```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

      struct Data {
            char title[50];
            char author[50];
            int stockNumber;
            float wholesalePrice;
            float retailPrice;
            int wholesaleQuantity;
            int retailQuantity;
      };

      typedef struct Node {
            struct Data book;
            struct Node *next;
            /* pointers to node */
      } Node;

void getDataAndBuildList(Node **headPtr);
Node *createNodeAndGetData(void);
void insert(Node **headPtr, Node *newNodePtr);
void delete(Node **headPtr, int prodNumberToDelete);
void getUserOption(Node **head);
double calculateTotalRevenue(const Node *head);
double calculateInvestmentInInventory(const Node *head);
double calculateTotalWholesaleCost(const Node *head);
double calculateTotalProfit(const Node *head);
int calculateTotalBooksSold(const Node *head);
double calculateAverageProfit(const Node *head);
void printList(const Node *head);
void freeAllNodes(Node **headPtr);

int main() {
      Node *head = NULL;
      getDataAndBuildList(&head);
      getUserOption(&head);
      return 0;
}

void getDataAndBuildList(Node **headPtr) {
      Node *newNodePtr;
      printf("Please enter data about the books.\n\n");
      while (newNodePtr = createNodeAndGetData()) {
            insert(headPtr, newNodePtr);
      }
}

Node *createNodeAndGetData(void) {
      Node *newNodePtr;
      newNodePtr = malloc (sizeof(Node));
      if (newNodePtr == NULL) {
            printf("Error: memory could not be allocated for enough nodes. ");
            printf("Terminating program!\n");
            exit (0);
      }
      else {
            scanf("%[^\n]", newNodePtr->book.title);
            if (strcmp(newNodePtr->book.title, "END_DATA") == 0) {
                  /* free Node if end of book data detected */
                  free(newNodePtr);
                  return NULL;
            }
            else {
                  /* consume newline before author string */
                  getchar();
                  scanf("%[^\n]s", newNodePtr->book.author);
                  scanf("%i", &newNodePtr->book.stockNumber);
                  scanf("%f", &newNodePtr->book.wholesalePrice);
                  scanf("%f", &newNodePtr->book.retailPrice);
                  scanf("%i", &newNodePtr->book.wholesaleQuantity);
                  scanf("%i", &newNodePtr->book.retailQuantity);
                  /* consume newline before next title string */
                  getchar();
            }
            return newNodePtr;
      }
}


void insert(Node ** headPtr, Node *newNodePtr) {
    Node *traversePtr = *headPtr;
      if(traversePtr == NULL){
            /* The list is empty */
            *headPtr = newNodePtr;
            newNodePtr->next = NULL;

      }
      else if(newNodePtr->book.stockNumber < traversePtr->book.stockNumber ){
        /* insert the list as the first node in the list */
            *headPtr = newNodePtr;
        newNodePtr->next = traversePtr;
      }
      else{
            /* Insert the node into the sorted list */
            Node *priorNodePtr = traversePtr;
            /* a pointer to the node before the node where we are */
          traversePtr = traversePtr->next;
            while(traversePtr != NULL && newNodePtr->book.stockNumber >
traversePtr->book.stockNumber){
                  /* while we didn't reach the end and the stock number of the new
node is less than
                   the stocknumber of the current node, here is the place to insert
the new node.
