Contents

1 Basic Test Results 2

2 ./manageStudents.c 4

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...
manageStudents.c: In function ‘initilaizeStudent’:
manageStudents.c:325:2: warning: ‘gets’ is deprecated [-Wdeprecated-declarations]
  gets( line );
  ^~~~
In file included from manageStudents.c:21:
/usr/include/stdio.h:583:14: note: declared here
 extern char *gets (char *__s) __wur __attribute_deprecated__;
              ^~~~
/usr/bin/ld: /tmp/cchGcomD.o: in function `initilaizeStudent':
manageStudents.c:(.text+0x52c): warning: the `gets' function is dangerous and should not be used.
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  david ponarovsky
 * @version 1.0
 * @date 5 Nov 2019
 *
 * @brief System to keep track of the cooking times.
 *
 * @section LICENSE
 * This program is not a free software; bla bla bla...
 *
 * @secton DESCRIPTION
 * The system keeps track of the cooking times.
 * Input  : Times to be measured.
 * Process: Summing times if needed, waiting for the correct time and
 *          notifying the user.
 * Output : Notification.
 */

// ------------------------------ includes ------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include "manageStudents.h"
// ... rest of includes from the system
// ... all my includes

// -------------------------- const definitions -------------------------

// -------------------------- const definitions -------------------------
#define UPPER_BOUND_LINE_SIZE 150
#define UPPER_BOUND_FIELD_SIZE 40
#define UPPER_BOUND_INPUT_LINES 5000

// informative massage
const char * ENTER_STUDENT =  "Enter student info. To exit press q, then enter";

const char * ERRORNAME              = "ERROR: name can only contain alphabetic characters or '-' and spaces";
const char * ERRORCITYNAME          = "ERROR: city can only contain alphabetic characters or '-'";
const char * ERRORCOUNTRYNAME       = "ERROR: country can only contain alphabetic characters or '-'";
const char * ERRORIDES              = "ERROR: id must contain ten digits and cannot start with zero";
const char * ERRORAGES              = "ERROR: age can only be integer between 18 to 80";
const char * ERRORGRADES            = "ERROR: grade can only be integer between 0 to 100";
const char * BESTSTUDENTOUT         = "best student info is: ";
const char * BESTOPT                = "best";
const char * MERGEOPT               = "merge";
const char * QUICKOPT               = "quick";
const char * USAGE                                  = "USAGE: ./manageStudents (best|quick|merge)";
const int CONTINUE      = 1;
const int STOP          = 0;
const int LOWERGRADE    = 0;
const int UPPERGRADE    = 100;
const int LOWERAGE      = 18;
const int UPPERAGE      = 80;
const char QUIT         = 'q';
const char ZERO         = '0';
const int DROPLINE      = 0;

static unsigned long ids              [ UPPER_BOUND_INPUT_LINES ] = {0};
```

```
60  static int ages                 [ UPPER_BOUND_INPUT_LINES ] = {0};
61  static int grades               [ UPPER_BOUND_INPUT_LINES ] = {0};
62  static char names               [ UPPER_BOUND_INPUT_LINES ][ UPPER_BOUND_FIELD_SIZE ] = {0};
63  static char countrys            [ UPPER_BOUND_INPUT_LINES ][ UPPER_BOUND_FIELD_SIZE ] = {0};
64  static char citys               [ UPPER_BOUND_INPUT_LINES ][ UPPER_BOUND_FIELD_SIZE ] = {0};
65  static int students = 0;
66  static int lines = 0;
67  static int order [UPPER_BOUND_INPUT_LINES];
68  static int worktype [UPPER_BOUND_INPUT_LINES] = {0};
69  const int AGESPARAMTYPE         = 1;
70  const int GRADESPARAMTYPE       = 2;
71  const int NAMESPARAMTYPE        = 3;
72  const int COUNTRYSPARAMTYPE     = 4;
73  const int CITYSPARAMTYPE        = 5;
74  const int IDSPARAMTYPE          = 6;
75  typedef int (*function)(int, int);
76  typedef int (*scan_function) ( char param[] );
77  typedef int (*check_function) ( char param[] );
78
79
80  // ------------------------- functions definitions ------------------------
81
82  void resetStudent( );
83  int checkAges(  ) ;
84  int checkGrades(  ) ;
85  int checkNames(  ) ;
86  int checkCountrys(  ) ;
87  int initilaizeStudent();
88  char peekStdin();
89  void popSpaces();
90  int isLetter(char c);
91  int isSpace(char c);
92  int parseNameWithSpaces(int scan_feedback, char * str);
93  void initilaizeStudentsList();
94  double studentFactor( int student );
95  int bestStudent();
96  void initilaizeSort();
97  void mergesort(int start, int end, function compareFunction);
98  void quicksort(int start, int end, function compareFunction);
99  void merge(int start_1, int end_1, int start_2, int end_2, function compareFunction);
100 int compareGrades(int student1, int student2);
101 int compareNames(int student1,  int student2);
102
103 // ----------------------------- functions -----------------------------
104
105 /**
106  * @brief The main function. Actually, does nothing here.
107  * @return 0, to tell the system the execution ended without errors.
108  */
109 char peekStdin()
110 {
111     char temp = getchar();
112     ungetc(temp, stdin);
113     return temp;
114 }
115 /**
116  * @brief The main function. Actually, does nothing here.
117  * @return 0, to tell the system the execution ended without errors.
118  */
119 void popSpaces()
120 {
121     while( isSpace( peekStdin() ) )
122     {
123         getchar();
124     }
125 }
126
127 /**
```

