



Rendu TP : TP1 SDD Agenda

BALLEJOS Lilian
LEGER Bertrand
Isima INP
Année Universitaire 2021 - 2022

Enseignant TP : CROMBEZ Loic
Date du rendu : 7 mars 2022

Sommaire

1	Présentation du TP	1
2	Présentation de la structure	2
2.1	Schéma de la structure	2
2.2	Format des fichiers en entrée et en sortie	2
3	Présentation des Fonctions	3
3.1	tete.h et tete.c	3
3.2	semaine.h et semaine.c	4
3.3	action.h et action.c	4
3.4	commun.h et commun.c	5
3.4.1	Lecture et Ecriture dans les fichiers	5
3.4.2	Gestion d'allocation	5
3.4.3	Ajout et Suppression	6
3.4.4	Affichage	6
3.5	contigue.h et contigue.c	7
4	Liste des cas	7
4.1	Lecture	8
4.2	Ajout	9
4.3	Suppression	11
4.4	Contiguë	15
5	Annexe	17

1 Présentation du TP

Ce TP a pour but d'implémenter une liste simplement chaînée qui représente un pseudo Agenda. Nous avons ici séparé notre structure de donnée en 3 sous structures :

- La structure **"tête"** qui est la tête fictive qui pointe sur la première semaine,
- La structure **"semaine"** qui contient un numéro de semaine, une année (toutes concaténées dans la même chaîne de caractère), un pointeur sur la semaine suivante ainsi qu'un pointeur sur la liste des actions qui ont lieu cette semaine,
- La structure **"action"** qui possède un numéro correspondant à un jour de la semaine, une heure (le tout concaténées dans la même chaîne de caractère), le nom de l'action et enfin un pointeur vers l'action suivante.

Nous avons créé un fichier source ".c" par structure (tete.c, action.c, semaine.c), un fichier "commun.c" qui contient toutes les fonctions en lien avec l'ensemble des 3 structures et enfin un fichier main.c qui contient le main.

Tous les fichiers sources sont stockés dans le dossier "source", les fichiers "headers" dans le dossier header, les fichiers objets dans le dossier "objet" et enfin les entrées et sorties de notre programme dans les dossiers "entree" et "sauvegarde". Un makefile est fourni pour tout compiler et génère l'exécutable **"Programme"**.

2 Présentation de la structure

2.1 Schéma de la structure

Voici un schéma de la structure que nous avons mis en place :

