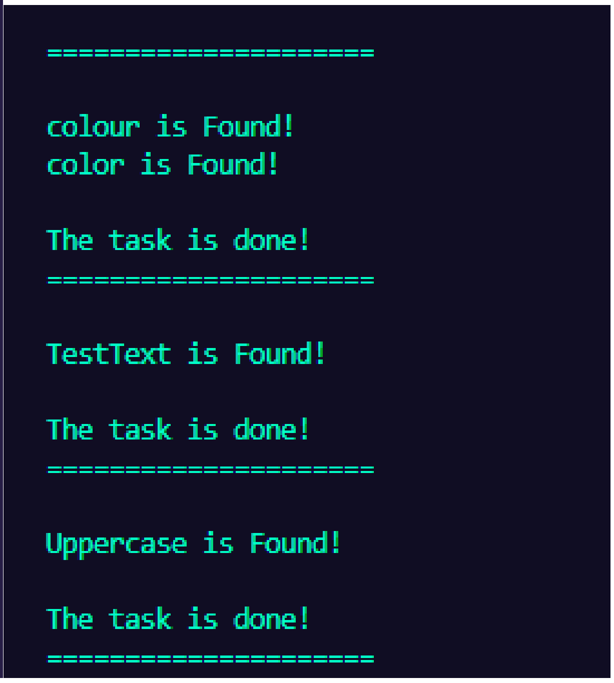


```
# To create groups and maintain order
GROUP(pattern):
    """
    Parentheses are used to create groups.
    A group represents a sub-pattern that can be repeated or used conditionally.
    """

# To define a set of characters
CHARACTER_SET(chars):
    """
    Square brackets are used to define a set of characters.
    Any character within the set can match.
    """

# For specifying repetition counts
REPEAT(pattern, times):
    """
    Curly braces specify repetition count.
    For example, {2,4} means the pattern can repeat at least twice and at most four times.
    """

# To check the start and end of a text
POSITION(pattern):
    """
    The caret (^) represents the beginning of a text, while the dollar sign ($) represents the end.
    """

# To represent any single character
ANY_CHAR():
    """
    The dot (.) represents any single character.
    It matches any character except newline.
    """

# To use escape characters
ESCAPE(char):
    """
    The backslash (\) is the escape character.
    It is used to treat meta characters as literal characters.
    """

# For optional or conditional matching
OPTIONAL(pattern):
    """
    The question mark (?), makes a pattern optional.
    The pattern can be either absent or appear once.
    """

# For zero or more repetitions
ZERO_OR_MORE(pattern):
    """
    The asterisk (*) represents zero or more repetitions.
    It allows for any number of repeats.
    """

# For one or more repetitions
ONE_OR_MORE(pattern):
    """
    The plus (+) represents one or more repetitions.
    The pattern must occur at least once.
    """

# To provide alternative options
ALTERNATE(pattern1, pattern2):
    """
    The pipe (|) represents alternate options.
    It can match either one pattern or the other.
    """
```

