```
/*******************************************************************************
 *    validateEvent.c
 *
 *        All of this function used to check data in event data, print event
 *        and get current time.
 *
 *    Created by Natthawat Tungruethaipak(Tong) ID : 3426
 *        16 NOVEMBER 2017
 *
 *******************************************************************************
 */

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

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

/* LOCAL FUNCTION. Check day, month, year and time that they're exists or not.
 * If date is exists, return 1. But if date doesn't exists, return 0.
 * ARGUMENT:
 *     date - Struct of date that want to validate.
 */
int checkDateExist(DATE_T date)
    {
    /* Check time it's between '00:00 - 23:59' or not. */
    if ((date.hour < 0) || (date.hour > 23) ||
        (date.minute < 0) || (date.minute > 59))
        return 0;
    /* Check month that have 30 days. */
    if ((date.month == 4) || (date.month == 6) || (date.month == 9)
        || (date.month == 11))
        {
        if ((date.day >=1) && (date.day <= 30))
            return 1;
        }
    /* Check February month. First check leap year, and then check days. */
    else if (date.month == 2)
        {
        /* Leap year. */
        if (((date.year % 4) == 0) && ((date.day >= 1) && (date.day <= 29)))
            return 1;
        /* Common year. */
        else if (((date.year % 4) != 0) && ((date.day >= 1) && (date.day <= 28)))
            return 1;
        }
    /* Check month that have 31 days. */
    else
        {
        if ((date.day >= 1) && (date.day <= 31))
            return 1;
        }
    return 0; /* Otherwise, date doesn't exists. */
    }
```

```
/* LOCAL FUNCTION. Compare between 2 dates. Return 1 - If first date is ealier
 * than second date. Return 0 - If first date is later than or equal to second date.
 * ARGUMENT:
 *     date1 - Struct of first date that want to compare.
 *     date2 - Struct of second date that want to compare.
 */
int dateCpr(DATE_T date1, DATE_T date2)
    {
    /* Compare which year is later or ealier. */
    if (date1.year > date2.year)
        return 0;
    else if (date1.year < date2.year)
        return 1;
    /* If years are the same, Compare which month is later or ealier. */
    else if (date1.month > date2.month)
        return 0;
    else if (date1.month < date2.month)
        return 1;
    /* If months are the same, Compare which day is later or ealier. */
    else if (date1.day > date2.day)
        return 0;
    else if (date1.day < date2.day)
        return 1;
    /* If days are the same, Compare which hour is later or ealier. */
    else if (date1.hour > date2.hour)
        return 0;
    else if (date1.hour < date2.hour)
        return 1;
    /* If hours are the same, Compare which minutes is later or ealier. */
    else if (date1.minute > date2.minute)
        return 0;
    else if (date1.minute < date2.minute)
        return 1;
    return 0;   /* Otherwise, both days are the same. */
    }
```

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

/* PUBLIC FUNCTION. Get the date to day and current time.
 * ARGUMENT:
 *     pDate - Address of struct date that want to get current time.
 */
void dateToday(DATE_T *pDate)
    {
    char input[LENGTH] = {0}; /* Get input from date. */
    DATE_T temp;              /* Temporary structure of date to keep date today. */
    time_t t = time(NULL);
    struct tm * tm = localtime(&t);
    /* Get current day. */
    strftime(input, sizeof(input), "%d", tm);
    sscanf(input, "%d", &temp.day);

    /* Get current month. */
    strftime(input, sizeof(input), "%m", tm);
    sscanf(input, "%d", &temp.month);

    /* Get current year. */
    strftime(input, sizeof(input), "%G", tm);
    sscanf(input, "%d", &temp.year);

    /* Get current hour. */
    strftime(input, sizeof(input), "%H", tm);
    sscanf(input, "%d", &temp.hour);

    /* Get current minute. */
    strftime(input, sizeof(input), "%M", tm);
    sscanf(input, "%d", &temp.minute);
    memcpy(pDate, &temp, sizeof(DATE_T));
    return;
    }
```