```
128  * @brief isDigit checking if the charter is digit.
129  * @return 1 if the charter is digit else 0
130  */
131 int isDigit( char c )
132 {
133     return ( c <= '9' && c >= '0');
134 }
135
136 /**
137  * @brief The main function. Actually, does nothing here.
138  * @return 0, to tell the system the execution ended without errors.
139  */
140 int isLetter(char c)
141 {
142     return (c == '-') || ( (c >= 'A') && (c <= 'z') );
143 }
144 /**
145  * @brief The main function. Actually, does nothing here.
146  * @return 0, to tell the system the execution ended without errors.
147  */
148 int isSpace(char c)
149 {
150     return c == ' ' || c == '\t' || c == '\n';
151 }
152 /**
153  * @brief printing the error to stdout.
154  */
155 void printError(const char * error)
156 {
157     printf( "%s\nin line %d\n", error, lines);
158 }
159 /**
160  * @brief checking that the input has entered in the type format.
161  */
162 int checkScan( int scan_feedback)
163 {
164   if ( scan_feedback != 1 )
165   {
166
167     return DROPLINE;
168   }
169   return CONTINUE;
170 }
171 /**
172  * @brief checking that the input which entered is string.
173  */
174 int checkStr( int scan_feedback, char * str)
175 {
176   if ( !checkScan(scan_feedback))
177   {
178     return DROPLINE;
179   }
180   else
181   {
182     char * pointer = str;
183     for ( ; *pointer ; pointer++ )
184     {
185       if ( ! isLetter( *pointer ) )
186       {
187
188         return DROPLINE;
189       }
190     }
191   }
192   return CONTINUE;
193 }
194
195 int checkStrContainDigits(char param [])
```

```
196 {
197     char * pointer = param;
198     for (; *pointer; pointer++ )
199     {
200         if ( !isDigit(*pointer) )
201         {
202             return DROPLINE;
203         }
204     }
205     return CONTINUE;
206 }
207 
208 /**
209  * @brief checking that the input which entered is integer in given range.
210  */
211 int checkInt( int scan_feedback, int val, int lower, int upper)
212 {
213     if ( !checkScan(scan_feedback))
214     {
215         return DROPLINE;
216     }
217     else
218     {
219         if ( ( val >= lower ) && ( val <= upper ))
220         {
221             return CONTINUE;
222         }
223         return DROPLINE;
224     }
225 }
226 
227 /**
228  * @brief parsing the names of the student, dealing with naems which contains-
229  *              -a spaces. store the name in the last empty cell inside the global-
230  *                -static names array.
231  * @return nothing.
232  */
233 int parseNameWithSpaces(int scan_feedback, char * str)
234 {
235   if ( !checkScan(scan_feedback))
236   {
237     return DROPLINE;
238   }
239   else
240   {
241     char * pointer = str;
242     for ( ; *pointer ; pointer++ )
243     {
244       if ((*pointer != ' ') && (!isLetter( *pointer )))
245       {
246         return DROPLINE;
247       }
248     }
249   }
250   return CONTINUE;
251 
252 }
253 /**
254  * @brief rest the student fields.
255  */
256 void restStrField( char field [] )
257 {
258     for ( int i = 0; i < UPPER_BOUND_FIELD_SIZE; i++ )
259     {
260         field[i] = 0;
261     }
262 }
263 /**
```