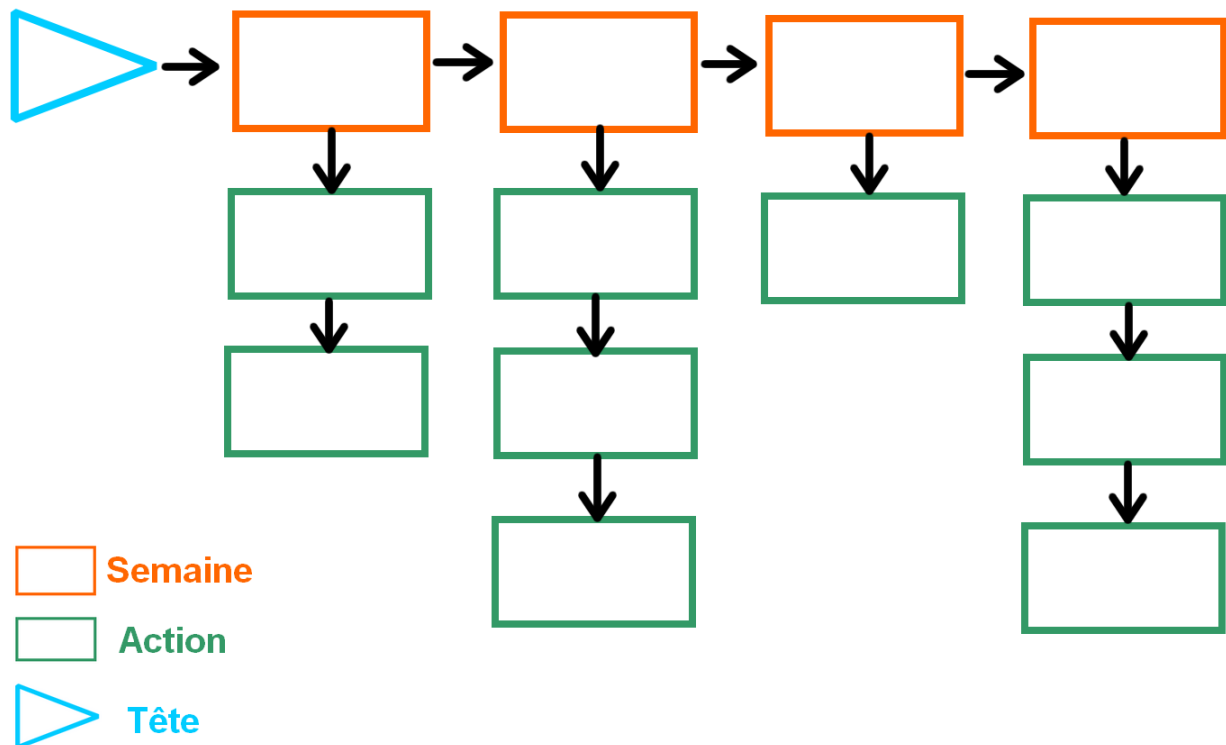


FIGURE 1 – Schéma de notre SDD "Agenda"

2.2 Format des fichiers en entrée et en sortie

Les formats des fichiers d'entrée et de sortie pour notre programme sont exactement les mêmes. Chaque ligne correspond à une activité et contient précisément dans cette ordre ces informations :

- une année sur 4 bytes
- un numéro de semaine sur 2 bytes
- un chiffre compris entre 1 et 7 correspondant à un jour (lundi, mardi ...)
- une heure sur 2 bytes (08 pour 8h par exemple)
- Un nom pour l'activité noté précisément sur 10 bytes (on complète avec des espaces si besoin)

Regardons maintenant un exemple de fichier d'entrée de notre programme et son fichier de sortie correspondant.

```
202215108TPs de SDD
202118109Devoir
202215110Apprenti
202215111TPs de SDD
202215112Boulot
202116113EPS
202215114TD de CSN
202315114Work Hard
202215106Boulot
202217106Tds de SDD
202116103EPS
202116713Muscu
202116813Docteur
```

FIGURE 2 – Notre fichier d'entrée

Comme dit avant les informations sont bien notées dans le bon ordre. Une chose tout de même à remarquer ici : dans cet exemple nous avons fait exprès de glisser une erreur dans la ligne "**Docteur**" : en effet le chiffre des jours qui est censé être compris entre 1 et 7 vaut ici 8 !

Regardons maintenant les fichiers de sortie obtenu :

```
202116103EPS
202116113EPS
202116713Muscu
202118109Devoir
202215106Boulot
202215108TPs de SDD
202215110Apprenti
202215111TPs de SDD
202215112Boulot
202215114TD de CSN
202217106Tds de SDD
202315114Work Hard
```

FIGURE 3 – Notre fichier de sortie

Après le passage dans notre programme les informations stockées dans notre fichier d'entrée on était triées par ordre croissant de leur date dans notre SDD ! Ainsi, à la sortie, toutes nos informations sont écrites dans l'ordre croissant dans notre fichier de sortie ! De plus on remarque que la valeur de "**Docteur**" étant mauvaise, notre programme ne l'a pas ajoutée dans la SDD et ainsi elle n'est pas dans le fichier de sortie !.

3 Présentation des Fonctions

Passons maintenant à la présentation de toutes nos fonctions.

Mais avant cela nous nous devons de faire une aparté : nous avons pris la décision dans ce TP, dans chaque structure contenant des chaînes de caractères de rajouter le caractère '`\0`' en fin de chaîne. En effet ce caractère est un élément essentiel du langage C et nous n'avons pas voulu nous en séparer. Sinon nous aurions été obligé de recoder toutes les fonctions de **string.h**. Ainsi dans chaque structure les chaînes de caractères sont sur un bit de plus afin de stocker ce dit caractère !

3.1 `tete.h` et `tete.c`

Le fichier **tete.h** possède la structure représentant notre tête fictive. Ainsi elle est seulement composée d'un pointeur pointant sur une structure de type "semaine".

```
1     typedef struct tete{
2         semaine * debut;
3     } tete;
4
```

FIGURE 4 – Structure "tete"

La seule fonction unique à la structure "tete" est **AlloueTete** qui comme son nom l'indique va allouer de la mémoire à notre tête fictive. La fonction prend en argument un pointeur indirect de type "tete" et renvoie 0 si tout va bien ou -1 si l'allocation a raté.

