Contents

1 Basic Test Results

```
Running...
Opening tar file
waredb.c
makefile
CheckParenthesis.c
OK
Tar extracted O.K.
Checking files...
OK
Making sure files are not empty...
OK
Compilation check...
Compiling...
gcc -Wextra -Wall waredb.c -o waredb
gcc -Wextra -Wall CheckParenthesis.c -o CheckParenthesis
OK
Compilation seems OK! Check if you got warnings!

========================
= Checking coding style =
========================
waredb.c(38, 9):  struct_comment {Doxygen Comment should be provided in front of struct/union def(Product).}
waredb.c(77, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (yearComp) in impl file.}
waredb.c(88, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (barcodeCmpFunc) in impl file.}
waredb.c(99, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (nameCmpFunc) in impl file.}
waredb.c(115, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (monthcomp) in impl file.}
waredb.c(127, 6):  func_comment_imp {Doxygen Comment should be provided in front of function (sort) in impl file.}
waredb.c(140, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (isNumeric) in impl file.}
waredb.c(142, 0):  indent_tab {Do not use space for indent}
waredb.c(143, 0):  indent_tab {Do not use space for indent}
waredb.c(144, 2):  missing_braces {Use brace for even one statement in if/while/for clause}
waredb.c(147, 0):  indent_tab {Do not use space for indent}
waredb.c(149, 0):  indent_tab {Do not use space for indent}
waredb.c(155, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (numOfDigits) in impl file.}
waredb.c(174, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (checkParameters) in impl file.
waredb.c(174, 36):  opeartor_spaces {Provide space after operator ,}
waredb.c(179, 23):  opeartor_spaces {Provide space after operator ;}
waredb.c(219, 6):  func_comment_imp {Doxygen Comment should be provided in front of function (printToFile) in impl file.}
waredb.c(227, 5):  brace_line {The brace should be located in seperate line}
waredb.c(231, 3):  matching_braces_align {Matching Braces inside of function should be located in the same column}
waredb.c(239, 6):  func_comment_imp {Doxygen Comment should be provided in front of function (writeToFile) in impl file.}
waredb.c(264, 10):  fname_case {Do not start function name(FileToStructSent) with uppercase}
waredb.c(264, 10):  func_comment_imp {Doxygen Comment should be provided in front of function (FileToStructSent) in impl fil
waredb.c(278, 9):  opeartor_spaces {Provide space after operator ,}
waredb.c(303, 10):  func_comment_imp {Doxygen Comment should be provided in front of function (fileToStruct) in impl file.}
waredb.c(313, 20):  opeartor_spaces {Provide space after operator ,}
waredb.c(337, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (prodCmp) in impl file.}
waredb.c(373, 10):  func_comment_imp {Doxygen Comment should be provided in front of function (sortForItems) in impl file.}
waredb.c(400, 3):  missing_braces {Use brace for even one statement in if/while/for clause}
waredb.c(415, 10):  func_comment_imp {Doxygen Comment should be provided in front of function (recieved) in impl file.}
waredb.c(439, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (isThereEnough) in impl file.}
waredb.c(441, 18):  opeartor_spaces {Provide space after operator ,}
waredb.c(441, 24):  opeartor_spaces {Provide spaces b/w operator '='}
waredb.c(445, 5):  indent_block {Indent in the block. token(flag) seems to be located left column of previsous brace}
waredb.c(464, 10):  func_comment_imp {Doxygen Comment should be provided in front of function (sent) in impl file.}
