```c
/******************************************************************************
 *    database.c
 *
 *        This module uses for reading database, validating and sorting all of data
 *        (After read from database), creating new database, saving database, and
 *        writing dump file.
 *
 *    Created by Nathaphop Sundarabhogin(KLA) ID : 3420
 *        30 NOVEMBER 2017
 *
 ******************************************************************************
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ghostBuster.h"

/************** LOCAL FUNCTIONS, don't declare in header file ****************/

/* LOCAL FUNCTION. Call this function when open database first time. This function
 * will try to open database. If it can't open, that means database doesn't exist and
 * program will create a new one and return address after open file back to it.
 */
FILE *initialOpen()
    {
    FILE *pFile = NULL;     /* Get address after open file. */
    int zero = 0;           /* Write down the integer value in database. */
    pFile = fopen(DATAFILE, "rb");
    if (pFile == NULL)   /* If can't open read file, create new data file. */
        {
        printf("#SYSTEM: Data file not found, create new data file.\n");
        pFile = fopen(DATAFILE, "wb");
        if (pFile == NULL) /* If can't open a new file, program is crashed. */
            {
            printf("#SYSTEM: Error! Unable to create new data file!\n");
            printf("#SYSTEM: Program exit!\n");
            exit(0);
            }
        /* Write down amount of data in file. */
        fwrite(&zero, sizeof(int), 1, pFile);
        fclose(pFile);
        printf("#SYSTEM: Create data file success.\n\n");
        pFile = fopen(DATAFILE, "rb");
        /* After create new file, but program can't open it. Program is crashed. */
        if (pFile == NULL)
            {
            printf("#SYSTEM: Error! Program can't open read data file!\n");
            printf("#SYSTEM: Program exit!\n");
            exit(0);
            }
        }
    return pFile;
    }
```

```c
/* LOCAL FUNCTION. Write event data into text file.
 * ARGUMENTS:
 * event - Event that want to write down.
 * pFile - File output that want to write.
 */
void writeText(EVENT_T event, FILE *pFile)
    {
    /* Declare to print result message in text file. */
    char results[3][LENGTH] = {"SUCCESS","FAILURE","UNKNOWN"};
    if (event.result == 0) /* Result equal 0, means data has been deleted. */
        {
        fprintf(pFile, "Event Code: %4d-%04d\r\n",
                event.eventCode[0],
                event.eventCode[1]);
        fprintf(pFile, "Event has been deleted\r\n");
        }
    else
        {
        fprintf(pFile, "Event Code: %4d-%04d\r\n",
                event.eventCode[0],
                event.eventCode[1]);
        fprintf(pFile, "Date and Time Event: %02d/%02d/%04d %02d:%02d\r\n",
                event.dateEvent.day,
                event.dateEvent.month,
                event.dateEvent.year,
                event.dateEvent.hour,
                event.dateEvent.minute);
        fprintf(pFile, "Name Person Reportor: %s\r\n", event.nameReport);
        fprintf( pFile,"Phone Number: %s\r\n", event.phoneReport);
        fprintf(pFile, "Type of Event: %s\r\n", event.typeEvent);
        fprintf(pFile, "Latitude: %.04f\r\n", event.latitude);
        fprintf(pFile, "Longitude: %.04f\r\n", event.longitude);
        fprintf(pFile, "Date and Time Investigate: ");
        fprintf(pFile, "%02d/%02d/%04d %02d:%02d\r\n",
                event.dateInvest.day,
                event.dateInvest.month,
                event.dateInvest.year,
                event.dateInvest.hour,
                event.dateInvest.minute);
        fprintf(pFile, "Name Person Investigating: %s\r\n",
                event.nameInvest);
        fprintf(pFile, "Result: %s\r\n", results[event.result-1]);
        }
    fprintf(pFile, "\r\n");
    return;
    }
```

```
/* LOCAL FUNCTION. Sorting all of event code. If there are the same event code,
 * program will print error and exit program.
 * ARGUMENTS:
 * pEvent - All of event data that want to sort.
 * amount - Amount of data.
