Jonathan Liao

#### CS 305 HW 1 Report

**1. Questions (include these in your write-up):**

1a.

Meets all specifications

1b.

output.txt

Done

1c.

test\_case1-3 have been included in zip file with details.

2a.

Because after the function is done executing, any modifications to the object are not saved. For example, we do not want to permanently change the cost value of cereal or the name of a large t-shirt because those values are user defined and should be kept constant.

2b.

Using the pointer to the struct's data value changes it permanently. Receipt objects for example CAN be modified if they are inside the receipt items[] array. This is in case we want to remove them from the receipt or change the name of it. To access the n'th retail\_item element of a receipt's item array, you'd have to make this call:

receiptStruct->items[nth]

Where -> is used instead of the conventional '.'

3a. In the run\_test function, you will see comments about drawing picture 1 and picture 2. Draw the picture of the shop object at the point where the comment point to draw picture 1 is located. Use arrows to indicate pointers. The picture should include all components of the shop object.

Shop array

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Candy | Coffee | Cookie | Shirt | Eggs | Eggs | Oranges | Bread | Grapes | Yogurt | ‘\0’ |

\*shop

3b. Draw the picture of shop at the point of execution where draw picture 2 is located in run\_test.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Candy | Coffee | Cookie | Shirt | Eggs | Oranges | Bread | Grapes | Yogurt | NULL | ‘\0’ |

\*shop

4a.

One of the hardest things of C is the lack of a String data type or object. It was very frustrating to perform the simple concatenation of extra whitespace (or truncation of the word is too long) on the struct name. Instead of concatenation shortcuts that were taken for granted in Java, C forced me to think to the barebones of how computer science works, which was rewarding.

4b.

C is not object oriented, but the use of structs and objects are very similar. My knowledge of OO in Java was useful for this assignment and made the use of structs easy. Minor differences, for example, included method of declaration and getters and pointers. In Java, there are explict methods to be written to return object data values whereas C uses pointers only.

5a.

|  |  |  |
| --- | --- | --- |
| Compiler error message | What it means | How I resolved it |
| Seg fault Core Dumped | Accessing memory that's not valid. | Mallocing an array of a negative size is illegal because when it was time to assign elements, the compiler was assigning items to a null array. Causing it to crash. Thus, I made the code check before mallocing a negative size. |
| ';' expected | Missing something that is expected for complete syntax | Added ';' where appropriate and directed. |

5b.

This homework assignment took a lot longer than last one. At least twice as long, so about 6 hours plus the writing.

**Appendix A:** I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files.

**Appendix B**:

**retail\_item.h**

#ifndef RETAIL\_ITEM\_H

#define RETAIL\_ITEM\_H

#define MAX\_NAME\_LENGTH 16

/\* struct of retail\_item and its data values \*/

struct retail\_item {

int number;

char name[MAX\_NAME\_LENGTH + 1];

int price\_cents;

};

/\* here is an example of defining a point struct with a typedef;

\* delete this in your final code

typedef struct point {

double x;

double y;

} point;

\*/

/\* function prototypes -- should match your .c implementation \*/

struct retail\_item create\_retail\_item (int num, char \*name, int price);

void print(struct retail\_item ri);

#endif

**retail\_item.c**

#include <stdio.h>

#include "retail\_item.h"

#include "receipt.h"

#include <string.h>

#include <stdlib.h>

//create retail\_item struct

struct retail\_item create\_retail\_item(int num, char\* nam, int price) {

struct retail\_item item;

//concatenate white spaces to names less than 16 char.

//or truncate if name

int i = 0;

while(\*(nam+i) && i < MAX\_NAME\_LENGTH) {

item.name[i] = \*(nam+i);

i++;

}

while(i < MAX\_NAME\_LENGTH) {

item.name[i] = ' ';

i++;

}

item.name[i] = '\0';

if(price >= 0) {

item.price\_cents = price;

}

else {

item.price\_cents = 0;

}

if(num >= 0) {

item.number = num;

}

else {

item.number = 0;

}

return item;

}

//print item data values

void print(struct retail\_item s) {

printf("Item %d\t", s.number);

printf("%s\t", s.name);

printf("$%.2f\n", (float) s.price\_cents/100);

}

**receipt.h**

#ifndef RECEIPT\_H

#define RECEIPT\_H

#include "retail\_item.h"

/\* define receipt struct here \*/

struct receipt {

struct retail\_item\* items;

int num\_items;

int max\_items;

int total;

};

/\* function prototypes - should match your .c implementations \*/

struct receipt \* create\_receipt(int max\_it);

int add\_item(struct receipt \* rec, struct retail\_item item);

int delete\_item(struct receipt \* rec, int item\_num);

void print\_receipt(struct receipt \* rec);

void free\_receipt(struct receipt \* rec);

#endif

**receipt.c**

#include <stdio.h>

#include <stdlib.h>

#include "retail\_item.h"

#include "receipt.h"

//method to create a receipt and returns a struct receipt. Needs number of items passed in.

struct receipt \* create\_receipt(int max\_it) {

struct receipt \*rec = malloc(sizeof(struct receipt));

rec->num\_items = 0;

rec->total = 0;

//If negative or nonsensical value is passed in for receipt size, default to 10 retail\_items

if(max\_it > 0) {

rec->max\_items = max\_it;

}

else {

rec->max\_items = 10;

}

//allocate empty array of items

rec->items = malloc(sizeof(struct retail\_item)\*rec->max\_items);

return rec;

}

//add retail\_item to receipt's item[] array. Add's item's cost to receipt $total

int add\_item(struct receipt\* rec, struct retail\_item item) {

if (rec->num\_items >= rec->max\_items) {

printf("Cannot add another item to receipt. Maximum size has been reached.\n");

return -1;

}

else{

rec->items[rec->num\_items] = item;

rec->num\_items++;

rec->total += (float)item.price\_cents;

return 0;

}

}

//remove retail\_item at position of receipt. Also subtracts $total from receipt

int delete\_item(struct receipt\* rec, int itemNum) {

int i;

//if you try to delete something that does not exist, you'll get an error statement

if (rec->num\_items <= itemNum-1) {

printf("Item number %d not found. Did not delete.\n", itemNum);

return -1;

}

else{

//Overwrite the deleted element and collapse the array

for(i = itemNum; i < rec->num\_items - 1; i++) {

rec->items[i] = rec->items[(i+1)];

}

rec->total -= (rec->items[itemNum]).price\_cents;

rec->num\_items--;

return 0;

}

}

//display receipt values

void print\_receipt(struct receipt\* rec) {

int i;

for(i = 0; i < rec->num\_items; i++) {

if (i == 0) {

printf("Receipt:\n");

print(rec->items[i]);

}

else if(i == rec->num\_items -1) {

print(rec->items[i]);

printf("Total $%.2f\n\n",(float)rec->total/100);

}

else {

print(rec->items[i]);

}

}

}

//delete and free data.

void free\_receipt(struct receipt\* rec) {

free(rec->items);

free(rec);

}