

Contents

| | | |
|---|--------------------|---|
| 1 | Basic Test Results | 2 |
| 2 | CheckParenthesis.c | 4 |
| 3 | makefile | 7 |
| 4 | waredb.c | 8 |

1 Basic Test Results

```
1 Running...
2 Opening tar file
3 waredb.c
4 makefile
5 CheckParenthesis.c
6 OK
7 Tar extracted O.K.
8 Checking files...
9 OK
10 Making sure files are not empty...
11 OK
12 Compilation check...
13 Compiling...
14 gcc -Wextra -Wall waredb.c -o waredb
15 gcc -Wextra -Wall CheckParenthesis.c -o CheckParenthesis
16 OK
17 Compilation seems OK! Check if you got warnings!
18
19 =====
20 = Checking coding style =
21 =====
22 waredb.c(38, 9): struct_comment {Doxygen Comment should be provided in front of struct/union def(Product).}
23 waredb.c(77, 5): func_comment_imp {Doxygen Comment should be provided in front of function (yearComp) in impl file.}
24 waredb.c(88, 5): func_comment_imp {Doxygen Comment should be provided in front of function (barcodeCmpFunc) in impl file.}
25 waredb.c(99, 5): func_comment_imp {Doxygen Comment should be provided in front of function (nameCmpFunc) in impl file.}
26 waredb.c(115, 5): func_comment_imp {Doxygen Comment should be provided in front of function (monthcomp) in impl file.}
27 waredb.c(127, 6): func_comment_imp {Doxygen Comment should be provided in front of function (sort) in impl file.}
28 waredb.c(140, 5): func_comment_imp {Doxygen Comment should be provided in front of function (isNumeric) in impl file.}
29 waredb.c(142, 0): indent_tab {Do not use space for indent}
30 waredb.c(143, 0): indent_tab {Do not use space for indent}
31 waredb.c(144, 2): missing_braces {Use brace for even one statement in if/while/for clause}
32 waredb.c(147, 0): indent_tab {Do not use space for indent}
33 waredb.c(149, 0): indent_tab {Do not use space for indent}
34 waredb.c(155, 5): func_comment_imp {Doxygen Comment should be provided in front of function (numOfDigits) in impl file.}
35 waredb.c(174, 5): func_comment_imp {Doxygen Comment should be provided in front of function (checkParameters) in impl file.}
36 waredb.c(174, 36): opeartor_spaces {Provide space after operator ,}
37 waredb.c(179, 23): opeartor_spaces {Provide space after operator ;}
38 waredb.c(219, 6): func_comment_imp {Doxygen Comment should be provided in front of function (printToFile) in impl file.}
39 waredb.c(227, 5): brace_line {The brace should be located in seperate line}
40 waredb.c(231, 3): matching_braces_align {Matching Braces inside of function should be located in the same column}
41 waredb.c(239, 6): func_comment_imp {Doxygen Comment should be provided in front of function (writeToFile) in impl file.}
42 waredb.c(264, 10): fname_case {Do not start function name(FileToStructSent) with uppercase}
43 waredb.c(264, 10): func_comment_imp {Doxygen Comment should be provided in front of function (FileToStructSent) in impl file.}
44 waredb.c(278, 9): opeartor_spaces {Provide space after operator ,}
45 waredb.c(303, 10): func_comment_imp {Doxygen Comment should be provided in front of function (fileToStruct) in impl file.}
46 waredb.c(313, 20): opeartor_spaces {Provide space after operator ,}
47 waredb.c(337, 5): func_comment_imp {Doxygen Comment should be provided in front of function (prodCmp) in impl file.}
48 waredb.c(373, 10): func_comment_imp {Doxygen Comment should be provided in front of function (sortForItems) in impl file.}
49 waredb.c(400, 3): missing_braces {Use brace for even one statement in if/while/for clause}
50 waredb.c(415, 10): func_comment_imp {Doxygen Comment should be provided in front of function (recieved) in impl file.}
51 waredb.c(439, 5): func_comment_imp {Doxygen Comment should be provided in front of function (isThereEnough) in impl file.}
