

← →

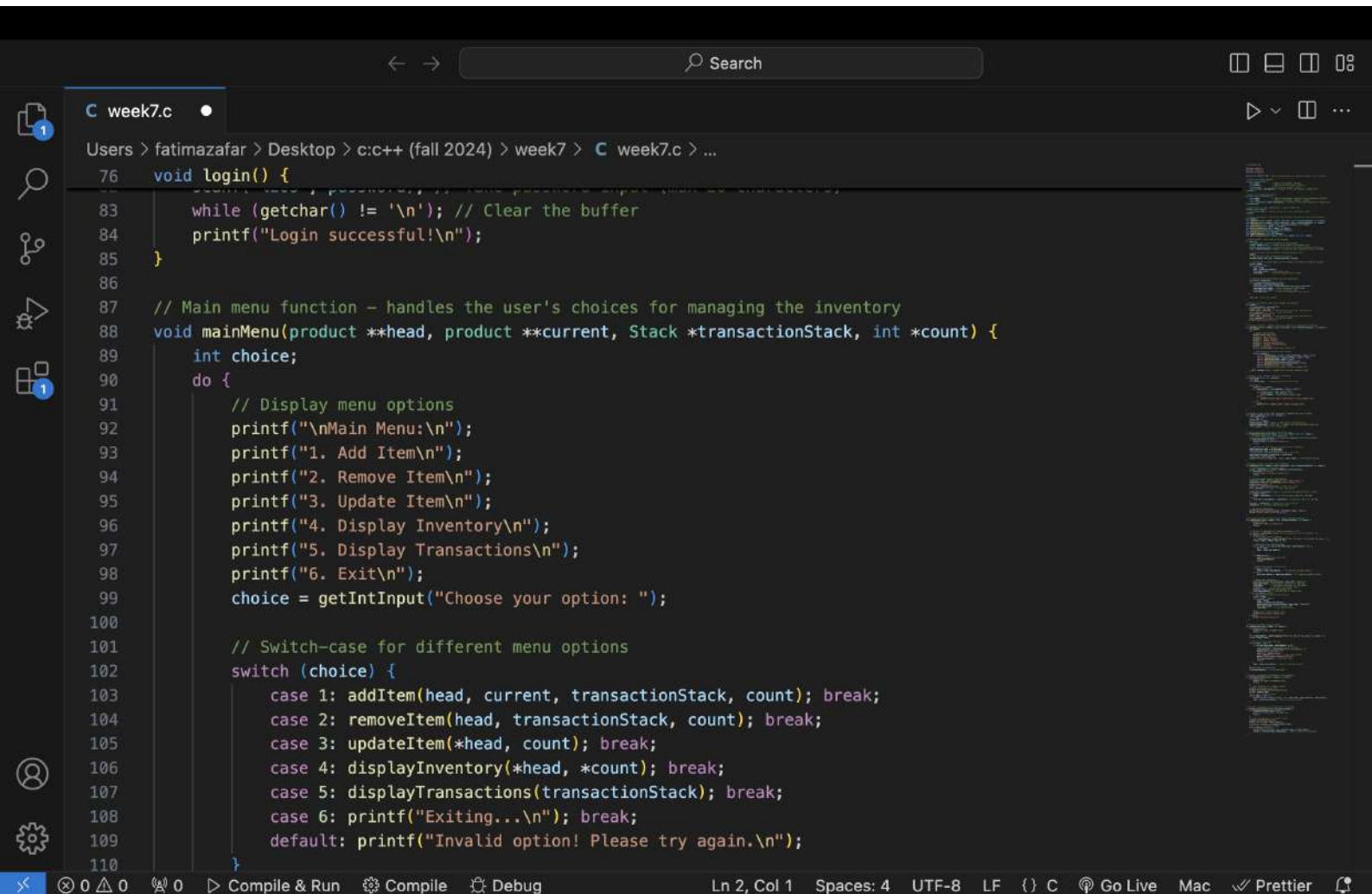
Search

week7.c

Users > fatimazafar > Desktop > c:c++ (fall 2024) > week7 > C week7.c > ...

```
42  int main() {
56      temp = head;
57      head = head->new_address;
58      free(temp->name); // Free product name
59      free(temp);      // Free the product struct itself
60  }
61
62  // Freeing the allocated memory for the transactions
63  transaction *tempTrans;
64  while (transactionStack.top != NULL) {
65      tempTrans = transactionStack.top;
66      transactionStack.top = transactionStack.top->next_transaction;
67      free(tempTrans->name); // Free transaction product name
68      free(tempTrans->type); // Free transaction type
69      free(tempTrans);      // Free the transaction struct itself
70  }
71
72  return 0; // Exit the program
73  }
74
75  // Function to simulate login with username and password
76  void login() {
77      char username[21], password[21];
78      printf("Enter username: ");
79      scanf("%20s", username); // Take username input (max 20 characters)
80      while (getchar() != '\n'); // Clear the buffer
81      printf("Enter password: ");
82      scanf("%20s", password); // Take password input (max 20 characters)
83      while (getchar() != '\n'); // Clear the buffer
```

Ln 2, Col 1 Spaces: 4 UTF-8 LF {} C Go Live Mac Prettier



Search

□ □ □ □

1

▶ ▾ □ ...

Ln 1, Col 2 Spaces: 4 UTF-8 LF {} C Go Live Mac ✓ Prettier 🔔

1

Search

Users > fatimazafar > Desktop > c:c++ (fall 2024) > week7 > C week7.c > ...

134

// Function to get string input dynamically (handles any size of input)

135

char *getStringInput(const char *prompt) {

136

char *input = NULL;

137

size_t len = 0;

138

printf("%s", prompt);

139

getline(&input, &len, stdin); // Read input line dynamically

140

input[strcspn(input, "\n")] = 0; // Remove the trailing newline character

141

return input; // Return the string input

142

}

143

144

// Push transaction to the stack (add to the top)

145

void pushTransaction(Stack *stack, const char *name, const char *type) {

