Contents

1 Basic Test Results 2

2 RailWayPlanner.c 4

1 Basic Test Results

```
Running...
Opening tar file
OK
Tar extracted O.K.
Checking files...
OK
Making sure files are not empty...
OK
Compilation check...
Compiling...
OK
Compilation seems OK! Check if you got warnings!

=====================
 Public test cases
=====================

=====================
Test #1
Running RailWayPlanner
OK
Running diff
OK
Test 1 passed.
=====================

=====================
Test #2
Running RailWayPlanner
OK
Running diff
OK
Test 2 passed.
=====================

=====================
Test #3
Running RailWayPlanner
OK
Running diff
OK
Test 3 passed.
=====================

=====================
Test #4
Running RailWayPlanner
OK
Running diff
OK
Test 4 passed.
=====================

=====================
Test #5
Running RailWayPlanner
OK
Running diff
OK
```

```
Test 5 passed.
====================

====================
Test #6
Running RailWayPlanner
OK
Running diff
OK
Test 6 passed.
====================

====================
Test #7
Running RailWayPlanner
OK
Running diff
OK
Test 7 passed.
====================

====================
Test #8
Running RailWayPlanner
OK
Running diff
OK
Test 8 passed.
====================

====================
Test #9
Running RailWayPlanner
OK
Running diff
OK
Test 9 passed.
====================

====================
Test #10
Running RailWayPlanner
OK
Running diff
OK
Test 10 passed.
====================

**********************************
*                                *
*   presubmission script passed  *
*       10/10 tests passed       *
*                                *
**********************************

========================
= Checking coding style =
========================
 ** Total Violated Rules      : 0
 ** Total Errors Occurs       : 0
 ** Total Violated Files Count: 0
```

2 RailWayPlanner.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "limits.h"
#include "ctype.h"

#define ARG_ERROR "Usage: RailwayPlanner <InputFile>"
#define FILE_ERROR "File doesn't exists."
#define EMPTY_ERROR "File is empty."
#define NO_SIGN -1
#define INFINITE_MIN_PRICE -1
#define LINE_ERROR "Invalid input in line: "
#define EXIT_FAILURE  1
#define EXIT_SUCCESS  0
#define LIMITERS ",\r\n\0"
#define MAX_LINE_SIZE 1025
#define NUM_OF_PARAMS 4
#define MSG_MIN_PRICE "The minimal price is: "
#define ARG_NUM 2
#define OUTPUT "railway_planner_output.txt"
#define INT_LEN 11

/**
 * yoav eshed 305384869
 */

/**
 * A Struct that represents A rail part by start sign, end sind , part price, prat length
 */
typedef struct RailPart
{
    char s;
    char e;
    long len;
    long price;
} RailPart;

/**
* A struct that contains the railway parameters such as desired railway length, desired
* connections per part, and the connection signs
*/
typedef struct Params
{
    long length;
    long conTypes;
    long numOfParts;
    char *signs;
    RailPart *parts;
} Params;

/**
 * A function That prints strings and numbers to the output file
 * @param txt: the text to be printed
 * @param value: a value to printed if exists
 * @return : if string printed successfully 1 if could not open output file
 */
int printfFunc(char *txt, char *value)
{
    FILE *outputFile = fopen(OUTPUT, "w");
```

```c
    if (outputFile == NULL)
    {
        return EXIT_FAILURE;
    }
    fprintf(outputFile, "%s%s", txt, value);
    fclose(outputFile);
    return EXIT_SUCCESS;
}

/**
 *  A function that prints a matrix
 * @param row :the number of rows of the matrix
 * @param col :the number of columns of the matrix
 * @return : The matrix as a two dimensional long array
 */
long **creatMatrix(long rows, long cols)
{
    long i, j;
    long **arr = (long **) malloc((rows) * sizeof(long *));
    if (arr == NULL)
    {
        return (long **) EXIT_FAILURE;
    }
    for (i = 0; i < rows; i++)
    {
        arr[i] = (long *) calloc(cols, sizeof(long));
        if (arr[i] == NULL)
        {
            return (long **) EXIT_FAILURE;
        }
    }
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            arr[i][j] = INT_MAX;
        }
    }
    for (i = 0; i < cols; i++)
    {
        arr[0][i] = 0;
    }
    return arr;
}