waredb.c(487, 12):  opeartor_spaces {Provide space after operator ;}
waredb.c(487, 24):  opeartor_spaces {Provide space after operator ;}
waredb.c(500, 5):  brace_line_align {The brace should be located in same column}
waredb.c(507, 28):  opeartor_spaces {Provide space after operator ,}
```

```
waredb.c(519, 10):  func_comment_imp {Doxygen Comment should be provided in front of function (clean) in impl file.}
waredb.c(525, 5):  align_long_cond_list {Incorrect align on condition list '||'. It should be aligned in column 13.}
waredb.c(526, 7):  align_long_cond_list {Incorrect align on condition list '&&'. It should be aligned in column 13.}
waredb.c(550, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (main) in impl file.}
waredb.c(603, 27):  opeartor_spaces {Provide space after operator ,}
waredb.c(622, 22):  opeartor_spaces {Provide space after operator ,}
waredb.c(622, 28):  opeartor_spaces {Provide space after operator ,}
waredb.c(633, 22):  opeartor_spaces {Provide space after operator ,}
waredb.c(633, 28):  opeartor_spaces {Provide space after operator ,}
waredb.c(640, 2):  brace_line_align {The brace should be located in same column}
waredb.c(650, 3):  brace_line_align {The brace should be located in same column}
waredb.c(654, 23):  opeartor_spaces {Provide space after operator ,}
waredb.c(654, 29):  opeartor_spaces {Provide space after operator ,}
CheckParenthesis.c(31, 6):  func_comment_imp {Doxygen Comment should be provided in front of function (closedBrackets) in im
CheckParenthesis.c(40, 2):  brace_line_align {The brace should be located in same column}
CheckParenthesis.c(43, 2):  brace_line_align {The brace should be located in same column}
CheckParenthesis.c(46, 2):  brace_line_align {The brace should be located in same column}
CheckParenthesis.c(57, 37):  function_braces {The brace for function definition should be located in start of line}
CheckParenthesis.c(65, 1):  function_braces {The brace for function definition should be located in same column}
CheckParenthesis.c(57, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (oneOfOpenBrackets) in
CheckParenthesis.c(62, 2):  brace_line_align {The brace should be located in same column}
CheckParenthesis.c(71, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (oneOfCloseBrackets) i
CheckParenthesis.c(77, 2):  brace_line_align {The brace should be located in same column}
CheckParenthesis.c(86, 5):  func_comment_imp {Doxygen Comment should be provided in front of function (main) in impl file.}
CheckParenthesis.c(128, 6):  brace_line {The brace should be located in seperate line}
CheckParenthesis.c(130, 4):  matching_braces_align {Matching Braces inside of function should be located in the same column}
CheckParenthesis.c(137, 4):  brace_line {The brace should be located in seperate line}
CheckParenthesis.c(139, 2):  matching_braces_align {Matching Braces inside of function should be located in the same column}
 ** Total Violated Rules      : 11
 ** Total Errors Occurs       : 66
 ** Total Violated Files Count: 2
```

2 CheckParenthesis.c

```c
/*
 * CheckParenthesis.c
 *
 *  Created on: Nov 9, 2014
 *      Author: mutazmanaa
 */
/**in this project I couldn't do coding style
 *in the lab computers I tried to do it many times but
 *it make problems on the although it is only to deal on comments
 *and spaces and I don't succeed to do this.
 */
#include <stdio.h>
#include <stdlib.h>

/*
*defines
*/
#define MAX_BRACKETS_NUMBER 5000
#define NUMBER_OF_ARGS 2

/*
*variables
*/
char closedBrackets(char const openBracket);
int oneOfOpenBrackets(char bracket);
int oneOfCloseBrackets(char bracket);

/*
*function get a bracket and return the revers one.
