```
#include <stdio.h>
#include "hashmap.h"
#define Q_SIZE 5


/**
Array Definition and Functions
**/
typedef struct {
  int *array;
  size_t used;
  size_t size;
} Array;

void initArray(Array *a, size_t initialSize);
void insertArray(Array *a, int element);
int isInArray(Array *a, int element);
void freeArray(Array *a);


/**
String Preparation Functions
**/
void encode_qgram(char *s);
void strip_char(char *str, char strip);
void prepare_and_copy_str(char *original, char *dest);


/**
Bit Streak Calculation Functions
**/
void append_weights(int I[], map_t map, char* qgram, int W);
int get_set_bits(int n);
int populate_streak_max_arr(Array *M, int I[], int length_of_S);


/**
String Extraction Functions
**/
int count_lines(FILE *f);
void extract_strings(FILE *f, char *S[]);


/**
QGram Extraction And Hashmap Population Functions
**/
typedef struct data_struct_s{
    char key_string[Q_SIZE];
    Array indices;
} data_struct_t;

void populate_qgram_array(char *s, char *Q_of_s[], int num_of_qgrams);
void add_qgram_to_map(map_t *map, char *qgram, int index);
```

```c
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>
#include "qgjoin.h"
#include "hashmap.h"

static const uint_fast8_t encoding_table[128] = {
    // Punctuation
        [' '] = 28,      ['-'] = 28, ['_'] = 28,

        // Numbers
        ['0'] = 'O' - '@', ['1'] = 'I' - '@', ['2'] = 'Z' - '@',
    ['3'] = 27, ['4'] = 'A' - '@', ['5'] = 'S' - '@', ['6'] = 'G' - '@',
    ['7'] = 'T' - '@', ['8'] = 'B' - '@', ['9'] = 'Q' - '@',

    // Capital letters
    ['A'] = 'A' - '@', ['B'] = 'B' - '@', ['C'] = 'C' - '@', ['D'] = 'D' - '
@',
    ['E'] = 'E' - '@', ['F'] = 'F' - '@', ['G'] = 'G' - '@', ['H'] = 'H' - '@
',
    ['I'] = 'I' - '@', ['J'] = 'J' - '@', ['K'] = 'K' - '@', ['L'] = 'L' - '@',
    ['M'] = 'M' - '@', ['N'] = 'N' - '@', ['O'] = 'O' - '@', ['P'] = 'P' - '
@',
    ['Q'] = 'Q' - '@', ['R'] = 'R' - '@', ['S'] = 'S' - '@', ['T'] = 'T' - '@
',
    ['U'] = 'U' - '@', ['V'] = 'V' - '@', ['W'] = 'W' - '@', ['X'] = 'X' - '
@',
    ['Y'] = 'Y' - '@', ['Z'] = 'Z' - '@',

    // lower-case letters
    ['a'] = 'A' - '@', ['b'] = 'B' - '@', ['c'] = 'C' - '@', ['d'] = 'D' - '@'
,
    ['e'] = 'E' - '@', ['f'] = 'F' - '@', ['g'] = 'G' - '@', ['h'] = 'H' - '@'
,
    ['i'] = 'I' - '@', ['j'] = 'J' - '@', ['k'] = 'K' - '@', ['l'] = 'L' - '@',
    ['m'] = 'M' - '@', ['n'] = 'N' - '@', ['o'] = 'O' - '@', ['p'] = 'P' - '@
',
    ['q'] = 'Q' - '@', ['r'] = 'R' - '@', ['s'] = 'S' - '@', ['t'] = 'T' - '@',
    ['u'] = 'U' - '@', ['v'] = 'V' - '@', ['w'] = 'W' - '@', ['x'] = 'X' - '
@',
    ['y'] = 'Y' - '@', ['z'] = 'Z' - '@',
};


/**
Initialize an Array.

    :param a: Array for which memory is to be allocated for.
    :type a: Struct Array ptr

    :param initialSize: initial length of the array.
    :type initialSize: size_t

    :return: void
    :return type: void
**/
void initArray(Array *a, size_t initialSize) {
  a->array = (int *)malloc(initialSize * sizeof(int));
```

```c
  a->used = 0;
  a->size = initialSize;
};


/**
Insert given element into array.

    :param a: Array into which the given element is to be inserted.
    :type a: Struct Array ptr

    :param element: element to be looked up.
