Padron 89397 **arguments.c** Page 1/1

```c
//
//  arguments.c
//  TP1
//
//  Created by GastÃ³n Montes on 08/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>

#include "arguments.h"
#include "string_functions.h"

static char *const kCommaSeparator  = ",";

void argumentsCreate(Arguments *arguments, char *arguments_line) {
    char *token;

    // Read equals param value.
    char *save_str;
    token = stringStrtok_r(arguments_line, kCommaSeparator, &save_str);
    arguments->equals = atoi(token);

    // Read differents param value.
    token = stringStrtok_r(NULL, kCommaSeparator, &save_str);
    arguments->different = atoi(token);

    // Read gap param value.
    token = stringStrtok_r(NULL, kCommaSeparator, &save_str);
    arguments->gap = atoi(token);

    // Read max length param value.
    token = stringStrtok_r(NULL, kCommaSeparator, &save_str);
    arguments->max_length = atoi(token);
}
```

Padron 89397 **arguments.h** Page 1/1

```c
//
//  arguments.h
//  TP1
//
//  Created by GastÃ³n Montes on 08/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

#ifndef TP1_arguments_h
#define TP1_arguments_h

typedef struct {
    int equals;
    int different;
    int gap;
    int max_length;
} Arguments;

/**
 * @params - arguments_line: the line that contains the arguments in CSV values.
 */
void argumentsCreate(Arguments *arguments, char *arguments_line);

#endif
```

Padron 89397 **arguments_parser.c** Page 1/1

```c
//
//  arguments_parser.c
//  TP1
//
//  Created by GastÃ³n Montes on 02/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

#include <stdio.h>
#include "arguments_parser.h"

static int const FIRST_ARGUMENT_INDEX   = 1;
static int const SECOND_ARGUMENT_INDEX  = 2;

// pragma mark - Private methods.
void argumentsParserParseArguments(ArgumentsParser *parser,
                                   int arguments_count) {
    switch (arguments_count) {
        case 0:
            parser->parse_validation = ArgumentsParserCodeNoArguments;
            break;
        case 1:
            // No arguments passed.
            parser->parse_validation = ArgumentsParserCodeNoArguments;
            break;
        case 2:
            // Only 1 argument.
            parser->parse_validation = ArgumentsParserCodeOneArgument;
            break;
        case 3:
            // 2 arguments ok.
            parser->parse_validation = ArgumentsParserCodeTwoArguments;
            break;
        default:
            // More than 2 arguments.
            parser->parse_validation = ArgumentsParserCodeTooManyArguments;
            break;
    }
    parser->arguments_count = arguments_count;
}

void argumentsParserFirstArgument(ArgumentsParser *parser,
                                  const char *arguments[]) {
    parser->first_argument = NULL;
    if (arguments[FIRST_ARGUMENT_INDEX] != NULL) {
        parser->first_argument = (char *)arguments[FIRST_ARGUMENT_INDEX];
    }
}

void argumentsParserSecondArgument(ArgumentsParser *parser,
                                   const char *arguments[]) {
    parser->second_argument = NULL;
    if (arguments[SECOND_ARGUMENT_INDEX] != NULL) {
        parser->second_argument = (char *)arguments[SECOND_ARGUMENT_INDEX];
    }
}

// pragma mark - Public methods.
void argumentsParserCreate(ArgumentsParser *arguments_parser,
                           int arguments_count,
                           const char *arguments[]) {
    argumentsParserParseArguments(arguments_parser, arguments_count);
    argumentsParserFirstArgument(arguments_parser, arguments);
    argumentsParserSecondArgument(arguments_parser, arguments);
}
```

Padron 89397 **arguments_parser.h** Page 1/1

```c
//
//  arguments_parser.h
//  TP1
//
//  Created by GastÃ³n Montes on 02/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

#ifndef TP1_arguments_parser_h
#define TP1_arguments_parser_h

typedef enum {
    ArgumentsParserCodeNoArguments,
    ArgumentsParserCodeOneArgument,
    ArgumentsParserCodeTwoArguments,
    ArgumentsParserCodeTooManyArguments
} ArgumentsParserCode;

typedef struct {
    ArgumentsParserCode parse_validation;
    char *first_argument;
    char *second_argument;
    int arguments_count;
} ArgumentsParser;

void argumentsParserCreate(ArgumentsParser *arguments_parser,
                           int arguments_count,
                           const char *arguments[]);

#endif
```

