CMP-5015Y Coursework 1 - C Programming

100116544

Thu, 16 Nov 2017 11:55

Contents

StockItem.h

```
/*****************************************************************************
Description : Implements an abstract data type consisting of an instance of a
              item struct. Contains all the details to do with an item along
              with a set of interface functions.
 *****************************************************************************/

#ifndef STOCKITEM_H
#define STOCKITEM_H

typedef struct ItemStruct
{
    char comType[12];
    char code[12];
    int stock;
    int price;
    char additional[10];
}
StockItem;


/*****************************************************************************
Procedure   : item_new
Parameters  : char comType[] - Type of item
              char code[] - Code of item
              int stock - Stock for the item
              int price - Price of the item
              char additional - Any additional information for the item
Returns     : StockItem* - pointer to newly created item
Description : Allocate memory for a new Item and initialise it.  Generates an
              error message and the program terminates if insufficient memory
              is available.
 *****************************************************************************/
StockItem *item_new(char comType[],char code[], int stock, int price, char
   additional[]);

/*****************************************************************************
Procedure   : item_free
Parameters  : StockItem *item - pointer to an item
Returns     : none
Description : Frees the memory taken by an item.
 *****************************************************************************/
void item_free(StockItem *item);

/*****************************************************************************
Procedure   : item_print
Parameters  : StockItem *item - pointer to a StockItem
Returns     : none
Description : Prints out all of an items details.
 *****************************************************************************/
void item_print(StockItem *item);

/*****************************************************************************
Procedure   : item_getComType
Parameters  : StockItem *item - pointer to a StockItem.
Returns     : item->comType - the comtype of an item.
Description : Accessor function returning the comtype for a StockItem struct.
 *****************************************************************************/
static inline char *item_getComType(StockItem *item)
{
    return item->comType;
}
```

```
61
/*************************************************************************
63 Procedure   : item_getCode
Parameters  : StockItem *item - pointer to a StockItem.
65 Returns     : item->code - the code of an item.
Description : Accessor function returning the code for a StockItem struct.
67  *************************************************************************/
static inline char *item_getCode(StockItem *item)
69 {
    return item->code;
71 }

73 /*************************************************************************
Procedure   : item_getStock
75 Parameters  : StockItem *item - pointer to a StockItem.
Returns     : item->stock - the stock of an item.
77 Description : Accessor function returning the stock for a StockItem struct.
 *************************************************************************/
79 static inline int item_getStock(StockItem *item)
{
81     return item->stock;
}
83
/*************************************************************************
85 Procedure   : item_getPrice
Parameters  : StockItem *item - pointer to a StockItem.
87 Returns     : item->price - the price of an item.
Description : Accessor function returning the price for a StockItem struct.
89  *************************************************************************/
static inline int item_getPrice(StockItem *item)
91 {
    return item->price;
93 }

95 /*************************************************************************
Procedure   : item_getAdditional
97 Parameters  : StockItem *item - pointer to a StockItem.
Returns     : item->additional - the additional information for an item.
99 Description : Accessor function returning the additional information for a
    StockItem struct.
 *************************************************************************/
101 static inline char *item_getAdditional(StockItem *item)
{
103     return item->additional;
}
105
#endif /* STOCKITEM_H */
```

StockItem.c

```c
/*****************************************************************************
Description : Implements an abstract data type consisting of an instance of a
              item struct. Contains all the details to do with an item along
              with a set of interface functions.
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "StockItem.h"


// creates new stock item instance by allocating memory
StockItem *item_new(char comType[], char code[], int stock, int price, char
    additional[])
{
    StockItem *item = (StockItem*)malloc(sizeof(StockItem));

    if (item == NULL)
    {
        fprintf(stderr, "Error: Unable to allocate "
                "memory in item_new()\n");

        exit(EXIT_FAILURE);
    }

    strcpy(item->comType, comType);
    strcpy(item->code, code);
    item->stock = stock;
    item->price = price;
    strcpy(item->additional, additional);

    return item;
}

// frees up allocated memory for given item
void item_free(StockItem *item)
{
     free(item);
}

// prints a given item
void item_print(StockItem *item)
{
     printf("%s,%s,%d,%d,%s", item->comType, item->code,
               item->stock, item->price, item->additional);
}
```

