```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <stdbool.h>

// global variables
enum {LCS, ED, SW, NONE} alg_type; // which algorithm to run
char *alg_desc; // description of which algorithm to run
char *result_string; // text to print along with result from algorithm
char *x, *y; // the two strings that the algorithm will execute on
char *filename; // file containing the two strings
int xLen, yLen, alphabetSize; // lengths of two strings and size of alphabet
bool iterBool = false, recNoMemoBool = false, recMemoBool = false; // which type
 of dynamic programming to run
bool printBool = false; // whether to print table
bool readFileBool = false, genStringsBool = false; // whether to read in strings
 from file or generate strings randomly

// functions follow

// determine whether a given string consists only of numerical digits
bool isNum(char s[]) {
        int i;
        bool isDigit=true;
        for (i=0; i<strlen(s); i++)
                isDigit &= s[i]>='0' && s[i]<='9';
        return isDigit;
}

// get arguments from command line and check for validity (return true if and on
ly if arguments illegal)
bool getArgs(int argc, char *argv[]) {
        int i;
        alg_type = NONE;
        xLen = 0;
        yLen = 0;
        alphabetSize = 0;
        for (i = 1; i < argc; i++) // iterate over all arguments provided (argum
ent 0 is name of this module)
                if (strcmp(argv[i],"-g")==0) { // generate strings randomly
                        if (argc>=i+4 && isNum(argv[i+1]) && isNum(argv[i+2]) &&
 isNum(argv[i+3])) { // must be three numerical arguments after this
                                xLen=atoi(argv[i+1]); // get length of x
                                yLen=atoi(argv[i+2]); // get length of y
                                alphabetSize = atoi(argv[i+3]); // get alphabet
size
                                genStringsBool = true; // set flag to generate s
trings randomly
                                i+=3; // ready for next argument
                        }
                        else
                                return true; // must have been an error with -g
arguments
                }
                else if (strcmp(argv[i],"-f")==0) { // read in strings from file
                        if (argc>=i+2) { // must be one more argument (filename)
 after this)
                                i++;
                                filename = argv[i]; // get filename
                                readFileBool = true; // set flag to read in stri
```



```
ngs from file
                        }
                        else
                                return true; // must have been an error with -f
argument
                }
                else if (strcmp(argv[i],"-i")==0) // iterative dynamic programmi
ng
                        iterBool = true;
                else if (strcmp(argv[i],"-r")==0) // recursive dynamic programmi
ng without memoisation
                        recNoMemoBool = true;
                else if (strcmp(argv[i],"-m")==0) // recursive dynamic programm
ing with memoisation
                        recMemoBool = true;
                else if (strcmp(argv[i],"-p")==0) // print dynamic programming t
able
                        printBool = true;
                else if (strcmp(argv[i],"-t")==0) // which algorithm to run
                        if (argc>=i+2) { // must be one more argument ("LCS" or
"ED" or "SW")
                                i++;
                                if (strcmp(argv[i],"LCS")==0) { // Longest Comm
on Subsequence
                                        alg_type = LCS;
                                        alg_desc = "Longest Common Subsequence";
                                        result_string = "Length of a longest common subs
equence is";
                                }
                                else if (strcmp(argv[i],"ED")==0) { // Edit Dis
tance
                                        alg_type = ED;
                                        alg_desc = "Edit Distance";
                                        result_string = "Edit distance is";
                                }
                                else if (strcmp(argv[i],"SW")==0) { // Smith-Wa
terman Algorithm
                                        alg_type = SW;
                                        alg_desc = "Smith-Waterman algorithm";
                                        result_string = "Length of a highest scoring local s
imilarity is";
                                }
                                else
                                        return true; // none of these; illegal c
hoice
                        }
                        else
                                return true; // algorithm type not given
                else
                        return true; // argument not recognised
                // check for legal combination of choices; return true (illegal)
 if user chooses:
                // - neither or both of generate strings and read strings from f
ile
                // - generate strings with length 0 or alphabet size 0
                // - no algorithm to run
                // - no type of dynamic programming
                return !(readFileBool ^ genStringsBool) || (genStringsBool && (x
Len <=0 || yLen <= 0 || alphabetSize <=0)) || alg_type==NONE || (!iterBool && !r
