Contents

1 Basic Test Results

```
Running...
Opening tar file
manageStudents.c
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
=====================

=====================
Running test...
OK
Running test...
OK
Test 1 Succeed.
Info: find best student out of list of 1 students.
=====================

=====================
Running test...
OK
Running test...
OK
Test 2 Succeed.
Info: find best student out of list of 1 students, where student's info in in valid.
=====================

=====================
Running test...
OK
Running test...
OK
Test 12 Succeed.
Info: sort a list of 1 student with merge sort.
=====================

=====================
Running test...
OK
Running test...
OK
Test 13 Succeed.
Info: sort a list of 1 student with merge sort, where student's info is invalid.
=====================

=====================
Running test...
OK
Running test...
OK
```

```
Test 17 Succeed.
Info: sort a list of 1 student with quick sort.
====================


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

2 manageStudents.c

```c
/**
 * @file manageStudents.c
 * @author  Brahan Wassan <brahan>
 * @version 1.0
 * @date 13 Nov 2019
 *
 * @brief System to keep track of all the Students who enrolled in some UNI
 *
 * @section DESCRIPTION
 * The system help the UNI to manage the Students.
 * Input  : students enrolled in the UNI.
 * Process: checks if the user input is valid, and then print the Best student,
 * or an sorted array of all the students according to the user OP input
 * Output : best students, or an array of all the students sorted by name or grade.
 */

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

#define USAGE_ERR "Usage: number of argument must match specified format: <manageStudents> <operation>"
#define START_MSG "Enter student info. To exit press q, then enter"
#define ID_ERR_MSG "ERROR: id must be a 10 digits number that does not start with 0\n"
#define NAME_ERR_MSG "ERROR: name can only contain alphabetic characters, whitespaces or '-'\n"
#define GRADE_ERR_MSG "ERROR: grade must be an integer between 0 and 100\n"
#define AGE_ERR_MSG "ERROR: age must be an integer between 18 and 120\n"
#define COUNTRY_ERR_MSG "ERROR: country can only contain alphabetic characters or '-'\n"
#define CITY_ERR_MSG "ERROR: city can only contain alphabetic characters or '-'\n"
#define GENERAL_ERR_MSG "ERROR: info must match specified format\n"
#define BEST_STUDENT_PRINT_FORMAT "%s%s\t%s\t%d\t%u\t%s\t%s\t\n"
#define STUDENT_PRINT_FORMAT "%s\t%s\t%d\t%u\t%s\t%s\t\n"
#define STUDENT_INPUT_FORMAT "%s %[^\t] %[^\t] %[^\t] %[^\t] %[^\t]"
#define ERR_PRINT_FORMAT "%s%s %d\n"
#define IN_LINE "in line"
#define BEST_STUDENT "best student info is: "
#define  MAX_ARGUMENT 41
#define MAX_ARR_LENGTH 151
#define MAX_INPUT 5001
#define VALID_ID_LEN 10
#define OP_IDX 1
#define BEST "best"
#define QUICK "quick"
#define MERGE "merge"
#define MAX_AGE 120
#define MIN_AGE 18
#define MAX_GRADE 100
#define MIN_GRADE 0
#define LINUX_EXIT "q\n"
#define WIN_EXIT "q\r\n"
#define LOW_NUM_VAL_ASCI 48
#define HI_NUM_VAL_ASCI 57
#define VAL_NUM_FIELD 6
#define DECIMAL_FACTOR 10
#define NO_STUDENTS 0
#define MIN_PROG_ARGS 2
#define NO_INPUT_Q 1
#define FIN_INPUT 2
#define INT_TO_CHAR '0'
```

```c
#define UPPER_A_ASCII 65
#define LOWER_A_ASCII 97
#define UPPER_Z_ASCII 90
#define LOWER_Z_ASCII 122
#define DASH_ASCII 45
#define SPACE_ASCII 32
#define ID_FORMAT_ERR 2
#define SINGLE_MSG_FORMAT "%s\n"
#define FAIL_GRADE 0
#define MERGE_SORT_DIV_FACTOR 2

/**
 * defines one students , a student is entity that defined by id, name,age,grade,country,city
 */
typedef struct
{
    char name[MAX_ARGUMENT], city[MAX_ARGUMENT],
            country[MAX_ARGUMENT], id[MAX_ARGUMENT];
    int age, grade;
    float studentVal;