Inventory.h

```
/*************************************************************
Description : Implements an abstract data type representing a
              collection of StockItem structs. Includes
                        interface functions as well as functions to be
                        used by the main.
*************************************************************/
#include "StockItem.h"
#ifndef  INVENTORY_H
#define INVENTORY_H

typedef struct NodeStruct
{
    StockItem *item;
    struct NodeStruct *next;
}
Node;

typedef struct InventoryListStruct
{
    Node *first;
    Node *last;
}
InventoryList;

/*************************************************************************
Procedure   : inventoryList_new
Parameters  : none
Returns     : InventoryList *inventoryList - pointer to newly created Inventory
   List
Description : Allocate memory for a new linked list of Stock Items and
              initialise it.
 *************************************************************************/
InventoryList *inventoryList_new();

/*************************************************************************
Procedure   : inventoryList_add
Parameters  : InventoryList *inventoryList - pointer to Inventory List which
                        the item will be added to
              StockItem *item - pointer to a StockItem to be added to the list
Returns     : none
Description : Inserts an item to the end of the list.
 *************************************************************************/
void inventoryList_add(InventoryList *inventoryList, StockItem *item);

/*************************************************************************
Procedure   : inventoryList_insert
Parameters  : InventoryList *inventoryList - pointer to Inventory List which
                        the item will be added to
              StockItem *item - pointer to the StockItem to be added to the list
Returns     : none
Description : Inserts an item to the start of the list
 *************************************************************************/
void inventoryList_insert(InventoryList *inventoryList, StockItem *item);

/*************************************************************************
Procedure   : inventoryList_length
Parameters  : InventoryList *inventoryList - pointer to a Inventory List
Returns     : int - number of items in inventory list.
Description : Determines the number of items stored in the inventory list.
 *************************************************************************/
int inventoryList_length(InventoryList *inventoryList);
```

```
61
   /*************************************************************************
63 Procedure    : inventoryList_clear
   Parameters   : InventoryList *inventoryList - pointer to a Inventory List
65 Returns      : none
   Description  : Clears the inventory list.
67  *************************************************************************/
   void inventoryList_clear(InventoryList *inventoryList);
69
   /*************************************************************************
71 Procedure    : inventoryList_print
   Parameters   : InventoryList *inventoryList - pointer to a Inventory List
73 Returns      : none
   Description  : Prints all items in the list.
75  *************************************************************************/
   void inventoryList_print(InventoryList *inventoryList);
77
   /*************************************************************************
79 Procedure    : removeWhitespace
   Parameters   : char *readin - a pointer to a string which will have its white
81                spaces removed
   Returns      : char - number of items in inventory list.
83 Description  : Determines the number of items stored in the inventory list.
85  *************************************************************************/
   char *removeWhitespace(char *readin);

87 /*************************************************************************
   Procedure    : inventoryList_addFromFile
89 Parameters   : InventoryList *inventoryList - pointer to a Inventory List
                 char file[] - stores the file to be opened and read from
91 Returns      : none
   Description  : Reads in items to be put into a inventory list
93  *************************************************************************/
   void inventoryList_addFromFile(InventoryList *inventoryList, char file[]);
95
   /*************************************************************************
97 Procedure    : inventoryList_sortPrice
   Parameters   : InventoryList *inventoryList - pointer to a Inventory List
99 Returns      : none
   Description  : Sorts the inventory list by price in ascending order
101 *************************************************************************/
   void inventoryList_sortPrice(InventoryList *inventoryList);
103
   /*************************************************************************
105 Procedure    : inventoryList_reduceQuantity
   Parameters   : InventoryList *inventoryList - pointer to a Inventory List
107               char searchCode[] - the code from the sale that has been read
                  int quantity - the quantity from the sale that has been read
109 Returns      : int - 0 for sale failed and 1 for sale was successful
   Description  : Reduces the stock quantity for each item based on a sale
111               that has been read
113 *************************************************************************/
   int inventoryList_reduceQuantity(InventoryList *inventoryList, char searchCode[],
       int quantity);

115 /*************************************************************************
   Procedure    : inventoryList_searchAdditional
117 Parameters   : InventoryList *inventoryList - pointer to a Inventory List
                 char searchAdditional - the additional information that is
119              being searched
   Returns      : int - the total stock for the items with the additional
121               information that was searched
   Description  : Gets the total stock of all items with the additional information
```

```
123                 search term provided
     ************************************************************************/
125 int inventoryList_searchAdditional(InventoryList *inventoryList, char
        searchAdditional[]);

127 /************************************************************************
    Procedure   : inventoryList_getResistance
129 Parameters  : InventoryList *inventoryList - pointer to a Inventory List
    Returns     : int - the total resistance
131 Description : Determines the total resistance of all the resistors
                  left in stock
133  ************************************************************************/
    int inventoryList_getResistance(InventoryList *inventoryList);
135
    #endif /* INVENTORY_H */
```