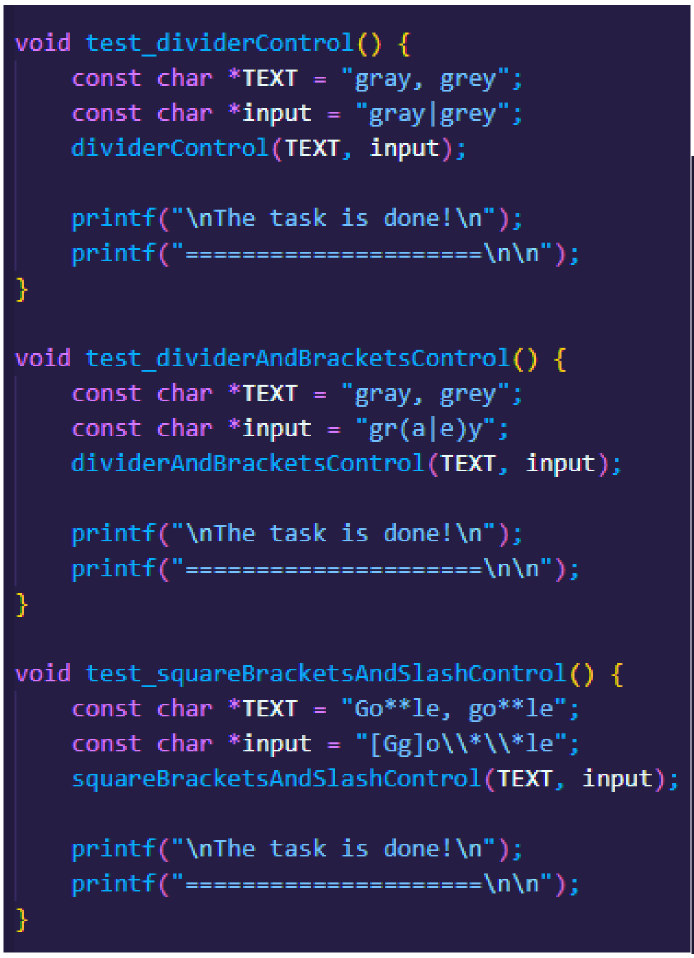
4. Demonstration



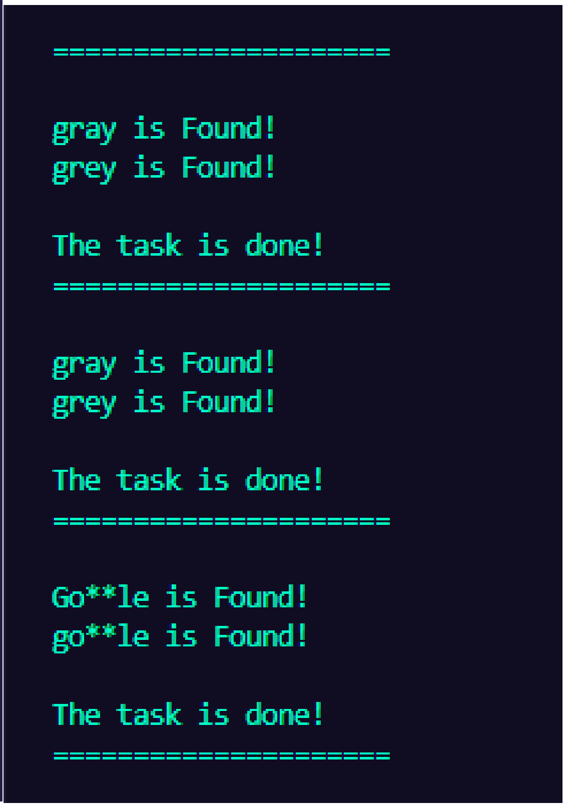
```c
// Test fonksiyonları
void test_questionMarkControl() {
    const char *TEXT = "color, colour";
    const char *input = "colou?r";
    printf("====================\n\n");

    questionMarkControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}

void test_exceptControl() {
    const char *TEXT = "TestText";
    const char *input = "^*?&@[]";
    exceptControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}

void test_upperCaseControl() {
    const char *TEXT = "Uppercase";
    const char *input = "([A-Z])\\w+";
    upperCaseControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}
```