```
264  * @brief printing the student to stdot.
265  */
266 void printStudent(int student)
267 {
268     printf("%lu\t%s\t%d\t%d\t%s\t%s\t\n", ids[student], names[student],
269      grades[student], ages[student], countrys[student], citys[student]  );
270 }
271
272 /**
273  * @brief rest the field of the given student.
274  */
275 void resetStudent( )
276 {
277     ids[students]             = 0;
278     grades[students]         = 0;
279     ages[students]             = 0;
280     restStrField(names[students]);
281     restStrField(citys[students]);
282     restStrField(countrys[students]);
283     //scanf("%[^\n]");
284     //char line [ UPPER_BOUND_LINE_SIZE ] = {0};
285     //gets(line);
286 }
287
288
289 /**
290  * @brief scanning word while using '\t' as dilameter.
291  * @return the position of the end of the word inside the line.
292  */
293 int scanWord(char line[], char param[], int start )
294 {
295     int i = start;
296
297     for (int k = 0; (line[i] != '\t') && (line[i] != '\n') ; i++ )
298     {
299         param[k++] = line[i];
300     }
301     return i + 1;
302 }
303
304 /**
305  * @brief initilaize the students by asking for the parameters from the user-
306  * -and store them into the static arrays.
307  * @return 0 if the user press 'q' otherwise returns 1.
308  */
309 int initilaizeStudent()
310 {
311     // requesting for input student.
312      printf("%s\n", ENTER_STUDENT);
313     // gettig rid of spaces.
314
315     // check if the user press 'q'.
316     if ( peekStdin() == QUIT )
317     {
318         // poping 'q' from the stdin stream.
319         getchar();
320         // than return 0, which will stops input loop.
321         return STOP;
322     }
323
324     char line [ UPPER_BOUND_LINE_SIZE ] = {0};
325     gets( line );
326
327     char paramId [ UPPER_BOUND_FIELD_SIZE ]             = {0};
328     char paramName [ UPPER_BOUND_FIELD_SIZE ]         = {0};
329     char paramGrade [ UPPER_BOUND_FIELD_SIZE ]         = {0};
330     char paramAge [ UPPER_BOUND_FIELD_SIZE ]             = {0};
331     char paramCity [ UPPER_BOUND_FIELD_SIZE ]         = {0};
```

```
332     char paramCountry [ UPPER_BOUND_FIELD_SIZE ]     = {0};
333
334     int start = 0;
335     start = scanWord(line, paramId, start );
336     start = scanWord(line, paramName, start );
337     start = scanWord(line, paramGrade, start );
338     start = scanWord(line, paramAge, start );
339     start = scanWord(line, paramCountry, start );
340     start = scanWord(line, paramCity, start );
341
342     if (paramId[0] == ZERO || (checkStrContainDigits(paramId) == DROPLINE) )
343     {
344         resetStudent();
345       printError( ERRORIDES );
346       return CONTINUE;
347     }
348     int digits_count = 0;
349     for (digits_count = 0; paramId[digits_count]; digits_count++ )
350     {
351
352     }
353     // parsing the student's id, and store in the id's.
354     int scan_feedback = sscanf(paramId, "%lu", &ids[students] );
355     if ( (digits_count != 10) || checkScan(scan_feedback) == DROPLINE)
356     {
357         printError( ERRORIDES );
358         resetStudent();
359         return CONTINUE;
360     }
361     scan_feedback = sscanf(paramName, "%[^\t]", names[students] );
362     // parsing and storing the student's name.
363     if (parseNameWithSpaces(scan_feedback, names[students]) == DROPLINE)
364     {
365         printError( ERRORNAME );
366         resetStudent();
367         return CONTINUE;
368     }
369     // parding and stroing the rest of the parameters.
370
371     scan_feedback = sscanf(paramGrade, "%d", &grades[students]);
372     if (checkInt(scan_feedback, grades[students], LOWERGRADE, UPPERGRADE)
373         == DROPLINE || (checkStrContainDigits(paramGrade) ==
374         DROPLINE))
375     {
376         printError( ERRORGRADES );
377         resetStudent();
378         return CONTINUE;
379     }
380
381     scan_feedback = sscanf(paramAge, "%d", &ages[students]);
382     if (checkInt(scan_feedback, ages[students], LOWERAGE,
383         UPPERAGE) == DROPLINE  || (checkStrContainDigits(paramAge) ==
384         DROPLINE))
385     {
386         printError( ERRORAGES );
387         resetStudent();
388         return CONTINUE;
389     }
390
391     scan_feedback = sscanf(paramCountry, "%s", countrys[students] );
392     if (checkStr( scan_feedback, countrys[students]) == DROPLINE )
393     {
394         printError( ERRORCOUNTRYNAME );
395         resetStudent();
396         return CONTINUE;
397     }
398     scan_feedback = sscanf(paramCity, "%s", citys[students] );
399     if (checkStr( scan_feedback, citys[students]) == DROPLINE )
```

