

TP1 Algorithmique et Structure de Données

**Compte-rendu : Gestion de l’échéancier**

BONHOMME PAUL & LAMRANI RIME

Table des matières

[1. Présentation générale 3](#_Toc97589887)

[1.1. Description de l’objet du TP 3](#_Toc97589888)

[1.2. Description et schéma de la structure de données et des fichiers utilisés 3](#_Toc97589889)

[1.3 Organisation du code source 6](#_Toc97589890)

[2. Détails des différentes fonctionnalités 6](#_Toc97589891)

[3. Compte rendu d’exécution 7](#_Toc97589892)

[3.1 MakeFile 7](#_Toc97589893)

[3.2 Jeux de test complets 8](#_Toc97589894)

[4. Annexes 8](#_Toc97589895)

# Présentation générale

## Description de l’objet du TP

Tout d’abord, pour avoir plus d’information sur le projet, veuillez ouvrir et lire le **README** présent dans le projet.

Ce fichier vous donnera toutes les informations **importantes** pour **exécuter** le projet et reprend les éléments de description du projet.

Des captures d’écrans du README sont en annexe **à la fin** du compte rendu.

Voici le lien sur le GitHub du projet : <https://github.com/pabonhomme/projet_sdd_isima/tree/main>

Le but de ce projet est de proposer différentes actions sur des listes doublement chainées. En effet, nous avons une liste de semaines avec, pour chaque semaine, une liste d’actions pour un jour et une heure donnée. Nous devions remplir ces listes à partir d’un fichier texte.

De plus, nous devons pouvoir supprimer une action précise, rechercher une liste d’action en fonction d’un motif donné et enfin, pouvoir sauvegarder les données dans un fichier de sauvegarde.

## Description et schéma de la structure de données et des fichiers utilisés

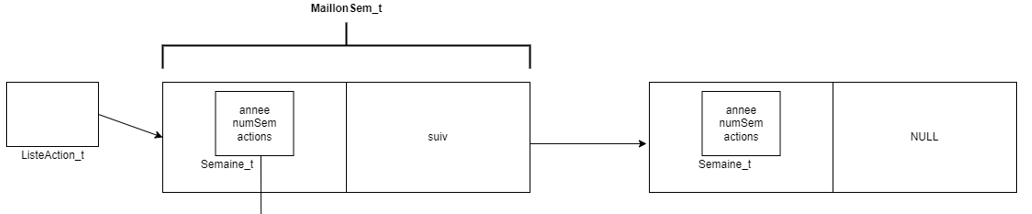
Une image contenant texte

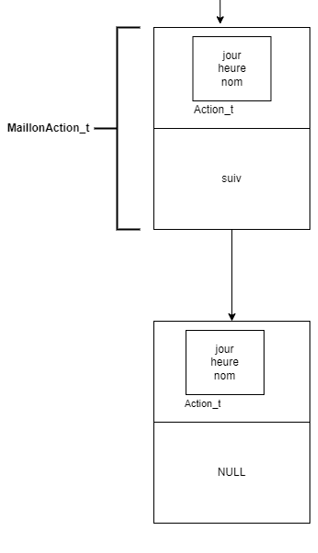
Description générée automatiquementUne image contenant texte

Description générée automatiquement

**Figure 1 : structure de données semaine et action**

Lors de la lecture, nous avons décidé de séparer ces informations en deux structures. Une Semaine\_t qui contient l'année et le numéro de semaine, l'autre nommé Action\_t contient le jour de la semaine, l'heure de la journée et le nom de l'action.