52 waredb.c(441, 18): opeartor_spaces {Provide space after operator ,}
53 waredb.c(441, 24): opeartor_spaces {Provide spaces b/w operator '='}
54 waredb.c(445, 5): indent_block {Indent in the block. token(flag) seems to be located left column of preVIOUS brace}
55 waredb.c(464, 10): func_comment_imp {Doxygen Comment should be provided in front of function (sent) in impl file.}
56 waredb.c(487, 12): opeartor_spaces {Provide space after operator ;}
57 waredb.c(487, 24): opeartor_spaces {Provide space after operator ;}
58 waredb.c(500, 5): brace_line_align {The brace should be located in same column}
59 waredb.c(507, 28): opeartor_spaces {Provide space after operator ,}
```

```

60 waredb.c(519, 10): func_comment_imp {Doxygen Comment should be provided in front of function (clean) in impl file.}
61 waredb.c(525, 5): align_long_cond_list {Incorrect align on condition list '||'. It should be aligned in column 13.}
62 waredb.c(526, 7): align_long_cond_list {Incorrect align on condition list '&&'. It should be aligned in column 13.}
63 waredb.c(550, 5): func_comment_imp {Doxygen Comment should be provided in front of function (main) in impl file.}
64 waredb.c(603, 27): opeartor_spaces {Provide space after operator ,}
65 waredb.c(622, 22): opeartor_spaces {Provide space after operator ,}
66 waredb.c(622, 28): opeartor_spaces {Provide space after operator ,}
67 waredb.c(633, 22): opeartor_spaces {Provide space after operator ,}
68 waredb.c(633, 28): opeartor_spaces {Provide space after operator ,}
69 waredb.c(640, 2): brace_line_align {The brace should be located in same column}
70 waredb.c(650, 3): brace_line_align {The brace should be located in same column}
71 waredb.c(654, 23): opeartor_spaces {Provide space after operator ,}
72 waredb.c(654, 29): opeartor_spaces {Provide space after operator ,}
73 CheckParenthesis.c(31, 6): func_comment_imp {Doxygen Comment should be provided in front of function (closedBrackets) in impl file.}
74 CheckParenthesis.c(40, 2): brace_line_align {The brace should be located in same column}
75 CheckParenthesis.c(43, 2): brace_line_align {The brace should be located in same column}
76 CheckParenthesis.c(46, 2): brace_line_align {The brace should be located in same column}
77 CheckParenthesis.c(57, 37): function_braces {The brace for function definition should be located in start of line}
78 CheckParenthesis.c(65, 1): function_braces {The brace for function definition should be located in same column}
79 CheckParenthesis.c(57, 5): func_comment_imp {Doxygen Comment should be provided in front of function (oneOfOpenBrackets) in impl file.}
80 CheckParenthesis.c(62, 2): brace_line_align {The brace should be located in same column}
81 CheckParenthesis.c(71, 5): func_comment_imp {Doxygen Comment should be provided in front of function (oneOfCloseBrackets) in impl file.}
82 CheckParenthesis.c(77, 2): brace_line_align {The brace should be located in same column}
83 CheckParenthesis.c(86, 5): func_comment_imp {Doxygen Comment should be provided in front of function (main) in impl file.}
84 CheckParenthesis.c(128, 6): brace_line {The brace should be located in seperate line}
85 CheckParenthesis.c(130, 4): matching_braces_align {Matching Braces inside of function should be located in the same column}
86 CheckParenthesis.c(137, 4): brace_line {The brace should be located in seperate line}
87 CheckParenthesis.c(139, 2): matching_braces_align {Matching Braces inside of function should be located in the same column}
88 ** Total Violated Rules      : 11
89 ** Total Errors Occurs      : 66
90 ** Total Violated Files Count: 2

