

Padron 89397

arguments.c

Page 1/1

```

1 //
2 // arguments.c
3 // TP1
4 //
5 // Created by Gast n Montes on 08/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #include <stdio.h>
10 #include <stdlib.h>
11
12 #include "arguments.h"
13 #include "string_functions.h"
14
15 static char *const kCommaSeparator = ",";
16
17 void argumentsCreate(Arguments *arguments, char *arguments_line) {
18     char *token;
19
20     // Read equals param value.
21     char *save_str;
22     token = stringStrtok_r(arguments_line, kCommaSeparator, &save_str);
23     arguments->equals = atoi(token);
24
25     // Read differents param value.
26     token = stringStrtok_r(NULL, kCommaSeparator, &save_str);
27     arguments->different = atoi(token);
28
29     // Read gap param value.
30     token = stringStrtok_r(NULL, kCommaSeparator, &save_str);
31     arguments->gap = atoi(token);
32
33     // Read max length param value.
34     token = stringStrtok_r(NULL, kCommaSeparator, &save_str);
35     arguments->max_length = atoi(token);
36 }

```

Padron 89397

arguments.h

Page 1/1

```

1 //
2 // arguments.h
3 // TP1
4 //
5 // Created by Gast n Montes on 08/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #ifndef TP1_arguments_h
10 #define TP1_arguments_h
11
12 typedef struct {
13     int equals;
14     int different;
15     int gap;
16     int max_length;
17 } Arguments;
18
19 /**
20  * @params - arguments_line: the line that contains the arguments in CSV values.
21  */
22 void argumentsCreate(Arguments *arguments, char *arguments_line);
23
24 #endif

```

Padron 89397

arguments_parser.c

Page 1/1

```

1 //
2 // arguments_parser.c
3 // TP1
4 //
5 // Created by Gast n Montes on 02/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #include <stdio.h>
10 #include "arguments_parser.h"
11
12 static int const FIRST_ARGUMENT_INDEX = 1;
13 static int const SECOND_ARGUMENT_INDEX = 2;
14
15 // pragma mark - Private methods.
16 void argumentsParserParseArguments(ArgumentsParser *parser,
17                                   int arguments_count) {
18     switch (arguments_count) {
19         case 0:
20             parser->parse_validation = ArgumentsParserCodeNoArguments;
21             break;
22         case 1:
23             // No arguments passed.
24             parser->parse_validation = ArgumentsParserCodeNoArguments;
25             break;
26         case 2:
27             // Only 1 argument.
28             parser->parse_validation = ArgumentsParserCodeOneArgument;
29             break;
30         case 3:
31             // 2 arguments ok.
32             parser->parse_validation = ArgumentsParserCodeTwoArguments;
33             break;
34         default:
35             // More than 2 arguments.
36             parser->parse_validation = ArgumentsParserCodeTooManyArguments;
37             break;
38     }
39     parser->arguments_count = arguments_count;
40 }
41
42 void argumentsParserFirstArgument(ArgumentsParser *parser,
43                                  const char *arguments[]) {
44     parser->first_argument = NULL;
45     if (arguments[FIRST_ARGUMENT_INDEX] != NULL) {
46         parser->first_argument = (char *)arguments[FIRST_ARGUMENT_INDEX];
47     }
48 }
49
50 void argumentsParserSecondArgument(ArgumentsParser *parser,
51                                   const char *arguments[]) {
52     parser->second_argument = NULL;
53     if (arguments[SECOND_ARGUMENT_INDEX] != NULL) {
54         parser->second_argument = (char *)arguments[SECOND_ARGUMENT_INDEX];
55     }
56 }
57
58 // pragma mark - Public methods.
59 void argumentsParserCreate(ArgumentsParser *arguments_parser,
60                           int arguments_count,
61                           const char *arguments[]) {
62     argumentsParserParseArguments(arguments_parser, arguments_count);
63     argumentsParserFirstArgument(arguments_parser, arguments);
64     argumentsParserSecondArgument(arguments_parser, arguments);
65 }

