#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_UNIVERSITIES 100

// Structure to handle University details

struct University {

int ID;

char univ\_code[10];

char univ\_name[10];

char univ\_address[10];

char univ\_email[15];

char univ\_website[20];

};

//Function Declarations

void VirtualThinkers\_university\_create(struct University u[], int \*count);

void VirtualThinkers\_university\_update(struct University u[], int count);

void VirtualThinkers\_university\_retrieve(struct University u[], int \*count);

void VirtualThinkers\_university\_delete(struct University u[], int \*count);

void VirtualThinkers\_university\_MergeSort(struct University u[], int left, int right, int sort\_by);

void merge(struct University arr[], int left, int right, int sort\_by);

void VirtualThinkers\_university\_storing(struct University u[], int \*count, int mode, int univ\_code);

void VirtualThinkers\_university\_displayMenu();

void VirtualThinkers\_university\_LinearSearch(struct University u[], int count);

// Function to handle all file operations

void VirtualThinkers\_university\_storing(struct University u[], int \*count, int mode, int ID) {

FILE \*file = NULL;

// Writing new university details into the file (create or update)

if (mode == 1) {

file = fopen("university\_setting.txt", "w");

if (file == NULL) {

printf("File cannot be opened for writing.\n");

return;

}

for (int i = 0; i < \*count; i++) {

fprintf(file, "%d\n%s\n%s\n%s\n%s\n%s\n", u[i].ID,u[i].univ\_code, u[i].univ\_name,

u[i].univ\_address, u[i].univ\_email, u[i].univ\_website);

}

fclose(file);

printf("University data is saved successfully into the file!\n");

}

// Deleting selected university from the file

else if (mode == 3) {

file = fopen("university\_setting.txt", "r");

if (file == NULL) {

printf("The file cannot be opened for reading.\n");

return;

}

struct University temp\_u[MAX\_UNIVERSITIES];

int temp\_count = 0;

int found = 0;

// Reading all universities into a temporary array

while (fscanf(file, "%d\n", &temp\_u[temp\_count].ID) == 1) {

fscanf(file, "%[^\n]\n", temp\_u[temp\_count].univ\_code);

fscanf(file, "%[^\n]\n", temp\_u[temp\_count].univ\_name);

fscanf(file, "%[^\n]\n", temp\_u[temp\_count].univ\_address);

fscanf(file, "%[^\n]\n", temp\_u[temp\_count].univ\_email);

fscanf(file, "%[^\n]\n", temp\_u[temp\_count].univ\_website);

if (temp\_u[temp\_count].ID == ID) {

found = 1; // if university found, mark it as 1

continue;

}

temp\_count++;

}

fclose(file);

if (!found) {

printf("University with ID %d is not found.\n", ID);

return;

}

// Updating the list of universities back into the file after deleting a university

\*count = temp\_count; // Updating count after deletion

file = fopen("university\_setting.txt", "w");

if (file == NULL) {

printf("The file cannot be opened for writing.\n");

return;

}

for (int i = 0; i < \*count; i++) {

fprintf(file, "%d\n%s\n%s\n%s\n%s\n%s\n",temp\_u[i].ID, temp\_u[i].univ\_code, temp\_u[i].univ\_name,

temp\_u[i].univ\_address, temp\_u[i].univ\_email, temp\_u[i].univ\_website);

}

fclose(file);

printf("University with ID %d has been deleted successfully.\n", ID);

}

}

// Create new university

void VirtualThinkers\_university\_create(struct University u[], int \*count) {

if (\*count >= MAX\_UNIVERSITIES) {

printf("Maximum Count Reached! It is not possible to add any more universities.\n");

return;

}

printf("Enter the University ID: ");

scanf("%d", &u[\*count].ID);

// Checking for same university ID

for (int i = 0; i < \*count; i++) {

if (u[i].ID == u[\*count].ID) {

printf("Error: University with ID %d already exists. Please enter a different ID.\n", u[\*count].ID);

printf("Enter a unique University ID: ");

scanf("%d", &u[\*count].ID);

i = -1;