*/
char closedBrackets(char const openBracket)
{

    char closeBracket = 0;

    if (openBracket == '{')
    {
        closeBracket = '}';
    } else if (openBracket == '(')
    {
        closeBracket = ')';
    } else if (openBracket == '<')
    {
        closeBracket = '>';
    } else if (openBracket == '[')
    {
        closeBracket = ']';
    }

    return closeBracket;
}

/*
*get char and check if it is an opening bracket,
*return 1 of opening bracket and 0 else
*/
int oneOfOpenBrackets(char bracket) {
    if (bracket == '{' || bracket == '(' || bracket == '<' || bracket == '[')
    {
```

```c
        return 1;
    } else
    {
        return 0;
    }
}

/*
*get char and check if it's closing bracket
*reurn 1 if closing one and 0 else
*/
int oneOfCloseBrackets(char bracket)
{
    if (bracket == '}' || bracket == ')' || bracket == '>' || bracket == ']')
    {
        return 1;
    } else
    {
        return 0;
    }
}

/*
* my main deal with valid opening of file
*scaning the ncharecters and check the brackets
*/
int main(int argc, char* argv[])
{
    FILE* fileStream = fopen(argv[1], "r");

    if (argc != NUMBER_OF_ARGS)
    {

        printf("please supply a file!\n");
        printf("usage: CheckParenthesis %s\n", argv[1]);

        exit(EXIT_FAILURE);
    }

    if (fileStream == 0)
    {
        printf("%s:no such file\n", argv[1]);
        exit(EXIT_FAILURE);
    }

    int bra = 0;
    int check = 0;
    int bracketsCounter = 0;
    char brackets[MAX_BRACKETS_NUMBER];

    while ((bra = fgetc(fileStream)) != EOF)
    {
        check = oneOfOpenBrackets(bra);

        if (check)
        {
            brackets[bracketsCounter] = closedBrackets(bra);
            bracketsCounter++;
        }

        check = oneOfCloseBrackets(bra);

        if (check)
        {
            if (bracketsCounter == 0 || bra != brackets[bracketsCounter - 1])
            {
                printf("Bad structure\n");
                exit(EXIT_SUCCESS);
```

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

```
            } else {
                bracketsCounter--;
            }
        }
    }

    if (bracketsCounter != 0)
    {
        printf("Bad structure\n");
    } else {
        printf("Ok\n");
    }

    fclose(fileStream);
    return 0;

}
```

3 makefile

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

```
waredb:
    gcc -Wextra -Wall waredb.c -o waredb
CheckParenthesis:
    gcc -Wextra -Wall CheckParenthesis.c -o CheckParenthesis
all: waredb CheckParenthesis

clean:
    rm -f *.o waredb CheckParentheses
```

4 waredb.c

```c
/*
 * waredb.c
 *
 *  Created on: Nov 14, 2014
 *      Author: mutazmanaa
 */

/*
 * This program is dealing with warehouse , update receieved and sent product,
 * and remove date expiration products passed.
 * I use malloc and free cause I free the all memory allocated to the main
 * structure array and helper strutures allocated during the program.
 * I assume that when I get correct line in the ware file until the date, I skip
 * to the next line and deal if it get more parameters, but between the fields I
 * print siutable error message. LIke ("namer"    1234    5453.5    2014-12    "dsdsad"
 * ) it is correct.. otherwise errors are catching. that in the future I can
 * update the only the regex.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <math.h>
#include <ctype.h>

#define N 5000
#define M 1000
#define MAX_NAME_LENGTH 20
#define EPSILON 0.001
#define SIZE_OF_BARCODE 4


/* struct product contain the sutabile fields and one other flag that help me in
 * functions.
 */
typedef struct Product
{
    char name[MAX_NAME_LENGTH];
    int barCode;
    double quantity;
    int year;
    int month;
    int flag;
} Product;

/*
 * Declarations to all functions
 */




void printToFile(FILE* file, Product* items, int length);
void sort(Product *Array, int *length);
int isThereEnough(Product *items, Product prod, int len);
int checkParameters(Product* array, int lenght);
int isNumeric(char *str);
```

```c
int numOfDigits(int number);
Product* sortForItems(Product* array, int* length);
Product* fileToStruct(FILE* fp, int* length);
void writeToFile(char* file, Product* items, int length);
int monthcomp(const void * a, const void * b);
int yearComp(const void * a, const void * b);
int barcodeCmpFunc(const void * a, const void * b);
int nameCmpFunc(const void * a, const void * b);




/*
 * function related to sort that comparing two years.