3.2 semaine.h et semaine.c

Passons à **semaine.h** qui contient la structure "semaine".

```
1     typedef struct semaine{
2         char annee_semaine[7];
3         struct action * liste_action;
4         struct semaine * suivant;
5     } semaine;
6
```

FIGURE 5 – Structure "semaine"

Comme avec "tete", la seule fonction en lien avec cette structure est **AlloueSemaine** qui a pour but d'allouer de la mémoire à un élément de type "semaine". Cette fonction prend en argument un pointeur indirect sur une structure de type semaine, une string contenant une année et une autre string avec un numéro de semaine. Elle va tout de même contrôler si jamais les informations qu'on lui a donné en paramètre sont valides (numéro de semaine bien compris entre 1 et 53). Elle renvoie 0 si tout c'est bien passé et -1 si l'allocation a raté ou les valeurs entrées sont mauvaises. L'année ainsi que le numéro de semaine sont concaténés dans le champs **annee_semaine**, cela nous évitera des tests en trop dans de futures fonctions.

3.3 action.h et action.c

Enfin notre dernière structure est la structure "**action**" dont voici la forme :

```
1     typedef struct action{
2         char jour_heure[4];
3         char nom[11];
4         struct action * suivant;
5     } action;
6
```

FIGURE 6 – Structure "action"

La fonction en lien avec celle-ci est **AlloueAction** qui va allouer de la mémoire pour une case action. Elle prend en argument un pointeur indirect sur une action ainsi que 3 strings possédants respectivement, une valeur entre 1 et 7 pour le jour, une heure et un nom. La fonction va vérifier si le jour ainsi que l'heure sont valides ! Si tout c'est bien passé (allocation réussie et champs valides) la fonction renvoi 0 sinon elle renvoi -1.

3.4 commun.h et commun.c

Passons à toutes les fonctions ayant un impact sur toute notre SDD !

3.4.1 Lecture et Ecriture dans les fichiers

Commençons avec les gestions d'entrée et sortie :

La fonction "**LectureFichier**" va tenter d'ouvrir le fichier en lien avec le nom qu'il reçoit en paramètre. Si cela est possible il va lire tout son contenu et stocker dans différentes strings les informations qu'il en tire. Une fois à la fin d'une ligne, on oublie pas d'enlever le caractère '\n' afin de passer à la ligne suivante puis on appelle la fonction **AjoutStruct** en lui donnant toutes les informations que l'on vient de récupérer. Une fois cela fait, on ferme le fichier. La fonction renvoie 1 si la lecture c'est bien passé, 0 sinon.

La fonction "**EcritureFichier**" va elle prendre en argument un nom de fichier et la tête fictive de notre SDD. Si on a pas de problème d'ouverture de fichier, on va parcourir entièrement notre SDD. C'est à dire semaine après semaine tout en parcourant toutes les actions de chaque semaine. Elle va, à chaque action, rajouter une ligne dans notre fichier avec les informations correspondant à celle-ci. On fait bien attention de rajouter un '\n' à chaque fin de ligne sauf la dernière pour respecter le format standard de nos fichiers. Grâce à cela un fichier produit par notre programme pourra être bien évidemment mis ensuite en entrée !

3.4.2 Gestion d'allocation

Voyons maintenant toutes les fonctions utilisant des allocations ou des libérations de mémoire !

La fonction "**CreationElement**" est une fonction outil qui prend en argument 5 strings avec toutes les informations possibles d'une activité (année, numéro de semaine, jour, heure, nom) ainsi qu'un pointeur indirect sur une semaine. Il va allouer en mémoire la semaine ainsi que l'action correspondant aux informations données et les relier entre eux ! La fonction renvoie 1 si tout s'est bien passé, 0 si une allocation a ratée et -1 si les 2 ont ratées !

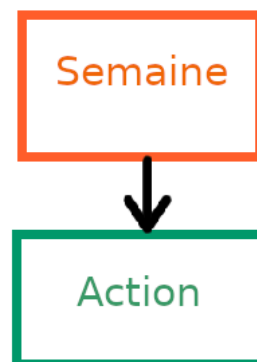


FIGURE 7 – Représentation de l'élément crée

La fonction "**LiberationElement**" fais l'inverse, elle libère un élément de la forme semaine pointant vers action. Le duo de ces des fonctions va nous faciliter la tâche lors de l'utilisation

de la fonction **"SupprStruct"**.

