```c
// Name: Rodrigo Ignacio Rojas Garcia
// Course Number: ECE 2230
// Section: 001
// Semester: Spring 2017
// Assignment Number: 1

// Libraries Declaration Section
#include "inventory.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


// Functions Declaration Section

// Function creates an inventory structure variable pointer, then the function call
oc is used to allocate memory for this structure pointer
// finally, the address of the allocated memory is returned.
struct inventory *inventory_create()
{
    struct inventory *created_inventory;
    created_inventory = (struct inventory*)calloc(1,sizeof(struct inventory));
    return created_inventory;
};

// Function is passed the address of the inventory and the address of an item that
wants to be added. The function check first check if the key_data
// entered by the user has been entered before so the same data is not repeated. If
 no repetition exits, memory is allocated for the item and
// a 0 is returned. If there was no space available in the invetory or the key_data
 was repeated, a -1 is returned.
int inventory_add(struct inventory *inventory_pointer, struct inventory_item *item)
{
    // Variable Declaration Section
    int c1; // Used as a counter in for loop
    int added = -2; // Used to store if item has been added to inventory
    int repetition = 0; // Used to check that the key_data has not been repeated

    // For loop will run until it reaches the the maximum number of items or it is
breaked from
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        // If statement checks if dynamic memory has been allocated for an item
        if (inventory_pointer->allocated_slots[c1] == 1)
        {
            // If dynamic memory has been allocated for an item, it will check if t
he item_key match, if so recognizes repetition of item and breaks from loop
            if (inventory_pointer->items[c1]->item_key == item->item_key)
            {
                repetition = -1;
                break;
            }
        }
    }
    // If there were no repetitions of the item_key, then this if statement will ru
n
    if (repetition == 0)
    {
        for (c1 = 0; c1 < MAXITEMS; c1++)
        {
            // If there is no dynamic memory allocated, it will allocate dynamic me
mory for the item and will assign 1 to allocated_slots meaning it has an item
            if (inventory_pointer->allocated_slots[c1] == -1)
            {
                inventory_pointer->items[c1] = item;
                inventory_pointer->allocated_slots[c1] = 1;
                added = 0;
                break;
            }
        }
    }

    // If data was succsefully added, it will return a 0
    if (added == 0)
    {
        return 0;
    }

    // If data was not successuflly added, it will return a -1
    else if (repetition == -1 || added != 0)
    {
        return -1;
    }

    return 0;
};

// Functions is passed the address of an inventory structure and the key data numbe
r that wants to be looked up. The function then procees to
// check which item slots have been allocated with memory and which ones have not.
If the item slot has been allocated it will seach if the
// item_key matches with the key_data passed and if so it will store the location o
f it into variable location and break from the loop.
// After the breaking from the loop, it will return the address of the item. If no
item found with matching key_data a NULL is returned.
struct inventory_item *inventory_lookup(struct inventory *inventory_pointer, int ke
y_data)
{
    // Variable Declaration Section
    int c1;
    int found = -1;
    int location = -1;

    // For loop will run until c1 reaches the number greater than MAXITEMS or it is
 breaked from
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        // If statement will look in allocated slots that have been allocated and w
ill see if key_data entered is found in this slots
        if (inventory_pointer->allocated_slots[c1] == 1 && inventory_pointer->items
[c1]->item_key == key_data)
        {
            found = 0;
            location = c1;
            break;
        }
    }

    // If the key_data is found, the address of the item is returned
    if (found == 0)
    {
        return inventory_pointer->items[location];
    }
    // If the key_data is not found, the function returns a NULL
    else
    {
        return NULL;
    }
};

// Functions is passed the address of an inventory and the number of a key_data. Th
e function uses the key_data number passed to search for an item_key
// in one of the item slots that have been allocated for a match. If a match is fou
nd, the allocated memory is freed and a 1 is returned. If no item was
```

```c
// found, a -1 is returned.