    :type element: int

    :return: void
    :return type: void
**/
void insertArray(Array *a, int element) {
  if (a->used == a->size) {
    a->size *= 2;
    a->array = (int *)realloc(a->array, a->size * sizeof(int));
  }
  a->array[a->used++] = element;
}


/**
Check if given integer is present in the given Array a.

    :param a: Array in which the given element's presence is to be checked.
    :type a: Struct Array ptr

    :param element: element to be looked up.
    :type element: int

    :return: True or False as int value
    :return type: int
**/
int isInArray(Array *a, int element){
    int i;
    if (a->used!=0){
        for(i=0; i < a->used; i++){
            if(a->array[i]==element){
                return 1;
            }
        }
    }
    return 0;
}


/**
Free allocated memory for given Array.

    :param a: Array whose allocated memory is to be freed.
    :type a: Struct Array ptr

    :return: void
    :return type: void
**/
void freeArray(Array *a) {
  free(a->array);
  a->array = NULL;
  a->used = a->size = 0;
}


/*
Convert to 6bit ASCII. Expects Null terminated string.

    :param s: string to be converted to 6Bit ASCII.
    :type s: char ptr

    :return: void
    :return type: void
*/
void encode_qgram(char *s){
    char* p;
    for (p=s; *p!='\0'; ++p){
       *p = encoding_table[*p];
    }
}


/**
Strip a character x from given string s in-place. Expects Null-terminated string

    :param str: string to strip given character from
    :type str: char ptr

    :param strip: character to strip from str
    :type strip: char

    :return: void
    :return type: void
**/
void strip_char(char *str, char strip){
    char *p, *q;
    for (q = p = str; *p!='\0'; p++)
        if (*p != strip)
            *q++ = *p;
    *q++ = '\0';
}


/**
Prepare a string for Qgram extraction.

    :param original: string to be prepared for extraction. Must be Null terminat
ed.
    :type original: char ptr

    :param dest: Destination char ptr to store modified original in.
    :type dest: char ptr

    :return: void
    :return type: void
**/
void prepare_and_copy_str(char *original, char *dest){
    memcpy(dest, original, strlen(original)+1);
    strip_char(dest, '\n');
    strip_char(dest, ',');
    strip_char(dest, '.');
```

```c
    char* p;
    for (p=dest; *p!='\0'; ++p){
       *p = toupper(*p);
    }
}


/**
Returns sum of bits set for given integer.

    :param n: integer to count bits in.
    :type n: int

    :return: sum of set bits of n.
    :return type: int
**/
int get_set_bits(int n){
    unsigned int count = 0;
    while(n){
        n &= (n-1);
        count++;
    }
    return count;
}


/**
Add weights for the given qgram at the index of the string its
related to in I.

    :param I: array holding the weights of each String of S.
    :type I: int array

    :param map: Hashmap object storing qgram:indices pairs.
    :type map: map_t

    :param qgram: Qgram string for which the weight will be added.
    :type qgram: char ptr

    :param W: The weight to be added to I.
    :type W: int

    :return: void
    :return type: void
**/
void append_weights(int I[], map_t map, char* qgram, int W){
    data_struct_t* value;
    int j;
    if(hashmap_get(map, qgram, (void**)(&value))==MAP_OK){
        for(j=0; j < value->indices.used; j++){
            I[value->indices.array[j]] += W;
        }
    }
}


/**
Calculate streaks and get max streak length and respective string indices;
M is the set of indices of strings with the most occurrences of qgrams.

    :param M: Dynamically allocated Array Struct ptr, which holds all indices wh
ose
```

```c
              bit streak length is equal to the longest bit streak length.
    :type M: Array Pointer

    :param I: array holding the weights of each String of S.
    :type I: int array

    :param length_of_S: length of char ptr array S.
    :type length_of_S: int

    :return: max
    :return type: int
**/
int populate_streak_max_arr(Array *M, int I[], int length_of_S){
    int streak = 0;
    int max = 0;
    int i;

    for(i=0; i<length_of_S; i++){
        streak = get_set_bits(I[i]);
        if(streak > max){
            max = streak;
            freeArray(M);
            initArray(M, 8);
            insertArray(M, i);
        }
        else if (streak == max && streak != 0){
            insertArray(M, i);
        }
    }
    return max;
}


/**
extract strings from file and assign to array of strings.