/**
 * A function that calculates the minimum between two numbers
 * @param a :first number
 * @param b :second number
 * @return :the smaller number between the two
 */
long minInt(long a, long b)
{
    if (a < b)
    {
        return a;
    }
    return b;
}

/**
 * A function that finds the minimum price for rail in the required length by going over a matrix.
 * if there isn't a minimum value, the function will set the min to be -1.
 * @param mat
 * @param Data
 * @param out :output file
 */
void getPrice(long **mat, Params *Data)
```

```
128 {
129
130     char minPrice[INT_LEN];
131     long min = INT_MAX;
132     for (long i = 0; i < Data->conTypes; i++)
133     {
134         if (mat[Data->length][i] < min)
135         {
136             min = mat[Data->length][i];
137         }
138     }
139     if (min == INT_MAX)
140     {
141         min = INFINITE_MIN_PRICE;
142     }
143     sprintf(minPrice, "%ld", min);
144     printfFunc(MSG_MIN_PRICE, minPrice);
145     free(Data->parts);
146     free(Data->signs);
147     for (int len = 0; len < Data->length + 1; len++)
148     {
149         free(mat[len]);
150         mat[len] = NULL;
151     }
152     free(mat);
153 }
154
155 /**
156  * A function that retrieves the start index of the part
157  * @param sign : the sigh who's index we want
158  * @param Data : A struct that holds the rail parameters ,such as total cost, length, parts
159  * @return : the wanted index if exists -1 if not (because 0 and 1 could be indexes)
160  */
161
162 int getInd(char sign, Params *Data)
163 {
164     for (int i = 0; i <= Data->conTypes; i++)
165     {
166         if (sign == Data->signs[i])
167         {
168             return i;
169         }
170     }
171     return NO_SIGN;
172 }
173
174 /**
175  * A function that assigns the min value per matrix cell
176  * @param mat : The matrix
177  * @param Data : A struct that holds the rail parameters ,such as total cost, length, parts
178  * @param i : row index
179  * @param j : column index
180  * @param k : part index in parts array
181  */
182 void setMin(long **mat, Params *Data, long i, long j, long k)
183 {
184     long remainingLength = i - Data->parts[k].len;
185     if (remainingLength >= 0)
186     {
187         long sInd = getInd(Data->parts[k].s, Data);
188         mat[i][j] = minInt(mat[i][j], (mat[remainingLength][sInd] + Data->parts[k].price));
189     }
190 }
191
192 /**
193  * A function that goes over a matrix and fills her with the minimum costs for a rail by end sign
194  * and length based on the formula we were given in the exerecise
195  * @param mat : The matrix
```

```
196  * @param Data : A  struct that holds the rail parameters ,such as total cost, length, parts
197  */
198 void setPrices(long **mat, Params *Data)
199 {
200     for (long i = 1; i < Data->length + 1; i++)
201     {
202         for (long j = 0; j < Data->conTypes; j++)
203         {
204             for (long k = 0; k <= Data->numOfParts; k++)
205             {
206                 if (Data->parts[k].e == Data->signs[j])
207                 {
208                     setMin(mat, Data, i, j, k);
209                 }
210             }
211         }
212     }
213 }
214
215 /**
216  * A function that reads a line from a file and returns a number
217  * @param inp :the input file
218  * @param res: a pointer to a long number
219  * @return 0 if it read successfully , 1 if not.
220  */
221 int getNum(FILE *inp, long *res)
222 {
223     char *pEnd = NULL;
224     char string[INT_LEN];
225     fgets(string, MAX_LINE_SIZE, inp);
226     *res = strtol(string, &pEnd, 10);
227     if (strcmp(pEnd, "\n") != 0)
228     {
229         return EXIT_FAILURE;
230     }
231     if (*res < 0)
232     {
233         return EXIT_FAILURE;
234     }
235     return EXIT_SUCCESS;
236 }
237
238 /**
239  * A function that retrieves the connectors signs
240  * @param inp : the input file
241  * @param Data : A struct that holds the rail parameters ,such as total cost, length, parts
242  * @return 0 if the function worked successfully, 1 if not
243  */
244 int getSigns(FILE *inp, Params *Data)
245 {
246     int i = 0;
247     char string[MAX_LINE_SIZE];
248     fgets(string, MAX_LINE_SIZE, inp);
249     if (string[0] == 0)
250     {
251         return EXIT_FAILURE;
252     }
253     char *ptr = strtok(string, LIMITERS);
254     Data->signs = malloc((strlen(string) + 10) * sizeof(char));
255     if (Data->signs == NULL)
256     {
257         return EXIT_FAILURE;
258     }
259     while (ptr != NULL)
260     {
261         if (strlen(ptr) != 1)
262         {
263             return EXIT_FAILURE;
```

