

Documentation Mini Projet Langage C

Caculli Giorgio LA196672, Tyranowski Jędrzej LA196937
Haute École de Louvain en Hainaut (HELHa)

15 décembre 2020

Résumé

Documentation pour le projet de Langage procédural sur les listes chaînées. Projet basé sur le concept d'un centre de formations.

Mots-clés : liste, chaînée, c, noeud, centre, formation, tête

Table des matières

1	Introduction	3
1.1	Le langage C	3
1.2	Fonctions générales utilisées	3
1.2.1	Qu'est-ce l'allocation dynamique de mémoire?	6
1.2.2	Qu'est-ce <code>calloc()</code> ?	6
1.2.3	Qu'est-ce <code>malloc()</code> ?	6
1.2.4	Différences entre <code>calloc()</code> et <code>malloc()</code>	6
2	Listes chaînées	7
2.1	Qu'est-ce une liste chaînée?	7
2.2	Création d'un nouveau nœud	7
2.3	Insertion d'un nœud dans une liste chaînée	7
2.4	Suppression d'un nœud d'une liste chaînée	8
2.5	Affichage d'une liste chaînée	10
3	Énoncé	11
4	Programme	12
4.1	Mode d'emploi	12
4.1.1	Menu principale	12
4.1.2	Menu affichage	12
4.1.3	Liste des personnes	13
4.1.4	Liste des formations	13
4.1.5	Planning de la semaine	14
4.1.6	Menu ajout	15
4.1.7	Ajout d'une personne	15
4.1.8	Ajout d'une formation	17
4.1.9	Menu attribution	18
4.1.10	Menu suppression	20
4.1.11	Suppression d'une personne	20
4.1.12	Suppression d'une formation	21
4.1.13	Suppression d'une personne d'une formation spécifique	22
5	Code	23
5.1	Structures	23
5.2	Fonctions	25
5.2.1	Fonctions qui créent des nœuds	25
5.2.2	Fonctions qui affichent des listes chaînées	25
5.2.3	Fonctions qui ajoutent un nœud à une liste chaînée	25
5.2.4	Fonctions qui suppriment un nœud d'une liste chaînée	26
5.2.5	Fonctions qui servent de <code>getter()</code>	27
5.2.6	Fonctions qui affiches les différentes parties du menu interactif	27
5.2.7	Entièreté du code	28

1 Introduction

1.1 Le langage C

La langage de programmation utilisé lors du développement et la mise en œuvre du programme est le ANSI-C. Les différentes versions du langage disponibles lors du développement de ce programme sont :

- **ANSI-C** : La première version standardisée par le **American National Standard Institute**, abrégé en **ANSI** dans ce document, du langage **C** publiée en 1990.
- **C-99** : Révision de la version ANSI pour permettre aux développeurs d'utiliser les commentaires `//`, les booléens grâce à la librairie `<stdbool.h>`, la déclaration des `int` directement dans la boucle `for`, et d'autres modernisations de la syntaxe.
- **C-11** : Mise à jour du langage **C** pour permettre le support des `thread` afin de pouvoir faire du multi-threading.
- **C-17** : Révision de la version **C-11** qui n'ajoute aucune nouvelle fonctionnalité, mais corrige beaucoup bugs présents dans la version 11.

Dans les différentes applications que l'on a fait dans le cours de Langage procédural, la plupart des interactions que l'on a eu avec le langage **C**, notamment le fait de devoir déclarer un `int` avant une boucle `for`, ressemblaient fortement à l'ANSI-C. C'est pourquoi nous avons choisi d'utiliser cette version là.

1.2 Fonctions générales utilisées

Différentes fonctions ont été utilisées lors du développement de ce programme.

Voici une liste des fonctions clés utilisées :

- `fopen()` : Fonction qui sert à ouvrir un flux, la plupart du temps un fichier.

Exemple :

```
1 /* Admettons que le fichier donnees.dat existe */
2 #include <stdio.h>
3 int main() {
4     FILE *fichier_in = fopen( "donnees.dat", "r" );
5     /* On ouvre le fichier donnees.dat en lecture */
6     FILE *fichier_out = fopen( "resultats.txt", "w" );
7     /* Si le fichier resultats.txt n'existe pas on le cree, s'il existe toute information
       presente est ecrasee, puis on y accede en ecrute */
8     return 0;
9 }
```

- `fclose()` : Fonction qui sert à fermer tout flux ouvert.

Exemple :

```
1 /* Admettons que le fichier donnees.dat existe */
2 #include <stdio.h>
3 int main() {
4     FILE *fichier_in = fopen( "donnees.dat", "r" );
5     /* On ouvre le fichier donnees.dat en lecture */
6     FILE *fichier_out = fopen( "resultats.txt", "w" );
7     /* Si le fichier resultats.txt n'existe pas on le cree, s'il existe toute information
       presente est ecrasee, puis on y accede en ecrute */
8     fclose( fichier_out );
9     fclose( fichier_in );
10    /* On ferme les fichiers lorsqu'on doit plus travailler avec eux */
11    return 0;
12 }
```

- `printf()` : Fonction qui sert à afficher une chaîne de caractères dans la console.

Exemple :

```
1 #include <stdio.h>
2 int main() {
3     printf( "Hello World!\n" ); /* Affiche "Hello World!" dans la console */
4     return 0;
5 }
```

- `fprintf()` : Fonction qui sert à écrire une chaîne de caractères vers un flux spécifique.

Exemple :

```

1 #include <stdio.h>
2 int main() {
3     File *fichier_sortie = fopen( "fichier_sortie.txt", "w" );
4     /* On cree et on ouvre un fichier nomme fichier_sortie.txt en ecriture */
5     fprintf( fichier_sortie, "Hello World!\n" );
6     /* Ecrit "Hello World!" dans le fichier fichier_sortie.txt mais pas dans la console */
7     fprintf( stdout, "Hello World!\n" );
8     /* Affiche "Hello World!" dans la console mais pas dans le fichier de sortie */
9     return 0;
10 }

```

- **scanf()** : Fonction qui sert à extrapoler une entrée du clavier et stocker les informations extrapolées dans les paramètres déclarés dans la fonction.

Exemple :

```

1 #include <stdio.h>
2 int main() {
3     char prenom[50];
4     int age;
5     printf( "Comment t'appelles-tu ? " );
6     scanf( "%s", prenom ); /* Admettons que l'utilisateur insere "Jedrzej" */
7     printf( "Quel age as-tu %s ? ", prenom );
8     scanf( "%d", &age ); /* Admettons que l'utilisateur insere "21" */
9     printf( "Salut %s, je vois que tu as %d ans!\n", prenom, age );
10    /* Affiche "Salut Jedrzej, je vois que tu as 21 ans!" dans la console */
11    return 0;
12 }

```

- **fscanf()** : Fonction qui sert à extrapoler des entrées à partir d'un flux spécifique en respectant une structure précise, et stocker les informations extrapolées dans des paramètres déclarés dans la fonction.

Exemple :

```

1 /* Contenu dans fichier_entree.txt */
2 /* Jedrzej 21 */
3 #include <stdio.h>
4 int main() {
5     char prenom[50];
6     int age;
7     FILE *fichier_in = fopen( "fichier_entree.txt", "r" );
8     /* On ouvre un fichier nomme fichier_entree.txt en lecture */
9     fscanf( fichier_in, "%s %d\n", prenom, &age );
10    /* On lit le contenu de fichier_entree.txt */
11    printf( "Salut %s, je vois que tu as %d ans\n", prenom, age );
12    /* Affiche "Salut Jedrzej, je vois que tu as 21 ans!" dans la console */
13    return 0;
14 }

```

- **fgets()** : Fonction qui a le même principe que fscanf, mais garde les espaces.

Exemple :

```

1 /* Contenu dans fichier_entree.txt */
2 /* Caculli Giorgio 23 */
3 #include <stdio.h>
4 int main() {
5     char nom_prenom[15];
6     int age;
7     FILE *fichier_in = fopen( "fichier_entree.txt", "r" );
8     fgets( nom_prenom, 16, fichier_in );
9     /* Lit les 15 caracteres depuis le fichier de donnees fichier_entree.txt */
10    fscanf( fichier_in, "%d", &age );
11    /* Lit l'age depuis le fichier de donnees fichier_entree.txt */
12    printf( "%s %d\n", nom_prenom, age );
13    /* Affiche "Caculli Giorgio 23" dans la console */
14    return 0;
15 }

```

- **strcpy()** : Fonction qui sert à copier une chaîne de caractères vers une autres chaîne de caractères.

Exemple :

```

1 #include <stdio.h>
2 #include <string.h>
3 int main() {
4     char prenom[10];

```

```

5  char source[10] = "Giorgio";
6  strcpy( prenom, source );
7  printf( "%s\n", prenom ); /* Affiche "Giorgio" dans la console */
8  return 0;
9 }

```

- **sizeof()** : Fonction qui renvoie la quantité de mémoire (en bytes) qu'une entité va occuper dans la Random Access Memory, ou mémoire vive en français, abrégé en RAM dans ce document.

Exemple :

```

1 #include <stdio.h>
2 int main() {
3     printf( "Taille de char: %lu\n", sizeof( char ) );
4     /* Affiche 1 dans la console */
5     printf( "Taille de int: %lu\n", sizeof( int ) );
6     /* Affiche 4 dans la console */
7     printf( "Taille de long: %lu\n", sizeof( long ) );
8     /* Affiche 8 dans la console */
9     printf( "Taille de int[4]: %lu\n", sizeof( int[4] ) );
10    /* Affiche 16 dans la console, soit 4*4=16 */
11    return 0;
12 }

```

- **memcpy()** : Fonction qui copie n octets depuis une zone mémoire vers une autre zone mémoire.

Exemple :

```

1 #include <stdio.h>
2 #include <string.h>
3 int main() {
4     int src[] = {1, 2, 3};
5     size_t n = sizeof(src) / sizeof(src[0])
6     /* En sachant que la taille d'un int = 4 bytes, et qu'il y a 3 int dans src, on
       obtient 4 * 3 = 12 bytes utilisés. On divise la taille totale du vecteur, soit 12,
       par la taille du premier element du vecteur, soit 4, donc 12 / 4 = 3 elements dans
       le vecteur */
7     int dest[n]; /* On initialise le vecteur dest avec la meme taille de src, ici 3 */
8     memcpy( dest, src, sizeof(dest) ); /* On copie les informations de src vers dest */
9     int i;
10    for( i = 0; i < n; i++ ) {
11        /* On parcourt tous les elements dans dest, du premier (indice 0) jusqu'au troisieme
           (indice 2) */
12        printf( "%d ", dest[i] ); /* Affiche 1 2 3 dans la console */
13    }
14    return 0;
15 }

```

- **getchar()** : Fonction qui sert à lire un caractère depuis un flux.

Exemple :

```

1 #include <stdio.h>
2 int main() {
3     char lettre;
4     printf( "Inserez une lettre: " ); /* Admettons que l'utilisateur insere "a" */
5     char c = getchar();
6     printf( "Lettre inseree : %c\n", c );
7     /* Affiche "a" dans la console */
8 }

```

- **system()** : Fonction qui permet de lancer l'exécution d'une commande sur le système d'exploitation hôte.

Exemple :

```

1 #include <stdlib.h>
2 int main() {
3     char *commande = "dir";
4     system( commande );
5     /* Execute la commande "dir", fonction qui sert a afficher ce qui est present dans le
       repertoire */
6     return 0;
7 }

```

- **calloc()** : Fonction qui permet de faire de l'allocation dynamique de mémoire.

Exemple :

```

1 #include <stdio.h>

```

```

2 #include <stdlib.h>
3 int main() {
4     int *pointer;
5     int i, n;
6     printf( "Nombres d'elements a ajouter: " );
7     scanf( "%d", &n ); /* Admettons que l'utilisateur insere 3 */
8     pointer = ( int * ) calloc( n, sizeof( int ) );
9     for( i = 0; i < n; i++ )
10    {
11        printf( "Entrez le numero N.%d ", i + 1 );
12        scanf( "%d", &pointer[i] ); /* Admettons que l'utilisateur insere 1 2 et 3 */
13    }
14    for( i = 0; i < n; i++ )
15    {
16        printf( "%d ", pointer[i] );
17        /* Affiche 1 2 3 dans la console */
18    }
19    return 0;
20 }

```

1.2.1 Qu'est-ce l'allocation dynamique de mémoire ?

Une allocation dynamique de mémoire est le processus d'allouer de la mémoire lors de l'exécution d'un programme. Il existe quatre fonctions en C qui peuvent être utilisées pour allouer et libérer de la mémoire : `calloc()`, `free()`, `realloc()` et `malloc()`. Ces fonctions sont accessibles lors de l'introduction du header `<stdlib.h>` dans le code.

1.2.2 Qu'est-ce `calloc()` ?

`calloc()` est une fonction qui renvoie un pointeur vers un espace mémoire suffisamment libre pour stocker un tableau au nombre d'objets indéterminé et à la taille spécifiée. Si c'est pas le cas, la fonction renverra `NULL`. Le stockage est initialisé à zéro.

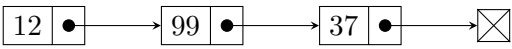
1.2.3 Qu'est-ce `malloc()` ?

`malloc()` est une fonction qui renvoie un pointeur vers un espace mémoire non initialisé. Si l'allocation n'est pas possible, la fonction renverra `NULL`. Si l'espace affecté par l'allocation est saturé, les résultats ne seront pas définis.

1.2.4 Différences entre `calloc()` et `malloc()`

<code>malloc()</code>	<code>calloc()</code>
<code>void * malloc(size_t n);</code>	<code>void * calloc(size_t n, size_t size);</code>
<code>malloc()</code> prend un argument (le nombre d'octets)	<code>calloc()</code> prend deux arguments (le nombre de blocs et la taille de chaque bloc)
<code>malloc()</code> est plus rapide que <code>calloc()</code>	<code>calloc()</code> prends plus de temps que <code>malloc()</code> car la mémoire doit être initialisée à zéro

2 Listes chaînées

Voici une représentation d'une liste chaînée : 

2.1 Qu'est-ce une liste chaînée ?

Une liste chaînée est une séquence de structures de données, qui sont liées par des nœuds. Chaque nœud contient une connexion vers un autre nœud.

Il existe trois types de listes chaînées :

- La **liste chaînée linéaire** : Une liste chaînée que ne peut être parcourue dans une seule direction.
- La **double liste chaînée** : Une liste chaînée qui peut être parcourue en deux directions : à partir de la tête ou à partir de la queue.
- La **liste chaînée circulaire** : Une liste chaînée où le premier nœud fait toujours référence au dernier nœud.

Dans le cas de notre projet, on a travaillé avec la liste chaînée linéaire.

Voici à quoi ressemble une structure pour une liste chaînée linéaire en C :

```
1 typedef struct noeud
2 {
3     int donnee;           /* L'information que l'on souhaite stocker dans le noeud */
4     struct noeud *suivant; /* Le lien vers le prochain noeud */
5 } noeud;
6 noeud *tete = NULL;      /* Le point de demarrage d'une liste chainee */
```

Contrairement à un vecteur, qui lui ressemblerait à ça :


```
1 int donnee[10];
```

Les deux vont très bien stocker des `int` que l'on a nommé `donnee`, cependant, un vecteur possède une taille fixe, il ne saura stocker que 10 éléments maximum dans notre cas. Tandis que dans le cas d'une liste chaînée, tant qu'il y a de l'espace en mémoire, il saura stocker un nombre indéfini de nœuds, et par conséquent, un nombre indéfini de `int donnee`.

2.2 Création d'un nouveau nœud

Afin de pouvoir créer un nœud que l'on stockera dans notre liste chaînée, il faudra d'abord lui allouer une taille dans la mémoire. Toujours en se basant par le bout de code vu précédemment, et le bout de code vu dans le chapitre précédent sur l'allocation de mémoire, on obtiendrait le code suivant :

```
1 noeud *creer_noeud( int data )
2 {
3     noeud *temporaire = ( noeud * ) calloc( 1, sizeof( noeud ) );
4     /* On cree un noeud temporaire ou l'on stockera les informations que l'on souhaite */
5     temporaire->data = data; /* La donnee que l'on souhaite stocker */
6     temporaire->suivant = NULL; /* Lien vers le prochain noeud s'il existe */
7     return tmp; /* On retourne le noeud qui sera stocke dans la liste chainee */
8 }
9 noeud *treinte_sept = creer_noeud( 37 );
10 free( treinte_sept );
```

Le résultat obtenu serait un nœud du style : 

Le langage C ne fait pas de **Garbage Collection**, ce qui revient à dire que toute manipulation de mémoire doit être manipulée manuellement. Donc, lorsqu'on a besoin de libérer de la mémoire de données que l'on peut écraser, on utilise la fonction `free()`.

Dans ce cas-ci, le nœud est créé mais il n'est pas encore stocké dans la liste chaînée. On verra dans la section suivant comment on peut stocker ce nouveau nœud dans la liste chaînée.

