13.6 Practice Programs

Sample solutions to these problems will be included by the beginning of 2013 at the following website:

http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3223/book/

For all the problems in this section, use the following struct to represent a node in a linked list:

```
struct node {
    int value;
    struct node* next;
};
```

1) Write a function that takes in a pointer to the front of a linked list and returns the number of values in the linked list. The function prototype is provided below:

```
int num_nodes(struct node* front);
```

2) Write a function that takes in a pointer to the front of a linked list and returns the sum of the values in the linked list. The function prototype is provided below:

```
int sum_nodes(struct node* front);
```

3) Write a function that takes in a pointer to the front of a linked list and returns the sum of every other value in the list, starting with the first. Make sure your function doesn't commit any NULL pointer errors. The function prototype is provided below:

```
int sum_every_other_node(struct node* front);
```

4) Write a function that takes in a pointer to the front of a linked list and returns the sum of the values in the linked list that are divisible by 7. The function prototype is provided below:

```
int sum_nodes_div7(struct node* front);
```

5) Write a function that takes in a pointer to the front of a linked list, moves the first node of the list to the end of the list and returns a pointer to the new front of the list. If the list contains zero or one element, no changes need to be made and the original front should be returned. The function prototype is provided below:

```
struct node* front_to_back(struct node* front);
```

6) Write a function that takes in a pointer to the front of a linked list, moves the last node of the list to the front of the list and returns a pointer to the new front of the list. If the list contains zero or one element, no changes need to be made and the original front should be returned. The function prototype is provided below:

```
struct node* back_to_front(struct node* front);
```

7) Write a function that takes in a pointer to the front of a linked list and an integer, and creates a node storing that integer and inserts it into the back of the given linked list, returning a pointer to the front of the resulting list. The function prototype is provided below:

```
struct node* insert_back(struct node* front);
```

8) Write a function that takes in a pointer to the front of a linked list and deletes the first node in the list returning a pointer to the rest of the list. If the original list is empty, do nothing and return NULL. The function prototype is provided below:

```
struct node* delete_front(struct node* front);
```

9) Write a function that takes in a pointer to the front of a linked list and deletes the last node in the list returning a pointer to the front of the list. If the original list is empty, do nothing and return NULL. The function prototype is provided below:

```
struct node* delete_last(struct node* front);
```

10) Write a function that takes in pointers to two linked lists that are stored in increasing numerical order and prints out the values that are common to both lists. The function prototype is provided below:

```
struct node* print_common_vals(struct node* lista,
                               struct node* listb);
```