// Seth, there are some functions that are labled structures

Simple Inventory and Transaction

Tracking Program  
By

Arguelles, Zeth Raphael T.,   
Cuevas, Gabriel Dylan M.

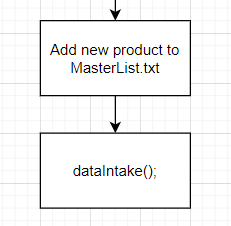
The Problem

ABC Trading is a company that sells school products in a wholesale manner. They are keeping a list of all their products in the inventory master file. The file contains the product ID, product description, supplier name, item price and item qty, ROP (reorder point), category and UOM (unit of measurement). The company also keeps file for sales, purchases and out of stock items. The SALES FILE is used to record all the sales transactions for all the items. The PURCHASES FILE is used to record all the items that were delivered. It is also used to update the master file of all the changes for a certain product. The OUT OF STOCK is used to record all the items that need to be replenished or reordered. To know which item to order, the company set a reorder for all their items. All items with a price lower than 200 has a reorder point quantity of 5, all items with a price ranging from 200 –500 pesos has a reorder point of 10 and beyond that, the item has a reorder point of 15. If the quantity of an item reached its reorder point or become lower than the reorder point, the company will putt hat item in the out of stock list to be reordered soon.

The Solution

The dynamic duo was hired in order to create a simple inventory management and sales tracking software for a certain wholesale company. While viewing the problem the duo was presented with, the duo had a pretty grasp clear on the 3 things that they have to keep in mind in creating the program, these 3 things are.

* Reliability of the Program in terms of keeping records
  + The reliability of the program in terms of keeping records is very important as the company may lose money if the program encounters problems that may lead to the lost of data. This is why we’ve created a two way system shown below,



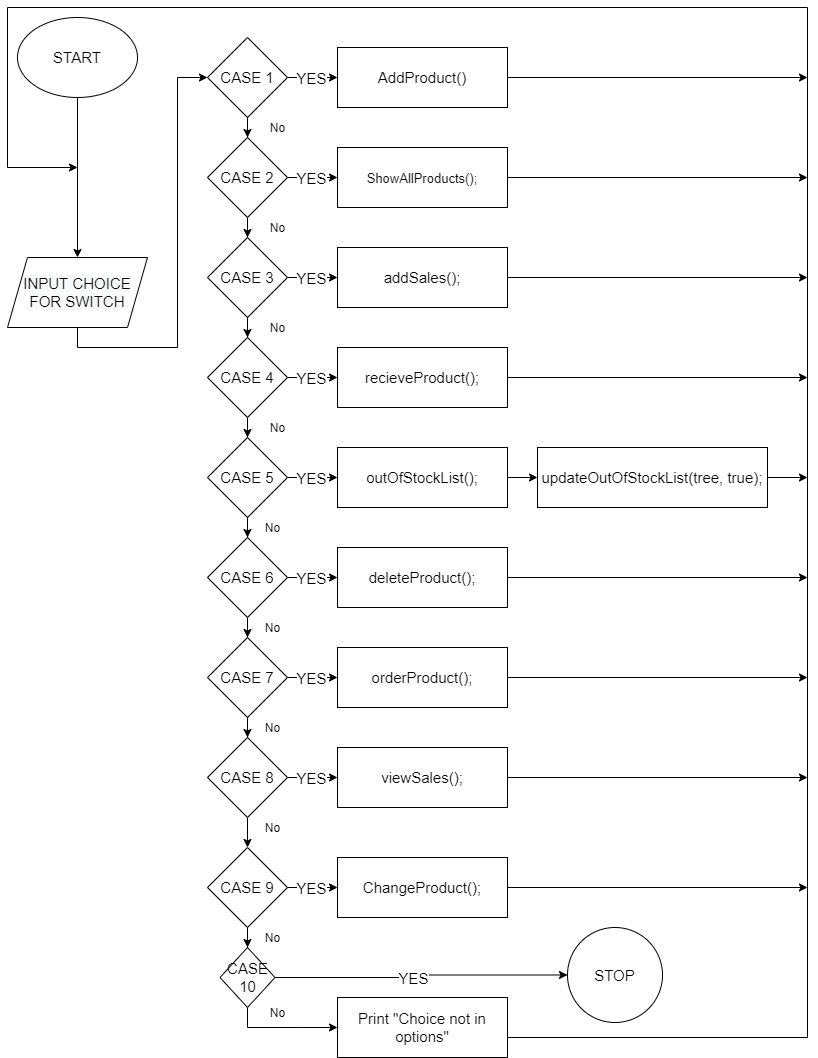
The flowchart above only shows one thing, it shows that the program will update both secondary and primary program at the same time, this ensures that the data would be saved for every transaction or change in the program.

* Scalability of the program

The program’s use of a binary tree architecture will allow it to be scalable to thousand of products within the company without any takeaway in performance, reliability, or speed.

* Speed of the program

The program uses a binary tree in storing the date of each product, there are implementations to the program that allows it to create the most efficient tree every time the program is run. Using the tree shall greatly increase the programs speed as it allows the program to traverse all the product in the least number of comparisons/searching step needed.

Program Flow

Above is the program flow of the main program. This entails a simple and elegant yet very effective solution. The dynamic duo has used the modularity of each function to for its advantage of each program in order to make the code easily understandable and easily modifiable in the case where the company must add or remove features from the program itself.

Now there are 4 main functions that is essential in the business operations of the program,

* + Order Product
    - This is a function made in order to order products that are out of stock. Below is a flow chart on how it works.

Diagram

Description automatically generated

* + Receive Product
    - Is a function that is created in order to receive those products that are ordered and has arrived the facility, below is a flowchart of this function.

Diagram

Description automatically generated

* + Add Sales
    - Is a function that is created in order to add sales of products into the system, below is a function on how it works.

Diagram, shape, polygon

Description automatically generated

* + Add Product
    - Is a function that allows the user to add a new product line into there system, below is a flowchart showing how it works.

Diagram

Description automatically generated

Additionally, there are 2 helper functions that are also essential to these business operations, namely they are:

* + Show All Products
    - A Function that simply shows all the products in the system, there is a variety of this function that only shows products whose quantity is at or below the reorder point

Diagram

Description automatically generated

* + Delete Product
    - A function that is created in order to delete or discontinue products

Diagram

Description automatically generated

There are many more functions that are included, however, due to there sheer number, the duo has decided to focus on the important front end function on this paper, however, there would be a in depth view of both the code and analysis of the functions within that may shed light on how these functions work

The Code

1. #include <iostream>
2. #include <string>
3. #include <iomanip>
4. #include <queue>
5. #include <fstream>
6. #include <sstream>
7. #include <windows.h>
8. #include <stdio.h>
9. #include <vector>

12. **HANDLE** console = GetStdHandle(STD\_OUTPUT\_HANDLE);
14. **using** **namespace** std;