2.3 Insertion d'un nœud dans une liste chaînée

À l'état actuel des choses, on peut supposer que la liste chaînée soit vide : 

Une procédure d'ajout dans une liste chaînée pourrait être la suivante :

1. On initialise le noeud temporaire que l'on ajoutera à la base de données.

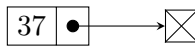
2. On associe **data** au pointeur **data** présent dans la structure **noeud**.
3. On initialise le prochain nœud de la liste ***suivant** à **NULL**.
4. Si la tête ***tete** de la liste est **NULL**, alors la tête devient le nouveau nœud. On arrête la fonction d'ajout là.
5. Sinon, on fait une copie de la tête dans le nœud ***suivant** que l'on avait initialisé à **NULL**.
6. On déclare la tête comme étant le nœud temporaire que l'on a initialisé.

Dans une fonction, cette procédure ressemblerait à ça :

```

1 int ajouter_noeud( noeud *tete, int data )
2 {
3     noeud *temporaire = creer_noeud( data );
4     /* On cree le noeud temporaire qui stocke la donnee */
5     if( tete == NULL )
6     {
7         tete = temporaire;
8         /* On le stocke directement temporaire a la place de tete si tete est NULL */
9         return 1;
10    }
11    temporaire->suivant = tete;
12    /* On fait une copie de tete (le noeud stocke precedemment) dans le suivant du noeud
13       temporaire (au lieu de le laisser a NULL) */
14    tete = temporaire;
15    /* On fait devenir le noeud temporaire la nouvelle tete */
16    return 1;
17 }
18 noeud *tete = NULL;
19 ajouter_noeud( tete, 37 );

```

Si la fonction a réussi, la liste chaînée ressemblera à ça : 

Admettons que l'on souhaite ajouter une autre donnée à notre liste chaînée, on refait appel à la fonction :

```

16 ajouter_noeud( tete, 99 );

```

Si la fonction a réussi, la liste chaînée ressemblera à ça : 

2.4 Suppression d'un nœud d'une liste chaînée

Pour que l'on puisse supprimer un nœud, il faudra d'abord passer par différentes étapes de vérification :

1. On vérifie que la tête de la liste ***tete** ne soit pas **NULL**, si oui, on arrête la fonction.
2. On vérifie si le premier élément de la liste correspond à la donnée **data** de la liste que l'on souhaite supprimer.
3. Si oui :
 - (a) On crée un nœud temporaire qui stockera la donnée suivante dans la liste.
 - (b) On libère l'espace mémoire occupé par head avec la fonction **free(tete)**.
 - (c) On attribue à **tete** le nœud temporaire que l'on avait créé.
 - (d) On arrête la fonction.
4. Sinon, on parcourt l'entièreté de la liste jusqu'au moment où l'on trouve la formation qui a le même **donnee** que la **donnee** en paramètre.
5. Si on le trouve, on pivote l'élément qui suit vers l'élément que l'on vient de supprimer.
6. On arrête la fonction, si réussite, on obtient 1, si pas, on obtient 0.

Cette procédure pourrait être écrite de la manière suivante :

```

1 int supprimer_noeud( noeud *tete, int data )
2 {
3     if( tete == NULL )
4     {
5         return 0;
6     }

```



```

7  noeud *temporaire = NULL;
8  if( tete->data == data )
9  {
10     temporaire = tete->suivant;
11     free( tete );
12     tete = temporaire;
13     return 1;
14 }
15 while( tete != NULL )
16 {
17     if( tete->suivant->data == data )
18     {
19         temporaire = tete->suivant;
20         tete->suivant = temporaire->suivant;
21         free( temporaire );
22         return 1;
23     }
24     tete = tete->suivant;
25 }
26 return 0;
27 }
28 noeud *tete = NULL;
29 ajouter_noeud( tete, 37 );
30 ajouter_noeud( tete, 99 );
31 ajouter_noeud( tete, 42 );
32 ajouter_noeud( tete, 12 );
33 ajouter_noeud( tete, 10 );
34 supprimer_noeud( tete, 10 );
35 supprimer_noeud( tete, 42 );

```

Analysons ce que le code fait :

28. On crée la tête à NULL
29. On ajoute dans la liste chaînée la variable 37 qui sera stockée dans un nœud.
30. On ajoute dans la liste chaînée la variable 99 qui sera stockée dans un nœud.
31. On ajoute dans la liste chaînée la variable 42 qui sera stockée dans un nœud.
32. On ajoute dans la liste chaînée la variable 12 qui sera stockée dans un nœud.
33. On ajoute dans la liste chaînée la variable 10 qui sera stockée dans un nœud.

À l'état actuel des choses, la liste ressemble à ça :



34. On supprime le nœud avec la valeur 10.
35. On supprime le nœud avec la valeur 42.

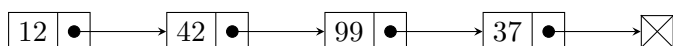
Au final la liste ressemblera à ça :



Voici comment la suppression interagit avec la liste chaînée et le nombre 10 :

1. Est-ce que la liste est vide ? (`tete == NULL` ?)
 - (a) Non car on a des nœuds dans notre liste, on continue dans la fonction.
2. Est-ce que la donnée `data` équivaut à la donnée `tete->donnee` ?
 - (a) Oui car `tete->donnee = 10` et `data = 10`.
3. On fait un backup de `tete->suivant` dans un nœud temporaire.
4. On libère le nœud `tete`.
5. On attribue à `tete` le nœud temporaire.

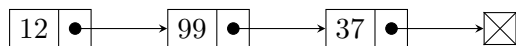
A ce stade de la modification, la liste ressemble à ça :



Voici comment la suppression interagit avec la liste chaînée et le nombre 42 :

1. Est-ce que la liste est vide ? (`tete == NULL` ?)

- (a) Non car on a des noeuds dans notre liste, on continue dans la fonction.
- 2. Est-ce que la donnée `data` équivaut à la donnée `tete->donnee` ?
 - (a) Non car `tete->donnee = 12` et `data = 42`.
- 3. On commence à parcourir toute la liste tant que `tete != NULL`.
- 4. On vérifie si `tete->suivant->donnee == data`
 - (a) Si c'est pas le cas, on continue à parcourir la liste.
- 5. Si c'est le cas, on fait un backup du suivant du suivant dans un nœud temporaire.
- 6. On libère `tete->suivant`.
- 7. On attribue à `tete->suivant` le nœud `tete->suivant->suivant` (on pivote la liste vers la gauche et on libère l'espace mémoire qui était occupée par le 42).
 A ce stade de la modification, la liste ressemble à ça :



Si la fonction a réussi, on obtiendra 1 comme résultat, si non, on obtiendra 0.

2.5 Affichage d'une liste chaînée

Pour l'affichage d'une liste chaînée, on parcourt l'entièreté de la liste chaînée `noeud`. Tant que la tête n'est pas `NULL`, on affichera les informations que l'on souhaite afficher de chaque nœud présente dans la liste chaînée.

Généralement, le code en langage C ressemblerait à ça :

```

1 void afficher_liste( noeud *tete )
2 {
3     while( tete != NULL )
4     {
5         printf( "%d ", tete->donnee );
6         /* Affiche 12 99 37 de maniere recursive */
7         tete = tete->suivant;
8     }
9 }
10 afficher_liste(tete);

```

Si on se base sur la liste chaînée :

Ce qui se passe dans le code est la manipulation suivante :

- 1. On vérifie si `tete` est différent de `NULL`.
- 2. Si c'est le cas, on affiche dans la console la valeur `tete->donnee`
- 3. On continue à parcourir la liste, donc `tete` devient `tete->suivant`
- 4. Même procédure tant que `tete != NULL`.

Donc, l'affichage sera : 12 99 37 dans la console.

3 Énoncé

Le centre de formations propose diverses formations dans différents domaines.

Chaque **formation** est caractérisée par :

- Un identifiant unique (un nombre entier)
- Un nombre de prérequis (Un nombre entier de 0 à 10)
- Les identifiants des formations qui seraient des prérequis (On suppose qu'une formation ne peut avoir que 10 prérequis maximum)
- Le nombre de jours où la formations a lieu (un entier de 1 à 7)
- Le jour précis (sous forme de int 1 - lundi, 2 - mardi, etc...)
- L'heure de début de la formation (en format 08.15)
- La durée de la formation en heures (un nombre entier)
- Un coût fixe (Un float positif, par exemple 175.55)
- Un nom (40 caractères maximum)

Le nombre de formations est indéfini.

Un nombre indéfini de personnes peuvent participer à n'importe quelle formation, bien des formateurs que des étudiants.

Une **personne** est caractérisée par :

- Un identifiant unique (Un nombre entier)
- Un nom de famille (25 caractères maximum)
- Un prénom (25 caractères maximum)
- S'il/elle est formateur ou pas (1 formateur, 0 étudiant)
- Le nombre de formations auquel il/elle participe (On suppose que dans une année, une personne quelconque ne puisse participer qu'à 30 formations maximum)
- Dans le cas d'un formateur, le nombre de jour où il/elle aurait des indisponibilités (de 1 à 7)
- Suivi par le jours précis où il/elle aurait des indisponibilités (1 - lundi, 2 - mardi, etc...)
- Dans le cas d'un étudiant, il se peut qu'il/elle ait droit à une remise. (1 - oui, 0 - non)
- Si c'est le cas, le pourcentage de sa remise sera précisé (de 0 à 100)

Points importants à maîtriser :

- Étant donné qu'il n'y a pas de nombre maximal de personnes qui peuvent être inscrits dans le centre de formations, et qu'il n'y a pas de nombre maximal de formations qu'un centre de formations puisse proposer, le stockage par vecteurs des diverses personnes et formations n'est pas permis. Dès lors il faut initialiser des listes chaînées qui stockeront les diverses informations.
- Le logiciel doit pouvoir afficher la liste des personnes ainsi que la liste des formations, et à partir de ces données, générer un planning pour la semaine.
- Il doit être également possible d'ajouter et de supprimer des nouvelles personne ou personnes dans leurs respectives bases de données.
- La possibilité de pouvoir attribuer une personne à une formation et de pouvoir la retirer d'une formation doit être prise en compte.
- Toutes les opérations doivent être dynamiques, donc, toute manipulation doit être possible à partir du moment où le programme est lancé jusqu'au moment où il est arrêté.

4 Programme

Le programme saura faire les manipulations suivantes :

- Ajouter des nouvelles personnes ou formations dans leurs respectives bases de données
- Supprimer des personnes ou des formations entièrement de leurs respectives bases de données
- Supprimer une personne spécifique d'une formation spécifique
- Afficher les informations stockées dans les respectives bases de données.
- Afficher un planning de la semaine détaillé
- Sauvegarder toute modification faite

En résultat, il est demandé d'obtenir le planning détaillé de la semaine dans un fichier `CaculliTyranowski.res`

4.1 Mode d'emploi

4.1.1 Menu principale

Tous les traitements se feront dans la console du logiciel en lisant l'entrée du clavier.

```
Projet par Giorgio Caculli et Jędrzej Tyranowski
*****
* MENU PRINCIPALE                                     *
*****
* 1: Afficher la liste des ętudiants/formateurs ou la liste des formations *
* 2: Ajouter une nouvelle personne ou formation a la base de donnees      *
* 3: Attribuer une personne a une formation                               *
* 4: Supprimer une formation, une personne ou une personne d'une formation *
* 0: Quitter le programme                                                 *
*****
* Que voudriez-vous faire ? █
```

Pour naviguer dans le menu, il faut frapper entre 0 et 4 sur le clavier et confirmer avec Enter.

4.1.2 Menu affichage

```
*****
* MENU AFFICHAGE                                       *
*****
* 1. Liste des personnes                               *
* 2. Liste des formations                             *
* 3. Planning de la semaine                           *
* 0. Retour                                           *
*****
* Que voudriez-vous afficher ? █
```

Choisissez une option parmi 1, 2 et 3 et confirmez avec Enter pour continuer.

4.1.3 Liste des personnes

```
* ID Nom                                Prenom                                Statut                                *
* -----                                -----                                -----                                *
* 10 Goossens                           Pierre                               Formateur                             *
* 9 Willems                             Jean                                Formateur                             *
* 8 Janssens                             Stephanie                           Formateur                             *
* 7 Peeters                             Nicolas                             Formateur                             *
* 6 Dupont                              Ghislain                            Formateur                             *
* 5 Brown                               Donald                              Formateur                             *
* 4 Smith                               John                                Etudiant                              *
* 3 Berg                                Adam                                Etudiant                              *
* 2 Kowal                               Simon                               Etudiant                              *
* 1 Mertens                             George                              Etudiant                              *
* *****                                *****                                *****                                *
* MENU AFFICHAGE                                *
* *****                                *****                                *****                                *
* 1. Liste des personnes                                *
* 2. Liste des formations                                *
* 3. Planning de la semaine                                *
* 0. Retour                                *
* *****                                *****                                *****                                *
* Que voudriez-vous afficher ? █
```

Affiche l'ID, le nom de famille, le prénom et le statut (étudiant ou formateur) sous la forme d'une liste.

4.1.4 Liste des formations

```
* ID Nom                                Prix                                *
* -----                                -----                                *
* 8 Medecine                             150.00                             *
* 7 Microbiologie                        185.35                             *
* 6 Jeux videos                          179.26                             *
* 5 Sociologie                           389.95                             *
* 4 Biologie                             378.87                             *
* 3 Java                                 175.45                             *
* 2 Latin                                152.25                             *
* 1 Anglais                              360.00                             *
* *****                                *****                                *****                                *
* MENU AFFICHAGE                                *
* *****                                *****                                *****                                *
* 1. Liste des personnes                                *
* 2. Liste des formations                                *
* 3. Planning de la semaine                                *
* 0. Retour                                *
* *****                                *****                                *****                                *
* Que voudriez-vous afficher ? █
```

Affiche l'ID, le nom et le prix de chaque formation disponible dans la base de données sous la forme d'une liste.

4.1.5 Planning de la semaine

```
*****
Cours du: Lundi
*****
ID: 7 - Nom formation: Microbiologie
Participants dans la formation:
Formateurs:
  8 Janssens Stephanie
Etudiants:
  2 Kowal Simon
  4 Smith John

De: 11.00 - A 13.00
Prerequis: Biologie

ID: 1 - Nom formation: Anglais
Participants dans la formation:
Formateurs:
  9 Willems Jean
Etudiants:
  1 Mertens George
  2 Kowal Simon
  3 Berg Adam
  4 Smith John

De: 14.00 - A 17.00
Prerequis: Aucun

*****
Cours du: Mardi
*****
ID: 3 - Nom formation: Java
Participants dans la formation:
Formateurs:
  6 Dupont Ghislain
Etudiants:
  1 Mertens George
  2 Kowal Simon
  3 Berg Adam

De: 14.00 - A 16.00
Prerequis: Aucun
```

Les informations sont groupées par jour suivi par le nom de la formation, les différents formateurs, les élèves qui y participent et l'horaiare. Les prérequis pour les formations sont dictées en bas, exemple Biologie est un prérequis pour la formation Microbiologie

4.1.6 Menu ajout

```
*****
* MENU AJOUT                                     *
*****
* 1. Ajouter une nouvelle personne a la base de donnees      *
* 2. Ajouter une nouvelle formation a la base de donneees     *
* 0. Retour                                                    *
*****
* Que voudriez-vous ajouter a la base de donnees ? █
```

Il est possible d'ajouter une nouvelle personne à la base de données ou ajouter une nouvelle formation à la base de données.

4.1.7 Ajout d'une personne

```
*****
* MENU AJOUT                                     *
*****
* 1. Ajouter une nouvelle personne a la base de donnees      *
* 2. Ajouter une nouvelle formation a la base de donneees     *
* 0. Retour                                                    *
*****
* Que voudriez-vous ajouter a la base de donnees ? 1
* Nom de famille de la personne: Zuckerberg
* Prenom de la personne: Mark
* Est-ce que cette personne est un formateur ou un etudiant ? (f/e) f
* Est-ce que ce formateur as des jours ou il/elle est indisponible ? (o/n) o
* Combien de jours serait ce formateur indisponible ? 2
* 1. lundi
* 2. mardi
* 3. mercredi
* 4. jeudi
* 5. vendredi
* 6. samedi
* 7. dimanche
* Quel serait son jour d'indisponibilite N.1 ? 1
* Quel serait son jour d'indisponibilite N.2 ? 7
* Etes vous █ur de vouloir ajouter formateur: Mark Zuckerberg a la base de donnees ? (o/n) o
```

L'ajout d'une personne se passe de la manière suivante :

1. Il faut insérer son nom de famille.
2. Il faut insérer son prénom.
3. Il faut insérer son statut (étudiant ou formateur), "f" pour formateur, "e" pour étudiant.
4. Il faut insérer le nombre de jours où le formateur est indisponible.
5. Dépendamment du nombre de jour, il faudra attribuer le jour correspondant à ceux dans la liste (1 - lundi, 2 - mardi, etc. . .)
6. Il faut confirmer par "o" ou "n" (oui ou non) si l'on souhaite effectivement ajouter la nouvelle personne dans la base de données.

```
* Zuckerberg Mark a ete ajoute(e) a la base de donnees avec succes *
*****
* MENU AJOUT                                     *
*****
* 1. Ajouter une nouvelle personne a la base de donnees      *
* 2. Ajouter une nouvelle formation a la base de donneees     *
* 0. Retour                                                    *
*****
* Que voudriez-vous ajouter a la base de donnees ? █
```

Si la personne a été correctement ajoutée à la base de données, un message de confirmation apparaîtra en haut du menu.

```

*****
* MENU AJOUT *
*****
* 1. Ajouter une nouvelle personne a la base de donnees *
* 2. Ajouter une nouvelle formation a la base de donnees *
* 0. Retour *
*****
* Que voudriez-vous ajouter a la base de donnees ? 1
* Nom de famille de la personne: Tyranowski
* Prenom de la personne: Jędrzej
* Est-ce que cette personne est un formateur ou un etudiant ? (f/e) e
* Est-ce que cet etudiant beneficie d'une reduction ? (o/n) o
* Combien de pourcentage de reduction beneficie cet etudiant ? 20
* Etes vous sur de vouloir ajouter etudiant: Jędrzej Tyranowski a la base de donnees ? (o/n) o

* Tyranowski Jędrzej a ete ajoute(e) a la base de donnees avec succes *
*****
* MENU AJOUT *
*****
* 1. Ajouter une nouvelle personne a la base de donnees *
* 2. Ajouter une nouvelle formation a la base de donnees *
* 0. Retour *
*****
* Que voudriez-vous ajouter a la base de donnees ? █

```

Lorsqu'on ajout un ętudiant, il faut indiquer son nom de famille, son pręnom et la remise ęventuelle. Męme message de confirmation pour l'ętudiant.

