**Code For Shopping Cart:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Product {

int categoryNumber;

char name[50];

int productId;

float price;

int quantity;

int decrementedCount;

struct Product \*next;

};

struct Stack {

struct Product \*product;

struct Stack \*next;

};

struct Queue {

struct Product \*product;

struct Queue \*next;

};

struct BSTNode {

int productId;

struct Product \*product;

struct BSTNode \*left, \*right;

};

struct GraphNode {

int categoryNumber;

struct Product \*product;

struct GraphNode \*next;

};

int getNextProductId(int categoryNumber) {

static int electronicsId = 1000;

static int groceriesId = 2000;

static int booksId = 3000;

static int beautyProductsId = 4000;

static int clothingId = 5000;

switch (categoryNumber) {

case 101: return electronicsId++;

case 102: return groceriesId++;

case 103: return booksId++;

case 104: return beautyProductsId++;

case 105: return clothingId++;

default: return -1;

}

}

void addProduct(struct Product \*\*head, int categoryNumber, char \*name, float price, int quantity) {

struct Product \*newProduct = (struct Product \*)malloc(sizeof(struct Product));

newProduct->categoryNumber = categoryNumber;

strcpy(newProduct->name, name);

newProduct->productId = getNextProductId(categoryNumber);

newProduct->price = price;

newProduct->quantity = quantity;

newProduct->decrementedCount = 0;

newProduct->next = \*head;

\*head = newProduct;

printf("Product added: %s, ID: %d, Price: %.2f, Quantity: %d\n", newProduct->name, newProduct->productId, newProduct->price, newProduct->quantity);

}

void displayProductsByCategory(struct Product \*head, int categoryNumber) {

struct Product \*current = head;

int found = 0;

printf("Displaying products for Category %d:\n", categoryNumber);

while (current) {

if (current->categoryNumber == categoryNumber) {

printf("Product ID: %d, Name: %s, Price: %.2f, Quantity: %d\n",

current->productId, current->name, current->price, current->quantity);

found = 1;

}

current = current->next;

}

if (!found) {

printf("No products found in this category.\n");

}

}

void updateProduct(struct Product \*head, int productId, int quantity) {

struct Product \*current = head;

while (current) {

if (current->productId == productId) {

current->quantity = quantity;

printf("Product ID %d updated to quantity: %d\n", productId, quantity);

return;

}

current = current->next;

}

printf("Product with ID %d not found.\n", productId);

}

struct Product \*searchProductById(struct Product \*head, int productId, int categoryNumber) {

struct Product \*current = head;

while (current) {

if (current->productId == productId && current->categoryNumber == categoryNumber) {

return current;

}

current = current->next;

}

return NULL;

}

void push(struct Stack \*\*stackTop, struct Product \*product) {

struct Stack \*newNode = (struct Stack \*)malloc(sizeof(struct Stack));

newNode->product = product;

newNode->next = \*stackTop;

\*stackTop = newNode;

}

struct Product \*pop(struct Stack \*\*stackTop) {

if (\*stackTop == NULL) {

printf("Stack is empty.\n");

return NULL;

}

struct Stack \*temp = \*stackTop;

struct Product \*product = temp->product;

\*stackTop = temp->next;

free(temp);

return product;

}

void displayDeletedProducts(struct Stack \*stackTop) {

if (!stackTop) {

printf("No deleted products to display.\n");

return;

}

printf("Deleted Products (Stack Order):\n");

while (stackTop) {

printf("Product ID: %d, Name: %s, Price: %.2f, Quantity: %d, Category: %d\n",

stackTop->product->productId, stackTop->product->name, stackTop->product->price, stackTop->product->quantity, stackTop->product->categoryNumber);

stackTop = stackTop->next;

}

}

void findLowestStockInCategory(struct Product \*head, int categoryNumber) {

struct Product \*current = head;

struct Product \*lowestStockProduct = NULL;

int lowestStock = -1;

while (current) {

if (current->categoryNumber == categoryNumber) {

if (lowestStockProduct == NULL || current->quantity < lowestStock) {

lowestStockProduct = current;

lowestStock = current->quantity;

}

}

current = current->next;