```

2 CheckParenthesis.c

```
1  /*
2   * CheckParenthesis.c
3   *
4   * Created on: Nov 9, 2014
5   * Author: mutazmanaa
6   */
7  /**in this project I couldn't do coding style
8   *in the lab computers I tried to do it many times but
9   *it make problems on the although it is only to deal on comments
10  *and spaces and I don't succeed to do this.
11  */
12  #include <stdio.h>
13  #include <stdlib.h>
14
15  /*
16  *defines
17  */
18  #define MAX_BRACKETS_NUMBER 5000
19  #define NUMBER_OF_ARGS 2
20
21  /*
22  *variables
23  */
24  char closedBrackets(char const openBracket);
25  int oneOfOpenBrackets(char bracket);
26  int oneOfCloseBrackets(char bracket);
27
28  /*
29  *function get a bracket and return the revers one.
30  */
31  char closedBrackets(char const openBracket)
32  {
33
34      char closeBracket = 0;
35
36      if (openBracket == '{')
37      {
38          closeBracket = '}';
39      } else if (openBracket == '(')
40      {
41          closeBracket = ')';
42      } else if (openBracket == '<')
43      {
44          closeBracket = '>';
45      } else if (openBracket == '[')
46      {
47          closeBracket = ']';
48      }
49
50      return closeBracket;
51  }
52
53  /*
54  *get char and check if it is an opening bracket,
55  *return 1 of opening bracket and 0 else
56  */
57  int oneOfOpenBrackets(char bracket) {
58      if (bracket == '{' || bracket == '(' || bracket == '<' || bracket == '[')
59      {
```

```

60         return 1;
61     } else
62     {
63         return 0;
64     }
65 }
66
67 /*
68  *get char and check if it's closing bracket
69  *return 1 if closing one and 0 else
70  */
71 int oneOfCloseBrackets(char bracket)
72 {
73     if (bracket == '}' || bracket == ')' || bracket == '>' || bracket == ']')
74     {
75         return 1;
76     } else
77     {
78         return 0;
79     }
80 }
81
82 /*
83  * my main deal with valid opening of file
84  * scanning the ncharacters and check the brackets
85  */
86 int main(int argc, char* argv[])
87 {
88     FILE* fileStream = fopen(argv[1], "r");
89
90     if (argc != NUMBER_OF_ARGS)
91     {
92
93         printf("please supply a file!\n");
94         printf("usage: CheckParenthesis %s\n", argv[1]);
95
96         exit(EXIT_FAILURE);
97     }
98
99     if (fileStream == 0)
100     {
101         printf("%s: no such file\n", argv[1]);
102         exit(EXIT_FAILURE);
103     }
104
105     int bra = 0;
106     int check = 0;
107     int bracketsCounter = 0;
108     char brackets[MAX_BRACKETS_NUMBER];
109
110     while ((bra = fgetc(fileStream)) != EOF)
111     {
112         check = oneOfOpenBrackets(bra);
113
114         if (check)
115         {
116             brackets[bracketsCounter] = closedBrackets(bra);
117             bracketsCounter++;
118         }
119
120         check = oneOfCloseBrackets(bra);
121
122         if (check)
123         {
124             if (bracketsCounter == 0 || bra != brackets[bracketsCounter - 1])
125             {
126                 printf("Bad structure\n");
127                 exit(EXIT_SUCCESS);

```

-2/-2 Missing documentation
(code='missing_documentation')
course guidelines require the use of
doxygen

```
128         } else {
129             bracketsCounter--;
130         }
131     }
132 }
133
134 if (bracketsCounter != 0)
135 {
136     printf("Bad structure\n");
137 } else {
138     printf("Ok\n");
139 }
140
141 fclose(fileStream);
142 return 0;
143
144 }
```

3 makefile

```
1  waredb:
2      gcc -Wextra -Wall waredb.c -o waredb
3  CheckParenthesis:
4      gcc -Wextra -Wall CheckParenthesis.c -o CheckParenthesis
5  all: waredb CheckParenthesis
6
7  clean:
8      rm -f *.o waredb CheckParentheses
```

0/0 you do not
compile with -Wvla in
your make file.
(code='no_wvla')

