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 3 Succeed.
Info: sort a list of 1 student with merge sort.
=====================

=====================
Running test...
OK
Running test...
OK
Test 4 Succeed.
Info: sort a list of 1 student with merge sort, where student's info is invalid.
=====================

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

```
Test 5 Succeed.
Info: sort a list of 1 student with quick sort.
====================

****************************************
*     Presubmission script passed      *
****************************************

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

2 manageStudents.c

```c
#include <stdio.h>
#include <string.h>

#define ID_ERROR "ERROR:Id should be in exactly 10 numbers format, when the first number is not 0\n"
#define NAME_ERROR "ERROR: name can only contain alphabetic characters or '-'\n"
#define COUNTRY_ERROR "ERROR: country can only contain alphabetic characters or '-'\n"
#define CITY_ERROR  "ERROR: city can only contain alphabetic characters or '-'\n"
#define GRADE_ERROR "ERROR: Invalid grade value, grade should be a number between 0 to 100\n"
#define AGE_ERROR "ERROR: Invalid Age value, age should be a number between 18 to 120\n"
#define INPUT_ERROR "ERROR: info must match specified format\n"
#define INPUT_REQUEST "Enter student info. To exit press q, then enter\n"
#define ARGUMENT_ERROR "USAGE: sortStudents , use best ,quick or merge as args for the program\n"
#define EOF_ERROR "ERROR: could not read info\n"
#define LINE_LENGTH 61
#define FIELD_LENGTH 61
#define SUCCESS 0
#define FAILURE 1
#define ITEM_AMOUNT 6
#define ID_LENGTH 10
#define MIN_ID 1000000000
#define MAX_ID 9999999999
#define MAX_GRADE 100
#define MAX_AGE 120
#define MIN_AGE 18
#define TEMP_SIZE 2751
#define RECORDS_SIZE 5500
#define ARG_BEST  "best"
#define ARG_QUICK  "quick"
#define ARG_MERGE  "merge"
#define QUIT_SIGNAL "q\n"
#define THRESHOLD -1
#define ARG_NUM 2



/**
 * yoav eshed 305384869 - EX1 - student information management
 */


/**
 * struct that represents a student with the following details: id number, name, grade,age,country
 * and city
 */
typedef struct Student
{
    long id;
    char name[FIELD_LENGTH];
    unsigned int grade;
    unsigned int age;
    char country[FIELD_LENGTH];
    char city[FIELD_LENGTH];
} Student;

/**
 * A function that calculates the first digit.
 * The function calculates the first digit of a number and returns it
 * @param num: the number which is analysed
 * @return: the first digit of the number
```

```
 */
long firstDigit(long num)
{
    long first;
    first = num;
    while (first >= ID_LENGTH)
    {
        first = first / ID_LENGTH;
    }
    return first;
}

/**
 * A test for i.d number requirements
 * The function gets a number and makes sure it is ten digit long and the first number is not zero.
 * @param id_val: and id number enterd by the user.
 * @return: 0 if the i.d number is up to the requirements and 1 if not.
 */
int idTest(long idVal)
{
    if ((idVal < MIN_ID || idVal > MAX_ID) || firstDigit(idVal) == 0)
    {
        return FAILURE;
    }
    return SUCCESS;
}

/**
 * A test for string requirements
 * The function gets a string and makes sure that it doesnt contain anything else but letters,
 * spaces or dashes
 * @param expression_val: a string
 * @return: 0 if the string is up to the requirements , 1 if not.
 */
int wordTest(char *expression)
{
    int i = 0;
    long length = strlen(expression);
    for (i = 0; i < length; i++)
    {
        if ((expression[i] >= 'a' && expression[i] <= 'z') || (expression[i] >= 'A' &&
            expression[i] <= 'Z') ||
            (expression[i] == ' ') || (expression[i] == '-'))
        {
            continue;
        }
        return FAILURE;
    }
    return SUCCESS;
}

/**
 *A test for grade value requirements
 * The test makes sure the grade value is between 0 and 100
 * @param grade_val : The student's grade
 * @return 0 if the grade value is up to requirements and 1 if not
 */
int gradeTest(unsigned int gradeVal)
{
    if (gradeVal > MAX_GRADE)
    {
        return FAILURE;
    }
    return SUCCESS;
}

/**
 * *A test for age value requirements
```

```c
 * The age value should be between 18 and 120
 * @param age_val: the students grade
 * @return 0 if the age value is up to requirements and 1 if not
 */
int ageTest(unsigned int ageVal)
{
    if (ageVal < MIN_AGE || ageVal > MAX_AGE)
    {
        return FAILURE;
    }
    return SUCCESS;
}

