```
// Name: Rodrigo Ignacio Rojas Garcia
// Course Number: ECE 2230
// Section: 001
// Semester: Spring 2017
// Assignment Number: 1.5

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


// Functions Declaration Section

// Function will create a structure of an inventory and make it a pointer which the
n will be allocated with memory and set the location of cursor to -1 to signify tha
t it has not been used before. Then
// it will set each item pointer within the inventory to NULL and it will return th
e address of the dynamic memory allocated
struct inventory *inventory_create()
{
    // Variable Declaration Section
    struct inventory *inventory_pointer; // Used to allocate memory to an inventory
 and returned
    int c1; // Used as a counter in loop

    // Allocates dynamic memory for inventory structure which will later be returne
d for the address of the allocated memory
    inventory_pointer = (struct inventory*)calloc(1, sizeof(struct inventory));

    // Sets the location to -1
    inventory_pointer->location = -1;

    // Loop will set each item pointer to NULL to signify that the pointer has not
been allocated with dynamic memory
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        inventory_pointer->items[c1] = NULL;
    }

    return inventory_pointer;
}

// Function will first look if there has been an item that has been allocated with
memory, if so, it will see if the key_data has already been entered. If true, it wi
ll determine that a repetition has
// occurred and will return a NULL. If there is no item that has the same key_data
and there is space to allocate a new item, then it will set the pointer of the item
 to point to the inventory_item
// item pointer allocated memory and will return a 0, if not it will return a NULL.
int inventory_add(struct inventory *inventory_pointer, struct inventory_item *item)
{
    // Variable Declaration Section
    int c1;
    int success = -2;
    int repetition = -2;

    // For loop will see which items in the inventory have been allocated and will
find if the key_data entered by the user matches a key_data already entered.
    // If the key_data has been already entered, the loop will set repetition to -1
, and will break from loop, otherwise it will keep going until reaching value of MA
XITEMS
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        if (inventory_pointer->items[c1] != NULL)
        {
            if (inventory_pointer->items[c1]->item_key == item->item_key)
            {
                repetition = -1;
                break;
            }
        }
    }

    // If the key_data has not been entered, then the if statemenet will run and se
e in the For loop which items in the array have not yet been allocated with dynamic
 memory
    // If there is an item without allocated memory, it will set the address alloca
ted in lab1.c into the inventory item, set success to 0, and break from loop.
    if (repetition == -2)
    {
        for (c1 = 0; c1 < MAXITEMS; c1++)
        {
            if (inventory_pointer->items[c1] == NULL)
            {
                inventory_pointer->items[c1] = item;
                success = 0;
                break;
            }
        }
    }

    // If the addition of the item was successfull, the function will return a 0
    if (success == 0)
    {
        return 0;
    }

    // If the addition of the item was unsuccesfull by either having the same key_d
ata or there not being enough space in the inventory, it will return a -1
    else
    {
        return -1;
    }

    return 0;
};

// Function will first set the location of the inventory to -1 to signify that ther
e has not been an item located yet. Then it will go through a loop looking for an i
tem that has been
// allocated with memory and if so will see if the key_data passed matches the item
_key of any item in the array, if so it will return the address of this item, if no
t it will return
// NULL to signify that there was no item with that item_key
struct inventory_item *inventory_lookup(struct inventory *inventory_pointer, int ke
y_data)
{
    // Variable Declaration Section
    int c1;
    int found = -2;

    // Sets location of cursor to -1 each time to signify that nothing has been loc
ated yet
    inventory_pointer->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->items[c1] != NULL && inventory_pointer->items[c1]->i
```

```
tem_key == key_data)
        {
            found = 0;
            inventory_pointer->location = c1;
            break;
        }
    }

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

// Function will first the location of the inventory to -1 to signify that no locat
ion has been used. Then it goes through a loop that will see if an item that has be
en allocated
// with memory and if so, will see if the item matches with the key_data entered. I
f so, it will break from loop and will set that pointer to NULL and return 0. If no
 item found
// it returns a -1 signifying that there is no item in the inventory with that key_
data
int inventory_delete(struct inventory *inventory_pointer, int key_data)
{
   // Variable Declaration Section
    int c1;
    int found = -2;

    // Sets location of cursor to -1 each time to signify that nothing has been loc
ated yet
    inventory_pointer->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->items[c1] != NULL && inventory_pointer->items[c1]->i
tem_key == key_data)
        {
            found = 0;
            inventory_pointer->location = c1;
            break;
        }
    }

    // If the item with matching key_data is found, it will set the item pointer to
 NULL and will return 0 to signify sucess
    if (found == 0)
    {
        inventory_pointer->items[inventory_pointer->location] = NULL;
        return 0;
    }
    // If an items with a matching key_data was not found, it will return -1 to sig
nify that no item was found
    else
    {
        return -1;
    }
};

// Function will go through a loop searching for the first item in the item array a
nd if found it will break from loop and return the address of the item. If no item
is in inventory
// it will return NULL.
struct inventory_item *inventory_first(struct inventory *inventory_pointer)
{
    // Variable Declaration Section
    int c1; // Used as a counter in loop
    int location; // Used to store the index to locate the item
    int found = -2; // Used to sifnify if an item was found or not.

    // For loop will run unti it reaches the number of MAXITEMS while trying to loc
ate the first item that has been allocated with memory in the inventory
    for (c1 = 0; c1 < MAXITEMS; c1++)
    {
        // If there was an item found, it will break from loop and save the index o
f the item into location and set found to 0
        if (inventory_pointer->items[c1] != NULL)
        {
            found = 0;
            location = c1;
            break;
        }
    }

    // If the first item was found in the inventory, it will set the location in in
vtory to the next index that follows and return the address of the found item
    if (found == 0)
    {
        inventory_pointer->location = location;
        return inventory_pointer->items[location];
    }

    // If there were no items located, it will return a NULL
    else
    {
        return NULL;
    }
};

// Function will go through a loop which will see if there is another item in the a
rray that has been allocated with memory and if so it will break from loop and reut
rn the address
// of the item. If no another item in the array it will return a NULL
struct inventory_item *inventory_next(struct inventory *inventory_pointer)
{
    // Variable Declaration Section
    int c1; // Used as a counter in loop
    int location; // Used to save the index number of array
    int found = 2; // Used to signify if an item has been found or not

    // For loop will run until the number of MAXITEMS is reached while searching fo
r the next item that has been allocated with dynamic memory
    for (c1 = inventory_pointer->location + 1; c1 < MAXITEMS; c1++)
    {
        // If an item has been allocated with memory, it will set found to 0, save
the index of the item into location and will break from the loop
        if (inventory_pointer->items[c1] != NULL)
        {
            found = 0;
            location = c1;
            break;
        }
    }

    // If the next item was found, it will set the location in the inventory to the
```

```
 next index in the item array and return the address of the next item that was foun
d
    if (found == 0)
    {
        inventory_pointer->location = location;
        return inventory_pointer->items[location];
    }

    // If no next item was found, it will return a NULL
    else
    {
        return NULL;
    }
}

// Function inventory_destroy will set inventory_pointer to NULL, and if successful
l, it will return a 0, otherwise a -1 will be returned
int inventory_destroy(struct inventory *inventory_pointer)
{
    free(inventory_pointer);
    return 0;
}
```