 */

int yearComp(const void * a, const void * b)
{
    Product *orderA = (Product *) a;
    Product *orderB = (Product *) b;

    return (orderA->year - orderB->year);
}

/*
 * function related to sort that comparing two barecodes.
 */
int barcodeCmpFunc(const void * a, const void * b)
{
    Product *orderA = (Product *) a;
    Product *orderB = (Product *) b;

    return (orderA->barCode - orderB->barCode);
}

/*
 * function related to sort that compairing two names.
 */
int nameCmpFunc(const void * a, const void * b)
{
    Product *orderA = (Product *) a;
    Product *orderB = (Product *) b;

    return (strcmp(orderB->name, orderA->name));
}

/*
 * a main fuction that get file and construct array of structures to save items
 * on it.
 */

/*
 * function related to sort that comparing two months.
 */
int monthcomp(const void * a, const void * b)
{
    Product *orderA = (Product *) a;
    Product *orderB = (Product *) b;

    return (orderA->month - orderB->month);
}
/*
 * sort function that sort all items : first by barcode,then by expire date
 * and then by name
 *
 */
void sort(Product *Array, int* length)
```

```c
{

    qsort(Array, *length, sizeof(Product), monthcomp);
    qsort(Array, *length, sizeof(Product), yearComp);
    qsort(Array, *length, sizeof(Product), nameCmpFunc);
    qsort(Array, *length, sizeof(Product), barcodeCmpFunc);
}

/*
 * check if  a strin is only number and not contain letters or other charecters.
 *
 */
int isNumeric(char *str)
{
  while(*str)
  {
    if(!isdigit(*str))
      return 0;
    str++;
  }

  return 1;
}

/*
 * function get a number and return number of digits.
 */
int numOfDigits(int number)
{
        if (number < 0)
        {
            return numOfDigits((number == INT_MIN) ? INT_MAX : -number);
        }
        if (number < 10)
        {
            return 1;
        }
        return 1 + numOfDigits (number / 10);

}


/*
 * function that get array of strucure and check if the parameters inside are
 * valid.
 */
int checkParameters(Product* array ,int length)
{

    int i;
    char tempChar[SIZE_OF_BARCODE];
    for(i = 0; i < length;i++)
    {
        sprintf(tempChar, "%d", (array[i].barCode));
        if(strlen(array[i].name) > MAX_NAME_LENGTH)
        {

            printf("unknown file format\n");
            return 1;
        }
        else if(!isNumeric(tempChar) || numOfDigits(array[i].barCode)
                !=4 || array[i].barCode < 0)
        {


            printf("unknown file format\n");
            return 1;
        }
```

```c

        else if(array[i].quantity < 0)
        {

            printf("unknown file format\n");
            return 1;
        }
        else if (array[i].month < 0 || array[i].month > 12 || array[i].year < 0)

        {

            printf("unknown file format\n");
            return 1;
        }


    }
    return 0;
}

/*
 * print to file.
 */
void printToFile(FILE* file, Product* items, int length)
{
    int i;
    for (i = 0; i < length; i++)
    {
        if (items[i].quantity == 0 || items[i].year == -1)
        {
            continue;
        } else {
            fprintf(file, "%s\t%d\t%.3lf\t%d-%d\n", (items[i].name),
                    (items[i].barCode), (items[i].quantity), (items[i].year),
                    (items[i].month));
        }
    }
}
/*
 * function the get a file and array and write the items one by as lines in
 * the file.
 */

void writeToFile(char* file, Product* items, int length)
{
    FILE* fpData = fopen(file, "w");
    if(fpData != NULL)
    {
        printToFile(fpData, items, length);
    }

    else
    {
        printf("Permission denied\n");
        exit(1);
    }

    fclose(fpData);


}


/*
 * a main fuction that get file and construct array of structures to save items
 * on it.