Inventory.c

```c
/*************************************************************
Description : Implements an abstract data type representing a
              collection of StockItem structs. Includes
                          interface functions as well as functions to be
                          used by the main.
*************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "Inventory.h"
#include "StockItem.h"
#include "Date.h"

// create new inventory list by allocating memory
InventoryList *inventoryList_new()
{
        // allocate memory
   InventoryList *inventoryList = (InventoryList*)malloc(sizeof(InventoryList));
   // check allocation was successful
   if (inventoryList == NULL)
   {
      fprintf(stderr, "Unable to allocate memory in inventoryList_new()\n");

      exit(EXIT_FAILURE);
   }
   // set first and last node to null
   inventoryList->first = NULL;
   inventoryList->last  = NULL;
   // return new list
   return inventoryList;
}

// add a stock item to the end of the list
void inventoryList_add(InventoryList *inventoryList, StockItem *item)
{
        // allocate memory to new node
        Node* node = (Node*)malloc(sizeof(Node));
        // check allocation was successful
        if (node == NULL)
        {
                fprintf(stderr, "Unable to allocate memory in inventoryList_add()
                   \n");
        exit(EXIT_FAILURE);
        }
        // store item with node
        node->item = item;
        // set next node to null
        node->next  = NULL;
        // check if last node of list is null
        if (inventoryList->last == NULL)
        {
                // if null, then store node in first/last node of list
                inventoryList->first = inventoryList->last = node;
        }
        else
        {
                // store node in last node of list
                inventoryList->last = inventoryList->last->next = node;
        }
}
```

```
62 // insert a stock item at start of list
   void inventoryList_insert(InventoryList *inventoryList, StockItem *item)
64 {
        // allocate memory to new node
66      Node* node = (Node*)malloc(sizeof(Node));
        // check allocation was successful
68      if (node == NULL)
        {
70              fprintf(stderr, "Unable to allocate memory in
                   inventoryList_insert()\n");
                exit(EXIT_FAILURE);
72      }
        // store item in new node
74      node->item = item;
        // set next node to first node of list
76      node->next  = inventoryList->first;

78      // if first node of list is NULL
        if (inventoryList->first == NULL)
80      {
                // store new node in first/last node of list
82              inventoryList->first = inventoryList->last = node;
        }
84      else
        {
86              // store new node at start of list
                inventoryList->first = node;
88      }
   }
90
   // get length of list
92 int inventoryList_length(InventoryList *inventoryList)
   {
94
        int length = 0;
96      // loop over all nodes and count
        for (Node* node = inventoryList->first; node != NULL; node = node->next)
98      {
                // increment length
100             length++;
        }
102     return length;
   }
104
   // clear list from memory
106 void inventoryList_clear(InventoryList *inventoryList)
   {
108     // loop until first node of list is empty
        while (inventoryList->first != NULL)
110     {
                // store first node
112             Node* node = inventoryList->first;
                // store first node of list as next node
114             inventoryList->first = node->next;
                // clear memory for node
116             free(node);
        }
118     // set last node to null
        inventoryList->last = NULL;
120 }

122 // print all stock items from list
```

```
void inventoryList_print(InventoryList *inventoryList)
{
        // loop through every node until null
        for(Node *node = inventoryList->first; node != NULL; node = node->next)
        {
                // print item
                item_print(node->item);
        }
}

/*************************************************************************
        Adapted whitespace remover from:
 * https://stackoverflow.com/questions/13084236/function-to-remove-spaces-from-
    string-char-array-in-c
 *************************************************************************/
char *removeWhitespace(char *readin)
{
    // make copies of string
    char *out = readin;
    char *input = readin;
    // loop through string till end
    for(; *readin != '\0'; ++readin){
            // if current char is not whitespace
            if(*readin != ' ')
                    // store char in input
                    *input++ = *readin;
    }
    *input = '\0';
    return out;
}

// add stock items to inventory list
void inventoryList_addFromFile(InventoryList *inventoryList, char file[])
{
        // open file
        FILE *readin = fopen(file, "r");
    // check file opened correctly
    if (readin == NULL) {
        printf("Unable to open %s.\n", file);
        exit(EXIT_FAILURE);
    }
        // create new stock item
    StockItem *item;
    // to store each line
    char line[255];
        // loop through file line by line
    while (fgets(line, sizeof(line), readin) != NULL)
    {
                // store each part of line by a comma and remove whitespace
                char* tempComType = removeWhitespace(strtok(line, ","));
                char* tempCode = removeWhitespace(strtok(NULL, ","));
                char* tempStock = removeWhitespace(strtok(NULL, ","));
                char* tempPrice = removeWhitespace(strtok(NULL, ","));
                char* tempAdditional = strtok(NULL, ",");
                // check that the line had a additional information
                if (tempAdditional == NULL) {
                        // if not provided then fill with null
                        tempAdditional = "NULL\n";
                } else {
                        // otherwise remove whitespace for additional information
                        removeWhitespace(tempAdditional);
                }
                // convert stock and price to ints
```

```
                int stockInt = atoi(tempStock);
                int priceInt = atoi(tempPrice);
                // create a new item using split up line of text
                item = item_new(tempComType, tempCode, stockInt, priceInt,
                    tempAdditional);
                // add the item to the end of the list
                inventoryList_add(inventoryList, item);
        }
}

// sort the linked list based on the price of items in ascending order
void inventoryList_sortPrice(InventoryList *inventoryList)
{
        // check list contains at least two items before sorting
        if (inventoryList->first != inventoryList->last){
                int isSorted;
                // loop until sorted
                while(!isSorted){
                        isSorted =1;
                        // loop through every node
                        for (Node *node=inventoryList->first; node->next!=NULL;
                            node=node->next){
                                // check to see if first node is greater than
                                    next and if so swap
                                if (node->item->price > node->next->item->price)
                                {
                                        StockItem *temp = node->item;
                                        node->item = node->next->item;
                                        node->next->item = temp;
                                        isSorted = 0;
                                }
                        }
                }
        }
}

// reduce the quantity of an item based on a transcation/sale being carried out
int inventoryList_reduceQuantity(InventoryList *inventoryList, char searchCode[],
    int quantity)
{

    // to store curret items code
    char *code;
    //  to store current items stock and new stock number
    int currentStock, newStock;
    // loop through every node/item of list
    for(Node *node = inventoryList->first; node != NULL; node = node->next){
            // store code of current item
        code = item_getCode(node->item);
            // check to see if current item code and sale code are the same
        if((strcmp(searchCode, code)) == 0){
            // get current stock of item
            currentStock = item_getStock(node->item);
             // calculate new stock
            newStock = currentStock - quantity;
            // if stock become less than 1
            if(newStock < 0){
                // transcation/sale has failed, do not change any values of item
                return 0;
            } else {
                // store newStock in current items stock
                node->item->stock = newStock;
                // transaction successful
```

```
244                     return 1;
                    }
246             }
        }
248 }

250 // search items based on there additional information and return a stock count
    int inventoryList_searchAdditional(InventoryList *inventoryList, char
        searchAdditional[])
252 {
        // will store current items additional information
254     char *additional;
        // will store total stock
256     int totalStock;
        // loop through all items
258     for(Node *node = inventoryList->first; node != NULL; node = node->next){
            // get additional information of current item
260         additional = item_getAdditional(node->item);
            // check to see if current additional and search additional are the same
262         if((strcmp(searchAdditional, additional)) == 0){
                // total up the stock if they are
264             totalStock = totalStock + item_getStock(node->item);
            }
266     }
        // return total stock
268     return totalStock;
    }
270
    // get total resistance of all items in stock
272 int inventoryList_getResistance(InventoryList *inventoryList)
    {
274     char *additional;
        char *comType;
276     int stock;
        for(Node *node = inventoryList->first; node != NULL; node=node->next){
278         comType = item_getComType(node->item);
            stock = item_getStock(node->item);
280         if((strcmp(comType, "resistor") == 0) && stock > 0){
                additional = item_getAdditional(node->item);
282             printf("%s", additional);

284             char *resistanceNum;
                char letter;
286             for (int i = 0; i < strlen(additional); i++){
                    if(isdigit(additional[i])){
288                     int len = strlen(resistanceNum);
                        resistanceNum[len] = additional[i];
290                     resistanceNum[len+1] = '\0';
                    } else {
292                     letter = additional[i];
                    }
294                 printf("%s, %c", resistanceNum, letter);
                }
296         }
        }
298 }
```