                   before the current(traverse) node */
                   priorNodePtr = traversePtr;
                   traversePtr = traversePtr->next;
            }
            priorNodePtr->next = newNodePtr;
          newNodePtr->next = traversePtr;
      }
      printf("Book stock number %i was added to the inventory \n", newNodePtr-
>book.stockNumber);
      /* WRITE THE CODE FOR THIS FUNCTION */
}

void getUserOption(Node **headPtr) {
      int option;
      Node *newNodePtr;
      int bookNumToDelete;
      do {
            printf("\nPlease enter an integer between 1 and 9 to select an 
operation on the data:\n");
            scanf("%i", &option);
            getchar();
            switch (option){
                  case 1:
                        printList (*headPtr);
                        break;
                  case 2:
                        printf("\nTotal revenue: %.2f\n", 
calculateTotalRevenue(*headPtr));
                        break;
                  case 3:
                        printf("\nTotal wholesale cost: %.2f\n", 
calculateTotalWholesaleCost(*headPtr));
                        break;
                  case 4:
                        printf("\nTotal investment in inventory: %.2f\n", 
calculateInvestmentInInventory(*headPtr));
                        break;
                  case 5:
                        printf("\nTotal profit: %.2f\n", 
calculateTotalProfit(*headPtr));
                        break;
                  case 6:
                        printf("\nTotal number of books sold = %i\n", 
calculateTotalBooksSold(*headPtr));
                        break;
                  case 7:
                        printf("\nAverage profit: %.2f\n", 
calculateAverageProfit(*headPtr));
                        break;
                  case 8:
                        printf("\nPlease enter the data for the book you wish to 
add:\n\n");
                        newNodePtr = createNodeAndGetData();
                        insert(headPtr, newNodePtr);

                        break;
                  case 9:
                        printf("\nPlease enter the book stock number of the book 
you wish to delete, ");
                        printf("followed by enter.\n");
                        scanf("%i", &bookNumToDelete);
                        delete (headPtr, bookNumToDelete);

                        break;
                  case 10:
                        freeAllNodes(headPtr);
                        break;
                  default:
                        printf("Valid option choices are 1 to 10. Please choose 
again!\n");
                        break;
            }
      } while (option != 10);
}

double calculateTotalRevenue(const Node *head) {
      double result = 0;

      while(head != NULL){
            /* If does not reach the end */
            result = result + head->book.retailPrice * head->book.retailQuantity;
            head = head->next;
      }/* code */

  return result;
 /* WRITE THE CODE FOR THIS FUNCTION */
}

double calculateInvestmentInInventory(const Node *head) {
   double result = 0;
      while(head != NULL){
            /* If does not reach the end */
            result = result + ((head->book.wholesaleQuantity - head-
>book.retailQuantity)*head->book.wholesalePrice);
            head = head->next;
      }/* code */
  return result;
/* WRITE THE CODE FOR THIS FUNCTION */
}

double calculateTotalWholesaleCost(const Node *head) {
  double result = 0;

      while(head != NULL){
            /* If does not reach the end */
            result = result + (head->book.wholesaleQuantity * head-
>book.wholesalePrice);
            head = head->next;
      }/* code */

  return result;
/* WRITE THE CODE FOR THIS FUNCTION */
}

double calculateTotalProfit(const Node *head) {

       return calculateTotalRevenue(head) - calculateTotalWholesaleCost(head) +
calculateInvestmentInInventory(head);
/* WRITE THE CODE FOR THIS FUNCTION */
}

int calculateTotalBooksSold(const Node *head) {
  int result = 0;

      while(head != NULL){
            /* If does not reach the end */
            result = result + head->book.retailQuantity;
            head = head->next;
      }/* code */

  return result;
```

```
/* WRITE THE CODE FOR THIS FUNCTION */
}

double calculateAverageProfit(const Node *head) {

  return calculateTotalProfit(head)/ calculateTotalBooksSold(head);
/* WRITE THE CODE FOR THIS FUNCTION */
}

void delete(Node ** headPtr, int stockNumToDelete) {

  Node *traversePtr;
  traversePtr = *headPtr;
  if(traversePtr == NULL){
        /* If the list is empty */
        printf("Error: try to delete node from an empty list. \n");
  }
  else if(traversePtr->book.stockNumber == stockNumToDelete){
        *headPtr = traversePtr->next;
        free(traversePtr);
        printf("Book stock number %i deleted from the inventory \n",
stockNumToDelete);}

else{
     Node *priorNodePtr = traversePtr;
     traversePtr = traversePtr->next;
            while(traversePtr != NULL && traversePtr->book.stockNumber !=
stockNumToDelete){

               traversePtr = traversePtr->next;
            }
            if(traversePtr !=NULL){
            priorNodePtr->next = traversePtr->next;
          free(traversePtr);
            printf("Book stock number %i deleted from the inventory \n",
stockNumToDelete);
            }
            else{
                  printf("Error: Book stock number (stockNumber) not found in the
inventory. \n");
            }
    }

  }
/* WRITE THE CODE FOR THIS FUNCTION */

void printList(const Node *head) {
      const Node *traversePtr = head;
      printf("\nBook list:\n");
      while (traversePtr != NULL) {      /* determine not at end of list */
            printf("%s\n", traversePtr->book.title);
            traversePtr = traversePtr->next;
      }
      printf("\n");
}

void freeAllNodes(Node **headPtr) {
      Node *traversePtr = *headPtr;
      Node *restOfListPtr = *headPtr;
      while (restOfListPtr != NULL) {         /* determine list is not empty */
            restOfListPtr = restOfListPtr->next;
            free(traversePtr);
            traversePtr = restOfListPtr;
      }
      *headPtr = NULL; /* set headPtr back to NULL after space freed */
}
```