



ECOLE NATIONAL SUPERIEUR DES MINES DE RABAT

DEPARTEMENT INFORMATIQUE

Structure de données

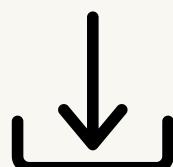
Gestion de contact téléphonique en language C

REALISE PAR:

**ELOUAQQAD MOHAMMED
OUSSAMA ELIDRYSY
HAJJAJI AHMED YASSINE
EZ-ZAOUI NASR-EDDINE**

SOMMAIRE

INTRODUCTION	3
LE TRAVAIL EFFECTUÉ	4
SDDS UTILISÉES	6
ALGORITHMES DES FONCTIONS PRINCIPALES	8
CODE SOURCE	11
JEUX D'ESSAI	19
VALEUR AJOUTÉE	22
CONCLUSION	23



1. INTRODUCTION

1.1 Contexte du projet

Ce projet consiste en la conception et le développement d'un gestionnaire de contacts en langage C. L'objectif principal est de créer une application capable de gérer efficacement une base de données de contacts personnels, permettant l'ajout, la modification, la suppression, la recherche et la sauvegarde des informations de contact.

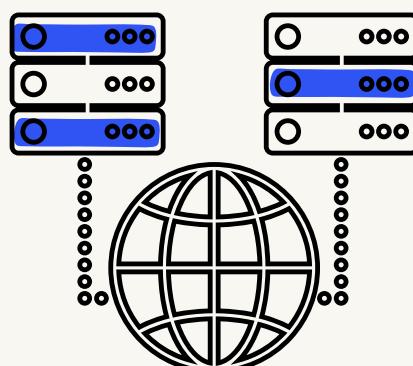
1.2 Objectifs du projet

- Développer une interface utilisateur intuitive en mode console
- Implémenter une structure de données efficace pour le stockage des contacts
- Permettre la gestion complète des contacts (CRUD : Create, Read, Update, Delete)
- Offrir des fonctionnalités de recherche avancées
- Assurer la persistance des données via la sauvegarde en fichier

1.3 Cahier des charges

Le système doit permettre de :

- Ajouter de nouveaux contacts avec nom, téléphone et email
- Supprimer des contacts existants
- Modifier les informations d'un contact
- Afficher tous les contacts dans l'ordre alphabétique
- Rechercher des contacts par différents critères
- Sauvegarder et charger les données depuis un fichier



2. DESCRIPTION DU TRAVAIL DE NOTRE ÉQUIPE SUR LE PROJET

Phase de conception théorique



Avant de commencer la programmation, nous avons pris le temps de faire une étude théorique approfondie du projet. Cette phase nous a permis de bien comprendre les besoins et les fonctionnalités à implémenter. En équipe, nous avons discuté et défini :

- Les structures de données nécessaires (notamment une structure Compte contenant les informations comme le nom, prénom, identifiant, solde, etc.).
- Les fonctions principales à développer, telles que : ajouter un compte, modifier un compte, rechercher un compte (par la première lettre, les deux premières lettres, ou le nom complet), afficher les comptes, sauvegarder et charger les données depuis un fichier.
- La méthode d'implémentation, en optant pour une gestion via des listes chaînées afin de permettre une manipulation dynamique des comptes.

Cette préparation nous a permis d'aborder le codage avec une vision claire et organisée.

Phase de développement individuel en équipe



Une fois la conception achevée, nous avons réparti les tâches entre les membres de l'équipe afin de travailler en parallèle sur différentes fonctionnalités du projet. Chaque membre s'est vu attribuer une mission précise :

- Un membre s'est chargé de l'implémentation de l'ajout et de la suppression des comptes.
- Un autre a développé les fonctions de recherche avec les différents critères (lettres et nom complet).
- Un autre a travaillé sur la sauvegarde et le chargement des comptes depuis un fichier.
- Et enfin, un autre a assuré la création de l'interface console et l'interaction avec l'utilisateur.

Cette méthode de travail nous a permis de gagner du temps tout en responsabilisant chaque membre de l'équipe.

Phase d'intégration, de test et d'amélioration

Une fois chaque partie du code terminée, nous avons procédé à une intégration progressive des différents modules. Cette étape a nécessité :

- La fusion du code en un seul fichier principal cohérent.
- L'exécution de nombreux tests pour identifier les erreurs (bugs) et les incompatibilités.
- Des corrections collaboratives afin de résoudre les problèmes détectés.
- Des améliorations de la clarté du code, de l'ergonomie de l'interface console, et de l'optimisation des fonctions.

Ce travail d'intégration et de validation a été essentiel pour garantir la stabilité et l'efficacité de notre programme.



