```
/*-------------------------------------------------------------------------+
File Name: functions.c
+-------------------------------------------------------------------------+

+-------------------------------------------------------------------------+
   Programming III COP4338
   Author: Daniel Gonzalez P#4926400
   assignment 5: Date Validate / Format
   Date: 11/08/2016 ELLECTION DAY!!!!

  program description
     Input: Accept input for the first program via the command-line arguments.
            Input will be the number of valid entries to be redirected from
            the dates input file (dates.dat). A zero indicates to input all
            entries from the dates input file. Your program should validate
            (day, month & year - see page 111 for validation ideas) and skip
            corrupt dates in the dates.dat file (see page 159 for scanning
            ideas). This validated input will then be piped out to the
            second program.
            The second program will accept these validated dates in the
            month/day/year format and convert each of them to the day,
            abbreviated month & year format – both exhibited above. The
            abbreviated month should consist of the first three letters of
            the month, capitalized. These converted results will be redirected
            to the output file (output.dat), followed by a copy of the
            original (dates.dat) data.

     Output: Generates an output file (output.dat) that contains a
             converted list of dates in day, abbreviated month & year
             format (i.e. 1 JAN 1900), followed by the original list of
             dates in month/day/year format (i.e. 1/1/1900). This output file
             will be the result of appending the input file (dates.dat), which
             is accessed by the first program, with the result output file
             (output.dat), generated by the second program.


  +-------------------------------------------------------------------------+
  | I Daniel Gonzalez #4926400 hereby certify that this collective work is  |
  | my own and none of it is the work of any other person or entity.        |
  +-------------------------------------------------------------------------+


  how to compile and execute:
    1.Open the terminal
        Go to the program folder that contains all the files required for
        the program to compile including all header files(*.h).
        Run the following command "make"

    2.Open the terminal
        Go to the program folder that contains all the files required for
        the program to compile including all header files(*.h).
        COMPILE: "gcc -Wall -w -lm readDate.c functions.c -o validateDate"
                 "gcc -Wall -w -lm format.c functions.c -o format"

    Program execution:
    From the terminal enter:
     "./validateDate < dates.dat [X] | ./format > output.dat"

     X: is the amount of validated dates




+-------------------------------------------------------------------------*/
#include "general.h"

/**-------------------------------------------------------------------------+
 * Adds line to a LineList structure
 * allocating more memory if necessary.
```

```
 * @param  lineList [The list of lines to append lines to.]
 * @param  line     [The line to be appended to the list]
 * @return          [TRUE if success otherwise FALSE]
 */
Boolean appendToLineList(LineList * lineList, const char *line){

    /* Initialize variables */
    int newCapacity = 0;
    newCapacity = lineList->capacity * 2;
    Line * temp = NULL;

    if (lineList->size == lineList->capacity){

        temp = (Line *) realloc(lineList->lines,
                sizeof(lineList->lines[0]) * newCapacity);

        if (temp != NULL){
            lineList->capacity = newCapacity;
            lineList->lines = temp;
        }else{
            return FALSE;
        }
    }else{
    }

    lineList->lines[lineList->size++].content[0] = '\0';
    strncat(lineList->lines[lineList->size++].content, line, MAX_LINE);
    return TRUE;

}



/**-------------------------------------------------------------------------+
 * Frees the allocated memory from a LineList.