17. **struct** bstnode {
18. **int** productID;
19. string productDescrption;
20. string supplierName;
21. **float** itemPrice;
22. **int** itemQuantity;
23. **int** reorderPoint;
24. string category;
25. string unitOfMeasurement;
26. **int** totalSales = 0;
27. **struct** bstnode \*left, \*right;
28. };
30. **struct** comparePID {
31. **bool** operator()(bstnode\* **const**& i1, bstnode\* **const**& i2) {
32. **return** i1->productID > i2->productID;
33. }
34. };
35. **struct** bstnode \*newNode(bstnode \*newProduct); // this gay
36. **struct** bstnode \* minValueNode(**struct** bstnode\*node);
37. **struct** bstnode\* insert(**struct** bstnode\* node, bstnode \*newProduct);
38. **struct** bstnode\* deleteNode(**struct** bstnode\* root, **int** productID);
39. bstnode\* sortedArrayToBST(vector<bstnode\*> arr,  **int** start, **int** end);
40. **void** treeClear(bstnode\*& root);
41. **struct** bstnode\* ProductIDFinder(**struct** bstnode\* root, **int** productID);
43. **void** cinClearer();
44. string consoleToTextFile(string consoleText);
45. string textFileToConsole(string textFileText);
47. //Functions that executes the features of the program
48. **void** addProduct();//mine
49. **void** dataIntake();// mine
50. **void** updateMasterList(**struct** bstnode \*root, **bool** isfirst); //mine
51. **void** showAllProducts();//zeth
52. **void** outOfStockList();//zeth
53. **void** showAllProductstoReorder();
54. **void** updateOutOfStockList(**struct** bstnode \*root, **bool** isfirst);//zeth
55. **void** deleteProduct(); // zeth
56. **bool** checkIfProductAlreadyExists(**struct** bstnode \*root, **int** key);
57. **bool** checkIfQuantityExists(**struct** bstnode \*root, **int** id, **int** qt);
58. **void** updateSalesFile(**struct** bstnode \*root, **int** quantity);
59. **void** addSales(); //zeth
60. **void** orderProduct();
61. **void** updateOrderFile(**struct** bstnode \*root, **int** quantity);
62. **void** recieveProduct();
63. **void** viewSales();
64. **void** clearSales(**struct** bstnode \*root);
65. **void** showSales(**struct** bstnode \*root);
66. **bool** ROPchecker(){
67. ifstream data;
68. **int** counter = 0;
69. data.open("MasterList.txt");
70. string dummy;
72. getline(data, dummy);
74. **int** tempproductID;
75. string tempproductDescrption;
76. string tempsupplierName;
77. **float** tempitemPrice;
78. **int** tempitemQuantity;
79. **int** tempreorderPoint;
80. string tempcategory;
81. string tempunitOfMeasurement;
82. **while**(data >> tempproductID >> tempproductDescrption >> tempsupplierName >> tempitemPrice >> tempitemQuantity >> tempreorderPoint >> tempcategory >> tempunitOfMeasurement){
83. **if** (tempitemQuantity  <= tempreorderPoint){
84. counter++;
85. }
86. }
87. **return** counter > 0;
89. }
91. **struct** bstnode \*tree = NULL;
93. **void** productChanger(){
94. system("cls");
95. showAllProducts();
96. **int** productID;
97. **int** choice;
98. cout << "\n Enter the product ID of the product : ";
99. cin >>productID;
100. cout << endl << "Choose Product detail to change";
101. cout << endl << endl << "[1] Price" ;
102. cout << endl << "[2] Description" ;
103. cout << endl << "[3] Suplier Name" ;
104. cout << endl << "[4] Unit Of Meausure" ;
106. cout << endl << endl << "Choice :";
108. cin >> choice;
109. **while**(choice < 0 || choice > 4){
110. cout << "Invalid Input\n Choice : ";
111. cin >> choice;
112. }
113. system("cls");
114. **switch** (choice){
115. **case** 1:
116. cout << "Item Price : " << ProductIDFinder(tree,productID)->itemPrice << endl;
117. cout << "New Item Price :";
118. cin >>ProductIDFinder(tree,productID)->itemPrice;
119. cinClearer();
120. //Check For ROP
121. **break**;
122. **case** 2:
123. cout << "Item Description : " << ProductIDFinder(tree,productID)->productDescrption << endl;
124. cout << "New Item Description :";
125. getline(cin,ProductIDFinder(tree,productID)->productDescrption);
126. **break**;
127. **case** 3:
128. cout << "Item Suplier Name : " << ProductIDFinder(tree,productID)->supplierName << endl;
129. cout << "New Item Suplier Name :";
130. getline(cin,ProductIDFinder(tree,productID)->supplierName);
131. **break**;
132. **case** 4:
133. cout << "Item UOM : " << ProductIDFinder(tree,productID)->unitOfMeasurement << endl;
134. cout << "New Item UOM:";
135. getline(cin,ProductIDFinder(tree,productID)->unitOfMeasurement);
136. **break**;

139. }
140. updateMasterList(tree,**true**);
141. }




147. **int** main(){
148. **while**(**true**){
149. RECT Window;
151. MoveWindow(GetConsoleWindow(), 0,0 , GetSystemMetrics(SM\_CXSCREEN) -600 ,  GetSystemMetrics(SM\_CYSCREEN), TRUE);// Gets the size of the monitor display in order to perfectly center the console
152. GetWindowRect(GetConsoleWindow(), &Window);
154. system("cls");
155. dataIntake();
156. **int** choice = 0;
158. // rearange this, I will do it, -cue
159. cout << "ABC Trading Company Inventory Management(Design Pending)" << endl << endl;
160. cout << "1. Add Product((product ID validation pending)" << endl;
161. cout << "2. Show all Products" << endl;
162. cout << "3. Record Sale of Products" << endl;
163. cout << "4. Recieve Products" << endl;
164. cout << "5. Show Items Needed to be reordered" << endl;
165. cout << "6. Delete Product" << endl;
166. cout << "7. Order Product" << endl;
167. cout << "8. View Sales" << endl;
168. cout << "9. Change Product Detail"<< endl;
169. cout << "10. Exit"<< endl;
171. **if**(ROPchecker()){
172. cout << "!!!SOME ITEMS ARE LOW IN STOCK!!!";
173. }
175. cout << "Enter Choice : ";
176. cin >> choice;
177. cinClearer();
178. system("cls");
179. **switch** (choice)
180. {
181. **case** 1:
182. addProduct();
183. **break**;
184. **case** 2:
185. showAllProducts();
186. cout << "\n\nPress any Enter to return to main menu\n\n";
187. cinClearer();
188. **break**;
189. **case** 3:
190. addSales();
191. **break**;
192. **case** 4:
193. recieveProduct();
194. **break**;
195. **case** 5:
196. outOfStockList();
197. **break**;
198. **case** 6:
199. showAllProducts();
200. deleteProduct();
201. **break**;
202. **case** 7:
203. orderProduct();
204. **break**;
205. **case** 8:
206. viewSales();
207. **break**;
208. **case** 9:
209. productChanger();
210. **case** 10:
211. exit(1);
212. **default**:
213. system("cls");
214. cout << "Choice not in choices, press enter to try again";
215. cinClearer();
216. **break**;
217. }
218. }
219. }
221. **void** viewSales(){// new function, used view sales
222. // rename variables so thats its not dead
223. ifstream salesFile;
224. string dummy; // by definition, this is used to contain the unencesasary
225. **int** productID, totalSales;
226. salesFile.open("SalesFile.txt");
227. getline(salesFile, dummy);
228. clearSales(tree);
229. **while**(salesFile >> productID >> dummy >> totalSales>> dummy){
230. **if**(checkIfProductAlreadyExists(tree, productID)){
231. ProductIDFinder(tree, productID)->totalSales +=totalSales;
232. }
234. }
235. salesFile.close();
236. cout  << left << setw(15) <<"productID" << left << setw(30) <<"Item Name" << left << setw(15) << "Total Sales" << endl;
237. showSales(tree);
238. cout << endl << endl << "Press Enter to return to main Menu";
239. cinClearer();

