**Case Study 2**

1. **Display**

A screenshot of a computer

Description automatically generated

1. **SelectRow**

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **SortBy**

A screenshot of a computer

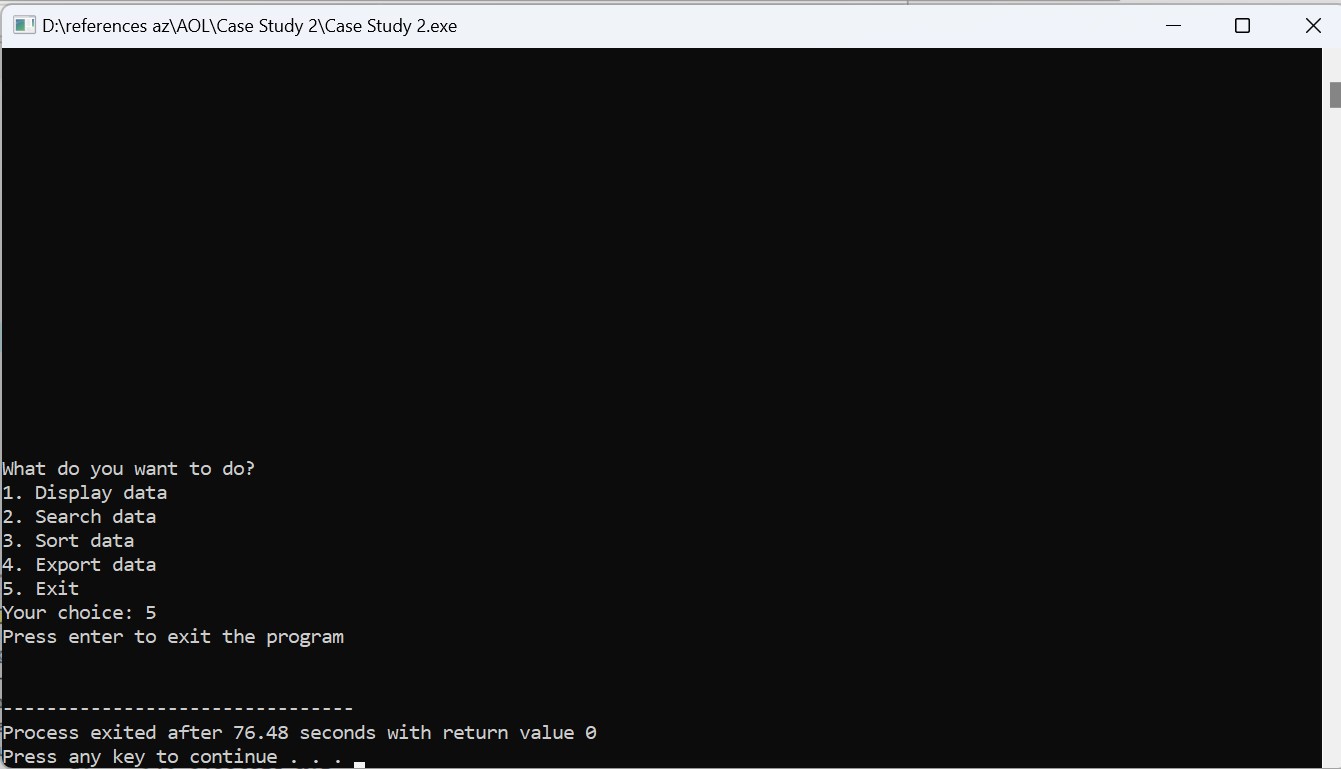
Description automatically generated

1. **Export**

A black screen with white text

Description automatically generated

1. **Exit**



**CODE**

#include <stdio.h>

#include <string.h>

struct file{

char location[20], city[20];

int price, room, bathroom, carpark;

char type[20], furnish[20];

};

// gives spaces

void clearScreen(){

int i;

for (i=0; i<30; i++){

puts("");

}

}

void printFormat(){

printf("%-26s %-20s %-10s %-8s %-12s %-10s %-12s %s\n", "Location", "City", "Price", "Rooms", "Bathrooms", "Carparks", "Type", "Furnish");