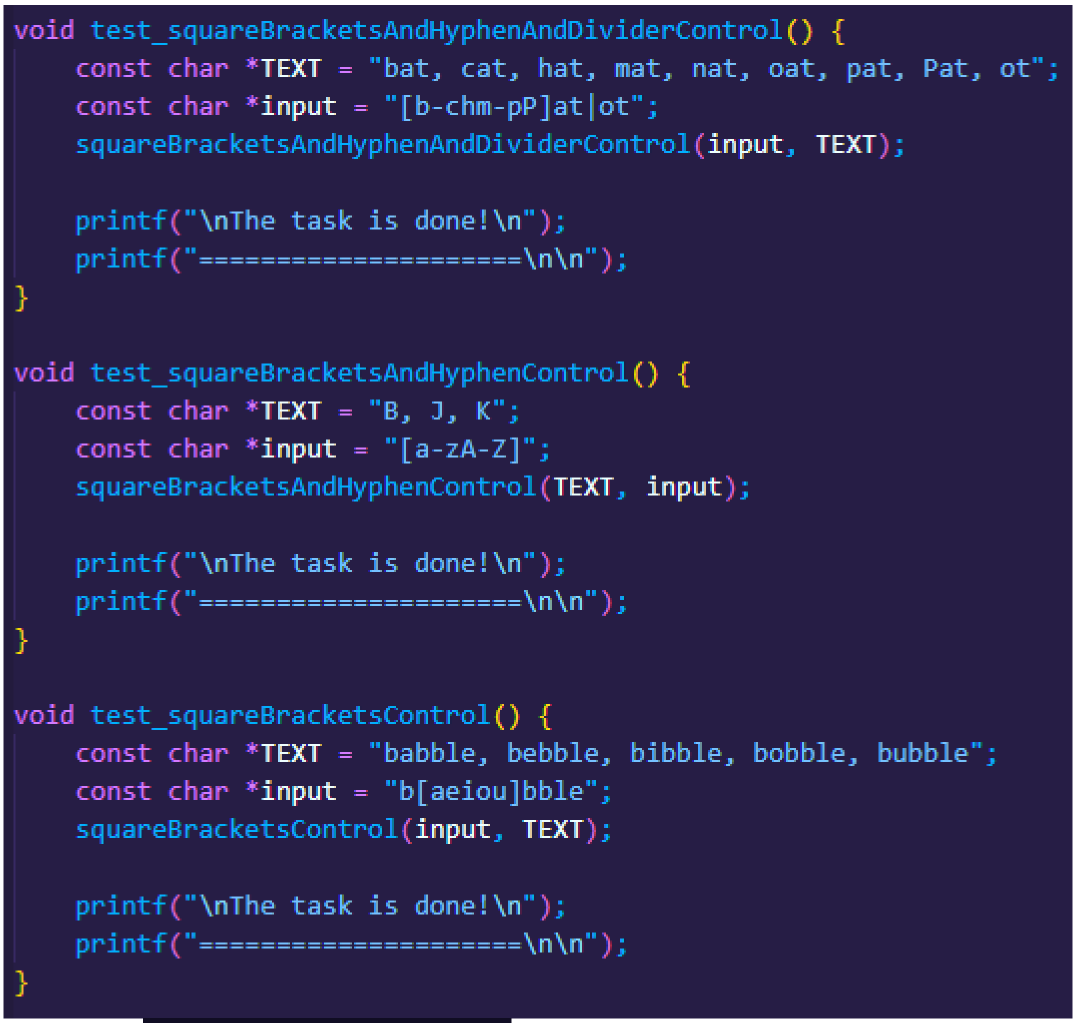
```
=====================


colour is Found!
color is Found!

The task is done!
=====================


TestText is Found!

The task is done!
=====================


Uppercase is Found!

The task is done!
=====================
```