```

Padron 89397

arguments_parser.h

Page 1/1

```

1 //
2 // arguments_parser.h
3 // TP1
4 //
5 // Created by Gast n Montes on 02/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #ifndef TP1_arguments_parser_h
10 #define TP1_arguments_parser_h
11
12 typedef enum {
13     ArgumentsParserCodeNoArguments,
14     ArgumentsParserCodeOneArgument,
15     ArgumentsParserCodeTwoArguments,
16     ArgumentsParserCodeTooManyArguments
17 } ArgumentsParserCode;
18
19 typedef struct {
20     ArgumentsParserCode parse_validation;
21     char *first_argument;
22     char *second_argument;
23     int arguments_count;
24 } ArgumentsParser;
25
26 void argumentsParserCreate(ArgumentsParser *arguments_parser,
27                           int arguments_count,
28                           const char *arguments[]);
29
30 #endif

```

Padron 89397

DNA_sequence.c

Page 1/2

```

1 //
2 // DNA_sequence.c
3 // TP1
4 //
5 // Created by Gast n Montes on 09/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12
13 #include "DNA_sequence.h"
14
15 static int const kZeroIntValue = 0;
16 static int const kOneIntValue = 1;
17
18 void dnaSequenceCreate(DNASequence *sequence, long int position) {
19     sequence->comparison_value = kZeroIntValue;
20     sequence->position_in_file = position;
21 }
22
23 /**
24  * Smith-Waterman comparition.
25  */
26 int dnaSequenceCompareSWValue(int **two_dim_array,
27                               int row_position,
28                               int column_position,
29                               Arguments *arguments,
30                               char *base_sequence,
31                               char *sequence_to_compare) {
32     int row_index = row_position - kOneIntValue;
33     int column_index = column_position - kOneIntValue;
34
35     // The values that I need to find the max value.
36     int gap_value = arguments->gap;
37     int left_up_value = two_dim_array[row_index][column_index];
38     int left_value = two_dim_array[row_position][column_index] + gap_value;
39     int up_value = two_dim_array[row_index][column_position] + gap_value;
40     int max_value = kZeroIntValue;
41
42     if (base_sequence[column_index] == sequence_to_compare[row_index]) {
43         left_up_value += arguments->equals;
44     } else {
45         left_up_value += arguments->different;
46     }
47
48     int values_array[4] = {max_value, left_up_value, left_value, up_value};
49     for (int i = 0; i < 4; i++) {
50         int value = values_array[i];
51
52         if (value > max_value) {
53             max_value = value;
54         }
55     }
56
57     return max_value;
58 }
59
60 void dnaSequenceCompare(DNASequence *base_sequence,
61                         char *base_char,
62                         DNASequence *sequence_to_compare,
63                         char *char_to_compare,
64                         Arguments *arguments_values) {
65     // + kOneIntValue because the first columns and row of zero.
66     unsigned long columns_number = strlen(base_char) + kOneIntValue;
67     unsigned long rows_number = strlen(char_to_compare) + kOneIntValue;
68
69     // rows_number mallocs, one for each row, plus one malloc for array of row
70     // arrays.
71     // An array of int arrays (a pointer to pointers to ints).
72     int **two_dimension_array;
73

