```
// LISTA DE CADASTRO DE CLIENTES - OFICINA MECテ�NICA AUTOMOTIVA
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <ctype.h>

#ifdef _WIN32
   #include <conio.h>
   #define LIMPAR_CMD "cls"
#else
   #define LIMPAR_CMD "clear"
   // ATENテ�テグ: Este getch simples sテウ lテェ o caractere, mas **nテ」o** permite a leitura de teclas como as setas ou ESC
   // sem buffer em sistemas POSIX (Linux/macOS). Para as setas funcionarem, テウ necessテ。rio usar a biblioteca <termios.h>.
   int getch(void) { return getchar(); }
#endif

#define ARQUIVO_CSV "clientes_oficina.csv"

#define MAX_NOME 100
#define MAX_DATA 30
#define MAX_TELEFONE 15
#define MAX_EMAIL 100
#define MAX_MODELO 30
#define MAX_COR 20
#define MAX_PLACA 8
#define MAX_SERVICO 1000

// Estrutura que representa um cliente
typedef struct ElementoLista {
   char nome[MAX_NOME];
   char dataCadastro[MAX_DATA];
   char telefone[MAX_TELEFONE];
   char email[MAX_EMAIL];
   char modeloCarro[MAX_MODELO];
   char corCarro[MAX_COR];
   char placaCarro[MAX_PLACA];
   char servicoRealizado[MAX_SERVICO];

   struct ElementoLista* proximo;
   struct ElementoLista* anterior;
```

```
} ElementoLista;

// Estrutura que representa a lista duplamente encadeada
typedef struct Lista {
    ElementoLista* inicio;
    ElementoLista* fim;
} Lista;

/* ---------- utilitテ。rios ---------- */

// Limpa a tela (portテ。til)
static void limpa_tela() {
    system(LIMPAR_CMD);
}

// Remove (\n) final de string vinda de fgets
static void remove_newline(char *s) {
    if (!s) return;
    size_t len = strlen(s);
    if (len == 0) return;
    if (s[len-1] == '\n') s[len-1] = '\0';
}

// Leitura segura de linha (fgets + strip)
static void leitura_segura(char *buf, size_t size) {
    if (!fgets(buf, (int)size, stdin)) {
        buf[0] = '\0';
        return;
    }
    remove_newline(buf);
}

// Compararテ｡テ」o case-insensitive portテ。vel
static int compara_ignorando_case(const char *a, const char *b) {
    while (*a && *b) {
        unsigned char ca = (unsigned char) tolower((unsigned char)*a);
        unsigned char cb = (unsigned char) tolower((unsigned char)*b);
        if (ca != cb) return ca - cb;
        a++; b++;
    }
    return (unsigned char) tolower((unsigned char)*a) - (unsigned char) tolower((unsigned
char)*b);
}
```

```
// Escapa aspas para CSV (duplica aspas) e coloca entre aspas
static void grava_campo_csv(FILE *f, const char *s) {
    fputc('"', f); //abre o campo com aspas
    if (s) {
        while (*s) {
            if (*s == '"') fputc('"', f); //garante que o campo comeтгe e termine com "", e qualquer
aspa " contida no texto seja lida como ""
            fputc(*s, f);
            s++;
        }
    }
    fputc('"', f); //fecha o campo com aspas
}

// Parse simples de linha CSV com campos entre aspas e separador ';'
static int parse_csv_line(const char *linha, char campos[][MAX_SERVICO], int max_campos) {
    int idx = 0;
    const char *p = linha;
    while (*p && idx < max_campos) {
        while (*p == ' ' || *p == '\t') p++;
        if (*p == '"') {
            p++; // pula aspas de abertura
            char *out = campos[idx];
            size_t oi = 0;
            while (*p) {
                if (*p == '"') {
                    if (*(p+1) == '"') { // aspas escapada
                        if (oi + 1 < MAX_SERVICO - 1) out[oi++] = '"';
                        p += 2;
                    } else {
                        p++; // fim do campo
                        break;
                    }
                } else {
                    if (oi + 1 < MAX_SERVICO - 1) out[oi++] = *p;
                    p++;
                }
            }
            out[oi] = '\0';
            while (*p && *p != ';') p++;
            if (*p == ';') p++;
            idx++;
```

```
        } else {
            // campo n╤┘o entre aspas (fallback)
