Compound Data – defining our own types!

1. Given the following definitions:

```
typedef struct date Date;
struct date {
    char month[4];  // 3 letter abbreviation of a month
    int day;        // valid day given month and year
    int year;       // valid year
};
```

   a. What is the output of the following code snippet?

```
Date d1;
Date d2 = { "Sep", 15, 1973};
printf("d1: %s %d, %d \n", d1.month, d1.day, d1.year);
printf("d2: %s %d, %d \n", d2.month, d2.day, d2.year);
```

```
d1: ??, ??
d2: Sep 15, 1973
```

   b. Assuming you cannot change the lines of code already written in part a, write the lines of code to follow that set `d1` to represent December 5, 2010.

```
strcpy(d1.month, "Dec");
d1.year = 2010;
d1.day = 5;
```

   c. What is the output of the following lines of code.
   Draw a diagram to help trace through the allocations, initializations and updates.

```
Date d = {"Sep", 15, 1973};
Date d_copy = d;
Date* d_ptr = &d;

printf("1: %d %d %d\n", d.year, d_copy.year, d_ptr->year);

d_ptr->year = 1986;

printf("2: %d %d %d\n", d.year, d_copy.year, d_ptr->year);

d_copy.year = 1792;

printf("3: %d %d %d\n", d.year, d_copy.year, d_ptr->year);

d.year = 2001;

printf("4: %d %d %d\n", d.year, d_copy.year, d_ptr->year);
```

```
1: 1973 1973 1973
2: 1986 1973 1986
3: 1986 1792 1986
4: 2001 1792 2001
```

2. You are really into music and you would like to design programs that allow you to keep track of your music collection. You have decided that there are three important pieces of data that describe a song: the title, the artist and the length of the song in seconds. Given the following symbolic constants:

```
#define MAX_STR_LEN 50
#define MAX_CHAR_ARRAY (MAX_STR_LEN+1)
```

a. Write the code to define this new data type:

```
typedef struct song Song;
struct song{
    char title[MAX_CHAR_ARRAY];
    char artist[MAX_CHAR_ARRAY];
    int length; // in seconds
};
```

b. Create and initialize an instance of your Song type with your song! Print out the fields to test your answer.

```
Song s = {"Let it be", "The Beatles", 330};
```

c. How would your Song type change if you decided it was important to track the release date of every song? Add this change to your answer in part a and update the code as necessary in part b.

```
typedef struct date Date;
struct date {
    char month[4]; // 3 letter abbreviation of a month
    int day;       // valid day given month and year
    int year;      // valid year
};

typedef struct song Song;
struct song{
    char title[MAX_CHAR_ARRAY];
    char artist[MAX_CHAR_ARRAY];
    int length; // in seconds
    Date release_date;
};

Song s = {"Let it be", "The Beatles", 330, {"May", 8, 1970}};
```

d. Given the new Song instance variable you created in part c, write the lines of code to **update** the length of the song to 347 seconds and the year of the release date to 2019 (don't do this by just changing the declaration and initial assignment)

```
s.length = 347;
s.release_date.year = 2019;
printf("%s, %s, %d on date: %s, %d, %d\n",
   s.title, s.artist, s.length,
   s.release_date.month, s.release_date.day, s.release_date.year);
```

3. Given the Song data type that you completed designing in Question 2c, write the following functions.
    a. Design a function called `create_song` that takes a title, artist, length of a song and a Date and creates and returns a new Song instance.

    b. Design a function called `print_song` that takes a pointer to a Song instance and prints all of the stored information about that song.