```
/* PUBLIC FUNCTION. Check string of date it is correct or not. If correct,
 * return correct. If not return error in each case.
 * ARGUMENT:
 *     dateStr - String of date that want to check.
 */
CHECK_STATUS checkDateStr(char dateStr[])
    {
    int i = 0;      /* Count loop. */
    if (strlen(dateStr) != 16)   /* Length of date string must be 16 characters. */
        return ERR_LEN;
    for(i = 0; i < strlen(dateStr); i++) /* Check it is digit or not. */
        {
        if (!(isdigit(dateStr[i])) && (i != 2) && (i != 5) && (i != 10)
            && (i != 13))
            return ERR_CHAR;
        }
    /* Check '-' and ':' it is in the correct position or not.*/
    if ((dateStr[2] != '-') || (dateStr[5] != '-') || (!isspace(dateStr[10])) ||
        (dateStr[13] != ':'))
        return ERR_LEN;
    return CORRECT;
    }
```

```
/* PUBLIC FUNCION. Check that date is not in the future. If 'pDateEvent' is NULL,
 * check that date is within past 10 years or not. But if 'pDateEvent' is not
 * NULL, check that date is ealier than event date or not. After validate, If it's
 * correct, return correct. But if not, return error in each case.
 * ARGUMENTS:
 *     date       - Struct of date that want to check.
 *     pDateEvent - Struct of event date if want to check earlier then event
 *                  date or not. If it's NULL, it will check within 10 years instead.
 */
CHECK_STATUS checkDate(DATE_T date, DATE_T *pDateEvent)
    {
    DATE_T temp;    /* Temporary date to compare. */
    if (checkDateExist(date) == 0) /* Check date and time is exist or not. */
        return ERR_FORM1;
    dateToday(&temp);
    if (dateCpr(date, temp) == 0) /* Check date is in future or not. */
        return ERR_FORM1;
    /* If 'pDateEvent' is 'NULL', check date within 10 year. */
    if (pDateEvent == NULL)
        {
        temp.year -= YEARSCOPE;
        temp.minute -= 1;
        if (dateCpr(temp, date) == 0)
            return ERR_FORM2;
        }
    /* Check date it's ealier than event date or not. */
    else if (pDateEvent != NULL)
        {
        memcpy(&temp, pDateEvent, sizeof(DATE_T));
        if (dateCpr(temp, date) == 0)
            return ERR_FORM3;
        }
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Validate string of name. If name is valid, return correct.
 * But if invalid, return error in each case.
 * ARGUMENT:
 *     nameStr[] - String of name that want to validate.
 */
CHECK_STATUS checkName(char nameStr[])
    {
    int i = 0;           /* Count loop. */
    char temp[LENGTH];   /* Temporary of string to validate. */
    strcpy(temp, nameStr);
    /* Loop check that character in name is allow or not. */
    for( i = 0; i < strlen(temp); i++)
        {
        if (!((isalpha(temp[i])) || (temp[i] == '\'') ||
            (temp[i] == '-') || (temp[i] == '.') ||
            (isspace(temp[i]))))
            return ERR_CHAR;
        }
    /* First character must be character. */
    if ((temp[0] == '\'') || (temp[0] == '-') ||
        (temp[0] == '.') || (isspace(temp[0])))
        return ERR_FORM1;
    /* Loop check each puntuation and change next character to upper case. */
    for( i = 0; i < strlen(temp); i++)
        {
        if ((temp[i] == '\'') || (temp[i] == '-')
            || (temp[i] == '.') || (isspace(temp[i])))
            {
            if (isalpha(temp[i+1]))
                temp[i+1] = toupper(temp[i+1]);
            /* If next character isn't alphabet or last, it's wrong format. */
            else if ((!isalpha(temp[i+1])) && (i<(strlen(temp)-1)))
                return ERR_FORM2;
            }
        }
    /* Last character shouldn't be punctuation except period. */
    if ((temp[strlen(temp)-1] == '\'') || (temp[strlen(temp)-1] == '-') ||
        (temp[strlen(temp)-1] == ' '))
        return ERR_FORM3;
    if (islower(temp[0]))
       temp[0] = toupper(temp[0]);
    strcpy(nameStr,temp);
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Validate string of phone. If phone is valid, return correct.
 * But if invalid, return error in each case.
 * ARGUMENT:
 *     phoneStr[] - String of phone that want to check.
 */
CHECK_STATUS checkPhone(char phoneStr[])
    {
    int count = 0;      /* Count loop. */
    int countDash = 0;  /* Count dash in phone number. */
    /* Check length of phone number that correct or not. */
    if ((strlen(phoneStr)-1 > 11) || (strlen(phoneStr)-1 < 8))
        return ERR_LEN;
    /* Loop check each character and count dash. */
    for(count = 0 ; count < strlen(phoneStr) ; count ++)
        {
        /* Phone number is not number or dash. */
        if (!isdigit(phoneStr[count]) && (phoneStr[count] != '-'))
            return ERR_CHAR;
        else if (phoneStr[count] == '-')
            countDash++;
        }
    /* If first number isn't '0' or second is '0', wrong city code. */
    if ((phoneStr[0] != '0') || (phoneStr[1] == '0'))
        return ERR_FORM1;