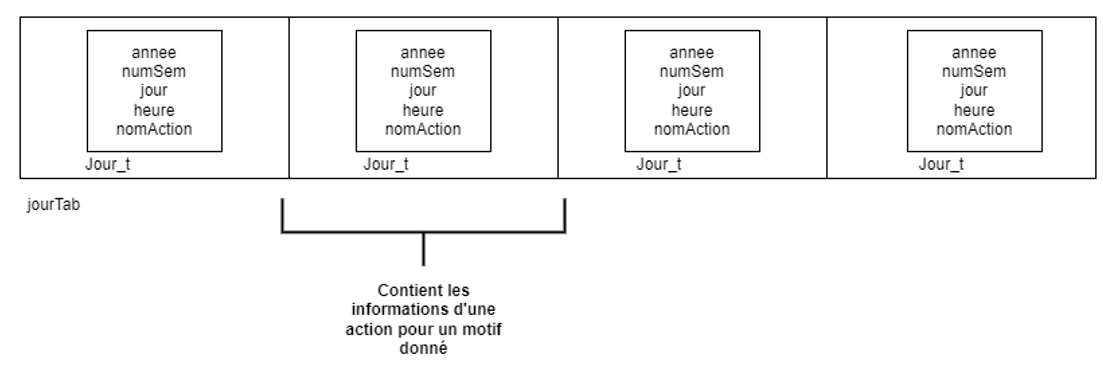
**Figure 2 : Schéma des structures de données**

La première a un pointeur de type ListeSemaine\_t sur des maillons du type MaillonSemaine\_t.

Le maillon semaine contient une structure Semaine\_t contenant un pointeur nommé actions sur un MaillonAction\_t et un pointeur sur le maillon semaine suivant.

Le maillon action contient une structure de type Action\_t avec les informations sur l'action et un pointeur sur le maillon action suivant.

**Pour voir les structures de données ainsi que leurs descriptions en plus clair et en plus gros : Ouvrir l'image "struct\_diagramme.png" présente dans le dossier.**



**Figure 3 : Schéma de la partie motif**

Une image contenant texte

Description générée automatiquement

**Figure 4 : structure de données semaine et action**

Pour la partie recherche de motif, nous avons créé une structure Jour\_t qui va regrouper toutes les informations d'une action pour un motif donné en paramètre. Nous créons un tableau (liste contiguë précisée dans la question 2) de structures Jour\_t limité à 8 éléments (variable MAX\_JOUR définie dans jour.h).

**Format du fichier :**

**Une image contenant texte, équipement électronique, clavier

Description générée automatiquement**

**Figure 5 : fichier semaines.txt**

Une ligne du fichier **"semaines.txt"** se présente sous cette forme : 202201108TPs de SDD

* 2022 -> année de la semaine
* 01 -> numéro de la semaine (01 à 52)
* 1 -> jour de la semaine (1 à 7)
* 08 -> heure de la journée (01 à 24)
* TPs de SDD -> nom de l'action

Nous sauvegardons les listes dans le fichier **"sauvegarde.txt"** avec la même forme de ligne que le fichier de lecture.

## 1.3 Organisation du code source

Pour chaque question, nous avons décidé de créer un fichier d’entête (fichier.h) et un fichier source (fichier.c) par type de données. Exemple : semaine.h et semaine.c

main.c contient la fonction main, le menu et l’affichage du menu.

test.c contient la fonction de test des différentes fonctions du projet.

# Détails des différentes fonctionnalités

**Les différentes fonctions sont toutes commentées dans les fichiers sources avec une description de leurs algorithmes.**

Nous ne les détaillerons pas dans ce compte rendu. Nous allons donc parler de comment nous vérifions que les données rentrées par l’utilisateur sont valides.

Une image contenant texte

Description générée automatiquement

**Figure 6 : Fonction main**

Dans la figure 6, nous vérifions que l’utilisateur donne le nom du fichier et pas un argument de plus.

Une image contenant texte

Description générée automatiquement

**Figure 7 : Validation des données**

Le code ci-dessus vérifie que l’utilisateur rentre un entier dans le choix de l’option du menu grâce au scanf dans la condition du while. Si c’est un caractère, une entrée, un espace, l’utilisateur devra ressaisir des données jusqu’à ce que ce soit un entier compris entre 1 et 6.

Dans le cas où l’utilisateur doit par exemple rentrer un motif en ligne de commande, ou l’année, le numéro de semaine ou l’heure d’une action à supprimer. Si l’utilisateur rentrait un nombre de caractère supérieur à la taille de la chaîne de caractère prévue et que l’on récupère la chaîne de caractère avec un scanf cela engendrait un **dépassement de tampon.** Cela provoque ce que l’on appelle un « Buffer Overflow ».