242. }
244. **void** addProduct(){
245. **char** tryAgain;
246. **do**{
247. **struct** bstnode \*temp =  **new** **struct** bstnode(); // we need to find out why we're doing this
248. **bool** checkId = **false**;
249. **int** productId = 0;
250. **do** {
251. cout << "Enter Product ID :";
252. cin >> productId;
253. cinClearer();
254. **if** (productId <= 0) //verifies the product ID inputted by the user
255. {
256. cout << "Invalid ID number! Negative values not allowed. Please Try again.\n";
257. checkId = **true**;
258. system("pause");
259. **continue**;
260. }
262. **else**
263. {
264. **if**(checkIfProductAlreadyExists(tree, productId))
265. {
266. cout << "Product with entered ID number already exists! Please try again.\n";
267. checkId = **true**;
268. system("pause");
269. **continue**;
270. }
271. **else**
272. { checkId = **false**; }
273. }
274. } **while**(checkId);
276. temp->productID = productId;
277. cout << "Enter Product Description :";
278. getline(cin,temp->productDescrption);
279. cout << "Enter Supplier Name :";
280. getline(cin, temp->supplierName);
281. cout << "Enter Product Price :";
282. cin >> temp->itemPrice;
283. cinClearer();
284. **while**(temp->itemPrice <= 0){
285. cout <<  "\n Product Price cannot be negative\n";
286. cout << "Enter Product Price :";
287. cin >> temp->itemPrice;
288. }
290. cout << "Enter Item quantity :";
291. cin >> temp->itemQuantity;
292. cinClearer();
293. **while**(temp->itemQuantity <= 0){
294. cout <<  "\n Item Quantitiy cannot be negative\n";
295. cout << "Enter Item quantity :";
296. cin >> temp->itemQuantity;
297. cinClearer();
298. }
299. cout << "Enter category :";
300. getline(cin,  temp->category );
301. cout << "Enter Unit of Measurement:";
302. getline(cin, temp->unitOfMeasurement);
303. **if**(temp->itemPrice < 200){
304. temp->reorderPoint = 5;
305. }**else** **if**(temp->itemPrice >= 200 && temp->itemPrice <= 500){
306. temp->reorderPoint = 10;
307. }**else**{
308. temp->reorderPoint =15;
309. }
310. updateMasterList(tree,**true**);
311. dataIntake();
312. cout << "Product has been added" << endl;
313. **do**{
314. cout << "Try again???[Y/N]";
315. cin >> tryAgain;
316. cinClearer();
317. tryAgain = toupper(tryAgain);
318. }**while**(tryAgain != 'Y' && tryAgain != 'N');
319. }**while**(toupper(tryAgain) == 'Y');
320. }
322. **void** cinClearer(){
323. cin.clear();
324. cin.ignore(1000, '\n');
325. }
327. string consoleToTextFile(string consoleText){
328. **for**(**int** i = 0 ; i < consoleText.length();i++){
329. **if**(consoleText.at(i) == ' '){
330. consoleText.at(i) = '`';
331. }
332. }
333. **return** consoleText;
334. }
335. string textFileToConsole(string textFileText){
336. **for**(**int** i = 0 ; i < textFileText.length();i++){
337. **if**(textFileText.at(i) == '`'){
338. textFileText.at(i) = ' ';
339. }
340. }
341. **return** textFileText;
342. }
344. **void** dataIntake(){
345. ifstream data;
346. **int** counter = 0;
347. data.open("MasterList.txt");
348. string dummy;
349. priority\_queue<bstnode\*,vector<bstnode\*>, comparePID> container;
351. treeClear(tree);
352. getline(data, dummy);
354. **int** tempproductID;
355. string tempproductDescrption;
356. string tempsupplierName;
357. **float** tempitemPrice;
358. **int** tempitemQuantity;
359. **int** tempreorderPoint;
360. string tempcategory;
361. string tempunitOfMeasurement;
362. **while**(data >> tempproductID >> tempproductDescrption >> tempsupplierName >> tempitemPrice >> tempitemQuantity >> tempreorderPoint >> tempcategory >> tempunitOfMeasurement){
363. **struct** bstnode \*temp = **new** **struct** bstnode();
364. temp->itemPrice = tempitemPrice;
365. temp->itemQuantity = tempitemQuantity;
366. temp->productID = tempproductID;
367. temp->reorderPoint = tempreorderPoint;
368. temp->productDescrption = textFileToConsole(tempproductDescrption);
369. temp->supplierName = textFileToConsole(tempsupplierName);
370. temp->category = textFileToConsole(tempcategory);
371. temp->unitOfMeasurement = textFileToConsole(tempunitOfMeasurement);
372. container.push(temp);
373. }
374. data.close();
375. vector<bstnode\*> arr;
376. **while**(!container.empty()){
377. arr.push\_back(container.top());
378. counter++;
379. container.pop();
381. }
383. tree = sortedArrayToBST(arr,0, arr.size()-1);
384. }
386. **void** updateMasterList(**struct** bstnode \*root, **bool** isfirst){
387. // This is a core function, this must not be changed
388. **if** (root != NULL) {
389. **if**(isfirst){
390. ofstream creator("MasterList.txt");
391. **if**(!creator.is\_open()){
392. system("cls");
393. cout << "MasterList Cannot be oppened, press enter to exit the program";
394. cinClearer();
395. exit(1);
396. }
397. creator.close();
398. }
399. fstream master;
400. master.open("MasterList.txt", ios::app);
402. **if**(!master.is\_open()){
403. system("cls");
404. cout << "MasterList Cannot be oppened, press enter to exit the program";
405. cinClearer();
406. exit(1);
407. }**else**{
408. **if**(isfirst){
409. master << left << setw(15) << "Product ID" << left << setw(30) << "Product Description"  << left << setw(15) << "Suplier Name" << left << setw(10) <<"Price"<< left << setw(15) <<"Quantitiy" << left << setw(15) <<"ROP"<< left << setw(15) << "Category"<< left << setw(15) << "Unit Of Measurment"<<endl;
410. }
411. master.close();
412. master.open("MasterList.txt", ios::app);
413. master << left << setw(15) << root->productID << left << setw(30) <<consoleToTextFile(root->productDescrption)   << left << setw(15) << consoleToTextFile(root->supplierName) << left << setw(10) <<root->itemPrice<< left << setw(15) <<root->itemQuantity << left << setw(15) <<root->reorderPoint<< left << setw(15) << consoleToTextFile(root->category)<< left << setw(15) << consoleToTextFile(root->unitOfMeasurement)<<endl;
414. master.close();
415. }
416. updateMasterList(root->left,**false**);
417. updateMasterList(root->right,**false**);
418. }
419. }
421. **void** updateOutOfStockList(**struct** bstnode \*root, **bool** isfirst)
422. {
423. **if** (root != NULL) {
424. **if** (isfirst) {
425. ofstream creator("OutOfStockList.txt");
426. **if** (!creator.is\_open()) {
427. system("cls");
428. cout << "OutOfStockList Cannot be oppened, press enter to exit the program";
429. cinClearer();
430. exit(1);
431. }
432. creator.close();
433. }
434. fstream master;
435. master.open("OutOfStockList.txt", ios::app);
437. **if** (!master.is\_open()) {
438. system("cls");
439. cout << "OutOfStockList Cannot be oppened, press enter to exit the program";
440. cinClearer();
441. exit(1);
442. }
443. **else** {
444. **if** (isfirst) {
445. master << left << setw(15) << "Product ID" << left << setw(30) << "Product Description" << left << setw(15) << "Suplier Name" << left << setw(10) << "Price" << left << setw(15) << "Quantitiy" << left << setw(15) << "ROP" << left << setw(15) << "Category" << left << setw(15) << "Unit Of Measurment" << endl;
446. }
447. master.close();
448. master.open("OutOfStockList.txt", ios::app);
449. **if** (root->itemQuantity <= root->reorderPoint)
450. {
451. // Must change this into a more stream line view, so the product ID, product decription, quantitiy, and rop are only seen.
452. master << left << setw(15) << root->productID << left << setw(30) << consoleToTextFile(root->productDescrption) << left << setw(15) << consoleToTextFile(root->supplierName) << left << setw(10) << root->itemPrice << left << setw(15) << root->itemQuantity << left << setw(15) << root->reorderPoint << left << setw(15) << consoleToTextFile(root->category) << left << setw(15) << consoleToTextFile(root->unitOfMeasurement) << endl;
453. }
454. master.close();
455. }
456. updateOutOfStockList(root->left, **false**);
457. updateOutOfStockList(root->right, **false**);
458. }
459. }
461. **void** updateSalesFile(**struct** bstnode \*root, **int** quantity){
462. ifstream input;
463. string dummy;
465. input.open("SalesFile.txt");
466. getline(input, dummy);
467. input.close();
469. ofstream output;
470. output.open("SalesFile.txt",ios::app);
471. **if**(dummy == ""){
472. output  << left << setw(15) << "Product ID" << left << setw(30) << "Product Description"  << left << setw(30) << "Order Quantity" << endl;
473. }
474. output  << left << setw(15) << root->productID << left << setw(30) << consoleToTextFile(root->productDescrption) << left << setw(30) << quantity << consoleToTextFile(root->unitOfMeasurement) << endl;
475. output.close();
476. }
478. **void** updateOrderFile(**struct** bstnode \*root, **int** quantity)
479. {
480. ifstream input;
481. string dummy;
483. input.open("orderFile.txt");
484. getline(input, dummy);
485. input.close();
487. ofstream output;
488. output.open("orderFile.txt",ios::app);
489. **if**(dummy == ""){
490. output  << left << setw(15) << "Product ID" << left << setw(30) << "Product Description"  << left << setw(30) << "Order Quantity" << endl;
491. }
492. output  << left << setw(15) << root->productID << left << setw(30) << consoleToTextFile(root->productDescrption) << left << setw(30) << quantity << endl;
493. output.close();
494. }
496. **void** showAllProducts(){
497. ifstream data;
498. **int** counter = 0;
499. data.open("MasterList.txt");
500. string dummy;
501. priority\_queue<bstnode\*,vector<bstnode\*>, comparePID> container;
503. getline(data, dummy);
505. **int** tempproductID;
506. string tempproductDescrption;
507. string tempsupplierName;
508. **float** tempitemPrice;
509. **int** tempitemQuantity;
510. **int** tempreorderPoint;
511. string tempcategory;
512. string tempunitOfMeasurement;
513. **while**(data >> tempproductID >> tempproductDescrption >> tempsupplierName >> tempitemPrice >> tempitemQuantity >> tempreorderPoint >> tempcategory >> tempunitOfMeasurement){
514. **struct** bstnode \*temp = **new** **struct** bstnode();
515. temp->itemPrice = tempitemPrice;
516. temp->itemQuantity = tempitemQuantity;
517. temp->productID = tempproductID;
518. temp->reorderPoint = tempreorderPoint;
519. temp->productDescrption = textFileToConsole(tempproductDescrption);
520. temp->supplierName = textFileToConsole(tempsupplierName);
521. temp->category = textFileToConsole(tempcategory);
522. temp->unitOfMeasurement = textFileToConsole(tempunitOfMeasurement);
523. container.push(temp);
524. }
525. data.close();
526. system("cls");
528. cout  << left << setw(15) << "Product ID" << left << setw(30) <<"DESCRIPTION" << left << setw(15) << "Suplier Name" << left << setw(10) <<"Price"<< left << setw(15) <<"Item Quantity" << left << setw(15) <<"Reorder Point"<< left << setw(15) << "Category"<< left << setw(15) << "Unit Of Measurement"<<endl;
529. **while**(!container.empty()){
530. //D- Do not worry about this, I will change window size in order to acommodate size of this thing.
531. cout  << left << setw(15) << container.top()->productID << left << setw(30) <<textFileToConsole(container.top()->productDescrption) << left << setw(15) << textFileToConsole(container.top()->supplierName) << left << setw(10) <<container.top()->itemPrice<< left << setw(15) <<container.top()->itemQuantity << left << setw(15) <<container.top()->reorderPoint<< left << setw(15) << textFileToConsole(container.top()->category)<< left << setw(15) << textFileToConsole(container.top()->unitOfMeasurement)<<endl;
532. container.pop();
533. }
535. }
537. **void** showAllProductstoReorder(){
538. ifstream data;
539. **int** counter = 0;
540. data.open("MasterList.txt");
541. string dummy;
542. priority\_queue<bstnode\*,vector<bstnode\*>, comparePID> container;
544. getline(data, dummy);
546. **int** tempproductID;
547. string tempproductDescrption;
548. string tempsupplierName;
549. **float** tempitemPrice;
550. **int** tempitemQuantity;
551. **int** tempreorderPoint;
552. string tempcategory;
553. string tempunitOfMeasurement;
554. **while**(data >> tempproductID >> tempproductDescrption >> tempsupplierName >> tempitemPrice >> tempitemQuantity >> tempreorderPoint >> tempcategory >> tempunitOfMeasurement){
555. **struct** bstnode \*temp = **new** **struct** bstnode();
556. temp->itemPrice = tempitemPrice;
557. temp->itemQuantity = tempitemQuantity;
558. temp->productID = tempproductID;
559. temp->reorderPoint = tempreorderPoint;
560. temp->productDescrption = textFileToConsole(tempproductDescrption);
561. temp->supplierName = textFileToConsole(tempsupplierName);
562. temp->category = textFileToConsole(tempcategory);
563. temp->unitOfMeasurement = textFileToConsole(tempunitOfMeasurement);
564. container.push(temp);
565. }
566. data.close();
567. system("cls");
569. cout  << left << setw(15) << "Product ID" << left << setw(30) <<"DESCRIPTION" << left << setw(15) << "Suplier Name" << left << setw(10) <<"Price"<< left << setw(15) <<"Item Quantity" << left << setw(15) <<"Reorder Point"<< left << setw(15) << "Category"<< left << setw(15) << "Unit Of Measurement"<<endl;
570. **while**(!container.empty()){
571. **if**(container.top()->itemQuantity <= container.top()->reorderPoint){
572. cout  << left << setw(15) << container.top()->productID << left << setw(30) <<textFileToConsole(container.top()->productDescrption) << left << setw(15) << textFileToConsole(container.top()->supplierName) << left << setw(10) <<container.top()->itemPrice<< left << setw(15) <<container.top()->itemQuantity << left << setw(15) <<container.top()->reorderPoint<< left << setw(15) << textFileToConsole(container.top()->category)<< left << setw(15) << textFileToConsole(container.top()->unitOfMeasurement)<<endl;
573. }
575. container.pop();
576. }
578. }