4 waredb.c

```
1  /*
2  * waredb.c
3  *
4  * Created on: Nov 14, 2014
5  * Author: mutazmanaa
6  */
7
8  /*
9  * This program is dealing with warehouse , update receieved and sent product,
10 * and remove date expiration products passed.
11 * I use malloc and free cause I free the all memory allocated to the main
12 * structure array and helper strutures allocated during the program.
13 * I assume that when I get correct line in the ware file until the date, I skip
14 * to the next line and deal if it get more parameters, but between the fields I
15 * print siutable error message. LIke ("namer" 1234 5453.5 2014-12 "dsdsad"
16 * ) it is correct.. otherwise errors are catching. that in the future I can
17 * update the only the regex.
18 *
19 */
20
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <string.h>
24 #include <limits.h>
25 #include <math.h>
26 #include <ctype.h>
27
28 #define N 5000
29 #define M 1000
30 #define MAX_NAME_LENGTH 20
31 #define EPSILON 0.001
32 #define SIZE_OF_BARCODE 4
33
34
35 /* struct product contain the sutabile fields and one other flag that help me in
36 * functions.
37 */
38 typedef struct Product
39 {
40     char name[MAX_NAME_LENGTH];
41     int barCode;
42     double quantity;
43     int year;
44     int month;
45     int flag;
46 } Product;
47
48 /*
49 * Declarations to all functions
50 */
51
52
53
54
55 void printToFile(FILE* file, Product* items, int length);
56 void sort(Product *Array, int *length);
57 int isThereEnough(Product *items, Product prod, int len);
58 int checkParameters(Product* array, int lenght);
59 int isNumeric(char *str);
```



```

60  int numOfDigits(int number);
61  Product* sortForItems(Product* array, int* length);
62  Product* fileToStruct(FILE* fp, int* length);
63  void writeToFile(char* file, Product* items, int length);
64  int monthcomp(const void * a, const void * b);
65  int yearComp(const void * a, const void * b);
66  int barcodeCmpFunc(const void * a, const void * b);
67  int nameCmpFunc(const void * a, const void * b);
68
69
70
71
72
73  /*
74   * function related to sort that comparing two years.
75   */
76
77  int yearComp(const void * a, const void * b)
78  {
79      Product *orderA = (Product *) a;
80      Product *orderB = (Product *) b;
81
82      return (orderA->year - orderB->year);
83  }
84
85  /*
86   * function related to sort that comparing two barecodes.
87   */
88  int barcodeCmpFunc(const void * a, const void * b)
89  {
90      Product *orderA = (Product *) a;
91      Product *orderB = (Product *) b;
92
93      return (orderA->barCode - orderB->barCode);
94  }
95
96  /*
97   * function related to sort that compairing two names.
98   */
99  int nameCmpFunc(const void * a, const void * b)
100  {
101      Product *orderA = (Product *) a;
102      Product *orderB = (Product *) b;
103
104      return (strcmp(orderB->name, orderA->name));
105  }
106
107  /*
108   * a main fuction that get file and construct array of structures to save items
109   * on it.
110   */
111
112  /*
113   * function related to sort that comparing two months.
114   */
115  int monthcomp(const void * a, const void * b)
116  {
117      Product *orderA = (Product *) a;
118      Product *orderB = (Product *) b;
119
120      return (orderA->month - orderB->month);
121  }
122  /*
123   * sort function that sort all items : first by barcode, then by expire date
124   * and then by name
125   *
126   */
127  void sort(Product *Array, int* length)