146

// Allocate memory for a new transaction

147

transaction *newTransaction = (transaction *)malloc(sizeof(transaction));

148

if (newTransaction == NULL) {

149

printf("Memory allocation failed!\n");

150

return;

151

}

152

// Duplicate the name and type strings for the transaction

153

newTransaction->name = strdup(name);

154

newTransaction->type = strdup(type);

155

// Insert the new transaction at the top of the stack

156

newTransaction->next_transaction = stack->top;

157

stack->top = newTransaction;

158

printf("Transaction added: %s - %s\n", name, type); // Confirmation message

159

}

160

161

// Function to add a new item to the inventory

162

void addItem(product **head, product **current, Stack *transactionStack, int *count) {

Ln 1, Col 2

Spaces: 4

UTF-8

LF

{ } C

Go Live

Mac

Prettier

← →

Search

☐ ☐ ☐ ☐

1

week7.c

Users > fatimazafar > Desktop > c:c++ (fall 2024) > week7 > C week7.c > ...

162 void addItem(product **head, product **current, Stack *transactionStack, int *count) {

163 // Allocate memory for the new product

164 product *newProduct = (product *)malloc(sizeof(product));

165 if (newProduct == NULL) {

166 printf("Memory allocation failed!\n");

167 return;

168 }

169 // Get the product details from the user

170 newProduct->name = getStringInput("Enter product name: ");

171 newProduct->quantity = getIntInput("Enter quantity: ");

172 printf("Enter price: ");

173 scanf("%f", &newProduct->price); // Read price input

174 while (getchar() != '\n'); // Clear input buffer

175

176 newProduct->new_address = NULL; // Initialize new_address pointer to NULL

177 if (*head == NULL) {

178 *head = newProduct; // If the list is empty, make this the head

179 } else {

180 (*current)->new_address = newProduct; // Otherwise, add it to the end

181 }

182 *current = newProduct; // Update the current pointer

183 (*count)++; // Increment the product count

184

185 // Record the transaction

186 pushTransaction(transactionStack, newProduct->name, "Added");