Nous avons donc utilisé la fonction fgets pour récupérer les données. Nous avons créé deux fonctions qui permettent de récupérer un nombre précis de caractères et de vider le buffer.

Une image contenant texte

Description générée automatiquement

**Figure 8 : Fonction de lecture de l’entrée standard**

La fonction ci-dessus lit les données rentrées en ligne de commande pour une longueur donnée en paramètre. Elle remplace ensuite le « \n » par « \0 ».

Une image contenant texte

Description générée automatiquement

**Figure 9 : Fonction qui vide le buffer**

On vide la suite du buffer après lecture pour ne pas induire en erreur les instructions suivantes et ne pas répéter les actions de vérifications plusieurs fois.

# Compte rendu d’exécution

## 3.1 MakeFile

Nous avons réalisé un MakeFile qui compile automatiquement et avec plusieurs options, notre code, en tapant make. Nous avons ajouté l’option OPTIONS avec -Wall, -Wextra et -g pour détecter le maximum de warning et avoir le code le plus propre possible.

Une image contenant texte

Description générée automatiquement

**Figure 8 : Code du MakeFile**

Nous allons maintenant voir des captures d’écran de la série de test que vous pouvez vous-même lire dans le fichier « test.c » ou en exécutant la commande 5 dans le menu du projet.

## 3.2 Jeux de test complets

Nous avons créé un fichier « test.c » qui contient une fonction qui va tester les différentes actions du projet. Tout est guidé, l’utilisateur n’a juste qu’à appuyer sur entrée pour faire avancer les tests.

Les captures d’écrans qui vont suivre sont l’exécution des tests que vous pouvez lancer vous-même en appuyant sur 5 dans le menu du projet.

Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement

Action et semaine qui vont être supprimée par la suite

**Figure 9 : Test de l’affichage**

Le test affiche les semaines et les actions présentes dans le fichier « semaines.txt ».

Une image contenant texte

Description générée automatiquement

**Figure 10 : Test de l’affichage**

Ci-dessus, nous pouvons voir que l’action TPs de syst n’est plus présente ainsi que la semaine 06 de 2019 qui a été supprimée en conséquence.

Une image contenant texte

Description générée automatiquement

**Figure 11 : Test de la recherche de motif « C »**

Sur la figure 11, nous avons recherché le motif « C », on remarque nous avons bien le TP de c et de C++ d’affiché. Le recherche de motif fonctionne correctement.

Une image contenant texte

Description générée automatiquement

**Figure 12 : Test de la recherche de motif « TP »**

Sur la figure 12, nous avons lancé la recherche pour le motif « TP », nous avons 7 jours affichés sur 9 présents initialement dans le fichier « semaines.txt » car nous venons de supprimer deux actions.

Une image contenant texte, intérieur, capture d’écran

Description générée automatiquement

**Figure 13 : Test de la sauvegarde**

Nous sauvegardons les données actuelles dans le fichier « sauvegarde.txt ».

Une image contenant texte

Description générée automatiquement

**Figure 14 : Fichier « sauvegarde.txt »**

On remarque que comparé à la **figure 5**, nous avons que 7 lignes. Nous avons bien supprimé deux actions précédemment donc la sauvegarde s’est bien passée.

Une image contenant texte

Description générée automatiquement

**Figure 15 : Tests des cas d’erreurs pour le motif**

Sur la figure 15, nous testons pour le motif « test\_err », aucune action ne contient ce motif donc la fonction renvoie « Aucune action contenant le motif « test\_err » ».

Une image contenant texte

Description générée automatiquement

**Figure 16 : Tests des cas d’erreurs pour la suppression d’action**

Nous essayons sur la figure 16 de supprimer une action qui n’existe pas. La fonction va détecter qu’elle n’existe pas et renvoyer : « L’action à supprimer n’existe pas ».

# Annexes

Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement