```c
/*****************************************************************************
 *    manageData.c
 *
 *        This module used for keep data in dynamic memory, print data,
 *        search data, add data and control database.
 *
 *    Created by Setthawut Leelawatthanapanit(Saab) ID : 3466
 *        1 DECEMBER 2017
 *
 *****************************************************************************
 */

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

EVENT_T* pEvent; /* Keep address of dynamic memory of event data */
int eventAmount; /* The amount of event data */

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

/* LOCAL FUNCTION. If type 2 is subset of type 1, return 1.
 * But if not, return 0.
 * ARGUMENT:
 *     type1[] - Type is compared.
 *     type2[] - Type compares.
 */
int typeCpr(char type1[], char type2[])
    {
    int i = 0;      /* Counter */
    int j = 0;      /* Counter */
    for (i = 0; i < strlen(type2); i++)
        { /* Loop check each character in type that will compare. */
        for (j = 0; j < strlen(type1)  && (i%2 == 0); j++)
            { /* Loop check each character in type that is compared. */
            /* Character of both type is the same. */
            if (type2[i]==type1[j])
                break;
            /* Character of both type is NOT the same. */
            else if (j+1 == strlen(type2))
                return 0;
            }
        }
    return 1;
    }
```

```
/* LOCAL FUNCTION. Allocate new memory for integer and if old memory is exists,
 * copy the old memory to new memory. When finished, return new address of memory.
 * ARGUMENT:
 *      pInt   - The last data memory.
 *      amount - Amount that want to allocate new one.
 */
int * allocInt(int *pInt, int amount)
    {
    int *pNew = NULL;
    pNew = (int*)calloc(amount, sizeof(int));
    if (pNew == NULL) /* This condition for realloc error. */
        {
        printf("#SYSTEM: Error! Program can not allocate more dynamic memory");
        printf(" for searching.\n#SYSTEM: Program exit!\n");
        exit(0);
        }
    if (pInt != NULL) /* This condition for last 'pInt' memmory is exist. */
        {
        memcpy(pNew, pInt, (amount-1)*sizeof(int));
        free(pInt);
        }
    return pNew;
    }
```

```
/***************** PUBLIC FUNCTIONS, declare in header file ******************/

/* PUBLIC FUNCTION. Function that manage event data to database. Get the command
 * to decide what to do to database.
 * ARGUMENT:
 *     command - Get number to know that what to do to database.
 *         1 - Start program. Read all data in database.
 *         2 - Save Data. Save all data to database.
 *         3 - Dump File. Dump text file output.
 *         4 - Save data and free data. Used when close program.
 */
void controlDatabase(int command)
    {
    if (command == 1)
        pEvent = readData(&eventAmount);
    else if (command == 2)
        saveData(pEvent, eventAmount);
    else if (command == 3)
        dumpFile(pEvent, eventAmount);
    else if (command == 4)
        {
        saveData(pEvent, eventAmount);
        free(pEvent);
        printf("#SYSTEM: Free all dynamic memory.\n");
        }
    return;
    }
```

```c
/* PUBLIC FUNCTION. Reallocate new one dynamic memory from last data, copy
 * new event into new dynamic memory and sorting it.
 * ARGUMENT:
 *     event - Event that want to add to data.
