```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX_ARRAY_SIZE 50

#define CURRENT_YEAR 2017


const char owner_filename[] = "owners";

const char car_filename[] = "car";


struct owner{

   long owner_ID;

   char name[MAX_ARRAY_SIZE];

};


struct car{

   long car_ID;

   int year;

   int n_owners;

   char colour[MAX_ARRAY_SIZE];

   int owners_ID[5];

};


struct node{

   struct car car_node;

   struct node *next_ptr;

};


struct node *head= NULL;


long add_owner(long car_ID);
```

```
void add_car(long owner_ID);

void link_car(long owner_ID);

long link_owner(void);

void create_linked_list(void);

void insert(struct car white_car);

int node_already_exists(struct car white_car);

void delete_car(void);

void delete_cars_from_list(void);

void clear(void);


int main(void){

   int c;


   printf("Press 'c' to add a new car\nPress 'a' to add a new owner\nPress 'd' to delete a car\nPress 'l'
to create a linked list of white cars\nPress 'r' to remove cars made before 1980 from the linked
list\n");

   while((c = getchar()) != EOF){

      switch(c){

      case 'a':

         clear();

         add_owner(0);

         break;

      case 'c':

         clear();

         add_car(0);

         break;

      case 'd':

         clear();

         delete_car();

         break;

      case 'l':

         clear();
```

```
            create_linked_list();
            break;
        case 'r':
            clear();
            delete_cars_from_list();
            break;
        default:
            clear();
            printf("Invalid command: %c\n", c);
            break;
        }
        puts("Press 'c' to add a new car, 'a' to add a new owner, 'd' to delete a car, 'l' to create a linked
list, 'r' to remove cars from the linked list");
    }
    return 0;
}

long add_owner(long car_ID){
    FILE *binary_file;

    if((binary_file = fopen(owner_filename, "a+b")) == NULL){
        perror(owner_filename);
        exit(EXIT_FAILURE);
    }

    struct owner holder;

    puts("Enter the owner's ID");

    if(1 != scanf("%li", &holder.owner_ID)){
        fprintf(stderr, "Unable to read number");
        exit(EXIT_FAILURE);
    }
    clear();

    while(holder.owner_ID <= 0){
        puts("The ID cannot be 0 or less. Please enter a new ID");
        if(1 != scanf("%li", &holder.owner_ID)){
            fprintf(stderr, "Unable to read number");
            exit(EXIT_FAILURE);
        }
        clear();
    }

    puts("Enter the owner's name");
    fgets(holder.name, sizeof(holder.name), stdin);

    struct owner temp;

    while((fread(&temp, sizeof(temp), 1, binary_file)) != 0){
        if(holder.owner_ID == temp.owner_ID){
            puts("Owner already exists");
            return -1;
        }
    }

    fwrite(&holder, sizeof(holder), 1, binary_file);
    fclose(binary_file);
    puts("Added new owner successfully");

    if(car_ID > 0){
        return holder.owner_ID;
    }
    else{
        puts("Press 'n' to add a new car for this owner, or 'e' to link him to an already existing car");
        int c;
        while((c = getchar()) != EOF){
            switch(c){
            case 'n':
                clear();
                add_car(holder.owner_ID);
                break;
            case 'e':
                clear();
                link_car(holder.owner_ID);
                break;
            default:
                clear();
                printf("Invalid command: %c\n", c);
                break;
            }
            puts("Press 'n' to add a new car for this owner, or 'e' to link him to an already existing car");
        }
    }
}


void add_car(long owner_ID){
    FILE *binary_file;

    if((binary_file = fopen(car_filename, "a+b")) == NULL){
        perror(car_filename);
        exit(EXIT_FAILURE);
```

```
}

struct car new_car;
new_car.n_owners = 0;

puts("Enter the car's ID");

if(1 != scanf("%li", &new_car.car_ID)){
   fprintf(stderr, "Unable to read number");
   exit(EXIT_FAILURE);
}
clear();

while(new_car.car_ID <= 0){
   puts("The ID cannot be 0 or less. Please enter a new ID");
   if(1 != scanf("%li", &new_car.car_ID)){
      fprintf(stderr, "Unable to read number");
      exit(EXIT_FAILURE);
   }
   clear();
}

puts("Enter the year of the making");

if(1 != scanf("%i", &new_car.year)){
   fprintf(stderr, "Unable to read number");
   exit(EXIT_FAILURE);
}
clear();

while((new_car.year < 1885) || (new_car.year > CURRENT_YEAR)){
    puts("The year of the making can only be between 1885 and the current year");
    if(1 != scanf("%i", &new_car.year)){
        fprintf(stderr, "Unable to read number");
        exit(EXIT_FAILURE);
    }
    clear();
}

puts("Enter the colour of the car");
fgets(new_car.colour, sizeof(new_car.colour), stdin);

for(int i = 0; new_car.colour[i] != '\n'; ++i){
    if(!(isalpha(new_car.colour[i]))){
        fprintf(stderr, "Invalid character: %c\n", new_car.colour[i]);
        exit(EXIT_FAILURE);
    }
    new_car.colour[i] = tolower(new_car.colour[i]);
}

struct car temp;

while((fread(&temp, sizeof(temp), 1, binary_file)) != 0){
    if(new_car.car_ID == temp.car_ID){
        puts("Car already exists");
        return;
    }
}

if(owner_ID > 0){
    new_car.owners_ID[0] = owner_ID;
    new_car.n_owners++;
    fwrite(&new_car, sizeof(new_car), 1, binary_file);
    puts("Added new car successfully to this owner");
    fclose(binary_file);
    return;
}

puts("Press 'n' to add a new owner for this car, or 'e' to link it to an already existing owner");
int c;
while((c = getchar()) != EOF){
    switch(c){
    case 'n':
        clear();
        if(new_car.n_owners < 5){
            long owner_ID = add_owner(new_car.car_ID);
            if(owner_ID > 0){
                new_car.owners_ID[new_car.n_owners] = owner_ID;
                new_car.n_owners++;
            }
        }
        else{
            puts("This car already has 5 owners");
        }
        break;
    case 'e':
        clear();
        if(new_car.n_owners < 5){
            long link_owner_ID = link_owner();
            if(link_owner_ID > 0){
                new_car.owners_ID[new_car.n_owners] = link_owner_ID;
                new_car.n_owners++;
                puts("Owner linked successfully");
            }
          }
          else{
            puts("This car already has 5 owners");
          }
          break;
        default:
          clear();
          printf("Invalid command: %c\n", c);
          break;
      }
      puts("Press 'n' to add a new owner for this car, or 'e' to link it to an already existing owner");
    }

    fwrite(&new_car, sizeof(new_car), 1, binary_file);
    fclose(binary_file);
    puts("Added new car successfully");
}


void link_car(long owner_ID){
    FILE *binary_file;

    if((binary_file = fopen(car_filename, "rb")) == NULL){
      perror(car_filename);
      exit(EXIT_FAILURE);
    }

    FILE *copy;

    if((copy = fopen("copy", "wb")) == NULL){
```

```
    perror("copy");
    exit(EXIT_FAILURE);
}

struct car linked_car;

puts("Enter the ID of the car you want to link this owner to");
long link_ID;
if(1 != scanf("%li", &link_ID)){
    fprintf(stderr, "Unable to read number");
    exit(EXIT_FAILURE);
}
clear();

while(link_ID <= 0){
    puts("The ID cannot be 0 or less. Please enter a new ID");
    if(1 != scanf("%li", &link_ID)){
        fprintf(stderr, "Unable to read number");
        exit(EXIT_FAILURE);
    }
    clear();
}

while((fread(&linked_car, sizeof(linked_car), 1, binary_file)) != 0){
    if(linked_car.car_ID == link_ID){
        if(linked_car.n_owners >= 5){
            puts("This car already has maximum number of owners");
            return;
        }
        else{
            rewind(binary_file);
```

```
while((fread(&linked_car, sizeof(linked_car), 1, binary_file)) != 0){
   if(linked_car.car_ID == link_ID){
      linked_car.owners_ID[linked_car.n_owners] = owner_ID;
      linked_car.n_owners++;
      fwrite(&linked_car, sizeof(linked_car), 1, copy);
      continue;
   }
   fwrite(&linked_car, sizeof(linked_car), 1, copy);
}

fclose(binary_file);
fclose(copy);

if((binary_file = fopen(car_filename, "wb")) == NULL){
   perror(car_filename);