            char *out = campos[idx];
            size_t oi = 0;
            while (*p && *p != ';' && *p != '\n' && *p != '\r') {
                if (oi + 1 < MAX_SERVICO - 1) out[oi++] = *p;
                p++;
            }
            out[oi] = '\0';
            if (*p == ';') p++;
            idx++;
        }
    }
    return idx;
}

/* ---------- gerenciamento da lista ---------- */

// Cria lista vazia
Lista* criarLista() {
    Lista* l = (Lista*)malloc(sizeof(Lista));
    if (!l) {
        perror("Erro ao alocar lista");
        exit(EXIT_FAILURE);
    }
    l->inicio = NULL;
    l->fim = NULL;
    return l;
}

// Libera toda a lista
void liberarLista(Lista* lista) {
    if (!lista) return;
    ElementoLista* cur = lista->inicio;
    while (cur) {
        ElementoLista* nxt = cur->proximo;
        free(cur);
        cur = nxt;
    }
    free(lista);
}

// Salva toda a lista em CSV (campo por campo com aspas e separador ';')
```

```
void salvarEmArquivo(Lista* lista) {
    if (!lista) return;
    FILE *f = fopen(ARQUIVO_CSV, "w");
    if (!f) {
        perror("Erro ao abrir arquivo para salvar");
        return;
    }

    // Cabeꟾalho opcional
    fprintf(f,
"\"Nome\";\"DataCadastro\";\"Telefone\";\"Email\";\"ModeloCarro\";\"CorCarro\";\"PlacaCarro\";\"S
ervico\"\n");

    ElementoLista* cur = lista->inicio;
    while (cur) {
        grava_campo_csv(f, cur->nome); fputc(';', f);
        grava_campo_csv(f, cur->dataCadastro); fputc(';', f);
        grava_campo_csv(f, cur->telefone); fputc(';', f);
        grava_campo_csv(f, cur->email); fputc(';', f);
        grava_campo_csv(f, cur->modeloCarro); fputc(';', f);
        grava_campo_csv(f, cur->corCarro); fputc(';', f);
        grava_campo_csv(f, cur->placaCarro); fputc(';', f);
        grava_campo_csv(f, cur->servicoRealizado); fputc('\n', f);
        cur = cur->proximo;
    }

    fclose(f);
}

// Carrega dados do CSV para a lista
void carregarDeArquivo(Lista* lista) {
    if (!lista) return;
    FILE *f = fopen(ARQUIVO_CSV, "r");
    if (!f)
        return;


    char linha[4096];
    int numLinha = 0;
    while (fgets(linha, sizeof(linha), f)) {
        numLinha++;
        // Ignora cabeꟾalho simples
        if (numLinha == 1 && strstr(linha, "Nome") && strstr(linha, "DataCadastro")) continue;
```

```
        char campos[8][MAX_SERVICO];
        for (int i = 0; i < 8; ++i) campos[i][0] = '\0';

        int ncampos = parse_csv_line(linha, campos, 8);
        if (ncampos < 8) continue; // linha invテ。lida -> ignora

        ElementoLista* novo = (ElementoLista*)malloc(sizeof(ElementoLista));
        if (!novo) {
            perror("Erro de memテウria ao carregar arquivo");
            fclose(f);
            return;
        }

        strncpy(novo->nome, campos[0], MAX_NOME-1); novo->nome[MAX_NOME-1] = '\0';
        strncpy(novo->dataCadastro, campos[1], MAX_DATA-1);
novo->dataCadastro[MAX_DATA-1] = '\0';
        strncpy(novo->telefone, campos[2], MAX_TELEFONE-1);
novo->telefone[MAX_TELEFONE-1] = '\0';
        strncpy(novo->email, campos[3], MAX_EMAIL-1); novo->email[MAX_EMAIL-1] = '\0';
        strncpy(novo->modeloCarro, campos[4], MAX_MODELO-1);
novo->modeloCarro[MAX_MODELO-1] = '\0';
        strncpy(novo->corCarro, campos[5], MAX_COR-1); novo->corCarro[MAX_COR-1] = '\0';
        strncpy(novo->placaCarro, campos[6], MAX_PLACA-1);
novo->placaCarro[MAX_PLACA-1] = '\0';
        strncpy(novo->servicoRealizado, campos[7], MAX_SERVICO-1);
novo->servicoRealizado[MAX_SERVICO-1] = '\0';

        //lista encadeada
        novo->proximo = NULL;
        novo->anterior = lista->fim;
        if (lista->fim == NULL) lista->inicio = novo;
        else lista->fim->proximo = novo;
        lista->fim = novo;
    }

    fclose(f);
}

/* Exibe um cliente (uma funテァテ」o auxiliar) */