    /* This condition for dash more than one. */
    else if (countDash > 1)
        return ERR_FORM2;
    /* This condition for dash is in wrong position. */
    else if ((phoneStr[2] != '-') && (phoneStr[3] != '-'))
        return ERR_FORM3;
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Validate string of type. If type is valid, return correct.
 * But if invalid, return error in each case.
 * ARGUMENT:
 *     typeStr[] - String of type that want to check.
 */
CHECK_STATUS checkType(char typeStr[])
    {
    int i = 0;                                  /* Count loop. */
    int j = 0;                                  /* Count loop. */
    int countType[5] = {0};                     /* Count each type. */
    char type[6] =  {'V','W','Z','G','O'};      /* Scope of type. */
    if (strlen(typeStr) > 9) /* Type is longer than 9 characters. */
        return ERR_LEN;
    /* Loop check each character are allowed or not */
    for (i = 0; i < strlen(typeStr); i++)
        {
        if (!((typeStr[i] == ',') || (typeStr[i] == 'V') || (typeStr[i] == 'W')
            || (typeStr[i] == 'G') || (typeStr[i] == 'Z') || (typeStr[i] == 'O')))
            return ERR_CHAR;
        }
    /* Type have comma next to each other. */
    if (strstr(typeStr, ",,") != NULL)
        return ERR_FORM1;
    /* First character must be character. */
    else if (typeStr[0] == ',')
        return ERR_FORM2;
    /* Last character mustn't be comma. */
    else if (typeStr[strlen(typeStr)-1] == ',')
        return ERR_FORM3;
    for(i = 0; i < strlen(typeStr); i++) /* Loop count each type in string. */
        {
        for (j = 0; j < 5; j++)
            {
            if (typeStr[i] == type[j])
                countType[j]++;
            if (countType[j] >= 2)  /* There are same type. */
                return ERR_FORM4;
            }
        }
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Get string of latitude or longitude and validate string
 * correct or not. If string of latitude/longitude is correct return correct,
 * But if not return error in each case.
 * ARGUMENTS:
 *     locationStr[] - String that want to check in latitude/longitude form.
 *     select      - Select to check latitude or longitude.
 *                   (1=latitude, 2=longitude)
 */
CHECK_STATUS checkStrLocation(char locationStr[], int select)
    {
    int i = 0;          /* Count loop. */
    int countDot = 0;   /* Count period. */
    if (select == 1) /* Check latitude. */
       {
       /* Length of latitude must be 7 characters. */
       if (strlen(locationStr) != 7)
             return ERR_FORM1;
        /* Position of period must be the fifth from the last string. */
        if (locationStr[strlen(locationStr) - 5] != '.')
             return ERR_FORM1;
       }
    else if (select == 2) /* Check longitude. */
       {
       /* Length of longitude must be between 7 and 8 characters. */
       if ((strlen(locationStr) != 7) && (strlen(locationStr) != 8))
             return ERR_FORM2;
        /* Position of period must be the fifth from the last string. */
        if (locationStr[strlen(locationStr) - 5] != '.')
             return ERR_FORM2;
       }
    for(i = 0 ; i < strlen(locationStr); i++)
        { /* Loop count each character in string. */
        /* Location must be digit. */
        if (!isdigit(locationStr[i]) && (locationStr[i] != '.'))
            return ERR_CHAR;
        }
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Check latitude/longitude is correct or not, which know by
 * 'select'. Select equal 1, check latitude. Select equal 2, check longitude.
 * If correct, return correct. But if not, return error in each case.
 * ARGUMENTS:
 *     location - Latitude/longitude that want to check.
 *     select   - Select to check latitude or longitude.
 *                (1=latitude, 2=longitude)
 */
CHECK_STATUS checkLocation(float location,int select)
    {
    if(select == 1) /* If select is 1, check latitude. */
        {
        if ((location < 8) || (location > 22))
            return ERR_FORM3;
        }
    else if (select == 2) /* If select is 2, check longitude. */
        {
        if ((location < 98) || (location > 102))
            return ERR_FORM4;
        }
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Check event code correct or not. If event code is correct,
 * return 1. But if not, return 0.
 * ARGUMENT:
 *     codeStr[] - Keep event code that want to check.
 */
CHECK_STATUS checkEventCodeStr(char codeStr[])
    {
    int i = 0;      /* Count loop. */
    if (strlen(codeStr) != 9) /* Length of event code is incorrect. */
        return ERR_LEN;
    for(i = 0; i < strlen(codeStr); i ++)
        {
        if ((!(isdigit(codeStr[i]))) && (i != 4)) /* Event code is not digit. */
            return ERR_CHAR;
        }
    if (codeStr[4] != '-') /* Position of dash is incorrect. */
        return ERR_FORM1;
    return CORRECT;
    }
```

