TP1 Algorithmique et Structure de Données

Compte-rendu : Gestion de l'échéancier

BONHOMME PAUL & LAMRANI RIME

Table des matières

1. Présentation générale

1.1. 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.

1.2. Description et schéma de la structure de données et des fichiers utilisés

```c
typedef enum
{
    faux,vrai
}Boolen_t;

typedef struct
{
    int  jour;
    char heure[3];
    char nom[11];
}Action_t;

typedef struct maillonAction_t
{
    Action_t                action;
    struct maillonAction_t * suiv;
}MaillonAction_t, *ListeAction_t;
```

```c
typedef struct
{
    char              annee[5];
    char              numSem[3];
    MaillonAction_t * actions;
}Semaine_t;

typedef struct maillonSem_t
{
    Semaine_t           semaine;
    struct maillonSem_t * suiv;
}MaillonSem_t, *ListeSem_t;
```

Figure 1 : structure de données semaine et action

Nous avons créé une structure de type Boolen_t qui peut valoir faux ou vrai.

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.

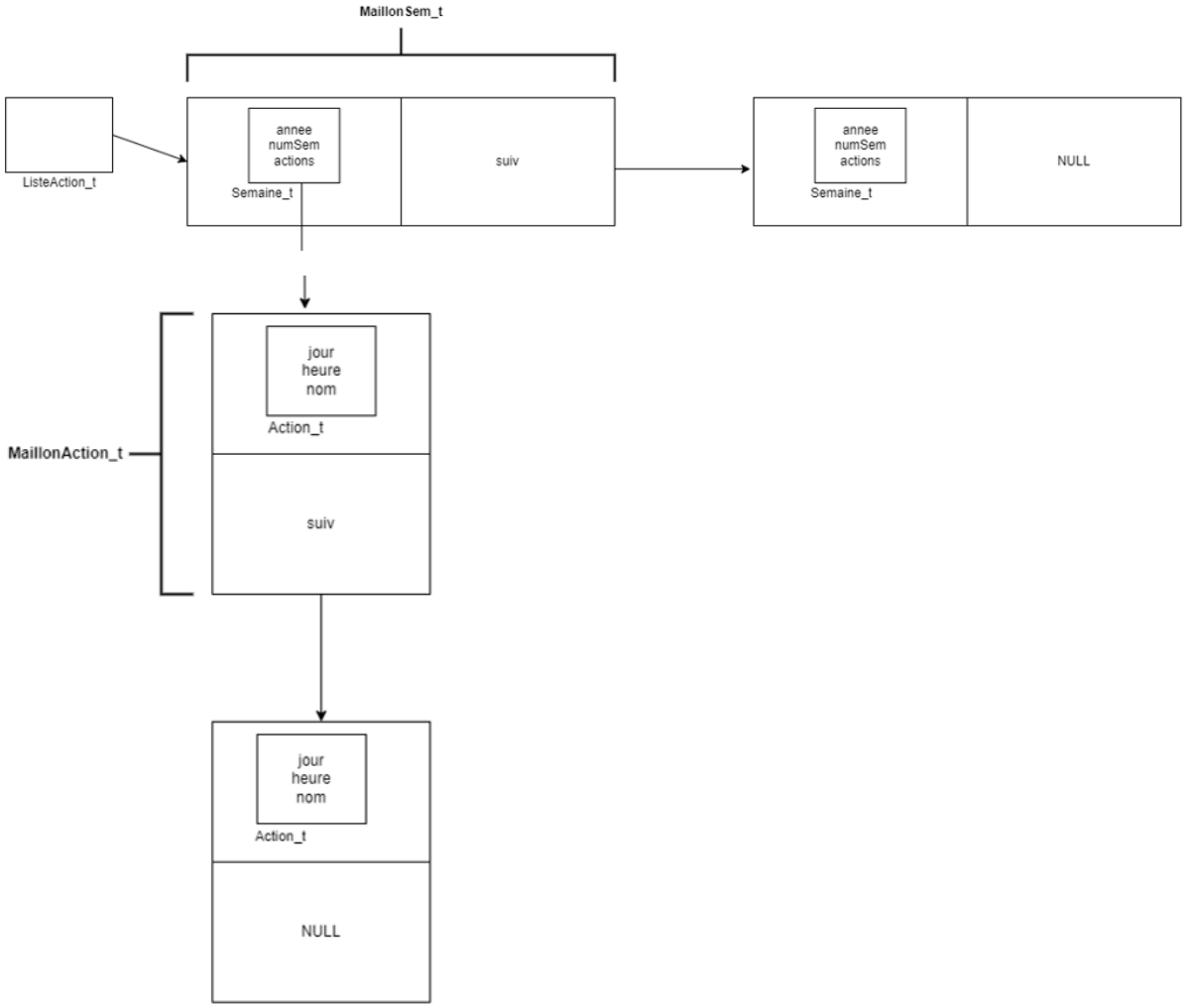


Figure 2 : Schéma des structures de données

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.