}

}

printf("Enter the University Code : ");

scanf(" %[^\n]", u[\*count].univ\_code);

printf("Enter the University Name (3-6 characters): ");

while (1) {

scanf(" %[^\n]", u[\*count].univ\_name);

int len = strlen(u[\*count].univ\_name);

if (len >= 3 && len <= 6) break;

else printf("Error: The name of the university must consist of 3-6 characters. Please enter again: ");

}

printf("Enter the University Address (3-6 characters): ");

while (1) {

scanf(" %[^\n]", u[\*count].univ\_address);

int len = strlen(u[\*count].univ\_address);

if (len >= 3 && len <= 6) break;

else printf("Error: The address of the university must consist of 3-6 characters. Please enter again: ");

}

printf("Enter the University Email (12-13 characters): ");

while (1) {

scanf("%s", u[\*count].univ\_email);

int len = strlen(u[\*count].univ\_email);

if (len >= 12 && len <= 13) break;

else printf("Error: The email of the university must consist of 12-13 characters. Please enter again: ");

}

printf("Enter the University Website: ");

scanf("%s", u[\*count].univ\_website);

(\*count)++;

// Storing the new university details into the file (mode 1 for writing)

VirtualThinkers\_university\_storing(u, count, 1, 0);

printf("University is added and saved successfully!\n");

}

// Update existing university details

void VirtualThinkers\_university\_update(struct University u[], int count) {

int ID;

int found = 0;

if (count == 0) {

printf("There are no universities available to update.\n");

return;

}

printf("Enter the University ID to update: ");

scanf("%d", &ID);

for (int i = 0; i < count; i++) {

if (u[i].ID == ID) {

found = 1;

printf("Updating the University Details\n");

printf("Enter the New University Code : ");

scanf(" %[^\n]", u[i].univ\_code);

printf("Enter the New University Name (3-5 characters): ");

while (1) {

scanf(" %[^\n]", u[i].univ\_name);

int len = strlen(u[i].univ\_name);

if (len >= 3 && len <= 6) break;

else printf("Error: The name of the university must consist of 3 to 6 characters. Please enter again: ");

}

printf("Enter the New University Address (3-6 characters): ");

while (1) {

scanf(" %[^\n]", u[i].univ\_address);

int len = strlen(u[i].univ\_address);

if (len >= 3 && len <= 6) break;

else printf("Error: The address of the university must consist of 3 to 6 characters. Please enter again: ");

}

printf("Enter the New University Email (12-13 characters): ");

while (1) {

scanf(" %[^\n]", u[i].univ\_email);

int len = strlen(u[i].univ\_email);

if (len >= 12 && len <= 13) break;

else printf("Error: The email of the university must consist of 12 to 13 characters. Please enter again: ");

}

printf("Enter the New University Website: ");

scanf(" %[^\n]", u[i].univ\_website);

// Storing the updated university details into the file

VirtualThinkers\_university\_storing(u, &count, 1, 0);

printf("University details are updated and saved successfully!\n");

break;

}

}

if (!found) {

printf("University with ID %d is not found.\n", ID);

}

}

// Delete university

void VirtualThinkers\_university\_delete(struct University u[], int \*count) {

int ID;

if (\*count == 0) {

printf("There are no universities available to delete.\n");

return;

}

printf("Enter the ID of the university you need to delete: ");

scanf("%d", &ID);

// Loop through to find and delete university

int found = 0;

for (int i = 0; i < \*count; i++) {

if (u[i].ID == ID) {

found = 1;

for (int j = i; j < \*count - 1; j++) {

u[j] = u[j + 1];

}

(\*count)--;

break;

}

}

if (!found) {

printf("University with ID %d is not found.\n", ID);

return;

}

// Storing the updated list of universities after deletion

VirtualThinkers\_university\_storing(u, count, 1, 0);

printf("University with ID %d has been deleted successfully.\n", ID);

}

// Merge Sort function to sort universities based on different criteria

void merge(struct University arr[], int left, int right, int sort\_by) {

if (left >= right) return;

int mid = left + (right - left) / 2;

merge(arr, left, mid, sort\_by);

merge(arr, mid + 1, right, sort\_by);

int n1 = mid - left + 1;

int n2 = right - mid;

struct University L[n1], R[n2];