 */

```

```c
Product* FileToStructSent(FILE* fp, int* length)
{

    Product* items = malloc(N * sizeof(*items));
    int i = 0;



    char line[M];
    int lineElementsCount;

    while (fgets(line, M, fp) != NULL )
    {
        lineElementsCount = sscanf(line, "%d\t%lf"
                                ,&(items[i].barCode), &(items[i].quantity));


            if( lineElementsCount == 2)
            {
                items[i].flag = 0;
                i++;
                continue;
            }
            else
            {

                printf("unknown file format\n");
                exit(1);
            }
    }

    *length = i;

    return items;

}

//-----------------------------------------------------------------------------

Product* fileToStruct(FILE* fp, int* length)
{

    Product* items = malloc(N * sizeof(*items));
    int i = 0;
    char line[M];
    int lineElementsCount;
    while (fgets(line, M, fp) != NULL )
    {
        lineElementsCount = sscanf(line, "%[^\t]%*c\t%d\t%lf\t%d-%d\n",
                (items[i].name),&(items[i].barCode), &(items[i].quantity),
                &(items[i].year)
                , &(items[i].month));


        if( lineElementsCount == 5)
        {
            items[i].flag = 0;
            i++;
            continue;
        }
        else
        {
            printf("unknown file format\n");
            exit(1);
        }
    }

    *length = i;
```

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

```
    return items;

}

//-----------------------------------------------------------------------------
int prodCmp(Product prod1, Product prod2)
{
//1 means prod1 is bigger
//-1 means prod2 is bigger
//0 means they are equal
    if (prod1.barCode > prod2.barCode)
    {
        return 1;
    }
    if (prod1.barCode < prod2.barCode)
    {
        return -1;
    }
    if (prod1.year > prod2.year)
    {
        return 1;
    }
    if (prod1.year < prod2.year)
    {
        return -1;
    }
    if (prod1.month > prod2.month)
    {
        return 1;
    }
    if (prod1.month < prod2.month)
    {
        return -1;
    }
    return strcmp(prod1.name, prod2.name);
}


/*
 * simply..sort the items by barcode and then expire date and then by neme.
 */
Product* sortForItems(Product* array, int* length)
{

    sort(array, length);


    int i, j;
    int newLength = *length;
    for (i = 0; i < *length - 1; i++)
    {
        if (prodCmp(array[i], array[i + 1]) == 0)
        {
            array[i].flag = 1;
            array[i + 1].quantity += array[i].quantity;
            newLength--;
        }
    }

    Product* items = malloc(newLength * sizeof(*items));
    if (!items)
    {
        return NULL ;
    }

    j = 0;
    for (i = 0; i < *length; i++)
    {
```

```c
        if (array[i].flag == 1)
            continue;
        items[j++] = array[i];

    }

    *length = newLength;

    return items;
}

/*
 * function that get items array and recieve items to our ware by file
 * and update our ware(items array).
 */
Product* recieved(Product* items, int* length, FILE* recievedFile)
{
    int receivedLength;
    Product* receivedItems = fileToStruct(recievedFile, &receivedLength);
    Product *allItems = malloc((receivedLength + *length) * sizeof(*allItems));

    int i, j = 0;
    for (i = 0; i < *length; ++i)
    {
        allItems[i] = items[i];
    }
    *length += receivedLength;
    for (; i < *length; ++i)
    {
        allItems[i] = receivedItems[j++];
    }
    Product* recievedTemp = sortForItems(allItems, length);
    return recievedTemp;
}

/*
 * function that get items array and another item to sent
 * and check if there is enough quantity to send.
 */
int isThereEnough(Product *items, Product prod, int len)
{
    int quantity = 0,flag =0, i;
    for (i = 0; i < len; ++i)
    {
        if (items[i].barCode == prod.barCode)
        {    flag = 1;
            quantity += items[i].quantity;

        }
    }
    if (flag && (quantity + EPSILON >= prod.quantity))
    {
        return 1;
    }

    return 0;
}

/*
 *function that get our ware and an order to sent..if our ware have enough
 *quantity we sent it and update our ware and if not printed error message.
