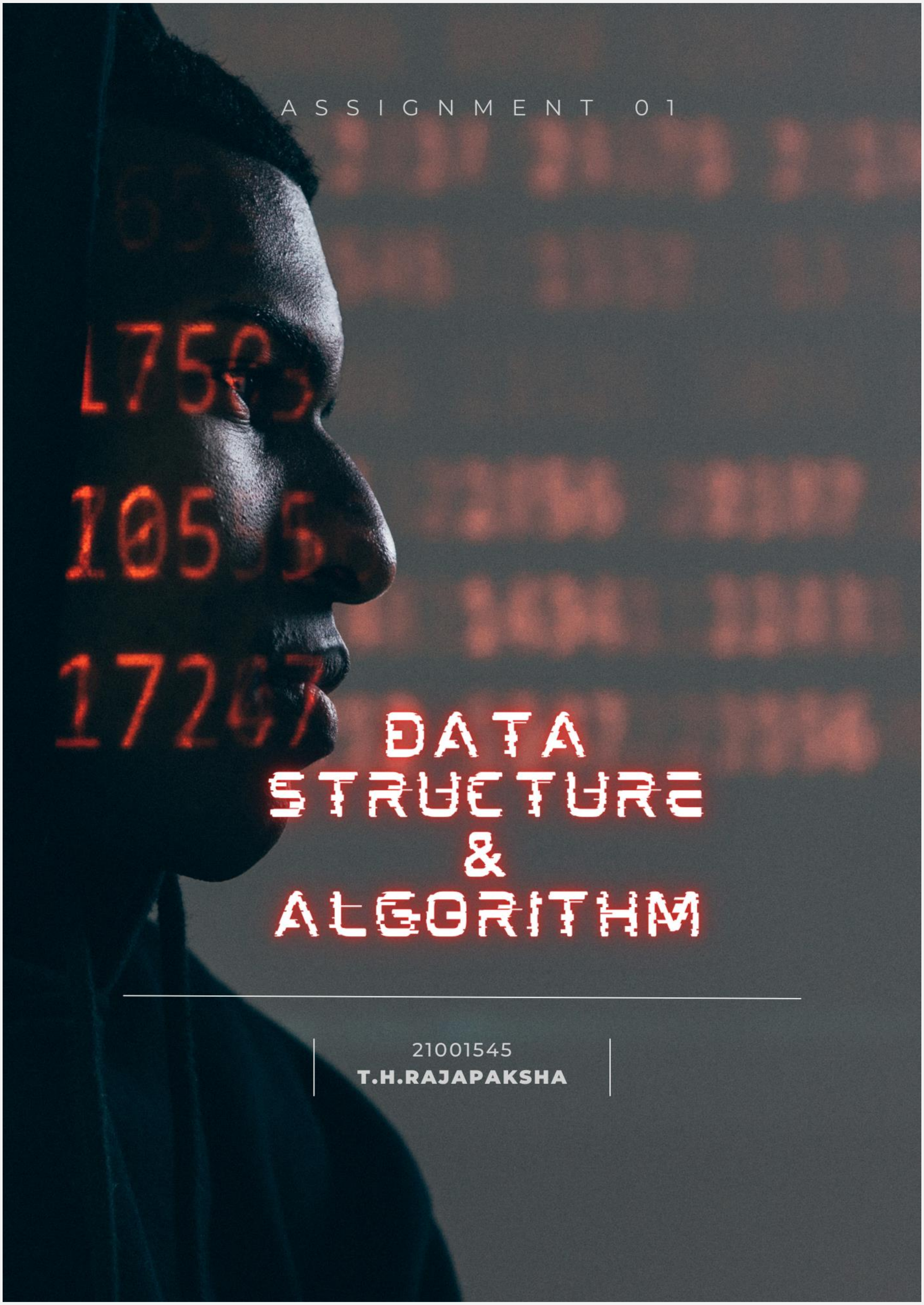


ASSIGNMENT 01

# DATA STRUCTURE & ALGORITHM

21001545
**T.H.RAJAPAKSHA**

# SCS 2201 – DATA STRUCTURE AND ALGORITHM III

## ASSIGNMENT 01

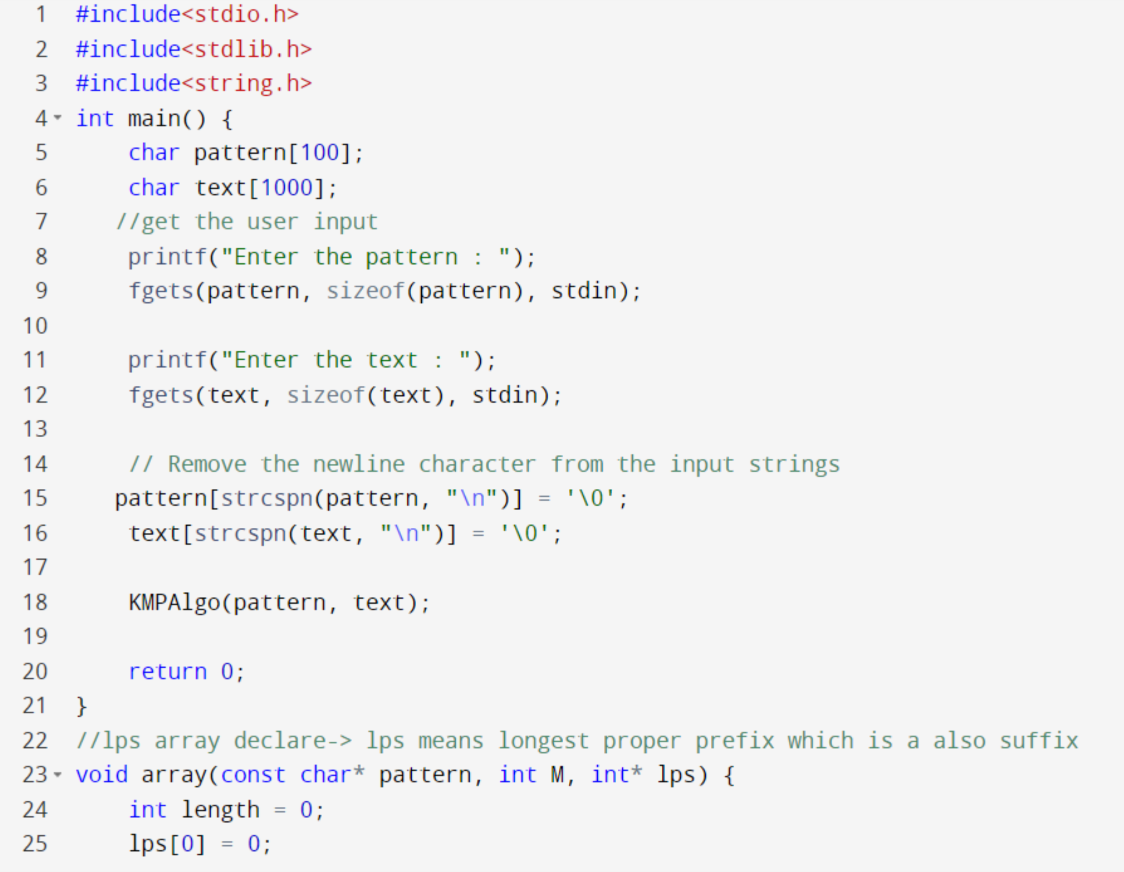
**Choose a Programming Language :**

I choose C programming . I think it's better for me.

