Mini-projet 3

Jean Simard

Jeudi 6 novembre 2008 – Vendredi 7 novembre 2008

1 Rendre le TP

Vous commencerez par commenter votre programme dans un fichier `README`. Vous n'oublierez pas d'indiquer votre nom dans ce fichier et tous les détails que vous jugerez nécessaires.

Après avoir mis vos fichiers dans un répertoire nommé `Simard_Jean` (vous remplacerez par votre nom et votre prénom), vous compresserez le répertoire dans un fichier `TP12-Simard_Jean.tar.bz2` (remplacez également par votre nom et votre prénom).

```
tar -cvjf TP12-Simard_Jean.tar.bz2 ./Simard_Jean/
```

Ensuite, envoyez ce fichier dans un mail ayant pour sujet (en remplaçant les noms et prénoms).

```
[CCI] TP12 Simard Jean
```

Tâchez de respecter précisément le format des noms de fichiers, de répertoires et de mail.

2 Rappel sur les fichiers

Pour écrire dans un fichier, nous allons utiliser le type `FILE`. Pour être plus précis, un simple pointeur sur ce type sera suffisant.

Afin d'ouvrir un fichier ou de créer ce fichier le cas échéant, nous allons utiliser la fonction `fopen`. Cette fonction prend en argument le nom du fichier puis une chaîne de caractère indiquant la manière dont le fichier est ouvert : en écriture ou en lecture. D'autres options sont disponibles mais ne seront pas traitées ici. La fonction renvoie un pointeur sur un `FILE`. Par exemple, pour ouvrir un fichier `TP12.txt` en lecture (on ne veut pas le modifier), on écrira

```
FILE * pFichier = NULL;
pFichier = fopen( "TP12.txt", "r" );
```

Ensuite, on va pouvoir fermer le fichier à l'aide de la fonction `close`. Cette fonction prend simplement le pointeur sur un `FILE` en entrée.

```
fclose( pFichier );
```

Entre le moment où le fichier s'ouvre et le moment où il se ferme, nous allons pouvoir effectuer des opérations de lectures ou des opérations d'écriture en fonction du mode d'ouverture du fichier.

Pour écrire dans un fichier, nous utiliserons la variante `fprintf` de la fonction `printf`. Cette fonction va écrire dans un fichier au lieu d'afficher à l'écran. Elle prend un argument supplémentaire qui sera le pointeur de fichier.

```
int i = 7;
fprintf( pFichier, "i = %d\n", i );
```

Nous venons donc d'écrire `i = 7` dans le fichier en ajoutant un passage à la ligne à la fin de la ligne.

Imaginons maintenant que nous désirions lire ce même fichier. Nous utiliserions alors une variante de la fonction `scanf` qui est la fonction `fscanf`. De la même façon que la fonction `fprintf`, cette fonction prend simplement un argument supplémentaire pour indiquer le fichier.

```
fscanf( pFichier, "i = %d\n", &i );
```

Ici, au lieu de lire les caractères entrés au clavier comme dans la fonction `scanf`, on lit les caractères dans le fichier.

3 Carnet d'adresses de courriel

Nous allons ici créer un programme permettant la gestion d'un carnet d'adresses de courriel. Pour la création d'un tel programme, plusieurs fonctions peuvent être utiles. Je vais donc proposer plusieurs fonctions possibles.

3.1 La structure

Nous allons travailler dans cet exercice avec des listes simplement chaînées. L'ensemble des contacts seront contenus dans cette liste simplement chaînée.

Exercice 1 *Écrire une structure pour qu'elle puisse contenir trois chaînes de caractères : le nom, le prénom et l'adresse de courriel. On pourra utiliser une chaîne de caractères de grande taille temporaire pour lire les noms, prénoms et adresses de courriel (comme déjà vu dans les TP précédents). Cette structure devra constituer les maillons de notre liste simplement chaînée.*

Correction de l'exercice 1 –

Code 1: Contacts.c

```
5  typedef struct sContact
6  {
7    char * cNom;
8    char * cPrenom;
9    char * cCourriel;
10   struct sContact * pSuiv;
11 } Contact;
```

3.2 Le menu

Pour ce programme, nous avons avoir besoin d'un menu déroulant pour choisir les différentes opérations qu'on va pouvoir effectuer.

Exercice 2 *Écrire la fonction* `main` *de telle manière qu'elle permette d'afficher un menu pour l'utilisateur. Ce menu affichera les différentes opérations proposées par le programme. Il faudra une structure* `switch` *qui appellera les différentes opérations possibles en fonction du choix de l'utilisateur. Après chaque opération, il faudra que le menu s'affiche de nouveau.*

Correction de l'exercice 2 –

CODE 2: Contacts.c

```
267 int main( int argc, char ** argv )
268 {
269   Contact * DebutCarnet = NULL;
270   char cChoix = '\0';
271   do
272   {
273     printf( "----- Menu -----\n" );
274     printf( "c - Creer un contact\n" );
275     printf( "m - Modifier un contact\n" );
276     printf( "s - Supprimer un contact\n" );
277     printf( "a - Afficher les contacts\n" );
278     printf( "e - Enregistrer le carnet d'adresses\n" );
279     printf( "i - Importer un carnet d'adresses\n" );
280     printf( "q - Quitter\n" );
281     printf( "Choix : " );
282     scanf( "%c", &cChoix );
283     switch( cChoix )
284     {
285       case 'c' :
286       case 'C' :
287         DebutCarnet = AjouterContact( DebutCarnet );
288         break;
289       case 'm' :
290       case 'M' :
291         DebutCarnet = ModifierContact( DebutCarnet );
292         break;
293       case 's' :
294       case 'S' :
295         DebutCarnet = SupprimerContact( DebutCarnet );
296         break;
297       case 'a' :
298       case 'A' :
299         AffichageContacts( DebutCarnet );
300         break;
301       case 'e' :
```

```
302         case 'E' :
303           EnregistrerFichier( DebutCarnet );
304           break;
305         case 'i' :
306         case 'I' :
307           DebutCarnet = ImporterFichier();
308           break;
309         case 'q' :
310         case 'Q' :
311           LibererCarnet( DebutCarnet );
312           return 0;
313         default :
314           printf( "Choix incorrect\n" );
315           break;
316       }
317       getchar();
318     } while( cChoix != 'q' );
319
320     return 0;
321 }
```

3.3 Création d'un contact

Nous pouvons tout d'abord écrire une fonction permettant de créer un nouveau contact. En effet, cette fonction va permettre de remplir notre liste chaînée.

Exercice 3 *Écrire une fonction afin de créer un nouveau contact et de l'ajouter dans la liste chaînée. Cette fonction devra demander à l'utilisateur un nom, un prénom puis une adresse de courriel correspondante. À partir de ces informations, on ajoutera ce nouveau contact dans la liste chaînée (au début, à la fin ou à l'endroit qui vous arrange). Il faudra donc utiliser la fonction* `malloc` *pour allouer l'espace mémoire nécessaire à la création d'un nouveau contact.*

Correction de l'exercice 3 –

Code 3: Contacts.c

```
32 Contact * AjouterContact( Contact * pDebut )
33 {
34   Contact * pNouveau = NULL;
35   Contact * pTemp = pDebut;
36   char cTemp[256];
37   pNouveau = (Contact *) malloc( sizeof( Contact ) );
38   printf( "----- Creation -----\n" );
39   printf( "Entrer votre nom : " );
40   scanf( "%s", cTemp );
41   pNouveau->cNom = (char *) malloc(\
42     ( strlen( cTemp ) + 1 ) * sizeof( char ) );
```

```
43    strcpy( pNouveau->cNom, cTemp );
44    printf( "Entrer votre prenom : " );
45    scanf( "%s", cTemp );
46    pNouveau->cPrenom = (char *) malloc(\
47      ( strlen( cTemp ) + 1 ) * sizeof( char ) );
48    strcpy( pNouveau->cPrenom, cTemp );
49    printf( "Entrer votre courriel : " );
50    scanf( "%s", cTemp );
51    pNouveau->cCourriel = (char *) malloc(\
52      ( strlen( cTemp ) + 1 ) * sizeof( char ) );
53    strcpy( pNouveau->cCourriel, cTemp );
54    pNouveau->pSuiv = NULL;
55    if( pDebut == NULL )
56    {
57      return pNouveau;
58    }
59    while( pTemp->pSuiv != NULL )
60    {
61      pTemp = pTemp->pSuiv;
62    }
63    pTemp->pSuiv = pNouveau;
64    return pDebut;
65  }
```

