**תרגיל בית 4**

שאלה 1:

**קלט:**

**Restaurant.h**

**#ifndef \_Restaurant**

**#define \_Restaurant**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

#define TABLE\_NUM 50

#define MAX\_NAME 50

typedef struct Product { // Product is a Meal in the restaurant - each Meal is a node

char\* ProductName;

double Price;

int Quantity; // Quantity available in kitchen inventory

struct Product\* next;

}Product, \* pProduct;

typedef struct Inventory { // A manager struct for list of Products, which represent the Inventory available

pProduct head;

pProduct tail;

}Inventory, \* PInventory;

typedef struct Table { // A manager struct for list of Products arrived at each Table, every meal is a node.

int TableNumber;

int Delete; // for DeleteItem use

int flag;

pProduct head;

pProduct tail;

}Table, \* PTable;

void CreateProducts(FILE\* Menu, PInventory Inventory); //Calling functions

void AddItems(char\* ProductName, int Quantity, PInventory Inventory);

void OrderItem(int TableNumber, int Quantity, char\* ProductName, Table\* Table\_arr, PInventory Inventory);

void RemoveItem(int TableNumber, Table\* Table\_arr);

void RemoveTable(int TableNumber, Table\* Table\_arr);

void Error\_msg(char\* msg);

**#endif**

**Restaurant.c**

**#include "Restaurant.h"**

void CreateProducts(FILE\* Menu, PInventory Inventory) //This function create Nodes with info about meals.

{

char temp\_name[MAX\_NAME];

pProduct temp\_product;

while (fscanf(Menu, "%s", temp\_name) != EOF) { //Scan until end of file

temp\_product = (pProduct)malloc(sizeof(Product)); // Allocation of product

if (temp\_product == NULL) {

while (Inventory->head != NULL) { //If failed

temp\_product = Inventory->head;

Inventory->head = Inventory->head->next;

free(temp\_product->ProductName);

free(temp\_product);

}

Error\_msg("\nMemory allocation failed");

}

temp\_product->ProductName = (char\*)malloc(strlen(temp\_name) + 1); //Allocation for name

if (temp\_product->ProductName == NULL) {

while (Inventory->head != NULL) {

temp\_product = Inventory->head;

Inventory->head = Inventory->head->next;

free(temp\_product->ProductName);

free(temp\_product);

}

Error\_msg("\nMemory allocation failed");

}

strcpy(temp\_product->ProductName, temp\_name); // Enterting the name to node

fscanf(Menu, "%d%lf", &(temp\_product->Quantity), &(temp\_product->Price)); //More info entered

temp\_product->next = NULL;

if (Inventory->head == NULL) //First item is head

Inventory->head = temp\_product;

else

Inventory->tail->next = temp\_product;

Inventory->tail = temp\_product; // inventory tail will now point to new last item

}

printf("The kitchen was created\n\n");

return;

}

void AddItems(char\* ProductName, int Quantity, PInventory Inventory) // This function adds to a product's quantity in inventory

{

pProduct temp = Inventory->head; // Set temp to be first item on the list //

if (Quantity <= 0) {

printf("Error - cannot add un positive quantity !\n\n");

return;

}

while (temp != NULL) { // Check all items

if (strcmp(temp->ProductName, ProductName) == 0) { // if the current Node on the list has the same name given as an argument //

temp->Quantity += Quantity;

printf("%d %s were added to the kitchen\n\n", Quantity, ProductName);

return; //return void

}

temp = temp->next; //Next node on list

}

printf("We don't have %s in our resturant, sorry!\n\n", ProductName);

return;

}

void OrderItem(int TableNumber, int Quantity, char\* ProductName, Table\* Table\_arr, PInventory Inventory) //This function basiclly order's the meal.

{

int i, index = TableNumber - 1;

pProduct NewOrder, temp = Inventory->head, Deletetemp;

if (TableNumber > 50 || TableNumber < 1) {

printf("Error - Invalid table number \n\n");

return;

}

while (temp != NULL) { // Check all items

if (strcmp(temp->ProductName, ProductName) == 0) { //Checking if required item is on the list

if (temp->Quantity >= Quantity) { // if Quantity is available go on

NewOrder = (pProduct)malloc(sizeof(Product)); //Memory Allocation for a new product node for desired table

if (NewOrder == NULL) { //Allocation failed

while (Inventory->head != NULL) { // Free all the memory allocations

Deletetemp = Inventory->head; // Free inventory

Inventory->head = Inventory->head->next;

free(Deletetemp->ProductName);

free(Deletetemp);

}

for (i = 0; i < TABLE\_NUM; i++) { //Free all items ordered for each table

while (Table\_arr[i].head != NULL) {

Deletetemp = Table\_arr[i].head;

Table\_arr[i].head = (Table\_arr[i].head)->next;

free(Deletetemp->ProductName);

free(Deletetemp);

}

}

Error\_msg("Memory allocation failed");

}

NewOrder->ProductName = (char\*)malloc(strlen(ProductName) + 1);

if (NewOrder->ProductName == NULL)

Error\_msg("Memory allocation failed");

strcpy(NewOrder->ProductName, ProductName); //Defining our new item

NewOrder->Price = temp->Price;