4.1.8 Ajout d'une formation

```
*****
* MENU AJOUT                                                                    *
*****
* 1. Ajouter une nouvelle personne a la base de donnees                        *
* 2. Ajouter une nouvelle formation a la base de donnees                      *
* 0. Retour                                                                    *
*****
* Que voudriez-vous ajouter a la base de donnees ? 2
* Nom de la formation: Interprete
* Cout de la formation: 250.00
* Est-ce que la formation a des prerequis ? (o/n) o
* ID Nom                                                                    *
* -----                                                                    *
* 8 Medecine                                                                    *
* 7 Microbiologie                                                                *
* 6 Jeux videos                                                                  *
* 5 Sociologie                                                                  *
* 4 Biologie                                                                    *
* 3 Java                                                                        *
* 2 Latin                                                                        *
* 1 Anglais                                                                    *
* Combien de prerequis faut-il ? 1
* ID du prerequis a rajouter: 1
* Combien de fois par semaine a le cours lieu ? 2
* 1. lundi
* 2. mardi
* 3. mercredi
* 4. jeudi
* 5. vendredi
* 6. samedi
* 7. dimanche
* A quel jour de la semain a le cours lieu N. 1 ? 3
* A quelle heure a-t-il lieu (eg. 08,15) ? 14.00
* Combien d'heures dure le cours ? 2.00
* A quel jour de la semain a le cours lieu N. 2 ? 5
* A quelle heure a-t-il lieu (eg. 08,15) ? 08.45
* Combien d'heures dure le cours ? 2.30
* Etes vous sur de vouloir ajouter la formation Interprete avec prix 250.00 a la base de donnees ? (o/n) 5
```

L'ajout d'une formation se passe de la manière suivante :

1. Il faut insérer son nom.
2. Il faut insérer son prix.
3. Il faut insérer insérer s'il y a des prérequis ou pas par "o" (oui) et "n" (non).
4. Si oui, il faut insérer le nombre de prérequis et l'ID des formations qui seraient des prérequis.
5. Ensuite il faut dire combien de fois par semaine la formation a lieu.
6. Dépendamment du nombre de jour, il faudra attribuer le jour correspondant à ceux dans la liste (1 - lundi, 2 - mardi, etc...)
7. Il faut insérer vers quelle heure commence la formation. (En format HH.MM)
8. Il faut insérer combien d'heures dure la formation. (En format HH.MM)
9. Il faut confirmer par "o" ou "n" (oui ou non) si l'on souhaite effectivement ajouter la nouvelle personne dans la base de données.

```
* Interprete a ete ajoutee a la base de donnees avec succes *
*****
* MENU AJOUT                                                                    *
*****
* 1. Ajouter une nouvelle personne a la base de donnees
* 2. Ajouter une nouvelle formation a la base de donnees
* 0. Retour                                                                    *
*****
* Que voudriez-vous ajouter a la base de donnees ? ^C
```

Un message de confirmation apparaîtra au dessus du menu.

4.1.9 Menu attribution

```
*****
* MENU ATTRIBUTION                                     *
*****
* Liste des cours                                     *
*****
* ID Nom                                              *
* -----
* 8 Medecine                                          *
* 7 Microbiologie                                   *
* 6 Jeux videos                                     *
* 5 Sociologie                                       *
* 4 Biologie                                         *
* 3 Java                                             *
* 2 Latin                                            *
* 1 Anglais                                          *
* 0 Retour                                           *
*****
* A quel cours voudriez vous attribuer une personne? █
```

Lorsqu'on rentre dans le menu d'attribution, il faudra sélectionner la formation dans laquelle on souhaite attribuer une personne, formateur ou étudiant.

```

*****
* MENU ATTRIBUTION
*****
* Liste des cours
*****
* ID Nom
* -----
* 8 Medecine
* 7 Microbiologie
* 6 Jeux videos
* 5 Sociologie
* 4 Biologie
* 3 Java
* 2 Latin
* 1 Anglais
* 0 Retour
*****
* A quel cours voudriez vous attribuer une personne? 1
*****
* Liste des personnes
*****
* Cours choisi: Anglais
*****
* ID Nom          PreNom          Statut
* -----
* 10 Goossens      Pierre          Formateur
* 9 Willems        Jean            Formateur
* 8 Janssens       Stephanie       Formateur
* 7 Peeters        Nicolas         Formateur
* 6 Dupont         Ghislain        Formateur
* 5 Brown          Donald          Formateur
* 4 Smith          John            Etudiant
* 3 Berg           Adam            Etudiant
* 2 Kowal          Simon           Etudiant
* 1 Mertens        George          Etudiant
* 0 Retour
*****
* Qui voudriez vous attribuer au cours de: Anglais ? 1
* Etes vous sur de vouloir attribuer Mertens George au cours de Anglais ? (o/n) o

```

L'attribution à une formation se passe de la manière suivante :

1. Il faut insérer l'ID de la formation à laquelle on souhaite ajouter un personne quelconque.
2. Il faut insérer l'ID de la personne à rajouter.
3. Il faut confirmer par "o" ou "n" (oui ou non) si l'on souhaite effectivement ajouter la nouvelle personne dans la base de données.

```

* Mertens George est deja present dans le cours de Anglais *
*****
* MENU PRINCIPALE
*****
* 1: Afficher la liste des etudiants/formateurs ou la liste des formations
* 2: Ajouter une nouvelle personne ou formation a la base de donnees
* 3: Attribuer une personne a une formation
* 4: Supprimer une formation, une personne ou une personne d'une formation
* 0: Quitter le programme
*****
* Que voudriez-vous faire ? █

```

Lors de la réussite de l'attribution de la personne dans la formation, un message de confirmation apparaîtra.

4.1.10 Menu suppression

```
*****
* MENU SUPPRESSION                                     *
*****
* 1. Supprimer une personne de la base de donnees      *
* 2. Supprimer une formation de la base de donnees     *
* 3. Supprimer une personne specifique d'une formation *
* 0. Retour                                             *
*****
* Que voudriez vous supprimer ? █
```

Voici à quoi ressemble l'interface par lequel il est possible de faire des suppressions.

1. Il est possible de supprimer entièrement une personne de la base de données. Par conséquent, la personne en question disparaîtra des formations aussi.
2. Il est possible de supprimer entièrement une formation de la base de données. Par conséquent, les gens qui participaient à la formation en question ne pourront plus y participer.
3. Il est possible de supprimer une personne précise d'une formation précise, bien la formation que la personne apparaîtront dans leurs respectives bases de données, mais ils ne seront plus liés ensemble.

4.1.11 Suppression d'une personne

```
*****
* MENU SUPPRESSION                                     *
*****
* 1. Supprimer une personne de la base de donnees      *
* 2. Supprimer une formation de la base de donnees     *
* 3. Supprimer une personne specifique d'une formation *
* 0. Retour                                             *
*****
* Que voudriez vous supprimer ? 1
* ID Nom                                Prenom          Statut      *
* -----
* 10 Goossens                           Pierre          Formateur   *
* 9 Willems                             Jean            Formateur   *
* 8 Janssens                             Stephanie       Formateur   *
* 7 Peeters                             Nicolas         Formateur   *
* 6 Dupont                               Ghislain        Formateur   *
* 5 Brown                               Donald          Formateur   *
* 4 Smith                               John            Etudiant    *
* 3 Berg                                Adam            Etudiant    *
* 2 Kowal                               Simon           Etudiant    *
* 1 Mertens                             George          Etudiant    *
* 0 Retour
* Quelle personne voudriez vous supprimer entierement ? █
```

Dans le cas d'une suppression d'une personne de la base de données, il suffit d'insérer l'ID de la personne à supprimer et la personne ne sera plus visible dans aucune formation. Comme toute manipulation, des messages de confirmation apparaîtront à la fin de la manipulation.

4.1.12 Suppression d'une formation

```
*****
* MENU SUPPRESSION                                     *
*****
* 1. Supprimer une personne de la base de donnees      *
* 2. Supprimer une formation de la base de donnees    *
* 3. Supprimer une personne specifique d'une formation *
* 0. Retour                                             *
*****
* Que voudriez vous supprimer ? 2
*****
* MENU SUPPRESSION : Liste des cours                    *
*****
* ID Nom                                                *
* -----
* 8 Medecine                                           *
* 7 Microbiologie                                     *
* 6 Jeux videos                                       *
* 5 Sociologie                                        *
* 4 Biologie                                          *
* 3 Java                                              *
* 2 Latin                                             *
* 1 Anglais                                           *
* 0 Retour                                             *
*****
* Quelle formation voudriez vous supprimer? █
```

Suivant la même logique que la suppression d'une personne, lors de la suppression d'une formation de la base de données, il suffira d'introduire son ID, appuyer sur Enter et confirmer avec un "o" (oui) ou annuler avec un "n" (non).

4.1.13 Suppression d'une personne d'une formation spécifique

```
*****
* MENU SUPPRESSION                                     *
*****
* 1. Supprimer une personne de la base de donnees      *
* 2. Supprimer une formation de la base de donnees     *
* 3. Supprimer une personne specifique d'une formation *
* 0. Retour                                             *
*****
* Que voudriez vous supprimer ? 3
*****
* MENU SUPPRESSION : Liste des cours                    *
*****
* 8 Medecine                                           *
* 7 Microbiologie                                      *
* 6 Jeux videos                                       *
* 5 Sociologie                                        *
* 4 Biologie                                           *
* 3 Java                                              *
* 2 Latin                                             *
* 1 Anglais                                           *
* 0 Retour                                             *
* De quelle formation voudriez vous supprimer quelqu'un ? 2
Cours choisi: Latin
* 1 Mertens      George      Etudiant      *
* 2 Kowal        Simon      Etudiant      *
* 3 Berg         Adam       Etudiant      *
* 4 Smith        John       Etudiant      *
* 5 Brown        Donald     Formateur     *
* 0 Retour                                             *
* Quelle personne voudriez vous supprimer de ce cours ? 4
* Etes vous sur de vouloir supprimer Smith John du cours Latin ? (o/n) o
```

La suppression d'une personne spécifique d'une formation spécifique se passe de la manière suivante :

1. Il faut insérer l'ID de la formation à partir duquel on souhaite supprimer une personne.
2. Il faut insérer l'ID de la personne à supprimer.
3. Il faut confirmer par "o" ou "n" (oui ou non) si l'on souhaite effectivement ajouter la nouvelle personne dans la base de données.

```
* Smith John a ete supprime du cours Latin avec succes *
```

```
ID: 2 - Nom formation: Latin
Participants dans la formation:
Formateurs:
5 Brown Donald
Etudiants:
1 Mertens George
2 Kowal Simon
3 Berg Adam
```

```
*****
* MENU SUPPRESSION                                     *
*****
* 1. Supprimer une personne de la base de donnees      *
* 2. Supprimer une formation de la base de donnees     *
* 3. Supprimer une personne specifique d'une formation *
* 0. Retour                                             *
*****
* Que voudriez vous supprimer ?
```

Si la suppression a réussi, un message affichera la réussite de la suppression et affichera le nouvel état de la formation.

5 Code

5.1 Structures

Voici les différentes structures qui ont été utilisées dans la conception du programme :

```
30 typedef struct personne
31 {
32     int id;
33     char nom[25];
34     char prenom[25];
35     int formateur;
36     int nb_formationen;
37     int formations[30];
38     int nb_jours_indisponible;
39     int jours_indisponible[7];
40     int reduction;
41     int val_reduction;
42 } personne;
```

Cette structure sert à stocker les informations qui composent une personne quelconque : étudiant ou formateur.

Voici ce que représente chaque partie de la structure :

- `int id` : L'identifiant unique de la personne.
- `char nom[25]` : Le nom de la personne (25 caractères maximum).
- `char prenom[25]` : Le prénom de la personne (25 caractères maximum).
- `int formateur` : 1 si la personne est un formateur, 0 si la personne est un étudiant.
- `int nb_formationen` : Le nombre de formations auquel la personne participera.
- `int formations[30]` : Vecteur qui stockera les identifiants des différentes formations auquel la personne participera (On suppose dans une année, une personne ne peut participer qu'à 30 formations maximum).
- `int nb_jours_indisponible` : Si la personne est un formateur, il se peut qu'il/elle ait des jours d'indisponibilité, cette variable va stocker le nombre de jours où cette personne est indisponible (maximum 7).
- `int jours_indisponibles[7]` : Le vecteur qui stockera les jours auquel le formateur ne sera pas disponible (1 - lundi, 2 - mardi, etc...).
- `int reduction` : Si la personne est un étudiant, il se peut qu'il ait une réduction sur son minerval, 1 s'il a droit à une réduction, 0 si pas.
- `int val_reduction` : Le pourcentage de réduction auquel un étudiant a droit.

```
51 typedef struct noeud_db_personne
52 {
53     personne *p;
54     struct noeud_db_personne *next;
55 } noeud_db_personne;
```

Cette structure sert à devenir les différents nœuds qui seront stockés dans la base de donnée, soit la structure `db_personne`. Voici ce qui représente chaque partie de la structure :

- `personne *p` : Le pointeur de la personne qui sera stocké dans ce nœud lors de sa création.
- `struct noeud_db_personne *next` : qui contiendra le tête lors qu'on créera un nouveau nœud, sinon NULL.

```
63 typedef struct db_personne
64 {
65     noeud_db_personne *head;
66 } db_personne;
```

Cette structure sert à contenir tous les différents nœuds `noeud_db_personne`. C'est à partir de cette structure que l'on stockera les différentes nœuds qui eux-mêmes stockeront leurs personnes respectives.

- `noeud_db_personne *head` : La tête de la liste chaînée qui stockera toutes les personnes.

```
73 typedef struct noeud_formation
74 {
75     personne *p;
76     struct noeud_formation *next;
77 } noeud_formation;
```

Cette structure va stocker les différentes personnes qui participeront à une formation spécifique.

— `personne *p` : La personne qui participera à la formation.

— `struct noeud_formation *next` : Le nœud de pour la prochaine personne qui sera stockée.

```
94 typedef struct formation
95 {
96     int id;
97     char nom[40];
98     float prix;
99     int nb_jours;
100    int jours[7];
101    float heures[24];
102    float durees[10];
103    int nb_prerequis;
104    int prerequis[10];
105    noeud_formation *head;
106 } formation;
```

Cette structure sert à stocker toutes les informations qui composent une formations. Voici ce que chaque partie représente :

- `int id` : L'identifiant unique de la formation.
- `char nom[40]` : Le nom de la formation (40 caractères maximum).
- `float prix` : Le coût de la formation.
- `int nb_jours` : Le nombre de jours par semaine où cette formation a lieu.
- `int jours[7]` : Vecteur contenant les jours où la formation a lieu.
- `float heures[24]` : Le nombre d'heures du début de la formation.
- `float durees[10]` : Les différentes durées de la formation lors de la semaine.
- `int nb_prerequis` : Le nombre de prérequis pour avoir accès à cette formation.
- `int prerequis[10]` : Vecteur contenant les identifiants des formations qui seraient des prérequis.
- `noeud_formation *head` : Étant donné qu'une formation stocke des personnes, elle-même est une liste chaînée qui stockera un nombre indéterminé de participants.

```
115 typedef struct noeud_db_formation
116 {
117     formation *f;
118     struct noeud_db_formation *next;
119 } noeud_db_formation;
```

Cette structure suit la même logique que la structure `noeud_db_personne`. Elle sert à stocker les différentes formations, qui eux-mêmes stockeront les personnes à leurs tour.

— `formation *f` : La formation qui sera stockée dans la base de données.

— `struct noeud_db_formation *next` : La prochaine formation qui sera stockée dans la base de données. NULL si pas de prochaine formation.

```
127 typedef struct db_formation
128 {
129     noeud_db_formation *head;
130 } db_formation;
```

Cette structure aussi suit la même logique que la structure `db_formation`. Elle sert de tête pour la la liste chaînée et c'est à partir de cette structure-ci que l'on démarrera les différentes interactions avec la base de données des formations.