La fonction **"LibererStruct"** va entièrement libérer toute notre SDD. On lui donne simplement en argument notre tête fictive. Elle va parcourir l'intégralité de la SDD et libérer au fur et à mesure toutes les actions et les semaines. AU final on oublie pas de libérer la tête fictive !

3.4.3 Ajout et Suppression

On arrive aux deux fonctions les plus complexes. Elles sont intégralement commentées dans notre fichier "commun.c" et nous allons brièvement les expliquer ici !

La fonction **"AjoutStruct"** va être appelée par la fonction **"LectureFichier"** qui lui fournit toutes les informations sur une ligne de fichier ! Elle va ensuite créer un élément de la forme de la figure 7. Si on rencontre un problème lors de la création de cet élément, on le libère et on ne l'ajoute pas à la SDD ! Nous avons ensuite différents cas d'ajout à gérer :

- L'ajout quand la SDD est vide : on fait simplement pointer notre tête fictive sur le nouvel élément.
- L'ajout quand la case semaine existe déjà ! A ce moment là, on doit vérifier si l'action que l'on souhaite ajouter ne doit pas être ajoutée en tête de la liste des actions de cette semaine. Si c'est le cas on modifie le pointeur "liste_action" de la dite semaine ! Sinon on fait un simple ajout avec les valeurs du pointeur précédent action ! On oublie pas de libérer la case semaine fraîchement car une semblable existe déjà dans la SDD !
- L'ajout quand la semaine n'existe pas ! A ce moment là on doit vérifier que la semaine à ajouter n'est pas à positionner en tête de SDD, si c'est le cas on change le pointeur de notre tête fictive, sinon on ajoute simplement notre semaine avec le pointeur précédent semaine !

La fonction **"SupprStruct"** prend elle en argument un pointeur sur notre tête fictive et sur un élément de la forme de la figure 7 (c'est à dire une semaine pointant sur une action). On va parcourir toute notre SDD en cherchant une semaine puis une action identique à celle prise en argument ! Une fois cela fait, si on cette action existe, on la supprime, sinon on fait rien ! Si l'action est bien supprimée on renvoie 1, si la semaine visée n'existe pas dans la SDD on renvoie 0 enfin si l'action n'existe pas on renvoie -1. Nous avons encore une fois plusieurs cas à traiter :

- La semaine n'existe pas : on ne fait rien
- L'action n'existe pas : on ne fait rien
- On supprime la première action d'une semaine. A ce moment là on doit changer l'action pointée par la semaine puis libérer la mémoire de cette action.
- On supprime une action pas en tête. A ce moment là on a juste à changer les pointeurs en faisant pointer le précédents de cette dite action sur la suivant de celle-ci puis de libérer la mémoire de cette action !
- Après suppression d'une action, on se rend compte que la semaine est maintenant vide ! On doit donc la supprimer elle aussi ! Soit c'est la première et on change le pointeur de la tête fictive soit on joue rediriger le pointeur précédent. Dans tous les cas on oublie pas de libérer la semaine à supprimer !

3.4.4 Affichage

La fonction **AffichageAgenda** est toute simple. Elle parcourt entièrement notre SDD et affiche pour chaque élément "semaine" toutes les actions qu'il contient ! On utilise un tableau

statique contenant tous les jours de la semaine pour faire un affichage plus propre. Le principe est simple : On récupère la valeur ascii du jour à afficher auquel on enlève la valeur ascii du char "1". Ainsi, pour Lundi on aura l'indice 0, mardi l'indice 1 ... On a plus qu'à afficher la string présente à l'indice du tableau voulu !

3.5 contigue.h et contigue.c

Enfin voyons les fonctions en lien avec la liste contiguë qui cherche un motif. Nous rappelons que l'on doit, à l'aide d'un motif pris en argument, chercher si celui-ci apparait dans le nom de nos actions et si c'est le cas, stocker la date de cette actions dans une liste contiguë.