```
/* PUBLIC FUNCTION. Check all data in event (except event code) correct or not.
 * If all of data in event is correct, return 1. But if not, return 0.
 * ARGUMENT:
 *     event - Struct of event that want to check data.
 */
int checkEvent(EVENT_T *event)
    {
    int check = 0;  /* Get result after send to check. */
    check = checkDate(event->dateEvent, NULL);
    /* Date event in database can earlier than 10 years. */
    if ((check == CORRECT) || (check == ERR_FORM2))
        check = checkDate(event->dateInvest, &event->dateEvent);
    if (check == CORRECT)
        check = checkPhone(event->phoneReport);
    if (check == CORRECT)
        check = checkType(event->typeEvent);
    if (check == CORRECT)
        check = checkName(event->nameReport);
    if (check == CORRECT)
        check = checkName(event->nameInvest);
    if (check == CORRECT)
        check = checkLocation(event->latitude, 1); /* Check latitude. */
    if (check == CORRECT)
        check = checkLocation(event->longitude, 2); /* Check longitude. */
    if ((event->result < 0) || (event->result > 3)) /* Check Result. */
        check = ERR_FORM1;
    if (event->eventCode[0] != (*event).dateEvent.year) /* Check event year. */
        check = ERR_FORM1;
    if (check != CORRECT) /* If it isn't correct return 0. */
        return 0;
    return 1; /* Otherwise, it's correct. */
    }
```

```
/* PUBLIC FUNCTION. Check event code it's correct or not. If correct, return 1
 * If not correct, return 0.
 * ARGUMENT:
 *     code[] - Event code that want to check.
 */
CHECK_STATUS checkEventCode(int code[])
    {
    DATE_T today;   /* Struct of the date today. */
    dateToday(&today);
    if (code[0] > today.year) /* Year mustn't be in the future. */
        return ERR_FORM1;
    /* Code mustn't be less than 0 or more than 9999. */
    else if ((code[1] < 1) || (code[1] > 9999))
        return ERR_FORM2;
    return CORRECT;
    }
```