```

Padron 89397

DNA_sequence.c

Page 2/2

```

74 // Allocate an array of rows_number pointers to ints.
75 two_dimension_array = (int **)malloc(sizeof(int *) * rows_number);
76
77 // For each row, malloc space for its buckets and add it to
78 // the array of arrays. (Number of buckets = columns_number).
79 for (int i = kZeroIntValue; i < rows_number; i++) {
80     two_dimension_array[i] = (int *)malloc(sizeof(int) * columns_number);
81 }
82
83 // I can access using [] notation:
84 // two_dimension_array[i] is a bucket in 2d_array, which is the address of
85 // a 1d array, on which you can use indexing to access its bucket int value.
86 // Initialize buckets with zero value.
87 for (int i = kZeroIntValue; i < rows_number; i++) {
88     for (int j = kZeroIntValue; j < columns_number; j++) {
89         two_dimension_array[i][j] = 0;
90     }
91 }
92
93 for (int i = kOneIntValue; i < rows_number; i++) {
94     for (int j = kOneIntValue; j < columns_number; j++) {
95         int compare_value = dnaSequenceCompareSWValue(two_dimension_array,
96                                                         i,
97                                                         j,
98                                                         arguments_values,
99                                                         base_char,
100                                                         char_to_compare);
101         two_dimension_array[i][j] = compare_value;
102     }
103 }
104
105 long row_index = rows_number - kOneIntValue;
106 long column_index = columns_number - kOneIntValue;
107 int compare_value = two_dimension_array[row_index][column_index];
108 sequence_to_compare->comparison_value = compare_value;
109
110 // Free the allocated memory.
111 // First free each row and then the array of rows.
112 for (int i = 0; i < rows_number; i++) {
113     free(two_dimension_array[i]);
114 }
115 free(two_dimension_array);
116 }

```

Padron 89397

DNA_sequence.h

Page 1/1

```

1 //
2 // DNA_sequence.h
3 // TP1
4 //
5 // Created by Gast n Montes on 09/09/14.
6 // Copyright (c) 2014 Gast n 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 } DNASquence;
18
19 /**
20  * @params - sequence_line: the line read from the file that contains the sequen
21  * ce of DNA.
22  * @params - position: The position in the file. For example, base sequence is g
23  * oing to hava position = 0;
24  */
25 void dnaSequenceCreate(DNASquence *sequence, long int position);
26
27 /**
28  * @params - base_sequence: The base sequence to compare.
29  * @params - base_sequence_char: The sequence base char.
30  * @params - sequence_to_compare: The sequence that I want to compare.
31  * @params - sequence_to_compare_char: The sequence to compare char.
32  * @params - arguments_values: the argument of Smith-Waterman algorithm.
33  */
34 void dnaSequenceCompare(DNASquence *base_sequence,
35                         char *base_sequence_char,
36                         DNASquence *sequence_to_compare,
37                         char *sequence_to_compare_char,
38                         Arguments *arguments_values);
39 #endif

```

Padron 89397

files.c

Page 1/2

```

1 //
2 // files.c
3 // TP1
4 //
5 // Created by Gast n Montes on 06/09/14.
6 // Copyright (c) 2014 Gast n 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 Gast n Montes on 06/09/14.
6  //  Copyright (c) 2014 Gast n 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 input file.
35  * @params - output_file_name: the name of the output file, it could be NULL if
36  * there is no output file.
37  */
38 void filesCreate(Files *files, int count, char *input_name, char *output_name);
39
40 /**
41  * Close both files.
42  */
43 void filesDestroy(Files *files);
44
45 /**
46  * Fill the line read from the file to buffer.
47  * @params - number_of_elements: Maxium number of chars to read.
48  * @return: FilesReadOperationCodeSuccess bytes were read, FilesReadOperationCod
49  eFail could not read or FilesReadOperationCodeEndOfFile end of file.
50  */
51 FilesReadOperationCode filesReadLine(Files *files,
52                                     char *buffer,
53                                     int number_of_elements);
54
55 /**
56  * Seek into the file to sequence_file_position and fill the line read to sequen
57  ce_buffer.
58  * @params - number_of_elements: Maxium number of chars to read.
59  * @params - sequence_file_position: The relative position to the file init to s
60  eek.
61  * @return: FilesReadOperationCodeSuccess bytes were read, FilesReadOperationCod
62  eFail could not read or FilesReadOperationCodeEndOfFile end of file.
63  */
64 FilesReadOperationCode filesReadLineInPosition(Files *files,
65                                                char *sequence_buffer,
66                                                int number_of_elements,
67                                                long sequence_file_position);
68
69 /**
70  * Print buffer into output file or standar output.
71  */
72 void filesPrint(Files *files, char *buffer);