Sales.h

```
/*************************************************************
Description : Implements an abstract data type representing a
              collection of Sale structs. Includes
              interface functions as well as functions to be
              used by the main.
*************************************************************/
#include "SalesStruct.h"
#include "Inventory.h"
#ifndef SALES_H
#define SALES_H

typedef struct NodeStruct2
{
    Sale *sale;
    struct NodeStruct2 *next;
}
Node2;

typedef struct SalesListStruct
{
    Node2 *first;
    Node2 *last;
}
SalesList;

/*************************************************************************
Procedure   : salesList_new
Parameters  : none
Returns     : SalesList *salesList - pointer to newly created Sales List
Description : Allocate memory for a new linked list of Sales and
              initialise it.
 *************************************************************************/
SalesList *salesList_new();

/*************************************************************************
Procedure   : salesList_add
Parameters  : SalesList *salesList - pointer to Sales List where the sale
              will be added
              Sale *sale - pointer to a Sale to be added to the list
Returns     : none
Description : Adds a sale to the end of the list.
 *************************************************************************/
void salesList_add(SalesList *salesList, Sale *sale);

/*************************************************************************
Procedure   : salesList_insert
Parameters  : SalesList *salesList - pointer to Sales List where the sale
              will be inserted
              Sale *sale - pointer to the Sale to be added to the list
Returns     : none
Description : Inserts a sale to the start of the list.
 *************************************************************************/
void salesList_insert(SalesList *salesList, Sale *sale);

/*************************************************************************
Procedure   : salesList_length
Parameters  : SalesList *salesList - pointer to a Sales List
Returns     : int - number of items in sales list
Description : Determines the number of sales stored in the sales list.
 *************************************************************************/
int salesList_length(SalesList *salesList);
```

```
62
   /*************************************************************************
64 Procedure    : salesList_clear
   Parameters   : SalesList *salesList - pointer to a Sales List
66 Returns      : none
   Description  : Clears the sales list.
68  *************************************************************************/
   void salesList_clear(SalesList *salesList);
70
   /*************************************************************************
72 Procedure    : salesList_print
   Parameters   : SalesList *salesList - pointer to a Sales List
74 Returns      : none
   Description  : Prints all sales in the list.
76  *************************************************************************/
   void salesList_print(SalesList *salesList);
78
   /*************************************************************************
80 Procedure    : salesList_addFromFile
   Parameters   : SalesList *inventoryList - pointer to a Sales List
82                char file[] - stores the file to be opened and read from
   Returns      : none
84 Description  : Reads in sales to be put into a sales list
    *             It also attempts the transaction and outputs a message if it fails
86  *************************************************************************/
   void salesList_addFromFile(SalesList *salesList, InventoryList *inventoryList,
      char file[], SalesList *failList);
88
   void salesList_printToFile(SalesList *salesList, char file[]);
90
   #endif /* SALES_H */
```

Sales.c

```c
/*************************************************************
Description : Implements an abstract data type representing a
              collection of Sale structs. Includes
              interface functions as well as functions to be
              used by the main.
*************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Sales.h"
#include "Date.h"

// creates new sales list by allocating memory
SalesList *salesList_new()
{
    // allocate memory
    SalesList *salesList = (SalesList*)malloc(sizeof(SalesList));
    // check allocation was successful
    if (salesList == NULL)
    {
      fprintf(stderr, "Unable to allocate memory in salesList_new()\n");
      exit(EXIT_FAILURE);
    }
    // set first and last node to null
    salesList->first = NULL;
    salesList->last = NULL;
    // return new list
    return salesList;
}

// add a stock item to the end of the list
void salesList_add(SalesList *salesList, Sale *sale)
{
    // allocate memory to the new node
    Node2* node =(Node2*)malloc(sizeof(Node2));
    // check allocation was successful
    if (node == NULL)
    {
        fprintf(stderr, "Unable to allocate memory in salesList_add()\n");
        exit(EXIT_FAILURE);
    }
    // store sale with node
    node->sale = sale;
    // set next node to null
    node->next = NULL;
    // check if last node of list is null
    if (salesList->last == NULL)
    {
        // if null, then store node in first/last node of list
        salesList->first = salesList->last = node;
    } else {
        // store node in last node of list
        salesList->last = salesList->last->next = node;
    }
}

// insert a sale at start of list
void salesList_insert(SalesList *salesList, Sale *sale)
{
    // allocate memory to new node
    Node2* node =(Node2*)malloc(sizeof(Node2));
```

```
62      // check allocation was successful
        if (node == NULL)
64      {
            fprintf(stderr, "Error: Unable to allocate memory in salesList_add()\n");
66          exit(EXIT_FAILURE);
        }
68      // store sale in new node
        node->sale = sale;
70      // set next node to first node of list
        node->next = salesList->first;
72      // if first node of list is null
        if(salesList->first == NULL)
74      {
            // store new node in first/last node of list
76          salesList->first = salesList->last = node;
        } else {
78          // store new node at start of list
            salesList->first = node;
80      }
    }
82
    // get length of list
84  int SalesList_length(SalesList *salesList)
    {
86      int length = 0;
        // loop over all nodes and count
88      for (Node2* node = salesList->first; node != NULL; node = node->next)
        {
90          // increment length
            length++;
92      }
        return length;
94  }

96  // clear list from memory
    void salesList_clear(SalesList *salesList)
98  {
        // loop until first node of list is empty
100     while (salesList->first != NULL)
        {
102         // store first node
            Node2* node = salesList->first;
104         // store first node of list as next node
            salesList->first = node->next;
106         // clear memory for node
            free(node);
108     }
        // set last node to null
110     salesList->last = NULL;
    }
112
    // print all sale items from list
114 void salesList_print(SalesList *salesList)
    {
116     // loop through every node until null
        for(Node2 *node = salesList->first; node != NULL; node = node->next){
118         // print sale
            sale_print(node->sale);
120     }
    }
122
    // add sale to sales list
124 void salesList_addFromFile(SalesList *salesList, InventoryList *inventoryList,
```

```
    char file[], SalesList *failList)
{
    // open file
    FILE *readin = fopen(file, "r");
    // check file opened
    if( readin == NULL) {
        printf("Unable to open %s.\n", file);
        exit(EXIT_FAILURE);
    }
    // create new sale
    Sale *complete;
    Sale *fail;
    // store each line
    char line[255];
    // loop through file line by line
    while(fgets(line, sizeof(line), readin) != NULL){
        // store each part of line seperated by comma and remove whitespace
        char* tempDate = removeWhitespace(strtok(line, ","));
        char* tempCode = removeWhitespace(strtok(NULL, ","));
        char* tempQuantity = removeWhitespace(strtok(NULL, ","));
        // covert quantity into an int
        int quantity = atoi(tempQuantity);
        // seperate date string by / and convert to int
        int tempDay = atoi(strtok(tempDate, "/"));
        int tempMonth = atoi(strtok(NULL, "/"));
        int tempYear = atoi(strtok(NULL, "/"));
        // create new date
        Date *newDate;
        // make instance of date with temp ints
        newDate = date_new(tempDay, tempMonth, tempYear);
        int saleComplete;
        // reduce quantity of item based on sale
        saleComplete = inventoryList_reduceQuantity(inventoryList, tempCode,
            quantity);
        // if sale failed then output message
        if (saleComplete == 0){
            fail = sale_new(newDate, tempCode, quantity);
            salesList_add(failList, fail);
        } else {
            // if sale was successful then make new sale instance
            complete = sale_new(newDate, tempCode, quantity);
            // add sale to list
            salesList_add(salesList, complete);
        }
    }
}

void salesList_printToFile(SalesList *salesList, char file[])
{
    FILE *f = fopen(file, "w");
    if (f == NULL)
    {
        printf("Error opening file!\n");
        exit(1);
    }

    for(Node2 *node = salesList->first; node != NULL; node = node->next){
        int day = node->sale->date->day;
        int month = node->sale->date->month;
        int year = node->sale->date->year;
        char *code = node->sale->code;
        int quantity = node->sale->quantity;
```

```
186             fprintf(f,"%d/%d/%d,%s,%d\n", day, month, year, code, quantity);
        }
188
}
```