— `noeud_db_formation *head` : La tête de la liste chaînée qui stockera les différentes formations.

Pourquoi avons-nous choisi cette approche? Principalement car on ne voulait pas stocker les différentes personnes et les différentes formations dans des vecteurs. On ne voulait pas qu'il y ait un nombre prédéfini de personnes et un nombre prédéfini de formations, on s'est donc fiés aux listes chaînées. Et c'est pourquoi maintenant il est possible de stocker autant de formations que la RAM nous permet ainsi que de stocker autant de formations que la RAM nous permet.

5.2 Fonctions

5.2.1 Fonctions qui créent des nœuds

Voici les différentes fonctions que l'on a du créer pour le bon fonctionnement du programme :

```
144 personne *creer_personne( char nom[], char prenom[], int formateur );
```

Cette fonction sert à créer un pointeur qui permettra d'initialiser les différentes informations présentes dans la structure **personne**. Lors de l'initialisation d'une personne, on n'aura besoin que du nom de famille de la personne, son prenom et s'il/elle est un formateur ou pas. Le reste des informations est manipulé par la suite lors des différentes interactions.

```
168 db_personne *creer_db_personne();
```

Cette fonction sert à initialiser le pointer **noeud_db_personne *head** dans la structure **db_personne** à NULL, afin que l'on puisse commencer à faire des manipulations avec cette structure.

```
303 formation *creer_formation( char nom[], float prix );
```

Cette fonction sert à créer un pointeur qui permettra d'initialiser les différentes informations présentes dans la structure **formation**. Lors de l'initialisation d'une formation, on n'aura besoin que du nom de la formation et de son prix. Le reste des informations est manipulé par la suite lors des différentes interactions.

```
442 db_formation *creer_db_formation();
```

Cette fonction sert à initialiser le pointer **noeud_db_formation *head** dans la structure **db_formation** à NULL, afin que l'on puisse commencer à faire des manipulations avec cette structure.

5.2.2 Fonctions qui affichent des listes chaînées

```
157 void afficher_personne( personne *p );
```

Cette fonction sert à afficher les informations de base qui caractérisent une **personne**. De manière générale, son identifiant, son nom de famille, son prénom et s'il est formateur ou étudiant.

```
257 void afficher_db_personne( db_personne *db );
```

Cette fonction parcourt l'entièreté de la base de données **db_personne *db**. Tant que la tête n'est pas NULL, on affichera les informations que l'on souhaite afficher de chaque personne présente dans la base de données.

```
408 void afficher_formation( formation *f );
```

Cette fonction sert à afficher les informations de base qui caractérisent une **formation**. De manière générale, son identifiant, son nom, son prix, ainsi que les personnes qui y participent.

```
559 void afficher_db_formation( db_formation *dbf );
```

Cette fonction parcourt l'entièreté de la base de données **db_formation *dbf**. Tant que la tête n'est pas NULL, on affichera les informations que l'on souhaite afficher de chaque formation présente dans la base de données.

5.2.3 Fonctions qui ajoutent un nœud à une liste chaînée

```
187 void ajouter_db_personne( db_personne *db, personne *p );
```

Cette fonction sert à initialiser un pointeur **noeud_db_personne *ndb** qui stockera **personne *p** dans la base de données **db_personne *db**. Ici, l'ajout dans la liste chaînée a lieu par le mécanisme suivant :

1. On initialise le noeud temporaire que l'on ajoutera à la base de données.
2. On associe **p** au pointeur **p** présent dans la structure **noeud_db_personne**.
3. On initialise le prochain nœud de la liste ***next** à NULL.
4. Si la tête ***head** de la base de donnée est NULL, alors la tête devient le nouveau nœud. On arrête la fonction d'ajout là.

5. Sinon, on fait une copie de la tête dans le nœud ***next** que l'on avait initialisé à NULL.
6. On déclare la tête comme étant le nœud temporaire que l'on a initialisé.

```
322 int ajouter_formation( formation *f, personne *p );
```

Cette fonction sert à initialiser un pointeur **noeud_formation *nf** qui stockera **personne *p** qui participera dans **formation *f**. Ici, l'ajout dans la liste chaînée à lieu par le mécanisme suivant :

1. On initialise le noeud temporaire que l'on ajoutera dans la formation.
2. On associe **p** au pointeur **p** présent dans la structure **noeud_formation**.
3. On initialise le prochain nœud de la liste ***next** à NULL.
4. Si la tête ***head** de la formation est NULL, alors la tête devient le nouveau nœud. On arrête la fonction d'ajout là.
5. Sinon, on fait une copie de la tête dans le nœud ***next** que l'on avait initialisé à NULL.
6. On déclare la tête comme étant le nœud temporaire que l'on a initialisé.

```
461 void ajouter_db_formation( db_formation *db, formation *f );
```

Cette fonction sert à initialiser un pointeur **noeud_db_formation *ndb** qui stockera **formation *f** dans la base de données **db_formation *db**. Ici, l'ajout dans la liste chaînée à lieu par le mécanisme suivant :

1. On initialise le noeud temporaire que l'on ajoutera à la base de données.
2. On associe **f** au pointeur **f** présent dans la structure **noeud_db_formation**.
3. On initialise le prochain nœud de la liste ***next** à NULL.
4. Si la tête ***head** de la base de donnée est NULL, alors la tête devient le nouveau nœud. On arrête la fonction d'ajout là.
5. Sinon, on fait une copie de la tête dans le nœud ***next** que l'on avait initialisé à NULL.
6. On déclare la tête comme étant le nœud temporaire que l'on a initialisé.

5.2.4 Fonctions qui suppriment un nœud d'une liste chaînée

```
216 int supprimer_db_personne( db_personne *dbp, int id );
```

Cette fonction sert à supprimer une personne de la base de données à partir de son identifiant. La démarche faite dans cette fonction est la suivant :

1. On vérifie que la tête de la base de données **dbp->head** ne soit pas NULL, si oui, on arrête la fonction.
2. On vérifie si le premier élément de la liste correspond à l'id de la personne que l'on souhaite supprimer.
3. Si oui :
 - (a) On crée un nœud temporaire qui stockera la personne suivante dans la liste.
 - (b) On libère l'espace mémoire occupé par head avec la fonction **free(dbp->head)**.
 - (c) On attribue à **dbp->head** le nœud temporaire que l'on avait créé.
 - (d) On arrête la fonction.
4. Sinon, on parcourt l'entièreté de la liste jusqu'au moment où l'on trouve la personne qui a le même id que l'id en paramètre.
5. Si on le trouve, on pivote l'élément qui suit vers l'élément que l'on vient de supprimer.
6. On arrête la fonction, si réussite, on obtient 1, si pas, on obtient 0.

```
361 int supprimer_personne_de_formation( formation *f, int id );
```

Cette fonction sert à supprimer une personne de la fonction à partir de son identifiant. La démarche faite dans cette fonction est la suivant :

1. On vérifie que la tête de la fonction `f->head` ne soit pas `NULL`, si oui, on arrête la fonction.
2. On vérifie si le premier élément de la liste correspond à l'id de la personne que l'on souhaite supprimer.
3. Si oui :
 - (a) On crée un nœud temporaire qui stockera la personne suivante dans la liste.
 - (b) On libère l'espace mémoire occupé par `head` avec la fonction `free(f->head)`.
 - (c) On attribue à `f->head` le nœud temporaire que l'on avait créé.
 - (d) On arrête la fonction.
4. Sinon, on parcourt l'entière de la liste jusqu'au moment où l'on trouve la personne qui a le même id que l'id en paramètre.
5. Si on le trouve, on pivote l'élément qui suit vers l'élément que l'on vient de supprimer.
6. On arrête la fonction, si réussite, on obtient 1, si pas, on obtient 0.

```
490 int supprimer_db_formation( db_formation *dbf, int id );
```

Cette fonction sert à supprimer une formation de la base de données à partir de son identifiant. La démarche faite dans cette fonction est la suivant :

1. On vérifie que la tête de la base de données `dbf->head` ne soit pas `NULL`, si oui, on arrête la fonction.
2. On vérifie si le premier élément de la liste correspond à l'id de la formation que l'on souhaite supprimer.
3. Si oui :
 - (a) On crée un nœud temporaire qui stockera la formation suivante dans la liste.
 - (b) On libère l'espace mémoire occupé par `head` avec la fonction `free(dbf->head)`.
 - (c) On attribue à `dbf->head` le nœud temporaire que l'on avait créé.
 - (d) On arrête la fonction.
4. Sinon, on parcourt l'entière de la liste jusqu'au moment où l'on trouve la formation qui a le même id que l'id en paramètre.
5. Si on le trouve, on pivote l'élément qui suit vers l'élément que l'on vient de supprimer.
6. On arrête la fonction, si réussite, on obtient 1, si pas, on obtient 0.

5.2.5 Fonctions qui servent de getter()

```
275 personne *get_personne( db_personne *db, char nom[], char prenom[], int formateur );
```

Cette fonction renvoie `NULL` si une personne d'un nom spécifique, d'un prénom spécifique, et s'il est formateur ou étudiant n'existe pas dans la base de données `db_personne *db`. Sinon, la fonction retourne la personne trouvée.

```
532 formation *get_formation( db_formation *dbf, char nom_formation[] );
```

Cette fonction renvoie `NULL` si une formation avec un nom spécifique n'existe pas dans la base de données `db_formation *dbf`. Sinon, la fonction retourne la formation trouvée.

5.2.6 Fonctions qui affiches les différentes parties du menu interactif

les fonctions ci-dessous ne servent qu'à afficher les différentes parties du menu interactif. Elle ne font aucune manipulation particulière autre qu'afficher du texte. Le `main()` est la seule fonction qui lit les données et les initialise dans la base de données avant que le menu soit affiché.

```
618 void menu_creer_formation( db_formation *f );
```

```
793 void menu_creer_personne( db_personne *p );
```

```

953 int menu_creer( db_formation *f, db_personne *p );

990 void menuajouter_formation( db_formation *f, db_personne *p );

1103 void menu_supprimer_personne( db_formation *dbf, db_personne *dbp );

1182 void menu_supprimer_formation( db_formation *dbf, db_personne *dbp );

1271 int menu_supprimer_personne_de_formation( db_formation *dbf );

1396 int menu_supprimer( db_formation *dbf, db_personne *dbp );

1436 int menu_affichage( db_formation *f, db_personne *p );

1489 int ecrire_planning( db_formation *dbf );

1572 int menu( db_formation *f, db_personne *p );

1704 int main( void );

```

5.2.7 Entièrement du code

```

1 #include <ctype.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
7 char *clear = "cls";
8 #elif __unix__ || __APPLE__ && __MACH__
9 char *clear = "clear";
10 #endif
11
12 /******
13 /*
14 /*
15  * int id : L'identifiant unique de la personne
16  * char nom[25] : Le nom de la personne (25 caracteres maximum)
17  * char prenom[25] : Le prenom de la personne (25 caracteres maximum)
18  * int formateur : 1 si la personne est un formateur, 0 si la personne est un etudiant
19  * int nb_ formations : Le nombre de formations auquel la personne participera
20  * int formations[30] : Vecteur qui stockera les identifiants des differentes formations
    auquel la personne participera
21  * (On suppose dans une annee, une personne ne peut participer qu'a 30 formations maximum)
22  * int nb_jours_indisponible : Si la personne est un formateur, il se peut qu'il/elle ait des
    jours d'indisponible,
23  * cette variable va stocker le nombre de jours ou cette personne est indisponible (maximum 7)
24  * int jours_indisponibles[7] : Le vecteur qui stockera les jours auquel le formateur ne sera
    pas disponible
25  * (1 - lundi, 2 - mardi, etc...)
26  * int reduction : Si la personne est un etudiant, il se peut qu'il ait une reduction sur son
    minerval,
27  * 1 s'il a droit a une reduction, 0 si pas
28  * int val_reduction : Le pourcentage de reduction auquel un etudiant a droit
29 */
30 typedef struct personne
31 {
32     int id;
33     char nom[25];
34     char prenom[25];
35     int formateur;
36     int nb_ formations;
37     int formations[30];
38     int nb_jours_indisponible;
39     int jours_indisponible[7];
40     int reduction;
41     int val_reduction;
42 } personne;
43
44 /*
45  * Cette structure sert a devenir les differents noeuds qui seront stockes dans la base de
    donnee,
46  * soit la structure db_personne.

```

```

47  * Voici ce qui represent chaque partie de la structure:
48  * personne *p : Le pointeur de la personne qui sera stocke dans ce nud lors de sa creation.
49  * struct noeud_db_personne *next : qui contiendra le tete lors qu'on creera un nouveau nud,
    sinon NULL.
50  */
51  typedef struct noeud_db_personne
52  {
53      personne *p;
54      struct noeud_db_personne *next;
55  } noeud_db_personne;
56
57  /*
58  * Cette structure sert a contenir tous les differentes noeuds noeud_db_personne.
59  * C'est a partir de cette structure que l'on stockera les differentes noeuds qui eux-memes
    stockeront
60  * leurs personnes respectives.
61  * noeud_db_personne *head : La tete de la liste chainees qui stockera toutes les personnes.
62  */
63  typedef struct db_personne
64  {
65      noeud_db_personne *head;
66  } db_personne;
67
68  /*
69  * Cette structure va stocker les differentes personnes qui participeront a une formation
    specifique.
70  * personne *p : La personne qui participera a la formation.
71  * struct noeud_formation *next : Le noeud de pour la prochaine personne qui sera stockee.
72  */
73  typedef struct noeud_formation
74  {
75      personne *p;
76      struct noeud_formation *next;
77  } noeud_formation;
78
79  /*
80  * Cette structure sert a stocker toutes les informations qui composent une formations.
81  * Voici ce que chaque partie represente:
82  * int id : L'identifiant unique de la formation.
83  * char nom[40] : Le nom de la formation (40 caracteres maximum).
84  * float prix : Le cout de la formation.
85  * int nb_jours : Le nombre de jours par semaine ou cette formation a cours.
86  * int jours[7] : Vecteur contenant les jours ou la formation a cours.
87  * float heures[24] : Le nombre d'heures du debut de la formation.
88  * float durees[10] : Les differentes durees du cours lors de la semaine.
89  * int nb_prerequis : Le nombre de prerequis pour avoir acces a cette formation.
90  * int prerequis[10] : Vecteur contenant les identifiants des formations qui seraient des
    prerequis.
91  * noeud_formation *head : Etant donne qu'une formation stocke des personnes,
92  * elle-meme est une liste chainees qui stockera un nombre indetermine de participants.
93  */
94  typedef struct formation
95  {
96      int id;
97      char nom[40];
98      float prix;
99      int nb_jours;
100     int jours[7];
101     float heures[24];
102     float durees[10];
103     int nb_prerequis;
104     int prerequis[10];
105     noeud_formation *head;
106  } formation;
107
108  /*
109  * Cette structure suit la meme logique que la structure noeud_db_personne.
110  * Elle sert a stocker les differentes formations, qui eux-memes stockeront les personnes a
    leurs tour.
111  * formation *f : La formation qui sera stockee dans la base de donnees.
112  * struct noeud_db_formation *next : La prochaine formation qui sera stockee dans la base de
    donnees.
113  * NULL si pas de prochaine formation.
114  */
115  typedef struct noeud_db_formation
116  {

```

```

117     formation *f;
118     struct noeud_db_formation *next;
119 } noeud_db_formation;
120
121 /*
122  * Cette structure aussi suit la meme logique que la structure db_formation.
123  * Elle sert de tete pour la la liste chainee et c'est a partir de cette structure-ci que l'on
124    demarrera
125  * les differentes interactions avec la base de donnees des formations.
126  * noeud_db_formation *head : La tete de la liste chainee qui stockera les differentes
127    formations.
128  */
129 typedef struct db_formation
130 {
131     noeud_db_formation *head;
132 } db_formation;
133
134 /*
135  * FIN STRUCTS
136  */
137
138 /*
139  * PERSONNE
140  */
141
142 /*
143  * Cette fonction sert a creer un pointeur qui permettra d'initialiser les differentes
144    informations presentes
145  * dans la structure personne.
146  * Lors de l'initialisation d'une personne, on n'aura besoin que du nom de famille de la
147    personne,
148  * son prenom et s'il/elle est un formateur ou pas.
149  * Le reste des informations est manipule par la suite lors des differentes interactions.
150  */
151 personne *creer_personne( char nom[], char prenom[], int formateur )
152 {
153     personne *e = ( personne * ) calloc( sizeof( personne ), sizeof( personne ) );
154     strcpy( e->nom, nom );
155     strcpy( e->prenom, prenom );
156     e->formateur = formateur;
157     return e;
158 }
159
160 /*
161  * Cette fonction sert a afficher les informations de base qui caracterisent une personne.
162  * De maniere generale, son identifiant, son nom de famille, son prenom et s'il est formateur
163    ou etudiant.
164  */
165 void afficher_personne( personne *p )
166 {
167     personne *tmp = p;
168     printf( "* %2d %-25s %-25s %-10s *\n",
169            tmp->id, tmp->nom, tmp->prenom, tmp->formateur ? "Formateur" : "Etudiant" );
170 }
171
172 /*
173  * Cette fonction sert a initialiser le pointer noeud_db_personne *head dans la structure
174    db_personne a NULL,
175  * afin que l'on puisse commencer a faire des manipulations avec cette structure.
176  */
177 db_personne *creer_db_personne()
178 {
179     db_personne *db = ( db_personne * ) calloc( sizeof( db_personne ), sizeof( db_personne ) )
180     ;
181     db->head = NULL;
182     return db;
183 }
184
185 /*
186  * Cette fonction sert a initialiser un pointeur noeud_db_personne *ndb qui stockera personne
187    *p
188  * dans la base de donnees db_personne *db.
189  * Ici, l'ajout dans la liste chainee a lieu par le mecanisme suivant:
190  * On initialise le noeud temporaire que l'on ajoutera a la base de donnees.
191  * On associe p au pointeur p present dans la structure noeud_db_personne.
192  * On initialise le prochain noeud de la liste *next a NULL.
193  * Si la tete *head de la base de donnee est NULL, alors la tete devient le nouveau noeud.
194  * On arrete la fonction d'ajout la.
195  * Sinon, on fait une copie de la tete dans le noeud *next que l'on avait initialise a NULL.

```

```

185  * On declare la tete comme etant le noeud temporaire que l'on a initialise.
186  */
187  void ajouter_db_personne( db_personne *db, personne *p )
188  {
189      noeud_db_personne *ndb = ( noeud_db_personne * ) calloc( sizeof( noeud_db_personne ),
190                          sizeof( noeud_db_personne ) );
191      ndb->p = p;
192      ndb->next = NULL;
193      if( db->head == NULL )
194      {
195          db->head = ndb;
196          return;
197      }
198      ndb->next = db->head;
199      db->head = ndb;
200  }
201  /*
202  * Cette fonction sert a supprimer une personne de la base de donnees a partir de son
203  * identifiant.
204  * La demarche faite dans cette fonction est la suivant:
205  * On verife que la tete de la base de donnees dbp->head ne soit pas NULL, si oui, on arrete
206  * la fonction.
207  * On verifie si le premier element de la liste correspond a l'id de la personne que l'on
208  * souhaite supprimer.
209  * Si oui:
210  * On cree un noeud temporaire qui stockera la personne suivante dans la liste.
211  * On libere l'espace memoire occupe par head avec la fonction free(dbp->head).
212  * On attribue a dbp->head le noeud temporaire que l'on avait cree.
213  * On arrete la fonction.
214  * Sinon, on parcourt l'entierete de la liste jusqu'au moment ou l'on trouve la personne qui a
215  * le meme id que
216  * l'id en parametre.
217  * Si on le trouve, on pivote l'element qui suit vers l'element que l'on vient de supprimer.
218  * On arrete la fonction, si reussite, on obtient 1, si pas, on obtient 0.
219  */
220  int supprimer_db_personne( db_personne *dbp, int id )
221  {
222      noeud_db_personne *ndbp = dbp->head;
223      if( ndbp == NULL )
224      {
225          return 0;
226      }
227      noeud_db_personne *tmp = NULL;
228      if( ndbp->p->id == id )
229      {
230          tmp = dbp->head->next;
231          free( dbp->head );
232          dbp->head = tmp;
233          return 1;
234      }
235      while( ndbp != NULL )
236      {
237          if( ndbp->next == NULL )
238          {
239              if( ndbp->p->id == id )
240              {
241                  return 1;
242              }
243              return 0;
244          }
245          if( ndbp->next->p->id == id )
246          {
247              tmp = ndbp->next;
248              ndbp->next = tmp->next;
249              free( tmp );
250              return 1;
251          }
252          ndbp = ndbp->next;
253      }
254      return 0;
255  }
256  /*
257  * Cette fonction parcourt l'entierete de la base de donnees db_personne *db. Tant que la tete
258  * n'est pas NULL,

```

```

255  * on affichera les informations que l'on souhaite afficher de chaque personne presente dans
256  * la base de donnees.
257  void afficher_db_personne( db_personne *db )
258  {
259      db_personne *tmpdb = db;
260      noeud_db_personne *tmpnodb = tmpdb->head;
261      printf( " * %2s %-25s %-25s %-9s          *\n", "ID", "Nom", "Prenom", "Statut" );
262      printf( " * -----*\n" );
263      while( tmpnodb != NULL )
264      {
265          afficher_personne( tmpnodb->p );
266          tmpnodb = tmpnodb->next;
267      }
268  }
269
270  /*
271  * Cette fonction renvoie NULL si une formation avec un nom specifique n'existe pas dans
272  * la base de donnees db_formation *dbf.
273  * Sinon, la fonction retourne la formation trouvee.
274  */
275  personne *get_personne( db_personne *db, char nom[], char prenom[], int formateur )
276  {
277      db_personne *tmpdb = db;
278      noeud_db_personne *tmpnodb = tmpdb->head;
279      while( tmpnodb != NULL )
280      {
281          if( strcmp( tmpnodb->p->nom, nom ) == 0 &&
282              strcmp( tmpnodb->p->prenom, prenom ) == 0 &&
283              tmpnodb->p->formateur == formateur )
284          {
285              return tmpnodb->p;
286          }
287          tmpnodb = tmpnodb->next;
288      }
289      return NULL;
290  }
291
292  /*                                  FIN PERSONNE                                  */
293  /******
294
295  /******
296  /*                                  FORMATION                                  */
297  /*
298  * Cette fonction sert a creer un pointeur qui permettra d'initialiser les differentes
299  * informations presentes dans
300  * la structure formation.
301  * Lors de l'initialisation d'une formation, on n'aura besoin que du nom de la formation et de
302  * son prix.
303  * Le reste des informations est manipule par la suite lors des differentes interactions.
304  */
305  formation *creer_formation( char nom[], float prix )
306  {
307      formation *tmp = ( formation * ) calloc( sizeof( formation ), sizeof( formation ) );
308      strcpy( tmp->nom, nom );
309      tmp->prix = prix;
310      tmp->head = NULL;
311      return tmp;
312  }
313
314  /*
315  * Cette fonction sert a initialiser un pointeur noeud_formation *nf qui stockera personne *p
316  * qui participera
317  * dans formation *f. Ici,
318  * l'ajout dans la liste chainee a lieu par le mecanisme suivant:
319  * On initialise le noeud temporaire que l'on ajoutera dans la formation.
320  * On associe p au pointeur p present dans la structure noeud_formation.
321  * On initialise le prochain noeud de la liste *next a NULL.
322  * Si la tete *head de la formation est NULL, alors la tete devient le nouveau noeud. On
323  * arrete la fonction d'ajout la.
324  * Sinon, on fait une copie de la tete dans le noeud *next que l'on avait initialise a NULL.
325  * On declare la tete comme etant le noeud temporaire que l'on a initialise.
326  */
327  int ajouter_formation( formation *f, personne *p )
328  {
329      noeud_formation *nf = ( noeud_formation * ) calloc( sizeof( noeud_formation ), sizeof(

```



```

    noeud_formation) );
325     nf->p = p;
326     nf->next = NULL;
327     if( f->head == NULL )
328     {
329         f->head = nf;
330         return 1;
331     }
332     noeud_formation *tmpnf = f->head;
333     while( tmpnf != NULL )
334     {
335         if( tmpnf->p->id == p->id )
336         {
337             return 0;
338         }
339         tmpnf = tmpnf->next;
340     }
341     nf->next = f->head;
342     f->head = nf;
343     return 1;
344 }
345
346 /*
347  * Cette fonction sert a supprimer une personne de la fonction a partir de son identifiant.
348  * La demarche faite dans cette fonction est la suivant:
349  * On verifie que la tete de la fonction f->head ne soit pas NULL, si oui, on arrete la
350  * fonction.
351  * On verifie si le premier element de la liste correspond a l'id de la personne que l'on
352  * souhaite supprimer.
353  * Si oui:
354  * On cree un noeud temporaire qui stockera la personne suivante dans la liste.
355  * On libere l'espace memoire occupe par head avec la fonction free(f->head).
356  * On attribue a f->head le noeud temporaire que l'on avait cree.
357  * On arrete la fonction.
358  * Sinon, on parcourt l'entierete de la liste jusqu'au moment ou l'on trouve la personne
359  * qui a le meme id que l'id en parametre.
360  * Si on le trouve, on pivote l'element qui suit vers l'element que l'on vient de supprimer.
361  * On arrete la fonction, si reussite, on obtient 1, si pas, on obtient 0.
362  */
361 int supprimer_personne_de_formation( formation *f, int id )
362 {
363     formation *tmpf = f;
364     noeud_formation *tmp = NULL;
365     if( f == NULL )
366     {
367         printf( "Formation pas trouvee\n" );
368         return 0;
369     }
370     noeud_formation *headf = tmpf->head;
371     if( headf == NULL )
372     {
373         return 0;
374     }
375     if( f->head->p->id == id )
376     {
377         tmp = f->head->next;
378         free( f->head );
379         f->head = tmp;
380         return 1;
381     }
382     while( headf != NULL )
383     {
384         if( headf->next == NULL )
385         {
386             if( headf->p->id == id )
387             {
388                 return 1;
389             }
390             return 0;
391         }
392         if( headf->next->p->id == id )
393         {
394             tmp = headf->next;
395             headf->next = tmp->next;
396             free( tmp );
397             return 1;

```

```

398     }
399     headf = headf->next;
400 }
401 return 0;
402 }
403
404 /*
405  * Cette fonction sert a afficher les informations de base qui caracterisent une formation.
406  * De maniere generale, son identifiant, son nom, son prix, ainsi que les personnes qui y
    participent.
407  */
408 void afficher_formation( formation *f )
409 {
410     formation *tmp = f;
411     noeud_formation *tmpnf = tmp->head;
412     printf( "ID: %d - Nom formation: %s\n", tmp->id, tmp->nom );
413     printf( "Participants dans la formation:\n" );
414     printf( "Formateurs:\n" );
415     while( tmpnf != NULL )
416     {
417         if( tmpnf->p->formateur == 1 )
418         {
419             printf( "%2d %s %s", tmpnf->p->id, tmpnf->p->nom, tmpnf->p->prenom );
420             if( tmpnf != NULL ) printf( "\n" );
421         }
422         tmpnf = tmpnf->next;
423     }
424     tmpnf = tmp->head;
425     printf( "Etudiants:\n" );
426     while( tmpnf != NULL )
427     {
428         if( tmpnf->p->formateur == 0 )
429         {
430             printf( "%2d %s %s", tmpnf->p->id, tmpnf->p->nom, tmpnf->p->prenom );
431             if( tmpnf != NULL ) printf( "\n" );
432         }
433         tmpnf = tmpnf->next;
434     }
435     printf( "\n" );
436 }
437
438 /*
439  * Cette fonction sert a initialiser le pointer noeud_db_formation *head dans la structure
    db_formation a NULL,
440  * afin que l'on puisse commencer a faire des manipulations avec cette structure.
441  */
442 db_formation *creer_db_formation()
443 {
444     db_formation *db = ( db_formation * ) calloc( sizeof( db_formation ), sizeof( db_formation
    ) );
445     db->head = NULL;
446     return db;
447 }
448
449 /*
450  * Cette fonction sert a initialiser un pointeur noeud_db_formation *ndb qui stockera
    formation *f dans
451  * la base de donnees db_formation *db.
452  * Ici, l'ajout dans la liste chainee a lieu par le mecanisme suivant:
453  * On initialise le noeud temporaire que l'on ajoutera a la base de donnees.
454  * On associe f au pointeur f present dans la structure noeud_db_formation.
455  * On initialise le prochain noeud de la liste *next a NULL.
456  * Si la tete *head de la base de donnee est NULL, alors la tete devient le nouveau noeud.
457  * On arrete la fonction d'ajout la.
458  * Sinon, on fait une copie de la tete dans le noeud *next que l'on avait initialise a NULL.
459  * On declare la tete comme etant le noeud temporaire que l'on a initialise.
460  */
461 void ajouter_db_formation( db_formation *db, formation *f )
462 {
463     noeud_db_formation *ndb = ( noeud_db_formation * ) calloc( sizeof( noeud_db_formation ),
    sizeof( noeud_db_formation ) );
464     ndb->f = f;
465     ndb->next = NULL;
466     if( db->head == NULL )
467     {
468         db->head = ndb;

```

```

469         return;
470     }
471     ndb->next = db->head;
472     db->head = ndb;
473 }
474
475 /*
476  * Cette fonction sert a supprimer une formation de la base de donnees a partir de son
477   identifiant.
478  * La demarche faite dans cette fonction est la suivante:
479  * On verifie que la tete de la base de donnees dbf->head ne soit pas NULL, si oui, on arrete
480   la fonction.
481  * On verifie si le premier element de la liste correspond a l'id de la formation que l'on
482   souhaite supprimer.
483  * Si oui:
484  * On cree un noeud temporaire qui stockera la formation suivante dans la liste.
485  * On libere l'espace memoire occupe par head avec la fonction free(dbf->head).
486  * On attribue a dbf->head le noeud temporaire que l'on avait cree.
487  * On arrete la fonction.
488  * Sinon, on parcourt l'entierete de la liste jusqu'au moment ou l'on trouve la formation qui
489   a le meme id que l'id en parametre.
490  * Si on le trouve, on pivote l'element qui suit vers l'element que l'on vient de supprimer.
491  * On arrete la fonction, si reussite, on obtient 1, si pas, on obtient 0.
492  */
493 int supprimer_db_formation( db_formation *dbf, int id )
494 {
495     noeud_db_formation *tmp = NULL;
496     if( dbf->head == NULL )
497     {
498         return 0;
499     }
500     if( dbf->head->f->id == id )
501     {
502         tmp = dbf->head->next;
503         free( dbf->head );
504         dbf->head = tmp;
505         return 1;
506     }
507     noeud_db_formation *tmpndbf = dbf->head;
508     while( tmpndbf != NULL )
509     {
510         if( tmpndbf->next == NULL )
511         {
512             if( tmpndbf->f->id == id )
513             {
514                 return 1;
515             }
516             return 0;
517         }
518         if( tmpndbf->next->f->id == id )
519         {
520             tmp = tmpndbf->next;
521             tmpndbf->next = tmpndbf->next->next;
522             free( tmp );
523             return 1;
524         }
525         tmpndbf = tmpndbf->next;
526     }
527     return 0;
528 }
529
530 /*
531  * Cette fonction renvoie NULL si une formation avec un nom specifique n'existe pas dans
532   la base de donnees db_formation *dbf.
533  * Sinon, la fonction retourne la formation trouvee.
534  */
535 formation *get_formation( db_formation *dbf, char nom_formation[] )
536 {
537     db_formation *tmpdbf = dbf;
538     noeud_db_formation *tmpndbf = tmpdbf->head;
539     while( tmpndbf != NULL )
540     {
541         if( strcmp( tmpndbf->f->nom, nom_formation ) == 0 )
542         {
543             return tmpndbf->f;
544         }
545     }
546 }

```

```

542         tmpndbf = tmpndbf->next;
543     }
544     return NULL;
545 }
546
547 /*
548  * Cette fonction sert a initialiser un pointeur noeud_db_formation *ndb qui stockera
549  * la base de donnees db_formation *db.
550  * Ici, l'ajout dans la liste chainee a lieu par le mecanisme suivant:
551  * On initialise le noeud temporaire que l'on ajoutera a la base de donnees.
552  * On associe f au pointeur f present dans la structure noeud_db_formation.
553  * On initialise le prochain noeud de la liste *next a NULL.
554  * Si la tete *head de la base de donnee est NULL, alors la tete devient le nouveau noeud.
555  * On arrete la fonction d'ajout la.
556  * Sinon, on fait une copie de la tete dans le noeud *next que l'on avait initialise a NULL.
557  * On declare la tete comme etant le noeud temporaire que l'on a initialise.
558  */
559 void afficher_db_formation( db_formation *dbf )
560 {
561     int i;
562     db_formation *tmpdbf = dbf;
563     noeud_db_formation *tmpndbf = tmpdbf->head;
564     char jour[7][9] = { "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"
565     " };
566     for( i = 1; i <= 7; i++ )
567     {
568         tmpndbf = tmpndbf->head;
569         printf( "
570         *****\n" );
571         printf( "Cours du: %s\n", jour[i - 1] );
572         printf( "
573         *****\n" );
574         while( tmpndbf != NULL )
575         {
576             formation *tmp = tmpndbf->f;
577             int j;
578             for( j = 0; j < tmp->nb_jours; j++ )
579             {
580                 if( tmp->jours[j] == i )
581                 {
582                     afficher_formation( tmp );
583                     printf( "De: %.2f - A %.2f\n", tmp->heures[j], tmp->heures[j] + tmp->
584                     durees[j] );
585                     printf( "Prerequis: " );
586                     if( tmp->nb_prerequis > 0 )
587                     {
588                         db_formation *tmpdb = dbf;
589                         noeud_db_formation *tmp_prereq_ndbf = tmpdb->head;
590                         while( tmp_prereq_ndbf != NULL )
591                         {
592                             formation *tmp_prereq = tmp_prereq_ndbf->f;
593                             int k;
594                             for( k = 0; k < tmp->nb_prerequis; k++ )
595                             {
596                                 if( tmp->prerequis[k] == tmp_prereq->id )
597                                 {
598                                     printf( "%s ", tmp_prereq->nom );
599                                 }
600                             }
601                             tmp_prereq_ndbf = tmp_prereq_ndbf->next;
602                         }
603                         printf( "\n\n" );
604                     }
605                     else
606                     {
607                         printf( "Aucun\n\n" );
608                     }
609                 }
610             }
611             tmpndbf = tmpndbf->next;
612         }
613     }
614 }
615
616 /*
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

613 /*****
614
615 *****/
616 /*          FONCTIONS GENERALES          */
617
618 void menu_creer_formation( db_formation *f )
619 {
620     db_formation *tmpdbf = f;
621     int i;
622     char nom[40];
623     float prix;
624     printf( "* Nom de la formation: " );
625     fgets( nom, 40, stdin );
626     if( strlen( nom ) > 0 && nom[ strlen( nom ) - 1 ] == '\n' )
627     {
628         nom[ strlen( nom ) - 1 ] = '\0';
629     }
630     formation *tmp = get_formation( f, nom );
631     if( tmp == NULL )
632     {
633         printf( "* Cout de la formation: " );
634         while( scanf( "%f", &prix ) != 1 )
635         {
636             printf( "* Option %.2f - INVALIDE: Min 0\n", prix );
637             printf( "* Cout de la formation: " );
638             scanf( "%f", &prix );
639             getchar();
640         }
641         while( prix < 0 )
642         {
643             printf( "* Option %.2f - INVALIDE: Min 0\n", prix );
644             printf( "* Cout de la formation: " );
645             scanf( "%f", &prix );
646             getchar();
647         }
648         formation *tmpf = creer_formation( nom, prix );
649         if( tmpdbf->head == NULL )
650         {
651             tmpf->id = 1;
652         }
653         else
654         {
655             tmpf->id = tmpdbf->head->f->id + 1;
656         }
657         char choix_prerequis[4];
658         printf( "* Est-ce que la formation a des prerequis ? (o/n) " );
659         scanf( "%s", choix_prerequis );
660         getchar();
661         while( strcmp( choix_prerequis, "o" ) != 0 && strcmp( choix_prerequis, "oui" ) != 0 &&
662                strcmp( choix_prerequis, "n" ) != 0 && strcmp( choix_prerequis, "non" ) != 0 )
663         {
664             printf( "* Options %s - INVALIDE: o pour oui et n pour non s'il vous plait\n",
665                    choix_prerequis );
666             printf( "* Est-ce que la formation a des prerequis ? (o/n) " );
667             scanf( "%s", choix_prerequis );
668             getchar();
669         }
670         if( strcmp( choix_prerequis, "o" ) == 0 || strcmp( choix_prerequis, "oui" ) == 0 )
671         {
672             noeud_db_formation *tmpndbf = tmpdbf->head;
673             printf( "* %2s %-40s                                *\n", "ID", "Nom" );
674             printf( "----- *\n" );
675             while( tmpndbf != NULL )
676             {
677                 printf( "* %2d %-40s                                *\n", tmpndbf->f->id,
678                        tmpndbf->f->nom );
679                 tmpndbf = tmpndbf->next;
680             }
681             tmpndbf = tmpdbf->head;
682             int nb_prerequis;
683             printf( "* Combien de prerequis faut-il ? " );
684             while( scanf( "%d", &nb_prerequis ) != 1 )
685             {
686                 printf( "* Option %d - INVALIDE: Max %d prerequis Min 0\n", nb_prerequis,
687                        tmpndbf->f->id );

```

```

685         printf( "* Combien de prerequis faut-il ? " );
686         scanf( "%d", &nb_prerequis );
687         getchar();
688     }
689     getchar();
690     while( nb_prerequis > tmpndbf->f->id || nb_prerequis < 0 )
691     {
692         printf( "* Option %d - INVALIDE: Max %d prerequis Min 0\n", nb_prerequis,
tmpndbf->f->id );
693         printf( "* Combien de prerequis faut-il ? " );
694         scanf( "%d", &nb_prerequis );
695         getchar();
696     }
697     tmpf->nb_prerequis = nb_prerequis;
698     for( i = 0; i < nb_prerequis; i++ )
699     {
700         int id_prerequis;
701         printf( "* ID du prerequis N.%d a rajouter: ", i + 1 );
702         while( scanf( "%d", &id_prerequis ) != 1 || nb_prerequis > tmpndbf->f->id )
703         {
704             printf( "* Option %d - INVALIDE: veuillez choisir 1 dans la liste\n",
id_prerequis );
705             printf( "* ID du prerequis N.%d a rajouter: ", i + 1 );
706             scanf( "%d", &id_prerequis );
707             getchar();
708         }
709         tmpf->prerequis[i] = id_prerequis;
710     }
711 }
712 int nb_jours;
713 printf( "* Combien de fois par semaine a la formation lieu ? " );
714 while( scanf( "%d", &nb_jours ) != 1 )
715 {
716     printf( "Option %d - INVALIDE: Max 7 jours Min 1 jour\n", nb_jours );
717     printf( "* Combien de fois par semaine a la formation lieu ? " );
718     scanf( "%d", &nb_jours );
719     getchar();
720 }
721 getchar();
722 while( nb_jours > 7 || nb_jours <= 0 )
723 {
724     printf( "Option %d - INVALIDE: Max 7 jours Min 1 jour\n", nb_jours );
725     printf( "* Combien de fois par semaine a laformation lieu ? " );
726     scanf( "%d", &nb_jours );
727     getchar();
728 }
729 tmpf->nb_jours = nb_jours;
730 printf( "* 1. lundi\n* 2. mardi\n* 3. mercredi\n* 4. jeudi\n* 5. vendredi\n* 6. samedi
\n* 7. dimanche\n" );
731 for ( i = 0; i < tmpf->nb_jours; i++ )
732 {
733     int jour;
734     printf( "* Quel jour de la semaine a la formation N. %d lieu ? ", i + 1 );
735     while( scanf( "%d", &jour ) != 1 || jour < 0 || jour > 7 )
736     {
737         printf( "* Option %d - INVALIDE: Max 7 Min 1\n", i );
738         printf( "* Quel jour de la semaine a la formation N. %d lieu ? ", i + 1 );
739         scanf( "%d", &jour );
740         getchar();
741     }
742     tmpf->jours[i] = jour;
743     float heure;
744     printf( "* A quelle heure debute la formation (ex. 08.15) ? " );
745     while( scanf( "%f", &heure ) != 1 || heure > 18 || heure < 6 )
746     {
747         printf( "* Option %.2f - INVALIDE: MIN 6 MAX 18\n", heure );
748         printf( "* A quelle heure debute la formation (ex. 08.15) ? " );
749         scanf( "%f", &heure);
750         getchar();
751     }
752     tmpf->heures[i] = heure;
753     float duree;
754     printf( "* Combien d'heures dure la formation ? " );
755     while( scanf( "%f", &duree ) != 1 || duree > 8 || duree < 1 )
756     {
757         printf( "* Option %.2f - INVALIDE: MIN 1 heures MAX 8 heures\n", duree );

```

```

758         printf( "* Combien d'heures dure la formation ? " );
759         scanf( "%f", &duree);
760         getchar();
761     }
762     tmpf->durees[i] = duree;
763 }
764 char confirmation[4];
765 printf( "* Etes vous sur de vouloir ajouter la formation %s avec prix %.2f a la base
de donnees ? (o/n) ",
766         tmpf->nom, tmpf->prix );
767 scanf( "%s", confirmation );
768 while( strcmp( confirmation, "o" ) != 0 && strcmp( confirmation, "oui" ) != 0 &&
769         strcmp( confirmation, "n" ) != 0 && strcmp( confirmation, "non" ) != 0 )
770 {
771     printf( "Veuillez inserer o / oui - n / non : " );
772     scanf( "%s", confirmation );
773 }
774 if( strcmp( confirmation, "o" ) == 0 || strcmp( confirmation, "oui" ) == 0 )
775 {
776     ajouter_db_formation( tmpdbf, tmpf );
777     system( clear );
778     printf( "* %s a ete ajoutee a la base de donnees avec succes *\n", tmpf->nom );
779 }
780 else
781 {
782     system( clear );
783     printf( "* %s n'a PAS ete ajoutee a la base de donnees *\n", tmpf->nom );
784 }
785 }
786 else
787 {
788     system( clear );
789     printf( "* Formation deja existante dans la base de donnees *\n" );
790 }
791 }
792
793 void menu_creer_personne( db_personne *p )
794 {
795     db_personne *tmpdbp = p;
796     char nom[25], prenom[25], choix_formateur[4];
797     int formateur, nb_jours_indisponible = 0, jours_indisponible[7], reduction = 0,
pourcent_reduction;
798     printf( "* Nom de famille de la personne: " );
799     scanf( "%s", nom );
800     printf( "* Prenom de la personne: " );
801     scanf( "%s", prenom );
802     printf( "* Est-ce que cette personne est un formateur ou un etudiant ? (f/e) " );
803     scanf( "%s", choix_formateur );
804     int i;
805     for( i = 0; choix_formateur[i]; i++ )
806     {
807         choix_formateur[i] = tolower( choix_formateur[i] );
808     }
809     while( strcmp( choix_formateur, "f" ) != 0 && strcmp( choix_formateur, "formateur" ) != 0
&&
810         strcmp( choix_formateur, "e" ) != 0 && strcmp( choix_formateur, "etudiant" ) != 0 )
811     {
812         printf( "* Option %s - INVALIDE: f ou e seulement sont acceptees
*\n",
813                 choix_formateur );
814         printf( "* Est-ce que cette personne est un formateur ou un etudiant ? (f/e) " );
815         scanf( "%s", choix_formateur );
816         for( i = 0; choix_formateur[i]; i++ )
817         {
818             choix_formateur[i] = tolower( choix_formateur[i] );
819         }
820     }
821     if( strcmp( choix_formateur, "f" ) == 0 || strcmp( choix_formateur, "formateur" ) == 0 )
822     {
823         formateur = 1;
824         char choix_indisponible[4];
825         printf( "* Est-ce que ce formateur as des jours ou il/elle est indisponible ? (o/n) "
);
826         scanf( "%s", choix_indisponible );
827         getchar();
828         while( strcmp( choix_indisponible, "o" ) != 0 && strcmp( choix_indisponible, "oui" )

```

```

829         != 0 &&
830         strcmp( choix_indisponible, "n" ) != 0 && strcmp( choix_indisponible, "non" )
831         != 0 )
832     {
833         printf( "* Option %s - INVALIDE: o ou n sont acceptees
834         *\n",
835                 choix_indisponible );
836         printf( "* Est-ce que ce formateur as des jours ou il/elle est indisponible ? (o/n
837         ) " );
838         scanf( "%s", choix_indisponible );
839         getchar();
840     }
841     if( strcmp( choix_indisponible, "o" ) == 0 || strcmp( choix_indisponible, "oui" ) == 0
842     )
843     {
844         printf( "* Combien de jours serait ce formateur indisponible ? " );
845         scanf( "%d", &nb_jours_indisponible );
846         getchar();
847         while( nb_jours_indisponible < 0 )
848         {
849             printf( "* Option %d - INVALIDE: Max 7 jours - Min 0
850             *\n",
851                     nb_jours_indisponible );
852             printf( "* Combien de jours serait ce formateur indisponible ? " );
853             scanf( "%d", &nb_jours_indisponible );
854             getchar();
855         }
856         if( nb_jours_indisponible > 0 )
857         {
858             while ( nb_jours_indisponible > 7 || nb_jours_indisponible < 0 )
859             {
860                 printf( "* Option %d - INVALIDE: Max 7 jours - Min 0
861                 *\n",
862                         nb_jours_indisponible );
863                 printf( "* Combien de jours serait ce formateur indisponible ? " );
864                 scanf( "%d", &nb_jours_indisponible );
865                 getchar();
866             }
867             printf( "* 1. lundi\n* 2. mardi\n* 3. mercredi\n* 4. jeudi\n* 5. vendredi\n*
868             6. samedi\n* 7. dimanche\n" );
869             for ( i = 0; i < nb_jours_indisponible; i++ )
870             {
871                 int jour;
872                 printf( "* Quel serait son jour d'indisponibilite N.%d ? ", i + 1 );
873                 scanf( "%d", &jour );
874                 jours_indisponible[ i ] = jour;
875             }
876         }
877     }
878     else
879     {
880         formateur = 0;
881         printf( "* Est-ce que cet etudiant beneficie d'une reduction ? (o/n) " );
882         char choix_reduction[4];
883         scanf( "%s", choix_reduction );
884         getchar();
885         while( strcmp( choix_reduction, "o" ) != 0 && strcmp( choix_reduction, "oui" ) != 0 &&
886               strcmp( choix_reduction, "n" ) != 0 && strcmp( choix_reduction, "non" ) != 0 )
887         {
888             printf( "* Option %s - INVALIDE: o ou n sont acceptees
889             *\n",
890                     choix_reduction );
891             printf( "* Est-ce que cet etudiant beneficie d'une reduction ? (o/n) " );
892             scanf( "%s", choix_reduction );
893             getchar();
894             while ( pourcent_reduction > 100 || pourcent_reduction < 0 )
895             {
896                 printf( "* Combien de pourcentage de reduction beneficie cet etudiant ? " );
897                 scanf( "%d", &pourcent_reduction );
898                 getchar();
899                 while ( pourcent_reduction > 100 || pourcent_reduction < 0 )
900                 {
901                     printf( "* Option %d - INVALIDE: Max 100 pourcent - Min 0
902                     *\n",
903                             pourcent_reduction );
904                     printf( "* Combien de pourcentage de reduction beneficie cet etudiant ? " );
905                     scanf( "%d", &pourcent_reduction );
906                     getchar();
907                 }
908             }
909         }
910     }
911 }

```



```

895         nb_jours_indisponible );
896         printf( "* Combien de pourcentage de reduction beneficie cet etudiant ? " );
897         scanf( "%d", &pourcent_reduction );
898     }
899     if( pourcent_reduction > 0 )
900     {
901         reduction = 1;
902     }
903 }
904 }
905 personne *tmpp = creer_personne( nom, prenom, formateur );
906 if( tmpdbp->head == NULL )
907 {
908     tmpp->id = 1;
909 }
910 else
911 {
912     tmpp->id = tmpdbp->head->p->id + 1;
913 }
914 if( tmpp->formateur == 0 )
915 {
916     tmpp->reduction = reduction;
917     if( reduction == 1 )
918     {
919         tmpp->val_reduction = pourcent_reduction;
920     }
921 }
922 else
923 {
924     tmpp->nb_jours_indisponible = nb_jours_indisponible;
925     if( nb_jours_indisponible > 0 )
926     {
927         memcpy( tmpp->jours_indisponible, jours_indisponible, sizeof(tmpp->
jours_indisponible) );
928     }
929 }
930 char confirmation[4];
931 printf( "* Etes vous sur de vouloir ajouter %s: %s %s a la base de donnees ? (o/n) ",
932         tmpp->formateur ? "formateur" : "etudiant", tmpp->prenom, tmpp->nom );
933 scanf( "%s", confirmation );
934 while( strcmp( confirmation, "o" ) != 0 && strcmp( confirmation, "oui" ) != 0 &&
935         strcmp( confirmation, "n" ) != 0 && strcmp( confirmation, "non" ) != 0 )
936 {
937     printf( "Veuillez inserer o / oui - n / non : " );
938     scanf( "%s", confirmation );
939 }
940 if( strcmp( confirmation, "o" ) == 0 || strcmp( confirmation, "oui" ) == 0 )
941 {
942     ajouter_db_personne( p, tmpp );
943     system( clear );
944     printf( "* %s %s a ete ajoute(e) a la base de donnees avec succes *\n", tmpp->nom,
tmpp->prenom );
945 }
946 else
947 {
948     system( clear );
949     printf( "* %s %s n'a PAS ete ajoute(e) a la base de donnees *\n", tmpp->nom, tmpp->
prenom );
950 }
951 }
952
953 int menu_cree( db_formation *f, db_personne *p )
954 {
955     int choix;
956     do
957     {
958         db_formation *tmpdbf = f;
959         db_personne *tmpdbp = p;
960         printf( "
*****\n" );
961         printf( "* MENU AJOUT
*\n" );
962         printf( "
*****\n" );
963         printf( "* 1. Ajouter une nouvelle personne a la base de donnees
*\n" );