La liste contiguë est toutes simples : c'est un tableau de char en 2 dimensions statique ayant un nombre de ligne limité défini par la valeur **MAX_TAILLE_CONTIGUE** présent dans contigue.h. Chaque string contenu dans cette liste est de taille **TAILLE_DATE_ACTION**. Cette taille est de 10 car l'on stocke les 6 bytes de l'année ainsi que du numéro de semaine puis les 3 bytes du jour et de l'heure et enfin le char '\0'. Nous avons mis la liste contiguë en variable globale dans le fichier contigue.h. Ainsi on commence à définir deux pointeurs de type "char", un de début et un de fin et on appelle la fonction **InitContigue** qui va les placer correctement.

La fonction **InitContigue** prend donc en argument notre liste contiguë et deux pointeurs indirects "debut_contigue" et "fin_contigue". Au départ la liste étant vide, on les place tous les deux au début de la liste c'est à dire qu'ils prennent l'adresse "contigue[0]". Après cela on utilisera plus la notation "[]" des tableaux en C mais bien juste nos 2 pointeurs de début et de fin.

Ensuite, la fonction **RemplisContigue** prend en argument un motif, notre tête fictive et nos pointeurs de début et de fin. On va créer un pointeur "parcours_contigue" qui aura pour but de parcourir la liste contiguë et un pointeur "dernier_element" qui pointe sur la dernière position possible où écrire dans notre liste contiguë. Si on essaye d'écrire plus loin on n'écrit plus dans la zone fixe allouée à notre liste. C'est donc pour cela que l'on vérifie dans le if si on a dépassé cette valeur ou pas ! On a donc maintenant plus qu'à parcourir entièrement toutes notre SDD et vérifier si le motif apparait dans le nom des actions (et que la liste contiguë n'est pas pleine), si c'est le cas alors on ajoute les informations de date dans la liste. La fonction va ensuite en toute fin placer le pointeur de fin de la liste contiguë qui correspond à l'adresse où se trouve la dernière chaîne de caractère de taille 10.

Au final **AfficheContigue** parcourt tous les éléments existants dans la liste contiguë jusqu'à arriver au pointeur fin. On a utilisé une boucle for pour respecter les bonnes normes de code. Étant donné qu'on connaît la taille de notre liste on passe par une boucle for ! On utilise le même tableau "tab_jour" que dans "AfficheStruct" pour afficher plus proprement les jours de la semaine.

4 Liste des cas

Nous allons ici vous présenter la liste des cas possibles et leurs jeux de donnée attitrés afin que vous puissiez tous les tester.

Pour voir un exemple moins précis avec la création d'un agenda un peu dense, nous vous avons fourni le fichier **classique** ! Mais maintenant regardons de manière plus précise tout les cas :

4.1 Lecture

Le premier cas de la section lecture est **lec_vide** : On donne simplement en entrée un fichier vide et on regarde comment le programme réagit. Ici on aura un affichage de l'agenda vide et un fichier en sortie vide ! On a aucun soucis de mémoire.

```

==-----== VOTRE AGENDA ==-----==
==-----==
==8340==
==8340== HEAP SUMMARY:
==8340==   in use at exit: 0 bytes in 0 blocks
==8340== total heap usage: 5 allocs, 5 frees, 6,072 bytes allocated
==8340==

```

FIGURE 8 – Résultat pour "lec_vide"

Ensuite on essaye d'entrer un fichier ayant une seule ligne avec une erreur sur l'action ! C'est **lec_semaine_fausse**. Comme prévu le programme ne va pas ajouter l'élément car il est faux et aura donc un agenda vide et aucune fuite mémoire à la fin ! Le fichier de sortie attitré et devenu un fichier vide !

```

==-----== VOTRE AGENDA ==-----==
==-----==
==8690==
==8690== HEAP SUMMARY:
==8690==   in use at exit: 0 bytes in 0 blocks
==8690== total heap usage: 6 allocs, 6 frees, 6,096 bytes allocated
==8690==

```

FIGURE 9 – Résultat pour "lec_semaine_fausse"

Enfin on essaye de donner un fichier avec une seule ligne qui contient une erreur sur l'action ! C'est **lec_action_fausse** ! Comme pour avant l'agenda et le fichier de sortie seront vide et on aura aucune fuite de mémoire !

```

==-----== VOTRE AGENDA ==-----==
==-----==
==8891==
==8891== HEAP SUMMARY:
==8891==   in use at exit: 0 bytes in 0 blocks
==8891== total heap usage: 6 allocs, 6 frees, 6,096 bytes allocated
==8891==

```

FIGURE 10 – Résultat pour "lec_action_fausse"

Note importante : A partir de maintenant nous n'entrons plus d'erreur dans les fichiers d'entrées ! Ainsi, nos fichiers de sortie auront exactement le même nombre de ligne que nos fichiers d'entrée. Cependant, étant triés dans l'agenda par ordre croissant, nos éléments seront triés dans le fichier de sortie !

