```c
/*@author Michal Pawlowski*/

#define _CRT_SECURE_NO_WARNINGS

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

typedef struct element {
    int liczba;
    struct element* pNext;
}element;

/** Funkcja sprawdza parametry wejściowe oraz wczytuje z nich dane do zmiennych
@date 2020-04-15
@param argc ilosc wczytanych parametrow
@param argv tablica wczytanych parametrów
@param ilosc_k wskaźnik na zmienną określającą ilość kolumn
@param ilosc_w wskaźnik na zmienną określającą ilość wierszy
@param zakres_d wskaźnik na zmienną określającą dolny zakres generowanych liczb
@param zakres_g wskaźnik na zmienną określającą górny zakres generowanych liczb
@return prawdę jeżeli podano prawidłowo parametry
*/
bool sprawdzArgumenty(int argc, char* argv[], int* ilosc_k, int* ilosc_w, int* zakres_d, int* zakres_g) {

    ///< sprawdzenie czy zostala podana zgodna ilosc argumentow

    char* k = "-k";
    char* w = "-w";
    char* d = "-d";
    char* g = "-g";
    int zlicz = 0;

    if (argc <= 9) {

        for (int i = 1; i < argc; i++)
        {
            if (!strcmp(argv[i], k)) {

                *ilosc_k = atoi(argv[i + 1]);
                zlicz++;
            }

            if (!strcmp(argv[i], w)) {

                *ilosc_w = atoi(argv[i + 1]);
                zlicz++;
```

```
50              }
51
52              if (!strcmp(argv[i], d)) {
53
54                  *zakres_d = atoi(argv[i + 1]);
55                  zlicz++;
56              }
57
58              if (!strcmp(argv[i], g)) {
59
60                  *zakres_g = atoi(argv[i + 1]);
61                  zlicz++;
62              }
63          }
64
65          if (zlicz == 4) return true;
66
67          else return false;
68      }
69
70      else return false;
71  }
72
73  /** Funkcja wypiuje pomoc
74  @date 2020-04-15
75  */
76  void pomoc() {
77
78      printf("------------------- P O M O C --------------------\n");
79      printf("Uwaga! Program nalezy uruchamiac z 4 parametrami:\n");
80      printf("-w ilosc wierszy\n");
81      printf("-k ilosc kolumn\n");
82      printf("-d dolny zakres generowanych liczb\n");
83      printf("-g gorny zakres generowanych liczb\n");
84      printf("------------------------------------------------\n");
85  }
86
87  /** Funkcja losująca
88  @date 2020-04-15
89  @param zakres_d zmienna określająca dolny zakres generowanej liczby
90  @param zakres_g zmienna określająca górny zakres generowanej liczby
91  @return liczbę pseudo losową
92  */
93  int los(int zakres_d, int zakres_g){
94
95      return zakres_d + rand() % (zakres_g - zakres_d + 1);
96  }
97
98  /** Funkcja generująca plik csv z danymi seudo losowymi
99  @date 2020-04-15
100 @param ilosc_k ilość kolumn
101 @param ilosc_w ilość wierszy
102 @param zakres_d dolny zakres generowanych liczb
```

```
103 @param zakres_g górny zakres generowanych liczb
104 */
105 void generuj(int ilosc_k, int ilosc_w, int zakres_d, int zakres_g) {
106
107     FILE* fp;
108     fp = fopen("dane.csv", "w");
109
110     if (!fp) {
111         printf("Blad zapisu!\n");
112         exit(1);
113     }
114
115     for (int i = 0; i < ilosc_w; i++) {
116
117         for (int j = 0; j < ilosc_k; j++) {
118
119             fprintf(fp, "%d", los(zakres_d, zakres_g));
120             if (j == ilosc_k - 1) fprintf(fp, "\n");
121             else fprintf(fp, ", ");
122         }
123     }
124     fclose(fp);
125 }
126
127 /** Funkcja generująca plik binarny dat z zapisanymi parametrami
128 @date 2020-04-15
129 @param ilosc_k ilość kolumn
130 @param ilosc_w ilość wierszy
131 @param zakres_d dolny zakres generowanych liczb
132 @param zakres_g górny zakres generowanych liczb
133 */
134 void zapiszParamDat(int ilosc_k, int ilosc_w, int zakres_d, int zakres_g) {
135
136     FILE* fDat;
137     fDat = fopen("propertis.dat", "wb");
138