3. SDDS UTILISÉES

Les SSD utilisées :

* Contact :

nom1	tel1	Mail1	→
------	------	-------	---

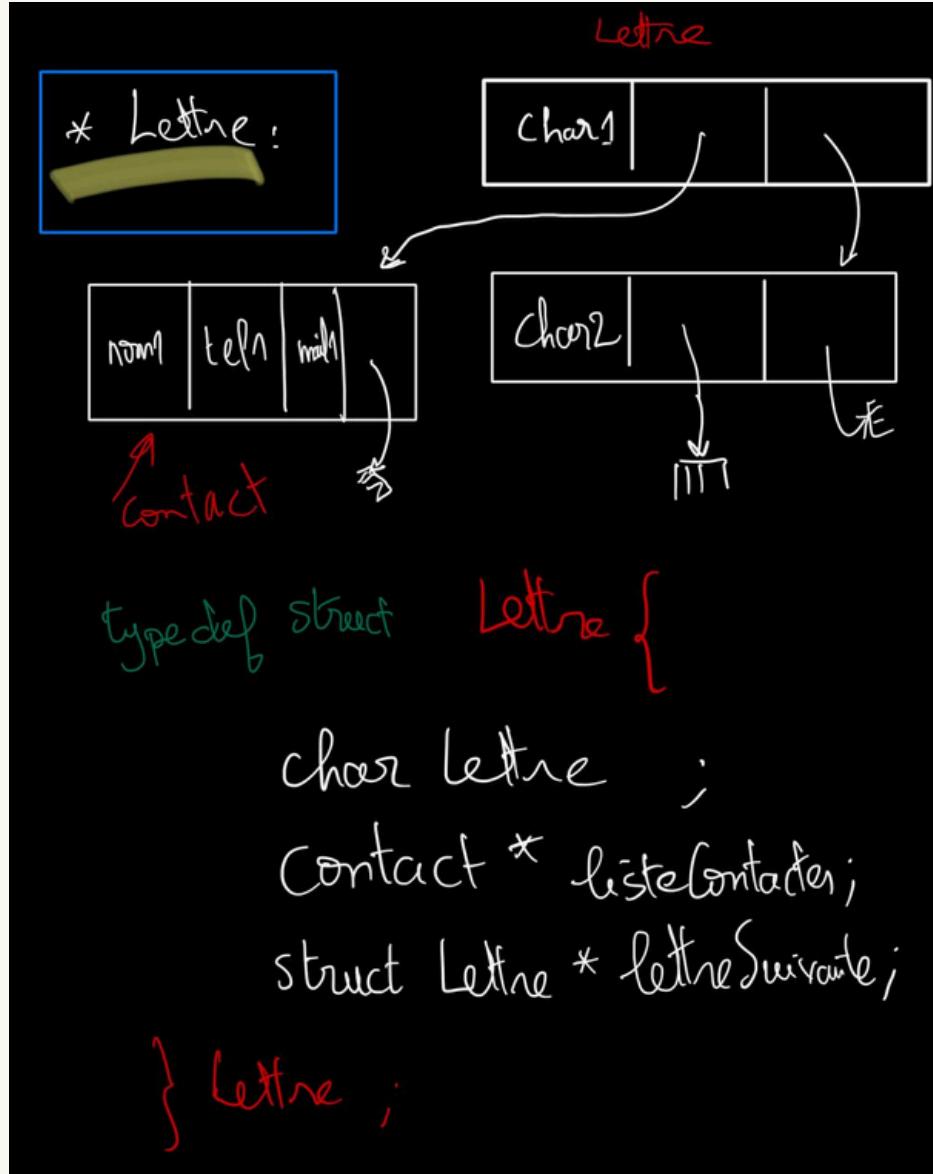
nom2	tel2	Mail2	↓
------	------	-------	---

```
type def struct Contact {  
    char nom[20];  
    char tel [20];  
    char mail [20];  
    struct Contact* suivant;  
} Contact;
```

✓ Structure Contact :

La structure Contact permet de stocker les informations d'un contact. Elle contient :

- **char nom[20]** : le nom du contact,
- **char tel[20]** : le numéro de téléphone,
- **char mail[20]** : l'adresse e-mail,
- **struct Contact* suivant** : un pointeur vers le contact suivant, ce qui permet de créer une liste chaînée de contacts.



✓ Structure Lettre :

La structure Lettre permet de regrouper les contacts par la première lettre de leur nom. Elle contient :

- **char lettre :** la lettre associée à ce groupe de contacts (exemple : 'A', 'B', etc.),
- **Contact* listeContact :** un pointeur vers la liste des contacts dont le nom commence par cette lettre,
- **struct Lettre* lettreSuivante :** un pointeur vers la lettre suivante, ce qui permet de créer une liste chaînée de lettres.

Ces deux structures sont utilisées ensemble pour organiser un répertoire de contacts par ordre alphabétique.

4. ALGORITHMES DES FONCTIONS PRINCIPALES

1 ajouterContact()

un ajout de contact par nom numero et email

```
2 // Fonction pour ajouter un contact (création de la lettre si besoin)
3 void ajouterContact(Lettre* listeLettres, const char* nom, const char* tel, const char* mail) {
4     Si (nom == NULL || nom[0] == '\0') alors
5         Ecrire("Nom invalide.\n");
6         return;
7     Finsi;
8     char premiereLettre = toupper(nom[0]);
9     Lettre *precLettre = NULL, *courantLettre = *listeLettres;
10    // Recherche de la lettre ou de la position d'insertion
11    Tantque (courantLettre && courantLettre->lettre < premiereLettre) faire
12        precLettre = courantLettre;
13        courantLettre = courantLettre->lettreSuivante;
14    FTQ
15    // Cas 1 : La lettre n'existe pas, il faut la créer et l'insérer
16    Si (!courantLettre || courantLettre->lettre != premiereLettre) alors
17        Lettre* nouvelleLettre = malloc(sizeof(Lettre));
18        Simon
19            Si (nouvelleLettre == NULL) alors
20                printf("Erreur d'allocation memoire.\n");
21                exit(1);
22            Finsi
23        nouvelleLettre->lettre = premiereLettre;
24        nouvelleLettre->listesContacts = NULL;
25        nouvelleLettre->lettreSuivante = courantLettre;
26    Si (precLettre == NULL) alors
27        // Insertion en tête
28        *listeLettres = nouvelleLettre;
29    Sinon
30        precLettre->lettreSuivante = nouvelleLettre;
31    Finsi
32    courantLettre = nouvelleLettre;
33 }
34
35
36
37
38 }
```

```
40 // Ajout du contact dans la liste de la lettre (ordre alphabétique)
41 Contact *prec = NULL, *curr = courantLettre->listesContacts;
42 Tantque (curr && comparer_noms_sans_casse(nom, curr->nom) > 0) faire
43     prec = curr;
44     curr = curr->suivant;
45 FTQ
46 Si (curr && comparer_noms_sans_casse(nom, curr->nom) == 0)
47     Ecrire("Un contact avec ce nom existe déjà.\n");
48     return;
49 Finsi
50 Contact* nouveau = creerContact(nom, tel, mail);
51 Si (prec == NULL) alors
52     nouveau->suivant = courantLettre->listesContacts;
53     courantLettre->listesContacts = nouveau;
54 Sinon
55     prec->suivant = nouveau;
56     nouveau->suivant = curr;
57 Finsi
58 Ecrire("Contact ajouté avec succès.\n");
59 }
60 }
```

2 modifierContact()

fonction de modification d'un contact en utilisant un input de NOM

```
208 // Fonction pour modifier un contact (modifie tel et/ou mail)
209 void modifierContact(Lettre* listeLettres, const char* nom, const char* nouveauTel, const char* nouveauMail) {
210     Si (nom == NULL || nom[0] == '\0') alors
211         printf("Nom invalide.\n");
212         return;
213     Finsi;
214     char premiereLettre = toupper(nom[0]);
215     Lettre* l = listeLettres;
216     // Recherche de la lettre correspondante
217     Tantque (l && l->lettre < premiereLettre) Faire
218         l = l->lettreSuivante;
219     FTQ
220     Si (!l || l->lettre != premiereLettre) alors
221         printf("Aucun contact trouvé avec ce nom.\n");
222         return;
223     Finsi;
224     Contact* c = l->listesContacts;
225     Tantque (c) Faire
226         Si (comparer_noms_sans_casse(nom, c->nom) == 0) alors
227             Si (nouveauTel && nouveauTel[0] != '\0') alors
228                 strcpy(c->tel, nouveauTel, sizeof(c->tel)-1);
229                 c->tel[sizeof(c->tel)-1] = '\0';
230             Finsi
231             Si (nouveauMail && nouveauMail[0] != '\0') alors
232                 strcpy(c->mail, nouveauMail, sizeof(c->mail)-1);
233                 c->mail[sizeof(c->mail)-1] = '\0';
234             Finsi
235             Ecrire("Contact modifié avec succès.\n");
236             return;
237         Finsi
238         c = c->suivant;
239     FTQ
240     Ecrire("Aucun contact trouvé avec ce nom.\n");
241 }
```

3 supprimer contact()

fonction supprimer qui sert à la suppression d'un seul contact

```
1 // Fonction pour supprimer un contact par nom
2 void supprimerContact(Lettre** listeLettres, const char* nom) {
3     Si (nom == NULL || nom[0] == '\0') alors
4         printf("Nom invalide.\n");
5         return;
6     Finsi
7     char premiereLettre = toupper(nom[0]);
8     Lettre *preLettre = NULL, *courantLettre = *listeLettres;
9
10    // Recherche de la lettre correspondante
11    Tantque (courantLettre && courantLettre->lettre < premiereLettre) Faire
12        preLettre = courantLettre;
13        courantLettre = courantLettre->lettreSuivante;
14    FTQ
15    Si (!courantLettre || courantLettre->lettre != premiereLettre) alors
16        printf("Aucun contact trouve avec ce nom.\n");
17        return;
18    Finsi
19
20    // Recherche du contact à supprimer
21    Contact *prec = NULL, *curr = courantLettre->listeContacts;
22    Tantque (curr && comparer_noms_sans_casse(nom, curr->nom) != 0) faire
23        prec = curr;
24        curr = curr->suivant;
25    FTQ
26    Si (!curr) alors
27        printf("Aucun contact trouve avec ce nom.\n");
28        return;
29    Finsi
30
31    // Suppression du contact
32    Si (prec == NULL) alors
33        courantLettre->listeContacts = curr->suivant;
34    Sinon
35        prec->suivant = curr->suivant;
36    Finsi
37    free(curr);
38    printf("Contact supprime avec succes.\n");
39
40    // Si la lettre n'a plus de contacts, la supprimer aussi
41    Si (courantLettre->listeContacts == NULL) alors
42        Si (preLettre == NULL) alors
43            *listeLettres = courantLettre->lettreSuivante;
44        Sinon
45            preLettre->lettreSuivante = courantLettre->lettreSuivante;
46        Finsi
47        free(courantLettre);
48    Finsi
49 }
```

4 sauvegarderContacts()

```
244 // Fonction pour sauvegarder tous les contacts dans un fichier texte
245 void sauvegarderContacts(const Lettre* listeLettres, const char* nomFichier) {
246     FILE* f = fopen(nomFichier, "w");
247     Si (!f) alors
248         Ecrire("Erreur lors de l'ouverture du fichier pour la sauvegarde.\n");
249         return;
250     }
251     const Lettre* l = listeLettres;
252     Tantque (l) Faire
253         const Contact* c = l->listeContacts;
254         Tantque (c) Faire
255             fprintf(f, "%s;%s;%s\n", c->nom, c->tel, c->mail);
256             c = c->suivant;
257         FTQ
258         l = l->lettreSuivante;
259     FTQ
260     fclose(f);
261     Ecrire("Contacts sauvegardes dans le fichier '%s'.\n", nomFichier);
262 }
```

5 rechargerContacts()

fonction qui recharge tous les contacts existant dans un fichier et qui écrase tous les contacts existant

```
279 // Fonction pour recharger les contacts depuis un fichier texte
280 void rechargerContacts(Lettre** listeLettres, const char* nomFichier) {
281     libererContacts(listeLettres);
282     FILE* f = fopen(nomFichier, "r");
283     Si (!f) alors
284         Ecrire("Erreur lors de l'ouverture du fichier pour le rechargement.\n");
285         return;
286     Finsi
287     char ligne[256];
288     Tantque (fgets(ligne, sizeof(ligne), f)) alors
289         char nom[100], tel[20], mail[100];
290         // Suppression du saut de ligne
291         ligne[strcspn(ligne, "\r\n")] = 0;
292         Si (sscanf(ligne, "%99[^:]:%19[^:]:%99[^:]\n", nom, tel, mail) == 3) alors
293             ajouterContact(listeLettres, nom, tel, mail);
294         Finsi
295     FTQ
296     fclose(f);
297     Ecrire("Contacts rechargés depuis le fichier '%s'.\n", nomFichier);
298 }
299
300 }
```

6 AFFICHAGE DES CONTACTS

fonction d'affichage de tous les contacts déjà ajouté

```
111 // Fonction pour afficher tous les contacts par ordre alphabétique
112 void afficherContacts(const Lettre* listeLettres) {
113     int vide = 1;
114     const Lettre* l = listeLettres;
115     Tantque (l) Faire
116         Si (l->listeContacts) alors
117             vide = 0;
118             break;
119         Finsi
120         l = l->lettreSuivante;
121     FTQ
122     Si (vide) alors
123         Ecrire("Aucun contact à afficher.\n");
124         return;
125     Finsi
126     Ecrire("Nom           | Tel           | Mail\n");
127     Ecrire("-----+-----+-----\n");
128     l = listeLettres;
129     Tantque (l) Faire
130         const Contact* c = l->listeContacts;
131         Tantque (c) Faire
132             printf("%-13s | %-13s | %-30s\n", c->nom, c->tel, c->mail);
133             c = c->suivant;
134         FTQ
135         l = l->lettreSuivante;
136     FTQ
137 }
138 }
```

5. DESCRIPTION DU TRAVAIL DE NOTRE ÉQUIPE SUR LE PROJET

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 // Définition de la structure Contact
7 typedef struct Contact {
8     char nom[100];
9     char tel[20];
10    char mail[100];
11    struct Contact* suivant;
12 } Contact;
13
14 // Définition de la structure Lettre
15 typedef struct Lettre {
16     char lettre;
17     Contact* listeContacts;
18     struct Lettre* lettreSuivante;
19 } Lettre;
20
21 // Fonction pour créer un nouveau contact
22 Contact* creerContact(const char* nom, const char* tel, const char* mail) {
23     Contact* c = malloc(sizeof(Contact));
24     if (c == NULL) {
25         printf("Erreur d'allocation mémoire.\n");
26         exit(1);
27     }
28     strcpy(c->nom, nom);
29     strcpy(c->tel, tel);
30     strcpy(c->mail, mail);
31     c->suivant = NULL;
32     return c;
33 }
34
35 // Fonction utilitaire pour comparer deux chaînes sans tenir compte de la casse
36 int comparer_noms_sans_casse(const char* nom1, const char* nom2) {
37     char buf1[100], buf2[100];
38     int i;
39     for (i = 0; nom1[i] && i < 99; i++) buf1[i] = tolower((unsigned char)nom1[i]);
40     buf1[i] = '\0';
41     for (i = 0; nom2[i] && i < 99; i++) buf2[i] = tolower((unsigned char)nom2[i]);
42     buf2[i] = '\0';
43     return strcmp(buf1, buf2);
44 }
45
46 // Fonction pour ajouter un contact (création de la lettre si besoin)
47 void ajouterContact(Lettre** listeLettres, const char* nom, const char* tel, const char* mail) {
48     if (nom == NULL || nom[0] == '\0') {
49         printf("Nom invalide.\n");
50         return;
51     }
52     char premiereLettre = toupper(nom[0]);
53     Lettre *precLettre = NULL, *courantLettre = *listeLettres;
```