3.4 Libérer l'espace mémoire

Nous créons une liste simplement chaînée en créant de nouveaux contacts au-fur-et-à-mesure que l'utilisateur les enregistre. À la fin du programme, nous allons donc devoir libérer cette mémoire.

Exercice 4 *Écrire une fonction qui libère la mémoire de la liste chaînée. Pour cela, il faudra écrire une fonction supplémentaire pour libérer l'espace mémoire d'un seul maillon de la liste. Il faudra donc utiliser la fonction* `free` *afin de libérer cette mémoire. Attention, il ne faudra pas oublier de libérer les chaînes de caractères dans chaque maillon avant de libérer la mémoire d'un maillon.*

Correction de l'exercice 4 –

Code 4: Contacts.c

```
13  void LibererContact( Contact * pContact )
14  {
15    if( pContact != NULL )
16    {
17      free( pContact->cNom );
18      free( pContact->cPrenom );
19      free( pContact->cCourriel );
20      free( pContact );
```

```
21    }
22    return;
23  }
```

Code 5: Contacts.c

```
192 void LibererCarnet( Contact * pDebut )
193 {
194   Contact * pTemp = NULL;
195   if( pDebut == NULL )
196   {
197     return;
198   }
199   while( pDebut->pSuiv != NULL )
200   {
201     pTemp = pDebut;
202     pDebut = pDebut->pSuiv;
203     LibererContact( pTemp );
204   }
205   return;
206 }
```

3.5 Afficher les contacts

Nous sommes en train de créer un carnet d'adresses de courriel. Il peut donc être utile d'afficher les noms et les adresses correspondantes.

Exercice 5 *Écrire une fonction qui affiche tous les contacts présents dans le carnet d'adresses.*

Correction de l'exercice 5 –

Code 6: Contacts.c

```
181 void AffichageContacts( Contact * c )
182 {
183   printf( "----- Affichage -----\n" );
184   while( c != NULL )
185   {
186     AffichageContact( c );
187     c = c->pSuiv;
188   }
189   return;
190 }
```

3.6 Enregistrer le carnet d'adresses

Dans ce programme, nous sommes capables de créer une liste chaînée au sein du programme. Mais lorsque nous allons quitter le programme, la liste n'existera plus. C'est pour cette raison que nous devons pouvoir enregistrer le carnet d'adresses dans un fichier.

Exercice 6 *Écrire une fonction qui enregistre les noms, prénoms et adresses de courriel dans un fichier. La fonction demandera à l'utilisateur de donner le nom du fichier. Pour le format du fichier, on écrira le nom, le prénom et l'adresse de courriel sur trois lignes distinctes. Une ligne vide sera écrite pour séparer deux contacts dans le fichier.*

Correction de l'exercice 6 –

Code 7: Contacts.c

```
208 void EnregistrerFichier( Contact * pDebut )
209 {
210   FILE * pFichier = NULL;
211   char cNomFichier[256];
212   printf( "----- Enregistrer -----\n" );
213   printf( "Entrer le nom du fichier : " );
214   scanf( "%s", cNomFichier );
215   pFichier = fopen( cNomFichier, "w" );
216   while( pDebut != NULL )
217   {
218     fprintf( pFichier, "%s\n", pDebut->cNom );
219     fprintf( pFichier, "%s\n", pDebut->cPrenom );
220     fprintf( pFichier, "%s\n", pDebut->cCourriel );
221     fprintf( pFichier, "\n" );
222     pDebut = pDebut->pSuiv;
223   }
224   return;
225 }
```

3.7 Charger un carnet d'adresses

Si nous sommes capables d'enregistrer un carnet d'adresses dans un fichier, nous devrions également pouvoir lire un fichier pour créer un carnet d'adresses.

