Fibonacci using Dynamic Programming

```
#include <stdio.h>


// Function to calculate Fibonacci number using dynamic programming (tabulation)
int fibonacci(int n) {
    if (n <= 1) {
        return n; // Base case: F(0) = 0, F(1) = 1
    }
    int dp[n + 1];
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2]; // Fill the dp array from bottom-up
    }
    return dp[n];
}


int main() {
    int n = 10;
    printf("Fibonacci number at position %d is %d\n", n, fibonacci(n));
    return 0;
}
```

Edit Distance using Dynamic Programming

```
#include <stdio.h>
#include <string.h>
int min(int x, int y, int z) {
   return (x < y) ? ((x < z) ? x : z) : ((y < z) ? y : z);
}
int editDistance(char* str1, char* str2, int m, int n) {
   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; // If first string is empty, insert all characters of second string
         } else if (j == 0) {
            dp[i][j] = i; // If second string is empty, remove all characters of first string
         } else if (str1[i - 1] == str2[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1]; // If characters are the same, no operation needed
         } else {
            dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]); // Min of insert, delete, replace
         }
      }
   }

   return dp[m][n];
}
int main() {
   char str1[] = "sunday";
   char str2[] = "saturday";
   printf("Minimum edit distance is %d\n", editDistance(str1, str2, strlen(str1), strlen(str2)));
   return 0;
}
```

Knapsack using Dynamic Programming

```
#include <stdio.h>
int max(int a, int b) {
   return (a > b) ? a : b;
}
int knapsack(int val[], int wt[], int n, int W) {
   int dp[n + 1][W + 1];
   for (int i = 0; i <= n; i++) {
      for (int w = 0; w <= W; w++) {
         if (i == 0 || w == 0) {
            dp[i][w] = 0; // Base case: If no items or no capacity, value is 0
         } else if (wt[i - 1] <= w) {
            dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]); // Include or exclude the item
         } else {
            dp[i][w] = dp[i - 1][w]; // Exclude the item
         }
      }
   }
   return dp[n][W];
}

int main() {
   int val[] = {60, 100, 120};
   int wt[] = {10, 20, 30};
   int W = 50;
   int n = sizeof(val) / sizeof(val[0]);

   printf("Maximum value in Knapsack = %d\n", knapsack(val, wt, n, W));
   return 0;
}
```

LCS using Dynamic Programming

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

// Function to calculate LCS using dynamic programming (tabulation)
#include <stdio.h>
#include <string.h>
int LCS_Tabulation(char *X, char *Y, int m, int n) {
   int L[m+1][n+1];
   for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
         if (i == 0 || j == 0)
            L[i][j] = 0;
         else if (X[i-1] == Y[j-1])
            L[i][j] = L[i-1][j-1] + 1;
         else
            L[i][j] = (L[i-1][j] > L[i][j-1]) ? L[i-1][j] : L[i][j-1];
      }
   }
   return L[m][n];
}
int main() {
   char X[] = "ad";
   char Y[] = "abcd";
   int m = strlen(X);
   int n = strlen(Y);
   printf("Length of LCS (Tabulation) is %d\n", LCS_Tabulation(X, Y, m, n));
   return 0;
}
```