```

55 // Recherche de la lettre ou de la position d'insertion
56 while (courantLettre && courantLettre->lettre < premiereLettre) {
57     precLettre = courantLettre;
58     courantLettre = courantLettre->lettreSuivante;
59 }
60
61 // Cas 1 : La lettre n'existe pas, il faut la créer et l'insérer
62 if (!courantLettre || courantLettre->lettre != premiereLettre) {
63     Lettre* nouvelleLettre = malloc(sizeof(Lettre));
64     if (nouvelleLettre == NULL) {
65         printf("Erreur d'allocation memoire.\n");
66         exit(1);
67     }
68     nouvelleLettre->lettre = premiereLettre;
69     nouvelleLettre->listeContacts = NULL;
70     nouvelleLettre->lettreSuivante = courantLettre;
71
72     if (precLettre == NULL) {
73         // Insertion en tête
74         *listesLettres = nouvelleLettre;
75     } else {
76         precLettre->lettreSuivante = nouvelleLettre;
77     }
78     courantLettre = nouvelleLettre;
79 }
80
81 // Ajout du contact dans la liste de la lettre (ordre alphabétique)
82 Contact *prec = NULL, *curr = courantLettre->listeContacts;
83 while (curr && comparer_noms_sans_casse(nom, curr->nom) > 0) {
84     prec = curr;
85     curr = curr->suivant;
86 }
87 if (curr && comparer_noms_sans_casse(nom, curr->nom) == 0) {
88     printf("Un contact avec ce nom existe déjà.\n");
89     return;
90 }
91 Contact* nouveau = creerContact(nom, tel, mail);
92 if (prec == NULL) {
93     nouveau->suivant = courantLettre->listeContacts;
94     courantLettre->listeContacts = nouveau;
95 } else {
96     prec->suivant = nouveau;
97     nouveau->suivant = curr;
98 }
99 printf("Contact ajouté avec succès.\n");
00 }

```

```

102 // Fonction pour supprimer un contact par nom
103 void supprimerContact(Lettre** listeLettres, const char* nom) {
104     if (nom == NULL || nom[0] == '\0') {
105         printf("Nom invalide.\n");
106         return;
107     }
108     char premiereLettre = toupper(nom[0]);
109     Lettre *precLettre = NULL, *courantLettre = *listeLettres;
110
111     // Recherche de la lettre correspondante
112     while (courantLettre && courantLettre->lettre < premiereLettre) {
113         precLettre = courantLettre;
114         courantLettre = courantLettre->lettreSuivante;
115     }
116     if (!courantLettre || courantLettre->lettre != premiereLettre) {
117         printf("Aucun contact trouve avec ce nom.\n");
118         return;
119     }
120
121     // Recherche du contact à supprimer
122     Contact *prec = NULL, *curr = courantLettre->listeContacts;
123     while (curr && comparer_noms_sans_casse(nom, curr->nom) != 0) {
124         prec = curr;
125         curr = curr->suivant;
126     }
127     if (!curr) {
128         printf("Aucun contact trouve avec ce nom.\n");
129         return;
130     }
131
132     // Suppression du contact
133     if (prec == NULL) {
134         courantLettre->listeContacts = curr->suivant;
135     } else {
136         prec->suivant = curr->suivant;
137     }
138     free(curr);
139     printf("Contact supprime avec succes.\n");
140
141     // Si la lettre n'a plus de contacts, la supprimer aussi
142     if (courantLettre->listeContacts == NULL) {
143         if (precLettre == NULL) {
144             *listeLettres = courantLettre->lettreSuivante;
145         } else {
146             precLettre->lettreSuivante = courantLettre->lettreSuivante;
147         }
148         free(courantLettre);
149     }
150 }
```

```

151 // Fonction pour afficher tous les contacts par ordre alphabétique
152 void afficherContacts(const Lettre* listeLettres) {
153     int vide = 1;
154     const Lettre* l = listeLettres;
155     while (l) {
156         if ([l->listeContacts]) {
157             vide = 0;
158             break;
159         }
160         l = l->lettreSuivante;
161     }
162     if (vide) {
163         printf("Aucun contact à afficher.\n");
164         return;
165     }
166     printf("Nom           | Tel           | Mail\n");
167     printf("-----+-----+-----\n");
168     l = listeLettres;
169     while (l) {
170         const Contact* c = l->listeContacts;
171         while (c) {
172             printf("%-13s | %-13s | %-30s\n", c->nom, c->tel, c->mail);
173             c = c->suivant;
174         }
175         l = l->lettreSuivante;
176     }
177 }
178
179 // Fonction pour chercher des contacts selon différents critères
180 void chercherContact(const Lettre* listeLettres, const char* critere) {
181     if (critere == NULL || critere[0] == '\0') {
182         printf("Critère de recherche invalide.\n");
183         return;
184     }
185     int critereLen = strlen(critere);
186     int trouve = 0;
187     if (critereLen == 1) {
188         // Recherche par première lettre
189         char lettreRecherchée = toupper(critere[0]);
190         const Lettre* l = listeLettres;
191         while (l && l->lettre < lettreRecherchée) l = l->lettreSuivante;
192         if (l && l->lettre == lettreRecherchée) {
193             const Contact* c = l->listeContacts;
194             while (c) {
195                 if (!trouve) {
196                     printf("Nom           | Tel           | Mail\n");
197                     printf("-----+-----+-----\n");
198                 }
199                 printf("%-13s | %-13s | %-30s\n", c->nom, c->tel, c->mail);
200                 trouve = 1;
201                 c = c->suivant;
202             }
203         }
204     } else if (critereLen == 2) {

```