583. **void** outOfStockList()
584. {
585. ifstream data;
586. **int** counter = 0;
587. data.open("MasterList.txt");
588. string dummy;
589. priority\_queue<bstnode\*,vector<bstnode\*>, comparePID> container;
591. getline(data, dummy);
593. **int** tempproductID;
594. string tempproductDescrption;
595. string tempsupplierName;
596. **float** tempitemPrice;
597. **int** tempitemQuantity;
598. **int** tempreorderPoint;
599. string tempcategory;
600. string tempunitOfMeasurement;
601. **while**(data >> tempproductID >> tempproductDescrption >> tempsupplierName >> tempitemPrice >> tempitemQuantity >> tempreorderPoint >> tempcategory >> tempunitOfMeasurement){
602. **struct** bstnode \*temp = **new** **struct** bstnode();
603. temp->itemPrice = tempitemPrice;
604. temp->itemQuantity = tempitemQuantity;
605. temp->productID = tempproductID;
606. temp->reorderPoint = tempreorderPoint;
607. temp->productDescrption = textFileToConsole(tempproductDescrption);
608. temp->supplierName = textFileToConsole(tempsupplierName);
609. temp->category = textFileToConsole(tempcategory);
610. temp->unitOfMeasurement = textFileToConsole(tempunitOfMeasurement);
611. container.push(temp);
612. }
613. data.close();
614. system("cls");
615. cout << "Out of Stock List: \n";
616. **while**(!container.empty()){
617. **if** (container.top()->itemQuantity <= container.top()->reorderPoint){
618. //D- Do not worry about this, I will change window size in order to acommodate size of this thing.
619. cout  << left << setw(15) << container.top()->productID << left << setw(30) <<consoleToTextFile(container.top()->productDescrption) << left << setw(15) << consoleToTextFile(container.top()->supplierName) << left << setw(15) <<container.top()->itemQuantity<< consoleToTextFile(container.top()->unitOfMeasurement) << left << setw(15) <<container.top()->reorderPoint<< consoleToTextFile(container.top()->unitOfMeasurement)<< left << setw(15) << consoleToTextFile(container.top()->category)<<endl;
620. }
621. container.pop();
622. }
623. updateOutOfStockList(tree, **true**);
624. }
626. **bool** checkIfProductAlreadyExists(**struct** bstnode \*root, **int** key)
627. {
628. **if**(root == NULL)
629. {
630. **return** **false**;
631. }
632. **if**(root->productID == key)
633. {
634. **return** **true**;
635. }
636. **if**(root->productID > key)
637. {
638. **return** checkIfProductAlreadyExists(root->left, key);
639. }
640. **return** checkIfProductAlreadyExists(root->right, key);
641. }
642. **bool** checkIfQuantityExists(**struct** bstnode \*root, **int** id, **int** qt)
643. {
644. **if**(root == NULL)
645. {
646. **return** **false**;
647. }
648. **if**(root->productID == id && root->itemQuantity < qt)
649. {
650. **return** **true**;
651. }
652. **if**(root->productID > id)
653. {
654. **return** checkIfQuantityExists(root->left, id, qt);
655. }
656. **return** checkIfQuantityExists(root->right, id, qt);
657. }
659. **void** deleteProduct()
660. {
661. **int** productId = 0;
662. **bool** idChecker = **false**;
663. cout << "Product Deletion: \n";
664. **do** {
665. cout << "Enter Product ID: "; cin >> productId;
666. cinClearer();
667. **if** (productId <= 0)
668. {
669. cout << "Invalid ID number! Negative values not allowed. Please Try again.\n";
670. idChecker = **true**;
671. system("pause");
672. }
673. **else**
674. {
675. **if**(checkIfProductAlreadyExists(tree, productId))
676. {
677. idChecker = **false**;
678. }
679. **else**
680. {
681. cout << "This product does not exist! Please enter valid ID.\n";
682. idChecker = **true**;
683. system("pause");
684. }
685. }
686. } **while**(idChecker);
687. tree = deleteNode(tree, productId);
688. updateMasterList(tree,**true**);
689. dataIntake();
690. }
692. **void** addSales()
693. {
694. **int** counter = 0;
695. **char** option;
696. **do** {
697. showAllProducts();
698. **int** productId = 0;
699. **int** productQuantity = 0;
700. **bool** verification = **false**;
701. **while**(**true**){
702. cout << "Enter Product ID: "; cin >> productId; cout << endl;
703. cinClearer();
704. **if** (!checkIfProductAlreadyExists(tree, productId)){
705. cout << "Product does not exist\n";
706. verification = **true**;
707. system("pause");
708. **continue**;
709. }
710. **break**;
711. }
713. **do**
714. {
715. cout << "Enter sold quantity: "; cin >> productQuantity; cout << endl;
716. cinClearer();
717. **if** (productId < 0 || productQuantity <= 0)
718. {
719. cout << "Invalid Inputs! Please try again\n";
720. verification = **true**;
721. system("pause");
722. }
723. **else**
724. {
725. **if**(!checkIfQuantityExists(tree, productId, productQuantity))
726. {
727. ProductIDFinder(tree,productId)->itemQuantity -= productQuantity;
728. verification = **false**;
729. updateMasterList(tree, **true**);
730. dataIntake();
731. }
732. **else**
733. {
734. cout << "Quantity entered exceeds of what is available in stock\n";
735. verification = **true**;
736. system("pause");
737. }