 */
EVENT_T *sortingDatabase(EVENT_T *pEvent, int amount)
    {
    EVENT_T temp;   /* Temporary event data, used for copy. */
    int status = 1; /* Status when there are event swap. */
    int i = 0;      /* Count loop. */
    int j = 0;      /* Count loop. */
    printf("#SYSTEM: Sorting event data.\n");
    while (status == 1) /* Bubble Sorting */
        {
        /* If check event code and there aren't any event swapped, out of loop. */
        status = 0;
        for (i = 1; i < amount; i++)
            {
            /* Check to sort event year(yyyy). */
            if (pEvent[i-1].eventCode[0] > pEvent[i].eventCode[0])
                status = 1;
            /* Check to sort for event code(nnnn). */
            else if (pEvent[i-1].eventCode[0] == pEvent[i].eventCode[0])
                {
                if ((pEvent[i-1].eventCode[1] > pEvent[i].eventCode[1]))
                    status = 1;
                /* If there are a same code. So it's an error database. */
                else if ((pEvent[i-1].eventCode[1] == pEvent[i].eventCode[1]))
                    status = -1;
                }
            if (status == 1)  /* If status is '1', swapped event. */
                {
                memcpy(&temp, &pEvent[i], sizeof(EVENT_T));
                memcpy(&pEvent[i], &pEvent[i-1], sizeof(EVENT_T));
                memcpy(&pEvent[i-1], &temp, sizeof(EVENT_T));
                }
            /* If there are the same code, print error and exit. */
            else if (status == -1)
                {
                printf("#SYSTEM: Error! Have the same event code in database!\n");
                printf("#SYSTEM: Please change or remove database before run ");
                printf("program again.\n#SYSTEM: Program exit!\n");
                exit(0);
                }
            }
        }
    printf("#SYSTEM: Finished sorting event data.\n");
    return pEvent;
    }
```

```
/* LOCAL FUNCTION. Validate all event data after read file. If event code is
 * invalid, eject that data. If data in event is not valid, delete event data
 * (Keep event code). After validate, return new data and amount back.
 * ARGUMENTS: pEvent - All of event data,  pAmount - Amount of event data.
 */
EVENT_T *validateDatabase(EVENT_T *pEvent,int *pAmount)
    {
    EVENT_T *pNew = NULL;   /* Keep new dynamic memory. */
    int eventCode[2] = {0}; /* Keep event code before delete event data. */
    int i = 0;              /* Count loop. */
    int j = 0;              /* Count loop. */
    printf("#SYSTEM: Validate event data.\n");
    for(i = 0; i < (*pAmount); i++) /* Loop check each event code. */
        {
        if (checkEventCode(pEvent[i].eventCode) != CORRECT)
            { /* Event code is invalid. */
            printf("#SYSTEM: Event code is wrong! Delete data.\n");
            *pAmount -= 1;
            for (j = i; j < (*pAmount); j++) /* Copy next event to invalid event. */
                memcpy(&pEvent[j], &pEvent[j+1], sizeof(EVENT_T));
            /* Create new memory to keep event data after eject. */
            pNew = (EVENT_T*)calloc(*pAmount, sizeof(EVENT_T));
            if (pNew == NULL) /* This condition for calloc erroring. */
                {
                printf("#SYSTEM: Error! Program can not allocate more dynamic ");
                printf("memory for deleted data.\n#SYSTEM: Program exit!\n");
                exit(0);
                }
            memcpy(pNew, pEvent, (*pAmount)*sizeof(EVENT_T));
            free(pEvent);
            pEvent = pNew;
            }
        }
    for(i = 0; i < (*pAmount) ; i++) /* Loop check event data. */
        {
        /* If wrong data, delete that event data. */
        if ((checkEvent(&pEvent[i]) != 1) && (pEvent[i].result != 0))
            {
            printf("#SYSTEM: Data in event doesn't correct. Delete event.\n");
            eventCode[0] = pEvent[i].eventCode[0];
            eventCode[1] = pEvent[i].eventCode[1];
            memset(&pEvent[i], 0, sizeof(EVENT_T));
            pEvent[i].eventCode[0] = eventCode[0];
            pEvent[i].eventCode[1] = eventCode[1];
            }
        }
    printf("#SYSTEM: Finished validate event data.\n");
    return pEvent;
    }
```

```c
/***************** PUBLIC FUNCTION, declare in header file *****************/

/* PUBLIC FUNCTION. This function will read amount of data and allocate dynamic
 * memory with that number. Then read all of event data in database and keep it
 * in dynamic memory. After that validate all of data, Then send memory of
 * event data and amount of event back.