ecMemoBool && !recNoMemoBool);
}
```

```
// read strings from file; return true if and only if file read successfully
bool readStrings() {
        // open file for read given by filename
        FILE * file;
        file = fopen(filename, "r");
        // firstly we will measure the lengths of x and y before we read them in
 to memory
        if (file) { // file opened successfully
                // first measure length of x
                bool done = false;
                int i;
                do { // read from file until newline encountered
                        i = fgetc(file); // get next character
                        if (i==EOF) { // EOF encountered too early (this is firs
t string)
                                // print error message, close file and return fa
lse
                                printf("Incorrect file syntax\n");
                                fclose(file);
                                return false;
                        }
                        if ((char) i=='\n' || (char) i=='\r') // newline encounte
red
                                done = true; // terminate loop
                        else // one more character
                                xLen++; // increment length of x
                } while (!done);
                // next measure length of y
                if ((char) i=='\r')
                        fgetc(file); // get rid of newline character
                done = false;
                do { // read from file until newline or EOF encountered
                        int i = fgetc(file); // get next character
                        if (i==EOF || (char) i=='\n' || (char) i=='\r') // EOF or
 newline encountered
                                done = true; // terminate loop
                        else // one more character
                                yLen++; // increment length of y
                } while (!done);
                fclose(file);
                // if either x or y is empty then print error message and return
 false
                if (xLen==0 || yLen==0) {
                        printf("Incorrect file syntax\n");
                        return false;
                }
                // now open file again for read
                file = fopen(filename, "r");
                // allocate memory for x and y
                x = malloc(xLen * sizeof(char));
                y = malloc(yLen * sizeof(char));
                // read in x character-by-character
                for (i=0; i<xLen; i++)
                        x[i]=fgetc(file);
                i = fgetc(file); // read in newline between strings and discard
                if ((char) i=='\r')
                        fgetc(file); // read \n character and discard if previou
s character was \r
                // read in y character-by-character
                for (i=0; i<yLen; i++)
                        y[i]=fgetc(file);
                // close file and return boolean indicating success
```



```
                fclose(file);
                return true;
        }
        else { // notify user of I/O error and return false
                printf("Problem opening file %s\n",filename);
                return false;
        }
}

// generate two strings x and y (of lengths xLen and yLen respectively) uniforml
y at random over an alphabet of size alphabetSize
void generateStrings() {
        // allocate memory for x and y
        x = malloc(xLen * sizeof(char));
        y = malloc(yLen * sizeof(char));
        // instantiate the pseudo-random number generator (seeded based on curre
nt time)
        srand(time(NULL));
        int i;
        // generate x, of length xLen
        for (i = 0; i < xLen; i++)
                x[i] = rand()%alphabetSize +'A';
        // generate y, of length yLen
        for (i = 0; i < yLen; i++)
                y[i] = rand()%alphabetSize +'A';
}

// free memory occupied by strings
void freeMemory() {
        free(x);
        free(y);
}

// main method, entry point
int main(int argc, char *argv[]) {
        bool isIllegal = getArgs(argc, argv); // parse arguments from command li
ne
        if (isIllegal) // print error and quit if illegal arguments
                printf("Illegal arguments\n");
        else {
                printf("%s\n", alg_desc); // confirm algorithm to be executed
                bool success = true;
                if (genStringsBool)
                        generateStrings(); // generate two random strings
                else
                        success = readStrings(); // else read strings from file
                if (success) { // do not proceed if file input was problematic
                        // confirm dynamic programming type
                        // these print commamds are just placeholders for now
                        if (iterBool)
                                printf("Iterative version\n");
                        if (recMemoBool && (alg_type==LCS || alg_type==ED))
                                printf("Recursive version with memoisation\n");
                        if (recNoMemoBool && (alg_type==LCS || alg_type==ED))
                                printf("Recursive version without memoisation\n");
                        freeMemory(); // free memory occupied by strings
                }
        }
        return 0;
}
```