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

int compare = 0;

if (sort\_by == 1)

compare = L[i].univ\_code - R[j].univ\_code;

else if (sort\_by == 2)

compare = strcmp(L[i].univ\_name, R[j].univ\_name);

else if (sort\_by == 3)

compare = strcmp(L[i].univ\_email, R[j].univ\_email);

if (compare <= 0) {

arr[k++] = L[i++];

} else {

arr[k++] = R[j++];

}

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void VirtualThinkers\_university\_MergeSort(struct University u[], int left, int right, int sort\_by) {

if (left >= right) return;

int mid = left + (right - left) / 2;

VirtualThinkers\_university\_MergeSort(u, left, mid, sort\_by);

VirtualThinkers\_university\_MergeSort(u, mid + 1, right, sort\_by);

merge(u, left, right, sort\_by);

}

void VirtualThinkers\_university\_display(struct University u[], int count) {

if (count == 0) {

printf("No universities available to display.\n");

return;

}

printf("\n\t\t\t\tList of Universities (Sorted)\n\n");

printf("--------------------------------------------------------------------------------------------------\n");

printf("\tCode\tName\t\tAddress\t\tEmail\t\tWebsite\n");

printf("--------------------------------------------------------------------------------------------------\n");

for (int i = 0; i < count; i++) {

printf("\t%s\t\t%s\t\t%s\t\t%s\t%s\n", u[i].univ\_code, u[i].univ\_name, u[i].univ\_address, u[i].univ\_email, u[i].univ\_website);

}

printf("\n");

}

void VirtualThinkers\_university\_LinearSearch(struct University u[], int count) {

if (count == 0) {

printf("No universities available to search.\n");

return;

}

int search\_by;

printf("Select search criteria:\n");

printf("1. Search by Code\n");

printf("2. Search by Name\n");

printf("3. Search by Email\n");

printf("Enter your choice: ");

scanf("%d", &search\_by);

int found = 0;

if (search\_by == 1) {

char code\_to\_search[10];

printf("Enter the code of the university to search: ");

scanf(" %[^\n]", code\_to\_search);

for (int i = 0; i < count; i++) {

if (strcmp(u[i].univ\_code, code\_to\_search) == 0) {

printf("\nUniversity found:\n");

printf("CODE: %s\n", u[i].univ\_code);

printf("NAME: %s\n", u[i].univ\_name);

printf("ADDRESS: %s\n", u[i].univ\_address);

printf("EMAIL: %s\n", u[i].univ\_email);

printf("WEBSITE: %s\n", u[i].univ\_website);

found = 1;

break;

}

}

} else if (search\_by == 2) {

char name\_to\_search[10];

printf("Enter the name of the university to search: ");

scanf(" %[^\n]", name\_to\_search);

for (int i = 0; i < count; i++) {

if (strcmp(u[i].univ\_name, name\_to\_search) == 0) {

printf("\nUniversity found:\n");

printf("CODE: %s\n", u[i].univ\_code);

printf("NAME: %s\n", u[i].univ\_name);

printf("ADDRESS: %s\n", u[i].univ\_address);

printf("EMAIL: %s\n", u[i].univ\_email);

printf("WEBSITE: %s\n", u[i].univ\_website);

found = 1;

break;

}

}

} else if (search\_by == 3) {

char email\_to\_search[15];

printf("Enter the email of the university to search: ");

scanf(" %[^\n]", email\_to\_search);

for (int i = 0; i < count; i++) {

if (strcmp(u[i].univ\_email, email\_to\_search) == 0) {

printf("\nUniversity found:\n");

printf("CODE: %s\n", u[i].univ\_code);

printf("NAME: %s\n", u[i].univ\_name);

printf("ADDRESS: %s\n", u[i].univ\_address);

printf("EMAIL: %s\n", u[i].univ\_email);

printf("WEBSITE: %s\n", u[i].univ\_website);

found = 1;

break;

}

}

}

if (!found) {

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

}

}

void VirtualThinkers\_university\_retrieve(struct University u[], int \*count) {

FILE \*file = fopen("university\_setting.txt", "r");

if (file == NULL) {

printf("There are no universities available.\n");

return;

}

\*count = 0; // Resetting university count

while (fscanf(file, "%d\n", &u[\*count].ID) == 1) {

fscanf(file, "%[^\n]\n", u[\*count].univ\_code);

fscanf(file, "%[^\n]\n", u[\*count].univ\_name);

fscanf(file, "%[^\n]\n", u[\*count].univ\_address);

fscanf(file, "%[^\n]\n", u[\*count].univ\_email);

fscanf(file, "%[^\n]\n", u[\*count].univ\_website);