int inventory_delete(struct inventory *inventory_pointer, int key_data)
{
    // Variable Declaration Section
    int c1;
    int found = -1;

    // For loop will run until c1 reaches the number greater than MAXITEMS or it is
 breaked from
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        // If statement will look in allocated slots that have been allocated and w
ill see if key_data entered is found in this slots
        if (inventory_pointer->allocated_slots[c1] == 1 && inventory_pointer->items
[c1]->item_key == key_data)
        {
            found = 0;
            break;
        }
    }

    // If the key_data was found, the array pointer items will free the memory that
 it was pointed to and slot will be set to not allocated
    if (found == 0)
    {
        free(inventory_pointer->items[c1]);
        inventory_pointer->allocated_slots[c1] = -1;
        return 0;
    }
    // If key_data not found, returns a -1
    else
    {
        return -1;
    }

    return 0;
}

// Function is passed the address of an inventory which is used to search for the f
irst address of an item that has been allocated with memory
// and stored. If the first item is found, the address is returned. If nothing is f
ound, a NULL is returned.
struct inventory_item *inventory_first(struct inventory *inventory_pointer)
{
    // Variable Declaration Section
    int c1 = 0;
    int found = -1;

    // For loop looks for a slot that can be allocated with memory and if so sets f
ound to 0 and sets the first_item in inventory_pointer struct to that slot and brea
ks
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        if (inventory_pointer->allocated_slots[c1] == 1)
        {
            found = 0;
            inventory_pointer->first_item = c1;
            break;
        }
    }

    // If the first allocated slot is found, it returns the address
    if (found == 0)
    {
        return inventory_pointer->items[c1];
    }
    // If there are no slots, it returns NULL
    else
    {
        return NULL;
    }
}

// Functions is passed the address of an inventory which is then used for search fo
r the next items in the inventory and pass their adddresses. If
// items are found, the address is returned. If no item found, a NULL is returned.
struct inventory_item *inventory_next(struct inventory *inventory_pointer)
{
    // Variable Declaration Section
    static int count = 0;
    int found = -1;
    int slot;

    // If statement will check if there is another slot with allocated memory and s
ee that it does not matches with the first_item found before
    if (inventory_pointer->allocated_slots[count] == 1 && count != inventory_pointe
r->first_item)
    {
        found = 0;
        slot = count;
        count++;
    }
    else
    {
        found = -1;
        slot = count;
        count++;
    }

    // If the count searches as many items as it can, it will be set to 0
    if (count == MAXITEMS)
    {
        count = 0;
    }

    // If there is another item found, it will return the address
    if (found == 0)
    {
        return inventory_pointer->items[slot];
    }
    // If no other items found, returns a NULL
    else
    {
        return NULL;
    }

}

// Function is passed the address of an inventory which is used to free all allocat
ed memory used to store items entered by the user.
// The function will search for only spots that have been allocated and free the me
mory and then check that each of them have been freed
// by checking the allocated_slots array. If all allocated_slots array have a -1 in
 their slots, it means that there is no memory
// allocated in those slots for items, if there is not memory allocated count incre
ases each time. If the count matches the number
// of MAXITEMS it will return a 0 meaning that all slots have been freed, if not a
-1 is returned
int inventory_destroy(struct inventory *inventory_pointer)
{
    // Variable Declaration Section
    int c1;
    int count = 0;
    // For loop will run until c1 reaches the number greater than MAXITEMS or it is
 breaked from
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        // If statement will look in allocated item slots that have been allocated
and will see if key_data entered is found in this slots
        if (inventory_pointer->allocated_slots[c1] == 1)
        {
            free(inventory_pointer->items[c1]);
            inventory_pointer->allocated_slots[c1] = -1;
        }
    }

    // For loop will check that all allocated item slots have been freed
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        if (inventory_pointer->allocated_slots[c1] == -1)
        {
            count++;
        }
    }

    // If all allocated  item slots have been free, function will return 0
    if (count == MAXITEMS)
    {
        return 0;
    }
    // If there was an error freeing item slots, function will return -1
    else
    {
        return -1;
    }

    return 0;
}
```