}

if (lowestStockProduct) {

printf("Product with lowest stock in Category %d: ID: %d, Name: %s, Price: %.2f, Quantity: %d\n",

categoryNumber, lowestStockProduct->productId, lowestStockProduct->name,

lowestStockProduct->price, lowestStockProduct->quantity);

} else {

printf("No products found in this category.\n");

}

}

void categoryActions(struct Product \*\*head, struct Stack \*\*stackTop, struct Queue \*\*queueFront, struct Queue \*\*queueRear, struct GraphNode \*\*graph, struct BSTNode \*\*bstRoot, int categoryNumber) {

int choice, productId, quantity;

char name[50];

float price;

printf("\nCategory %d Menu:\n", categoryNumber);

switch (categoryNumber) {

case 101:

printf("1. Add Electronics\n2. Display Electronics\n3. Delete Electronics\n4. Search Electronics by ID\n5. Display Deleted Electronics\n6. Find Lowest Stock in Electronics\n7. Update Electronics Quantity\n");

break;

case 102:

printf("1. Add Groceries\n2. Display Groceries\n3. Delete Groceries\n4. Search Groceries by ID\n5. Display Deleted Groceries\n6. Find Lowest Stock in Groceries\n7. Update Groceries Quantity\n");

break;

case 103:

printf("1. Add Books\n2. Display Books\n3. Delete Books\n4. Search Books by ID\n5. Display Deleted Books\n6. Find Lowest Stock in Books\n7. Update Books Quantity\n");

break;

case 104:

printf("1. Add Beauty Products\n2. Display Beauty Products\n3. Delete Beauty Products\n4. Search Beauty Products by ID\n5. Display Deleted Beauty Products\n6. Find Lowest Stock in Beauty Products\n7. Update Beauty Products Quantity\n");

break;

case 105:

printf("1. Add Clothing\n2. Display Clothing\n3. Delete Clothing\n4. Search Clothing by ID\n5. Display Deleted Clothing\n6. Find Lowest Stock in Clothing\n7. Update Clothing Quantity\n");

break;

default:

printf("Invalid category number.\n");

return;

}

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter Product Name: ");

scanf("%s", name);

printf("Enter Product Price: ");

scanf("%f", &price);

printf("Enter Product Quantity: ");

scanf("%d", &quantity);

addProduct(head, categoryNumber, name, price, quantity);

break;

case 2:

displayProductsByCategory(\*head, categoryNumber);

break;

case 3:

printf("Enter Product ID to Delete: ");

scanf("%d", &productId);

struct Product \*productToDelete = searchProductById(\*head, productId, categoryNumber);

if (productToDelete) {

printf("Product deleted: ID: %d, Name: %s\n", productToDelete->productId, productToDelete->name);

struct Product \*current = \*head, \*prev = NULL;

while (current && current->productId != productId) {

prev = current;

current = current->next;

}

if (prev) {

prev->next = current->next;

} else {

\*head = current->next;

}

push(stackTop, productToDelete);

free(current);

} else {

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

}

break;

case 4:

printf("Enter Product ID to Search: ");

scanf("%d", &productId);

struct Product \*foundProduct = searchProductById(\*head, productId, categoryNumber);

if (foundProduct) {

printf("Product found: %s, ID: %d, Price: %.2f, Quantity: %d\n", foundProduct->name, foundProduct->productId, foundProduct->price, foundProduct->quantity);

} else {

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

}

break;

case 5:

displayDeletedProducts(\*stackTop);

break;

case 6:

findLowestStockInCategory(\*head, categoryNumber);

break;

case 7:

printf("Enter Product ID to Update Quantity: ");

scanf("%d", &productId);

printf("Enter New Quantity: ");

scanf("%d", &quantity);

updateProduct(\*head, productId, quantity);

break;

default:

printf("Invalid choice.\n");

break;

}

}

int main() {

struct Product \*productList = NULL;

struct Stack \*deletedProductsStack = NULL;

int categoryNumber;

while (1) {

printf("\nSelect a category:\n101. Electronics\n102. Groceries\n103. Books\n104. Beauty Products\n105. Clothing\n0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &categoryNumber);

if (categoryNumber == 0) {

break;

}

categoryActions(&productList, &deletedProductsStack, NULL, NULL, NULL, NULL, categoryNumber);

}

return 0;

}