741. }
742. } **while**(verification);
744. updateSalesFile(ProductIDFinder(tree,productId), productQuantity);
745. system("cls");
746. showAllProducts();
747. cout << "Enter Product ID: " << productId;
748. cout << "\n\nEnter Sold Quantity : " << productQuantity << endl;
749. cout << "\nProduct sale succesfully recorded" << endl;
750. cout << "Make another entry?[Y/N]: "; cin >> option;
751. counter++;
752. option = toupper(option);
753. } **while**(option == 'Y');
754. }
756. **void** orderProduct()
757. {
758. showAllProductstoReorder();
759. **char** tryAgain;
760. **do**
761. {
762. **int** productId = 0;
763. **int** productQuantity = 0;
764. **bool** verify = **false**;
765. **do**
766. {
767. cout << "Enter Product ID: "; cin >> productId; cout << endl;
768. cinClearer();
769. **if** (!checkIfProductAlreadyExists(tree, productId)){
770. cout << "Product does not exist\n";
771. verify = **true**;
772. system("pause");
773. **continue**;
774. }
775. cout << "Enter Desired Quantity: "; cin >> productQuantity; cout << endl;
776. **if** (productId < 0 || productQuantity < 0)
777. {
778. cout << "Invalid Inputs! Please try again\n";
779. verify = **true**;
780. system("pause");
781. }
782. **else**
783. {
784. **if** (checkIfProductAlreadyExists(tree, productId))
785. {
786. verify = **false**;
787. }
789. }
790. } **while** (verify);
791. updateOrderFile(ProductIDFinder(tree, productId), productQuantity);
792. cout << "Make another entry?[Y/N]: "; cin >> tryAgain;
793. tryAgain = toupper(tryAgain);
794. } **while** (tryAgain == 'Y');
795. }
797. **void** recieveProduct()
798. {
799. **int** orderNumber =1;
800. fstream orderFile;
801. string infoLine;
802. orderFile.open("orderFile.txt", ios::in);
803. **if** (orderFile.is\_open())
804. {
805. getline(orderFile,infoLine);
806. cout << "Order Number" <<infoLine << endl;
807. **while**(getline(orderFile, infoLine))
808. {
809. cout << "     "<<orderNumber<<"      "<<textFileToConsole(infoLine) << endl;
810. orderNumber++;
811. }
812. }
813. **else**
814. {
815. cout << "No order has Been created;" << endl;
816. }
818. orderFile.close();
819. **if**(orderNumber == 1){
820. system("cls");
821. cout << "No Order has been created";
822. cout << endl << "Press enter to return to main menu";
823. cinClearer();
824. **return**;
825. }
826. **int** productId;
827. **int** orderAmmount;
828. **int** counter = 0;
829. cout << "Enter the ID of product to recieve: "; cin >> productId;
830. cout << endl;
831. cinClearer();
832. ifstream orderReceive;
833. ofstream copy;
834. ofstream receivedList;
835. string transferer;
836. string receivedLine;
837. orderReceive.open("orderFile.txt");
838. copy.open("copy.txt");
839. receivedList.open("Recieved.txt", ios::app);
841. **while**(!orderReceive.eof()){
842. **if**(counter != productId){
843. getline(orderReceive,transferer);
844. copy << transferer;
845. **if**(!orderReceive.eof()){
846. copy << endl;
847. }
848. }**else**{
849. getline(orderReceive,transferer);
850. receivedList << transferer << endl;
851. receivedLine = transferer;
852. }
853. counter++;
854. }
855. orderReceive.close();
856. copy.close();
857. receivedList.close();
859. copy.open("orderFile.txt");
860. orderReceive.open("copy.txt");
861. **while**(!orderReceive.eof()){
862. getline(orderReceive,transferer);
863. copy << transferer;
864. **if**(!orderReceive.eof()){
865. copy << endl;
866. }
867. }
868. copy.close();
869. copy.open("copy.txt");
870. copy << receivedLine;
871. copy.close();
872. orderReceive.close();
873. orderReceive.open("copy.txt");
874. orderReceive >> productId >> transferer >> orderAmmount;
875. orderReceive.close();
876. **if**(checkIfProductAlreadyExists(tree, productId)){
877. ProductIDFinder(tree,productId)->itemQuantity += orderAmmount;
878. }**else**{
879. cout << "Product has been deleted, order file will be updated, however, it will not show up in the system";
880. }
882. orderReceive.close();
883. remove("copy.txt");
884. updateMasterList(tree, **true**);
885. }
886. // BS TREE Structs and Functions
888. **void** treeClear(bstnode\*& root) {
889. **if**(root!=NULL) {
890. treeClear(root->left);
891. treeClear(root->right);
892. **delete** root;
893. root = NULL;
894. }
895. }
897. **void** clearSales(**struct** bstnode \*root) {
898. **if** (root != NULL) {
899. clearSales(root->left);
900. root->totalSales = 0;
901. clearSales(root->right);
902. }
903. }
904. **void** showSales(**struct** bstnode \*root){
905. **if** (root != NULL) {
906. showSales(root->left);
907. **if**(root->totalSales != 0){
908. cout  << left << setw(15) << root->productID << left << setw(30) <<textFileToConsole(root->productDescrption) << left << setw(15) << root->totalSales << endl;
909. }
910. showSales(root->right);
911. }
912. }
914. **struct** bstnode\* insert(**struct** bstnode\* node, bstnode \*newProduct) {
916. **if** (node == NULL)
917. {
918. **return** newNode(newProduct);
919. }
921. **if** (newProduct->productID < node->productID)
922. {
923. node->left  = insert(node->left, newProduct);
924. }
925. **else**
926. {
927. node->right = insert(node->right, newProduct);
928. }
929. **return** node;
930. }
932. bstnode\* sortedArrayToBST(vector<bstnode\*> arr,  **int** start, **int** end){
933. **if** (start > end){ **return** NULL; }
934. **int** mid = (start + end)/2;
935. bstnode \*temp = newNode(arr[mid]);
936. temp->left = sortedArrayToBST(arr, start, mid - 1);
937. temp->right = sortedArrayToBST(arr, mid + 1, end);
938. **return** temp;
939. }
941. **struct** bstnode \* minValueNode(**struct** bstnode\*node) {
942. **struct** bstnode\* current = node;
943. **while** (current->left != NULL) { current = current->left; }
944. **return** current;
945. }
947. **void** printBT(**const** string& prefix, **const** bstnode\* node, **bool** isLeft)
948. {
949. **if**( node != NULL )
950. {
952. cout << prefix;
953. **if**(isLeft){
954. cout << (**char**)195 << (**char**)196<< (**char**)196;
955. }**else**{
956. cout << (**char**)192 << (**char**)196<< (**char**)196;
958. }
959. cout << node->productID << endl;
960. string container = "";
961. container += (**char**)179;
962. container += "   ";
963. printBT( prefix + (isLeft ? container : "    "), node->left, **true**);
964. printBT( prefix + (isLeft ? container : "    "), node->right, **false**);
965. }
966. }
968. **struct** bstnode \*newNode(bstnode \*newProduct) {
969. **struct** bstnode \*temp =  **new** **struct** bstnode();
970. temp->category = newProduct->category;
971. temp->itemPrice = newProduct->itemPrice;
972. temp->itemQuantity = newProduct->itemQuantity;
973. temp->productDescrption = newProduct->productDescrption;
974. temp->productID = newProduct->productID;
975. temp->reorderPoint = newProduct->reorderPoint;
976. temp->supplierName = newProduct->supplierName;
977. temp->unitOfMeasurement = newProduct->unitOfMeasurement;
978. newProduct->left = NULL;
979. newProduct->right = NULL;
980. **return** temp;
981. }
983. **void** printBT(**const** bstnode\* node){
984. printBT("", node, **false**);
985. }
987. **struct** bstnode\* deleteNode(**struct** bstnode\* root, **int** productID) {
988. **if** (root == NULL) { **return** root; }
990. **if** (productID < root->productID)
991. { root->left = deleteNode(root->left, productID); }
992. **else** **if** (productID > root->productID)
993. { root->right = deleteNode(root->right, productID); }
994. **else**
995. {
996. **if** (root->left == NULL) {
997. **struct** bstnode \*temp = root->right;
998. free(root);
999. **return** temp;
1000. } **else** **if** (root->right == NULL) {
1001. **struct** bstnode \*temp = root->left;
1002. free(root);
1003. **return** temp;
1004. }
1005. **struct** bstnode\* temp = minValueNode(root->right);
1006. root->productID = temp->productID;
1007. root->right = deleteNode(root->right, temp->productID);
1008. }
1009. **return** root;
1010. }
1012. **struct** bstnode\* ProductIDFinder(**struct** bstnode\* root, **int** productID) {
1013. // Base Cases: root is null or key is present at root
1014. **if** (root == NULL || root->productID == productID){ **return** root; }
1016. **if** (root->productID < productID) { **return** ProductIDFinder(root->right, productID); }
1017. **return** ProductIDFinder(root->left, productID);
1018. }