}

void printData(struct file data[], int i){

printf("%-26s %-20s %-10d %-8d %-12d %-10d %-12s %s\n", data[i].location, data[i].city, data[i].price, data[i].room, data[i].bathroom, data[i].carpark, data[i].type, data[i].furnish);

}

// 1. DISPLAY DATA

void print(struct file data[], int n){

int i;

printFormat();

for(i=1; i<=n; i++){

printData(data, i);

}

}

// 2.A. SEARCH DATA (Find and print data by the column (Location/City/Price/Rooms/Bathrooms/Carpark/Type/Furnish))

// using linear search

void findLocation(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].location) == 0){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].location) == 0){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findCity(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].city) == 0){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].city) == 0){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findPrice(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (target == data[i].price){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (target == data[i].price){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findRoom(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (target == data[i].room){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (target == data[i].room){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findBathroom(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (target == data[i].bathroom){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (target == data[i].bathroom){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findCarpark(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (target == data[i].carpark){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (target == data[i].carpark){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findType(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].type) == 0){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].type) == 0){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

void findFurnish(struct file data[], int idx, char search[], int target){

int i;

int found = 0;

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].furnish) == 0){

found = 1;

break;

}

}

if (found) {

printf("Data found. Detail of data:\n");

printFormat();

for(i=1; i<=idx; i++){

if (strcmpi(search, data[i].furnish) == 0){

printData(data, i);

}

}

} else {

printf("Data not found!\n");

}

}

// 2.B. SEARCH DATA (find the column)

void findColumn(struct file data[], int idx, char column[], char search[], int target){

if (strcmpi(column, "Location") == 0){

findLocation(data, idx, search, target);

} else if (strcmpi(column, "City") == 0){

findCity(data, idx, search, target);

} else if (strcmpi(column, "Price") == 0){

findPrice(data, idx, search, target);

} else if (strcmpi(column, "Rooms") == 0){

findRoom(data, idx, search, target);

}else if (strcmpi(column, "Bathrooms") == 0){

findBathroom(data, idx, search, target);

} else if (strcmpi(column, "Carparks") == 0){

findCarpark(data, idx, search, target);

} else if (strcmpi(column, "Type") == 0){

findType(data, idx, search, target);

} else if (strcmpi(column, "Furnish") == 0){

findFurnish(data, idx, search, target);

}

else { // if column is not found

printf("Data not found!\n");

}

}

// swap struct

void swap(struct file \*a, struct file \*b){

struct file temp = \*a;

\*a = \*b;

\*b = temp;

}

// 3.A. SORT DATA (get partition by the column)

int getPartition(struct file data[], int left, int right, char column[], char sort[]){

struct file pivot = data[right];

int idx = left-1;

int i;

// Column = Location

if (strcmpi(column, "Location") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (strcmp(data[i].location, pivot.location) > 0){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (strcmp(data[i].location, pivot.location) <= 0){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = City

else if (strcmpi(column, "City") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (strcmp(data[i].city, pivot.city) > 0){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (strcmp(data[i].location, pivot.city) <= 0){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = Price

else if (strcmpi(column, "Price") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (data[i].price > pivot.price){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (data[i].price <= pivot.price){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = Rooms

else if (strcmpi(column, "Rooms") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (data[i].room > pivot.room){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (data[i].room <= pivot.room){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = Bathrooms

else if (strcmpi(column, "Bathrooms") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (data[i].bathroom > pivot.bathroom){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (data[i].bathroom <= pivot.bathroom){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = Carpark

else if (strcmpi(column, "Carparks") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (data[i].carpark > pivot.carpark){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (data[i].carpark <= pivot.carpark){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = Type

else if (strcmpi(column, "Type") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (strcmp(data[i].type, pivot.type) > 0){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (strcmp(data[i].type, pivot.type) <= 0){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} // Column = Furnish

else if (strcmpi(column, "Furnish") == 0){

if (strcmpi(sort, "dsc") == 0){ // descending

for(i = left; i<right; i++){

if (strcmp(data[i].furnish, pivot.furnish) > 0){

idx++;

swap(&data[i], &data[idx]);

}

}

} else { // ascending

for(i = left; i<right; i++){

if (strcmp(data[i].furnish, pivot.furnish) <= 0){

idx++;

swap(&data[i], &data[idx]);