StockProgram.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "StockItem.h"
#include "Inventory.h"
#include "Sales.h"
#include "SalesStruct.h"

int main(int argc, char** argv) {
    // make new inventory list
    InventoryList *inventoryList = inventoryList_new();
    // add inventory from file to list
    inventoryList_addFromFile(inventoryList,"inventory.txt");

    // make new sales list
    SalesList *salesList = salesList_new();
    SalesList *failSalesList = salesList_new();
    /* add sales from file to list
     and run all transactions */
    salesList_addFromFile(salesList, inventoryList, "sales.txt", failSalesList);

    printf("Completed sales found in completedSales.txt.\n");
    printf("Failed sales found in failSales.txt.\n\n");

    //print all failed transactions to a seperate file
    salesList_printToFile(failSalesList, "failSales.txt");
    // print all successful transactions to a seperate file
    salesList_printToFile(salesList, "completedSales.txt");

    // task 1 - sort in order of increasing price
    printf("List of inventory, sorted in order of increasing price:\n");
    inventoryList_sortPrice(inventoryList);
    inventoryList_print(inventoryList);

    // task 3 - find how many NPN transistors are left in stock
    char *checkAdditional = "NPN";
    int stockCheck = inventoryList_searchAdditional(inventoryList,
       checkAdditional);
    printf("\nThere are %d NPN transistors left in stock.\n", stockCheck);

    // task 4 - get total resistance of all remaining resistors
    //int resistance = inventoryList_getResistance(inventoryList);
    //printf("\nThe total resistance of resistors in stock is: %d", resistance);
    return (EXIT_SUCCESS);
}
```

Date.c

```c
/*****************************************************************************
Description : Implements an abstract data type consisting of an instance of a
              date struct. Contains all of the information relating to a
              particular date and a set of interface functions.
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Date.h"

// creates new date instance by allocating memory
Date *date_new(int day, int month, int year)
{
    Date *date =(Date*)malloc(sizeof(Date));

    if (date == NULL)
    {
          fprintf(stderr, "Error: Unable to allocate "
              "memory in date_new()\n");
      exit(EXIT_FAILURE);
    }

   date->day = day;
   date->month = month;
   date->year = year;

   return date;
}

// frees up allocated memory for given date
void date_free(Date *date)
{
    free(date);
}

// prints a given date in dd/mm/yyyy format
void date_print(Date *date)
{
    printf("%d/%d/%d", date->day, date->month, date->year);
}
```

Date.h

```
#ifndef DATE_H
#define DATE_H

typedef struct DateStruct
{
    int day;
    int month;
    int year;
}
Date;

/**************************************************************************
Procedure   : date_new
Parameters  : int day - day of date struct
              int month - month of date struct
              int year - year of date struct.
Returns     : Date* - pointer to newly created Date
Description : Allocate memory for a new Date and initialise it.  Generates an
              error message and the program terminates if insufficient memory
              is available.
 **************************************************************************/
Date *date_new(int day, int month, int year);

/**************************************************************************
Procedure   : date_free
Parameters  : Date *date - pointer to a date
Returns     : none
Description : Frees the memory taken by a date.
 **************************************************************************/
void date_free(Date *date);

/**************************************************************************
Procedure   : date_print
Parameters  : Date *date - pointer to a date
Returns     : none
Description : Prints out all of a date in dd/mm/yyyy format
 **************************************************************************/
void date_print(Date *date);

/**************************************************************************
Procedure   : date_getDay
Parameters  : Date *day - pointer to a Date.
Returns     : date->day - the day of the date.
Description : Accessor function returning the day for a Date struct.
 **************************************************************************/
static inline int date_getDay(Date *date)
{
    return date->day;
}

/**************************************************************************
Procedure   : date_getMonth
Parameters  : Date *date - pointer to a Date.
Returns     : date->month - the month of the date.
Description : Accessor function returning the month for a Date struct.
 **************************************************************************/
static inline int date_getMonth(Date *date)
{
    return date->month;
}
```