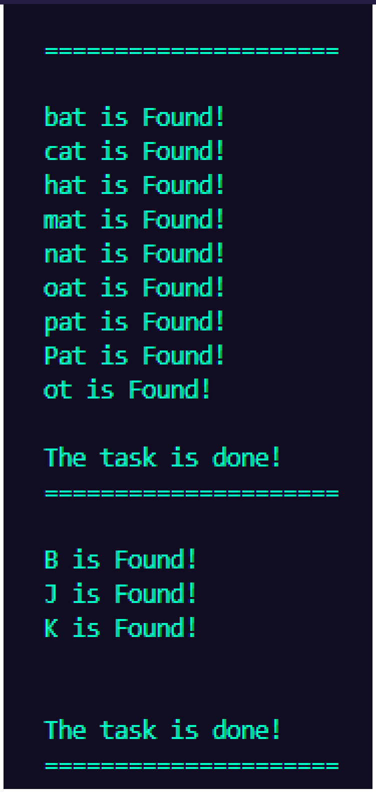
```c
void test_dividerControl() {
    const char *TEXT = "gray, grey";
    const char *input = "gray|grey";
    dividerControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_dividerAndBracketsControl() {
    const char *TEXT = "gray, grey";
    const char *input = "gr(a|e)y";
    dividerAndBracketsControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_squareBracketsAndSlashControl() {
    const char *TEXT = "Go**le, go**le";
    const char *input = "[Gg]o\\*\\*le";
    squareBracketsAndSlashControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}
```



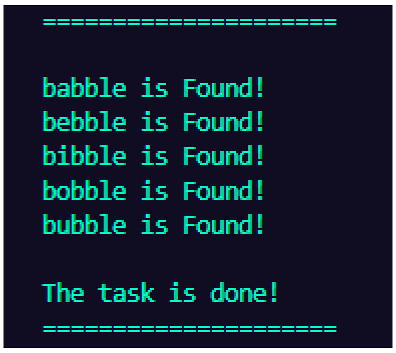
```
=====================

gray is Found!
grey is Found!

The task is done!
=====================

gray is Found!
grey is Found!

The task is done!
=====================

Go**le is Found!
go**le is Found!

The task is done!
=====================
```