    :param f: file pointer to read from
    :type f: FILE ptr

    :param S: char ptr array to store read lines in
    :type S: char* []

    :return: void
    :return type: void
**/
void extract_strings(FILE *f, char *S[]){
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    int i = 0;
    while((read = getline(&line, &len, f)) != -1){
        strip_char(line, '\n');
        S[i] = strdup(line);
        i++;
    }
}


/**
Count number of lines, i.e. strings in given file

    :param f: file ptr to count lines from
```

```c
    :type f: FILE ptr

    :return: number of counted lines
    :return type: int
**/
int count_lines(FILE *f){
    char c;
    int counter = 0;
    for (c = getc(f); c != EOF; c = getc(f)){
        if (c == '\n'){
            counter++;
        }
    }
    return counter;
}


/**
Extract QGrams from given string to given array;

    :param s: string array which contains the string to be split into Qgrams.
    :type s: char ptr

    :param Q_of_s: Array in which to store the extracted Qgrams.
    :type Q_of_s: char ptr array

    :param len_of_s: length of s
    :type len_of_s: int

    :return: void
    :return type: void
**/
void populate_qgram_array(char *s, char *Q_of_s[], int len_of_s){
    int i = 0;
    while(i + Q_SIZE-1 < len_of_s){
        Q_of_s[i] = malloc(Q_SIZE+1);
        memcpy(Q_of_s[i], s+i, Q_SIZE+1);
        Q_of_s[i][Q_SIZE] = '\0';
        encode_qgram(Q_of_s[i]);
        i++;
    }
}


/**
Add given qgram with given index to given hashtable.
Expects '\0' terminated strings.

    :param map: Hashmap ptr to which the key value pair is to be added.
    :type map: map_t ptr

    :param qgram: string to use as key to store index
    :type qgram: char ptr

    :param index: index to be stored in hashmap at key qgram
    :type index: int

    :return: void
    :return type: void
**/
void add_qgram_to_map(map_t *map, char *qgram, int index){
    int error;
    data_struct_t* value;

    if(hashmap_get(*map, qgram, (void**)(&value))==MAP_OK){
        insertArray(&(value->indices), index);
    }
    else{
        value = malloc(sizeof(data_struct_t));
        memcpy(value->key_string, qgram, Q_SIZE+1);
        initArray(&(value->indices), 8);
        insertArray(&(value->indices), index);
        error = hashmap_put(*map, value->key_string, value);
        assert(error==MAP_OK);
    }
}
```

```c
#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <assert.h>
#include "hashmap.h"
#include "qgjoin.h"


#define KEY_MAX_LENGTH (Q_SIZE)

void print_help(void){
        printf("Usage: qgjoin [OPTION] LEFT_LIST RIGHT_LIST\n");
        printf("  or: cat RIGHT_LIST | qgjoin LEFT_LIST \n");
        printf("\tCompare two lists of strings using a Q-gram-based Fuzzy Join algorithm.\n");
        printf("\tLEFT_LIST is expected to be a path to a file. It is read entirely into memory;\n");
        printf("\tit is your job to make sure it fits. The RIGHT_LIST argument can either be a file\n");
        printf("\tpath, or input piped via STDIN.\n");
        printf("\nLEFT_LIST\t\tList of strings to be kept in memory.\n");
        printf("\t\t\tType: FILE PATH");
        printf("\nRIGHT_LIST\t\tList of strings to compare LEFT_LIST against.\n");
        printf("\t\t\tType: FILE PATH or STDIN\n");
        printf("\n--help, -h\t\tDisplay this help output");
        printf("\n--version, -v\t\tDisplay the version of the program.\n");
}

void print_version(void){
        printf("QGJoin 1.0\n");
        printf("Copyright (C) 2007 Nils Diefenbach.\n");
        printf("License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>\n");
        printf("This is free software: you are free to change and redistribute it.\n");
        printf("There is NO WARRANTY, to the extent permitted by law.\n");
}

int main(int argc, char *argv[]){
    /*
    Outline

    1.) Load data from file using argv[1] and assign to S;

    2.) Get all QGrams of S and assign to Q, convert to 31 Char Alphabet;

    3.) Open file or stdin using argv[2], open with ptr f;

    4.) For line in f:
            copy, strip chars from and capitalize string t from line
            get qgrams of t
            get indices of strings in S for qgrams that match with qgrams in Q
            get bit streaks for qgrams
            filter q_grams by streak fulfilling condition X
            output q_grams with longest streaks

    5.) Clean up variables and allocated memory, exit.