139     if (!fDat) {
140         printf("Blad zapisu!\n");
141         exit(1);
142     }
143
144     fprintf(fDat, "%d ", ilosc_k);
145     fprintf(fDat, "%d ", ilosc_w);
146     fprintf(fDat, "%d ", zakres_d);
147     fprintf(fDat, "%d ", zakres_g);
148
149     fclose(fDat);
150 }
151
152 /** Funkcja generująca plik json z zapisanymi parametrami
153 @date 2020-04-15
154 @param ilosc_k ilość kolumn
155 @param ilosc_w ilość wierszy
```

```
156 @param zakres_d dolny zakres generowanych liczb
157 @param zakres_g górny zakres generowanych liczb
158 */
159 void zapiszParamJson(int ilosc_k, int ilosc_w, int zakres_d, int zakres_g) {
160
161     FILE* fJson;
162     fJson = fopen("propertis.json", "w");
163
164     if (!fJson) {
165         printf("Blad zapisu!\n");
166         exit(1);
167     }
168
169     fprintf(fJson, "{\n");
170     fprintf(fJson, "    %cparams%c: {\n", '"', '"');
171     fprintf(fJson, "        %ckolumny%c: %d,\n", '"', '"', ilosc_k);
172     fprintf(fJson, "        %cwiersze%c: %d,\n", '"', '"', ilosc_w);
173     fprintf(fJson, "        %czakres dolny%c: %d,\n", '"', '"', zakres_d);
174     fprintf(fJson, "        %czakres gorny%c: %d,\n", '"', '"', zakres_g);
175     fprintf(fJson, "    }\n");
176     fprintf(fJson, "}\n");
177
178     fclose(fJson);
179 }
180
181 /** Funkcja dodaje element do listy niepustej w sposób posortowany
182 @date 2020-04-02
183 @param pHead wskaźnik na początek listy
184 @param liczba wartość do dodania
185 */
186 void dodElem(element** pHead, int liczba) {
187
188     element* pHelp = *pHead;
189     element* pHelp2 = NULL;
190
191     if (liczba < pHelp->liczba) {
192         pHelp2 = (element*)malloc(sizeof(element));
193         pHelp2->liczba = liczba;
194         pHelp2->pNext = pHelp;
195         *pHead = pHelp2;
196     }
197     else {
198         while (pHelp->pNext && liczba >= pHelp->pNext->liczba)
199             pHelp = pHelp->pNext;
200
201         if (pHelp->pNext) {
202             pHelp2 = pHelp->pNext;
203             pHelp->pNext = NULL;
204             pHelp->pNext = (element*)malloc(sizeof(element));
205             pHelp->pNext->liczba = liczba;
206             pHelp->pNext->pNext = pHelp2;
207         }
208         else {
```

```
209             pHelp->pNext = (element*)malloc(sizeof(element));
210             pHelp->pNext->liczba = liczba;
211             pHelp->pNext->pNext = NULL;
212         }
213     }
214 }
215
216 /** Funkcja usuwa liste
217 @date 2020-04-02
218 @param pHead wskaźnik na początek lsty
219 */
220 void usunListe(element** pHead) {
221
222     element* pHelp = *pHead;
223
224     while (*pHead) {
225
226         *pHead = (*pHead)->pNext;
227         free(pHelp);
228         pHelp = *pHead;
229     }
230 }
231
232
233 /** Funkcja wypisuje liste
234 @date 2020-04-02
235 @param pHead wskaźnik na początek listy
236 */
237 void wypisz(element* pHead) {
238     while (pHead) {
239         printf("%d  ", pHead->liczba);
240         pHead = pHead->pNext;
241     }
242 }
243
244 /** Funkcja wypisuje wartość zadanego argumentem elementu. W wypadku podania ↩
       nr przekraczającego liczbę elementów, funkcja zwróci ostatni element z ↩
      listy
245 @date 2020-04-02
246 @param pHead wskaźnik na początek listy
247 @param nr numer elementu
248 @return wartość elementu
249 */
250 int wypiszEl(element* pHead, unsigned int nr) {
251
252     if (pHead) {
253
254         for (int i = 1; i < nr; i++) {
255
256             pHead = pHead->pNext;
257         }
258
259         return pHead->liczba;
```

```
260     }
261 }
262
263 /** Funkcja liczy ilosc elementów w liście
264 @date 2020-04-02
265 @param pHead wskaźnik na początek listy
266 @return ilość elementów
267 */
268 int policzEl(element* pHead) {
269
270     int i = 0;
271
272     while (pHead) {
273         i++;