} Student;
/**
 * an object that represent the student who achieve the greatset grade in the youngest age
 */
Student gBestStudent;
/**
 * all the valid inputs that the user has entered into the program
 */
int gStudentNumber = 0;
/**
 * an array that holds all the valid students the user has entered to the program
 */
Student gStudentArr[MAX_INPUT];

char getInput();

int checkValidity(char line[], int lineNumber);

void updateBestStudent(char const line[MAX_ARR_LENGTH]);

void addNewStudent(char const line[MAX_ARR_LENGTH]);

void mergeSort(Student arr[], int leftIdx, int rightIdx);

void quickSort(Student arr[], int leftIdx, int rightIdx);

void printStudentArr();

/**
 * main program, manage the manageStudents program
 * @param argc the number of argument the program got
 * @param argv the argument the program got
 * @return 0 if the program finished successfully, 1 otherwise
 */
int main(int argc, char *argv[])
{
    if (argc != MIN_PROG_ARGS)
    {
        printf(SINGLE_MSG_FORMAT, USAGE_ERR);
    }
    else
    {
        int inputSituation = getInput(gStudentArr);

        if (gStudentNumber == NO_STUDENTS)
        {
            return EXIT_FAILURE;
```

```
128         }
129         if (inputSituation == NO_INPUT_Q)
130         {
131             return EXIT_FAILURE;
132         }
133         if (strcmp(argv[OP_IDX], BEST) == false)
134         {
135             if (inputSituation == FIN_INPUT)
136             {
137                 printf(BEST_STUDENT_PRINT_FORMAT, BEST_STUDENT, gBestStudent.id,
138                        gBestStudent.name, gBestStudent.grade,
139                        gBestStudent.age, gBestStudent.country,
140                        gBestStudent.city);
141             }
142         }
143         else
144         {
145             if (strcmp(argv[OP_IDX], QUICK) == false)
146             {
147                 quickSort(gStudentArr, 0, gStudentNumber - 1);
148             }
149             if (strcmp(argv[OP_IDX], MERGE) == false)
150             {
151                 mergeSort(gStudentArr, 0, gStudentNumber - 1);
152             }
153             printStudentArr();
154         }
155     }
156     return EXIT_SUCCESS;
157 }
158 
159 /**
160  * prints all the students in the studentArray
161  */
162 void printStudentArr()
163 {
164     for (int i = 0; i < gStudentNumber; ++i)
165     {
166         printf(STUDENT_PRINT_FORMAT, gStudentArr[i].id,
167                gStudentArr[i].name, gStudentArr[i].grade,
168                gStudentArr[i].age, gStudentArr[i].country,
169                gStudentArr[i].city);
170     }
171 }
172 
173 /**
174  * the function gets input from the user according to predefined format
175  * @return 0 if the program got input, 1 otherwise
176  */
177 char getInput()
178 {
179     gBestStudent.studentVal = 0;
180     int lineCounter = 0;
181     char line[MAX_ARR_LENGTH];
182     bool isEqual = false;
183     while (lineCounter < MAX_INPUT)
184     {
185         printf(SINGLE_MSG_FORMAT, START_MSG);
186         fgets(line, MAX_ARR_LENGTH, stdin);
187         if ((strcmp(line, WIN_EXIT) != isEqual) && (strcmp(line, LINUX_EXIT) != isEqual))
188         {
189             int isValid = checkValidity(line, lineCounter);
190             if (isValid == isEqual)
191             {
192                 updateBestStudent(line);
193                 addNewStudent(line);
194             }
195             ++lineCounter;
```

```
196             continue;
197         }
198         if ((strcmp(line, WIN_EXIT) == isEqual) || (strcmp(line, LINUX_EXIT) == isEqual))
199         {
200             if (lineCounter == NO_STUDENTS)
201             {
202                 return NO_INPUT_Q;
203             }
204             return FIN_INPUT;
205         }
206     }
207     return EXIT_SUCCESS;
208 }
209
210 /**
211  * evaluate the students by grade/age
212  * @param grade the student grade
213  * @param age the student age
214  * @return the student score
215  */
216 float evaluateStudent(int grade, int age)
217 {
218     if (grade == FAIL_GRADE)
219     {
220         return FAIL_GRADE;
221     }
222     return (float) grade / (float) age;
223 }
224
225 /**
226  * chec if the id is valid
227  * @param id the input id
228  * @return 1 if the were problem with the id values, 2 if there were problem with the id format,
229  * 0 if there were no problems
230  */
231 int checkId(char const id[])
232 {
233     if (id[0] == INT_TO_CHAR)
234     {
235         return EXIT_FAILURE;
236     }
237     unsigned int size = strlen(id);
238     if (size != VALID_ID_LEN)
239     {
240         return EXIT_FAILURE;
241     }
242     else
243     {
244         unsigned int i;
245         for (i = 0; i < size; ++i)
246         {
247             if ((id[i] < LOW_NUM_VAL_ASCI || id[i] > HI_NUM_VAL_ASCI))
248             {
249                 return ID_FORMAT_ERR;
250             }
251         }
252         return EXIT_SUCCESS;
253     }
254 }
255
256 bool isUpperCase(char const letter)
257 {
258     if (!((letter >= UPPER_A_ASCII && letter <= UPPER_Z_ASCII)) && letter != DASH_ASCII)
259     {
260         return true;
261     }
262     return false;
263 }
```