void exibirCliente(ElementoLista* c) {
    if (!c) return;
    printf("\n--------------------------\n");
```

```
    printf("Nome: %s\n", c->nome);
    printf("Data de Cadastro: %s\n", c->dataCadastro);
    printf("Telefone: %s\n", c->telefone);
    printf("Email: %s\n", c->email);
    printf("Modelo do carro: %s\n", c->modeloCarro);
    printf("Cor do carro: %s\n", c->corCarro);
    printf("Placa do carro: %s\n", c->placaCarro);
    printf("Serviテアo(s) realizado(s): %s\n", c->servicoRealizado);
    printf("--------------------------\n");
}

/* Exibe toda a lista */
void exibirLista(Lista* lista) {
    if (!lista) return;
    if (lista->inicio == NULL) {
        limpa_tela();
        printf("\nLista Vazia!\n");
        return;
    }
    limpa_tela();
    ElementoLista* aux = lista->inicio;
    int cont = 1;
    printf("\n--- Lista de Clientes ---\n");
    while (aux) {
        printf("\nCliente %d:\n", cont++);
        exibirCliente(aux);
        aux = aux->proximo;
    }
    printf("\nQuantidade de contatos: %d\n", cont-1);
    printf("\nPressione Enter para continuar...");
    getchar();
    limpa_tela();
}

/* Inserir novo cliente (salva automaticamente) */
void inserirCliente(Lista* lista) {
    if (!lista) return;
    ElementoLista* novo = (ElementoLista*)malloc(sizeof(ElementoLista));
    if (!novo) {
        perror("Erro ao alocar memテウria");
        return;
    }
```

```
    limpa_tela();
    printf("\n--- Inserir Cliente ---\n");

    printf("Nome: ");
    leitura_segura(novo->nome, sizeof(novo->nome));

    printf("Data de cadastro (DD/MM/AAAA): ");
    leitura_segura(novo->dataCadastro, sizeof(novo->dataCadastro));

    printf("Telefone: ");
    leitura_segura(novo->telefone, sizeof(novo->telefone));

    printf("Email: ");
    leitura_segura(novo->email, sizeof(novo->email));

    printf("Modelo do carro: ");
    leitura_segura(novo->modeloCarro, sizeof(novo->modeloCarro));

    printf("Cor do carro: ");
    leitura_segura(novo->corCarro, sizeof(novo->corCarro));

    printf("Placa do carro: ");
    leitura_segura(novo->placaCarro, sizeof(novo->placaCarro));

    printf("Serviテアo(s) realizado(s): ");
    leitura_segura(novo->servicoRealizado, sizeof(novo->servicoRealizado));

    novo->proximo = NULL;
    novo->anterior = lista->fim;

    if (lista->fim == NULL)
        lista->inicio = novo;
    else
        lista->fim->proximo = novo;
    lista->fim = novo;

    // SALVAMENTO AUTOMテ⊃ICO apテウs inserテアテ」o
    salvarEmArquivo(lista);

    limpa_tela();
    printf("\nContato salvo!\n");
}
```

```
/* Buscar e exibir cliente pelo nome (case-insensitive) */
void buscar_exibir_cliente(Lista* lista) {
    if (!lista) return;
    if (lista->inicio == NULL) {
        limpa_tela();
        printf("\nLista Vazia!\n");
        return;
    }

    char nomeBusca[MAX_NOME];
    printf("Digite o nome do cliente que deseja buscar: ");
    leitura_segura(nomeBusca, sizeof(nomeBusca));

    ElementoLista* aux = lista->inicio;
    while (aux) {
        if (compara_ignorando_case(aux->nome, nomeBusca) == 0) {
            printf("\nCliente encontrado:\n");
            exibirCliente(aux);
            printf("\nPressione Enter para continuar...");
            getchar();
            limpa_tela();
            return;
        }
        aux = aux->proximo;
    }

    printf("\nCliente nao encontrado.\n");
    printf("\nPressione Enter para continuar...");
    getchar();
    limpa_tela();
}

/* Remover um elemento especテュfico da lista (nテ」o salva aqui; o chamador faz o salvamento)
*/
void remover_elemento(Lista* lista, ElementoLista* e) {
    if (!lista || !e) return;

    // Ajusta o ponteiro 'proximo' do elemento anterior
    if (e->anterior) e->anterior->proximo = e->proximo;
    else lista->inicio = e->proximo; // 'e' era o primeiro, o novo primeiro テゥ o prテウximo

    // Ajusta o ponteiro 'anterior' do elemento seguinte
    if (e->proximo) e->proximo->anterior = e->anterior;
```

```
    else lista->fim = e->anterior; // 'e' era o テコltimo, o novo テコltimo テゥ o anterior

    free(e);
}

/* Limpar toda a lista (com confirmaテァテ」o) - salva automaticamente */
void limparTodaLista(Lista* lista) {
    if (!lista) return;
    char resp[8];
    printf("Tem certeza que deseja apagar TODOS os clientes? (S/N): ");
    leitura_segura(resp, sizeof(resp));

    // CORREテ�テグ 1: Adiテァテ」o da chave '}' para fechar corretamente o bloco 'if'
    if (resp[0] != 'S' && resp[0] != 's') {
        printf("Operaテァao cancelada.\n");
        return;
    }

    ElementoLista* cur = lista->inicio;
    while (cur) {
        ElementoLista* nx = cur->proximo;
        free(cur);
        cur = nx;
    }
    lista->inicio = lista->fim = NULL;
    // SALVAMENTO AUTOMテ↘ICO apテウs remoテァテ」o completa
    salvarEmArquivo(lista);
    printf("Lista completamente removida!\n");
}

/* Percorrer lista usando setas (direita/esquerda) e ESC para sair */
void percorrerClientes(Lista* lista) {
    if (!lista || !lista->inicio) {
        limpa_tela();
        printf("\nLista Vazia!\n");
        return;
    }

    ElementoLista* atual = lista->inicio;
    int indice = 1;
    limpa_tela();
    printf("Contato %d:\n", indice);
    exibirCliente(atual);
```

```
    int tecla;
    do {
        printf("\nUse as setas (<- | ->) para navegar ou ESC para sair.\n");
        tecla = getch();
        if (tecla == 0 || tecla == 224) {
            tecla = getch();
            switch (tecla) {
                case 77: // direita
                    if (atual->proximo) {
                        atual = atual->proximo;
                        indice++;
                        limpa_tela();
                        printf("Contato %d:\n", indice);
                        exibirCliente(atual);
                    } else {
                        printf("\nFIM DA LISTA\n");
                    }
                    break;
                case 75: // esquerda
                    if (atual->anterior) {
                        atual = atual->anterior;
                        indice--;
                        limpa_tela();
                        printf("Contato %d:\n", indice);
                        exibirCliente(atual);
                    } else {
                        printf("\nINテ垢IO DA LISTA\n");
                    }
                    break;
            }
        }
    } while (tecla != 27); // enquanto テア apertrem ESC
    limpa_tela();
}

/* Buscar um contato e possibilitar editar ou remover (salvamento automテ。tico apテウs aテァテ」o)
*/
void buscar_editar_ou_remover(Lista* lista) {
    if (!lista || !lista->inicio) {
        limpa_tela();
        printf("\nLista Vazia!\n");
        return;
```

```
    }

    char nomeBusca[MAX_NOME];
    printf("Digite o nome do contato: ");
    leitura_segura(nomeBusca, sizeof(nomeBusca));

    ElementoLista* atual = lista->inicio;
    while (atual) {
        if (compara_ignorando_case(atual->nome, nomeBusca) == 0) {
            printf("\nContato encontrado:\n");
            exibirCliente(atual);

            printf("\nEscolha uma opテアao:\n");
            printf("1. Editar\n");
            printf("2. Remover\n");
            printf("3. Cancelar\n");
            printf("Opテアテ」o: ");

            // CORREテ�テグ 2.1 e 2.2: Declaraテアテ」o correta para uso com leitura_segura (char
array)
            char optbuf_str[8];
            leitura_segura(optbuf_str, sizeof(optbuf_str));
            int opcao = atoi(optbuf_str); // Converte string para int

            if (opcao == 1) {
                printf("\nO que deseja editar?\n");
                printf("1. Nome\n2. Data de Cadastro\n3. Telefone\n4. Email\n5. Modelo do carro\n6.