(\*count)++;

}

fclose(file);

if (\*count == 0) {

printf("There are no universities available to display.\n");

return;

}

printf("\n\t\t\t\tList of Universities\n\n");

printf("--------------------------------------------------------------------------------------------------\n");

printf("\tID\tCode\t\tName\t\tAddress\t\tEmail\t\tWebsite\n");

printf("--------------------------------------------------------------------------------------------------\n");

for (int i = 0; i < \*count; i++) {

printf("\t%d\t%s\t\t%s\t\t%s\t%s\t\t%s\n",u[i].ID,u[i].univ\_code,u[i].univ\_name,u[i].univ\_address,u[i].univ\_email,u[i].univ\_website);

}

printf("\n");

}

void VirtualThinkers\_university\_Compare\_Sorting(){

printf("\nComparision of Merge and Quick Sort\n");

printf("\nMerge Sort\n");

printf("1. Merge sort also follows the divide & conquer approach. \nit finds a mid element and divides the data continuously into 2 parts till each array has 1 element\n Then it sorts those sub arrays and merges back into a single array \n returns the sorted array.\n");

printf("2. It is a stable sorting algorithm.\n");

printf("3. It works faster in execution due to its recursive calls, there are multiple calls and list is sorted very fast.\n");

printf("4. Space complexity of Merge sort is O(n).\n");

printf("5. Works well on any size of data.\n");

printf("6. It sorts externally.\n");

printf("\n\tQuick Sort\n");

printf("1. QuickSort follows the divide & conquer approach \nthat selects an element as pivot element and divides/ partitions the array or \nlist around the picked pivot element by placing the pivot element in its correct position\n in the sorted array.\n The partition or array does not follow any ration only follows the position of pivot element in sorted array.\n");

printf("2. It is an unstable sorting algorithm.\n");

printf("3. Slow in execution due to iterative calls.\n");

printf("4. The space complexity of quicksort is O(log(n)) in the average case, and O(n) in the worst case.\n");

printf("5. It is not efficient for larger set of data.\n");

printf("6. Sorting is done internally.\n\n");

}

void VirtualThinkers\_university\_Compare\_Search(){

printf("\nComparision of Linear and binary Search\n");

printf("\n\tBinary Search\n");

printf("1. needs sorted data as input.\n");

printf("2. Only single dimension arrays/ lists can be used due to divide and conquer approach.\n");

printf("3. Binary Search follows the divide and conquer approach.\n It divides the array / list into 2 halves from the mid element, \nthen compares if the mid element is equal to the search element, if yes then return the mid index or position value. Else not equal, it checks if search element is smaller or greater than mid element. If smaller than mid, \nthen this process is repeated on first half or array / list. \nElse the search value greater than mid element, then same process repeated on second half of list / array.\n");

printf("4. Binary search is more efficient for large, ordered lists.\n");

printf("5. Faster in execution as only half of the data is to be checked and only 1 element is checked if same or not.\n");

printf("6. Space complexity of the binary search is O(1).\n");

printf("\n\tLinear Search\n");

printf("1. works the same for sorted and unsorted data.\n(ie. order of data does not affect the working of algorithm)\n");

printf("2. Multidimensional array can be used as it uses an iterative approach.\n");

printf("3. Linear search also called as the Sequential search follows an iterative approach.\n It iterates through each element of the list and compares it with the element to be searched.\n If the search element is found, it returns the position or index of the element.\n Else not found it returns a value (-1) NULL to show the element is not found.\n");

printf("4. Linear search is better for smaller lists or unordered data.\n");

printf("5. Slower in execution as iteration is done on every element by comparing each element every iteration.\n");

printf("6. Space complexity of the linear search is O(1).\n\n");

}

void VirtualThinkers\_university\_compexity\_Sorting(){

printf("Time Complexity of Sort Algorithms:\n");

printf("\n1. Quick Sort:");

printf("\nBest Case: O(n\*log(n))");

printf("\nAverage Case: O(n\*log(n))");

printf("\nWorst Case: O(n^2)");

printf("\n2. Merge Sort:");

printf("\nBest Case: O(n log n)");

printf("\nAverage Case: O(n log n)");

printf("\nWorst Case: O(n log n)\n");

