```c
/**
 * linked.c
 *
 * week 7 section
 * fall 2013
 *
 * practice using linked lists
 */

#include <cs50.h>
#include <stdbool.h>
#include <stdio.h>

// typedef a node for the linked list
typedef struct node
{
    int n;
    struct node* next;
}
node;

// function prototypes
bool insert_node(int value);
void print_nodes(node* list);
void free_nodes(node* list);

// global variable for the head of the list
node* head = NULL;

int main(void)
{
    // offer the user two options
    while (true)
    {
        printf("Please choose an option (0, 1, 2): ");
        int option = GetInt();

        switch (option)
        {
            // quit
            case 0:
                free_nodes(head);
                printf("Goodbye!\n");
                return 0;

            // insert int into linked list
            case 1:
                printf("Please enter an int: ");
```

```c
                int v = GetInt();
                char* success = insert_node(v) ? "was" : "was not";
                printf("The insert %s successful.\n", success);
                break;

            // print all ints
            case 2:
                print_nodes(head);
                break;

            default:
                printf("Not a valid option.\n");
                break;
        }
    }
}

/**
 * Create a new node for a given value and insert it into a list.
 */
bool insert_node(int value)
{
    // TODO
    return false;
}

/**
 * Print out all of the ints in a list.
 */
void print_nodes(node* list)
{
    // TODO
}

/**
 * Frees all of the nodes in a list upon exiting the program.
 */
void free_nodes(node* list)
{
    // TODO
}
```

```c
/**
 * linked_completed.c
 *
 * week 7 section
 * fall 2013
 *
 * practice using linked lists
 */

#include <cs50.h>
#include <stdbool.h>
#include <stdio.h>

// typedef a node for the linked list
typedef struct node
{
    int n;
    struct node* next;
}
node;

// function prototypes
bool insert_node(int value);
void print_nodes(node* list);
void free_nodes(node* list);

// global variable for the head of the list
node* head = NULL;

int main(void)
{
    // offer the user two options
    while (true)
    {
        printf("Please choose an option (0, 1, 2): ");
        int option = GetInt();

        switch (option)
        {
            // quit
            case 0:
                free_nodes(head);
                printf("Goodbye!\n");
                return 0;

            // insert int into linked list
            case 1:
                printf("Please enter an int: ");
```

```c
                int v = GetInt();
                char* success = insert_node(v) ? "was" : "was not";
                printf("The insert %s successful.\n", success);
                break;

            // print all ints
            case 2:
                print_nodes(head);
                break;

            default:
                printf("Not a valid option.\n");
                break;
        }
    }
}

/**
 * Create a new node for a given value and insert it into a list.
 */
bool insert_node(int value)
{
    // malloc space for a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
        return false;
    }
    new_node->n = value;

    // first insertion
    if (head == NULL)
    {
        new_node->next = head;
        head = new_node;
        return true;
    }

    // keep track of current and previous nodes
    node* curr = head;
    node* prev = NULL;

    while (curr != NULL)
    {
        // don't insert duplicates
        if (value == curr->n)
        {
            free(new_node);
```

```
 97.             return false;
 98.         }
 99.
100.         // keep looking for correct spot
101.         else if (value > curr->n)
102.         {
103.             prev = curr;
104.             curr = curr->next;
105.
106.             if (curr == NULL)
107.             {
108.                 break;
109.             }
110.         }
111.
112.         else if (value < curr->n)
113.         {
114.             break;
115.         }
116.     }
117.
118.     // insert the node in the correct position
119.     new_node->next = curr;
120.     if (prev == NULL)
121.     {
122.         head = new_node;
123.     }
124.     else
125.     {
126.         prev->next = new_node;
127.     }
128.
129.     return true;
130. }
131.
132. /**
133.  * Print out all of the ints in a list.
134.  */
135. void print_nodes(node* list)
136. {
137.     // save a counter
138.     int counter = 0;
139.
140.     node* curr = head;
141.     while (curr != NULL)
142.     {
143.         counter++;
144.         printf("Node %d: %d\n", counter, curr->n);
```

```c
        curr = curr->next;
    }
}

/**
 * Frees all of the nodes in a list upon exiting the program.
 */
void free_nodes(node* list)
{
    // don't lose the rest of the list
    node* curr = head;
    node* prev = NULL;

    while (curr != NULL)
    {
        prev = curr;
        curr = curr->next;
        free(prev);
    }
}
```