CSCI 320, Fall 2011, Homework 7
DUE: Thursday, November 3, 10 pm

1. Write C code that satisfies all of the following. Be sure to include lots of comments so that we can follow your logic. Make sure that your file includes your name!

2. Implement airport flight information data structure that contains the following data for the planes leaving on a given day:
- Flight number
- Originating airport (three characters)
- Destination airport (three characters)
- Departure time (four characters)

If this definition must be visible to the main method and other methods, place it before the start of the main method – this makes it globally accessible.

3. Allocate at least five flight information structures on the heap and store appropriate data in them. The following statement allocates enough space for a single record and returns a pointer to it in the variable new_ptr, which must be declared to be a pointer to the structure type described above.

```
/* Allocate space on heap for one structure, assign pointer to new_ptr */
struct flight_info * new_ptr = (struct flight_info *)malloc(sizeof(struct
flight_info));
```

```
/* Put data in structure */
new_ptr->flightNumber = 125;
(void)strcpy((* new_ptr).origAirport, "CHS"); /* copy a string */
(void)strcpy (new_ptr->destAirport, "CLT");
(void)strcpy (new_ptr-> departureTime, "1000");
```

In addition to `<stdio.h>`, include `<stdlib.h>` and `<string.h>` for malloc and strcpy.

Since, we cannot do input from within codepad, creating multiple structures requires repeating the same code with different values over and over. Just cut and paste and change the data.

4. Build two linked lists of pointers to flight information structures.
- The first list references the structures in ascending order according to flight number.
- The second list references the structures in ascending order according to flight time.

Notes on #4:
- There is only one structure containing information for a specific flight. But, both lists have nodes that reference that information. The list nodes don't store structure, but pointers to structures.
- When new flight information structure is created, both lists are traversed and the links are placed appropriately. **Write one or more insert method(s), to add pointers to the new data to the lists.**
- To simplify your "insert" method, you may design your linked lists to include a dummy first node, so that you are never inserting into an empty list.

5. Include code to display the contents (flight information referenced) from your lists.
6. Copy your code into a text file and submit that text file. The program should be completely self-contained and not require external input.

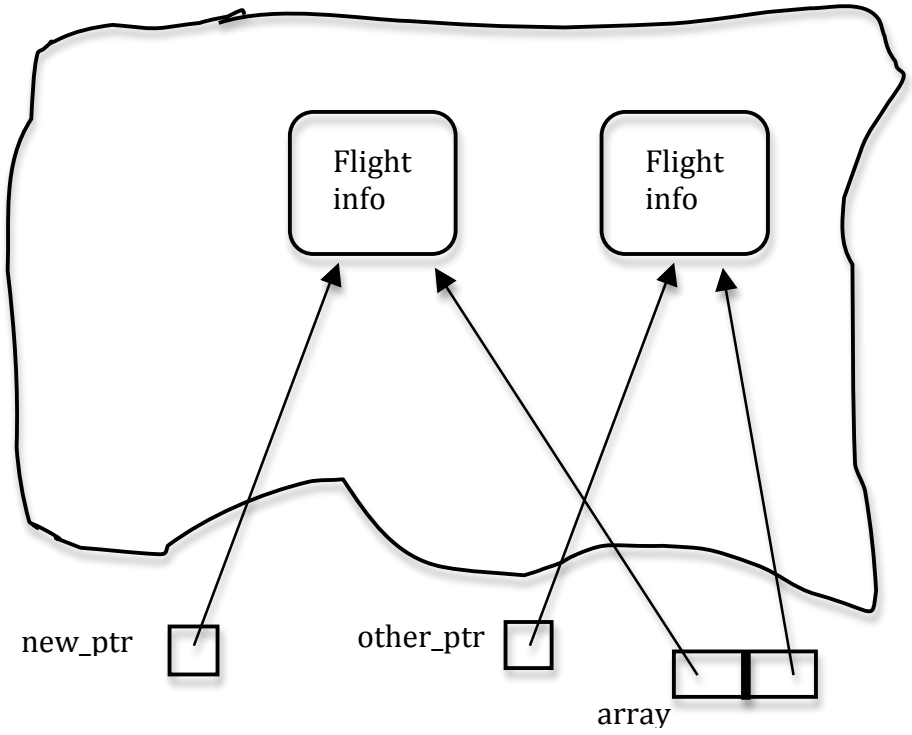
Collaboration policy in effect for this assignment:

- You may only submit solutions generated by your own work.
- You may discuss C with classmates and instructors.
- You may discuss these problems with classmates and instructors. If it helps you to learn from others how to solve the problem, that is fine. The objective is that you CAN and DO solve the problem.
- You MAY NOT copy the work of others. You MAY NOT give your work to others.
- You may not look for solutions on the internet, you should craft your own.

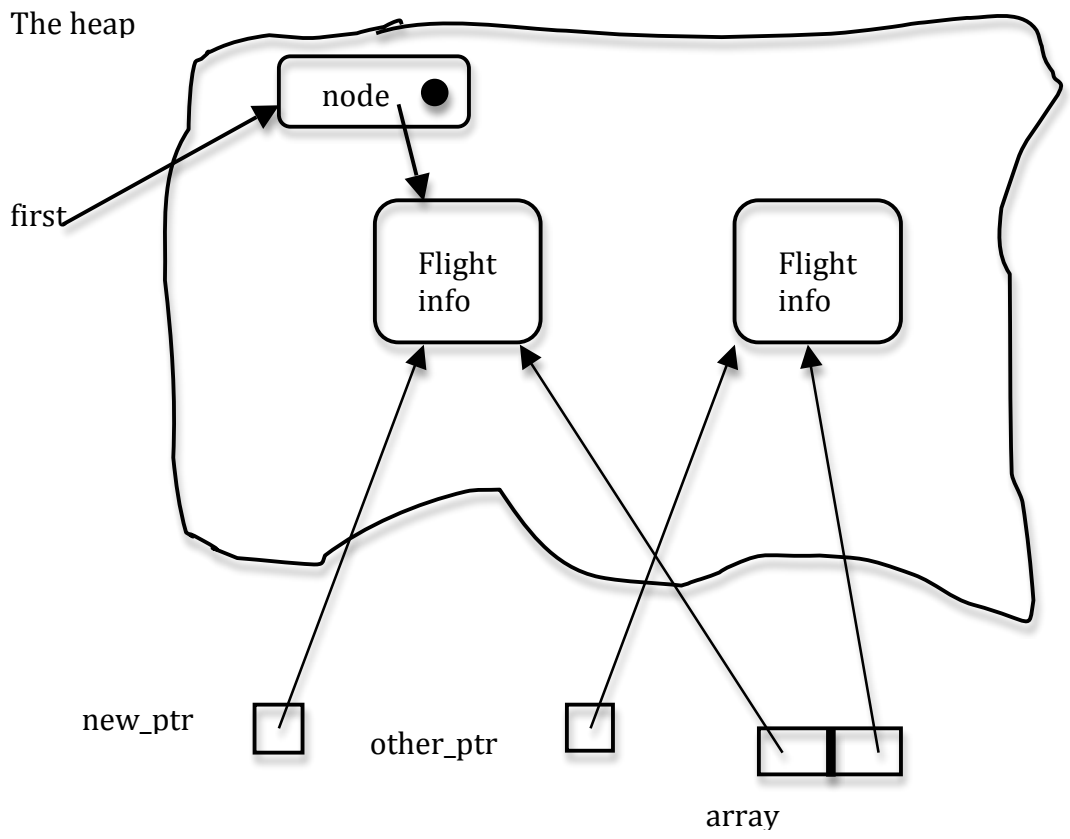
If structures are allocated, through "malloc" they are stored on the heap, and pointers to them are stored outside of the heap (on the run-time stack if the pointers are declared within the main or any other method. (I'll just draw the stuff not on the hope outside of the bubble.)

```
struct flight_info *new_ptr =(struct flight_info *)malloc(sizeof(struct
flight_info));

/* statements to store data in the structure */

struct flight_info *other_ptr =(struct flight_info *)malloc(sizeof(struct
flight_info));

/* statements to store data in the structure */

/* An array of pointers to flight info */

struct flight_info *array[2];

array[0] = new_ptr; array[1] = other_ptr;

/* A linked list of pointers to structures – since you may need any number of
nodes in the linked list – those nodes should be allocated on the heap. If you
needed a fixed number, you might allocate them on the stack.*/
```