4.2 Ajout

Pour la partie ajout, les éléments dont on observe l'ajout sont toujours écrit sur la **dernière ligne** de leur fichier de test !

On commence donc avec l'ajout du premier élément ! C'est **ajout_premier** ! On obtient un agenda avec tout naturellement un seul élément !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_premier

===== VOTRE AGENDA =====

==== Année 2021 Semaine 16 ====
Le Lundi à 03h : coursd'EPS

=====
```

FIGURE 11 – Résultat pour "ajout_premier"

Ensuite on va essayer d'ajouter un élément "semaine" en tête de liste alors que l'agenda contient déjà des éléments ! C'est **ajout_tete_semaine**. On voit bien que l'élément "TETE" a été ajouté en tête de liste !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_tete_semaine

===== VOTRE AGENDA =====

==== Année 2021 Semaine 15 ====
Le Dimanche à 13h : TETE

==== Année 2021 Semaine 16 ====
Le Lundi à 03h : coursd'EPS
Le Mardi à 13h : Automate

=====
```

FIGURE 12 – Résultat pour "ajout_tete_semaine"

On fait de même mais en ajoutant ensuite une semaine en fin de liste ! C'est **ajout_fin_semaine**. L'élément "FIN" apparait bien en fin de liste !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_fin_semaine

===== VOTRE AGENDA =====

==== Année 2021 Semaine 16 ====
Le Lundi à 03h : coursd'EPS
Le Mardi à 13h : Automate

==== Année 2022 Semaine 16 ====
Le Dimanche à 13h : FIN

=====
```

FIGURE 13 – Résultat pour "ajout_fin_semaine"

Puis pour finir au niveau de l'insertion des semaines, on fait une insertion classique au milieu de la SDD! C'est **ajout_classique_semaine**! On voit bien que "Classique" c'est bien ajouté au milieu de la SDD, à sa place!

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_classique_semaine

===== VOTRE AGENDA =====

===== Année 2020 Semaine 16 =====
Le Lundi à 03h : cours d'EPS

===== Année 2021 Semaine 16 =====
Le Mardi à 13h : Automate

===== Année 2022 Semaine 18 =====
Le Mardi à 09h : Classique

===== Année 2023 Semaine 16 =====
Le Dimanche à 13h : EPS      1

=====
```

FIGURE 14 – Résultat pour "ajout_classique_semaine"

Passons au ajout d'action avec **ajout_tete_action** qui va ajouter une action en tête de semaine! Comme prévu, l'élément "DEBUT" est bien en tête de l'unique semaine de la SDD!

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_tete_action

===== VOTRE AGENDA =====

===== Année 2021 Semaine 16 =====
Le Lundi à 01h : DEBUT
Le Lundi à 03h : cours d'EPS
Le Mardi à 13h : Automate
Le Dimanche à 13h : EPS      1

=====
```

FIGURE 15 – Résultat pour "ajout_tete_action"

On va maintenant ajouter une action en fin de semaine en utilisant **ajout_fin_action**! Le résultat est bon, on voit bien l'élément "FIN" en fin de l'unique semaine de la SDD!

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_fin_action

===== VOTRE AGENDA =====

===== Année 2021 Semaine 16 =====
Le Lundi à 03h : cours d'EPS
Le Mardi à 13h : Automate
Le Dimanche à 13h : FIN

=====
```

FIGURE 16 – Résultat pour "ajout_fin_action"

Pour fini avec les ajouts, on fait un ajout d'action classique en milieu de semaine avec **ajout_classique_action** !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme ajout_classique_action

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 16 ====
Le Lundi à 03h : cours d'EPS
Le Mardi à 21h : Classique
Le Mercredi à 13h : Automate
Le Dimanche à 13h : EPS      1