Code Inventory

|  |  |
| --- | --- |
| **Libraries** | |
| **<iostream>** | Used for basic input and output operations |
| **<queue>** | Used for containing product information into a queue |
| **<string>** | Used for non-numeric information about product entries |
| **<iomanip>** | Used for the spacing of data in the text file and console |
| **<fstream>** | Used for the creation and manipulation of files |
| **<windows.h>** | Allows access to Win32 API functions |
| **<stdio.h>** | Allows the use of streams |

|  |  |  |
| --- | --- | --- |
| **Function** | | |
| Type | Name | Description |
| **Function(struct)** | inorder | Performs inorder traversal through the binary tree. |
| **Function(const string&, const bstnode, bool)** | printBT | Prints the binary search tree in a horizontal tree format. |
| **Function(const bstnode\*)** | printBT | Calls the previous function of the same name. |
| **Function(bstnode\*&)** | treeClear | Deletes all nodes present in the tree and returns root to NULL. |
| **Function(void)** | cinClearer | Cleans up input field. |
| **String(string)** | consoleToTextFile | Converts spaces in a string so the computer may read the text file. |
| **String(string)** | textFileToConsole | Converts the characters used to fill up the spaces in a text file back into blank spaces to be displayed in the console |
| **Function(void)** | addProduct | Creates a new product entry and takes information pertaining to said entry. |
| **Function(void)** | dataIntake | Reads from the Master List text file text file and puts data into a queue. |
| **Function(struct, bool)** | updateMasterList | Saves changes to the Master List text file. |
| **Function(void)** | showAllProducts | Prints all product entries from the Master List. |
| **Function(void)** | outOfStockList | Displays which products are potentially out of stock |
| **Function(struct, bool)** | updateOutOfStockList | Saves the entries of out of stock products to a text file |
| **Function(void)** | deleteProduct | Deletes a chosen entry from the Master List and Binary tree. |
| **Bool(struct, int)** | checkIfProductAlreadyExists | Returns true if a product ID already exists within the BST. |
| **Bool(struct, int, int)** | checkIfQuantityExists | Returns true if the quantity of chosen product is greater than the desired quantity. |
| **Function(struct, int)** | updateSalesFile | Stores sales transactions into a text file. |
| **Function(void)** | addSales | Validates and accepts user input of product ID and desired quantity. |
| **Function(void)** | orderProduct | Validates and accepts user input of product ID and order quantity. |
| **Function(struct, int)** | updateOrderFile | Stores order requests into text file. |
| **Function(void)** | recieveProduct | Reads order file and adds quantity to master list. |
| **Function(void)** | viewSales | Used to display the contents of the Sales file. |
| **Function(struct bstnode)** | clearSales | Returns total sales to zero |
| **Function(struct bstnode)** | showSales | Displays the product entries that have sales greater than zero from the BST |
| **Bool(void)** | ROPchecker | Returns true if product is calculated to be out of stock |
| **Function(void)** | productChanger | Edits the details of chosen product entry |

|  |  |  |
| --- | --- | --- |
| **Bstnode functions** | | |
| **Struct bstnode** | \*newNode | Used to create new nodes that contain a new entry for the binary tree |
| **Function(bstnode)** | \*minValueNode | Returns the node with the smallest key value. |
| **Function(struct bstnode, bstnode)** | \*insert | Adds a new node with a new product ID into the BST. |
| **Function(struct bstnode, int)** | \*deleteNode | Deletes the node with the corresponding product ID. |
| **Function(vector, int, int)** | sortedArrayToBST | Sorts an array and returns a struct. |
| **Function(struct bstnode, int)** | ProductIDFinder | Traverses through the BST to find the product entry with the same entered product ID. |

|  |  |  |
| --- | --- | --- |
| **Structs** | | |
| Type | Name | Description |
| **struct** | bstnode | A structure used a basis for the binary search tree. Stores important data related to the product entries. |
| **struct** | comparePID | A structure that allows priority queues to differentiate each entry using their unique product identification numbers. |

|  |  |  |
| --- | --- | --- |
| **Global Variables** | | |
| Type | Name | Description |
| **Struct bstnode** | tree | Initializes to NULL. |

|  |  |  |
| --- | --- | --- |
| **Bstnode Variables** | | |
| Type | Name | Description |
| **int** | productID | Stores identification number for each product entry |
| **string** | productDescription | Stores the name of each product |
| **string** | supplierName | Stores name of the supplier of each product |
| **float** | itemPrice | Stores the price value associated with each product |
| **int** | itemQuantity | Stores the amount each product has in stock |
| **int** | reorderPoint | Stores the reorder point of each product |
| **string** | category | Stores the name of the category each product is listed under |
| **string** | unitOfMeasurement | Stores the types of units used to measure the product’s quantity |
| **int** | totalSales | Stores the total sales of all products listed |
| **Struct bstnode** | \*left | Aids in the navigation of the BST. |
| **Struct bstnode** | \*right | Aids in the navigation of the BST. |

|  |  |  |
| --- | --- | --- |
| **Main Variables** | | |
| Type | Name | Description |
| **Int** | choice | This variable is used for selecting options in the main menu by accessing the cases in the switch function. |

|  |  |  |
| --- | --- | --- |
| **addproduct Variables** | | |
| Type | Name | Description |
| **Struct bstnode** | \*temp | Creates a temporary binary search tree. |
| **bool** | checkId | Verifies the product ID input by the user to see if it exists. |
| **Int** | productId | Stores the inputted product ID. |

|  |  |  |
| --- | --- | --- |
| **dataIntake Variables** | | |
| Type | Name | Description |
| **int** | counter | Used to increment within a while loop, |
| **string** | dummy | Used to store unnecessary data |
| **Priority\_queue** | container | Used to store all information retrieved from the text file |
| **int** | tempProductID | Used to temporarily store the product IDs read from the text file. |
| **String** | tempProductDescription | Used to temporarily store the product descriptions read from the text file. |
| **string** | tempsupplierName | Used to temporarily store the supplier’s read from the text file. |
| **float** | tempitemPrice | Used to temporarily store the prices read from the text file. |
| **int** | tempitemQuantity | Used to temporarily store the quantities read from the text file. |
| **Int** | tempreorderPoint | Used to temporarily store each product’s reorder point that were read from the text file. |
| **String** | tempcategory | Used to temporarily store the product categories read from the text file. |
| **string** | tempunitOfMeasurement | Used to temporarily store each product’s unit of measurement that were read from the text file. |

|  |  |  |
| --- | --- | --- |
| **updateMasterList Variables** | | |
| Type | Name | Description |
| **bool** | isFirst | Determines if a new text file should be created and if the first line needs to be written |
| **Struct bstnode** | \*root | Allows access to the data stored in the BST. |

|  |  |  |
| --- | --- | --- |
| **updateOutOfStockList Variables** | | |
| Type | Name | Description |
| **bool** | isFirst | Determines if a new text file should be created and if the first line needs to be written |
| **Struct bstnode** | \*root | Allows access to the data stored in the BST. |

|  |  |  |
| --- | --- | --- |
| **updateSalesFile Variables** | | |
| Type | Name | Description |
| **Struct bstnode** | \*root | Allows access to the data stored in the BST. |
| **int** | quantity | Stores the quantity inputted by the user |
| **String** | dummy | Stores the first line of the text file to see if it is blank |

|  |  |  |
| --- | --- | --- |
| **updateOrderFile Variables** | | |
| Type | Name | Description |
| **Struct bstnode** | \*root | Allows access to the data stored in the BST. |
| **int** | quantity | Stores the quantity inputted by the user |
| **String** | dummy | Stores the first line of the text file to see if it is blank |

|  |  |  |
| --- | --- | --- |
| **showAllProducts Variables** | | |
| Type | Name | Description |
| **string** | dummy | Used to store unnecessary data |
| **Priority\_queue** | container | Used to store all information retrieved from the text file |
| **int** | tempProductID | Used to temporarily store the product IDs read from the text file. |
| **String** | tempProductDescription | Used to temporarily store the product descriptions read from the text file. |
| **string** | tempsupplierName | Used to temporarily store the supplier’s read from the text file. |
| **float** | tempitemPrice | Used to temporarily store the prices read from the text file. |
| **int** | tempitemQuantity | Used to temporarily store the quantities read from the text file. |
| **Int** | tempreorderPoint | Used to temporarily store each product’s reorder point that were read from the text file. |
| **String** | tempcategory | Used to temporarily store the product categories read from the text file. |
| **string** | tempunitOfMeasurement | Used to temporarily store each product’s unit of measurement that were read from the text file. |