```
400         {
401             printError( ERRORCITYNAME );
402             resetStudent();
403             return CONTINUE;
404         }
405         // increasing the student counter by one.
406         students++;
407         return CONTINUE;
408     }
409     /**
410      * @brief initilaizes studens untill the 'initilaizeStudent' function return '0'
411      * -which occuer when the user pressing 'q' - the signal which indecate exit-
412      * -operation.
413      * @return nothing.
414      */
415     void initilaizeStudentsList()
416     {
417         while( initilaizeStudent() )
418         {
419             lines++;
420         }
421     }
422     /**
423      * @brief calculate and return the ration of the student grade relative to his-
424      * -age.
425      * @return the ration of the student grade relative to his age.
426      */
427     double studentFactor( int student )
428     {
429         return ((double)grades[student] / ages[student]);
430     }
431     /**
432      * @brief comparing studens by their grades.
433      * @return ture if the first student grade is lower or equal than the other.
434      */
435     int compareGrades(int student1, int student2)
436     {
437         return grades[student1] <= grades[student2];
438     }
439     /**
440      * @brief comparing studens by their names ( abc order ).
441      * @return ture if the first student name is preceding the other.
442      */
443     int compareNames(int student1,  int student2)
444     {
445         return strcmp(names[student2] , names[student1]) > 0;
446     }
447
448     /**
449      * @brief returns the best student by the ratio of the grade
450      *  relative to the age.
451      * @return the index of the best student.
452      */
453     int bestStudent()
454     {
455         // first define the first student as the best one.
456         // todo : handle empty array ...
457         double max = studentFactor(0);
458         int beststudent = 0;
459         // iterating over the rest of the students.
460         for ( int i = 1 ; i < students ; i++ )
461         {
462             // if found batter student than exchange him with the best one.
463             if ( studentFactor(i) > max )
464             {
465                 max = studentFactor(i);
466                 beststudent = i;
467             }
```

```
468         }
469         return beststudent;
470     }
471     /**
472      * @brief initilaize the order array which will storing (after the sorting)
473      *  the position of i' student in sorted order.
474      * @return nothing.
475      */
476     void initilaizeSort()
477     {
478         for (int i = 0; i < students; i++)
479         {
480             order[i] = i;
481         }
482     }
483     /**
484      * @brief the merge function, which mearge two sorted segments by given-
485      *  -comparing function. in the first phase the function will store the sorted-
486      *  -elements into the 'worktype' which is a spair static array at length 5000-
487      *  -(UPPER_BOUND_INPUT_LINES). than in the second phase the function will copy-
488      *  -the content beck to order array. the worktype is anloged to a temp-
489      *  -variable which defined when executing swapping between two variables.
490      * @return nothing.
491      */
492     void merge(int start_1, int end_1, int start_2, int end_2, function compareFunction)
493     {
494         // defining the cursor which will be running on the segments.
495         int cursor_1 = start_1, cursor_2 = start_2;
496         // the cursor_merged will be used to store in worktype.
497         int cursor_merged = start_1;
498         // while there is a cursor which have not reach yet to the end of hi's
499         // -segment.
500         while ( cursor_1 <= end_1 && cursor_2 <= end_2  )
501         {
502             // storing the lower than the two students which are being examined.
503             // and increasing the cursor of the chosen segment.
504             if ( compareFunction(order[cursor_1], order[cursor_2]) )
505             {
506                 worktype[cursor_merged++] = order[cursor_1++];
507             }
508             else
509             {
510                 worktype[cursor_merged++] = order[cursor_2++];
511             }
512         }
513         // if the first cursor not reach to the end of his segment than push-
514         // the rest of the elements to the end of the worktype.
515         while( cursor_1 <= end_1 )
516         {
517             worktype[cursor_merged++] = order[cursor_1++];
518         }
519         // repet on the above to the second cursor.
520         while( cursor_2 <= end_2 )
521         {
522             worktype[cursor_merged++] = order[cursor_2++];
523         }
524         // copying back the elements from the worktype into the order array.
525         for ( int position = start_1 ; position <= end_2  ; position++)
526         {
527             order[position] = worktype[position];
528             worktype[position] = 0;
529         }
530     }
531     /**
532      * @brief implemetion of the merge sort.
533      * @return nothing.
534      */
535     void mergesort(int start, int end, function compareFunction)
```