==--==
```

FIGURE 17 – Résultat pour "ajout_classique_action"

4.3 Suppression

Note : Pour la partie suppression, nous avons modifié le fichier "main.c" afin de modifier la valeur de l'élément à supprimer ! Si vous souhaitez tester certains comportements, nous vous avons laissé un exemple de suppression !

Ici les fichiers de sortie ne contiendront pas l'élément supprimé par le programme !

Commençons avec le cas où l'on supprime la seule semaine existante dans la SDD et qu'elle devient vide ! C'est **suppr_semaine_unique** ! Le résultat après suppression est l'agenda vide !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_semaine_unique

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 15 ====
Le Lundi à 06h : Supprimer

==--==

-> Suppression <-

==--== VOTRE AGENDA ==--==

==--==
```

FIGURE 18 – Résultat pour "suppr_semaine_unique"

On va maintenant supprimer la semaine en tête avec **suppr_semaine_tete** ! Ici on voit bien que la semaine en tête a bien disparu !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_semaine_tete

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 15 ====
Le Lundi à 06h : Supprimer

==== Année 2022 Semaine 15 ====
Le Lundi à 08h : cm de java

==--==

-> Suppression <-

==--== VOTRE AGENDA ==--==

==== Année 2022 Semaine 15 ====
Le Lundi à 08h : cm de java

==--==
```

FIGURE 19 – Résultat pour "suppr_semaine_tete"

Ici on va supprimer la dernière semaine de la SDD avec **suppr_semaine_fin** :

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_semaine_fin

==--== VOTRE AGENDA ==--==

==== Année 2023 Semaine 15 ====
Le Vendredi à 10h : Docteur

==== Année 2024 Semaine 15 ====
Le Lundi à 11h : Supprimer

==--==

-> Suppression <-

==--== VOTRE AGENDA ==--==

==== Année 2023 Semaine 15 ====
Le Vendredi à 10h : Docteur

==--==
```

FIGURE 20 – Résultat pour "suppr_semaine_fin"

Enfin pour en finir avec les semaines on va supprimer une semaine au milieu de la SDD avec **suppr_semaine_classique** :

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_semaine_classique

==--== VOTRE AGENDA ==--==

==== Année 2022 Semaine 15 ====
Le Lundi à 08h : cm de java

==== Année 2023 Semaine 15 ====
Le Vendredi à 10h : Supprimer

==== Année 2024 Semaine 15 ====
Le Lundi à 11h : Docteur

==--==

-> Suppression <-

==--== VOTRE AGENDA ==--==

==== Année 2022 Semaine 15 ====
Le Lundi à 08h : cm de java

==== Année 2024 Semaine 15 ====
Le Lundi à 11h : Docteur

==--==
```

FIGURE 21 – Résultat pour "suppr_semaine_classique"

Passons maintenant aux actions : ici avec **suppr_action_tete**, on va supprimer une action en tête de semaine !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_action_tete

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 16 ====
Le Lundi à 03h : Supprimer
Le Lundi à 13h : Automate

==--==

-> Suppression <-

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 16 ====
Le Lundi à 13h : Automate

==--==
```

FIGURE 22 – Résultat pour "suppr_action_tete"

Ici on supprime une action en fin de semaine avec **suppr_action_fin**.

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_action_fin  
  
===== VOTRE AGENDA =====  
  
===== Année 2021 Semaine 16 =====  
Le Mercredi à 13h : Basketball  
Le Dimanche à 09h : Supprimer  
  
=====  
  
> Suppression <-  
  
===== VOTRE AGENDA =====  
  
===== Année 2021 Semaine 16 =====  
Le Mercredi à 13h : Basketball  
  
=====
```

FIGURE 23 – Résultat pour "suppr_action_fin"

Ensuite, ici, avec **suppr_action_classique**, on supprime une action en milieu de semaine !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_action_classique  
  
===== VOTRE AGENDA =====  
  
===== Année 2021 Semaine 16 =====  
Le Lundi à 13h : Automate  
Le Mercredi à 13h : Supprimer  
Le Dimanche à 09h : Basketball  
  
=====  
  
> Suppression <-  
  
===== VOTRE AGENDA =====  
  
===== Année 2021 Semaine 16 =====  
Le Lundi à 13h : Automate  
Le Dimanche à 09h : Basketball  
  
=====
```

FIGURE 24 – Résultat pour "suppr_action_classique"

Enfin avec le fichier **suppr_action_classique**, on a essayé de supprimer une action puis une semaine qui existait pas ! On remarque que rien n'est supprimé !