Padron 89397 **DNA_sequence.c** Page 1/2

```c
//
//  DNA_sequence.c
//  TP1
//
//  Created by GastÃ³n Montes on 09/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

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

#include "DNA_sequence.h"

static int const kZeroIntValue  = 0;
static int const kOneIntValue   = 1;

void dnaSequenceCreate(DNASequence *sequence, long int position) {
    sequence->comparison_value  = kZeroIntValue;
    sequence->position_in_file  = position;
}

/**
 * Smith-Waterman comparition.
 */
int dnaSequenceCompareSWValue(int **two_dim_array,
                              int row_position,
                              int column_position,
                              Arguments *arguments,
                              char *base_sequence,
                              char *sequence_to_compare) {
    int row_index       = row_position - kOneIntValue;
    int column_index    = column_position - kOneIntValue;

    // The values that I need to find the max value.
    int gap_value = arguments->gap;
    int left_up_value   = two_dim_array[row_index][column_index];
    int left_value      = two_dim_array[row_position][column_index] + gap_value;
    int up_value        = two_dim_array[row_index][column_position] + gap_value;
    int max_value       = kZeroIntValue;

    if (base_sequence[column_index] == sequence_to_compare[row_index]) {
        left_up_value += arguments->equals;
    } else {
        left_up_value += arguments->different;
    }

    int values_array[4] = {max_value, left_up_value, left_value, up_value};
    for (int i = 0; i < 4; i++) {
        int value = values_array[i];

        if (value > max_value) {
            max_value = value;
        }
    }

    return max_value;
}

void dnaSequenceCompare(DNASequence *base_sequence,
                        char *base_char,
                        DNASequence *sequence_to_compare,
                        char *char_to_compare,
                        Arguments *arguments_values) {
    // + kOneIntValue because the first columns and row of zero.
    unsigned long columns_number    = strlen(base_char) + kOneIntValue;
    unsigned long rows_number       = strlen(char_to_compare) + kOneIntValue;

    // rows_number mallocs, one for each row, plus one malloc for array of row
    // arrays.
    // An array of int arrays (a pointer to pointers to ints).
    int **two_dimension_array;

```

Padron 89397 **DNA_sequence.c** Page 2/2

```c
    // Allocate an array of rows_number pointers to ints.
    two_dimension_array = (int **)malloc(sizeof(int *) * rows_number);

    // For each row, malloc space for its buckets and add it to
    // the array of arrays. (Number of buckets = columns_number).
    for (int i = kZeroIntValue; i < rows_number; i++) {
        two_dimension_array[i] = (int *)malloc(sizeof(int) * columns_number);
    }

    // I can access using [] notation:
    // two_dimension_array[i] is a bucket in 2d_array, which is the address of
    // a 1d array, on which you can use indexing to access its bucket int value.
    // Initialize buckets with zero value.
    for (int i = kZeroIntValue; i < rows_number; i++) {
        for (int j = kZeroIntValue; j < columns_number; j++) {
            two_dimension_array[i][j] = 0;
        }
    }

    for (int i = kOneIntValue; i < rows_number; i++) {
        for (int j = kOneIntValue; j < columns_number; j++) {
            int compare_value = dnaSequenceCompareSWValue(two_dimension_array,
                                                          i,
                                                          j,
                                                          arguments_values,
                                                          base_char,
                                                          char_to_compare);
            two_dimension_array[i][j] = compare_value;
        }
    }

    long row_index       = rows_number - kOneIntValue;
    long column_index    = columns_number - kOneIntValue;
    int compare_value = two_dimension_array[row_index][column_index];
    sequence_to_compare->comparison_value = compare_value;

    // Free the allocated memory.
    // First free each row and then the arraw of rows.
    for (int i = 0; i < rows_number; i++) {
        free(two_dimension_array[i]);
    }
    free(two_dimension_array);
}
```

