Experiment #9		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Implementation of Programs on Dynamic Programming - I.

Aim/Objective: To understand the concept and implementation of Basic programs on Dynamic Programming problems.

Description: The students will understand and able to implement programs on Dynamic Programming problems.

Pre-Requisites:

Knowledge: Dynamic Programming and its related problems in C/C++/Python

Tools: Code Blocks/Eclipse IDE

Pre-Lab:

A student named Satish is eagerly waiting to attend tomorrow's class. As he searched the concepts of tomorrow's lecture in the course handout and started solving the problems, in the middle he was stroked in the concept of strings because he was poor in this concept so help him so solve the problem, given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert 'str1' into 'str2'.

1. Insert 2.Remove 3.Replace

Input

str1 = "cat", str2 = "cut"

Output

1

We can convert str1 into str2 by replacing 'a' with 'u'.

Input

str1 = "sunday", str2 = "saturday"

Output

3

- **Procedure/Program:**

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

int min(int a, int b, int c) {
   if (a <= b && a <= c) return a;
   if (b <= a && b <= c) return b;
   return c;
}

int minEditDistance(char str1[], char str2[]) {
   int m = strlen(str1);
   int n = strlen(str2);
   int dp[m + 1][n + 1];

   for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
         if (i == 0) {
            dp[i][j] = j;
         }
         else if (j == 0) {
            dp[i][j] = i;
         }
         else if (str1[i - 1] == str2[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1];
         }
         else {
            dp[i][j] = 1 + min(dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]);
         }
      }
   }

   return dp[m][n];
}

int main() {
   char str1[] = "cat";
```

```
    char str2[] = "cut";
    printf("Minimum edit distance: %d\n", minEditDistance(str1, str2));

    char str3[] = "sunday";
    char str4[] = "saturday";
    printf("Minimum edit distance: %d\n", minEditDistance(str3, str4));

    return 0;
}
```

- **Data and Result:**

Data:

- Input 1: str1 = "cat", str2 = "cut".
- Input 2: str1 = "sunday", str2 = "saturday".

Result:

- Minimum edit distance for "cat" to "cut" is 1.
- Minimum edit distance for "sunday" to "saturday" is 3.

- **Inference Analysis:**

Analysis:

- The algorithm uses dynamic programming to calculate the distance.
- It considers insertion, deletion, and replacement operations.

Inferences:

- Edit distance measures similarity between two strings efficiently.
- Dynamic programming optimizes the solution with time complexity O(m * n).

Experiment #**9**		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

In-Lab:

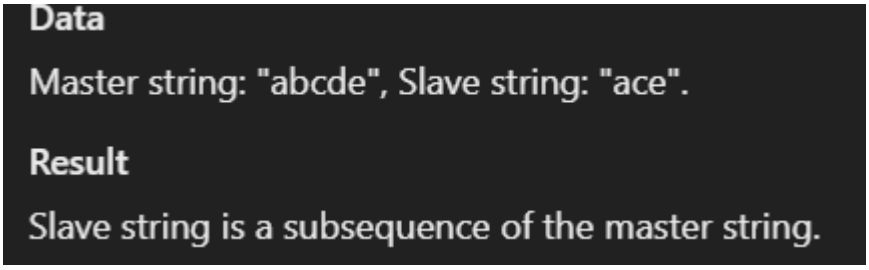
Bhanu is a student at KL University who likes playing with strings, after reading a questionfrom their lab workbook for the ADA Course she found what is meant by a subsequence. (A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements). So, she created 2 strings out of which one she considered as a master string and the other one as a slave string. She challenged her friend Teju to find out whether the slave string is a subsequence ofthe master string or not, As Teju is undergoing her CRT classes she decided to code the logic for this question. Help her in building the logic and write a code using Dynamic programming concept.

- **Procedure/Program:**

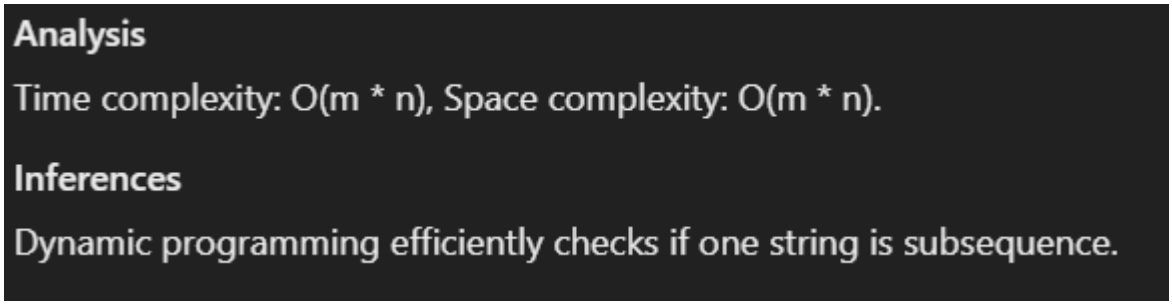
```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