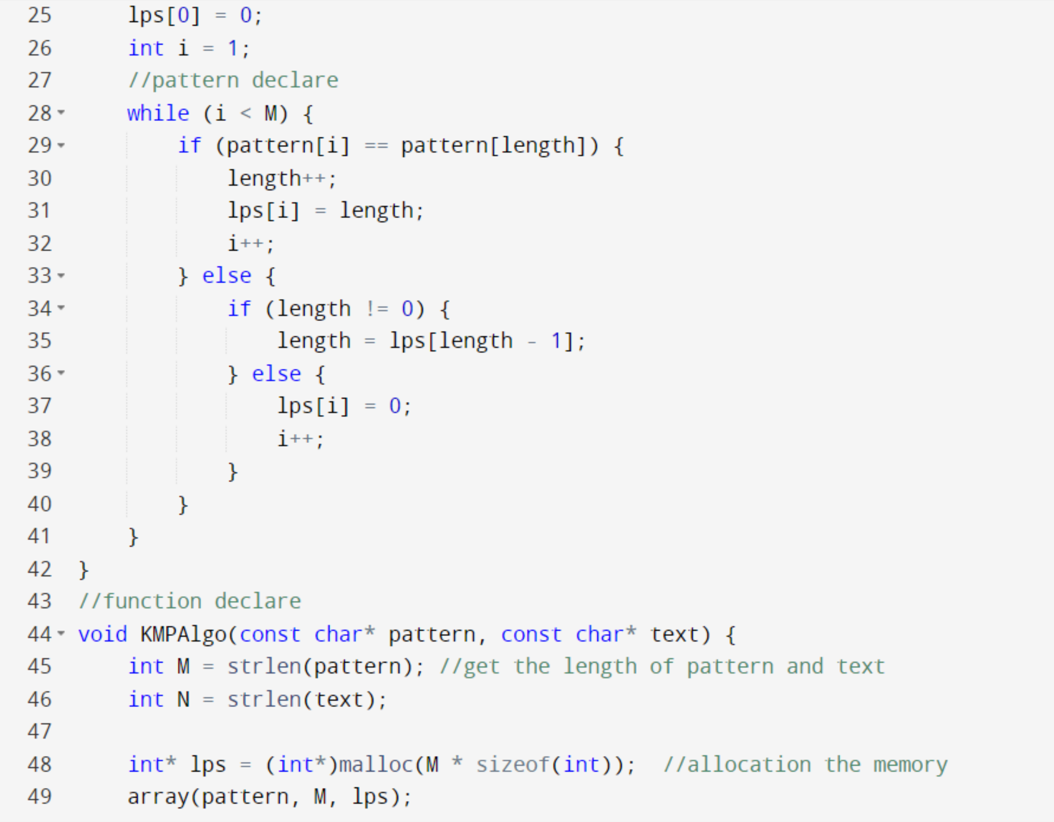
**Pattern Matching :**

I used Knuth-Morris-Pratt algorithm. This algorithm is a string searching algorithm that efficiently finds occurrences of a given pattern within a longer text. Making it more efficient than other naïve pattern matching algorithm .The KMP algorithm has a time complexity of O(M+N) → N is the length of text and M is the length of pattern. KMP algo is a powerfull and efficient string matching algorithm used to largest text and complex pattern.