```
62 /*****************************************************************************
   Procedure   : date_getYear
64 Parameters  : Date *day - pointer to a Date.
   Returns     : date->year - the year of the date.
66 Description : Accessor function returning the year for a Date struct.
    *****************************************************************************/
68 static inline int date_getYear(Date *date)
   {
70     return date->year;
   }
72
   #endif /* DATE_H */
```

SalesStruct.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "SalesStruct.h"
#include "Date.h"

// creates a new sale instance by allocating memory
Sale *sale_new(Date *date, char code[], int quantity)
{
    Sale *sale= (Sale*)malloc(sizeof(Sale));
    if(sale == NULL)
    {
        fprintf(stderr, "Error: Unable to allocate "
              "memory in sale_new()\n");
        exit(EXIT_FAILURE);
    }

    sale->date = date;
    strcpy(sale->code, code);
    sale->quantity = quantity;

    return sale;
}

// frees up allocated memory for given sale
void sale_free(Sale *sale)
{
    free(sale);
}

// prints a given sale in a list format
void sale_print(Sale *sale){
    date_print(sale_getDate(sale));
    printf(",%s,%d\n", sale_getCode(sale), sale_getQuantity(sale));
}
```

SalesStruct.h

```
/*****************************************************************************
Description : Implements an abstract data type consisting of an instance of a
              Sales struct. Contains all of the information relating to a
              particular sale and a set of interface functions.
 *****************************************************************************/
#ifndef SALESSTRUCT_H
#define SALESSTRUCT_H
#include "Date.h"

typedef struct SalesStruct
{
    Date* date;
    char code[12];
    int quantity;
}
Sale;

/*****************************************************************************
Procedure   : sale_new
Parameters  : Date date - the date of the sale as a Date struct
              char code[] - item code
              int quantity - the quantity of the item
Returns     : Sale* - pointer to newly created Sale
Description : Allocate memory for a new Sale and initialise it.  Generates an
              error message and the program terminates if insufficient memory
              is available.
 *****************************************************************************/
Sale *sale_new(Date *date, char code[], int quantity);

/*****************************************************************************
Procedure   : sale_free
Parameters  : Sale *sale - pointer to a Sale
Returns     : none
Description : Frees the memory taken by a sale.
 *****************************************************************************/
void sale_free(Sale *sale);

/*****************************************************************************
Procedure   : sale_print
Parameters  : Sale *sale - pointer to a Sale
Returns     : none
Description : Prints out all details to do with a sale
 *****************************************************************************/
void sale_print(Sale *sale);

/*****************************************************************************
Procedure   : sale_getDate
Parameters  : Sale *sale - pointer to a Sale
Returns     : sale->date - the date of the sale as a Date struct
Description : Accessor function returning the date of a sale.
 *****************************************************************************/
static inline Date *sale_getDate(Sale *sale)
{
    return sale->date;
}

/*****************************************************************************
Procedure   : sale_getCode
Parameters  : Sale *sale - pointer to a Sale
Returns     : sale->code - the code for the sale
Description : Accessor function returning the code of the sale.
```

```
 ******************************************************************************/
static inline char *sale_getCode(Sale *sale){
    return sale->code;
}

/******************************************************************************
Procedure   : sale_getQuantity
Parameters  : Sale *sale - pointer to a Sale
Returns     : sale->quantity - the quantity of a sale
Description : Accessor function returning the quantity of a sale.
 ******************************************************************************/
static inline int sale_getQuantity(Sale *sale){
    return sale->quantity;
}

#endif /* SALESSTRUCT_H */
```

Application

Compiler Invocation

```
gcc.exe -Wall -o electronics.exe -std=c11 -lm StockItem.c Inventory.c Sales.c
    StockProgram.c Date.c SalesStruct.c
```

Compiler Messages

```
Inventory.c: In function 'inventoryList_reduceQuantity':
Inventory.c:248:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
Inventory.c: In function 'inventoryList_getResistance':
Inventory.c:298:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
Inventory.c:288:31: warning: 'resistanceNum' may be used uninitialized in this function [-Wmaybe-uninitial
                     int len = strlen(resistanceNum);
                               ^
```