274         pHead = pHead->pNext;
275     }
276     return i;
277 }
278
279 /** Funkcja podaje element o maksymalnej wartości z listy posortowanej rosnąco
280 @date 2020-04-02
281 @param pHead wskaźnik na początek listy
282 @return wartość elementu o największej wartości
283 */
284 int elMax(element* pHead) {
285
286     if (pHead) {
287
288         while (pHead->pNext) pHead = pHead->pNext;
289         return pHead->liczba;
290     } else return 0;
291 }
292
293
294 /** Funkcja wylicza średnią wartości elementów z listy
295 @date 2020-04-02
296 @param pHead wskaźnik na początek listy
297 @return wartość średnią
298 */
299 float srednia(element* pHead) {
300
301     if (pHead) {
302
303         float i = 0;
304         float suma = 0;
305
306         while (pHead) {
307
308             i++;
309             suma += pHead->liczba;
310             pHead = pHead->pNext;
311         }
```

```
312
313         return suma / i;
314     }
315     else return 0;
316 }
317
318 /** Funkcja wylicza medianę wartości elementów z listy
319 @date 2020-04-02
320 @param pHead wskaźnik na początek listy
321 @return miedianę elementów
322 */
323 float mediana(element* pHead) {
324
325     if (pHead) {
326
327         float dwa = 2;
328
329         if (policzEl(pHead) % 2) {
330             return wypiszEl(pHead, (policzEl(pHead)/2)+1);
331         }
332         else {
333             return (wypiszEl(pHead, (policzEl(pHead) / 2)) + wypiszEl(pHead,
                    (policzEl(pHead) / 2) + 1)) / dwa;
334         }
335     }
336     else return 0;
337 }
338
339 /** Funkcja generująca plik csv ze statystyką danych z wycztanego pliku
340 @date 2020-04-15
341 @param ilosc_k ilość kolumn
342 @param ilosc_w ilość wierszy
343 */
344 void generator(int ilosc_k, int ilosc_w) {
345
346     FILE* fCsvW;
347     fCsvW = fopen("stats.csv", "w");
348
349     if (!fCsvW) {
350         printf("Blad zapisu!\n");
351         exit(1);
352     }
353
354     fprintf(fCsvW, "%cmax%c, %cmin%c, %cavg%c, %cmed%c\n", '"', '"', '"',
          '"', '"', '"', '"', '"');
355
356     FILE* fCsv;
357     fCsv = fopen("dane.csv", "r");
358
359     if (!fCsv) {
360         printf("Blad odczytu!\n");
361         exit(1);
362     }
```

```
363     else {
364
365         for (int x = 0; x < ilosc_w; x++) {
366
367             int buf;
368             char bufC;
369
370             fscanf(fCsv, "%d", &buf);
371             fscanf(fCsv, "%c", &bufC);
372
373             element* pHead = NULL;
374             pHead = (element*)malloc(sizeof(element));
375             pHead->liczba = buf;
376             pHead->pNext = NULL;
377
378             for (int y = 1; y < ilosc_k; y++) {
379
380                 fscanf(fCsv, "%d", &buf);
381                 if(y != ilosc_k-1) fscanf(fCsv, "%c", &bufC);
382                 dodElem(&pHead, buf);
383             }
384
385             wypisz(pHead);
386             printf("\n");
387
388             fprintf(fCsvW, "%d, ", elMax(pHead));
389             fprintf(fCsvW, "%d, ", pHead->liczba);
390             fprintf(fCsvW, "%.2f, ", srednia(pHead));
391             fprintf(fCsvW, "%.2f\n", mediana(pHead));
392
393             usunListe(&pHead);
394         }
395     }
396
397     fclose(fCsv);
398     fclose(fCsvW);
399 }
400
401 int main(int argc, char* argv[]) {
402
403     int ilosc_k, ilosc_w, zakres_d, zakres_g;
404
405     if (!sprawdzArgumenty(argc, argv, &ilosc_k, &ilosc_w, &zakres_d,
        &zakres_g)) {
406
407         pomoc();
408     }
409
410     else {
411
412         srand((unsigned int)time(NULL));
413         generuj(ilosc_k, ilosc_w, zakres_d, zakres_g);
414         zapiszParamDat(ilosc_k, ilosc_w, zakres_d, zakres_g);
```

```
415             zapiszParamJson(ilosc_k, ilosc_w, zakres_d, zakres_g);
416             generator(ilosc_k, ilosc_w);
417         }
418     }
419
```