 */


Product* sent(Product* items, int* length, FILE* sentFile)
{
    int neededLength;
    Product* neededItems = FileToStructSent(sentFile, &neededLength);
```

```c
    if( checkParameters(neededItems, *length))
    {
            exit(1);
    }


    int i;
    for (i = 0; i < neededLength; ++i)
    {

        if (!isThereEnough(items, neededItems[i], *length))
        {
            printf("not enough items in warehouse\n");
            return items;
        }
    }
    int j;
    for (i = 0; i < neededLength; ++i)
    {
        for(j = 0;j < *length;j++)
        {

            if (items[j].barCode == neededItems[i].barCode)
            {
                items[j].quantity -= neededItems[i].quantity;
                if (!(items[j].quantity > 0))
                {
                    neededItems[i].quantity = -items[j].quantity;
                    items[j].quantity = 0;

                    continue;
                } else
                {
                    break;
                }
            }
        }
    }

    items = sortForItems(items,length);
    free(neededItems);

    return items;
}



/*
 * clean function that remove items from our array that his date is expired.
 */

Product* clean(Product* items, int* length, int year, int month)
{
    int i = 0;
    for (i = 0; i < *length; i++)
    {
        if (items[i].quantity == 0 || year > items[i].year
                || (month != 0 && month > items[i].month
                        && year == items[i].year))
        {
            items[i].year = -1;
        }

        else if (year == 0)
        {
            return items;
        }

```

```c
    }

    return items;

}


/* my main is deal with the input of the use and check what functions to do
 * related to my input.
 * note: I can do the check of files validity in functin and to return files but
 *I prefered to do it in the main cause I use it once and I like to see my main
 *I do any thing.
 *
 */
int main(int argc, char *argv[])

{
    if (argc != 4)
    {
        printf("USAGE: waredb %s %s %s \n", argv[1], argv[2], argv[3]);

        return 1;
    }



    FILE *fpData = fopen(argv[1], "r");
    FILE *fpOrder;


    if (fpData == NULL )
    {

        printf("%s: no such file\n", argv[1]);
        return 1;
    }


    if (strcmp(argv[2], "clean") != 0)
    {
        fpOrder = fopen(argv[3], "r");
        if (fpOrder == NULL )
        {

            printf("%s: no such file\n", argv[3]);
            return 1;
        }

        if (fpData == NULL && fpOrder == NULL )
        {

            printf("%s: no such file\n", argv[1]);
            return 1;
        }

    }

/*
 * can I sent my file to be structured on array of structure and check
 * parameters.
 */
    int length;
    Product* items = fileToStruct(fpData, &length);

/*
 * check parameters validity
 */
    if( checkParameters(items,length))
```

```
    {
            return 0;
    }

/*
 * sort for our ware.
 */
    items = sortForItems(items, &length);

    fclose(fpData);

/*
 * deal with sent input.
 */
    if (strcmp(argv[2], "sent") == 0)
    {
        items = sent(items, &length, fpOrder);

        writeToFile(argv[1],items,length);

    }

/*
 * deal with recieved input.
 */
    else if (strcmp(argv[2], "received") == 0)
    {
        items = recieved(items, &length, fpOrder);

        writeToFile(argv[1],items,length);


/*
 * deal with clean input
 */
    } else if (strcmp(argv[2], "clean") == 0)
    {
        int year, month;
        if (sscanf(argv[3], "%d-%d", &year, &month) != 2)

        {

            printf("USAGE: waredb %s %s %s \n", argv[1], argv[2], argv[3]);
            return 0;

        }else
        {

            items = clean(items, &length, year, month);

            writeToFile(argv[1],items,length);
        }
    }

/*
 * close files and free memory.
 */
    if (strcmp(argv[2], "clean") != 0)
    {
        fclose(fpOrder);
    }

    free(items);


    return 0;
}
```

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