}

}

}

} else { // if column's data isn't found

return -2;

}

idx++;

swap(&data[right], &data[idx]);

return idx;

}

// 3.B. SORT DATA (quick sort)

void quickSort(struct file data[], int left, int right, char column[], char sort[]){

if (left>= right){

return;

}

int partition = getPartition(data,left,right,column,sort);

quickSort(data, left, partition-1, column, sort);

quickSort(data, partition+1, right, column, sort);

}

// 4. EXPORT DATA

void writeData(struct file data[], char skip[], int idx, char fileName[]){

FILE \*fr = fopen(fileName, "w");

if (fr==NULL) {

perror("Error");

return;

}

// write data into the file

fprintf(fr, "%s\n", skip);

int i;

for(i=1; i<=idx; i++){

fprintf(fr, "%s,%s,%d,%d,%d,%d,%s,%s\n", data[i].location, data[i].city, data[i].price, data[i].room, data[i].bathroom, data[i].carpark, data[i].type, data[i].furnish);

}

fclose(fr);

}

int main(){

struct file data[5000];

char skip[105];

int idx; // initialize index

FILE \*fp = fopen("file.csv", "r");

if (fp==NULL){

perror("Error");

return 1;

} else {

while(!feof(fp)){

// skip the 1st line in csv file

if (idx == 0){

fscanf(fp, "%[^\n]\n", skip);

} else {

// read data

fscanf(fp, "%[^,],%[^,],%d,%d,%d,%d,%[^,],%[^\n]\n", &data[idx].location, &data[idx].city, &data[idx].price, &data[idx].room, &data[idx].bathroom, &data[idx].carpark, &data[idx].type, &data[idx].furnish);

}

idx++;

}

fclose(fp);

}

int choose;

do {

clearScreen(); // gives spaces

printf("What do you want to do?\n");

printf("1. Display data\n");

printf("2. Search data\n");

printf("3. Sort data\n");

printf("4. Export data\n");

printf("5. Exit\n");

int n, target;

char column[20], search[20], sort[5], fileName[20];

do{

printf("Your choice: ");

scanf("%d", &choose); getchar();

} while (choose <1 || choose > 5); // validation, user can only choose between 1 to 5

switch(choose){

case 1: // DISPLAY DATA

do {

printf("Number of rows: ");

scanf("%d", &n); getchar();

} while (n < 1); // n must be a positive integer

puts("");

if (n > idx){ // if n > total number of rows then print all data

print(data, idx); // Display all data

} else {

print(data, n); // Display n rows of data

}

break;

case 2: // SEARCH DATA

printf("Choose column: ");

scanf("%s", column); getchar();

printf("What data do you want to find? ");

// if the column is price/rooms/bathrooms/carpark then the data must be an integer

if (strcmpi(column,"Price")==0 || strcmpi(column,"Rooms")==0 || strcmpi(column,"Bathrooms")==0 || strcmpi(column,"Carpark")==0){

scanf("%d", &target); getchar();

} else { // or else the data must be a string

scanf("%s", search); getchar();

}

puts("");

// search and display data

findColumn(data, idx, column, search, target);

break;

case 3: // SORT DATA

printf("Choose column: ");

scanf("%s", column); getchar();

printf("Sort ascending or descending? ");

scanf("%s", sort); getchar();

puts("");

if (getPartition(data,1,idx-1,column,sort) == -2){ // if column is not found

printf("Data not found!\n");

}

else { // if column is found

// Sort Data

quickSort(data, 1, idx-1, column, sort);

printf("Data found. Detail of sorted data:\n");

print(data, 10); // Display data

}

break;

case 4: // EXPORT DATA

printf("File name: ");

scanf("%[^\n]", fileName); getchar();

puts("");

strcat(fileName, ".csv"); // add .csv to the file name

writeData(data, skip, idx, fileName); // write data into the file

printf("Data succsessfully written to file %s!\n", fileName);

break;

case 5: // EXIT PROGRAM

printf("Press enter to exit the program\n");

break;

} getchar();

} while (choose != 5); // the program won't stop until user chose 5

return 0;

}