```
264             }
265             strcpy(&Data->signs[i], ptr);
266             ptr = strtok(NULL, LIMITERS);
267             i++;
268         }
269         Data->conTypes = i;
270         return EXIT_SUCCESS;
271     }
272 
273     /**
274      * A function that checks if a string is valid
275      * @param string : current string read from file
276      * @return true if the string is in the correct format, 1 if not
277      */
278     int stringCheck(char *string)
279     {
280         if (string[1] != ',')
281         {
282             return EXIT_FAILURE;
283         }
284         if (string[3] != ',')
285         {
286             return EXIT_FAILURE;
287         }
288         for (unsigned int i = 4; i < strlen(string) - 1; i++)
289         {
290 
291             if (!isdigit(string[i]) && (string[i] != ','))
292             {
293                 return EXIT_FAILURE;
294             }
295         }
296         return EXIT_SUCCESS;
297     }
298 
299     /**
300      * A function that checks if all the parameters of a part are valid
301      * @param start: Rail Part start sign
302      * @param start: Rail Part end sign
303      * @param len : Rail part Length
304      * @param cost : rail part cost
305      * @param Data : A struct that holds the rail parameters ,such as total cost, length, parts
306      * @return : 0 if the part is valid, 1 if not
307      */
308     int partCheck(char start, char end, long len, long cost, Params *Data)
309     {
310         if (cost <= 0)
311         {
312             return EXIT_FAILURE;
313         }
314         if (len <= 0)
315         {
316             return EXIT_FAILURE;
317         }
318         if (getInd(start, Data) < 0)
319         {
320             return EXIT_FAILURE;
321         }
322         if (getInd(end, Data) < 0)
323         {
324             return EXIT_FAILURE;
325         }
326         return EXIT_SUCCESS;
327     }
328 
329     /**
330      * A function that reads the rail parts data and inserts in into the data structs
331      * @param inp : the input file
```

```
 * @param op : the output file
 * @param Data : A struct that holds the rail parameters ,such as total cost, length, parts
 * @return 0 if all the parts entered successfully, one if not.
 */
int readData(FILE *inp, Params *Data)
{
    Data->parts = (RailPart *) calloc(10, sizeof(RailPart));
    if (Data->parts == NULL)
    {
        return EXIT_FAILURE;
    }
    RailPart *moreParts = NULL;
    char start = 0, end = 0;
    char lineIndex[INT_LEN];
    int cost = 0, len = 0;
    int i = 0;
    char string[MAX_LINE_SIZE];
    while (fgets(string, MAX_LINE_SIZE, inp) != NULL)
    {
        if (stringCheck(string))
        {
            sprintf(lineIndex, "%d.", i + 4);
            printfFunc(LINE_ERROR, lineIndex);
            return EXIT_FAILURE;
        }
        if (sscanf(string, "%c,%c,%d,%d", &start, &end, &len, &cost) != NUM_OF_PARAMS)
        {
            sprintf(lineIndex, "%d.", i + 4);
            printfFunc(LINE_ERROR, lineIndex);
            free(Data->parts);
            return EXIT_FAILURE;
        }
        if (partCheck(start, end, len, cost, Data))
        {
            sprintf(lineIndex, "%d.", i + 4);
            printfFunc(LINE_ERROR, lineIndex);
            free(Data->parts);
            return EXIT_FAILURE;
        }
        RailPart newPart = {.s = start, .e = end, .len = len, .price = cost};
        i++;
        moreParts = (RailPart *) realloc(Data->parts, i * sizeof(RailPart));
        if (moreParts != NULL)
        {
            Data->parts = moreParts;
            Data->parts[i - 1] = newPart;
        }
        else
        {
            sprintf(lineIndex, "%d.", i + 4);
            printfFunc(LINE_ERROR, lineIndex);
            return EXIT_FAILURE;
        }
    }
    Data->numOfParts = i;
    return EXIT_SUCCESS;
}