Padron 89397 **DNA_sequence.h** Page 1/1

```
1  //
2  //  DNA_sequence.h
3  //  TP1
4  //
5  //  Created by GastÃ³n Montes on 09/09/14.
6  //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7  //
8
9  #ifndef TP1_DNA_sequence_h
10 #define TP1_DNA_sequence_h
11
12 #include "arguments.h"
13
14 typedef struct {
15     int comparison_value;
16     long position_in_file;
17 } DNASequence;
18
19 /**
20  * @params − sequence_line: the line read from the file that contains the sequence of DNA.
21  * @params − position: The position in the file. For example, base sequence is going to hava position = 0;
22  */
23 void dnaSequenceCreate(DNASequence *sequence, long int position);
24
25 /**
26  * @params − base_sequence: The base sequence to compare.
27  * @params − base_sequence_char: The sequence base char.
28  * @params − sequence_to_compare: The sequence that I want to compare.
29  * @params − sequence_to_compare_char: The sequence to compare char.
30  * @params − arguments_values: the argument of Smith−Waterman algorithm.
31  */
32 void dnaSequenceCompare(DNASequence *base_sequence,
33                         char *base_sequence_char,
34                         DNASequence *sequence_to_compare,
35                         char *sequence_to_compare_char,
36                         Arguments *arguments_values);
37
38 #endif
```

Padron 89397 **files.c** Page 1/2

```
1  //
2  //  files.c
3  //  TP1
4  //
5  //  Created by GastÃ³n Montes on 06/09/14.
6  //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7  //
8
9  #include "files.h"
10 #include "string_functions.h"
11
12 #include <string.h>
13
14 static char *const kFilesOpenModeRead           = "rb";
15 static char *const kFilesOpenModeWrite          = "w";
16 static int const kReadOffset                    = 2;
17 static char *const kEndOfLine                   = "\n";
18
19 void filesCreateInputFile(Files *files, char *input_file_name) {
20     if (strlen(input_file_name) > 0) {
21         files->input_file   = fopen(input_file_name, kFilesOpenModeRead);
22
23         // Input file can not be NULL, output file can.
24         if (files->input_file == NULL) {
25             files->operation_code = FilesOperationCodeFail;
26         } else {
27             files->operation_code = FilesOperationCodeSuccess;
28         }
29     } else {
30         files->operation_code = FilesOperationCodeFail;
31     }
32 }
33
34 void filesCreateOutputFile(Files *files, char *output_file_name) {
35     if (strlen(output_file_name) > 0) {
36         files->output_file   = fopen(output_file_name, kFilesOpenModeWrite);
37
38         // Input file can not be NULL, output file can.
39         if (files->output_file == NULL) {
40             files->operation_code = FilesOperationCodeFail;
41         } else {
42             files->operation_code = FilesOperationCodeSuccess;
43         }
44     } else {
45         files->operation_code = FilesOperationCodeFail;
46     }
47 }
48
49 void filesCreate(Files *files, int count, char *input_name, char *output_name) {
50     files->input_file   = NULL;
51     files->output_file  = NULL;
52
53     switch (count) {
54         case 2:
55             filesCreateInputFile(files, input_name);
56             break;
57         case 3:
58             filesCreateInputFile(files, input_name);
59
60             if (files->operation_code != FilesOperationCodeFail) {
61                 filesCreateOutputFile(files, output_name);
62             }
63             break;
64         default:
65             break;
66     }
67 }
68
69 void filesDestroy(Files *files) {
70     if (files->input_file != NULL) {
71         fclose(files->input_file);
72     }
73
```