```

```

128 {
129
130     qsort(Array, *length, sizeof(Product), monthcomp);
131     qsort(Array, *length, sizeof(Product), yearComp);
132     qsort(Array, *length, sizeof(Product), nameCmpFunc);
133     qsort(Array, *length, sizeof(Product), barcodeCmpFunc);
134 }
135
136 /*
137  * check if a string is only number and not contain letters or other characters.
138  *
139  */
140 int isNumeric(char *str)
141 {
142     while(*str)
143     {
144         if(!isdigit(*str))
145             return 0;
146         str++;
147     }
148
149     return 1;
150 }
151
152 /*
153  * function get a number and return number of digits.
154  */
155 int numOfDigits(int number)
156 {
157     if (number < 0)
158     {
159         return numOfDigits((number == INT_MIN) ? INT_MAX : -number);
160     }
161     if (number < 10)
162     {
163         return 1;
164     }
165     return 1 + numOfDigits (number / 10);
166 }
167
168
169
170 /*
171  * function that get array of structure and check if the parameters inside are
172  * valid.
173  */
174 int checkParameters(Product* array ,int length)
175 {
176
177     int i;
178     char tempChar[SIZE_OF_BARCODE];
179     for(i = 0; i < length;i++)
180     {
181         sprintf(tempChar, "%d", (array[i].barCode));
182         if(strlen(array[i].name) > MAX_NAME_LENGTH)
183         {
184
185             printf("unknown file format\n");
186             return 1;
187         }
188         else if(!isNumeric(tempChar) || numOfDigits(array[i].barCode)
189             !=4 || array[i].barCode < 0)
190         {
191
192             printf("unknown file format\n");
193             return 1;
194         }
195     }

```

```

196
197     else if(array[i].quantity < 0)
198     {
199
200         printf("unknown file format\n");
201         return 1;
202     }
203     else if (array[i].month < 0 || array[i].month > 12 || array[i].year < 0)
204     {
205
206         printf("unknown file format\n");
207         return 1;
208     }
209 }
210
211 }
212
213 return 0;
214 }
215
216 /*
217  * print to file.
218  */
219 void printToFile(FILE* file, Product* items, int length)
220 {
221     int i;
222     for (i = 0; i < length; i++)
223     {
224         if (items[i].quantity == 0 || items[i].year == -1)
225         {
226             continue;
227         } else {
228             fprintf(file, "%s\t%d\t%.3lf\t%d-%d\n", (items[i].name),
229                 (items[i].barCode), (items[i].quantity), (items[i].year),
230                 (items[i].month));
231         }
232     }
233 }
234 /*
235  * function the get a file and array and write the items one by as lines in
236  * the file.
237  */
238
239 void writeToFile(char* file, Product* items, int length)
240 {
241     FILE* fpData = fopen(file, "w");
242     if(fpData != NULL)
243     {
244         printToFile(fpData, items, length);
245     }
246
247     else
248     {
249         printf("Permission denied\n");
250         exit(1);
251     }
252
253     fclose(fpData);
254
255 }
256
257
258
259 /*
260  * a main fuction that get file and construct array of structures to save items
261  * on it.
262  */
263

```

```

264 Product* FileToStructSent(FILE* fp, int* length)
265 {
266
267     Product* items = malloc(N * sizeof(*items));
268     int i = 0;
269
270
271
272     char line[M];
273     int lineElementsCount;
274
275     while (fgets(line, M, fp) != NULL )
276     {
277         lineElementsCount = sscanf(line, "%d\t%lf"
278                                     , &(items[i].barCode), &(items[i].quantity));
279
280
281         if( lineElementsCount == 2)
282         {
283             items[i].flag = 0;
284             i++;
285             continue;
286         }
287         else
288         {
289
290             printf("unknown file format\n");
291             exit(1);
292         }
293     }
294
295     *length = i;
296
297     return items;
298 }
299
300 //-----
301
302 Product* fileToStruct(FILE* fp, int* length)
303 {
304
305     Product* items = malloc(N * sizeof(*items));
306     int i = 0;
307     char line[M];
308     int lineElementsCount;
309     while (fgets(line, M, fp) != NULL )
310     {
311         lineElementsCount = sscanf(line, "%[^\t]*c\t%d\t%lf\t%d-%d\n",
312                                     (items[i].name), &(items[i].barCode), &(items[i].quantity),
313                                     &(items[i].year)
314                                     , &(items[i].month));
315
316
317         if( lineElementsCount == 5)
318         {
319             items[i].flag = 0;
320             i++;
321             continue;
322         }
323         else
324         {
325
326             printf("unknown file format\n");
327             exit(1);
328         }
329     }
330
331     *length = i;