187 printf("Product added successfully!\n");

188 }

189

190 // Function to remove an item (or all items) from the inventory

< 0 0 0 0 0

Compile & Run

Compile

Debug

Ln 1, Col 2

Spaces: 4

UTF-8

LF

{ } C

Go Live

Mac

Prettier

←

→

Search

📄

📄

📄

📄

C week7.c

Users > fatimazafar > Desktop > c:c++ (fall 2024) > week7 > C week7.c > ...

190 // Function to remove an item (or all items) from the inventory

191 void removeItem(product **head, Stack *transactionStack, int *count) {

192 if (*count == 0) {

193 printf("No items to remove.\n");

194 return;

195 }

196

197 // Ask user if they want to remove one product or all

198 int choice = getIntInput("Remove (1) one product or (2) all products: ");

199 if (choice == 1) {

200 // Removing a specific product

201 char *nameToRemove = getStringInput("Enter the name of the product to remove: ");

202 product *temp = *head, *prev = NULL;

203

204 // Search for the product by name

205 while (temp != NULL && strcmp(temp->name, nameToRemove) != 0) {

206 prev = temp;

207 temp = temp->new_address;

208 }

209

210 if (temp == NULL) {

211 printf("Product not found.\n");

212 free(nameToRemove);

213 return;

214 }

215

216 // Remove the product from the list

217 if (prev == NULL) {

218 *head = temp->new_address; // If removing the head product