```

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 Gast n Montes on 14/09/14.
6 // Copyright (c) 2014 Gast n 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, DNASSequence 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 Gast n Montes on 14/09/14.
6  // Copyright (c) 2014 Gast n 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

```

1 //
2 // list_node.c
3 // TP1
4 //
5 // Created by Gast n Montes on 13/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #include <stdio.h>
10 #include <stdlib.h>
11
12 #include "list_node.h"
13
14 struct ListNode *listNodeCreate(DNASequence sequence) {
15     struct ListNode *new_node;
16     new_node = (struct ListNode *)malloc(sizeof(struct ListNode));
17     new_node->sequence = sequence;
18     new_node->next = NULL;
19
20     return new_node;
21 }
22
23 void listNodeDestroy(struct ListNode *node) {
24     free(node);
25 }

```

Padron 89397

list_node.h

Page 1/1

```

1 //
2 // list_node.h
3 // TP1
4 //
5 // Created by Gast n Montes on 13/09/14.
6 // Copyright (c) 2014 Gast n Montes. All rights reserved.
7 //
8
9 #ifndef TP1_list_node_h
10 #define TP1_list_node_h
11
12 #include "DNA_sequence.h"
13
14 struct ListNode {
15     DNASequence sequence;
16     struct ListNode *next;
17 };
18
19 /**
20  * Create a new node with the sequence asociated.
21  * @params - sequence: The sequence asociated to the new node.
22  */
23 struct ListNode *listNodeCreate(DNASequence sequence);
24
25 /**
26  * Free the resources allocated by the creation of a new node.
27  */
28 void listNodeDestroy(struct ListNode *node);
29
30 #endif

```


Padron 89397

main.c

Page 1/3

```

1  //
2  //  main.c
3  //  TP1
4  //
5  //  Created by Gast n Montes on 26/08/14.
6  //  Copyright (c) 2014 Gast n Montes. All rights reserved.
7  //
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12
13 #include "arguments_parser.h"
14 #include "DNA_sequence.h"
15 #include "list_node.h"
16 #include "arguments.h"
17 #include "files.h"
18 #include "list.h"
19
20 static int const kNormalOutput          = 0;
21 static int const kInvalidArguments      = 1;
22 static int const kFilesError            = 2;
23 static int const kMaxNumberOfCharInFirstline = 64;
24
25 FilesReadOperationCode createSequence(Files *files,
26                                     DNASequence *sequence,
27                                     char *buffer,
28                                     Arguments *arguments) {
29     long int position_in_file = ftell(files->input_file);
30     FilesReadOperationCode operation_read_code;
31     operation_read_code = filesReadLine(files, buffer, arguments->max_length);
32
33     if (operation_read_code != FilesReadOperationCodeFail) {
34         // Create the sequence.
35         dnaSequenceCreate(sequence, position_in_file);
36     }
37
38     return operation_read_code;
39 }
40
41 void printResults(LinkedList *list, Files *files, Arguments *arguments) {
42     struct ListNode *temporal_node = list->head_node;
43
44     while (temporal_node != NULL) {
45         int kSequenceMaxLength = arguments->max_length;
46         char char_to_print[kSequenceMaxLength];
47         filesReadLineInPosition(files, char_to_print,
48                                kSequenceMaxLength,
49                                temporal_node->sequence.position_in_file);
50         filesPrint(files, char_to_print);
51
52         temporal_node = temporal_node->next;
53     }
54 }
55
56 FilesReadOperationCode compareSequences(Files *files,
57                                         LinkedList *linked_list,
58                                         Arguments *arguments,
59                                         DNASequence *base_sequence,
60                                         char *base_sequence_string) {
61     while (!feof(files->input_file)) {
62         DNASequence sequence_to_compare;
63         int kSequenceMaxLength = arguments->max_length;
64         char sequence_to_compare_string[kSequenceMaxLength];
65         FilesReadOperationCode operation_read_code;
66         operation_read_code = createSequence(files,
67                                             &sequence_to_compare,
68                                             sequence_to_compare_string,
69                                             arguments);
70
71         if (operation_read_code != FilesReadOperationCodeSuccess) {
72             return operation_read_code;
73         }

