CODE ARTISANSHIP UNSW HS1511

BY JASONYU

UNSW HS1511

DOOMSDAY

```
int dayOfWeek (int doomsday, int leapYear, int month, int day) {
    int originDoomsday = getCummulativeDays(2,28,FALSE);
    int dayCount = getCummulativeDays(month,day,leapYear);
    if (leapYear == TRUE) {
        originDoomsday += 1;
    }
    int totalDifference = dayCount - originDoomsday;
    int dayDifference;
    int dayOfWeek;
    if (totalDifference < 0) {
        totalDifference *= -1;
        dayDifference = (DAYS_PER_WEEK) - (totalDifference % 7);
    } else {
        dayDifference = totalDifference % 7;
    }
    dayOfWeek = ((doomsday + dayDifference) % 7);
    return (dayOfWeek);
}

int checkLongMonth(int month) {
    int flag = 0;
    if (month < AUG && month % 2 == 1) {
        flag = 1;
    } else if (month >= AUG && month % 2 == 0) {
        flag = 1;
    }
    return flag;
}

int getCummulativeDays(int month, int day, int leapYear) {
    int currentMonth = 1;
    int dayCounter = 0;
    while (currentMonth < month) {
        if (currentMonth == FEB) {
            dayCounter += FEB_COUNT;
        } else if (checkLongMonth(currentMonth) == TRUE) {
            dayCounter += LONGM;
        } else {
            dayCounter += SHORTM;
        }
        currentMonth += 1;
    }
    dayCounter += day;
    if (leapYear == TRUE && month > FEB) {
        dayCounter += 1;
    }
    return dayCounter;
}
```

1. Readable and Understandable variable and function names.
2. Consistent White Spaces
3. If and While Loop syntax is well constructed. Meaningful counter names.
4. Constant variables being used.

DOOMSDAY CONTINUED

```c
// Include Libraries
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
// Days of Week as NUM
#define THURSDAY 0
#define FRIDAY   1
#define SATURDAY 2
#define SUNDAY   3
#define MONDAY   4
#define TUESDAY  5
#define WEDNESDAY 6
// Months in Year as NUM
#define JAN 1
#define FEB 2
#define MAR 3
#define APR 4
#define MAY 5
#define JUN 6
#define JUL 7
#define AUG 8
#define SEP 9
#define OCT 10
#define NOV 11
#define DEC 12
// Typical Month Lengths and Feb Count
#define SHORTM 30
#define LONGM 31
#define FEB_COUNT 28
// Booleans
#define TRUE 1
#define FALSE 0
// Days in a Week
#define DAYS_PER_WEEK 7

// Functions
int checkLongMonth(int month);
int dayOfWeek (int doomsday, int leapYear, int month, int day);
int getCummulativeDays(int month, int day, int leapYear);
void tests(void);
```

Functions, Constants and tests

```c
void tests(void) {
    assert (dayOfWeek (THURSDAY,  FALSE,  4,  4) == THURSDAY);
    assert (dayOfWeek (FRIDAY,    FALSE,  6,  6) == FRIDAY);
    assert (dayOfWeek (MONDAY,    FALSE,  8,  8) == MONDAY);
    assert (dayOfWeek (WEDNESDAY, FALSE, 10, 10) == WEDNESDAY);
    assert (dayOfWeek (FRIDAY,    FALSE, 12, 12) == FRIDAY);
    assert (dayOfWeek (THURSDAY,  FALSE,  9,  5) == THURSDAY);
    assert (dayOfWeek (FRIDAY,    FALSE,  5,  9) == FRIDAY);
    assert (dayOfWeek (SUNDAY,    FALSE,  7, 11) == SUNDAY);
    assert (dayOfWeek (TUESDAY,   FALSE, 11,  7) == TUESDAY);
    assert (dayOfWeek (WEDNESDAY, FALSE,  3,  7) == WEDNESDAY);
    assert (dayOfWeek (THURSDAY,  FALSE,  4,  5) == FRIDAY);
    assert (dayOfWeek (SATURDAY,  FALSE,  6,  5) == FRIDAY);
    assert (dayOfWeek (MONDAY,    FALSE,  8,  9) == TUESDAY);
    assert (dayOfWeek (WEDNESDAY, FALSE, 10,  9) == TUESDAY);
    assert (dayOfWeek (FRIDAY,    FALSE, 12, 20) == SATURDAY);
    assert (dayOfWeek (THURSDAY,  FALSE,  9,  9) == MONDAY);
    assert (dayOfWeek (FRIDAY,    FALSE,  5,  5) == MONDAY);
    assert (dayOfWeek (SUNDAY,    FALSE,  7,  7) == WEDNESDAY);
    assert (dayOfWeek (TUESDAY,   FALSE, 11, 11) == SATURDAY);
    assert (dayOfWeek (THURSDAY,  FALSE,  3, 30) == SATURDAY);
    assert (dayOfWeek (TUESDAY,   FALSE,  2,  28) == TUESDAY);
    assert (dayOfWeek (TUESDAY,   FALSE,  2,  27) == MONDAY);
    assert (dayOfWeek (THURSDAY,  FALSE,  1,  3)  == THURSDAY);
```