🔍

🔗

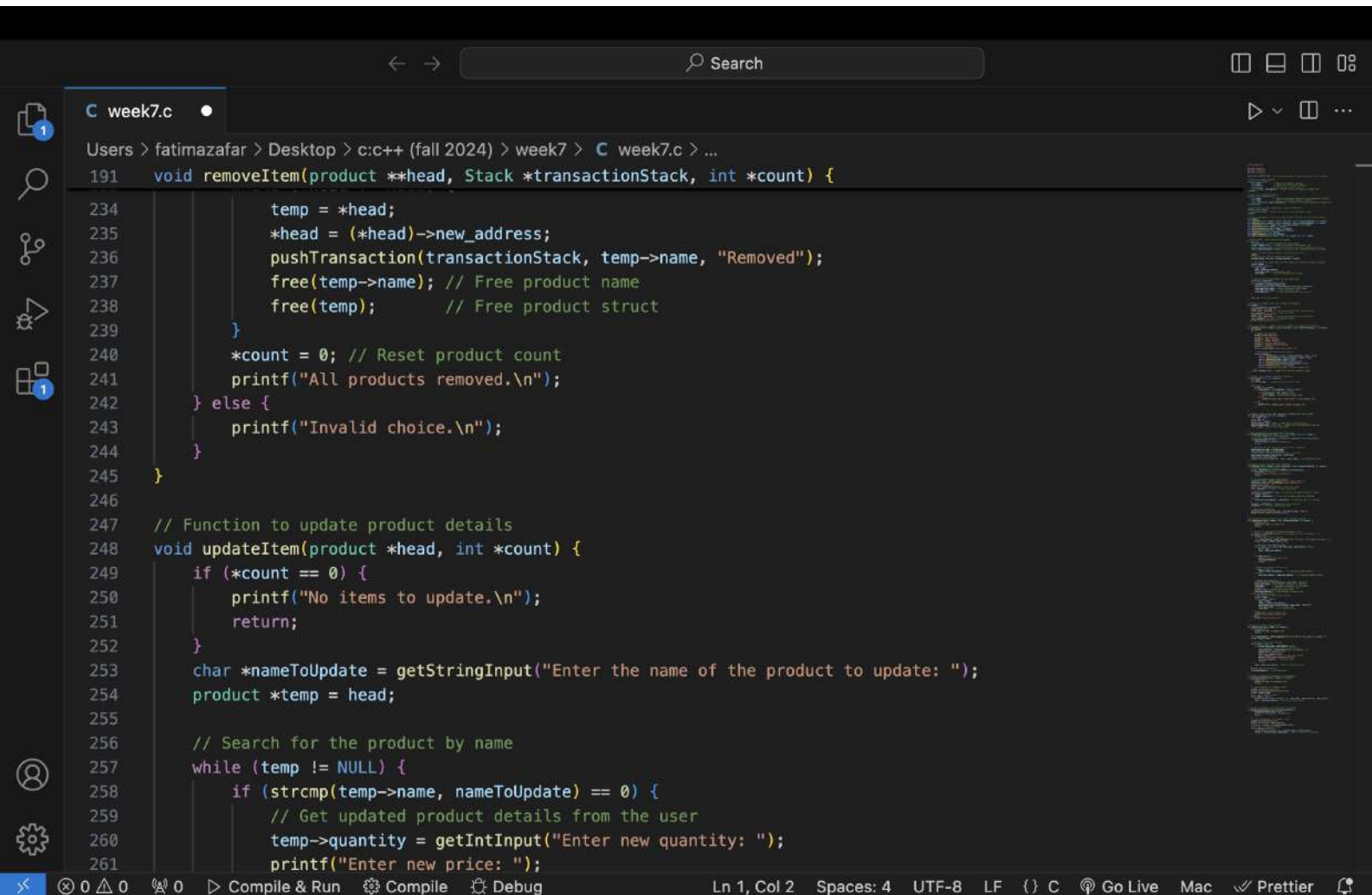
🔧

📁

👤

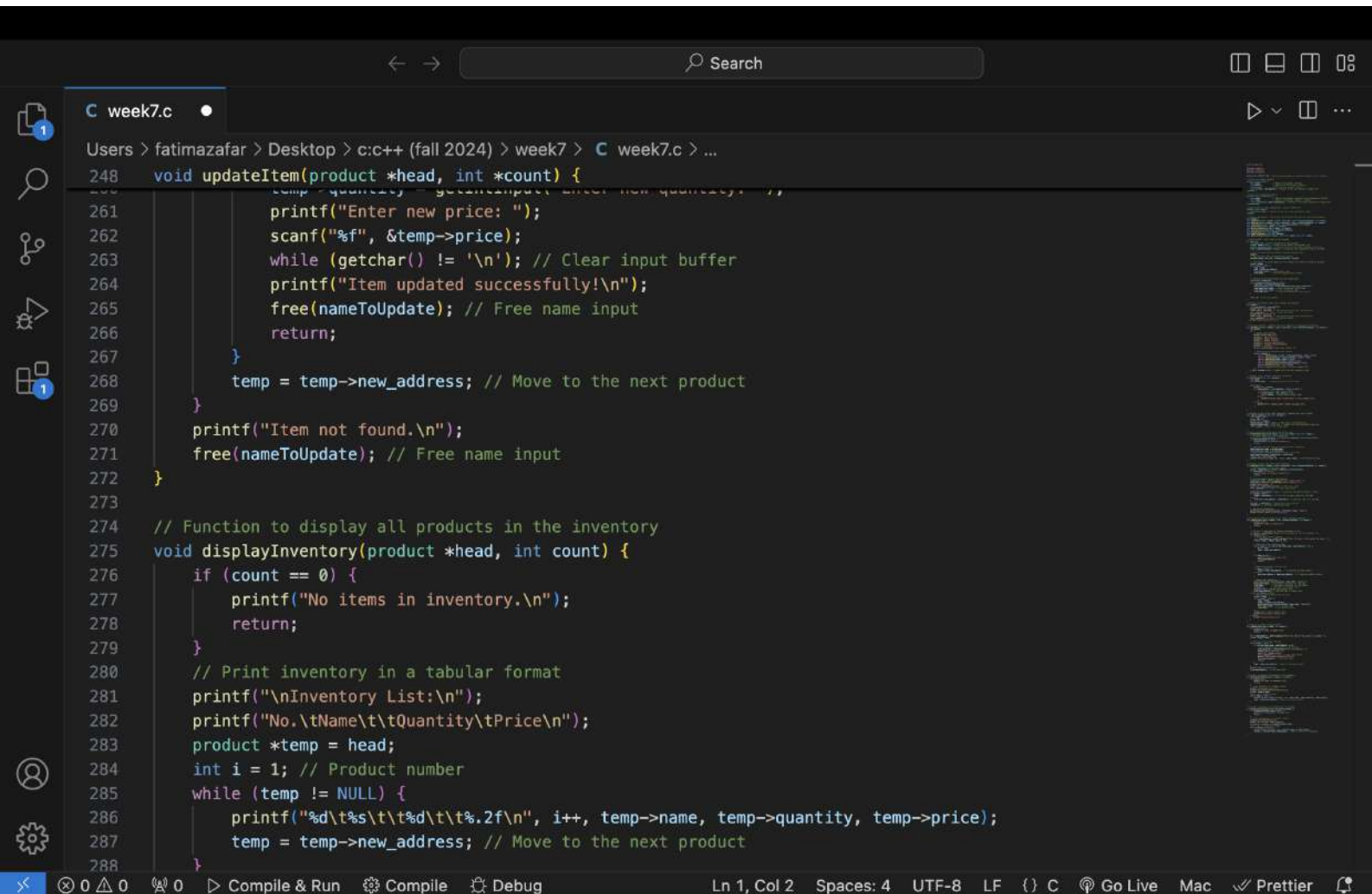
⚙️

Ln 1, Col 2 Spaces: 4 UTF-8 LF {} C Go Live Mac Prettier



```
191 void removeItem(product **head, Stack *transactionStack, int *count) {
234     temp = *head;
235     *head = (*head)->new_address;
236     pushTransaction(transactionStack, temp->name, "Removed");
237     free(temp->name); // Free product name
238     free(temp);      // Free product struct
239 }
240 *count = 0; // Reset product count
241 printf("All products removed.\n");
242 } else {
243     printf("Invalid choice.\n");
244 }
245 }
246
247 // Function to update product details
248 void updateItem(product *head, int *count) {
249     if (*count == 0) {
250         printf("No items to update.\n");
251         return;
252     }
253     char *nameToUpdate = getStringInput("Enter the name of the product to update: ");
254     product *temp = head;
255
256     // Search for the product by name
257     while (temp != NULL) {
258         if (strcmp(temp->name, nameToUpdate) == 0) {
259             // Get updated product details from the user
260             temp->quantity = getIntInput("Enter new quantity: ");
261             printf("Enter new price: ");
```

Ln 1, Col 2 Spaces: 4 UTF-8 LF {} C Go Live Mac Prettier



```