Padron 89397 **files.c** Page 2/2

```
74      if (files->output_file != NULL) {
75          fclose(files->output_file);
76      }
77  }
78
79  FilesReadOperationCode filesReadLine(Files *files,
80                                       char *buffer,
81                                       int number_of_elements) {
82      // + 2 because fgets read n - 1 characters,
83      // or up to \n line stroke or EOF.
84      // I must read the \n character too from the line.
85      char *read;
86      read = fgets(buffer, number_of_elements+kReadOffset, files->input_file);
87
88      char *save;
89      buffer = stringStrtok_r(buffer, kEndOfLine, &save);
90
91      if (feof(files->input_file)) {
92          files->operation_code = FilesOperationCodeSuccess;
93          return FilesReadOperationCodeEndOfFile;
94      }
95
96      if (strlen(read) == 0) {
97          files->operation_code = FilesOperationCodeFail;
98          return FilesReadOperationCodeFail;
99      } else {
100         files->operation_code = FilesOperationCodeSuccess;
101         return FilesReadOperationCodeSuccess;
102     }
103 }
104
105 FilesReadOperationCode filesReadLineInPosition(Files *files,
106                                                char *sequence_buffer,
107                                                int number_of_elements,
108                                                long sequence_file_position) {
109     fseek(files->input_file, sequence_file_position, SEEK_SET);
110
111     return filesReadLine(files, sequence_buffer, number_of_elements);
112 }
113
114 void filesPrint(Files *files, char *buffer) {
115     if (files->output_file != NULL) {
116         fputs(buffer, files->output_file);
117         fputs("\n", files->output_file);
118     } else {
119         // Write in the standar output.
120         printf("%s\n", buffer);
121     }
122 }
```

Padron 89397 **files.h** Page 1/2

```
1   //
2   //  files.h
3   //  TP1
4   //
5   //  Created by GastÃ³n Montes on 06/09/14.
6   //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7   //
8
9   #include <stdio.h>
10
11  #ifndef TP1_files_h
12  #define TP1_files_h
13
14  typedef enum {
15      FilesOperationCodeSuccess,
16      FilesOperationCodeFail,
17  } FilesOperationCode;
18
19  typedef enum {
20      FilesReadOperationCodeFail,
21      FilesReadOperationCodeSuccess,
22      FilesReadOperationCodeEndOfFile
23  } FilesReadOperationCode;
24
25  typedef struct {
26      FILE *input_file;
27      FILE *output_file;
28      FilesOperationCode operation_code;
29  } Files;
30
31  /**
32   * Create the files.
33   * @params - files_count: The number of arguments.
34   * @params - input_file_name: the name of the imput file.
35   * @params - output_file_name: the name of the output file, it could be NULL if
    there is no output file.
36   */
37  void filesCreate(Files *files, int count, char *input_name, char *output_name);
38
39  /**
40   * Close both files.
41   */
42  void filesDestroy(Files *files);
43
44  /**
45   * Fill the line read from the file to buffer.
46   * @params - number_of_elements: Maxium number of chars to read.
47   * @return: FilesReadOperationCodeSuccess bytes were read, FilesReadOperationCod
    eFail could not read or FilesReadOperationCodeEndOfFile end of file.
48   */
49  FilesReadOperationCode filesReadLine(Files *files,
50                                       char *buffer,
51                                       int number_of_elements);
52
53  /**
54   * Seek into the file to sequence_file_position and fill the line read to sequen
    ce_buffer.
55   * @params - number_of_elements: Maxium number of chars to read.
56   * @params - sequence_file_position: The relative position to the file init to s
    eek.
57   * @return: FilesReadOperationCodeSuccess bytes were read, FilesReadOperationCod
    eFail could not read or FilesReadOperationCodeEndOfFile end of file.
58   */
59  FilesReadOperationCode filesReadLineInPosition(Files *files,
60                                                 char *sequence_buffer,
61                                                 int number_of_elements,
62                                                 long sequence_file_position);
63
64  /**
65   * Print buffer into output file or standar output.
66   */
67  void filesPrint(Files *files, char *buffer);
68
```