MANDELBROT

Planning

- EXPLORE MANDELBROT
- COMPLETE BMP TASKS
- USE WIKIPEDIA TO UNDERSTAND MANDELBROT
- WRITE ESCAPE STEPS
- CREATE SERVER
- MODIFY TO SERVE MANDELBROT
- ADD COLOR

KNOWLEDGE ISLAND

```
// STRUCTS
typedef struct _player {
    // Campuses
    path campuses[PATH_LIMIT];
    // Arc Grants
    path arcs[PATH_LIMIT];
    // GO8 Campuses
    path upgradedCampuses[PATH_LIMIT];
    int ThD;
    int BPS;
    int BHW;
    int MJ;
    int MTV;
    int MM;
    // Achievements, used in the final calculation of KPI points
    int numCampuses;
    int numGO8Campuses;
    int numARCGrants;
    int numIPs;
    int numPublications;
    // Records whether player's most ARC and most Publications achievement is first time
    int firstTimeMostARCGrants;
    int firstTimeMostPublications;
    // First player to 150 KPI wins
    int numKPI;
} newPlayer;
```

- Correct Syntax
- Comments
- Specific Struct Syntax
- Good Variable Names
- Hash defined values

KNOWLEDGE ISLAND

```
static int compareCoords(coordinate coordOne,coordinate coordTwo) {
    return coordOne.row == coordTwo.row && coordOne.col == coordTwo.col;
}

static coordinate getCoordinate(path pathOne) {
    int pathIndex = 0;
    int flag = 0;
    coordinate coord = {0,3}; // row 0,col 3
    coordinate history[PATH_LIMIT];
    history[0] = coord;
    int currentIndex = 0;
    while (pathIndex < PATH_LIMIT && flag == 0) {
        char command = pathOne[pathIndex];
        coord = updateCoord(coord,command,&history,&currentIndex);
        pathIndex ++;
        if (command == '\0') {
            flag = 1;
        }
    }
    return coord;
}
```

1. Usage of sub functions to help make other functions easier to understand,

2. Static function usage. These are great for repeated tasks where given functions have similar needs.

3. Usage of flag variables and pointers

LINKED LISTS

```c
#include <stdlib.h>
#include <stdio.h>


typedef struct _node *link;

typedef struct _node {
    int value;
    link next;
} node;

typedef struct _list {
    link head;
} *list;

void reverse(list l) {
    link prev = NULL;
    link current = l->head;
    link next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    l->head = prev;
}

list zip(list l1,list l2) {
    int count = 0;
    link l1Value = l1->head;
    link nextl1Value = NULL;
    link l2Value = l2->head;
    link nextl2Value = NULL;

    list newList = malloc(sizeof(list));
    newList->head = l1Value;

    while (l1Value != NULL && l2Value != NULL) {
        nextl1Value = l1Value->next;
        nextl2Value = l2Value->next;
        l1Value->next = l2Value;
        l2Value->next = nextl1Value;
        l1Value = nextl1Value;
        l2Value = nextl2Value;
        count ++;
```

1. Clean and concise code. Good variable names.

2. Correct usage of structs and pointer struct -> notation.

3. Understanding of pointers and structs. This understanding helped my create the zip and reverse function.