   exit(EXIT_FAILURE);
}

if((copy = fopen("copy", "rb")) == NULL){
   perror("copy");
   exit(EXIT_FAILURE);
}

while((fread(&linked_car, sizeof(linked_car), 1, copy)) != 0){
   fwrite(&linked_car, sizeof(linked_car), 1, binary_file);
}
fclose(binary_file);
fclose(copy);
remove("copy");
puts("Linked car to owner successfully");
return;
```

```
            }
        }
    }
    fclose(binary_file);
    fclose(copy);
    remove("copy");
    puts("Car not found");
}


long link_owner(void){
    FILE *binary_file;

    puts("Enter the ID of the owner you want to link this car to");
    long owner_ID;

    if(1 != scanf("%li", &owner_ID)){
        fprintf(stderr, "Unable to read number");
        exit(EXIT_FAILURE);
    }
    clear();

    while(owner_ID <= 0){
        puts("The ID cannot be 0 or less. Please enter a new ID");
        if(1 != scanf("%li", &owner_ID)){
            fprintf(stderr, "Unable to read number");
            exit(EXIT_FAILURE);
        }
        clear();
    }
    if((binary_file = fopen(owner_filename, "rb")) == NULL){
        perror(owner_filename);
        exit(EXIT_FAILURE);
    }

    struct owner temp;

    while((fread(&temp, sizeof(temp), 1, binary_file)) != 0){
        if(owner_ID == temp.owner_ID){
            return owner_ID;
        }
    }
    fclose(binary_file);
    puts("Owner not found");
    return -1;
}

void delete_car(void){
    puts("Enter the ID of the car you want to delete");
    long delete_cID;

    if(1 != scanf("%li", &delete_cID)){
        fprintf(stderr, "Unable to read number");
        exit(EXIT_FAILURE);
    }

    while(delete_cID <= 0){
        puts("The ID cannot be 0 or less. Please enter a new ID");
        if(1 != scanf("%li", &delete_cID)){
            fprintf(stderr, "Unable to read number");
            exit(EXIT_FAILURE);
        }
        clear();
    }
    FILE *binary_file;

    if((binary_file = fopen(car_filename, "rb")) == NULL){
        perror(car_filename);
        exit(EXIT_FAILURE);
    }

    struct car temp;

    int found_car = 0;
    while((fread(&temp, sizeof(temp), 1, binary_file)) != 0){
        if(temp.car_ID == delete_cID){
            found_car = 1;
            break;
        }
    }

    if(found_car == 0){
        puts("Car not found");
        return;
    }

    struct car other_cars;
    rewind(binary_file);

    for(int i = 0; i < temp.n_owners; ++i){
        while((fread(&other_cars, sizeof(other_cars), 1, binary_file)) != 0){
            if(other_cars.car_ID == delete_cID){
                continue;
            }
            for(int j = 0; j < other_cars.n_owners; ++j){
                if(temp.owners_ID[i] == other_cars.owners_ID[j]){
                    for(int k = i; k < temp.n_owners; ++k){
                        temp.owners_ID[k] = temp.owners_ID[k+1];
                    }
                    temp.n_owners--;
                    --i;
                    break;
                }
            }
        }
        rewind(binary_file);
    }

    for(int i = 0; i < temp.n_owners; ++i){
        printf("the IDs left are%i\n", temp.owners_ID[i]);
    }
    fclose(binary_file);

    FILE *owner_file;

    struct owner owner_temp;

    if((owner_file = fopen(owner_filename, "rb")) == NULL){
        perror(owner_filename);
        exit(EXIT_FAILURE);
    }

    FILE *owner_copy_ptr;
```

```
if((owner_copy_ptr = fopen("owner_copy", "wb")) == NULL){
    perror("owner_copy");
    exit(EXIT_FAILURE);
}

int flag = 1;
while(fread(&owner_temp, sizeof(owner_temp), 1, owner_file) != 0){
    flag = 1;
    for(int i = 0; i < temp.n_owners; ++i){
        if(owner_temp.owner_ID == temp.owners_ID[i]){
            flag = 0;
            break;
        }
    }
    if(flag == 1){
        fwrite(&owner_temp, sizeof(owner_temp), 1, owner_copy_ptr);
    }
}



fclose(owner_file);
fclose(owner_copy_ptr);

if((owner_file = fopen(owner_filename, "wb")) == NULL){
    perror(owner_filename);
    exit(EXIT_FAILURE);
}

if((owner_copy_ptr = fopen("owner_copy", "rb")) == NULL){
    perror("owner_copy");
    exit(EXIT_FAILURE);
}