Cor do carro\n7. Placa do carro\n8. Serviテアo realizado\n9. Tudo\n0. Voltar\n");
                printf("Opテアao: ");

                leitura_segura(optbuf_str, sizeof(optbuf_str));
                int ed = atoi(optbuf_str);

                switch (ed) {
                    case 1: printf("Digite o novo nome: "); leitura_segura(atual->nome,
sizeof(atual->nome)); break;
                    case 2: printf("Digite a nova data: "); leitura_segura(atual->dataCadastro,
sizeof(atual->dataCadastro)); break;
                    case 3: printf("Digite o novo telefone: "); leitura_segura(atual->telefone,
sizeof(atual->telefone)); break;
                    case 4: printf("Digite o novo email: "); leitura_segura(atual->email,
sizeof(atual->email)); break;
                    case 5: printf("Digite o modelo do carro: "); leitura_segura(atual->modeloCarro,
```

```
sizeof(atual->modeloCarro)); break;
                case 6: printf("Digite a cor do carro: "); leitura_segura(atual->corCarro,
sizeof(atual->corCarro)); break;
                case 7: printf("Digite a placa do carro: "); leitura_segura(atual->placaCarro,
sizeof(atual->placaCarro)); break;
                case 8: printf("Digite o(s) servico(s): "); leitura_segura(atual->servicoRealizado,
sizeof(atual->servicoRealizado)); break;
                case 9:
                    printf("Nome: "); leitura_segura(atual->nome, sizeof(atual->nome));
                    printf("Data de cadastro: "); leitura_segura(atual->dataCadastro,
sizeof(atual->dataCadastro));
                    printf("Telefone: "); leitura_segura(atual->telefone, sizeof(atual->telefone));
                    printf("Email: "); leitura_segura(atual->email, sizeof(atual->email));
                    printf("Modelo do carro: "); leitura_segura(atual->modeloCarro,
sizeof(atual->modeloCarro));
                    printf("Cor do carro: "); leitura_segura(atual->corCarro, sizeof(atual->corCarro));
                    printf("Placa: "); leitura_segura(atual->placaCarro, sizeof(atual->placaCarro));
                    printf("Servi??o(s): "); leitura_segura(atual->servicoRealizado,
sizeof(atual->servicoRealizado));
                    break;
                case 0:
                    printf("Edicao cancelada.\n");
                    return;
                default:
                    printf("Opcao invalida!\n");
            }
            // SALVAMENTO AUTOM?TICO ap?s edi??o
            salvarEmArquivo(lista);
            printf("\nContato atualizado com sucesso!\n");
            return;
        } else if (opcao == 2) {
            char resp[8];
            printf("Tem certeza que deseja remover este contato? (S/N): ");
            leitura_segura(resp, sizeof(resp));

            // CORRE?? 2.3: Corre??o da compara??o l?gica para 'S' ou 's'
            if (resp[0] == 'S' || resp[0] == 's') {
                remover_elemento(lista, atual);
                // SALVAMENTO AUTOM?TICO ap?s remo??o
                salvarEmArquivo(lista);
                printf("\nRemovido com sucesso!\n");
            } else {
                printf("Remocao cancelada.\n");
```

```
            }
            return;
        } else {
            printf("Operacao cancelada.\n");
            return;
        }
    }
    atual = atual->proximo;
  }
  printf("\nContato nao encontrado.\n");
}

/* ---------- main ---------- */

int main() {
  setlocale(LC_ALL, "portuguese");

  Lista* lista = criarLista();
  carregarDeArquivo(lista);

  char optbuf[16];
  int opcao = -1;

  for (;;) {
    printf("\n\n-----------------------------------\n");
    printf("         Oficina ESPARTANOS!\n");
    printf("\n\n-----------------------------------\n");
    printf("Menu:\n");
    printf("1. Exibir lista de clientes\n");
    printf("2. Inserir cliente\n");
    printf("3. Buscar e exibir cliente\n");
    printf("4. Remover lista completa de clientes\n");
    printf("5. Percorrer lista de clientes (setas)\n");
    printf("6. Editar ou Remover cliente (buscar)\n");
    printf("0. Sair\n");
    printf("Escolha uma opcao: ");

    leitura_segura(optbuf, sizeof(optbuf));
    opcao = atoi(optbuf);

    switch (opcao) {
      case 1: exibirLista(lista); break;
      case 2: inserirCliente(lista); break;
```

```
            case 3: buscar_exibir_cliente(lista); break;
            case 4: limpa_tela(); limparTodaLista(lista); break;
            case 5: percorrerClientes(lista); break;
            case 6: buscar_editar_ou_remover(lista); break;
            case 0:
                printf("Saindo...\n");
                // garante salvar ao sair
                salvarEmArquivo(lista);
                liberarLista(lista);
                return 0;
            default:
                printf("Opテァテ」o invテ。lida! Tente novamente.\n");
                break;
        }
    }

    return 0;
}
```