Exercice 7 *Écrire une fonction qui chargera les noms, prénoms et adresses de courriel depuis un fichier. La fonction demandera à l'utilisateur de donner le nom du fichier. Cette fonction devra créer la liste chaînée contenant toutes les informations présentes dans le fichier.*

Correction de l'exercice 7 –

Code 8: Contacts.c

```
227 Contact * ImporterFichier()
228 {
229   FILE * pFichier = NULL;
230   Contact * pDebut = NULL;
231   Contact * pTemp = NULL;
232   char cNomFichier[256];
233   char cTemp[256];
234   printf( "----- Importer -----\n" );
235   printf( "Entrer le nom du fichier : " );
236   scanf( "%s", cNomFichier );
237   pFichier = fopen( cNomFichier, "r" );
238   if( pFichier == NULL )
239   {
240     printf( "Erreur de lecture du fichier\n" );
241     return NULL;
242   }
243   while( fscanf( pFichier, "%s\n", cTemp ) != EOF )
244   {
245     pTemp = (Contact *) malloc( sizeof( Contact ) );
246     pTemp->cNom = (char *) malloc(\
247       strlen( cTemp ) * sizeof( char ) );
248     strcpy( pTemp->cNom, cTemp );
249     if( fscanf( pFichier, "%s\n", cTemp ) != EOF )
250     {
251       pTemp->cPrenom = (char *) malloc(\
252         strlen( cTemp ) * sizeof( char ) );
253       strcpy( pTemp->cPrenom, cTemp );
254     }
255     if( fscanf( pFichier, "%s\n\n", cTemp ) != EOF )
256     {
257       pTemp->cCourriel = (char *) malloc(\
258         strlen( cTemp ) * sizeof( char ) );
259       strcpy( pTemp->cCourriel, cTemp );
260     }
261     pTemp->pSuiv = pDebut;
262     pDebut = pTemp;
263   }
264   return pDebut;
265 }
```

3.8 Suppression d'un contact

Dans ce carnet, nous devrions pouvoir supprimer un contact. Pour effectuer cette opération, vous allez devoir utiliser la fonction `strcmp` de la bibliothèque `string`. Cette fonction va vous permettre de comparer deux chaînes de caractères. La fonction prend deux arguments qui seront les deux chaînes de caractères à comparer et renvoie la valeur zéro si les deux chaînes sont identiques.

Exercice 8 *Écrire une fonction qui permette de supprimer une contact du carnet d'adresses. Cette fonction devra demander à l'utilisateur le nom du contact à supprimer. Il est possible que plusieurs contacts correspondent au nom dans le carnet. C'est pour cette raison que le programme devra proposer de confirmer ou non la suppression pour chaque contact dont le nom correspond.*

Correction de l'exercice 8 –

CODE 9: Contacts.c

```
67  Contact * SupprimerContact( Contact * pDebut )
68  {
69    Contact * pTemp = pDebut;
70    Contact * pPrec = NULL;
71    char cNom[256];
72    char cChoix = '\0';
73    printf( "----- Suppression -----\n" );
74    printf( "Entrer le nom a supprimer : " );
75    scanf( "%s", cNom );
76    while( pTemp != NULL )
77    {
78      if( strcmp( pTemp->cNom, cNom ) == 0 )
79      {
80        printf( "----- Suppression de %s %s (%s) -----\n",\
81          pTemp->cPrenom, pTemp->cNom, pTemp->cCourriel );
82        printf( "o - Oui\n" );
83        printf( "n - Non\n" );
84        printf( "Choix : " );
85        getchar();
86        scanf( "%c", &cChoix );
87        if( cChoix == 'o' )
88        {
89          if( pPrec != NULL )
90          {
91            pPrec->pSuiv = pTemp->pSuiv;
92            LibererContact( pTemp );
93            pTemp = pPrec->pSuiv;
94          }
95          else
96          {
97            pDebut = pTemp->pSuiv;
98            LibererContact( pTemp );
99            pTemp = pDebut;
100         }
101       }
102     }
103     else
104     {
105       pPrec = pTemp;
106       pTemp = pTemp->pSuiv;
```

```
107       }
108     }
109     return pDebut;
110 }
```