```

```

964     printf( "* 2. Ajouter une nouvelle formation a la base de donnees
*\n" );
965     printf( "* 0. Retour
*\n" );
966     printf( "
*****\n" );
967     printf( "* Que voudriez-vous ajouter a la base de donnees ? " );
968     scanf( "%d", &choix );
969     getchar();
970     switch( choix )
971     {
972         case 1:
973             menu_creer_personne( tmpdbp );
974             break;
975         case 2:
976             menu_creer_formation( tmpdbf );
977             break;
978         case 0:
979             system( clear );
980             break;
981         default:
982             system( clear );
983             printf( "/!\ Option %d - INVALIDE /!\n", choix );
984             break;
985     }
986 } while( choix != 0 );
987 return 0;
988 }
989
990 void menuajouter_formation( db_formation *f, db_personne *p )
991 {
992     db_formation *tmpdbf = f;
993     noeud_db_formation *tmpndbf = tmpdbf->head;
994     db_personne *tmpdbp = p;
995     noeud_db_personne *tmpndbp = tmpdbp->head;
996     printf( "*****\n" );
997     printf( "* MENU ATTRIBUTION *\n" );
998     printf( "*****\n" );
999     printf( "*****\n" );
1000    printf( "* Liste des cours *\n" );
1001    printf( "*****\n" );
1002    printf( " * %2s %-40s *\n", "ID", "Nom" );
1003    printf( " * ----- *\n" );
1004    while( tmpndbf != NULL )
1005    {
1006        formation *tmpf = tmpndbf->f;
1007        printf( " * %2d %-40s *\n", tmpf->id, tmpf->nom );
1008        tmpndbf = tmpndbf->next;
1009    }
1010    tmpndbf = tmpndbf->head;
1011    printf( "* 0 Retour *\n" );
1012    printf( "*****\n" );
1013    int cours;
1014    printf( "* A quelle formation voudriez vous attribuer une personne? " );
1015    scanf( "%d", &cours );
1016    getchar();
1017    while( cours < 0 && cours > tmpndbf->f->id )
1018    {
1019        printf( "* Valeur %d - INVALIDE! *\n", cours );
1020        printf( "* A quelle formation voudriez vous attribuer une personne? " );
1021        scanf( "%d", &cours );
1022        getchar();
1023    }
1024    if( cours <= 0 )
1025    {
1026        system( clear );
1027        return;

```

```

1028 }
1029 while( tmpndbf != NULL )
1030 {
1031     if( cours == tmpndbf->f->id )
1032     {
1033         printf( "
1034 *****\n" );
1035         printf( "* Liste des personnes
1036 *\n" );
1037         printf( "
1038 *****\n" );
1039         printf( "* Formation choisie: %-40s
1040 *\n", tmpndbf->f->nom );
1041         printf( "
1042 *****\n" );
1043         afficher_db_personne( tmpndbp );
1044         printf( "* 0 Retour
1045 *\n" );
1046         printf( "
1047 *****\n" );
1048         int p;
1049         printf( "* Qui voudriez vous attribuer a la formation : %s ? ", tmpndbf->f->nom );
1050         scanf( "%d", &p );
1051         getchar();
1052         while( p < 0 && p > tmpndbp->p->id )
1053         {
1054             printf( "* Valeur %d - INVALIDE\n", p );
1055             printf( "* Qui voudriez vous attribuer a la formation : %s ? ", tmpndbf->f->
1056 nom );
1057             scanf( "%d", &p );
1058             getchar();
1059         }
1060         if( p <= 0 )
1061         {
1062             system( clear );
1063             return;
1064         }
1065         while ( tmpndbp != NULL )
1066         {
1067             if ( p == tmpndbp->p->id )
1068             {
1069                 char confirmation[4];
1070                 printf( "* Etes vous sur de vouloir attribuer %s %s a la formation %s ? (o
1071 /n) ",
1072                     tmpndbp->p->nom, tmpndbp->p->prenom, tmpndbf->f->nom );
1073                 scanf( "%s", confirmation );
1074                 while( strcmp( confirmation, "o" ) != 0 && strcmp( confirmation, "oui" )
1075 != 0 &&
1076 strcmp( confirmation, "n" ) != 0 && strcmp( confirmation, "non" )
1077 != 0 )
1078                 {
1079                     printf( "Veuillez inserer o / oui - n / non : " );
1080                     scanf( "%s", confirmation );
1081                 }
1082                 if( strcmp( confirmation, "o" ) == 0 || strcmp( confirmation, "oui" ) == 0
1083 )
1084                 {
1085                     int res = ajouter_formation( tmpndbf->f, tmpndbp->p );
1086                     if ( res == 1 )
1087                     {
1088                         tmpndbp->p->nb_formation += 1;
1089                         tmpndbp->p->formations[ tmpndbp->p->nb_formation - 1 ] = tmpndbf
1090 ->f->id;
1091                         system( clear );
1092                         printf( "* %s %s a ete attribue(e) a la formation %s avec succes
1093 *\n",
1094                             tmpndbp->p->nom, tmpndbp->p->prenom, tmpndbf->f->nom );
1095                         break;
1096                     }
1097                     system( clear );
1098                     printf( "* %s %s est deja present dans la formation %s *\n",
1099                             tmpndbp->p->nom, tmpndbp->p->prenom, tmpndbf->f->nom );
1100                     break;
1101                 }
1102             }
1103             else
1104             {
1105                 system( clear );
1106             }
1107         }
1108     }
1109 }
1110

```

```

1091             printf( "* %s %s n'a PAS ete attribue(e) a la formation %s *\n" ,
1092                     tmpndbp->p->nom, tmpndbp->p->prenom, tmpndbf->f->nom );
1093         }
1094     }
1095     tmpndbp = tmpndbp->next;
1096 }
1097 break;
1098 }
1099 tmpndbf = tmpndbf->next;
1100 }
1101 }
1102
1103 void menu_supprimer_personne( db_formation *dbf, db_personne *dbp )
1104 {
1105     int idp;
1106     db_formation *tmpdbf = dbf;
1107     noeud_db_formation *tmpndbf = tmpdbf->head;
1108     db_personne *tmpdbp = dbp;
1109     noeud_db_personne *tmpndbp = tmpdbp->head;
1110     afficher_db_personne( tmpdbp );
1111     printf( "* 0 Retour\n" );
1112     printf( "* Quelle personne voudriez vous supprimer entierement ? " );
1113     scanf( "%d", &idp );
1114     getchar();
1115     while( idp < 0 && idp > tmpndbp->p->id )
1116     {
1117         printf( "* Option %d - INVALIDE\n", idp );
1118         printf( "* Quelle personne voudriez vous supprimer entierement ? " );
1119         scanf( "%d", &idp );
1120         getchar();
1121     }
1122     if( idp <= 0 )
1123     {
1124         system( clear );
1125         return;
1126     }
1127     while( tmpndbf != NULL )
1128     {
1129         formation *tmpf = tmpndbf->f;
1130         noeud_formation *tmpnf = tmpf->head;
1131         while( tmpnf != NULL )
1132         {
1133             if( tmpnf->p->id == idp )
1134             {
1135                 supprimer_personne_de_formation( tmpf, tmpnf->p->id );
1136                 break;
1137             }
1138             tmpnf = tmpnf->next;
1139         }
1140         tmpndbf = tmpndbf->next;
1141     }
1142     tmpndbf = tmpdbf->head;
1143     tmpndbp = tmpdbp->head;
1144     while( tmpndbp != NULL )
1145     {
1146         if( tmpndbp->p->id == idp )
1147         {
1148             char nom[40], prenom[40];
1149             strcpy( nom, tmpndbp->p->nom );
1150             strcpy( prenom, tmpndbp->p->prenom );
1151             char confirmation[4];
1152             printf( "* Etes vous sur de vouloir supprimer %s %s entierement de la base de
1153 donnees ? (o/n) ",
1154                     nom, prenom );
1155             scanf( "%s", confirmation );
1156             while( strcmp( confirmation, "o" ) != 0 && strcmp( confirmation, "oui" ) != 0 &&
1157                     strcmp( confirmation, "n" ) != 0 && strcmp( confirmation, "non" ) != 0 )
1158             {
1159                 printf( "Veuillez inserer o / oui - n / non : " );
1160                 scanf( "%s", confirmation );
1161             }
1162             if( strcmp( confirmation, "o" ) == 0 || strcmp( confirmation, "oui" ) == 0 )
1163             {
1164                 supprimer_db_personne( tmpdbp, idp );
1165                 system( clear );

```

```

1165         printf( "* %s %s a ete supprime(e) entierement de la base de donnees *\n",
1166                 nom, prenom );
1167     }
1168     else
1169     {
1170         system( clear );
1171         printf( "* %s %s n'a PAS ete supprimer de la base de donnees *\n",
1172                 nom, prenom );
1173     }
1174     break;
1175 }
1176 tmpndbp = tmpndbp->next;
1177 }
1178 tmpndbf = tmpndbf->head;
1179 tmpndbp = tmpndbp->head;
1180 }
1181
1182 void menu_supprimer_formation( db_formation *dbf, db_personne *dbp )
1183 {
1184     int idf;
1185     db_formation *tmpndbf = dbf;
1186     noeud_db_formation *tmpndbf = tmpndbf->head;
1187     db_personne *tmpndbp = dbp;
1188     noeud_db_personne *tmpndbp = tmpndbp->head;
1189     printf( "*****\n" );
1190     printf( "* MENU SUPPRESSION : Liste des formations *\n" );
1191     printf( "*****\n" );
1192     printf( "* %2s %-40s *\n", "ID", "Nom" );
1193     printf( "* ----- *\n" );
1194     while( tmpndbf != NULL )
1195     {
1196         formation *tmpf = tmpndbf->f;
1197         printf( "* %2d %-40s *\n", tmpf->id, tmpf->nom );
1198         tmpndbf = tmpndbf->next;
1199     }
1200     tmpndbf = tmpndbf->head;
1201     printf( "* 0 Retour *\n" );
1202     printf( "*****\n" );
1203     printf( "* Quelle formation voudriez vous supprimer? " );
1204     scanf( "%d", &idf );
1205     getchar();
1206     printf( "*****\n" );
1207     while( idf < 0 && idf > tmpndbf->f->id )
1208     {
1209         printf( "* Option %d - INVALIDE\n", idf );
1210         printf( "* Quelle formation voudriez vous supprimer? " );
1211         scanf( "%d", &idf );
1212         getchar();
1213     }
1214     if( idf <= 0 )
1215     {
1216         system( clear );
1217         return;
1218     }
1219     while( tmpndbf != NULL )
1220     {
1221         formation *tmpf = tmpndbf->f;
1222         if( idf == tmpf->id )
1223         {
1224             char confirmation[4];
1225             printf( "* Etes vous sur de vouloir supprimer %s entierement de la base de donnees\n" );
1226                 tmpf->nom );
1227             scanf( "%s", confirmation );
1228             while( strcmp( confirmation, "o" ) != 0 && strcmp( confirmation, "oui" ) != 0 &&
1229                 strcmp( confirmation, "n" ) != 0 && strcmp( confirmation, "non" ) != 0 )
1230             {
1231                 printf( "Veuillez inserer o / oui - n / non : " );
1232                 scanf( "%s", confirmation );

```

```

1233     }
1234     if( strcmp( confirmation, "o" ) == 0 || strcmp( confirmation, "oui" ) == 0 )
1235     {
1236         supprimer_db_formation( tmpdbf, idf );
1237         while ( tmpndbf != NULL )
1238         {
1239             personne *tmpp = tmpndbf->p;
1240             int k;
1241             for ( k = 0; k < tmpp->nb_formation - 1; k++ )
1242             {
1243                 if ( tmpp->formation[ k ] == tmpf->id )
1244                 {
1245                     int l;
1246                     for ( l = k; l < tmpp->nb_formation; l++ )
1247                     {
1248                         tmpp->formation[ l ] = tmpp->formation[ l + 1 ];
1249                     }
1250                     tmpp->nb_formation -= 1;
1251                     break;
1252                 }
1253             }
1254             tmpndbf = tmpndbf->next;
1255         }
1256         system( clear );
1257         printf( "* %s a ete supprimee de la base de donnees *\n", tmpf->nom );
1258     }
1259     else
1260     {
1261         system( clear );
1262         printf( "* %s n'a PAS ete supprimee de la base de donnees *\n", tmpf->nom );
1263     }
1264     break;
1265 }
1266 tmpdbf = tmpdbf->next;
1267 }
1268 tmpdbf = tmpdbf->head;
1269 }
1270
1271 int menu_supprimer_personne_de_formation( db_formation *dbf )
1272 {
1273     int idf;
1274     db_formation *tmpdbf = dbf;
1275     noeud_db_formation *tmpndbf = tmpdbf->head;
1276     printf( "*****\n" );
1277     printf( "* MENU SUPPRESSION : Liste des formations *\n" );
1278     printf( "*****\n" );
1279     while ( tmpndbf != NULL )
1280     {
1281         formation *tmpf = tmpndbf->f;
1282         printf( "* %2d %-40s *\n", tmpf->id, tmpf->nom );
1283         tmpndbf = tmpndbf->next;
1284     }
1285     tmpndbf = tmpdbf->head;
1286     printf( "* 0 Retour *\n" );
1287     printf( "* De quelle formation voudriez vous supprimer quelqu'un ? " );
1288     scanf( "%d", &idf );
1289     getchar();
1290     while( idf < 0 && idf > tmpndbf->f->id )
1291     {
1292         printf( "* Option %d - INVALIDE\n", idf );
1293         printf( "* De quelle formation voudriez vous supprimer quelqu'un ? " );
1294         scanf( "%d", &idf );
1295         getchar();
1296     }
1297     if ( idf <= 0 )
1298     {
1299         system( clear );
1300         return 0;
1301     }
1302     while( tmpndbf != NULL )
1303     {
1304         if( idf == tmpndbf->f->id )

```

```

1305     {
1306         printf( "Cours choisi: %s\n", tmpndbf->f->nom );
1307         formation *tmpf = tmpndbf->f;
1308         noeud_formation *tmpnf = tmpf->head;
1309         if( tmpnf == NULL )
1310             {
1311                 system( clear );
1312                 printf( "* /\n La formation est vide /\n\n" );
1313                 return 0;
1314             }
1315         while( tmpnf != NULL )
1316             {
1317                 personne *tmpp = tmpnf->p;
1318                 printf( "* %2d %-25s %-25s %-10s          *\n",
1319                     tmpp->id, tmpp->nom, tmpp->prenom, tmpp->formateur ? "Formateur" : "
Etudiant" );
1320                 tmpnf = tmpnf->next;
1321             }
1322         tmpnf = tmpf->head;
1323         printf( "* 0 Retour
*\n" );
1324         int idp;
1325         printf( "* Quelle personne voudriez vous supprimer de cette formation ? " );
1326         scanf( "%d", &idp );
1327         getchar();
1328         while( idp < 0 && idp > tmpnf->p->id )
1329             {
1330                 printf( "* Option %d - INVALIDE\n", idp );
1331                 printf( "* Quelle personne voudriez vous supprimer de cette formation ? " );
1332                 scanf( "%d", &idp );
1333                 getchar();
1334             }
1335         if( idp <= 0 )
1336             {
1337                 system( clear );
1338                 return 0;
1339             }
1340         while( tmpnf != NULL )
1341             {
1342                 personne *tmpp = tmpnf->p;
1343                 if( tmpp->id == idp )
1344                     {
1345                         char confirmation[4];
1346                         printf( "* Etes vous sur de vouloir supprimer %s %s de la formation %s ? (
o/n) ",
1347                             tmpp->nom, tmpp->prenom, tmpf->nom );
1348                         scanf( "%s", confirmation );
1349                         while( strcmp( confirmation, "o" ) != 0 && strcmp( confirmation, "oui" )
!= 0 &&
1350                             strcmp( confirmation, "n" ) != 0 && strcmp( confirmation, "non" )
!= 0 )
1351                             {
1352                                 printf( "Veuillez inserer o / oui - n / non : " );
1353                                 scanf( "%s", confirmation );
1354                             }
1355                         if( strcmp( confirmation, "o" ) == 0 || strcmp( confirmation, "oui" ) == 0
)
1356                             {
1357                                 supprimer_personne_de_formation( tmpf, tmpp->id );
1358                                 int k;
1359                                 for ( k = 0; k < tmpp->nb_formation - 1; k++ )
1360                                     {
1361                                         if ( tmpp->formations[ k ] == tmpf->id )
1362                                             {
1363                                                 int l;
1364                                                 for ( l = k; l < tmpp->nb_formation; l++ )
1365                                                     {
1366                                                         tmpp->formations[ l ] = tmpp->formations[ l + 1 ];
1367                                                     }
1368                                                         tmpp->nb_formation -= 1;
1369                                                         break;
1370                                                     }
1371                                 }
1372                                 system( clear );
1373                                 printf( "* %s %s a ete supprime de la formation %s avec succes *\n",