 * ARGUMENT:
 *     eventAmount - Address of event amount.
 */
EVENT_T *readData(int *eventAmount)
    {
    EVENT_T *pEvent = NULL;  /* Keep structure of event in dynamic memory. */
    FILE *pFile = NULL;      /* Pointer of file when open file success. */
    int amount = 0;          /* Get amount of data. */
    int i = 0;               /* Count loop. */
    printf("\n#STSTEM: Start reading data file.\n");
    pFile = initialOpen();
    /* If read amount doesn't success, that means database wrong format. */
    if (fread(&amount, sizeof(int), 1, pFile) != 1)
        {
        printf("#SYSTEM: Error! Program can't read amount of data in file!\n");
        printf("#SYSTEM: Program exit!\n");
        exit(0);
        }
    if (amount == 0) /* If amount equal 0, return NULL. */
        {
        fclose(pFile);
        *eventAmount = 0;
        return NULL;
        }
    pEvent = (EVENT_T*)calloc(amount, sizeof(EVENT_T));
    if (pEvent == NULL) /* This condition for calloc error. */
        {
        printf("#SYSTEM: Error! Program can not allocate dynamic memory for ");
        printf("keep data.\n#SYSTEM: Program exit!\n");
        exit(0);
        }
    /* If read event doesn't success, that means database wrong format. */
    if (fread(&pEvent[i], sizeof(EVENT_T), amount, pFile) != amount)
        {
        printf("#SYSTEM: Error! Program can't read event data in file!\n");
        printf("#SYSTEM: Program exit!\n");
        exit(0);
        }
    fclose(pFile);
    pEvent = validateDatabase(pEvent, &amount); /* Send to validate data. */
    pEvent = sortingDatabase(pEvent, amount);   /* Send to sort data. */
    printf("#SYSTEM: Reading file success.\n\n");
    *eventAmount = amount;
    return pEvent;
    }
```

```
/* PUBLIC FUNCTION. Get event data and amount of data then write down all of event
 * data to database.
 * ARGUMENTS:
 *     pEvent      - Dynamic memory all of event data.
 *     eventAmount - Amount of event data.
 */
void saveData(EVENT_T * pEvent, int eventAmount)
    {
    FILE *pFile = NULL;     /* Pointer of file when open file success. */
    printf("#SYSTEM: Now program is saving. Please don't close program.\n");
    pFile = fopen(DATAFILE, "wb");
    if (pFile == NULL)   /* If can't open file, that mean program is crashed. */
        {
        printf("#SYSTEM: Error! Program can't open data file to save.\n");
        printf("#SYSTEM: Program exit!\n");
        exit(0);
        }
    /* Write amount of data and all of data in to database file. */
    fwrite(&eventAmount, sizeof(int), 1, pFile);
    if (eventAmount != 0)
        fwrite(pEvent, sizeof(EVENT_T), eventAmount, pFile);
    fclose(pFile);
    printf("#SYSTEM: Saving success!\n");
    return;
    }
```

```
/* PUBLIC FUNCTION. Open text file and write all of event data into text file.
 * ARGUMENTS:
 *     pEvent      - Dynamic memory all of event data.
 *     eventAmount - Amount of event data.
 */
void dumpFile(EVENT_T * pEvent, int eventAmount)
    {
    FILE *pFile = NULL; /* Pointer of file when open file success. */
    int i = 0;          /* Count loop. */
    pFile = fopen(DUMPFILE, "w");
    if (pFile == NULL) /* If it can't open file, program is crashed. */
        {
        printf("#SYSTEM: Error! Program can't open dump file!\n");
        printf("#SYSTEM: Program exit!\n");
        exit(0);
        }
    if (eventAmount == 0)   /* This condition for there aren't any data. */
        {
        fprintf (pFile,"====Doesn't have any event data====\r\n");
        fprintf (pFile,"Please, add the event first.\r\n");
        }
    for ( i = 0; (i < eventAmount) && (eventAmount != 0); i++)
        writeText(pEvent[i], pFile);   /* Write data into dump file. */
    fclose(pFile);
    return;
    }
```