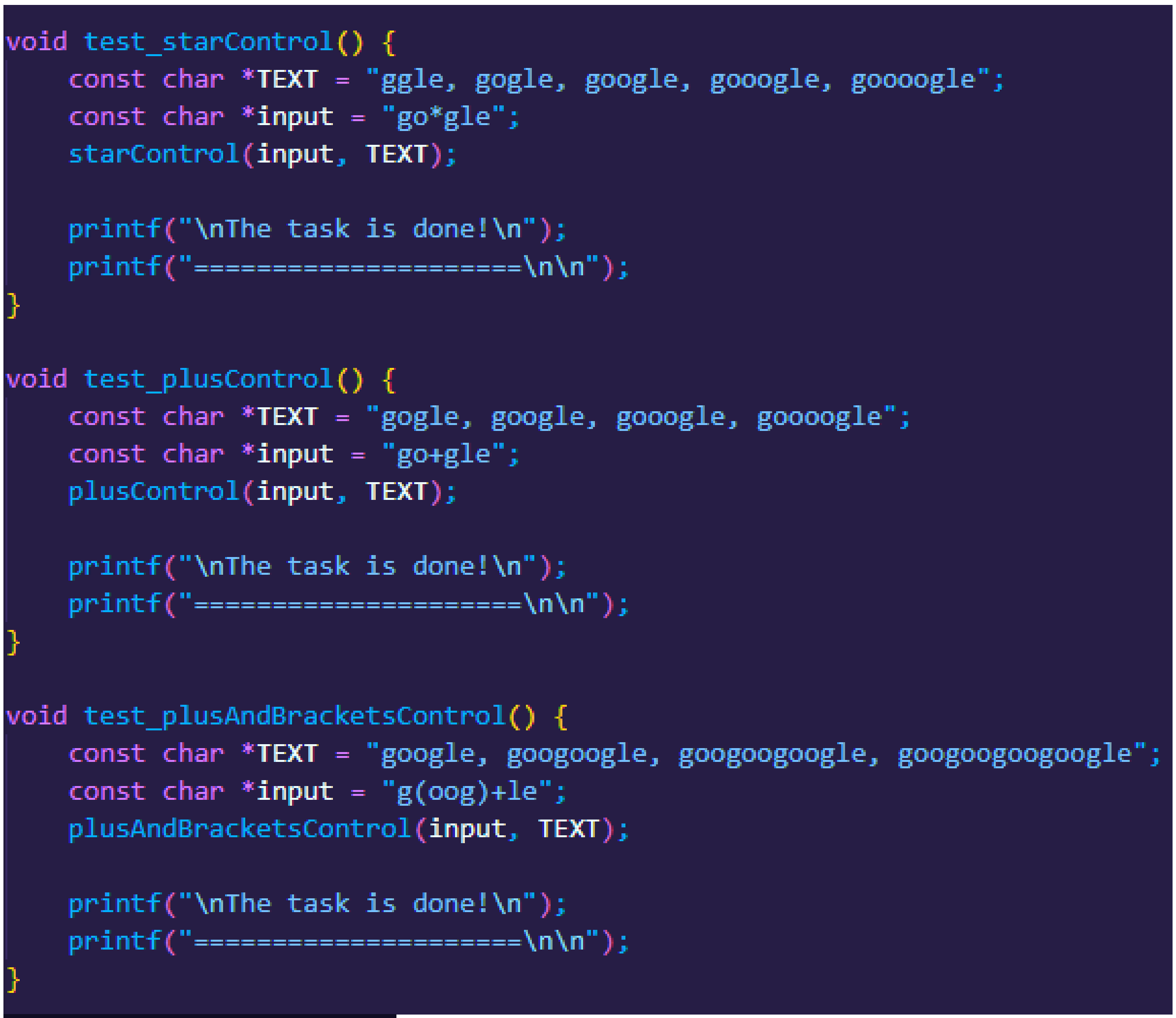
```c
void test_squareBracketsAndHyphenAndDividerControl() {
    const char *TEXT = "bat, cat, hat, mat, nat, oat, pat, Pat, ot";
    const char *input = "[b-chm-pP]at|ot";
    squareBracketsAndHyphenAndDividerControl(input, TEXT);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_squareBracketsAndHyphenControl() {
    const char *TEXT = "B, J, K";
    const char *input = "[a-zA-Z]";
    squareBracketsAndHyphenControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_squareBracketsControl() {
    const char *TEXT = "babble, bebble, bibble, bobble, bubble";
    const char *input = "b[aeiou]bble";
    squareBracketsControl(input, TEXT);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}
```



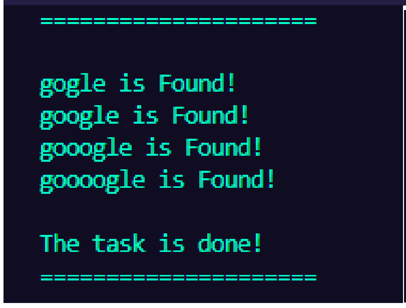
```
=====================

bat is Found!
cat is Found!
hat is Found!
mat is Found!
nat is Found!
oat is Found!
pat is Found!
Pat is Found!
ot is Found!

The task is done!
=====================

B is Found!
J is Found!
K is Found!


The task is done!
=====================
```



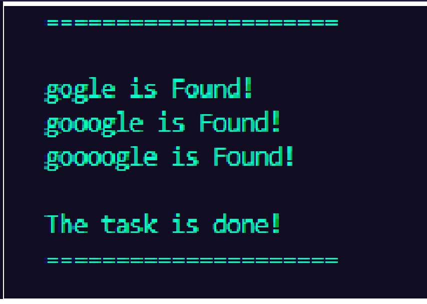
```
=====================

babble is Found!
bebble is Found!
bibble is Found!
bobble is Found!
bubble is Found!

The task is done!
=====================
```