```
264
265 bool isLowerCase(char const letter)
266 {
267     if (!(((letter >= LOWER_A_ASCII) && (letter <= LOWER_Z_ASCII)) && letter != DASH_ASCII))
268     {
269         return true;
270     }
271     return false;
272 }
273
274 /**
275  * check if the given strings (name,country,city) are valid
276  * @param string the string we want to check
277  * @param isName flag that indicate if the input is the name
278  * @return 1 if there is problem with the string,0 otherwise
279  */
280 int checkStrings(char const string[], int isName)
281 {
282     int size = (int) strlen(string);
283     for (int i = 0; i < size; ++i)
284     {
285         if (isName)
286         {
287             bool isTwoNames = (string[i] != SPACE_ASCII);
288             if (isLowerCase(string[i]) && isTwoNames && isUpperCase(string[i]))
289             {
290                 return EXIT_FAILURE;
291             }
292         }
293         else
294         {
295             if (isLowerCase(string[i]) && isUpperCase(string[i]))
296             {
297                 return EXIT_FAILURE;
298             }
299         }
300     }
301     return EXIT_SUCCESS;
302 }
303
304 /**
305  * checks if the age is in the given limit 18-120
306  * @param age the input age
307  * @return 1 if there is problem,0 otherwise
308  */
309 int checkAge(int age)
310 {
311     if (age > MAX_AGE || age < MIN_AGE)
312     {
313         return EXIT_FAILURE;
314     }
315     return EXIT_SUCCESS;
316 }
317
318 /**
319  *  check if the supposed to be integer fields contain only digits
320  * @param number the given input
321  * @return 1 if there is problem,0 otherwise
322  */
323 int checkDigits(const char number[MAX_ARGUMENT])
324 {
325     int numi, i;
326     int size = (int) strlen(number);
327     for (i = 0; i < size; ++i)
328     {
329         numi = (int) (number[i]);
330         if (numi < LOW_NUM_VAL_ASCI || numi > HI_NUM_VAL_ASCI)
331         {
```

```
332             return EXIT_FAILURE;
333         }
334     }
335     return EXIT_SUCCESS;
336 }
337
338 /**
339  * converts char digit array to integer
340  * @param number the given input
341  * @return the integer
342  */
343 int convertCharToInt(char const number[MAX_ARGUMENT])
344 {
345     int i, integer = 0;
346     int size = (int) strlen(number);
347     for (i = 0; i < size; ++i)
348     {
349         integer = integer * DECIMAL_FACTOR + (number[i] - INT_TO_CHAR);
350     }
351     return integer;
352 }
353
354 /**
355  * checks if the grade is within the limit 0-100
356  * @param grade the given grade
357  * @return 1 if there is a problem, 0 otherwise
358  */
359 int checkGrade(int grade)
360 {
361     if (grade > MAX_GRADE || grade < MIN_GRADE)
362     {
363         return EXIT_FAILURE;
364     }
365     return EXIT_SUCCESS;
366 }
367
368 /**
369  * add a new student to the student array
370  * @param line valid input line
371  */
372 void addNewStudent(char const line[MAX_ARR_LENGTH])
373 {
374     {
375         char name[MAX_ARGUMENT], city[MAX_ARGUMENT], country[MAX_ARGUMENT],
376                 id[MAX_ARGUMENT], age[MAX_ARGUMENT], grade[MAX_ARGUMENT];
377         sscanf(line, STUDENT_INPUT_FORMAT, id, name, grade, age, country, city);
378         int valGrade = convertCharToInt(grade);
379         int valAge = convertCharToInt(age);
380         float stuVal = evaluateStudent(valGrade, valAge);
381
382         Student newStudent;
383         newStudent.studentVal = stuVal;
384         strcpy(newStudent.name, name);
385         strcpy(newStudent.id, id);
386         strcpy(newStudent.country, country);
387         strcpy(newStudent.city, city);
388         newStudent.grade = valGrade;
389         newStudent.age = valAge;
390         gStudentArr[gStudentNumber] = newStudent;
391         ++gStudentNumber;
392     }
393 }
394
395 /**
396  * update the best student global variable
397  * @param line valid student input
398  */
399 void updateBestStudent(char const line[MAX_ARR_LENGTH])
```