/**
 * A function that runs all the tests required to make sure that the input the user entered is
 * valid.
 * @param id_value: the student's  id number
 * @param name_value: the student's name
 * @param grade_value: the student's grade
 * @param age_value: the student's age
 * @param country_value : the student's country of origin
 * @param city_value: the student's city of origin
 * @return 0 if all test passed, 1 if not
 */
int inputTest(long idValue, char *nameValue, unsigned int gradeValue, unsigned int ageValue,
              char *countryValue, char *cityValue, unsigned int lineNum)
{
    if (idTest(idValue) != 0)
    {
        printf(ID_ERROR "in line %d\n", lineNum);
        return FAILURE;
    }
    if (wordTest(nameValue) != 0)
    {
        printf(NAME_ERROR "in line %d\n", lineNum);
        return FAILURE;
    }
    if (gradeTest(gradeValue) !=0)
    {
        printf(GRADE_ERROR "in line %d\n", lineNum);
        return FAILURE;
    }
    if (ageTest(ageValue) !=0)
    {
        printf(AGE_ERROR "in line %d\n", lineNum);
        return FAILURE;
    }
    if (wordTest(countryValue) != 0)
    {
        printf(COUNTRY_ERROR "in line %d\n", lineNum);
        return FAILURE;
    }
    if (wordTest(cityValue) !=0)
    {
        printf(CITY_ERROR "in line %d\n", lineNum);
        return FAILURE;
    }
    return SUCCESS;
}

/**
 * A function that creates a student struct with given details
 * The function receives six different student info items and creates a student struct for each
 * students,then it loads the student struct into a larger student struct which contains all
 * of the students that had been  * entered to the system
 * @param id_value: the student's  id number
 * @param name_value: the student's name
 * @param grade_value: the student's grade
```

```
 * @param age_value: the student's age
 * @param country_value : the student's country of origin
 * @param city_value: the student's city of origin
 * @param index : the number of the students in the system
 * @param records: students records archive
 * @return student struct
 */

Student infoLoading(long idVal, char *nameVal, unsigned int gradeVal, unsigned int ageVal,
                    char *countryVal, char *cityVal, unsigned int index, Student *records)
{
    Student newStudent;
    strcpy(newStudent.name, nameVal);
    strcpy(newStudent.country, countryVal);
    strcpy(newStudent.city, cityVal);
    newStudent.id = idVal;
    newStudent.grade = gradeVal;
    newStudent.age = ageVal;
    records[index] = newStudent;
    return newStudent;
}


/**
 * A function that swaps two students structs in an array of structs.
 * @param first : student struct
 * @param second :student struct
 */
void swap(Student *first, Student *second)
{
    Student temp;
    temp = *first;
    *first = *second;
    *second = temp;
}

/**
 * a function that sets the partition as the last object in the array, moves all smaller name
 * structs to the left of the pivot, and all the bigger name structs to the right of the pivot.
 * @param records: struct array of structs
 * @param left: lowest index in the array
 * @param right: Highest index in the array
 * @return
 */
int partition(Student *records, int left, int right)
{
    Student pivot;
    strcpy(pivot.name, records[right].name);

    int i = (left - 1);

    for (int j = left; j <= right - 1; j++)
    {

        if (strcmp(records[j].name, pivot.name) < 0)
        {
            i++;
            swap(&records[i], &records[j]);
        }
    }
    swap(&records[i + 1], &records[right]);
    return (i + 1);
}

/**
 *  A function that preforms quicksort on an struct array of structs
 * @param records :struct of student structs
 * @param left: lowest index in the array
```

```
 * @param right: Highest index in the array
 */
void quickSort(Student *records , int left, int right)
{
    if (left < right)
    {
        int pivot = partition(records, left, right);
        quickSort(records, left, pivot - 1);
        quickSort(records, pivot + 1, right);
    }
}

/**
 * A function that merges arrays
 * The function receives and array and three numbers left
 * @param records :array of structs
 * @param left: The start index of the array
 * @param mid: The middle index of the array
 * @param right: The end index of the array
 */
void merge(Student *records, int left, int mid, int right)
{
    int mergedSize = right - left + 1;
    Student temp[TEMP_SIZE];
    int k = 0;
    int i = left;
    int j = mid + 1;

    while (i <= mid && j <= right)
    {
        if (records[i].grade < records[j].grade)
        {
            temp[k++] = records[i++];
        }
        else
        {
            temp[k++] = records[j++];
        }
    }

    while (i <= mid)
    {
        temp[k++] = records[i++];
    }

    while (j <= right)
    {
        temp[k++] = records[j++];
    }

    for (k = 0; k < mergedSize; ++k)
    {
        records[left + k] = temp[k];
    }
}

/**
 * A function that preforms merge sort on an array of student structs by comparing grades
 * @param records :array of structs
 * @param left: The start index of the array
 * @param right: The end index of the array
 */
void mergeSort(Student *records, int left, int right)
{
    if (left < right)
    {
        int mid = (left + right) / 2;
        mergeSort(records, left, mid);
```

```
332         mergeSort(records, mid + 1, right);
333         merge(records, left, mid, right);
334     }
335 }
336 
337 /**
338  * A function for printing an array of structs by details
339  * @param array: the array to be printed
340  * @param length: length of the array
341  */
342 void printFunc(Student *arr, unsigned int length)
343 {
344     unsigned int ind;
345     for (ind = 0; ind < length; ind++)
346     {
347         printf("%ld,%s,%u,%u,%s,%s\n", arr[ind].id, arr[ind].name, arr[ind].grade, arr[ind].age,
348                arr[ind].country, arr[ind].city);
349     }
350 }
351 
352 /**
353  * A function that checks the program initial argument and executes the wanted process accordingly
354  * @param argument : program's input argument
355  * @param arr: student's records array
356  * @param best_s : The best student's struct
357  * @param arr_length: the length of the array
358  */
359 int argHandler(const char *argument, Student *arr, Student bestStud,  int arrLength,
360                float best)
361 {
362     if (strcmp(argument, ARG_BEST) == SUCCESS)
363     {
364         if (best > THRESHOLD)
365         {
366             printf("best student info is: %ld,%s,%u,%u,%s,%s\n", bestStud.id, bestStud.name,
367                    bestStud.grade, bestStud.age, bestStud.country, bestStud.city);
368         }
369         return SUCCESS;
370     }
371     if (strcmp(argument, ARG_MERGE) == SUCCESS)
372     {
373         mergeSort(arr, 0, arrLength - 1);
374         printFunc(arr, arrLength);
375         return SUCCESS;
376     }
377     if (strcmp(argument, ARG_QUICK) == SUCCESS)
378     {
379         quickSort(arr, 0, arrLength - 1);
380         printFunc(arr, arrLength);
381         return SUCCESS;
382     }
383     return FAILURE;
384 }
385 
386 /**
387  * A Function for checking input arguments
388  * The function checks if an argument has been entered, and if it is one of the three allowed
389  * arguments if it is the function runs, if not ,the function prints and error an exits.
390  * @param arg1 :A string containing an argument
391  * @return 0 if the argument is valid and 1 if not.
392  */
393 int argCheck(const char *arg1)
394 {
395     if ((strcmp(arg1, ARG_BEST) == 0 || strcmp(arg1, ARG_QUICK) == 0 || (strcmp(arg1, ARG_MERGE))
396         == 0))
397     {
398         return FAILURE;
399     }
```

```
400     return SUCCESS;
401 }
402
403 int main(int argc, char *argv[])
404 {
405     unsigned int counter = 0;
406     if (argc != ARG_NUM)//argument validity
407         // check
408     {
409         printf(ARGUMENT_ERROR);
410         return FAILURE;
411     }
412     if (argCheck(argv[1]) == 0)
413     {
414         printf(ARGUMENT_ERROR);
415         return FAILURE;
416     }
417         Student bestStudent = {.id = 0, .name = "", .grade = -1, .age = 1, .country = "", .city = ""};
418     int studentNo = 0;
419     float bestScore = THRESHOLD;
420     Student records[RECORDS_SIZE];
421     char user_line[LINE_LENGTH] = {0};
422     while (strcmp(user_line, QUIT_SIGNAL) != 0)
423     {
424         printf(INPUT_REQUEST);
425 //        fgets(user_line, LINE_LENGTH, stdin);
426         if (fgets(user_line, LINE_LENGTH, stdin) == NULL)
427         {
428             printf(EOF_ERROR "in line %d\n", counter);
429             return FAILURE;
430         }
431         long id = -1;
432         unsigned int grade = -1;
433         unsigned int age = 0;
434         unsigned int itemsNO;
435         char name[FIELD_LENGTH];
436         char country[FIELD_LENGTH];
437         char city[FIELD_LENGTH];
438         if (strcmp(user_line, QUIT_SIGNAL) == 0)
439         {
440             argHandler(argv[1], records, bestStudent, studentNo, bestScore);
441             return SUCCESS;
442         }
443
444         itemsNO = sscanf(user_line, "%ld,%[^,],%u,%u,%[^,],%[^,\n]", &id, name, &grade, &age,
445                          country, city); //dividing user input into variables
446         if (itemsNO != ITEM_AMOUNT) //valid input test
447         {
448             printf(INPUT_ERROR "in line %d\n", counter);
449
450         }
451         else
452         {
453             if (!(inputTest(id, name, grade, age, country, city, counter)))
454             {
455                 Student curStudent = infoLoading(id, name, grade, age, country, city, studentNo,
456                                                  records); //creating new student struct
457                 studentNo++;
458                 float score = ((float) grade / (float) age); //finding the best student
459                 if (bestScore < score)
460                 {
461                     bestScore = score;
462                     bestStudent = curStudent;
463                 }
464             }
465         }
466         counter++;
467     }
```

```
468         argHandler(argv[1], records, bestStudent, studentNo, bestScore);
469         return SUCCESS;
470     }
471
472
473
```