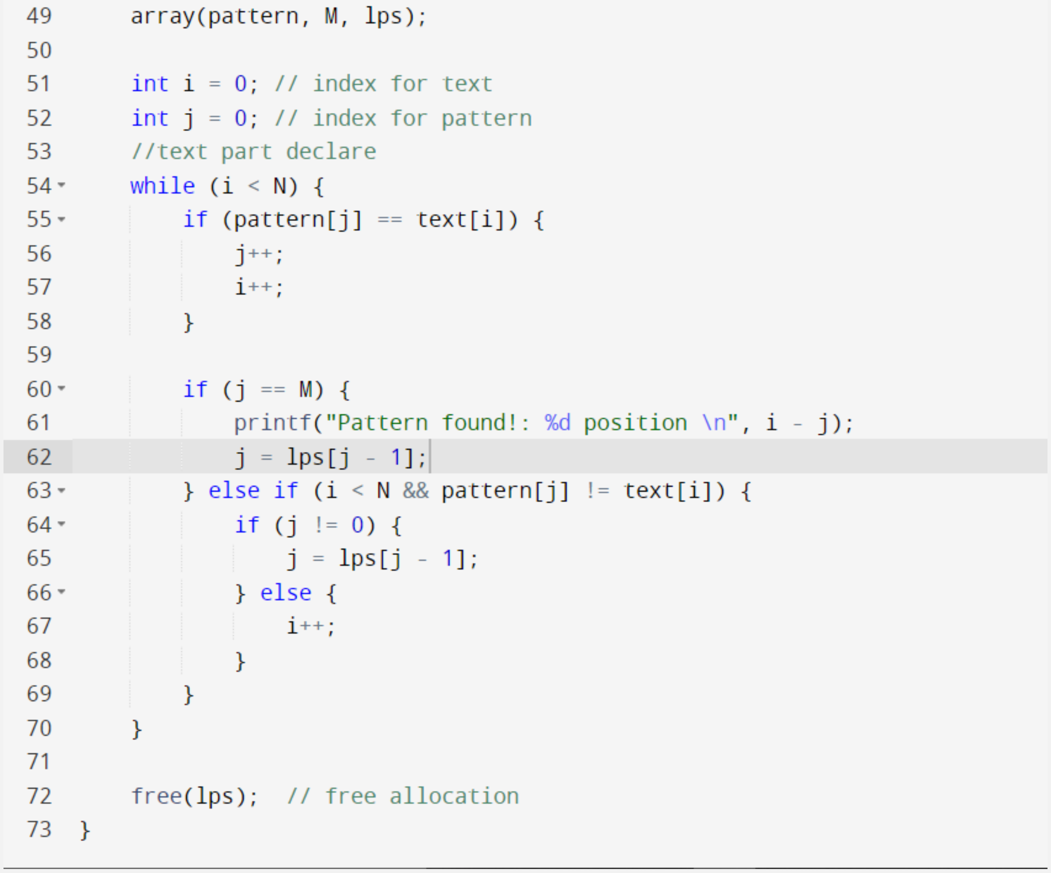
**Source Code :**



```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main() {
    char pattern[100];
    char text[1000];
   //get the user input
    printf("Enter the pattern : ");
    fgets(pattern, sizeof(pattern), stdin);

    printf("Enter the text : ");
    fgets(text, sizeof(text), stdin);

    // Remove the newline character from the input strings
   pattern[strcspn(pattern, "\n")] = '\0';
    text[strcspn(text, "\n")] = '\0';

    KMPAlgo(pattern, text);

    return 0;
}
//lps array declare-> lps means longest proper prefix which is a also suffix
void array(const char* pattern, int M, int* lps) {
    int length = 0;
    lps[0] = 0;
```



```
    lps[0] = 0;
    int i = 1;
    //pattern declare
    while (i < M) {
        if (pattern[i] == pattern[length]) {
            length++;
            lps[i] = length;
            i++;
        } else {
            if (length != 0) {
                length = lps[length - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}
//function declare
void KMPAlgo(const char* pattern, const char* text) {
    int M = strlen(pattern); //get the length of pattern and text
    int N = strlen(text);

    int* lps = (int*)malloc(M * sizeof(int));  //allocation the memory
    array(pattern, M, lps);
```



```
    array(pattern, M, lps);

    int i = 0; // index for text
    int j = 0; // index for pattern
    //text part declare
    while (i < N) {
        if (pattern[j] == text[i]) {
            j++;
            i++;
        }

        if (j == M) {
            printf("Pattern found!: %d position \n", i - j);
            j = lps[j - 1];
        } else if (i < N && pattern[j] != text[i]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }

    free(lps);  // free allocation
}
```

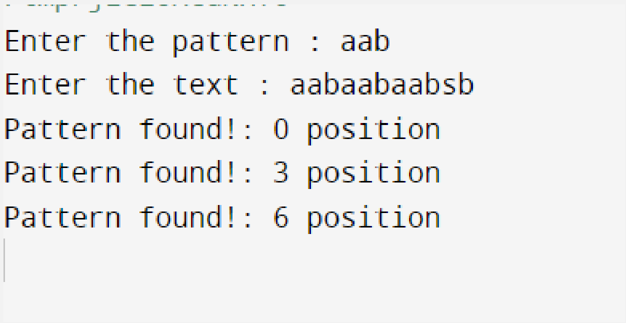
**README File:**

**Compilation :** compile the code using ->” gcc pattern_match.c –o pattern_match –lm” command

**Run** : run this programing using “./pattern_match

**Input files** : text.txt and pattern.txt

**Output :**



```
Enter the pattern : aab
Enter the text : aabaabaabsb
Pattern found!: 0 position
Pattern found!: 3 position
Pattern found!: 6 position
```

**Test data :** Prepare "pattern.txt" and "text.txt" files with different patterns and texts to test the program. Run the program with different input files to verify its correctness.

**Pattern.txt** : [a-z]+

**Text.txt :** This is a sample text for pattern matching. The program will search for all occurrences of lowercase words.

T.H.RAJAPAKSHA

21001545

2021/CS/154

**--------------------END----------------------**