 */
void addEvent(EVENT_T event)
    {
    EVENT_T *pNew = NULL;    /* Get new dynamic memory. */
    int i = 0;               /* Used for count loop. */
    eventAmount += 1;   /* Plus one event amount to reallocate new memory. */
    pNew = (EVENT_T*)calloc(eventAmount, sizeof(EVENT_T));
    if (pNew == NULL) /* This condition for realloc error. */
        {
        printf("#SYSTEM: Error! Program can not allocate more dynamic memory");
        printf(" for keep data.\n#SYSTEM: Program exit!\n");
        exit(0);
        }
    /* If pEvent is exist, copy from old to new one. */
    if(pEvent != NULL)
        {
        memcpy(pNew, pEvent, (eventAmount-1)*sizeof(EVENT_T));
        memcpy(&pNew[eventAmount-1], &event, sizeof(EVENT_T));
        }
    /* Sorting new event into structure array of event data */
    for ((i = eventAmount - 1); i != 0; i--)
        {
        if (pNew[i].eventCode[0] < pNew[i-1].eventCode[0])
            {
            memcpy(&pNew[i], &pNew[i-1], sizeof(EVENT_T));
            memcpy(&pNew[i-1], &event, sizeof(EVENT_T));
            }
        else if (event.eventCode[0] >= pNew[i-1].eventCode[0])
            {
            memcpy(&pNew[i],&event,sizeof(EVENT_T));
            break;
            }
        }
    if (pEvent != NULL)
        free(pEvent);
    pEvent = pNew;
    controlDatabase(2); /* Save all of event to database */
    printf("\n");
    return;
    }
```

```
/* PUBLIC FUNCION. Print data of each event.
 * ARGUMENTS:
 *     pInt   - Position in data that want to print event data.
 *     amount - The amount of position.
 */
void printEachEvent(int *pInt, int amount)
    {
    int i = 0;          /* Count loop. */
    int eventYear = 0;  /* Keep event year to print head event year. */
    /* This condition for if there is not any data event. */
    if ((eventAmount == 0) || (amount == 0))
        {
        printf("========== Doesn't have any data ==========\n\n");
        return;
        }
    for (i = 0; i < amount; i++)
        {
        /* This condition for print head event year */
        if (eventYear != pEvent[pInt[i]].eventCode[0])
            {
            eventYear = pEvent[pInt[i]].eventCode[0];
            printf("========== EVENT YEAR %d ==========\n\n", eventYear);
            }
        printEvent(pEvent[pInt[i]]);
        }
    return;
    }
```

```
/* PUBLIC FUNCTION. Loop print all of event data to terminal line */
void printAllEvent()
    {
    int i = 0;          /* Count loop. */
    int eventYear = 0;  /* Keep event year to print head event year. */
    printf("\n=================== Display all event ====================\n\n");
    if (eventAmount == 0) /* This condition for if there is not any data event. */
        {
        printf("========== Doesn't have any data ==========\n\n");
        return;
        }
    for (i = 0; i < eventAmount; i++) /* Loop send each data to print event. */
        {
        /* This condition for print head event year. */
        if (eventYear != pEvent[i].eventCode[0])
            {
            eventYear = pEvent[i].eventCode[0];
            printf("========== EVENT YEAR %d ==========\n\n", eventYear);
            }
        printEvent(pEvent[i]);
        }
    printf("==========================================================\n\n");
    return;
    }
```

```
/* PUBLIC FUNCTION. Loop find the lastest code in year that want to know. If found
 * the code, return that count plus 1 (New code for that year). But if not found
 * or there aren't any data, return 1.
 * ARGUMENT:
 *     eventYear - Year that want to run event code.
 */
int runEventCode(int eventYear)
    {
    int code = 0;            /* Keep code to run event code. */
    int i = 0;               /* Count loop. */
    if (eventAmount == 0) /* If there isn't any data, return 1 for a new data. */
        return 1;
    /* Loop check which code in 'eventYear' is the lastest. */
    for (i = 0; i < eventAmount; i++)
        {
        if (eventYear == pEvent[i].eventCode[0])
            {
            if (code < pEvent[i].eventCode[1])
                code = pEvent[i].eventCode[1];
            else if (code = pEvent[i].eventCode[1])
                {
                printf("#SYSTEM: Error! There are the same event code!\n");
                printf("#SYSTEM: Please change or remove data.\n");
                printf("#SYSTEM: Program exit!\n");
                exit(0);
                }
            }
        }
    if (code+1 > 9999)  /* If it's maximum possible of event code, return 0. */
        return 0;
    return code+1;
    }
```

```
/* PUBLIC FUNCTION. Search event code in data. If there is event code in data,
 * print information and return '1' and position back.