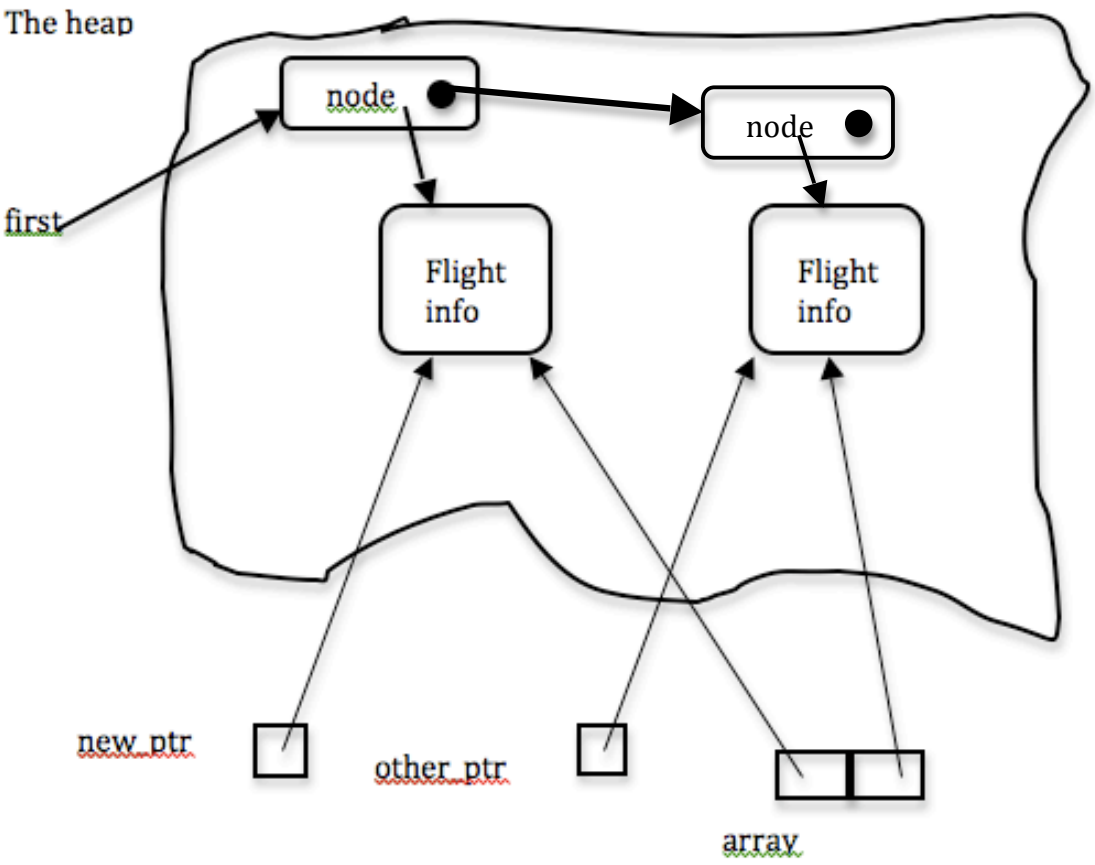
The heap



```
/* Nodes in the linked list will look like. */
struct node {           /* a node in the linked list */
         struct flight_info *data;
         struct node *next;
}
/* generate a node */
struct node *node_ptr =(struct node *)malloc(sizeof(struct node));
/* add data */
node_ptr->data = new_ptr;
node_ptr->next = NULL;
/* Create a list pointer */
struct node first = node_ptr; /* initialize */
```



Then, if a second node is added to the end of the list, and that second node points to the other flight info structure, the list looks something like:


The heap
node
node
first
Flight info
Flight info
new_ptr
other_ptr
array