}

void VirtualThinkers\_university\_compexity\_Search(){

printf("Time Complexity of Search Algorithms:\n");

printf("\n1. Linear Search:");

printf("\nBest Case: O(1)");

printf("\nAverage Case: O(n)");

printf("\nWorst Case: O(n)");

printf("\n2. Binary Search:");

printf("\nBest Case: O(1)");

printf("\nAverage Case: O(log n)");

printf("\nWorst Case: O(log n)\n");

}

void VirtualThinkers\_university\_Sort\_details() {

printf("\nMerge Sort Algorithm \nDivide and Conquer:\n");

printf("---------------------------------\n");

printf("Merge Sort is a divide and conquer algorithm that recursively divides an array into two halves,\n sorts each half, and then merges the sorted halves into a single sorted array.");

printf("\nBase Case:\n\tThe array has only 1 or no elements, the array is already sorted.\n The algorithm stops dividing when it reaches this condition.");

printf("\nRecursive Division:\n");

printf("For an array segment arr[left...right]:\n");

printf("\nFind the middle index: mid = (left + right) / 2.");

printf("\nRecursively apply Merge Sort to the first half: MergeSort(arr, left, mid).");

printf("\nRecursively apply Merge Sort to the second half: MergeSort(arr, mid + 1, right).");

printf("\n\tMerge Step:");

printf("\nAfter both halves are sorted, merge them back together in sorted order:");

printf("\nCreate Temporary Arrays:");

printf("\nCreate two temporary arrays to hold the elements of the left and right halves.");

printf("\nL[] holds elements from arr[left] to arr[mid].");

printf("\nR[] holds elements from arr[mid + 1] to arr[right].");

printf("\nInitialize Pointers:");

printf("\ni will track the current index in L[].");

printf("\nj will track the current index in R[].");

printf("\nk will track the position in the original array where the smallest element should go.");

printf("\nMerge Elements in Sorted Order:");

printf("\nWhile there are elements in both L[] and R[], compare L[i] with R[j]:");

printf("\nIf L[i] is smaller, place L[i] in arr[k] and increment i and k.");

printf("\nIf R[j] is smaller, place R[j] in arr[k] and increment j and k.");

printf("\nCopy Remaining Elements:");

printf("\nIf there are any remaining elements in L[], copy them to arr[k].");

printf("\nIf there are any remaining elements in R[], copy them to arr[k].");

printf("\n\tEnd of Recursive Calls:");

printf("\nThe recursive calls continue until all array segments have been sorted and merged,\nresulting in a fully sorted array.");

printf("\nSummary of steps:\nDivide the array into two halves.");

printf("\nSort each half recursively.\nMerge the two sorted halves into a sorted whole.");

printf("---------------------------------\n");

printf("\nQuick Sort \nDivide and Conquer:\n");

printf("----------------------------------\n");

printf("\nQuick Sort is a divide and conquer algorithm,\nwhich recursively partitions an array around a pivot");

printf("\nPlacing elements smaller than the pivot on its left and elements greater than the pivot on its right");

printf("\nBase Case:\nIf the array segment has zero or one element, it is already sorted.");

printf("\nQuick Sort stops dividing when it reaches this base case.");

printf("\nPartition Step:");

printf("\nFor an array segment arr[left...right]:\nChoose a Pivot:");

printf("\nSelect a pivot element from the array. Common choices include:\nThe first element (arr[left]).\nThe last element (arr[right]).\nThe middle element (arr[(left + right) / 2]).");

printf("\nHere, let's assume the pivot is arr[right] (the last element in the segment).");

printf("\nReorder Elements:");

printf("\nInitialize two pointers:\ni starts at left - 1 (just before the first element).\nj traverses from left to right - 1.");

printf("\nFor each element arr[j]:");

printf("\nIf arr[j] is less than or equal to the pivot, \nincrement i and swap arr[i] with arr[j] to move smaller elements to the left of the pivot.");

printf("\n\tPlace the Pivot:");

printf("\nAfter the loop, place the pivot at its correct position by swapping arr[i + 1] with arr[right].");

