MCS 360 Exercise Set #2
(to be turned in Tuesday, Sep 18, at the discussion section)

1. Consider the list in the handout *Linked Structures in C*. There are some operations that we cannot perform efficiently if the list is large. For example,
   i) Print out the names in the list in reverse order.
   ii) Given a pointer \( p \) to a node in the list, insert a new node *before* node \( p \),
   iii) Given a pointer \( p \) to a node in the list, delete node \( p \).

We can perform each of these operations efficiently if we add one more field to our `Node` structure — a pointer to the previous node in the list (NULL for the first node). We will also need a pointer, call it `tail`, to the last node of the list, just as we already have a pointer `head` to the first node. It makes sense to place `head` and `tail` in a structure, say called `DList` (for doubly-linked list). The code in the handout and the example can be replaced by:

```c
struct Node {
    char *name;
    struct Node *next;          /* Points to next node in list */
    struct Node *previous;      /* Points to previous node in list */
};
typedef struct Node Node;

Node *get_node( char *name, Node *next, Node *previous) {
    Node *p = (Node *)checked_malloc(sizeof(Node));
    p->name = (char *)checked_malloc(strlen(name)+1));
    strcpy( p->name, name);
    p->next = next;
    p->previous = previous;
    return p;
}

struct DList {
    Node *head;          /* Points to first node in list */
    Node *tail;          /* Points to last node in list */
};
typedef struct DList DList;
```

![Diagram of doubly-linked list structure]
Write C language code for

a) A function

    void print_reverse( DList *list);

called to print the names in the list in reverse order.

b) A function

    void insert_before( Node *p, char *name, DList *list);

called to insert a new node containing a copy of name before node p of list.

c) A function

    void remove( Node *p, DList *list);

called to remove Node p from the list, and free it.

In each case above, you may assume that list points to a valid list structure, such as illustrated above. In (b) and (c), you may assume that p does point to some node in the list. The case in which this node is the first or last node may need special handling. You may use any functions from MCS360.c, or from the ANSI C library.

2. Do Exercise E1, parts (b) and (c), in the exercises for Section 3.1 (page 88)