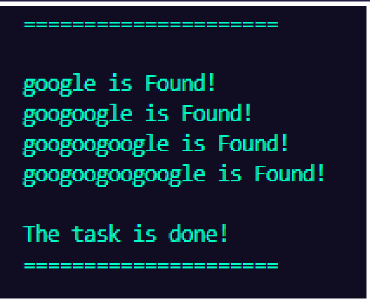
```c
void test_starControl() {
    const char *TEXT = "ggle, gogle, google, gooogle, goooogle";
    const char *input = "go*gle";
    starControl(input, TEXT);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_plusControl() {
    const char *TEXT = "gogle, google, gooogle, goooogle";
    const char *input = "go+gle";
    plusControl(input, TEXT);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_plusAndBracketsControl() {
    const char *TEXT = "google, googoogle, googoogoogle, googoogoogoogle";
    const char *input = "g(oog)+le";
    plusAndBracketsControl(input, TEXT);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}
```



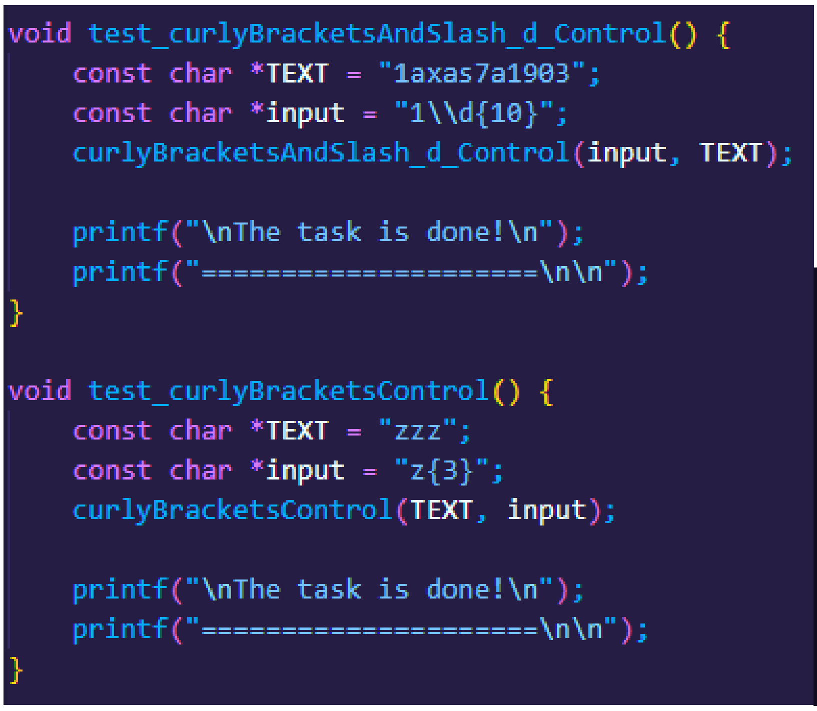
```
=====================

gogle is Found!
google is Found!
gooogle is Found!
goooogle is Found!

The task is done!
=====================
```



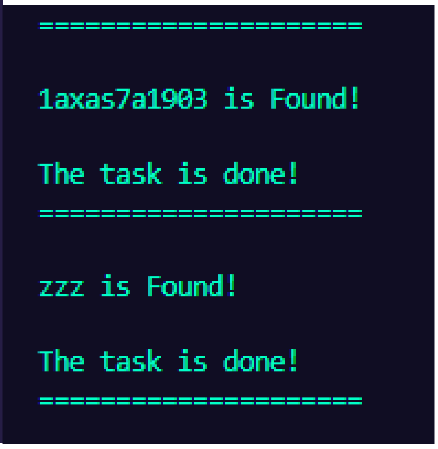
```
=====================

gogle is Found!
gooogle is Found!
goooogle is Found!

The task is done!
=====================
```