```

205 } else if (critereLen == 2) {
206     // Recherche par premiere et deuxieme lettre
207     char lettreRecherchee = toupper(critere[0]);
208     char deuxiemeLettre = tolower(critere[1]);
209     const Lettre* l = listeLettres;
210     while (l && l->lettre < lettreRecherchee) l = l->lettreSuivante;
211     if (l && l->lettre == lettreRecherchee) {
212         const Contact* c = l->listeContacts;
213         while (c) {
214             if (tolower((unsigned char)c->nom[1]) == deuxiemeLettre) {
215                 if (!trouve) {
216                     printf("Nom           | Tel           | Mail\n");
217                     printf("-----+-----+-----\n");
218                 }
219                 printf("%-13s | %-13s | %-30s\n", c->nom, c->tel, c->mail);
220                 trouve = 1;
221             }
222             c = c->suivant;
223         }
224     }
225 } else {
226     // Recherche par nom complet (insensible a la casse)
227     const Lettre* l = listeLettres;
228     while (l) {
229         const Contact* c = l->listeContacts;
230         while (c) {
231             if (comparer_noms_sans_casse(critere, c->nom) == 0) {
232                 if (!trouve) {
233                     printf("Nom           | Tel           | Mail\n");
234                     printf("-----+-----+-----\n");
235                 }
236                 printf("%-13s | %-13s | %-30s\n", c->nom, c->tel, c->mail);
237                 trouve = 1;
238             }
239             c = c->suivant;
240         }
241         l = l->lettreSuivante;
242     }
243 }
244 if (!trouve) {
245     printf("Aucun contact trouve pour le critere donne.\n");
246 }
247 }

// Fonction pour modifier un contact (modifie tel et/ou mail)
249 void modifierContact(Lettre* listeLettres, const char* nom, const char* nouveauTel, const char* nouveauMail) {
250     if (nom == NULL || nom[0] == '\0') {
251         printf("Nom invalide.\n");
252         return;
253     }
254     char premiereLettre = toupper(nom[0]);
255     Lettre* l = listeLettres;
256     // Recherche de la lettre correspondante
257     while (l && l->lettre < premiereLettre) l = l->lettreSuivante;
258     if (!l || l->lettre != premiereLettre) {

```

```

259     if (!l || l->lettre != premiereLettre) {
260         printf("Aucun contact trouve avec ce nom.\n");
261         return;
262     }
263     Contact* c = l->listeContacts;
264     while (c) {
265         if (comparer_noms_sans_casse(nom, c->nom) == 0) {
266             if (nouveauTel && nouveauTel[0] != '\0') {
267                 strncpy(c->tel, nouveauTel, sizeof(c->tel)-1);
268                 c->tel[sizeof(c->tel)-1] = '\0';
269             }
270             if (nouveauMail && nouveauMail[0] != '\0') {
271                 strncpy(c->mail, nouveauMail, sizeof(c->mail)-1);
272                 c->mail[sizeof(c->mail)-1] = '\0';
273             }
274             printf("Contact modifie avec succes.\n");
275             return;
276         }
277         c = c->suivant;
278     }
279     printf("Aucun contact trouve avec ce nom.\n");
280 }
281
282 // Fonction pour sauvegarder tous les contacts dans un fichier texte
283 void sauvegarderContacts(const Lettre* listeLettres, const char* nomFichier) {
284     FILE* f = fopen(nomFichier, "w");
285     if (!f) {
286         printf("Erreur lors de l'ouverture du fichier pour la sauvegarde.\n");
287         return;
288     }
289     const Lettre* l = listeLettres;
290     while (l) {
291         const Contact* c = l->listeContacts;
292         while (c) {
293             fprintf(f, "%s;%s;%s\n", c->nom, c->tel, c->mail);
294             c = c->suivant;
295         }
296         l = l->lettreSuivante;
297     }
298     fclose(f);
299     printf("Contacts sauvegardes dans le fichier '%s'.\n", nomFichier);
300 }
301
302 // Fonction pour libérer toute la mémoire de la liste des contacts
303 void libererContacts(Lettre** listeLettres) {
304     Lettre* l = *listeLettres;
305     while (l) {
306         Contact* c = l->listeContacts;
307         while (c) {
308             Contact* tmpC = c;
309             c = c->suivant;
310             free(tmpC);
311         }
312         Lettre* tmpL = l;
313         l = l->lettreSuivante;

```

```

314         free(tmpL);
315     }
316     *listeLettres = NULL;
317 }
318
319 // Fonction pour recharger les contacts depuis un fichier texte
320 void rechargerContacts(Lettre** listeLettres, const char* nomFichier) {
321     libererContacts(listeLettres);
322     FILE* f = fopen(nomFichier, "r");
323     if (!f) {
324         printf("Erreur lors de l'ouverture du fichier pour le rechargement.\n");
325         return;
326     }
327     char ligne[256];
328     while (fgets(ligne, sizeof(ligne), f)) {
329         char nom[100], tel[20], mail[100];
330         // Suppression du saut de ligne
331         ligne[strcspn(ligne, "\r\n")] = 0;
332         if (sscanf(ligne, "%99[^;];%19[^;];%99[^;]", nom, tel, mail) == 3) {
333             ajouterContact(listeLettres, nom, tel, mail);
334         }
335     }
336     fclose(f);
337     printf("Contacts recharges depuis le fichier '%s'.\n", nomFichier);
338 }
339
340 int main() {
341     Lettre* listeLettres = NULL;
342     int choix;
343     char nom[100], tel[20], mail[100], critere[100], fichier[128];
344
345     do {
346         printf("\n--- Gestionnaire de contacts ---\n");
347         printf("1. Ajouter un contact\n");
348         printf("2. Supprimer un contact\n");
349         printf("3. Modifier un contact\n");
350         printf("4. Afficher tous les contacts\n");
351         printf("5. Chercher un contact\n");
352         printf("6. Sauvegarder les contacts dans un fichier\n");
353         printf("7. Recharger les contacts depuis un fichier\n");
354         printf("0. Quitter\n");
355         printf("Votre choix : ");
356         if (scanf("%d", &choix) != 1) {
357             while (getchar() != '\n'); // vider le buffer
358             choix = -1;
359         }
360         getchar(); // consommer le \n
361
362         switch (choix) {
363             case 1:
364                 printf("Nom : ");
365                 fgets(nom, sizeof(nom), stdin);
366                 nom[strcspn(nom, "\r\n")] = 0;
367                 printf("telephone : ");

```