Application Invocation

```
electronics.exe
```

Messages to STDOUT

```
Completed sales found in completedSales.txt.
Failed sales found in failSales.txt.

List of inventory, sorted in order of increasing price:
resistor,RES_1R0,0,1,1R0
resistor,RES_10R,4,1,10R
resistor,RES_100R,0,1,100R
resistor,RES_1K0,0,1,1K0
resistor,RES_10K,0,1,10K
resistor,RES_100K,177,1,100K
resistor,RES_1M0,2,1,1M0
resistor,RES_10M,0,1,10M
resistor,RES_1R1,471,1,1R1
resistor,RES_11R,0,1,11R
resistor,RES_110R,191,1,110R
resistor,RES_1K1,0,1,1K1
resistor,RES_11K,0,1,11K
resistor,RES_110K,371,1,110K
resistor,RES_1M1,493,1,1M1
resistor,RES_11M,62,1,11M
resistor,RES_1R2,507,1,1R2
resistor,RES_12R,512,1,12R
resistor,RES_120R,392,1,120R
resistor,RES_1K2,0,1,1K2
resistor,RES_12K,1,1,12K
resistor,RES_120K,98,1,120K
resistor,RES_1M2,480,1,1M2
resistor,RES_12M,0,1,12M
resistor,RES_1R3,0,1,1R3
resistor,RES_13R,0,1,13R
resistor,RES_130R,0,1,130R
resistor,RES_1K3,306,1,1K3
resistor,RES_13K,217,1,13K
resistor,RES_130K,419,1,130K
resistor,RES_1M3,76,1,1M3
resistor,RES_13M,126,1,13M
```

```
resistor,RES_1R5,343,1,1R5
resistor,RES_15R,1,1,15R
resistor,RES_150R,438,1,150R
resistor,RES_1K5,440,1,1K5
resistor,RES_15K,234,1,15K
resistor,RES_150K,0,1,150K
resistor,RES_1M5,0,1,1M5
resistor,RES_15M,380,1,15M
resistor,RES_1R6,203,1,1R6
resistor,RES_16R,348,1,16R
resistor,RES_160R,0,1,160R
resistor,RES_1K6,0,1,1K6
resistor,RES_16K,127,1,16K
resistor,RES_160K,215,1,160K
resistor,RES_1M6,0,1,1M6
resistor,RES_16M,265,1,16M
resistor,RES_1R8,0,1,1R8
resistor,RES_18R,404,1,18R
resistor,RES_180R,43,1,180R
resistor,RES_1K8,206,1,1K8
resistor,RES_18K,90,1,18K
resistor,RES_180K,227,1,180K
resistor,RES_1M8,0,1,1M8
resistor,RES_18M,391,1,18M
resistor,RES_2R0,325,1,2R0
resistor,RES_20R,286,1,20R
resistor,RES_200R,36,1,200R
resistor,RES_2K0,194,1,2K0
resistor,RES_20K,0,1,20K
resistor,RES_200K,0,1,200K
resistor,RES_2M0,0,1,2M0
resistor,RES_20M,334,1,20M
resistor,RES_2R2,0,1,2R2
resistor,RES_22R,0,1,22R
resistor,RES_220R,0,1,220R
resistor,RES_2K2,458,1,2K2
resistor,RES_22K,0,1,22K
resistor,RES_220K,229,1,220K
resistor,RES_2M2,50,1,2M2
resistor,RES_22M,322,1,22M
resistor,RES_2R4,295,1,2R4
resistor,RES_24R,220,1,24R
resistor,RES_240R,0,1,240R
resistor,RES_2K4,0,1,2K4
resistor,RES_24K,0,1,24K
resistor,RES_240K,0,1,240K
resistor,RES_2M4,0,1,2M4
resistor,RES_24M,1,1,24M
resistor,RES_2R7,89,1,2R7
resistor,RES_27R,189,1,27R
resistor,RES_270R,0,1,270R
resistor,RES_2K7,498,1,2K7
resistor,RES_27K,254,1,27K
resistor,RES_270K,313,1,270K
resistor,RES_2M7,519,1,2M7
resistor,RES_27M,0,1,27M
resistor,RES_3R0,401,1,3R0
resistor,RES_30R,0,1,30R
resistor,RES_300R,465,1,300R
resistor,RES_3K0,1,1,3K0
resistor,RES_30K,101,1,30K
resistor,RES_300K,0,1,300K
resistor,RES_3M0,46,1,3M0
```

```
resistor,RES_30M,215,1,30M
resistor,RES_3R3,0,1,3R3
resistor,RES_33R,0,1,33R
resistor,RES_330R,462,1,330R
resistor,RES_3K3,16,1,3K3
resistor,RES_33K,314,1,33K
resistor,RES_330K,0,1,330K
resistor,RES_3M3,160,1,3M3
resistor,RES_33M,253,1,33M
resistor,RES_3R6,227,1,3R6
resistor,RES_36R,413,1,36R
resistor,RES_360R,430,1,360R
resistor,RES_3K6,0,1,3K6
resistor,RES_36K,131,1,36K
resistor,RES_360K,0,1,360K
resistor,RES_3M6,0,1,3M6
resistor,RES_36M,300,1,36M
resistor,RES_3R9,437,1,3R9
resistor,RES_39R,0,1,39R
resistor,RES_390R,0,1,390R
resistor,RES_3K9,391,1,3K9
resistor,RES_39K,299,1,39K
resistor,RES_390K,170,1,390K
resistor,RES_3M9,0,1,3M9
resistor,RES_39M,0,1,39M
resistor,RES_4R3,438,1,4R3
resistor,RES_43R,0,1,43R
resistor,RES_430R,0,1,430R
resistor,RES_4K3,196,1,4K3
resistor,RES_43K,0,1,43K
resistor,RES_430K,327,1,430K
resistor,RES_4M3,209,1,4M3
resistor,RES_43M,38,1,43M
resistor,RES_4R7,0,1,4R7
resistor,RES_47R,0,1,47R
resistor,RES_470R,0,1,470R
resistor,RES_4K7,0,1,4K7
resistor,RES_47K,368,1,47K
resistor,RES_470K,181,1,470K
resistor,RES_4M7,437,1,4M7
resistor,RES_47M,1,1,47M
resistor,RES_5R1,0,1,5R1
resistor,RES_51R,0,1,51R
resistor,RES_510R,0,1,510R
resistor,RES_5K1,0,1,5K1
resistor,RES_51K,0,1,51K
resistor,RES_510K,0,1,510K
resistor,RES_5M1,274,1,5M1
resistor,RES_51M,0,1,51M
resistor,RES_5R6,495,1,5R6
resistor,RES_56R,461,1,56R
resistor,RES_560R,0,1,560R
resistor,RES_5K6,296,1,5K6
resistor,RES_56K,0,1,56K
resistor,RES_560K,142,1,560K
resistor,RES_5M6,1,1,5M6
resistor,RES_56M,399,1,56M
resistor,RES_6R2,39,1,6R2
resistor,RES_62R,0,1,62R
resistor,RES_620R,0,1,620R
resistor,RES_6K2,0,1,6K2
resistor,RES_62K,410,1,62K
resistor,RES_620K,178,1,620K