```

Padron 89397

main.c

Page 2/3

```

74
75     dnaSequenceCompare(base_sequence,
76                        base_sequence_string,
77                        &sequence_to_compare,
78                        sequence_to_compare_string,
79                        arguments);
80     linkedListAddNode(linked_list, sequence_to_compare);
81 }
82
83 return FilesReadOperationCodeSuccess;
84 }
85
86 int main(int argc, const char *argv[]) {
87     ArgumentsParser arguments_parser;
88     DNASequence base_sequence;
89     LinkedList *linked_list;
90     Arguments arguments;
91     Files files;
92
93     // Parse the console arguments.
94     argumentsParserCreate(&arguments_parser, argc, argv);
95     ArgumentsParserCode parse_validation;
96     parse_validation = arguments_parser.parse_validation;
97
98     if (parse_validation != ArgumentsParserCodeOneArgument
99         && parse_validation != ArgumentsParserCodeTwoArguments) {
100         return kInvalidArguments;
101     }
102
103     // Create the input and output files
104     filesCreate(&files, arguments_parser.arguments_count,
105               arguments_parser.first_argument,
106               arguments_parser.second_argument);
107
108     if (files.operation_code == FilesOperationCodeFail) {
109         // Free the files resources
110         filesDestroy(&files);
111         return kFilesError;
112     }
113
114     // Read the first line of the file with the arguments.
115     char firstLine[kMaxNumberOfCharInFirstline];
116     FilesReadOperationCode operation_read_code;
117     operation_read_code = filesReadLine(&files,
118                                       firstLine,
119                                       kMaxNumberOfCharInFirstline);
120
121     if (operation_read_code != FilesReadOperationCodeSuccess) {
122         // Can not read the first line with the parameters.
123         filesDestroy(&files);
124         return kFilesError;
125     }
126
127     // Create the arguments model.
128     argumentsCreate(&arguments, firstLine);
129     int kSequenceMaxLenght = arguments.max_length;
130     char base_sequence_string[kSequenceMaxLenght];
131
132     // Create the base sequence.
133     operation_read_code = createSequence(&files,
134                                       &base_sequence,
135                                       base_sequence_string,
136                                       &arguments);
137
138     // Read the base sequence string.
139     filesReadLineInPosition(&files,
140                           base_sequence_string,
141                           kSequenceMaxLenght,
142                           base_sequence.position_in_file);
143
144     // Return file error if can not read the sequence.
145     if (operation_read_code == FilesReadOperationCodeFail) {
146         // 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 sequences to base sequence.
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 Gast n Montes on 23/09/14.
6  //  Copyright (c) 2014 Gast n 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 Gast�n Montes on 23/09/14.	
6	// Copyright (c) 2014 Gast�n 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
1		
2	1 ./arguments.c..... Pag. 1 (37 lines)	
3	2 ./arguments.h..... Pag. 2 (25 lines)	
4	3 ./arguments_parser.c..... Pag. 3 (66 lines)	
5	4 ./arguments_parser.h..... Pag. 4 (31 lines)	
6	5 ./DNA_sequence.c..... Pag. 5 (117 lines)	
7	6 ./DNA_sequence.h..... Pag. 7 (39 lines)	
8	7 ./files.c..... Pag. 8 (123 lines)	
9	8 ./files.h..... Pag. 10 (70 lines)	
10	9 ./list.c..... Pag. 12 (105 lines)	
11	10 ./list.h..... Pag. 14 (36 lines)	
12	11 ./list_node.c..... Pag. 15 (26 lines)	
13	12 ./list_node.h..... Pag. 16 (31 lines)	
14	13 ./main.c..... Pag. 17 (175 lines)	
15	14 ./string_functions.c..... Pag. 20 (39 lines)	
16	15 ./string_functions.h..... Pag. 21 (18 lines)	