Padron 89397 **files.h** Page 2/2

```
69 #endif
```

Padron 89397 **list.c** Page 1/2

```
1  //
2  //  list.c
3  //  TP1
4  //
5  //  Created by GastÃ³n Montes on 14/09/14.
6  //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7  //
8
9  #include "list.h"
10 #include "files.h"
11
12 #include <stdio.h>
13 #include <stdlib.h>
14
15 void linkedListCreate(LinkedList **linked_list) {
16     LinkedList *new_list    = (LinkedList *)malloc(sizeof(LinkedList));
17     new_list->head_node     = NULL;
18     *linked_list = new_list;
19 }
20
21 void linkedListInsertNode(LinkedList *list,
22                           struct ListNode *previous_node,
23                           struct ListNode *current_node,
24                           struct ListNode *new_node) {
25     // Insert the new node at the beginning or as next of the list head node.
26     if (previous_node == NULL) {
27         int current_node_comparison = current_node->sequence.comparison_value;
28         int new_node_comparison_value = new_node->sequence.comparison_value;
29         if (current_node_comparison == new_node_comparison_value) {
30             long current_node_pos = current_node->sequence.position_in_file;
31             long new_node_pos = new_node->sequence.position_in_file;
32             if (current_node_pos > new_node_pos) {
33                 new_node->next  = current_node;
34                 list->head_node = new_node;
35             } else {
36                 new_node->next      = current_node->next;
37                 current_node->next  = new_node;
38             }
39         } else {
40             // Current node comparison value < New node comparison value.
41             new_node->next  = current_node;
42             list->head_node = new_node;
43         }
44     } else if (current_node == NULL) {
45         // Insert at the end.
46         previous_node->next = new_node;
47     } else {
48         int current_node_comparison = current_node->sequence.comparison_value;
49         int new_node_comparison_value = new_node->sequence.comparison_value;
50         if (current_node_comparison == new_node_comparison_value) {
51             long current_node_pos = current_node->sequence.position_in_file;
52             long new_node_pos = new_node->sequence.position_in_file;
53             if (current_node_pos > new_node_pos) {
54                 new_node->next      = current_node;
55                 previous_node->next = new_node;
56             } else {
57                 new_node->next      = current_node->next;
58                 current_node->next  = new_node;
59             }
60         } else {
61             // Current node comparison value < New node comparison value.
62             new_node->next      = current_node;
63             previous_node->next = new_node;
64         }
65     }
66 }
67
68 void linkedListAddNode(LinkedList *linked_list, DNASequence sequence) {
69     // Create the new node.
70     struct ListNode *new_node = listNodeCreate(sequence);
71
72     if (linked_list->head_node == NULL) {
73         linked_list->head_node = new_node;
```

Padron 89397 **list.c** Page 2/2

```
74      } else {
75          // Nodes for iteration.
76          struct ListNode *prev_node      = NULL;
77          struct ListNode *current_node   = linked_list->head_node;
78
79          // Search through to find correct position to insert the new node.
80          int new_node_value = new_node->sequence.comparison_value;
81          int current_node_value = current_node->sequence.comparison_value;
82          while (current_node != NULL && new_node_value < current_node_value) {
83              prev_node       = current_node;
84              current_node    = current_node->next;
85          }
86
87          linkedListInsertNode(linked_list, prev_node, current_node, new_node);
88      }
89  }
90
91  void linkedListDestroy(LinkedList *linked_list) {
92      struct ListNode *current_node = linked_list->head_node;
93      struct ListNode *next_node;
94
95      while (current_node != NULL) {
96          next_node = current_node->next;
97
98          listNodeDestroy(current_node);
99
100         current_node = next_node;
101     }
102
103     free(linked_list);
104 }
```

Padron 89397 **list.h** Page 1/1

```
1   //
2   //  list.h
3   //  TP1
4   //
5   //  Created by GastÃ³n Montes on 14/09/14.
6   //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7   //
8
9   #ifndef TP1_list_h
10  #define TP1_list_h
11
12  #include "list_node.h"
13
14  typedef struct {
15      struct ListNode *head_node;
16  } LinkedList;
17
18  /**
19   * Create a new linkedList with the first node that contains sequence.
20   * @params - sequence: The sequence for the head node.
21   */
22  void linkedListCreate(LinkedList **linked_list);
23
24  /**
25   * Add node to the linked_list in order.
26   * @params - sequence: The sequence to the new node.
27   */
28  void linkedListAddNode(LinkedList *linked_list, DNASequence sequence);
29
30  /**
31   * Free all the resources allocated by the list.
32   */
33  void linkedListDestroy(LinkedList *linked_list);
34
35  #endif
```