while(fread(&owner_temp, sizeof(owner_temp), 1, owner_copy_ptr) != 0){
    fwrite(&owner_temp, sizeof(owner_temp), 1, owner_file);
}

fclose(owner_file);
fclose(owner_copy_ptr);
remove("owner_copy");



if((binary_file = fopen(car_filename, "rb")) == NULL){
    perror(car_filename);
    exit(EXIT_FAILURE);
}

FILE *car_copy_ptr;

if((owner_copy_ptr = fopen("car_copy", "wb")) == NULL){
    perror("car_copy");
    exit(EXIT_FAILURE);
}

while((fread(&other_cars, sizeof(other_cars), 1, binary_file)) != 0){
    if(other_cars.car_ID == delete_cID){
        puts("found car");
        continue;
    }
    fwrite(&other_cars, sizeof(other_cars), 1, car_copy_ptr);
}
```

```
    fclose(binary_file);
    fclose(car_copy_ptr);

    if((binary_file = fopen(car_filename, "wb")) == NULL){
        perror(car_filename);
        exit(EXIT_FAILURE);
    }

    if((car_copy_ptr = fopen("car_copy", "rb")) == NULL){
        perror("car_copy");
        exit(EXIT_FAILURE);
    }

    while((fread(&other_cars, sizeof(other_cars), 1, car_copy_ptr)) != 0){
        fwrite(&other_cars, sizeof(other_cars), 1, binary_file);
    }
    fclose(binary_file);
    fclose(car_copy_ptr);
    remove("car_copy");
}


void create_linked_list(){
    FILE *binary_file;

    if((binary_file = fopen(car_filename, "rb")) == NULL){
        perror(car_filename);
        exit(EXIT_FAILURE);
    }
    struct car temp;
    const char insert_colour[] = "white";

    int flag = 0;
    while((fread(&temp, sizeof(temp), 1, binary_file)) != 0){
        if(strstr(temp.colour, insert_colour)){
            if(!node_already_exists(temp)){
                insert(temp);
            }
            flag = 1;
        }
    }
    if(flag){
        puts("Created linked list successfully");
    }
    else{
        puts("No white cars were found");
    }
    fclose(binary_file);
}

void insert(struct car white_car){
    struct node *temp = head;
    struct node *prev;
    struct node *new_node;

    if(head == NULL){
        head = malloc(sizeof(struct node));
        head->car_node = white_car;
        head->next_ptr = NULL;
        return;
    }

    if(head->car_node.year > white_car.year){
        new_node = malloc(sizeof(struct node));
        new_node->car_node = white_car;
        new_node->next_ptr = head;
        head = new_node;
        return;
    }

    while((temp != NULL) && (temp->car_node.year < white_car.year)){
        prev = temp;
        temp = temp->next_ptr;
    }

    new_node = malloc(sizeof(struct node));
    new_node->car_node = white_car;
    new_node->next_ptr = temp;
    prev->next_ptr = new_node;
}

int node_already_exists(struct car white_car){
    struct node *temp = head;

    while(temp != NULL){
        if(temp->car_node.car_ID == white_car.car_ID){
            return 1;
        }
        temp = temp->next_ptr;
    }
    return 0;
```

```
}

void delete_cars_from_list(void){
   struct node *temp;

   if(head == NULL){
      puts("Linked list is empty");
      return;
   }

   while(head->car_node.year < 1980){
      temp = head;
      if(head->next_ptr == NULL){
         head = NULL;
         free(temp);
         return;
      }
      else{
         head = head->next_ptr;
         free(temp);
      }
   }

   struct node *cur = head;
   struct node *prev = NULL;

   while(cur != NULL){
      if(cur->car_node.year < 1980){
         temp = cur;
         prev->next_ptr = cur->next_ptr;
         cur = cur->next_ptr;
```

```
            free(temp);
            puts("Node Deleted...");
            continue;
        }
        prev = cur;
        cur = cur->next_ptr;
    }
}


void clear(void){
    int c;
    while((c = getchar()) != '\n' && (c != EOF));
}
```

```
Welcome to Carpooling System
1. Register Vehicle
2. Add Trip
3. Send Request
4. Approve/Reject Request
5. Make Payment
6. Leave Feedback
7. Display Available Vehicles
8. Display Pending Trips
9. Display Payments
10. Display Feedbacks
0. Exit
Enter your choice: 1
Enter vehicle details:
Name: bmw
Vehicle Type (Luxury/Regular): regular
Cost per Km: 12km/h
```

Vehicle registered successfully.