```

7     printf("telephone : ");
8     fgets(tel, sizeof(tel), stdin);
9     tel[strcspn(tel, "\r\n")] = 0;
0     printf("Email : ");
1     fgets(mail, sizeof(mail), stdin);
2     mail[strcspn(mail, "\r\n")] = 0;
3     ajouterContact(&listeLettres, nom, tel, mail);
4     break;
5 case 2:
6     printf("Nom du contact a supprimer : ");
7     fgets(nom, sizeof(nom), stdin);
8     nom[strcspn(nom, "\r\n")] = 0;
9     supprimerContact(&listeLettres, nom);
0     break;
1 case 3:
2     printf("Nom du contact a modifier : ");
3     fflush(stdin);
4     fgets(nom, sizeof(nom), stdin);
5     nom[strcspn(nom, "\r\n")] = 0;
6     printf("Nouveau telephone (laisser vide pour ne pas changer) : ");
7     fflush(stdin);
8     fgets(tel, sizeof(tel), stdin);
9     tel[strcspn(tel, "\r\n")] = 0;
0     printf("Nouvel email (laisser vide pour ne pas changer) : ");
1     fflush(stdin);
2     fgets(mail, sizeof(mail), stdin);
3     mail[strcspn(mail, "\r\n")] = 0;
4     modifierContact(listeLettres, nom, tel, mail);
5     break;
6 case 4:
7     afficherContacts(listeLettres);
8     break;
9 case 5:
0     printf("Critere de recherche (nom complet, 1ere lettre ou 1ere+2eme lettre) : ");
1     fgets(critere, sizeof(critere), stdin);
2     critere[strcspn(critere, "\r\n")] = 0;
3     chercherContact(listeLettres, critere);
4     break;
5 case 6:
6     printf("Nom du fichier pour sauvegarder : ");
7     fgets(fichier, sizeof(fichier), stdin);
8     fichier[strcspn(fichier, "\r\n")] = 0;
9     sauvegarderContacts(listeLettres, fichier);
0     break;
1 case 7:
2     printf("Nom du fichier a recharger : ");
3     fgets(fichier, sizeof(fichier), stdin);
4     fichier[strcspn(fichier, "\r\n")] = 0;
5     rechargerContacts(&listeLettres, fichier);
6     break;
7 case 0:
8     printf("Au revoir !\n");
9     break;
0 default:
1     printf("Choix invalide.\n");

```

```

423         if (choix != 0) {
424             char rep[8];
425             printf("Voulez-vous revenir au menu ? (o/n) : ");
426             fgets(rep, sizeof(rep), stdin);
427             if (tolower(rep[0]) != 'o') break;
428         }
429     } while (choix != 0);

431     libererContacts(&listeLettres);
432     return 0;
433 }

```

6. JEUX D'ESSAI

```
--- Gestionnaire de contacts ---
1. Ajouter un contact
2. Supprimer un contact
3. Modifier un contact
4. Afficher tous les contacts
5. Chercher un contact
6. Sauvegarder les contacts dans un fichier
7. Recharger les contacts depuis un fichier
0. Quitter
Votre choix : |
```



Maintenant, nous allons simuler les actions de l'utilisateur pour chaque option du menu afin de vérifier que le programme réagit correctement et affiche les résultats attendus.