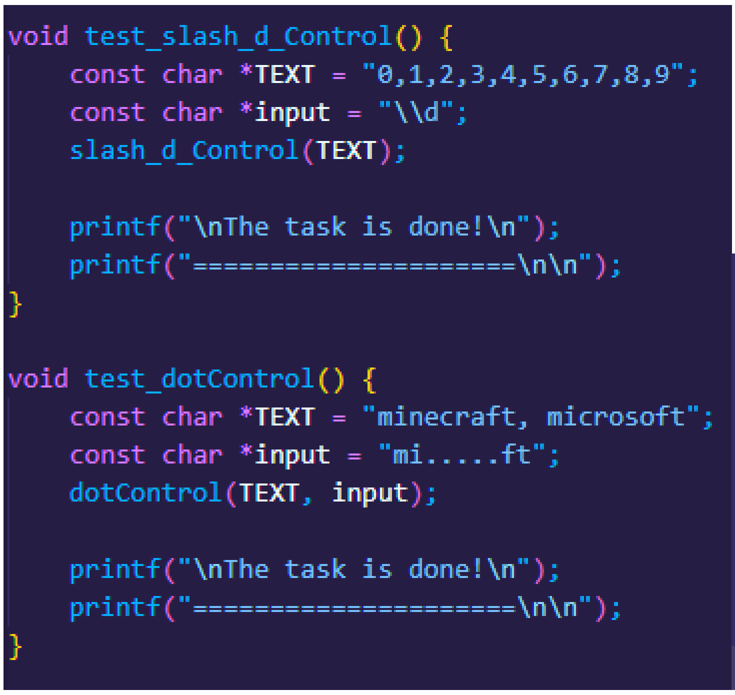
```
=====================

google is Found!
googoogle is Found!
googoogoogle is Found!
googoogoogoogle is Found!

The task is done!
=====================
```



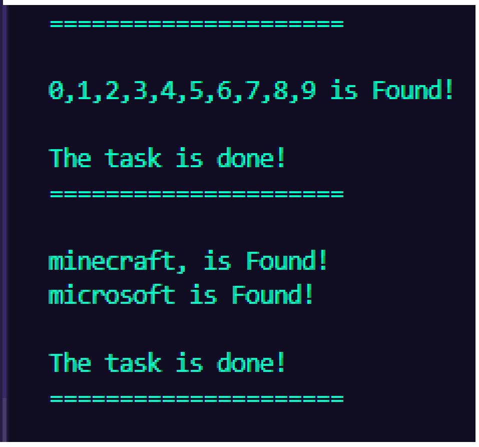
```c
void test_curlyBracketsAndSlash_d_Control() {
    const char *TEXT = "1axas7a1903";
    const char *input = "1\\d{10}";
    curlyBracketsAndSlash_d_Control(input, TEXT);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}

void test_curlyBracketsControl() {
    const char *TEXT = "zzz";
    const char *input = "z{3}";
    curlyBracketsControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}
```