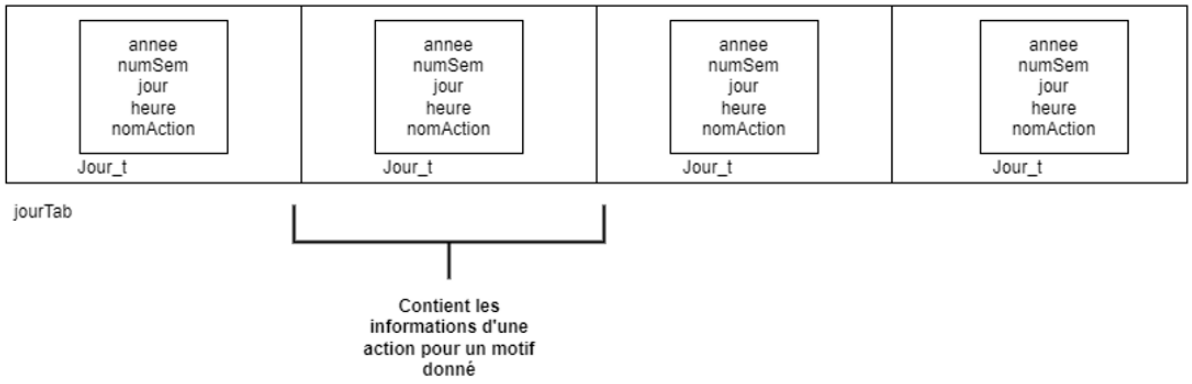


Figure 3 : Schéma de la partie motif

```c
#define MAX_JOUR 8 // taille maximale du tableau de jours pour un motif donné

typedef struct
{
    char annee[5];
    char numSem[3];
    int  jour;
    char heure[3];
    char nomAction[11];
}Jour_t;
```

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 :

```
1   202201108TPs de SDD
2   202202108TPs d'algo
3   202003108TPs d'java
4   202104108TPs d'prob
5   202305108TPs d'math
6   201906108TPs d'syst
7   202007108TPs de C
8   202208108TPs de C++
9   202305302TPs d'phys
10
```

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.

2. 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.

```c
int main(int argc, char *argv[])
{
    if(argc==2) // si l'utilisateur a bien fourni un argument lors de l'éxécution
    {
        menu(argv[1]);
    }
    else
    {
        printf("Vous n'avez pas fourni le nom du fichier à lire\n");
    }
    return 0;
}
```

Figure 6 : Fonction main

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

```c
printf("Tapez votre choix :\n");
while(scanf("%d",&choix)==0)
    {
        affichMenu();
        printf("Caractere non pris en charge\n");

        printf("Tapez votre choix :\n");
        scanf("%d",&choix);
        getchar();
    }
```

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.

```c
void lireChaine(char *chaine, int longueur)
{
    char *positionEntree = NULL;

    if (fgets(chaine, longueur, stdin) != NULL) // si on lit des données
    {
        positionEntree = strchr(chaine, '\n'); // On recherche l'"Entrée"
        if (positionEntree != NULL) // Si on a trouvé le retour à la ligne
        {
            *positionEntree = '\0'; // On remplace ce caractère par \0
        }
        else
        {
            viderBuffer(); // on vide le buffer
        }
    }
    else
    {
        viderBuffer(); // on vide le buffer
    }
}
```

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 ».

```c
void viderBuffer()
{
    int c = 0;
    while (c != '\n' && c != EOF) // tant que l'on a pas atteint la fin du buffer
    {
        c = getchar();
    }
}
```

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.

Gestion de la mémoire :