    c. Design a function called `add_time` that takes a pointer to a Song and a number representing additional time in seconds, the function should add the additional time to the length field of the given Song.

```c
#include <stdio.h>
#include <string.h>
#define MAX_STR_LEN 50
#define MAX_CHAR_ARRAY (MAX_STR_LEN+1)

typedef struct date Date;
struct date {
    char month[4]; // 3 letter abbreviation of a month
    int day;       // valid day given month and year
    int year;      // valid year
};

typedef struct song Song;
struct song{
    char title[MAX_CHAR_ARRAY];
    char artist[MAX_CHAR_ARRAY];
    int length; // in seconds
    Date release_date;
};

Song create_song(char title[], char artist[], int length, Date release);
void print_song(Song* song_ptr);
void add_time(Song* song_ptr, int time);

int main() {
    Date dt = {"May", 8, 1970};
    Song s = create_song("Let it be", "The Beatles", 330, dt);
    print_song(&s);
    add_time(&s, 100);
    print_song(&s);

    return 0;
}

/* Purpose: creates a Song instance with given data
 * Parameters: char title[], a valid string
 *             char artist[], a valid string
 *             int length,  >0
 *             Date release, date the song was released
 * Returns: Song, a copy of the created Song instance
 */
Song create_song(char title[], char artist[], int length, Date release) {
    Song s;
    strcpy(s.title, title);
    strcpy(s.artist, artist);
    s.length = length;
    s.release_date = release;
    return s;
}

/* Purpose: prints the fields of the Song pointed to by song_ptr
 * Parameters: Song* song_ptr
 * Returns: nothing
 */
void print_song(Song* song_ptr) {
    printf("%s, %s, %d on date: %s, %d, %d\n",
           song_ptr->title, song_ptr->artist, song_ptr->length,
           song_ptr->release_date.month, song_ptr->release_date.day,
           song_ptr->release_date.year);
}

/* Purpose: prints the fields of the Song pointed to by song_ptr
 * Parameters: Song* song_ptr
 *             int time, in seconds, >0
 * Returns: nothing
 */
void add_time(Song* song_ptr, int time) {
    song_ptr->length += time;
}
```

4. In your `main` declare an array that will hold 10 Song instances. Create 3 Song instances and store them at indices 0, 1 and 2 of the array respectively.

```
Song song_list[10];

Date dt1 = {"May", 8, 1970};
song_list[0] = create_song("Let it be", "The Beatles", 210, dt1);

Date dt2 = {"Oct", 22, 2015};
song_list[1] = create_song("Sorry", "Justin Bieber", 199, dt2);

Date dt3 = {"Jan", 21, 1981};
song_list[2] = create_song("Sunglasses at Night", "Corey Hart", 244, dt3);

print_songs(song_list, 3);

printf("\nadding 100 seconds to all songs\n");
add_time_to_all(song_list, 3, 100);
print_songs(song_list, 3);
```

5. Design a function that takes an array of songs and the number of songs in the array. The function should print the stored information for every song in the array. Make sure to test your function!

```
/* Purpose: prints the fields of the Song instances in songs
 * Parameters: Song songs[]
 *             int num_songs, number of Song instances in songs
 * Returns: nothing
 */
void print_songs(Song songs[], int num_songs) {
    int i;
    for(i=0; i<num_songs; i++) {
        print_song(&songs[i]);
    }
}
```

6. Design a function that takes an array of songs and the number of songs in the array and an amount of time in seconds. The function should and add the given amount of time to every song in the array. Make sure to test your function!

```
/* Purpose: adds given time to each Song instance in songs
 * Parameters: Song songs[]
 *             int num_songs, number of Song instances in songs
 *             int time, in seconds, >0
 * Returns: nothing
 */
void add_time_to_all(Song songs[], int num_songs, int time) {
    int i;
    for(i=0; i<num_songs; i++) {
        add_time(&songs[i], time);
    }
}
```

7. Why is it advantageous to pass a pointer to a Song in the functions `print_song` and `add_time`?

   Updates the existing instance, otherwise we wouldn't see the change to the song that was passed as it would be a copy that the function would be updating.
   Eliminates overhead of copying struct instances with when they are passed as arguments and returned from a function.

8. Would it be valid to change the `create_song` to return a pointer to the Song created within the function?

   No, you cannot return a pointer to a variable created in local scope of a function this way.