|  |  |  |
| --- | --- | --- |
| **outOfStockList Variables** | | |
| Type | Name | Description |
| **string** | dummy | Used to store unnecessary data |
| **Priority\_queue** | container | Used to store all information retrieved from the text file |
| **int** | tempProductID | Used to temporarily store the product IDs read from the text file. |
| **String** | tempProductDescription | Used to temporarily store the product descriptions read from the text file. |
| **string** | tempsupplierName | Used to temporarily store the supplier’s read from the text file. |
| **float** | tempitemPrice | Used to temporarily store the prices read from the text file. |
| **int** | tempitemQuantity | Used to temporarily store the quantities read from the text file. |
| **Int** | tempreorderPoint | Used to temporarily store each product’s reorder point that were read from the text file. |
| **String** | tempcategory | Used to temporarily store the product categories read from the text file. |
| **string** | tempunitOfMeasurement | Used to temporarily store each product’s unit of measurement that were read from the text file. |

|  |  |  |
| --- | --- | --- |
| **checkIfProductAlreadyExists Variables** | | |
| Type | Name | Description |
| **Struct bstnode** | \*root | Allows access to the data stored in the BST. |
| **int** | key | Stores the key value inputted by the user and to be used to compare with product IDs stored in the BST |

|  |  |  |
| --- | --- | --- |
| **checkIfQuantityExists Variables** | | |
| Type | Name | Description |
| **Struct bstnode** | \*root | Allows access to the data stored in the BST. |
| **int** | Id | Stores the product ID inputted by the user and to be used to compare with the other IDs stored in the BST |
| **int** | qt | Stores the desired quantity inputted by the user and to be used to compare with the actual quantity of a chosen product listed in the BST |

|  |  |  |
| --- | --- | --- |
| **deleteProduct Variables** | | |
| Type | Name | Description |
| **int** | productId | Stores the product ID inputted by the user |
| **bool** | idChecker | Used to determine if user has re-input their value. |

|  |  |  |
| --- | --- | --- |
| **orderProduct Variables** | | |
| Type | Name | Description |
| **char** | option | Used to determine if the user wants to add another entry into the sales file |
| **int** | productId | Stores the product ID inputted by the user and to be used to compare with product IDs stored in the BST |
| **int** | productQuantity | Stores the desired quantity inputted by the user and to be used to compare with the actual quantity of a chosen product listed in the BST. It will then be sent and recorded into the sales file. |
| **bool** | verification | Used determine if user has re-input their values in case of invalid ID or quantity. |

|  |  |  |
| --- | --- | --- |
| **addSales Variables** | | |
| Type | Name | Description |
| **char** | tryAgain | Used to determine if the user wants to add another entry into the sales file |
| **int** | productId | Stores the product ID inputted by the user and to be used to compare with product IDs stored in the BST |
| **int** | productQuantity | Stores the desired quantity inputted by the user and to be used to compare with the actual quantity of a chosen product listed in the BST. It will then be sent and recorded into the sales file. |
| **bool** | verify | Used determine if user has re-input their values in case of invalid ID or quantity. |
| **int** | counter | Used for incrementation |

|  |  |  |
| --- | --- | --- |
| **recieveProduct Variables** | | |
| Type | Name | Description |
| **int** | orderNumber | Used to display a numbered list when outputting the entries from the order file. |
| **string** | infoLine | Stores each line from the text file and displays them |
| **int** | productId | Stores the product ID inputted by the user. |
| **int** | counter | Used for incrementation |
| **string** | transferer | Stores lines of text from one file and sends to new files |
| **int** | orderAmount | Stores the desired quantity entered by the user |

|  |  |  |
| --- | --- | --- |
| **treeClear Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | root | Allows access to the data stored in the BST. |

|  |  |  |
| --- | --- | --- |
| **clearSales Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | root | Allows access to the data stored in the BST. |

|  |  |  |
| --- | --- | --- |
| **showSales Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | root | Allows access to the data stored in the BST. |

|  |  |  |
| --- | --- | --- |
| **inorder Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | root | Allows access to the data stored in the BST. |

|  |  |  |
| --- | --- | --- |
| **insert Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | node | Allows the access and manipulation of the BST |
| **Bstnode** | \*newProduct | Contains the product ID of a new entry for the Master List |

|  |  |  |
| --- | --- | --- |
| **sortedArrayToBST Variables** | | |
| Type | Name | Description |
| **vector** | arr | A vector used to temporarily store the contents of a stack |
| **int** | start | The position of the first variable |
| **int** | end | The position of the last variable |
| **bstnode** | \*temp | Returns a sorted tree |

|  |  |  |
| --- | --- | --- |
| **minValueNode Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | node | Allows the access and manipulation of the BST |
| **Struct bstnode\*** | current | Hold the minimum value found |

|  |  |  |
| --- | --- | --- |
| **printBT Variables** | | |
| Type | Name | Description |
| **Const string&** | prefix | Used to print out the characters on the screen |
| **Const bstnode\*** | node | Allows the access and manipulation of the BST |
| **bool** | isLeft | Used to determine whether the number is left or right of the BST |
| **string** | container | Used to hold the characters used to create lines and spaces |

|  |  |  |
| --- | --- | --- |
| **newNode Variables** | | |
| Type | Name | Description |
| **bstnode\*** | \*newProduct | Pointer used to navigate through the BST |
| **Struct bstnode** | \*temp | Holds a copy of the BST |

|  |  |  |
| --- | --- | --- |
| **deleteNode Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | root | Allows access to the data stored in the BST. |
| **int** | productID | Holds the product ID inputted by the user |

|  |  |  |
| --- | --- | --- |
| **ProductIDFinder Variables** | | |
| Type | Name | Description |
| **Struct bstnode\*** | root | Allows access to the data stored in the BST. |
| **int** | productID | Holds the product ID inputted by the user |

|  |  |  |
| --- | --- | --- |
| **ROPchecker Variables** | | |
| Type | Name | Description |
| **int** | counter | Used for incrementing |
| **string** | dummy | Used to store unnecessary data |
| **int** | tempProductID | Used to temporarily store the product IDs read from the text file. |
| **String** | tempProductDescription | Used to temporarily store the product descriptions read from the text file. |
| **string** | tempsupplierName | Used to temporarily store the supplier’s read from the text file. |
| **float** | tempitemPrice | Used to temporarily store the prices read from the text file. |
| **int** | tempitemQuantity | Used to temporarily store the quantities read from the text file. |
| **Int** | tempreorderPoint | Used to temporarily store each product’s reorder point that were read from the text file. |
| **String** | tempcategory | Used to temporarily store the product categories read from the text file. |
| **string** | tempunitOfMeasurement | Used to temporarily store each product’s unit of measurement that were read from the text file. |

|  |  |  |
| --- | --- | --- |
| **viewSales Variables** | | |
| Type | Name | Description |
| **string** | dummy | Used to temporarily hold string data from text file |
| **int** | productID | Holds the product ID retrieved from the text file |
| **Int** | totalSales | Holds the item’s total sale data retrieved from the sales text file |