Nous avons géré l'allocation et la libération de mémoire. Nous avons créé une fonction de libération des maillons que nous avons alloués au départ.

```
Tapez votre choix :
6
==199==
==199== HEAP SUMMARY:
==199==     in use at exit: 64 bytes in 2 blocks
==199==   total heap usage: 23 allocs, 21 frees, 6,136 bytes allocated
==199==
==199== 64 bytes in 2 blocks are definitely lost in loss record 1 of 1
==199==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==199==    by 0x10A5B1: insererEnTeteAction (in /mnt/d/ISIMA/S2/SDD/projet_sdd_isima/projet)
==199==    by 0x10A658: insererAction (in /mnt/d/ISIMA/S2/SDD/projet_sdd_isima/projet)
==199==    by 0x109CBE: lireSemaine (in /mnt/d/ISIMA/S2/SDD/projet_sdd_isima/projet)
==199==    by 0x109DA0: chargeSemaine (in /mnt/d/ISIMA/S2/SDD/projet_sdd_isima/projet)
==199==    by 0x109482: menu (in /mnt/d/ISIMA/S2/SDD/projet_sdd_isima/projet)
==199==    by 0x109885: main (in /mnt/d/ISIMA/S2/SDD/projet_sdd_isima/projet)
==199==
==199== LEAK SUMMARY:
==199==    definitely lost: 64 bytes in 2 blocks
==199==    indirectly lost: 0 bytes in 0 blocks
==199==      possibly lost: 0 bytes in 0 blocks
==199==    still reachable: 0 bytes in 0 blocks
==199==         suppressed: 0 bytes in 0 blocks
==199==
==199== For lists of detected and suppressed errors, rerun with: -s
==199== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Figure 9 : Code du MakeFile

Cependant, comme nous pouvons le voir sur la figure 9, 2 blocks de mémoires sont définitivement perdus. Après différents tests, nous nous sommes rendu compte que les 9 actions du fichier étaient bien libérées et les 8 semaines aussi.

Nous avons 4 free qui se font automatiquement. Cela donne un total de 21 free sur 23 allocs d'après Valgrind. Nous ne comprenons pas d'où viennent ces fuites de mémoires et n'avons pas réussi à les trouver afin de les libérer.

3. 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.