```
{
    // stopping condition for the reqursive calls.
    if (start == end)
    {
        return;
    }
    // parting the segment to two equals segments. and than sorting each of them.
    int middle = (start + end) / 2;
    mergesort(start, middle, compareFunction);
    mergesort(middle + 1, end, compareFunction);
    // merging the sorted segments.
    merge(start, middle, middle + 1, end, compareFunction);

}

/**
 * @brief swapping two elements in the sorteted array.
 * @return nothing
 */
void swap(int i, int j)
{
    int temp = order[i];
    order[i] = order[j];
    order[j] = temp;
}

/**
 * @brief implemetion of the quicksort sort.
 * @return nothing.
 */
void quicksort(int start, int end, function compareFunction)
{

    if (  (start + 1) >= end )
    {
        return;
    }
    // chosing random parttion
    int segement = start +  (rand() % (end - start));
    // exchanging the loweres values which left to the parttion
    for (int i = start; i < segement; i++ )
    {
        if ( compareFunction(order[segement], order[i]) )
        {
            swap(segement, i);
        }
    }
    // exchanging the loweres values which right to the parttion
    for (int i = segement + 1; i < end; i++ )
    {
        if ( compareFunction(order[i], order[segement]) )
        {
            swap(segement, i);
        }
    }
    quicksort(start, segement, compareFunction);
    quicksort(segement + 1 , end, compareFunction);
}

/**
 * @brief printing the sorted array.
 * @return nothing
 */
void printStudentsSortedOrder()
{
    for ( int k = 0; k < students; k++ )
    {
        printStudent(order[k]);
```

```
604         }
605     }
606
607     /**
608      * @brief printing the USAGE format.
609      * @return nothing
610      */
611     void printUsage()
612     {
613         printf("%s\n", USAGE);
614     }
615
616     /**
617      * @brief The main function. parsing the command line arguments and executing
618      * the requested command of the user.
619      * @return 0 if the program have been run suecssfully.
620      */
621     int main(int argc, char const *argv[])
622     {
623
624         if (argc == 2)
625         {
626
627             if ( strcmp(argv[1], BESTOPT) == 0 )
628             {
629                 initilaizeStudentsList();
630                 if ( students > 0)
631                 {
632                     int beststudent = bestStudent();
633                     printf("%s", BESTSTUDENTOUT);
634                     printStudent(beststudent);
635                 }
636                 else
637                 {
638                     return 1;
639                 }
640             }
641             else if ( strcmp(argv[1], MERGEOPT) == 0  )
642             {
643                 initilaizeStudentsList();
644
645                 if ( students == 0 )
646                 {
647                     return 1;
648                 }
649
650                 initilaizeSort();
651                 mergesort(0 , students - 1, &compareGrades);
652                 printStudentsSortedOrder();
653             }
654             else if ( strcmp(argv[1], QUICKOPT) == 0  )
655             {
656                 initilaizeStudentsList();
657
658                 if ( students == 0 )
659                 {
660                     return 1;
661                 }
662
663                 initilaizeSort();
664                 quicksort(0 , students, &compareNames);
665                 printStudentsSortedOrder();
666             }
667             else
668             {
669                 printUsage();
670                 return 1;
671             }
```

```
672         }
673         else
674         {
675             printUsage();
676             return 1;
677         }
678 
679         /* code */
680         return 0;
681     }
```