```
400 {
401     char name[MAX_ARGUMENT], city[MAX_ARGUMENT], country[MAX_ARGUMENT],
402             id[MAX_ARGUMENT], age[MAX_ARGUMENT], grade[MAX_ARGUMENT];
403     sscanf(line, STUDENT_INPUT_FORMAT, id, name, grade, age, country, city);
404     int valGrade = convertCharToInt(grade);
405     int valAge = convertCharToInt(age);
406     float stuVal = evaluateStudent(valGrade, valAge);
407     if (stuVal > gBestStudent.studentVal || (stuVal == 0 && gBestStudent.studentVal == 0 && gStudentNumber == 0))
408     {
409         gBestStudent.studentVal = stuVal;
410         strcpy(gBestStudent.name, name);
411         strcpy(gBestStudent.id, id);
412         strcpy(gBestStudent.country, country);
413         strcpy(gBestStudent.city, city);
414         gBestStudent.grade = valGrade;
415         gBestStudent.age = valAge;
416     }
417 }
418
419 /**
420  * checks the validity of a single input line
421  * @param line the user input
422  * @param lineNumber the input line number, will be used to print the error line
423  * @return 1 if there is a problem with the user input,0 otherwise
424  */
425 int checkValidity(char line[MAX_ARR_LENGTH], int lineNumber)
426 {
427     char name[MAX_ARGUMENT], city[MAX_ARGUMENT], country[MAX_ARGUMENT],
428             id[MAX_ARGUMENT], age[MAX_ARGUMENT], grade[MAX_ARGUMENT];
429
430     int argsNum = sscanf(line, STUDENT_INPUT_FORMAT,
431                          id, name, grade, age, country, city);
432     bool isEqual = false;
433     if (argsNum < VAL_NUM_FIELD)
434     {
435         printf(ERR_PRINT_FORMAT, GENERAL_ERR_MSG, IN_LINE, lineNumber);
436         return EXIT_FAILURE;
437     }
438     int idSituation = checkId(id);
439     if (idSituation == true)
440     {
441         printf(ERR_PRINT_FORMAT, ID_ERR_MSG, IN_LINE, lineNumber);
442         return EXIT_FAILURE;
443     }
444     if (idSituation == ID_FORMAT_ERR)
445     {
446         printf(ERR_PRINT_FORMAT, GENERAL_ERR_MSG, IN_LINE, lineNumber);
447         return EXIT_FAILURE;
448     }
449     else if (checkStrings(name, true) != isEqual)
450     {
451         printf(ERR_PRINT_FORMAT, NAME_ERR_MSG, IN_LINE, lineNumber);
452         return EXIT_FAILURE;
453     }
454     else if ((checkDigits(grade) != isEqual) || checkGrade(convertCharToInt(grade)) != isEqual)
455     {
456         printf(ERR_PRINT_FORMAT, GRADE_ERR_MSG, IN_LINE, lineNumber);
457         return EXIT_FAILURE;
458     }
459     else if ((checkDigits(age) != isEqual) || (checkAge(convertCharToInt(age)) != isEqual))
460     {
461         printf(ERR_PRINT_FORMAT, AGE_ERR_MSG, IN_LINE, lineNumber);
462         return EXIT_FAILURE;
463     }
464     else if (checkStrings(city, false) != isEqual)
465     {
466         printf(ERR_PRINT_FORMAT, CITY_ERR_MSG, IN_LINE, lineNumber);
467         return EXIT_FAILURE;
```

```
468         }
469         else if (checkStrings(country, false) != isEqual)
470         {
471             printf(ERR_PRINT_FORMAT, COUNTRY_ERR_MSG, IN_LINE, lineNumber);
472             return EXIT_FAILURE;
473         }
474         return EXIT_SUCCESS;
475     }
476
477     ////////////////////////////////////// SORT//////////////////////////////////////////////
478     /**
479      * merges the subarrays
480      * @param arr the array we want to sort
481      * @param leftIdx the leftmost index of the block
482      * @param divPoint the array divide
483      * @param rightIdx
484      */
485     void merge(Student arr[], int leftIdx, int divPoint, int rightIdx)
486     {
487         int rightSubArrLen = rightIdx - divPoint;
488         int leftSubArrLen = divPoint - leftIdx + 1;
489         int idx1, idx2, idx3;
490         Student left[MAX_INPUT], right[MAX_INPUT];
491         for (idx2 = 0; idx2 < rightSubArrLen; idx2++)
492         {
493             right[idx2] = arr[divPoint + 1 + idx2];
494         }
495         for (idx1 = 0; idx1 < leftSubArrLen; idx1++)
496         {
497             left[idx1] = arr[leftIdx + idx1];
498         }
499         idx1 = 0, idx2 = 0, idx3 = leftIdx;
500         while (idx1 < leftSubArrLen && idx2 < rightSubArrLen)
501         {
502             if (left[idx1].grade <= right[idx2].grade)
503             {
504                 arr[idx3] = left[idx1];
505                 ++idx1;
506             }
507             else
508             {
509                 arr[idx3] = right[idx2];
510                 ++idx2;
511             }
512             ++idx3;
513         }
514
515         while (idx1 < leftSubArrLen)
516         {
517             arr[idx3] = left[idx1];
518             ++idx1;
519             ++idx3;
520         }
521
522         while (idx2 < rightSubArrLen)
523         {
524             arr[idx3] = right[idx2];
525             ++idx2;
526             ++idx3;
527         }
528     }
529
530     /**
531      * merges the two subarrays
532      * @param arr the array we want to sort
533      * @param leftIdx the leftmost index
534      * @param rightIdx the rightmost index
535      */
```

```
536 void mergeSort(Student arr[], int leftIdx, int rightIdx)
537 {
538     if (leftIdx < rightIdx)
539     {
540         int divPoint = leftIdx + (rightIdx - leftIdx) / MERGE_SORT_DIV_FACTOR;
541         mergeSort(arr, divPoint + 1, rightIdx);
542         mergeSort(arr, leftIdx, divPoint);
543         merge(arr, leftIdx, divPoint, rightIdx);
544     }
545 }
546
547 /**
548  * the swap function we saw in the lecture
549  * @param arg1 the first student
550  * @param arg2 the second student
551  */
552 void swap(Student *arg1, Student *arg2)
553 {
554     Student temp;
555     temp.studentVal = arg1->studentVal;
556     strcpy(temp.name, arg1->name);
557     strcpy(temp.id, arg1->id);
558     strcpy(temp.country, arg1->country);
559     strcpy(temp.city, arg1->city);
560     temp.grade = arg1->grade;
561     temp.age = arg1->age;
562
563     arg1->studentVal = arg2->studentVal;
564     strcpy(arg1->name, arg2->name);
565     strcpy(arg1->id, arg2->id);
566     strcpy(arg1->country, arg2->country);
567     strcpy(arg1->city, arg2->city);
568     arg1->grade = arg2->grade;
569     arg1->age = arg2->age;
570
571     arg2->studentVal = temp.studentVal;
572     strcpy(arg2->name, temp.name);
573     strcpy(arg2->id, temp.id);
574     strcpy(arg2->country, temp.country);
575     strcpy(arg2->city, temp.city);
576     arg2->grade = temp.grade;
577     arg2->age = temp.age;
578 }
579
580 /**
581  * helper function for the quickSort function, sort the partition blocs in the array
582  * @param arr the array we want to sort
583  * @param low the leftmost index in the array - defines the "block" left boarder
584  * @param high the rightmost index in the array - defines the "block" right boarder
585  * @return sorted block
586  */
587 int partition(Student arr[], int low, int high)
588 {
589     int idx2, idx1 = low - 1;
590     char pivot[MAX_ARGUMENT];
591     strcpy(pivot, arr[high].name);
592     for (idx2 = low; idx2 <= high - 1; ++idx2)
593     {
594         int isPrior = strcmp(arr[idx2].name, pivot);
595         if (isPrior <= 0)
596         {
597             ++idx1;
598             swap(&arr[idx1], &arr[idx2]);
599         }
600     }
601     swap(&arr[idx1 + 1], &arr[high]);
602     return (idx1 + 1);
603 }
```

```
604
605 /**
606  *
607  * @param arr the array we want to sort, will be sorted by name
608  * @param leftIdx the left boarder of the given array, used to calculate the divide point for the partition
609  * @param rightIdx the right boarder of the given array, used to calculate the divide point for the partition
610  */
611 void quickSort(Student arr[], int leftIdx, int rightIdx)
612 {
613     if (leftIdx < rightIdx)
614     {
615         int divPoint = partition(arr, leftIdx, rightIdx);
616         quickSort(arr, leftIdx, divPoint - 1);
617         quickSort(arr, divPoint + 1, rightIdx);
618     }
619 }
```