 * If it does not have, print message and return '0' back.
 * ARGUMENTS:
 *     code[]     - Keep event code that want to search.
 *     pPosition  - Keep position of data.
 */
int searchEventCode(int code[], int *pPosition)
    {
    int i = 0;      /* Count loop. */
    int status = 0; /* Keep value correction of finding information. */
    for (i = 0; i < eventAmount; i++)
        {
        if ((pEvent[i].result == 0) && (code[0] == pEvent[i].eventCode[0]) &&
           (code[1] == pEvent[i].eventCode[1]))
            { /* That event data has been deleted. */
            printf("#SYSTEM: That event code has been deleted.\n");
            return 0;
            }
        else if ((code[0] == pEvent[i].eventCode[0]) &&
           (code[1] == pEvent[i].eventCode[1]))
            { /* Have information that users want. */
            printEvent(pEvent[i]);
            *pPosition = i;
            return 1;
            }
        }
    printf("#SYSTEM: Information of %04d-%04d is not found.\n", code[0], code[1]);
    return 0;
    }
```

```
/* PUBLIC FUNCTION. Search data by using event year
 * and then return all of the position of data.
 * ARGUMENTS:
 *     eventYear  - Event year that want to search.
 *     pCountData - The amount of position.
 */
int * searchEventYear(int eventYear, int * pCountData)
    {
    int i = 0;              /* Counter. */
    int *pPosition = NULL;  /* Keep position of information in database. */
    for (i = 0; i < eventAmount; i++)
        {
        if (eventYear == pEvent[i].eventCode[0])
            { /* Event year has in database. */
            pPosition = allocInt(pPosition, *pCountData + 1);
            pPosition[*pCountData] = i;
            *pCountData += 1;
            }
        }
    return pPosition;
    }
```

```
/* PUBLIC FUNCTION. Search data by using result
 * and then return all of the position of data.
 * ARGUMENTS:
 *     result     - Result that want to search.
 *     pPosition  - Last position of data in database.
 *     pCountData - The amount of data.
 */
int * searchResult(int result, int * pPosition, int * pCountData)
    {
    int i = 0;                /* Counter. */
    int count = 0;            /* Count amount of new position. */
    int *pNewPosition = NULL; /* Keep integer of position temporary. */
    if (pPosition == NULL)
        { /* Don't have data that get before. */
        for (i = 0; i < eventAmount; i++)
            {
            if (result == pEvent[i].result)
                { /* Result has in database. */
                pNewPosition = allocInt(pNewPosition, *pCountData + 1);
                pNewPosition[*pCountData] = i;
                *pCountData += 1;
                }
            }
        }
    else if (pPosition != NULL)
        { /* Have data that get before. */
        for (i = 0; i < *pCountData; i++)
            {
            if (result == pEvent[pPosition[i]].result)
                { /* Result has in database. */
                pNewPosition = allocInt(pNewPosition, count + 1);
                pNewPosition[count] = i;
                count += 1;
                }
            }
        free(pPosition);
        *pCountData = count;
        }
    pPosition = pNewPosition;
    return pPosition;
    }
```

```
/* PUBLIC FUNCTION. Search data by using type of event
 * and then return all of the position of information
 * ARGUMENTS:
 *     type[]     - Type of event that want to search.
 *     pPosition  - Last position of data in database.
 *     pCountData - The amount of data.