```

-2/-2 Missing documentation
(code='missing_documentation')

```

332     return items;
333 }
334
335 //-----
336
337 int prodCmp(Product prod1, Product prod2)
338 {
339     //1 means prod1 is bigger
340     //-1 means prod2 is bigger
341     //0 means they are equal
342     if (prod1.barCode > prod2.barCode)
343     {
344         return 1;
345     }
346     if (prod1.barCode < prod2.barCode)
347     {
348         return -1;
349     }
350     if (prod1.year > prod2.year)
351     {
352         return 1;
353     }
354     if (prod1.year < prod2.year)
355     {
356         return -1;
357     }
358     if (prod1.month > prod2.month)
359     {
360         return 1;
361     }
362     if (prod1.month < prod2.month)
363     {
364         return -1;
365     }
366     return strcmp(prod1.name, prod2.name);
367 }
368
369
370 /*
371  * simply..sort the items by barcode and then expire date and then by neme.
372  */
373 Product* sortForItems(Product* array, int* length)
374 {
375
376     sort(array, length);
377
378
379     int i, j;
380     int newLength = *length;
381     for (i = 0; i < *length - 1; i++)
382     {
383         if (prodCmp(array[i], array[i + 1]) == 0)
384         {
385             array[i].flag = 1;
386             array[i + 1].quantity += array[i].quantity;
387             newLength--;
388         }
389     }
390
391     Product* items = malloc(newLength * sizeof(*items));
392     if (!items)
393     {
394         return NULL ;
395     }
396
397     j = 0;
398     for (i = 0; i < *length; i++)
399     {

```

```

400         if (array[i].flag == 1)
401             continue;
402         items[j++] = array[i];
403     }
404 }
405
406 *length = newLength;
407
408 return items;
409 }
410
411 /*
412  * function that get items array and recieve items to our ware by file
413  * and update our ware(items array).
414  */
415 Product* recieved(Product* items, int* length, FILE* recievedFile)
416 {
417     int receivedLength;
418     Product* receivedItems = fileToStruct(recievedFile, &receivedLength);
419     Product *allItems = malloc((receivedLength + *length) * sizeof(*allItems));
420
421     int i, j = 0;
422     for (i = 0; i < *length; ++i)
423     {
424         allItems[i] = items[i];
425     }
426     *length += receivedLength;
427     for (; i < *length; ++i)
428     {
429         allItems[i] = receivedItems[j++];
430     }
431     Product* recievedTemp = sortForItems(allItems, length);
432     return recievedTemp;
433 }
434
435 /*
436  * function that get items array and another item to sent
437  * and check if there is enough quantity to send.
438  */
439 int isThereEnough(Product *items, Product prod, int len)
440 {
441     int quantity = 0, flag = 0, i;
442     for (i = 0; i < len; ++i)
443     {
444         if (items[i].barCode == prod.barCode)
445         {
446             flag = 1;
447             quantity += items[i].quantity;
448         }
449     }
450     if (flag && (quantity + EPSILON >= prod.quantity))
451     {
452         return 1;
453     }
454
455     return 0;
456 }
457
458 /*
459  *function that get our ware and an order to sent..if our ware have enough
460  *quantity we sent it and update our ware and if not printed error message.
461  */
462
463
464 Product* sent(Product* items, int* length, FILE* sentFile)
465 {
466     int neededLength;
467     Product* neededItems = FileToStructSent(sentFile, &neededLength);