```

```
resistor,RES_6M2,263,1,6M2
resistor,RES_62M,214,1,62M
resistor,RES_6R8,1,1,6R8
resistor,RES_68R,0,1,68R
resistor,RES_680R,124,1,680R
resistor,RES_6K8,0,1,6K8
resistor,RES_68K,37,1,68K
resistor,RES_680K,59,1,680K
resistor,RES_6M8,97,1,6M8
resistor,RES_68M,0,1,68M
resistor,RES_7R5,143,1,7R5
resistor,RES_75R,0,1,75R
resistor,RES_750R,0,1,750R
resistor,RES_7K5,410,1,7K5
resistor,RES_75K,0,1,75K
resistor,RES_750K,0,1,750K
resistor,RES_7M5,81,1,7M5
resistor,RES_75M,0,1,75M
resistor,RES_8R2,0,1,8R2
resistor,RES_82R,0,1,82R
resistor,RES_820R,97,1,820R
resistor,RES_8K2,75,1,8K2
resistor,RES_82K,0,1,82K
resistor,RES_820K,183,1,820K
resistor,RES_8M2,0,1,8M2
resistor,RES_82M,523,1,82M
resistor,RES_9R1,0,1,9R1
resistor,RES_91R,493,1,91R
resistor,RES_910R,470,1,910R
resistor,RES_9K1,283,1,9K1
resistor,RES_91K,0,1,91K
resistor,RES_910K,1,1,910K
resistor,RES_9M1,293,1,9M1
resistor,RES_91M,459,1,91M
capacitor,CAP_10pF,106,12,10pF
capacitor,CAP_100pF,31,24,100pF
capacitor,CAP_1000pf,312,36,1000pf
capacitor,CAP_10nF,53,48,10nF
capacitor,CAP_100nF,0,60,100nF
capacitor,CAP_1000nF,81,72,1000nF
capacitor,CAP_10uF,367,84,10uF
capacitor,CAP_100uF,74,96,100uF
capacitor,CAP_11pF,0,11,11pF
capacitor,CAP_110pF,91,23,110pF
capacitor,CAP_1100pf,370,35,1100pf
capacitor,CAP_11nF,31,47,11nF
capacitor,CAP_110nF,194,59,110nF
capacitor,CAP_1100nF,0,71,1100nF
capacitor,CAP_11uF,65,83,11uF
capacitor,CAP_110uF,66,95,110uF
capacitor,CAP_12pF,340,10,12pF
capacitor,CAP_120pF,0,22,120pF
capacitor,CAP_1200pf,0,34,1200pf
capacitor,CAP_12nF,0,46,12nF
capacitor,CAP_120nF,356,58,120nF
capacitor,CAP_1200nF,228,70,1200nF
capacitor,CAP_12uF,465,82,12uF
capacitor,CAP_120uF,218,94,120uF
capacitor,CAP_13pF,0,9,13pF
capacitor,CAP_130pF,4,21,130pF
capacitor,CAP_1300pf,0,33,1300pf
capacitor,CAP_13nF,0,45,13nF
capacitor,CAP_130nF,0,57,130nF
```

```
capacitor,CAP_1300nF,91,69,1300nF
capacitor,CAP_13uF,0,81,13uF
capacitor,CAP_130uF,2,93,130uF
capacitor,CAP_15pF,3,8,15pF
capacitor,CAP_150pF,0,20,150pF
capacitor,CAP_1500pf,0,32,1500pf
capacitor,CAP_15nF,177,44,15nF
capacitor,CAP_150nF,142,56,150nF
capacitor,CAP_1500nF,287,68,1500nF
capacitor,CAP_15uF,276,80,15uF
capacitor,CAP_150uF,0,92,150uF
capacitor,CAP_16pF,103,7,16pF
capacitor,CAP_160pF,268,19,160pF
capacitor,CAP_1600pf,0,31,1600pf
capacitor,CAP_16nF,0,43,16nF
capacitor,CAP_160nF,0,55,160nF
capacitor,CAP_1600nF,15,67,1600nF
capacitor,CAP_16uF,44,79,16uF
capacitor,CAP_160uF,0,91,160uF
capacitor,CAP_18pF,0,6,18pF
capacitor,CAP_180pF,0,18,180pF
capacitor,CAP_1800pf,0,30,1800pf
capacitor,CAP_18nF,209,42,18nF
capacitor,CAP_180nF,539,54,180nF
capacitor,CAP_1800nF,0,66,1800nF
capacitor,CAP_18uF,258,78,18uF
capacitor,CAP_180uF,343,90,180uF
capacitor,CAP_20pF,0,5,20pF
capacitor,CAP_200pF,42,17,200pF
capacitor,CAP_2000pf,0,29,2000pf
capacitor,CAP_20nF,211,41,20nF
capacitor,CAP_200nF,0,53,200nF
capacitor,CAP_2000nF,0,65,2000nF
capacitor,CAP_20uF,0,77,20uF
capacitor,CAP_200uF,200,89,200uF
capacitor,CAP_22pF,0,4,22pF
capacitor,CAP_220pF,422,16,220pF
capacitor,CAP_2200pf,469,28,2200pf
capacitor,CAP_22nF,0,40,22nF
capacitor,CAP_220nF,452,52,220nF
capacitor,CAP_2200nF,0,64,2200nF
capacitor,CAP_22uF,406,76,22uF
capacitor,CAP_220uF,0,88,220uF
capacitor,CAP_24pF,323,3,24pF
capacitor,CAP_240pF,584,15,240pF
capacitor,CAP_2400pf,0,27,2400pf
capacitor,CAP_24nF,177,39,24nF
capacitor,CAP_240nF,0,51,240nF
capacitor,CAP_2400nF,235,63,2400nF
capacitor,CAP_24uF,6,75,24uF
capacitor,CAP_240uF,0,87,240uF
capacitor,CAP_27pF,325,2,27pF
capacitor,CAP_270pF,0,14,270pF
capacitor,CAP_2700pf,0,26,2700pf
capacitor,CAP_27nF,262,38,27nF
capacitor,CAP_270nF,170,50,270nF
capacitor,CAP_2700nF,149,62,2700nF
capacitor,CAP_27uF,0,74,27uF
capacitor,CAP_270uF,352,86,270uF
capacitor,CAP_30pF,417,1,30pF
capacitor,CAP_300pF,189,13,300pF
capacitor,CAP_3000pf,0,25,3000pf
capacitor,CAP_30nF,223,37,30nF
```

```
capacitor,CAP_300nF,0,49,300nF
capacitor,CAP_3000nF,0,61,3000nF
capacitor,CAP_30uF,0,73,30uF
capacitor,CAP_300uF,104,85,300uF
diode,BY126,0,12,NULL
diode,BY127,0,12,NULL
diode,1N4001,0,4,NULL
diode,1N4004,0,6,NULL
diode,1N4148,0,5,NULL
transistor,AC125,0,35,PNP
transistor,AC126,0,37,PNP
transistor,BC107,0,10,NPN
transistor,BC108,0,12,NPN
transistor,TIS88A,0,50,FET
transistor,OC44,0,50,PNP
transistor,2N2369A,0,18,NPN
IC,NE555,0,17,"Timer"
IC,LF356,0,45,"JFETop-amp"
IC,Z80A,0,800,"CPU"
IC,TL741,0,12,"op-amp"

There are 0 NPN transistors left in stock.
```

Messages to STDERR

None.