```

```

1374             tmpf->nom, tmpf->prenom, tmpf->nom );
1375         printf( "\n\n" );
1376         afficher_formation( tmpf );
1377         return 1;
1378     }
1379     else
1380     {
1381         system( clear );
1382         printf( "* %s %s n'a PAS ete supprime de la formation %s *\n",
1383             tmpf->nom, tmpf->prenom, tmpf->nom );
1384     }
1385 }
1386 tmpnf = tmpnf->next;
1387 }
1388 tmpnf = tmpf->head;
1389 }
1390 tmpndbf = tmpndbf->next;
1391 }
1392 tmpndbf = tmpdbf->head;
1393 return 0;
1394 }
1395
1396 int menu_supprimer( db_formation *dbf, db_personne *dbp )
1397 {
1398     int choix;
1399     do
1400     {
1401         db_formation *tmpdbf = dbf;
1402         db_personne *tmpdbp = dbp;
1403         printf( "
1404 *****\n" );
1405         printf( "* MENU SUPPRESSION
1406 *\n" );
1407         printf( "
1408 *****\n" );
1409         printf( "* 1. Supprimer une personne de la base de donnees
1410 *\n" );
1411         printf( "* 2. Supprimer une formation de la base de donnees
1412 *\n" );
1413         printf( "* 3. Supprimer une personne specifique d'une formation specifique
1414 *\n" );
1415         printf( "* 0. Retour
1416 *\n" );
1417         printf( "
1418 *****\n" );
1419         printf( "* Que voudriez vous supprimer ? " );
1420         scanf( "%d", &choix );
1421         getchar();
1422         switch( choix )
1423         {
1424             case 1:
1425                 menu_supprimer_personne( tmpdbf, tmpdbp );
1426                 break;
1427             case 2:
1428                 menu_supprimer_formation( tmpdbf, tmpdbp );
1429                 break;
1430             case 3:
1431                 menu_supprimer_personne_de_formation( tmpdbf );
1432                 break;
1433             case 0:
1434                 system( clear );
1435                 break;
1436             default:
1437                 system( clear );
1438                 break;
1439         }
1440     } while( choix != 0 );
1441     return 0;
1442 }
1443
1444 int menu_affichage( db_formation *f, db_personne *p )
1445 {
1446     int choix;
1447     do
1448     {
1449         db_formation *tmpdbf = f;

```



```

1442     db_personne *tmpdbp = p;
1443     noeud_db_formation *tmpndbf = tmpdbf->head;
1444     printf( "
*****\n" );
1445     printf( "* MENU AFFICHAGE
*\n" );
1446     printf( "
*****\n" );
1447     printf( "* 1. Liste des personnes
*\n" );
1448     printf( "* 2. Liste des formations
*\n" );
1449     printf( "* 3. Planning de la semaine
*\n" );
1450     printf( "* 0. Retour
*\n" );
1451     printf( "
*****\n" );
1452     printf( "* Que voudriez-vous afficher ? " );
1453     scanf( "%d", &choix );
1454     getchar();
1455     switch( choix )
1456     {
1457         case 1:
1458             system( clear );
1459             afficher_db_personne( tmpdbp );
1460             break;
1461         case 2:
1462             system( clear );
1463             printf( "* %2s %-40s %-6s          *\n", "ID", "Nom", "Prix"
);
1464             printf( "*
----- *\n" );
1465             while( tmpndbf != NULL )
1466             {
1467                 printf( "* %2d %-40s %6.2f          *\n",
1468                     tmpndbf->f->id, tmpndbf->f->nom, tmpndbf->f->prix );
1469                 tmpndbf = tmpndbf->next;
1470             }
1471             tmpndbf = tmpdbf->head;
1472             break;
1473         case 3:
1474             system( clear );
1475             afficher_db_formation( tmpdbf );
1476             break;
1477         case 0:
1478             system( clear );
1479             break;
1480         default:
1481             system( clear );
1482             printf( "/!\ Option %d - INVALIDE /!\n", choix );
1483             break;
1484     }
1485 } while( choix != 0 );
1486 return 0;
1487 }
1488
1489 void ecrire_planning( db_formation *dbf )
1490 {
1491     FILE *fres = fopen( "CaculliTyranowski.res", "w" );
1492     int i;
1493     db_formation *tmpdbf = dbf;
1494     noeud_db_formation *tmpndbf = tmpdbf->head;
1495     char jour[7][9] = { "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"
};
1496     for( i = 1; i <= 7; i++ )
1497     {
1498         tmpndbf = tmpdbf->head;
1499         fprintf( fres, "
*****\n" );
1500         fprintf( fres, "Cours du: %s\n", jour[i - 1] );
1501         fprintf( fres, "
*****\n" );
1502         while( tmpndbf != NULL )
1503         {
1504             formation *tmp = tmpndbf->f;

```

```

1505         int j;
1506         for( j = 0; j < tmp->nb_jours; j++ )
1507         {
1508             if( tmp->jours[j] == i )
1509             {
1510                 formation *tmp = tmpndbf->f;
1511                 noeud_formation *tmpnf = tmp->head;
1512                 fprintf( fres, "ID: %d - Nom formation: %s\n", tmp->id, tmp->nom );
1513                 fprintf( fres, "Participants dans la formation:\n" );
1514                 fprintf( fres, "Formateurs:\n" );
1515                 while( tmpnf != NULL )
1516                 {
1517                     if( tmpnf->p->formateur == 1 )
1518                     {
1519                         fprintf( fres, "%2d %s %s", tmpnf->p->id, tmpnf->p->nom, tmpnf->p
1520                             ->prenom );
1521                         if( tmpnf != NULL ) fprintf( fres, "\n" );
1522                         tmpnf = tmpnf->next;
1523                     }
1524                     tmpnf = tmp->head;
1525                     fprintf( fres, "Etudiants:\n" );
1526                     while( tmpnf != NULL )
1527                     {
1528                         if( tmpnf->p->formateur == 0 )
1529                         {
1530                             fprintf( fres, "%2d %s %s", tmpnf->p->id, tmpnf->p->nom, tmpnf->p
1531                                 ->prenom );
1532                             if( tmpnf != NULL ) fprintf( fres, "\n" );
1533                             tmpnf = tmpnf->next;
1534                         }
1535                         fprintf( fres, "\n" );
1536                         fprintf( fres, "De: %.2f - A %.2f\n", tmp->heures[j], tmp->heures[j] + tmp
1537                             ->durees[j] );
1538                         fprintf( fres, "Prerequis: " );
1539                         if( tmp->nb_prerequis > 0 )
1540                         {
1541                             db_formation *tmpdb = dbf;
1542                             noeud_db_formation *tmp_prereq_ndbf = tmpdb->head;
1543                             while( tmp_prereq_ndbf != NULL )
1544                             {
1545                                 formation *tmp_prereq = tmp_prereq_ndbf->f;
1546                                 int k;
1547                                 for( k = 0; k < tmp->nb_prerequis; k++ )
1548                                 {
1549                                     if( tmp->prerequis[k] == tmp_prereq->id )
1550                                     {
1551                                         fprintf( fres, "%s ", tmp_prereq->nom );
1552                                     }
1553                                     tmp_prereq_ndbf = tmp_prereq_ndbf->next;
1554                                 }
1555                                 fprintf( fres, "\n\n" );
1556                             }
1557                             else
1558                             {
1559                                 fprintf( fres, "Aucun\n\n" );
1560                             }
1561                         }
1562                     }
1563                     tmpndbf = tmpndbf->next;
1564                 }
1565             }
1566             fclose( fres );
1567 }
1568
1569 /*
1570  * Menu permettant a l'utilisateur d'interagir avec le programme
1571  */
1572 int menu( db_formation *f, db_personne *p )
1573 {
1574     int choix;
1575     do
1576     {
1577         printf( "

```

```

*****\n" );
1578     printf( "* MENU PRINCIPALE
        *\n" );
1579     printf( "
*****\n" );
1580     db_formation *tmpdbf = f;
1581     db_personne *tmpdbp = p;
1582     printf( "* 1: Afficher la liste des etudiants/formateurs ou la liste des formations
        *\n" );
1583     printf( "* 2: Ajouter une nouvelle personne ou formation a la base de donnees
        *\n" );
1584     printf( "* 3: Attribuer une personne a une formation
        *\n" );
1585     printf( "* 4: Supprimer une formation, une personne ou une personne d'une formation
        *\n" );
1586     printf( "* 0: Quitter le programme
        *\n" );
1587     printf( "
*****\n" );
1588     printf( "* Que voudriez-vous faire ? " );
1589     scanf( "%d", &choix );
1590     getchar();
1591     switch ( choix )
1592     {
1593     case 1:
1594         system( clear );
1595         menu_affichage( tmpdbf, tmpdbp );
1596         break;
1597     case 2:
1598         system( clear );
1599         menu_creer( tmpdbf, tmpdbp );
1600         break;
1601     case 3:
1602         system( clear );
1603         menuajouter_formation( tmpdbf, tmpdbp );
1604         break;
1605     case 4:
1606         system( clear );
1607         menu_supprimer( tmpdbf, tmpdbp );
1608         break;
1609     case 0:
1610         printf( "Voulez vous sauvegarder les changements ? (o/n) " );
1611         char choix_sauvegarde[4];
1612         scanf( "%s", choix_sauvegarde );
1613         if( strcmp( choix_sauvegarde, "o" ) == 0 || strcmp( choix_sauvegarde, "oui" )
== 0 )
1614         {
1615             FILE *fdat_f = fopen( "CaculliTyranowskiFormation.dat", "w" );
1616             FILE *fdat_p = fopen( "CaculliTyranowskiPersonne.dat", "w" );
1617             noeud_db_formation *tmpndbf = tmpdbf->head;
1618             noeud_db_personne *tmpndbp = tmpdbp->head;
1619             db_formation *out_f = creer_db_formation();
1620             while ( tmpndbf != NULL )
1621             {
1622                 ajouter_db_formation( out_f, tmpndbf->f );
1623                 tmpndbf = tmpndbf->next;
1624             }
1625             tmpndbf = out_f->head;
1626             db_personne *out_p = creer_db_personne();
1627             while ( tmpndbp != NULL )
1628             {
1629                 ajouter_db_personne( out_p, tmpndbp->p );
1630                 tmpndbp = tmpndbp->next;
1631             }
1632             tmpndbp = out_p->head;
1633             while ( tmpndbp != NULL )
1634             {
1635                 personne *tmpp = tmpndbp->p;
1636                 fprintf( fdat_p, "%02d %-24s %-24s %d %d ",
1637                         tmpp->id, tmpp->nom, tmpp->prenom, tmpp->formateur, tmpp->
nb_formation );
1638                 int i;
1639                 for ( i = 0; i < tmpp->nb_formation; i++ )
1640                 {
1641                     fprintf( fdat_p, "%d ", tmpp->formations[ i ] );
1642

```

```

1643         if ( tmppp->formateur == 0 )
1644         {
1645             fprintf( fdat_p, "    %d    ", tmppp->reduction );
1646             if ( tmppp->reduction > 0 )
1647             {
1648                 fprintf( fdat_p, "%d", tmppp->val_reduction );
1649             }
1650         } else
1651         {
1652             fprintf( fdat_p, "    %d    ", tmppp->nb_jours_indisponible );
1653             for ( i = 0; i < tmppp->nb_jours_indisponible; i++ )
1654             {
1655                 fprintf( fdat_p, "%d ", tmppp->jours_indisponible[ i ] );
1656             }
1657         }
1658         fprintf( fdat_p, "\n" );
1659         tmpndbp = tmpndbp->next;
1660     }
1661     tmpndbp = tmpdbp->head;
1662     while ( tmpndbf != NULL )
1663     {
1664         int i;
1665         formation *tmpf = tmpndbf->f;
1666         fprintf( fdat_f, "%02d %d ", tmpf->id, tmpf->nb_prerequis );
1667         if ( tmpf->nb_prerequis > 0 )
1668         {
1669             for ( i = 0; i < tmpf->nb_prerequis; i++ )
1670             {
1671                 fprintf( fdat_f, "%d ", tmpf->prerequis[ i ] );
1672             }
1673             fprintf( fdat_f, "%d    ", tmpf->nb_jours );
1674         } else
1675         {
1676             fprintf( fdat_f, "    %d    ", tmpf->nb_jours );
1677         }
1678         for ( i = 0; i < tmpf->nb_jours; i++ )
1679         {
1680             fprintf( fdat_f, "%d    %.2f    %.2f    ",
1681                     tmpf->jours[ i ], tmpf->heures[ i ], tmpf->durees[ i ] );
1682         }
1683         fprintf( fdat_f, "%.2f %-s\n", tmpf->prix, tmpf->nom );
1684         tmpndbf = tmpndbf->next;
1685     }
1686     tmpndbf = tmpdbf->head;
1687     fclose( fdat_f );
1688     fclose( fdat_p );
1689     ecrire_planning( tmpdbf );
1690     printf( "Changements sauvegardes!\n" );
1691 }
1692 printf( "Fermeture du programme...\n" );
1693 printf( "Au revoir!\n" );
1694 break;
1695 default:
1696     system( clear );
1697     printf( "/!\ \ Option %d - INVALIDE /!\ \n", choix );
1698     break;
1699 }
1700 } while ( choix != 0 );
1701 return 0;
1702 }
1703
1704 int main( void )
1705 {
1706     system( clear );
1707     printf( "Projet par Giorgio Caculli et Jedrzej Tyranowski\n" );
1708
1709     FILE *fdat_f = fopen( "CaculliTyranowskiFormation.dat", "r" );
1710     FILE *fdat_p = fopen( "CaculliTyranowskiPersonne.dat", "r" );
1711
1712     db_personne *dbp = creer_db_personne();
1713     db_formation *dbf = creer_db_formation();
1714
1715     int i = 1;
1716
1717     while( !feof( fdat_p ) )
1718     {

```

```

1719     char nom[25], prenom[25];
1720     int formateur;
1721     int id;
1722     fscanf( fdat_p, "%d %24s %24s %d", &id, nom, prenom, &formateur );
1723     if( feof( fdat_p ) )
1724     {
1725         break;
1726     }
1727     personne *tmp = creer_personne( nom, prenom, formateur );
1728     fscanf( fdat_p, "%d", &tmp->nb_formation );
1729     tmp->id = id;
1730     int j;
1731     for( j = 0; j < tmp->nb_formation; j++ )
1732     {
1733         fscanf( fdat_p, "%d", &tmp->formation[j] );
1734     }
1735     if( formateur == 0 )
1736     {
1737         fscanf( fdat_p, "%d", &tmp->reduction );
1738         if( tmp->reduction == 1 )
1739         {
1740             fscanf( fdat_p, "%d", &tmp->val_reduction );
1741         }
1742     }
1743     else if( formateur == 1 )
1744     {
1745         fscanf( fdat_p, "%d", &tmp->nb_jours_indisponible );
1746         for( j = 0; j < tmp->nb_jours_indisponible; j++ )
1747         {
1748             fscanf( fdat_p, "%d", &tmp->jours_indisponible[j] );
1749         }
1750     }
1751     ajouter_db_personne( dbp, tmp );
1752     i += 1;
1753 }
1754
1755 i = 1;
1756
1757 while( !feof( fdat_f ) )
1758 {
1759     int id;
1760     int nb_prerequis, nb_jours;
1761     float prix;
1762     char nom_formation[40];
1763     fscanf( fdat_f, "%d %d", &id, &nb_prerequis );
1764     int prerequis[ nb_prerequis ];
1765     int j;
1766     for( j = 0; j < nb_prerequis; j++ )
1767     {
1768         fscanf( fdat_f, "%d", &prerequis[j] );
1769     }
1770     fscanf( fdat_f, "%d", &nb_jours );
1771     int jours[ nb_jours ];
1772     float heures[ nb_jours ];
1773     float durees[ nb_jours ];
1774     for( j = 0; j < nb_jours; j++ )
1775     {
1776         fscanf( fdat_f, "%d %f %f", &jours[j], &heures[j], &durees[j] );
1777     }
1778     fscanf( fdat_f, "%f ", &prix );
1779     fgets( nom_formation, 40, fdat_f );
1780     if( feof( fdat_f ) )
1781     {
1782         break;
1783     }
1784     if( strlen( nom_formation ) > 0 && nom_formation[ strlen( nom_formation ) - 1 ] == '\n'
, )
    {
1785         nom_formation[ strlen( nom_formation ) - 1 ] = '\0';
1786     }
1787     formation *tmp = creer_formation( nom_formation, prix );
1788     tmp->id = id;
1789     tmp->nb_prerequis = nb_prerequis;
1790     for( j = 0; j < tmp->nb_prerequis; j++ )
1791     {
1792         tmp->prerequis[j] = prerequis[j];
1793     }

```

```

1794     }
1795     tmp->nb_jours = nb_jours;
1796     for( j = 0; j < tmp->nb_jours; j++ )
1797     {
1798         tmp->jours[j] = jours[j];
1799         tmp->heures[j] = heures[j];
1800         tmp->durees[j] = durees[j];
1801     }
1802     ajouter_db_formation( dbf, tmp );
1803     i += 1;
1804 }
1805
1806 db_personne *tmpdbp = dbp;
1807 noeud_db_personne *tmpndbp = tmpdbp->head;
1808
1809 while( tmpndbp != NULL )
1810 {
1811     db_formation *tmpdbf = dbf;
1812     noeud_db_formation *tmpndbf = tmpdbf->head;
1813     while( tmpndbf != NULL )
1814     {
1815         int j;
1816         for( j = 0; j < tmpndbp->p->nb_formation; j++ )
1817         {
1818             if( tmpndbp->p->formations[j] == tmpndbf->f->id )
1819             {
1820                 ajouter_formation( tmpndbf->f, tmpndbp->p );
1821             }
1822         }
1823         tmpndbf = tmpndbf->next;
1824     }
1825     tmpndbp = tmpndbp->next;
1826 }
1827
1828 fclose( fdat_p );
1829 fclose( fdat_f );
1830
1831 menu( dbf, dbp );
1832
1833 return 0;
1834 }

```

Glossaire

ANSI American National Standard Institute. 3

RAM Random Access Memory. 5, 24