 */
int * searchEventType(char type[], int * pPosition, int *pCountData)
    {
    int i = 0;                    /* Counter. */
    int count = 0;                /* Count amount of new position. */
    int *pNewPosition = NULL;     /* Keep integer of position temporary. */
    if (pPosition == NULL)
        { /* Don't have data that get before. */
        for (i = 0; i < eventAmount; i++)
            {
            if (typeCpr(pEvent[i].typeEvent, type) == 1)
                { /* Type of event has in database. */
                pNewPosition = allocInt(pNewPosition, *pCountData + 1);
                pNewPosition[*pCountData] = i;
                *pCountData += 1;
                }
            }
        }
    else if (pPosition != NULL)
        { /* Have data that get before. */
        for (i = 0; i < *pCountData; i++)
            {
            if (typeCpr(pEvent[pPosition[i]].typeEvent, type) == 1)
                { /* Type of event has in database. */
                pNewPosition = allocInt(pNewPosition, count + 1);
                pNewPosition[count] = i;
                count += 1;
                }
            }
        free(pPosition);
        *pCountData = count;
        }
    pPosition = pNewPosition;
    return pPosition;
    }
```

```c
/* PUBLIC FUNCTION. Get position of event data and ask user to modify.
 * If the user hits <CR> or get input, program will ask next information until finished.
 * In the end, program asks users to save data.
 * ARGUMENT: position - Position of data in event data.
 */
void modifyEvent(int position)
    {
    EVENT_T event;             /* Struct of information. */
    char input[LENGTH] = {0}; /* Get input from the terminal. */
    int temp = 0;              /* Store value temporary. */
    int status = 0;   /* Keep value correction after validate information. */
    memcpy(&event, &pEvent[position], sizeof(EVENT_T));
    printf("\n#SYSTEM: If don't want to change, hit to the next.\n");
    if (askName(event.nameReport, 1) == 1)
        status++;
    if (askPhone(event.phoneReport) == 1)
        status++;
    if (askType(event.typeEvent) == 1)
        status++;
    if (askLocation(&event.latitude, 1) == 1)
        status++;
    if (askLocation(&event.longitude, 2) == 1)
        status++;
    if (askDate(&event.dateInvest, &event.dateEvent) == 1)
        status++;
    if (askName(event.nameInvest, 2) == 1)
        status++;
    if (askResult(&event.result) == 1)
        status++;
    if (status == 0)
        {
        printf("\n#SYSTEM: Cancel to modify.\n");
        return;
        }
    printf("\n");
    printEvent(event);
    printf("Do you want to save information?(Y/N): ");
    while(1)
        {
        memset(input, 0, sizeof(input));
        fgets(input, sizeof(input), stdin);
        if (((input[0] == 'Y') || (input[0] == 'y')) && (strlen(input) == 2))
            { /* Users want to save information. */
            memcpy(&pEvent[position], &event, sizeof(EVENT_T));
            controlDatabase(2);
            return;
            }
        else if (((input[0] == 'N') || (input[0] == 'n')) && (strlen(input) == 2))
            return; /* Users don't want to save information. */
        printf("#SYSTEM: Input is invalid, try again(Y/N): ");
        }
    }
```

```
/* PUBLIC FUNCTION. Get position of event data and ask user
 * for sure to delete event code.
 * ARGUMENT:
 *     position - Position of data in database.
 */
void deleteEvent(int position)
    {
    char input[LENGTH] = {0};    /* Get input from the terminal. */
    int eventCode[2] = {0};      /* Keep last event code temporary. */
    printf("Do you want to delete information?(Y/N): ");
    while(1)
        {
        memset(input, 0, sizeof(input));
        fgets(input, sizeof(input), stdin);
        if (strlen(input) == 2)
            {
            if ((input[0] == 'Y') || (input[0] == 'y'))
                { /* Users want to delete information. */
                eventCode[0] = pEvent[position].eventCode[0];
                eventCode[1] = pEvent[position].eventCode[1];
                memset(&pEvent[position], 0, sizeof(EVENT_T));
                pEvent[position].eventCode[0] = eventCode[0];
                pEvent[position].eventCode[1] = eventCode[1];
                printEvent(pEvent[position]);
                controlDatabase(2);
                return;
                }
            else if ((input[0] == 'N') || (input[0] == 'n'))
                return; /* Users don't want to delete information. */
            }
        printf("#SYSTEM: Input is invalid, try again(Y/N): ");
        }
    }
```