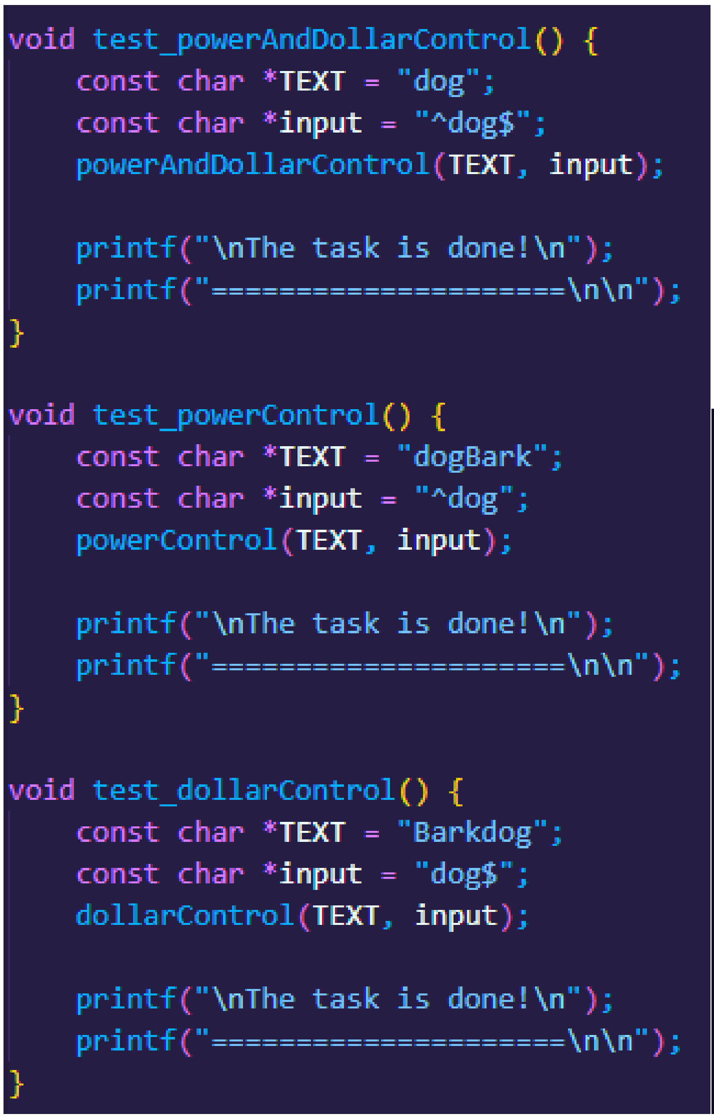
```
====================

1axas7a1903 is Found!

The task is done!
====================

zzz is Found!

The task is done!
====================
```



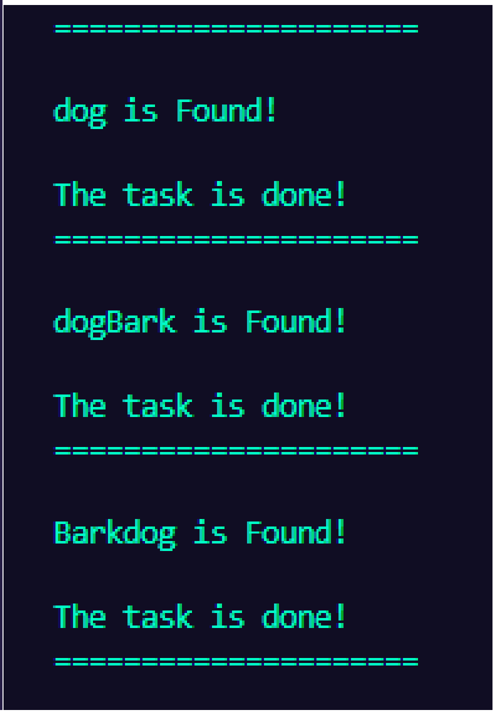
```c
void test_slash_d_Control() {
    const char *TEXT = "0,1,2,3,4,5,6,7,8,9";
    const char *input = "\\d";
    slash_d_Control(TEXT);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}

void test_dotControl() {
    const char *TEXT = "minecraft, microsoft";
    const char *input = "mi.....ft";
    dotControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("====================\n\n");
}
```



```
====================

0,1,2,3,4,5,6,7,8,9 is Found!

The task is done!
====================

minecraft, is Found!
microsoft is Found!

The task is done!
====================
```



```
void test_powerAndDollarControl() {
    const char *TEXT = "dog";
    const char *input = "^dog$";
    powerAndDollarControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_powerControl() {
    const char *TEXT = "dogBark";
    const char *input = "^dog";
    powerControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}

void test_dollarControl() {
    const char *TEXT = "Barkdog";
    const char *input = "dog$";
    dollarControl(TEXT, input);

    printf("\nThe task is done!\n");
    printf("=====================\n\n");
}
```



```
=====================

dog is Found!

The task is done!
=====================

dogBark is Found!

The task is done!
=====================

Barkdog is Found!

The task is done!
=====================
```

5. References

- https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions

- https://regexr.com/

.