    */

    if(argc < 2 || argc > 3){
        printf("qgjoin requires at least 1 file name as argument and no more than two file names! Type 'qgjoin
--help' for more information\n");
        exit(1);
```

```c
    }
    const char *argv1 = argv[1];
    const char *HELP_long = "--help";
    const char *HELP_short = "-h";
    const char *VERSION_long = "--version";
    const char *VERSION_short = "-v";

    if(!strcmp(argv1, HELP_long) || !strcmp(argv1, HELP_short)){
        print_help();
        exit(0);
    }
    else if(!strcmp(argv1, VERSION_long) || !strcmp(argv1, VERSION_short)){
        print_version();
        exit(0);
    }


    FILE *LEFT_LIST;
    FILE *RIGHT_LIST;
    LEFT_LIST = fopen(argv[1], "r");

    if (LEFT_LIST == NULL){
        printf("Could not open file %s!\n", argv[1]);
        exit(1);
    }

    if(argc == 3 && strcmp(argv[2], "-")){
        RIGHT_LIST = fopen(argv[2], "r");
        if(RIGHT_LIST == NULL){
            printf("Could not open file %s\n", argv[2]);
            exit(1);
        }
    }
    else{
        RIGHT_LIST = stdin;
    }


    int string_count_S;
    string_count_S = count_lines(LEFT_LIST);
    fseek(LEFT_LIST, 0, SEEK_SET);
    char **S;
    S = malloc(string_count_S * sizeof(char*));
    extract_strings(LEFT_LIST, S);
    fclose(LEFT_LIST);

    int i;
    int j;
    int qgram_count;
    map_t Q_of_S;
    map_t* Map;
    Map = &Q_of_S;
    Q_of_S = hashmap_new();

    for(i = 0; i < string_count_S; i++){

        char s[strlen(S[i])];
        prepare_and_copy_str(S[i], s);
        qgram_count = strlen(s) - (Q_SIZE-1);
        char *Qgrams_s[qgram_count];
        populate_qgram_array(s, Qgrams_s, strlen(s));
        for(j=0; j < qgram_count; j++){
            add_qgram_to_map(Map, Qgrams_s[j], i);
        }

        data_struct_t* value;
        for(j=0; j < qgram_count; j++){
            add_qgram_to_map(Map, Qgrams_s[j], i);
        }
    }



    char *line;
    size_t len = 0;
    ssize_t read;
    int W;
    int max;
    int *I = malloc(string_count_S * sizeof(int));
    while((read = getline(&line, &len, RIGHT_LIST)) != -1){
        if(strlen(line) < Q_SIZE){
            continue;
        }

        char t[strlen(line)];
        prepare_and_copy_str(line, t);

        qgram_count = strlen(t) - (Q_SIZE-1);
        char *Q_of_t[qgram_count];
        char *qgram_debug[qgram_count];
        populate_qgram_array(t, Q_of_t, strlen(t));


        i = 0;
        while(i + Q_SIZE-1 < strlen(t)){
            qgram_debug[i] = malloc(Q_SIZE+1);
            memcpy(qgram_debug[i], t+i, Q_SIZE+1);
            qgram_debug[i][Q_SIZE] = '\0';
            i++;
        }

        memset(I, 0, string_count_S * sizeof(int));
        W = 1;
        for(i=0; i<qgram_count; i++){
            append_weights(I, Q_of_S, Q_of_t[i], W);
            W *= 2;
        }

        Array M;
        Array* M_ptr = &M;
        initArray(&M, 8);
        max = populate_streak_max_arr(M_ptr, I, string_count_S);

        if (max > 0){
            for(i=0; i < M.used; i++){
                printf("%s\t%s\t%d\n", S[M.array[i]], t, max);
            }
        }
        freeArray(&M);

    }

    fclose(RIGHT_LIST);
    data_struct_t* value;
```

```
    int error;
    for(i = 0; i < string_count_S; i++){

        char s[strlen(S[i])];
        prepare_and_copy_str(S[i], s);
        qgram_count = strlen(s) - (Q_SIZE-1);
        char *q;
        q = malloc(Q_SIZE+1);
        while(i + Q_SIZE-1 < strlen(s)){
            memcpy(q, s+i, Q_SIZE+1);
            q[Q_SIZE] = '\0';
            encode_qgram(q);

            error = hashmap_get(Q_of_S, q, (void**)(&value));
            assert(error==MAP_OK);

            error = hashmap_remove(Q_of_S, q);
            assert(error==MAP_OK);

            i++;
         }
        free(q);

    }
    hashmap_free(Q_of_S);

    for(i=0; i < string_count_S; i++){
        free(S[i]);
    }
    free(S);
    free(I);
    return 0;
}
```