Padron 89397 **list_node.c** Page 1/1

```c
//
//  list_node.c
//  TP1
//
//  Created by GastÃ³n Montes on 13/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>

#include "list_node.h"

struct ListNode *listNodeCreate(DNASequence sequence) {
    struct ListNode *new_node;
    new_node = (struct ListNode *)malloc(sizeof(struct ListNode));
    new_node->sequence  = sequence;
    new_node->next      = NULL;

    return new_node;
}

void listNodeDestroy(struct ListNode *node) {
    free(node);
}
```

Padron 89397 **list_node.h** Page 1/1

```c
//
//  list_node.h
//  TP1
//
//  Created by GastÃ³n Montes on 13/09/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

#ifndef TP1_list_node_h
#define TP1_list_node_h

#include "DNA_sequence.h"

struct ListNode {
    DNASequence sequence;
    struct ListNode *next;
};

/**
 * Create a new node with the sequence asociated.
 * @params - sequence: The sequence asociated to the new node.
 */
struct ListNode *listNodeCreate(DNASequence sequence);

/**
 * Free the resources allocated by the creation of a new node.
 */
void listNodeDestroy(struct ListNode *node);

#endif
```

Padron 89397 **main.c** Page 1/3

```c
//
//  main.c
//  TP1
//
//  Created by GastÃ³n Montes on 26/08/14.
//  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
//

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

#include "arguments_parser.h"
#include "DNA_sequence.h"
#include "list_node.h"
#include "arguments.h"
#include "files.h"
#include "list.h"

static int const kNormalOutput                  = 0;
static int const kInvalidArguments              = 1;
static int const kFilesError                    = 2;
static int const kMaxNumberOfCharInFirstline    = 64;

FilesReadOperationCode createSequence(Files *files,
                                      DNASequence *sequence,
                                      char *buffer,
                                      Arguments *arguments) {
    long int position_in_file = ftell(files->input_file);
    FilesReadOperationCode operation_read_code;
    operation_read_code = filesReadLine(files, buffer, arguments->max_length);

    if (operation_read_code != FilesReadOperationCodeFail) {
        // Create the  sequence.
        dnaSequenceCreate(sequence, position_in_file);
    }

    return operation_read_code;
}

void printResults(LinkedList *list, Files *files, Arguments *arguments) {
    struct ListNode *temporal_node = list->head_node;

    while (temporal_node != NULL) {
        int kSequenceMaxLength = arguments->max_length;
        char char_to_print[kSequenceMaxLength];
        filesReadLineInPosition(files, char_to_print,
                                kSequenceMaxLength,
                                temporal_node->sequence.position_in_file);
        filesPrint(files, char_to_print);

        temporal_node = temporal_node->next;
    }
}

FilesReadOperationCode compareSequences(Files *files,
                                        LinkedList *linked_list,
                                        Arguments *arguments,
                                        DNASequence *base_sequence,
                                        char *base_sequence_string) {
    while (!feof(files->input_file)) {
        DNASequence sequence_to_compare;
        int kSequenceMaxLength = arguments->max_length;
        char sequence_to_compare_string[kSequenceMaxLength];
        FilesReadOperationCode operation_read_code;
        operation_read_code = createSequence(files,
                                             &sequence_to_compare,
                                             sequence_to_compare_string,
                                             arguments);

        if (operation_read_code != FilesReadOperationCodeSuccess) {
            return operation_read_code;
        }
```