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme suppr_action_classique

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 16 ====
Le Lundi à 13h : Automate
Le Mercredi à 13h : Supprimer
Le Dimanche à 09h : Basketball

==--==

-> Suppression <-

==--== VOTRE AGENDA ==--==

==== Année 2021 Semaine 16 ====
Le Lundi à 13h : Automate
Le Mercredi à 13h : Supprimer
Le Dimanche à 09h : Basketball

==--==
```

FIGURE 25 – Résultat quand l'action qu'on essaye de supprimer n'existe pas

4.4 Contiguë

Enfin on fini avec la partie sur la liste contiguë. Pour commencer testons de chercher le motif "ABC" qui n'apparait pas dans le fichier **con_motif_absent**. On voit que la liste contiguë reste vide :

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme con_motif_absent

---> Motif recherché: ABC <---

==== Date Correspondant au motif cherché ====
=====
```

FIGURE 26 – Résultat avec "con_motif_absent" et le motif "ABC"

Maintenant regardons si on a plus de motif que de place dans la liste! Dans le fichier **con_trop_motif**, on a 18 fois le motif "1" qui apparait et notre liste à une taille fixée de 16! On voit que seul les 16 premiers motifs sont affichés!


```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme con_trop_motif
---> Motif recherché: 1 <---

===== Date Correspondant au motif cherché =====
Année: 2021 Semaine 16: Le Lundi à 03h
Année: 2021 Semaine 16: Le Mardi à 13h
Année: 2021 Semaine 16: Le Dimanche à 13h
Année: 2021 Semaine 18: Le Mardi à 09h
Année: 2022 Semaine 15: Le Lundi à 06h
Année: 2022 Semaine 15: Le Lundi à 08h
Année: 2022 Semaine 15: Le Lundi à 11h
Année: 2022 Semaine 15: Le Lundi à 12h
Année: 2022 Semaine 15: Le Vendredi à 10h
Année: 2022 Semaine 15: Le Samedi à 14h
Année: 2022 Semaine 17: Le Lundi à 06h
Année: 2023 Semaine 15: Le Dimanche à 14h
Année: 2024 Semaine 15: Le Vendredi à 10h
Année: 2025 Semaine 15: Le Lundi à 11h
Année: 2026 Semaine 15: Le Lundi à 12h
Année: 2027 Semaine 15: Le Samedi à 14h
=====
```

FIGURE 27 – Résultat avec "con_trop_motif" et le motif "1"

Enfin dernier cas, regardons une recherche de motif classique (assez de place dans la liste). On utilise ici **con_motif_classique** et on lui demande de recherche le motif "TD" :

```
~/Bureau/SDD/tp1-sdd-isima/Agenda$ ./Programme con_motif_classique
---> Motif recherché: TD <---

===== Date Correspondant au motif cherché =====
Année: 2022 Semaine 15: Le Lundi à 06h
Année: 2022 Semaine 15: Le Lundi à 08h
Année: 2022 Semaine 17: Le Lundi à 06h
=====
```

FIGURE 28 – Résultat avec "con_motif_classique" et le motif "TD"

5 Annexe

Table des figures

1	Schéma de notre SDD "Agenda"	2
2	Notre fichier d'entrée	3
3	Notre fichier de sortie	3
4	Structure "tete"	4
5	Structure "semaine"	4
6	Structure "action"	4
7	Représentation de l'élément crée	5
8	Résultat pour "lec_vide"	8
9	Résultat pour "lec_semaine_fausse"	8
10	Résultat pour "lec_action_fausse"	8
11	Résultat pour "ajout_premier"	9
12	Résultat pour "ajout_tete_semaine"	9
13	Résultat pour "ajout_fin_semaine"	9
14	Résultat pour "ajout_classique_semaine"	10
15	Résultat pour "ajout_tete_action"	10
16	Résultat pour "ajout_fin_action"	10
17	Résultat pour "ajout_classique_action"	11
18	Résultat pour "suppr_semaine_unique"	11
19	Résultat pour "suppr_semaine_tete"	12
20	Résultat pour "suppr_semaine_fin"	12
21	Résultat pour "suppr_semaine_classique"	13
22	Résultat pour "suppr_action_tete"	13
23	Résultat pour "suppr_action_fin"	14
24	Résultat pour "suppr_action_classique"	14
25	Résultat quand l'action qu'on essaye de supprimer n'existe pas	15
26	Résultat avec "con_motif_absent" et le motif "ABC"	15
27	Résultat avec "con_trop_motif" et le motif "1"	16