/**
 * A function that acquires all of the data from the file using different function for each part
 * of the file, get num for the length of the rail and the types of connectors, get signs for
 * the types of connectors available, and read data for reading the parts
 * @param input : the input file
 * @param output : the output file
 * @param Data : A struct that holds the rail parameters ,such as total cost, length, parts
 * @return: 0 If data read successfully ,1 if not
 */
int dataAcquisition(FILE *input, Params *Data)
```

```
400 {
401     if (getNum(input, &Data->length))
402     {
403         printfFunc(LINE_ERROR, "1.");
404         return EXIT_FAILURE;
405     }
406     if (getNum(input, &Data->conTypes))
407     {
408         printfFunc(LINE_ERROR, "2.");
409         return EXIT_FAILURE;
410     }
411     if (Data->conTypes == 0)
412     {
413         printfFunc(LINE_ERROR, "2.");
414         return EXIT_FAILURE;
415     }
416     if (getSigns(input, Data))
417     {
418         printfFunc(LINE_ERROR, "3.");
419         return EXIT_FAILURE;
420     }
421     if (readData(input, Data))
422     {
423         free(Data->signs);
424         return EXIT_FAILURE;
425     }
426     return EXIT_SUCCESS;
427 }
428
429 /**
430  * A function that checks if the file is valid, by checking if there is a file, and if it isnt empty
431  * @param argc : number of arguments entered by the user
432  * @param file : the file name as an argument
433  * @param input : the input file
434  * @param output : the output file
435  * @return : 0 if the file has been read successfully 1 , if not
436  */
437 int fileReadTest(FILE *input)
438 {
439     if (input == NULL)
440     {
441         printfFunc(FILE_ERROR, "");
442         return EXIT_FAILURE;
443     }
444     int c = fgetc(input);
445     if (c == EOF)
446     {
447         printfFunc(EMPTY_ERROR, "");
448         return EXIT_FAILURE;
449     }
450     ungetc(c, input);
451     return EXIT_SUCCESS;
452 }
453
454 /**
455  * A function that checks if there are enough arguments in
456  * @param argc : the number of arguments entered
457  * @param output: the output file
458  * @return 0 if the argument number is correct, 1 if not
459  */
460 int argCheck(int argc)
461 {
462     if (argc != ARG_NUM)
463     {
464         printfFunc(ARG_ERROR, "");
465         return EXIT_FAILURE;
466     }
467     return EXIT_SUCCESS;
```

```
468 }
469
470 /**
471  * A program that gets an input file, reads it and calculates the minimal price for a rail
472  * with given length
473  * @return :minimal price if successful , or -1 if not
474  */
475 int main(int argc, char *argv[])
476 {
477     if (argCheck(argc))
478     {
479         return EXIT_FAILURE;
480     }
481     FILE *inputFile = fopen(argv[1], "r");
482     Params Data = {.length = 0, .conTypes = 0, .numOfParts = 0, .signs = NULL, .parts = NULL};
483     if (fileReadTest(inputFile))
484     {
485         return EXIT_FAILURE;
486     }
487     if (dataAcquisition(inputFile, &Data))
488     {
489         return EXIT_FAILURE;
490     }
491     fclose(inputFile);
492     long **matr = creatMatrix(Data.length + 1, Data.conTypes);
493     setPrices(matr, &Data);
494     getPrice(matr, &Data);
495     return EXIT_SUCCESS;
496 }
497
498
499
```