248 void updateItem(product *head, int *count) {
249     temp->quantity = getintInput("Enter new quantity: ");
261     printf("Enter new price: ");
262     scanf("%f", &temp->price);
263     while (getchar() != '\n'); // Clear input buffer
264     printf("Item updated successfully!\n");
265     free(nameToUpdate); // Free name input
266     return;
267 }
268 temp = temp->new_address; // Move to the next product
269 }
270 printf("Item not found.\n");
271 free(nameToUpdate); // Free name input
272 }
273
274 // Function to display all products in the inventory
275 void displayInventory(product *head, int count) {
276     if (count == 0) {
277         printf("No items in inventory.\n");
278         return;
279     }
280     // Print inventory in a tabular format
281     printf("\nInventory List:\n");
282     printf("No.\tName\t\tQuantity\tPrice\n");
283     product *temp = head;
284     int i = 1; // Product number
285     while (temp != NULL) {
286         printf("%d\t%s\t\t%d\t\t%.2f\n", i++, temp->name, temp->quantity, temp->price);
287         temp = temp->new_address; // Move to the next product
288     }
```

Ln 1, Col 2 Spaces: 4 UTF-8 LF {} C Go Live Mac Prettier

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

week7.c

Users > fatimazafar > Desktop > c:c++ (fall 2024) > week7 > C week7.c > ...