bool isSubsequence(char *master, char *slave) {
   int m = strlen(master);
   int n = strlen(slave);

   bool dp[m+1][n+1];

   for (int i = 0; i <= m; i++) {
      dp[i][0] = true;
   }

   for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= n; j++) {
         if (master[i - 1] == slave[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1];
         } else {
            dp[i][j] = dp[i - 1][j];
         }
      }
   }

   return dp[m][n];
```

```
}

int main() {
   char master[] = "abcde";
   char slave[] = "ace";

   if (isSubsequence(master, slave)) {
      printf("Yes, the slave string is a subsequence of the master string.\n");
   } else {
      printf("No, the slave string is not a subsequence of the master string.\n");
   }

   return 0;
}
```

- **Data and Results:**



- **Analysis and Inferences:**



Post-Lab:

SIRI studies at KL University and a person who is interested in Dynamic Programming, she created a question for you to solve. She decided to give a question related to palindrome, you need to use dynamic programming to solve this problem brute force is not allowed since she hates waiting too long to find the answer. The question follows like this find the longest palindromic subsequence. (Unlike substrings, subsequences are not required to occupy consecutive positions within theoriginal string.)

Input

ABBDCACB

Output

The length of the longest palindromic subsequence is 5

The longest palindromic subsequence is BCACB

- **Procedure/Program:**

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

#define MAX 1000

void longest_palindromic_subsequence(char s[]) {
   int n = strlen(s);
   int dp[MAX][MAX];

   for (int i = 0; i < n; i++) {
      dp[i][i] = 1;
   }

   for (int len = 2; len <= n; len++) {
      for (int i = 0; i < n - len + 1; i++) {
         int j = i + len - 1;
         if (s[i] == s[j]) {
            dp[i][j] = dp[i + 1][j - 1] + 2;
         } else {
            dp[i][j] = (dp[i + 1][j] > dp[i][j - 1]) ? dp[i + 1][j] : dp[i][j - 1];
         }
```

```
    }
}

printf("The length of the longest palindromic subsequence is %d\n", dp[0][n - 1]);

int i = 0, j = n - 1;
char subseq[MAX];
int index = 0;

while (i <= j) {
    if (s[i] == s[j]) {
        subseq[index++] = s[i];
        i++;
        j--;
    } else if (dp[i + 1][j] >= dp[i][j - 1]) {
        i++;
    } else {
        j--;
    }
}

char second_half[MAX];
int k = 0;
if (dp[0][n - 1] % 2 == 0) {
    for (int l = index - 1; l >= 0; l--) {
        second_half[k++] = subseq[l];
    }
} else {
    for (int l = index - 2; l >= 0; l--) {
        second_half[k++] = subseq[l];
    }
}
subseq[index] = '\0';

printf("The longest palindromic subsequence is ");
for (int m = 0; m < index; m++) {
    printf("%c", subseq[m]);
}
```

```
    for (int m = 0; m < k; m++) {
        printf("%c", second_half[m]);
    }
    printf("\n");
}

int main() {
    char s[] = "ABBDCACB";
    longest_palindromic_subsequence(s);
    return 0;
}
```

- **Data and Results:**

Data:

Input string is "ABBDCACB" for finding the longest subsequence.

Result:

The longest palindromic subsequence is BCACB, with length 5.

- **Analysis and Inferences:**

Analysis:

DP table is used to calculate subsequence length and reconstruct sequence.

Inferences:

Dynamic programming optimally finds subsequence, reducing brute force computation tim

Experiment #9		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the difference between top-down and bottom-up approaches in dynamic programming?

Top-down vs Bottom-up in Dynamic Programming:

- **Top-down**: Uses recursion and stores results (memoization).
- **Bottom-up**: Solves from smallest subproblems to larger ones iteratively.

2. What is memorization and how does it relate to dynamic programming?

Memoization in Dynamic Programming:

- **Memoization**: Storing results of subproblems to avoid repeated work.
- **Relation**: It's used in the top-down approach of dynamic programming to save computed results.

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.