NewOrder->Quantity = Quantity;

NewOrder->next = NULL;

if (Table\_arr[index].head == NULL) // if its the 1st item set tail to be it

Table\_arr[index].tail = NewOrder;

NewOrder->next = Table\_arr[index].head; // inserting to head of list

Table\_arr[index].head = NewOrder;

temp->Quantity -= Quantity;// Updating the item's inventory quantity

printf("%d %s were added to table number %d\n\n", Quantity, NewOrder->ProductName, TableNumber);

return;

}

printf("There are only %d %s available\n\n", temp->Quantity, temp->ProductName); // In case the item exist but there's not enough of the item.

return;

}

temp = temp->next; //Keep going in the list

}

printf("We don't have %s, sorry!\n\n", ProductName);

return;

}

void RemoveItem(int TableNumber, Table\* Table\_arr) //This function remove a meal from the table.

{

int index = TableNumber - 1;

pProduct ToRemove;

if (TableNumber > 50 || TableNumber < 1) { // Table number error

printf("Error - Invalid table number \n\n");

return;

}

else if (Table\_arr[index].head == NULL) { // Empty list

printf("Table number %d hasnt order yet \n\n", TableNumber);

return;

}

else if (Table\_arr[index].Delete == 1) { //In case table already deleted.

printf("This table already cancelled an order \n\n");

return;

}

ToRemove = Table\_arr[index].head; //Setting Node we want to remove

Table\_arr[index].head = (Table\_arr[index].head)->next; // Setting the 2nd item to be the head now

printf("%d %s was removed from table number %d\n\n", ToRemove->Quantity, ToRemove->ProductName, TableNumber);

free(ToRemove->ProductName); // Removing allocations of the node

free(ToRemove);

Table\_arr[index].Delete = 1; // Setting the delete to be 1, which means this table already deleted an item

return;

}

void RemoveTable(int TableNumber, Table\* Table\_arr) //This function basiclly close a table and check it's cost.

{

int index = TableNumber - 1;

pProduct ToDel, temp = Table\_arr[index].head;

double sum = 0;

if (temp == NULL) { //Table hasn't ordered anything yet.

printf("The table number %d is not ordered yet\n\n", TableNumber);

return;

}

while (temp != NULL) { // Check all items

printf("%d %s, ", temp->Quantity, temp->ProductName);

ToDel = temp; // Set this item to be deleted

sum += (temp->Price) \* (temp->Quantity); // Calculate check price

temp = temp->next; // Keep going

free(ToDel->ProductName); // Free the node

free(ToDel);

}

printf("%g nis+%g nis for tips,please!", sum, 0.15 \* sum); //Price

Table\_arr[index].tail = Table\_arr[index].head = NULL; // Reset table

Table\_arr[index].Delete = 0;

return;

}

void Error\_msg(char\* msg)

{

printf("%s", msg);

exit(1);

return;

}

**Main.c**

**#include "Restaurant.h"**

int main(){

int i, option, quantity, table\_num; //Defining

char name[MAX\_NAME];

Table Table\_arr[TABLE\_NUM], Temptable;

FILE\* Instructions, \* Manot;

Inventory main\_Inventory;

pProduct temp;

Temptable.head = Temptable.tail = NULL;

Temptable.Delete = 0;

for (i = 0; i < TABLE\_NUM; i++) {

Temptable.TableNumber = i + 1;

Table\_arr[i] = Temptable; //Save the adresses of the values

}

if ((Instructions = fopen("Instructions.txt", "rt")) == NULL) //Checking the files

Error\_msg("failed to open manot.txt file.");

if ((Manot = fopen("Manot.txt", "rt")) == NULL)

Error\_msg("failed to open manot.txt file.");

main\_Inventory.head = main\_Inventory.tail = NULL;

while (fscanf(Instructions, "%d", &option) != EOF) {

switch (option) {

case 1:

CreateProducts(Manot, &main\_Inventory); //Calling function

fclose(Manot);

break;

case 2:

fscanf(Instructions, "%s%d", name, &quantity);

AddItems(name, quantity, &main\_Inventory);

break;

case 3:

fscanf(Instructions, "%d%s%d", &table\_num, name, &quantity);

OrderItem(table\_num, quantity, name, Table\_arr, &main\_Inventory);

break;

case 4:

fscanf(Instructions, "%d", &table\_num);

RemoveItem(table\_num, &Table\_arr);

break;

case 5:

fscanf(Instructions, "%d", &table\_num);

RemoveTable(table\_num, &Table\_arr);

break;

}

}

fclose(Instructions);

while (main\_Inventory.head != NULL) { // free inventory list

temp = main\_Inventory.head;

main\_Inventory.head = (main\_Inventory.head)->next;

free(temp->ProductName);

free(temp);

}

for (i = 0; i < TABLE\_NUM; i++) { // free all items ordered on each table

while (Table\_arr[i].head != NULL) {

temp = Table\_arr[i].head;

Table\_arr[i].head = (Table\_arr[i].head)->next;

free(temp->ProductName);

free(temp);

}

}

return 0;

}

**פלט:**

Text

Description automatically generated