275

void displayInventory(product *head, int count) {

287

temp = temp->new_address; // Move to the next product

288

}

289

}

290

291

// Function to display all transactions in the stack

292

void displayTransactions(Stack *transactionStack) {

293

if (transactionStack->top == NULL) {

294

printf("No transactions recorded.\n");

295

return;

296

}

297

// Print transactions in a tabular format

298

printf("\nTransactions:\n");

299

printf("No.\tProduct Name\tType\n");

300

transaction *current = transactionStack->top;

301

int i = 1; // Transaction number

302

while (current != NULL) {

303

printf("%d\t%s\t\t\t\n", i++, current->name, current->type);

304

current = current->next_transaction; // Move to the next transaction

305

}

306

}

307

Ln 1, Col 2

Spaces: 4

UTF-8

LF

{ } C

Go Live

Mac

Prettier

Enter username: admin
Enter password: admin1234
Login successful!

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 1

Enter product name: Bags
Enter quantity: 50
Enter price: 1.25
Transaction added: Bags - Added
Product added successfully!

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 1

Enter product name: Shoes
Enter quantity: 30
Enter price: 0.75
Transaction added: Shoes - Added
Product added successfully!

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 4

Inventory List:

No.	Name	Quantity	Price
1	Bags	50	1.25
2	Shoes	30	0.75

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 3

Enter the name of the product to update: Bags

Enter new quantity: 60

Enter new price: 1.35

Item updated successfully!

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 4

Inventory List:

No.	Name	Quantity	Price
1	Bags	60	1.35
2	Shoes	30	0.75

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 5

Transactions:

No.	Product Name	Type
1	Shoes	Added
2	Bags	Added
3	Bags	Updated

Main Menu:

1. Add Item
2. Remove Item
3. Update Item
4. Display Inventory
5. Display Transactions
6. Exit

Choose your option: 6

Exiting...

Advantages of using dynamic memory, linked lists, and stacks in the IMS:

Since linked lists only use memory for active items, they are more memory-efficient for dynamic data like an inventory system. This contrasts with arrays that reserve a fixed block of memory regardless of how full the inventory is. Also, in linked lists when items are added or removed, the memory usage can grow or decrease accordingly.

With linked lists, you can easily add or delete products, making these operations quicker and easier. The previous version's array structure made it more challenging to implement new features without significant rewrites.

The stack records each transaction, whether it is an "add," "remove," or "update." Since the most recent transaction is at the top, you can easily keep track of changes. The previous version of the program lacked managing transactions, making it harder to review them.

The use of structs for products and transactions in the new version provides a clear structure for manipulating data. This improves code readability compared to the earlier version, where data was managed using arrays without clear definitions for each product or transaction.

GenAI Integration:

Pointers: GenAI or (ChatGPT in this case) suggested using NULL checks before dereferencing pointers so as to avoid segmentation errors as well as using const pointers for the data that is not meant to be modified.

Dynamic memory management: Smart memory management techniques were suggested in order to deal with larger lists dynamically, however not relevant for linked lists.

Linked lists Implementation: the suggestion was to add a hash map for faster lookups by product name for specific searches for a more advanced program in future.

Fair Contribution Sheet

- Group Number/Name..... Group 1

Enter average fair contribution scores here.


- Member name Mark

1... Fatima Zafar 10

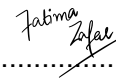
2... Marvo Amini 10

3... Alireza Eftekhari 10

- Name and signature of each member:

Signature ...  ... Date 14/10/24

Signature ... *Marvo Amini* ... Date 14.10.24

Signature ...  ... Date 14/10/24