 * @param lineList [lineList pointer to be deallocated]
 */
void freeLineList(LineList * lineList){

    free(lineList->lines);
    lineList->capacity = 0;
    lineList->size = 0;

}



/**-------------------------------------------------------------------------+
 * Initializes the DateKeys object and returns it
 * @param  line [line to be scanned for date]
 * @return      [dateKey object initialized if valid]
 */
DateKey getDateKeys(char * line){

    /* Initialize variables */
    DateKey tempDate;
    double day = 0.0;
    double month = 0.0;
    double year = 0.0;

    /* try to get three variales*/
    if (sscanf(line, "%lf/%lf/%lf", &month, &day, &year) != VALID_KEYS){
        tempDate.error = ERRORS_UNABLE_TO_READ;
        return tempDate;
    }else{
    }


```

```
140     if (hasValidDateType(&tempDate, month, day, year)){
141         tempDate.day = (int) day;
142         tempDate.month = (int) month;
143         tempDate.year = (int) year;
144     }else{
145         return tempDate;
146     }
147 
148     if(isValidDate(&tempDate)){
149         tempDate.error = NO_ERRORS;
150     }
151 
152     return tempDate;
153 }
154 
155 
156 /**-------------------------------------------------------------------------+
157  * Checks if the values for the Date are valid
158  * @param  date  [date to check]
159  * @param  month [month value]
160  * @param  day   [day value]
161  * @param  year  [year value]
162  * @return       [true if valid otherwise is false]
163  */
164 Boolean hasValidDateType(
165                             DateKey * date,
166                             double month,
167                             double day,
168                             double year
169                         ){
170 
171     if ((fmod(day, 1.0) != 0.0) ||
172         (fmod(month, 1.0) != 0.0) ||
173         (fmod(year, 1.0) != 0.0)){
174 
175         date->error = ERRORS_DECIMALS_IN_DATE;
176         return FALSE;
177 
178     }else{
179     }
180 
181     if ((day < INT_MIN || day > INT_MAX) ||
182         (month < INT_MIN || month > INT_MAX) ||
183         (year < INT_MIN || year > INT_MAX)){
184 
185         date->error = ERRORS_HUGE_DATE;
186         return FALSE;
187 
188     }else{
189     }
190 
191     date->error = NO_ERRORS;
192     return TRUE;
193 
194 }
195 
196 
197 
198 /**-------------------------------------------------------------------------+
199  * Allocates memory for a line list to hold all
200  * the values from the input stream
201  * @param lineList [pointer to be initialized]
202  */
203 void initializeLineList(LineList * lineList){
204 
205     lineList->capacity = MIN_FILE_LINES;
206     lineList->lines = (Line *) malloc(sizeof(lineList->lines[0]) *
207                             lineList->capacity);
208     lineList->size = 0;
209 }
```

```
210
211
212 /**-------------------------------------------------------------------------+
213  * [Determines if it is a leap year]
214  * @param  year [year value]
215  * @return      [returns true if leap year otherwise false]
216  */
217 Boolean isLeapYear(int year){
218
219         return ((year % LEAP_YEAR) == 0)? TRUE : FALSE;
220 }
221
222
223
224 /**-------------------------------------------------------------------------+
225  * Check is the Date has valid values for the date
226  * @param  date [date to be checked]
227  * @return      [returns true if valid date otherwise false]
228  */
229 Boolean isValidDate(DateKey * date){
230
231     int monthDays [] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
232
233     if(isLeapYear(date->year)){
234         monthDays[1] = 29;
235     }else{
236     }
237
238     if (date->month < 1 || date->month > 12){
239         date->error  = ERRORS_INVALID_MONTH;
240         return FALSE;
241     }else{
242     }
243
244     if (date->day < 1 || date->day > monthDays[date->month - 1]){
245         date->error = ERRORS_INVALID_DAY;
246         return FALSE;
247     }else{
248     }
249
250     return TRUE;
251 }
252
253
254
255 /**-------------------------------------------------------------------------+
256  * pProcedure to print an error if the date could not be read or parsed
257  * @param stErr [the string to put the error]
258  * @param line  [line with the error]
259  * @param error [error number]
260  */
261 void printError(char * stErr, char * line, Error error){
262
263     strcpy(stErr, ERROR_DESCRIPTIONS[error]);
264     strncat(stErr,":\t ",MAX_LINE);
265     strncat(stErr, line, MAX_LINE);
266     fprintf(stderr,"%s", stErr);
267
268 }
269
270
271 /**-------------------------------------------------------------------------+
272  * [Prints a date in a specific format]
273  * @param strIn  [Buffer to fill with the output.]
274  * @param format [The format to use when printing.]
275  * @param date   [Date with all the values]
276  */
277 void printfDate(char * strIn, Format format, DateKey * date){
278
279     char monthName [MAX_MONTH_NAME];
```

```
280
281     strcpy(monthName, MONTH_NAMES_SHORT[date->month]);
282
283     if (format == DATE_FORMAT_SHORT_MONTH){
284
285         snprintf(strIn, MAX_LINE, "%2d %s %5d\n",
286                      date->day, monthName, date->year);
287
288     }else {
289
290         snprintf(strIn, MAX_LINE, "%d/%d/%d\n",
291                     date->month, date->day, date->year);
292
293     }
294 }
```