Padron 89397 **main.c** Page 2/3

```c

        dnaSequenceCompare(base_sequence,
                           base_sequence_string,
                           &sequence_to_compare,
                           sequence_to_compare_string,
                           arguments);
        linkedListAddNode(linked_list, sequence_to_compare);
    }

    return FilesReadOperationCodeSuccess;
}

int main(int argc, const char *argv[]) {
    ArgumentsParser arguments_parser;
    DNASequence base_sequence;
    LinkedList *linked_list;
    Arguments arguments;
    Files files;

    // Parse the console arguments.
    argumentsParserCreate(&arguments_parser, argc, argv);
    ArgumentsParserCode parse_validation;
    parse_validation = arguments_parser.parse_validation;

    if (parse_validation != ArgumentsParserCodeOneArgument
        && parse_validation != ArgumentsParserCodeTwoArguments) {
        return kInvalidArguments;
    }

    // Create the input and output files
    filesCreate(&files, arguments_parser.arguments_count,
                arguments_parser.first_argument,
                arguments_parser.second_argument);

    if (files.operation_code == FilesOperationCodeFail) {
        // Free the files resources
        filesDestroy(&files);
        return kFilesError;
    }

    // Read the first line of the file with the arguments.
    char firstLine[kMaxNumberOfCharInFirstline];
    FilesReadOperationCode operation_read_code;
    operation_read_code = filesReadLine(&files,
                                        firstLine,
                                        kMaxNumberOfCharInFirstline);

    if (operation_read_code != FilesReadOperationCodeSuccess) {
        // Can not read the first line with the parameters.
        filesDestroy(&files);
        return kFilesError;
    }

    // Create the arguments model.
    argumentsCreate(&arguments, firstLine);
    int kSequenceMaxLenght = arguments.max_length;
    char base_sequence_string[kSequenceMaxLenght];

    // Create the base secuence.
    operation_read_code = createSequence(&files,
                                         &base_sequence,
                                         base_sequence_string,
                                         &arguments);

    // Read the base sequence string.
    filesReadLineInPosition(&files,
                            base_sequence_string,
                            kSequenceMaxLenght,
                            base_sequence.position_in_file);

    // Return file error if can not read the sequence.
    if (operation_read_code == FilesReadOperationCodeFail) {
        // First sequence can not be read.
```

Padron 89397 **main.c** Page 3/3

```
147         filesDestroy(&files);
148         return kFilesError;
149     }
150 
151     // Create the list of sequences.
152     linkedListCreate(&linked_list);
153 
154     // Compare secuences to base secuence.
155     operation_read_code = compareSequences(&files,
156                                            linked_list,
157                                            &arguments,
158                                            &base_sequence,
159                                            base_sequence_string);
160 
161     // Return file error if can not read the sequence.
162     if (operation_read_code == FilesReadOperationCodeFail) {
163         // First sequence can not be read.
164         filesDestroy(&files);
165         return kFilesError;
166     }
167 
168     printResults(linked_list, &files, &arguments);
169 
170     linkedListDestroy(linked_list);
171     filesDestroy(&files);
172 
173     return kNormalOutput;
174 }
```

Padron 89397 **string_functions.c** Page 1/1

```
1  //
2  //  string_functions.c
3  //  TP1
4  //
5  //  Created by GastÃ³n Montes on 23/09/14.
6  //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7  //
8 
9  #include "string_functions.h"
10 
11 #include <stdio.h>
12 #include <string.h>
13 
14 char *stringStrtok_r(char *str, const char *delim, char **nextp) {
15     char *ret;
16 
17     if (str == NULL) {
18         str = *nextp;
19     }
20 
21     str += strspn(str, delim);
22 
23     if (*str == '\0') {
24         return NULL;
25     }
26 
27     ret = str;
28 
29     str += strcspn(str, delim);
30 
31     if (*str) {
32         *str++ = '\0';
33     }
34 
35     *nextp = str;
36 
37     return ret;
38 }
```

Padron 89397 **string_functions.h** Page 1/1

```
1  //
2  //  string_functions.h
3  //  TP1
4  //
5  //  Created by GastÃ³n Montes on 23/09/14.
6  //  Copyright (c) 2014 GastÃ³n Montes. All rights reserved.
7  //
8
9  #ifndef TP1_string_functions_h
10 #define TP1_string_functions_h
11
12 /**
13  * Safe reimplementation of function strtok_r of <string.h>
14  */
15 char *stringStrtok_r(char *str, const char *delim, char **nextp);
16
17 #endif
```

Padron 89397 **Table of Content** Page 1/1