3.9 Modification d'un contact

Dans le cas où l'utilisateur effectue une erreur de saisie ou modifie l'adresse de courriel d'un contact, il doit pouvoir modifier un contact.

Exercice 9 *Écrire une fonction qui permette de modifier un contact. Cette fonction devra demander à l'utilisateur le nom du contact à modifier. Il est possible que plusieurs contacts correspondent au nom dans le carnet. C'est pour cette raison que le programme devra proposer de modifier ou non chaque contact dont le nom correspond.*

Lorsque le programme propose de modifier un contact, il lui propose de modifier soit le nom, soit le prénom, soit l'adresse de courriel. Il doit donc également proposer de ne pas modifier le contact pour le cas où plusieurs contacts correspondent au nom donné par l'utilisateur.

Correction de l'exercice 9 –

Code 10: Contacts.c

```
112 Contact * ModifierContact( Contact * pDebut )
113 {
114   Contact * pTemp = pDebut;
115   char cChoix = '\0';
116   char cNom[256];
117   char cChaine[256];
118   printf( "----- Modification -----\n" );
119   printf( "Entrer le nom a modifier : " );
120   scanf( "%s", cNom );
121   while( pTemp != NULL )
122   {
123     if( strcmp( pTemp->cNom, cNom ) == 0 )
124     {
125       do
126       {
127         printf( "----- Modification de %s %s (%s) -----\n",\
128           pTemp->cPrenom, pTemp->cNom, pTemp->cCourriel );
129         printf( "n - Modifier le nom\n" );
130         printf( "p - Modifier le prenom\n" );
131         printf( "c - Modifier l'adresse de courriel\n" );
132         printf( "s - Ne pas modifier\n" );
133         printf( "Choix : " );
134         getchar();
135         scanf( "%c", &cChoix );
```

```
136           switch( cChoix )
137           {
138             case 'n' :
139             case 'N' :
140               printf( "Entrer le nouveau nom : " );
141               scanf( "%s", cChaine );
142               free( pTemp->cNom );
143               pTemp->cNom = (char *) malloc(\
144                 strlen( cChaine ) * sizeof( char ) );
145               strcpy( pTemp->cNom, cChaine );
146               break;
147             case 'p' :
148             case 'P' :
149               printf( "Entrer le nouveau prenom : " );
150               scanf( "%s", cChaine );
151               free( pTemp->cPrenom );
152               pTemp->cPrenom = (char *) malloc(\
153                 strlen( cChaine ) * sizeof( char ) );
154               strcpy( pTemp->cPrenom, cChaine );
155               break;
156             case 'c' :
157             case 'C' :
158               printf( "Entrer la nouvelle adresse de courriel : " );
159               scanf( "%s", cChaine );
160               free( pTemp->cCourriel );
161               pTemp->cCourriel = (char *) malloc(\
162                 strlen( cChaine ) * sizeof( char ) );
163               strcpy( pTemp->cCourriel, cChaine );
164               break;
165             case 's' :
166             case 'S' :
167               printf( "Pas de modification sur ce contact\n" );
168               break;
169             default :
170               printf( "Choix incorrect\n" );
171               cChoix = 'r';
172               break;
173           }
174         } while( cChoix == 'r' );
175       }
176       pTemp = pTemp->pSuiv;
177     }
178     return pDebut;
179   }
```