```c
/* inventory.c
 * Christopher Brant
 * cbrant
 * ECE 2230
 * Section 001
 * Spring 2017
 * Programming Assignment #1
 * Due on 1/23/17 at 11:30 PM
 * Professor Walt Ligon
*/

#include <stdio.h>
#include <stdlib.h>
#include "inventory.h"


struct inventory *inventory_create()
{
        int i; // Used as a counter

        // Malloc space for a struct inventory
        struct inventory *inv = (struct inventory *)malloc(sizeof(struct inventory)
);

        // Initialize the inventory cursor to 0
        inv->cursor = 0;

        // Set all inventory item slot pointers to NULL
        for (i = 0; i < ISIZE; i++)
                inv->slot[i] = NULL;

        return inv;
}

int inventory_add(struct inventory *inv, struct inventory_item *invitem)
{
        int i, success, opening = 0; // i is a counter, success is return error val
ue
                                                                // opening is used
 to signify when an open slot is found
        inv->cursor = 0;

        /*The following algorithm finds the first open slot in the
          inventory and inserts a new item into said slot. */
        if (inv->slot[inv->cursor] == NULL)
                inv->slot[inv->cursor] = invitem;
        else
        {
                i = inv->cursor;

                while (opening == 0 && i < ISIZE)
                {
                        if (inv->slot[i] == NULL)
                        {
                                inv->slot[i] = invitem;
                                inv->cursor = i;
                                opening = 1;
                        }
                        else
                                i++;
                }

                if (i == ISIZE && inv->slot[inv->cursor] == NULL)
                        success = -1;
        }

        // Error checking and error integer assignment
        if (inv->slot[inv->cursor] == invitem)
                success = 0;
        else
                success = -1;

        return success;
}

struct inventory_item *inventory_lookup(struct inventory *inv, int key)
{
        int i = 0; // counter
        struct inventory_item *found = NULL; // pointer to be returned

        /* Searches first for non NULL pointers and then if the found
           non NULL pointer contains the desired item, and returns its pointer */
        while (found == NULL && i < ISIZE)
        {
                if (inv->slot[i] != NULL && inv->slot[i]->item_key == key)
                {
                        inv->cursor = i;
                        found = inv->slot[i];
                }
                else
                        i++;
        }

        // If not found this will return a NULL pointer
        return found;
}

int inventory_delete(struct inventory *inv, int key)
{
        int returnerr, i = 0; // return error value and i is a counter
        int deleted = 0;          // deleted signifies when the item is correctly d
eleted

        // Searches for non NULL inventory slot with correct item key and frees the
 memory from that slot
        while (deleted == 0 && i < ISIZE)
        {
                if (inv->slot[i] != NULL && inv->slot[i]->item_key == key)
                {
                        inv->cursor = i;
                        free(inv->slot[i]);
                        inv->slot[i] = NULL;
                        deleted = 1;
                }
                else
                        i++;
        }

        if (deleted == 0)
                returnerr = -1;
        else
                returnerr = 0;

        return returnerr;
}

struct inventory_item *inventory_first(struct inventory *inv)
{
        inv->cursor = 0;

        // Returns the address of the first non NULL pointer in the inventory
        struct inventory_item *first = inv->slot[inv->cursor];  // Used as return v
alue pointer
```

```
        while (first == NULL && inv->cursor < ISIZE)
                first = inv->slot[inv->cursor++];

        if (inv->cursor == ISIZE)
                first = NULL;

        return first;
}

struct inventory_item *inventory_next(struct inventory *inv)
{
        int i = inv->cursor + 1;                // counter variable
        struct inventory_item *next;    // Return value pointer

        // Searches for the next non NULL pointer in the inventory and returns its
address
        while (i < ISIZE && inv->slot[i] == NULL)
                i++;

        if (i == ISIZE && inv->slot[i-1] == NULL)
                next = NULL;
        else if (inv->slot[i] != NULL)
        {
                next = inv->slot[i];
                inv->cursor = i;
        }

        return next;
}

int inventory_destroy(struct inventory *inv)
{
        int i, returnerr = 0;   // counter variable and return error value

        // Frees and NULLs out all non NULL inventory slots
        for (i = 0; i < ISIZE; i++)
        {
                if (inv->slot[i] != NULL)
                {
                        free(inv->slot[i]);
                        inv->slot[i] = NULL;
                }
        }

        // Frees and NULLs out the overall inventory.
        free(inv);
        inv = NULL;

        if (inv != NULL)
                returnerr = -1;

        return returnerr;
}
```