printf("\nThe pivot is now at index i + 1, \nwith all smaller elements on the left and all greater elements on the right.");

printf("\n\tRecursive Sort:");

printf("\nRecursively apply Quick Sort on the left and right partitions:");

printf("\nQuickSort(arr, left, i)sorts the elements on the left of the pivot.\nQuickSort(arr, i + 2, right)\nsorts the elements on the right of the pivot.");

printf("\n\tEnd of Recursive Calls:");

printf("\nThe recursive calls continue until all segments have been sorted, resulting in a fully sorted array.");

printf("\nSummary of Steps:");

printf("\nChoose a Pivot.\nPartition the array around the pivot, \nplacing smaller elements to its left and larger elements to its right.");

printf("\nRecursively Sort the left and right partitions.");

printf("----------------------------------\n");

}

void VirtualThinkers\_university\_Search\_details(){

printf("\nLinear Search Algorithm Pseudocode:\n");

printf("-----------------------------------\n");

printf("\nfunction linearSearch(arr, target):");

printf("\n for each element in arr:");

printf("\n if element == target:");

printf("\n return the index of element");

printf("\n return -1 (if target is not found)\n");

printf("-----------------------------------\n\n");

printf("\nBinary Search Algorithm Pseudocode:\n");

printf("------------------------------------\n");

printf("\nfunction binarySearch(arr, target):");

printf("\n low = 0");

printf("\n high = length of arr - 1");

printf("\n while low <= high:");

printf("\n mid = (low + high) / 2");

printf("\n if arr[mid] == target:");

printf("\n return mid");

printf("\n else if arr[mid] < target:");

printf("\n low = mid + 1");

printf("\n else:");

printf("\n high = mid - 1\n");

printf("\n return -1 (if target is not found)\n");

printf("-----------------------------------\n\n");

}

void VirtualThinkers\_university\_displayMenu() {

printf("\*\*\*\* University Module \*\*\*\*\*\*\*\*\n");

printf("1. Create New University\n");

printf("2. Update University\n");

printf("3. Retrieve University Details\n");

printf("4. Delete University Details\n");

printf("5. Sort University Details\n");

printf("6. Search University Details\n");

printf("7. Compare Sort Algorithms\n");

printf("8. Compare Search Algorithms\n");

printf("9. Display Time Complexity of Sorting Algorithm\n");

printf("10. Display Time Complexity of Searching Algorithm\n");

printf("11. Display Sort Algorithm Details\n");

printf("12. Display Search Algorithm Details\n");

printf("13. Exit Application\n");

}

int main() {

struct University uni[MAX\_UNIVERSITIES];

int count = 0;

int option,sort\_by;

do {

VirtualThinkers\_university\_displayMenu();

printf("Please select an option: ");

scanf("%d", &option);

getchar();

switch (option) {

case 1:

VirtualThinkers\_university\_create(uni, &count);

break;

case 2:

VirtualThinkers\_university\_update(uni, count);

break;

case 3:

VirtualThinkers\_university\_retrieve(uni, &count);

break;

case 4:

VirtualThinkers\_university\_delete(uni, &count);

break;

case 5:

printf("Choose sort criteria:\n1. Sort by Code\n2. Sort by Name\n3. Sort by Email\n");

scanf("%d", &sort\_by);

VirtualThinkers\_university\_MergeSort(uni, 0, count - 1, sort\_by);

printf("Universities sorted successfully.\n");

VirtualThinkers\_university\_display(uni, count); // Display sorted list

break;

case 6:

VirtualThinkers\_university\_LinearSearch(uni, count);

break;

case 7:

VirtualThinkers\_university\_Compare\_Sorting( );

break;

case 8:

VirtualThinkers\_university\_Compare\_Search( );

break;

case 9:

VirtualThinkers\_university\_compexity\_Sorting();

break;

case 10:

VirtualThinkers\_university\_compexity\_Search();

break;

case 11:

VirtualThinkers\_university\_Sort\_details();

break;

case 12:

VirtualThinkers\_university\_Search\_details();

break;

case 13:

printf("Exiting application...\n");

break;

default:

printf("Invalid option. Please enter a valid option.\n");

break;

}

} while (option != 13);

return 0;

}