```
--- Gestionnaire de contacts ---
1. Ajouter un contact
2. Supprimer un contact
3. Modifier un contact
4. Afficher tous les contacts
5. Chercher un contact
6. Sauvegarder les contacts dans un fichier
7. Recharger les contacts depuis un fichier
0. Quitter
Votre choix : 4
Nom           | Tel          | Mail
-----+-----+-----+
ahmed        | 0563672632   | ahmed@enim
aymane       | 06327622     | aymane@enim
mohammed     | 056757362    | mohammed@enim
nassr         | 065675372    | nassr@enim
oussama      | 0667382738   | oussama@enim
Voulez-vous revenir au menu ? (o/n) :
```

Afficher tous les contacts

modifier un contact

Votre choix : 3

Nom du contact a modifier : ahmed

Nouveau telephone (laisser vide pour ne pas changer) :

Nouvel email (laisser vide pour ne pas changer) : Ahmed1@enim

Contact modifie avec succes.

Voulez-vous revenir au menu ? (o/n) :

supprimer un contact

Votre choix : 2

Nom du contact a supprimer : ahmed

Contact supprime avec succes.

Voulez-vous revenir au menu ? (o/n) :

recherche par premiere letter

Votre choix : 5

Criterie de recherche (nom complet, 1ere lettre ou 1ere+2eme lettre) : a

Nom | Tel | Mail

-----	-----	-----
ahmed	0563672632	ahmed@enim
aymane	06327622	aymane@enim

Voulez-vous revenir au menu ? (o/n) :

recherche par premiere et 2 eme lettre

Votre choix : 5

Criterie de recherche (nom complet, 1ere lettre ou 1ere+2eme lettre) : na

Nom | Tel | Mail

-----	-----	-----
nassr	065675372	nassr@enim

Voulez-vous revenir au menu ? (o/n) :

recherche par nom complet

Votre choix : 5

Criterie de recherche (nom complet, 1ere lettre ou 1ere+2eme lettre) : oussama

Nom | Tel | Mail

-----	-----	-----
oussama	0667382738	oussama@enim

Voulez-vous revenir au menu ? (o/n) :

sauvgarder fichier

Votre choix : 6

Nom du fichier pour sauvegarder : fichier 1

Contacts sauvegardes dans le fichier 'fichier 1'.

Voulez-vous revenir au menu ? (o/n) :

ajout d'un contact

```
0. Quitter
Votre choix : 1
Nom : amine
telephone : 07645545467
Email : amine@enim
Contact ajoute avec succes.
Voulez-vous revenir au menu ? (o/n) :
```

fichier a recharger

```
ayoub;056363222;ayoub@enim
anas;06327622232;anas@enim
mina;0567573012;mina@enim
sara;0656753073;sara@enim
omar;066738405;omar@enim
```

recharge et affichage du fichier

```
Votre choix : 7
Nom du fichier a recharger : fichier 2
Contact ajoute avec succes.
Contacts recharges depuis le fichier 'fichier 2'.
Voulez-vous revenir au menu ? (o/n) : o

--- Gestionnaire de contacts ---
1. Ajouter un contact
2. Supprimer un contact
3. Modifier un contact
4. Afficher tous les contacts
5. Chercher un contact
6. Sauvegarder les contacts dans un fichier
7. Recharger les contacts depuis un fichier
0. Quitter
Votre choix : 4
Nom | Tel | Mail
-----+-----+-----
ahmed | 0563672632 | ahmed@enim
aymane | 06327622 | aymane@enim
mohammed | 056757362 | mohammed@enim
nassr | 065675372 | nassr@enim
oussama | 0667382738 | oussama@enim
Voulez-vous revenir au menu ? (o/n) : |
```

7.LES ATOUTS UNIQUES DE NOTRE CODE

1

Copier quelques options que l'on trouve dans le système d'exploitation iOS (iPhone)

L'ajout de contacts a un critère d'unicité de nom.

Si l'on ajoute un nom qui existe déjà, un message s'affiche pour nous informer qu'un contact précédent existe avec ce nom.

3

L'affichage des contacts

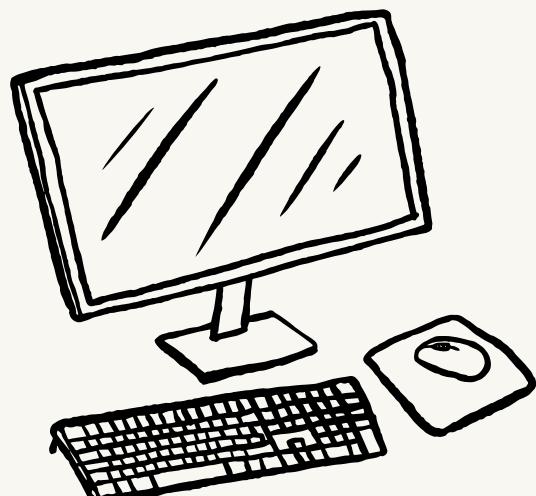
Lors de l'affichage de nos contacts, on aura toujours la première colonne (NOM; TEL; EMAIL:)

Ensuite viennent les contacts, un par un.

2

La fonction de rechargement en mémoire

Notre fonction de rechargement en mémoire écrase les anciens contacts et ajoute les nouveaux contacts partir du fichier rechargeé.



8. CONCLUSION

Ce projet de gestion de contacts en C a été une expérience extrêmement enrichissante, tant sur le plan technique que personnel. Il nous a offert une première vision concrète du développement logiciel, en nous permettant d'appliquer les concepts théoriques du langage C à un projet réel et fonctionnel. Grâce à ce travail, nous avons consolidé nos compétences en programmation (gestion de mémoire, structures de données, manipulation de fichiers, etc.), mais aussi développé des soft skills essentiels :

- Collaboration et travail d'équipe : Répartition des tâches, coordination et résolution collective des problèmes.
- Gestion du temps et organisation : Respect des deadlines et adaptation aux imprévus.
- Résolution de problèmes : Débogage, optimisation et recherche de solutions techniques.
- Communication : Présentation claire du code et des fonctionnalités.

En somme, ce projet a été une étape clé dans notre apprentissage, renforçant à la fois notre maîtrise du C et notre capacité à travailler efficacement en équipe. Nous sommes fiers du résultat obtenu et motivés à relever de nouveaux défis technologiques !

Un grand merci à notre professeur pour son accompagnement tout au long de ce projet.