```

```

468     if( checkParameters(neededItems, *length))
469     {
470         exit(1);
471     }
472
473
474     int i;
475     for (i = 0; i < neededLength; ++i)
476     {
477
478         if (!isThereEnough(items, neededItems[i], *length))
479         {
480             printf("not enough items in warehouse\n");
481             return items;
482         }
483     }
484     int j;
485     for (i = 0; i < neededLength; ++i)
486     {
487         for(j = 0; j < *length; j++)
488         {
489
490             if (items[j].barCode == neededItems[i].barCode)
491             {
492                 items[j].quantity -= neededItems[i].quantity;
493                 if (!(items[j].quantity > 0))
494                 {
495                     neededItems[i].quantity = -items[j].quantity;
496                     items[j].quantity = 0;
497
498                     continue;
499                 } else
500                 {
501                     break;
502                 }
503             }
504         }
505     }
506
507     items = sortForItems(items, length);
508     free(neededItems);
509
510     return items;
511 }
512
513
514
515 /*
516  * clean function that remove items from our array that his date is expired.
517  */
518
519 Product* clean(Product* items, int* length, int year, int month)
520 {
521     int i = 0;
522     for (i = 0; i < *length; i++)
523     {
524         if (items[i].quantity == 0 || year > items[i].year
525             || (month != 0 && month > items[i].month
526                 && year == items[i].year))
527         {
528             items[i].year = -1;
529         }
530
531         else if (year == 0)
532         {
533             return items;
534         }
535     }

```

```

536     }
537
538     return items;
539 }
540
541
542
543 /* my main is deal with the input of the use and check what functions to do
544 * related to my input.
545 * note: I can do the check of files validity in functin and to return files but
546 *I prepered to do it in the main cause I use it once and I like to see my main
547 *I do any thing.
548 *
549 */
550 int main(int argc, char *argv[])
551 {
552     if (argc != 4)
553     {
554         printf("USAGE: waredb %s %s %s \n", argv[1], argv[2], argv[3]);
555         return 1;
556     }
557
558     FILE *fpData = fopen(argv[1], "r");
559     FILE *fpOrder;
560
561     if (fpData == NULL )
562     {
563         printf("%s: no such file\n", argv[1]);
564         return 1;
565     }
566
567     if (strcmp(argv[2], "clean") != 0)
568     {
569         fpOrder = fopen(argv[3], "r");
570         if (fpOrder == NULL )
571         {
572             printf("%s: no such file\n", argv[3]);
573             return 1;
574         }
575
576         if (fpData == NULL && fpOrder == NULL )
577         {
578             printf("%s: no such file\n", argv[1]);
579             return 1;
580         }
581     }
582
583     int length;
584     Product* items = fileToStruct(fpData, &length);
585
586     /*
587     * can I sent my file to be structured on array of structure and check
588     * parameters.
589     */
590     if( checkParameters(items,length))
591

```



```

604     {
605         return 0;
606     }
607
608     /*
609     * sort for our ware.
610     */
611     items = sortForItems(items, &length);
612
613     fclose(fpData);
614
615     /*
616     * deal with sent input.
617     */
618     if (strcmp(argv[2], "sent") == 0)
619     {
620         items = sent(items, &length, fpOrder);
621
622         writeToFile(argv[1], items, length);
623     }
624
625
626     /*
627     * deal with recieved input.
628     */
629     else if (strcmp(argv[2], "received") == 0)
630     {
631         items = recieved(items, &length, fpOrder);
632
633         writeToFile(argv[1], items, length);
634     }
635
636     /*
637     * deal with clean input
638     */
639     } else if (strcmp(argv[2], "clean") == 0)
640     {
641         int year, month;
642         if (sscanf(argv[3], "%d-%d", &year, &month) != 2)
643
644         {
645
646             printf("USAGE: waredb %s %s %s \n", argv[1], argv[2], argv[3]);
647             return 0;
648
649         } else
650         {
651
652             items = clean(items, &length, year, month);
653
654             writeToFile(argv[1], items, length);
655         }
656     }
657
658     /*
659     * close files and free memory.
660     */
661     if (strcmp(argv[2], "clean") != 0)
662     {
663         fclose(fpOrder);
664     }
665
666     free(items);
667
668
669     return 0;
670 }

```

-3/-3 Function is too long. You should break it to sub-functions. (code=function_too_long')