```
1  CC=gcc
2  CFLAGS=-Wextra -Wall
3  LDFLAGS=
4  EXEC=projet
5
6  all: $(EXEC)
7
8  projet: main.o semaine.o jour.o action.o test.o
9      $(CC) -o $@ $^ $(LDFLAGS)
10     @echo "\nLancer le programme avec ./projet semaines.txt"
11
12 action.o: action.c action.h
13     $(CC) -o $@ -c $< $(CFLAGS)
14
15 semaine.o: semaine.c semaine.h
16     $(CC) -o $@ -c $< $(CFLAGS)
17
18 jour.o: jour.c jour.h
19     $(CC) -o $@ -c $< $(CFLAGS)
20
21 test.o: test.c jour.h
22     $(CC) -o $@ -c $< $(CFLAGS)
23
24 main.o: main.c jour.h
25     $(CC) -o $@ -c $< $(CFLAGS)
26
27 clean:
28     rm -rf *.o
29
30 rmExec: clean
31     rm -rf $(EXEC)
```

Figure 10 : 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.

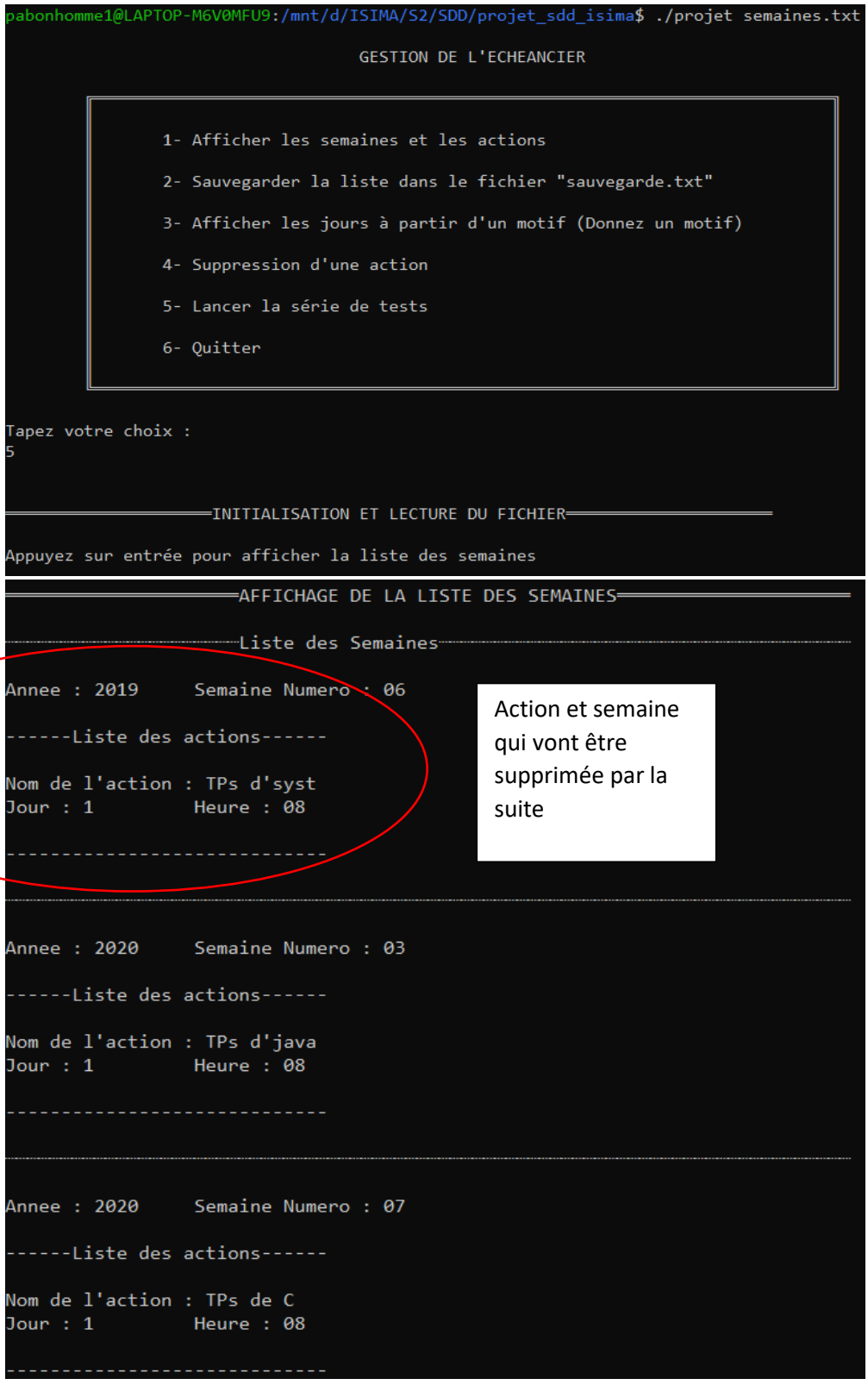


Figure 11 : Test de l'affichage

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

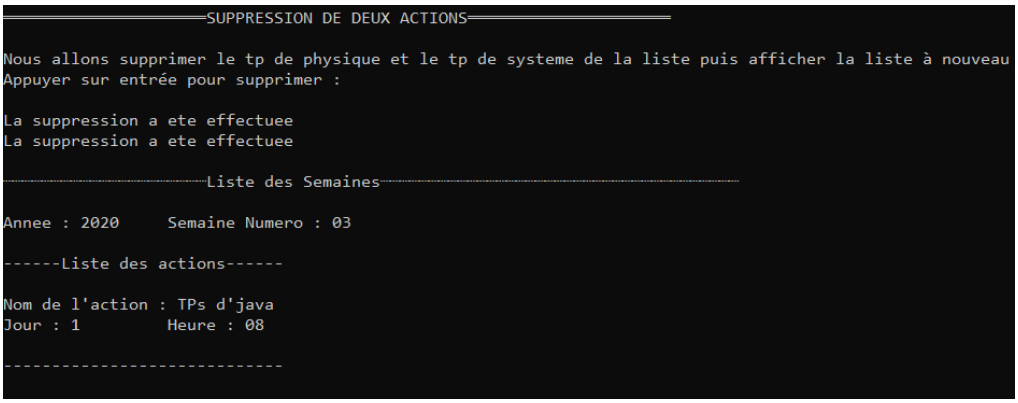


Figure 12 : Test de l'affichage

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.

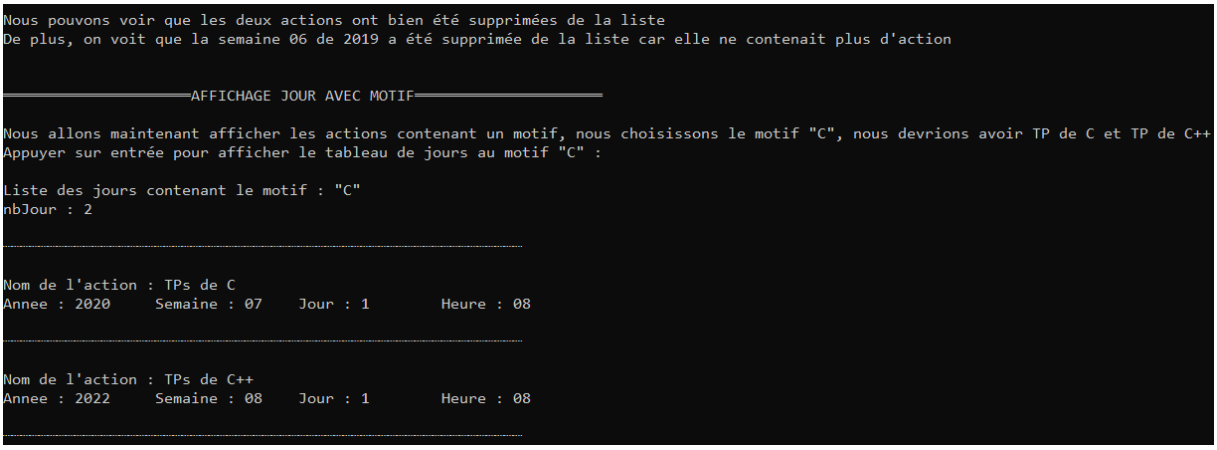


Figure 13 : Test de la recherche de motif « C »

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.

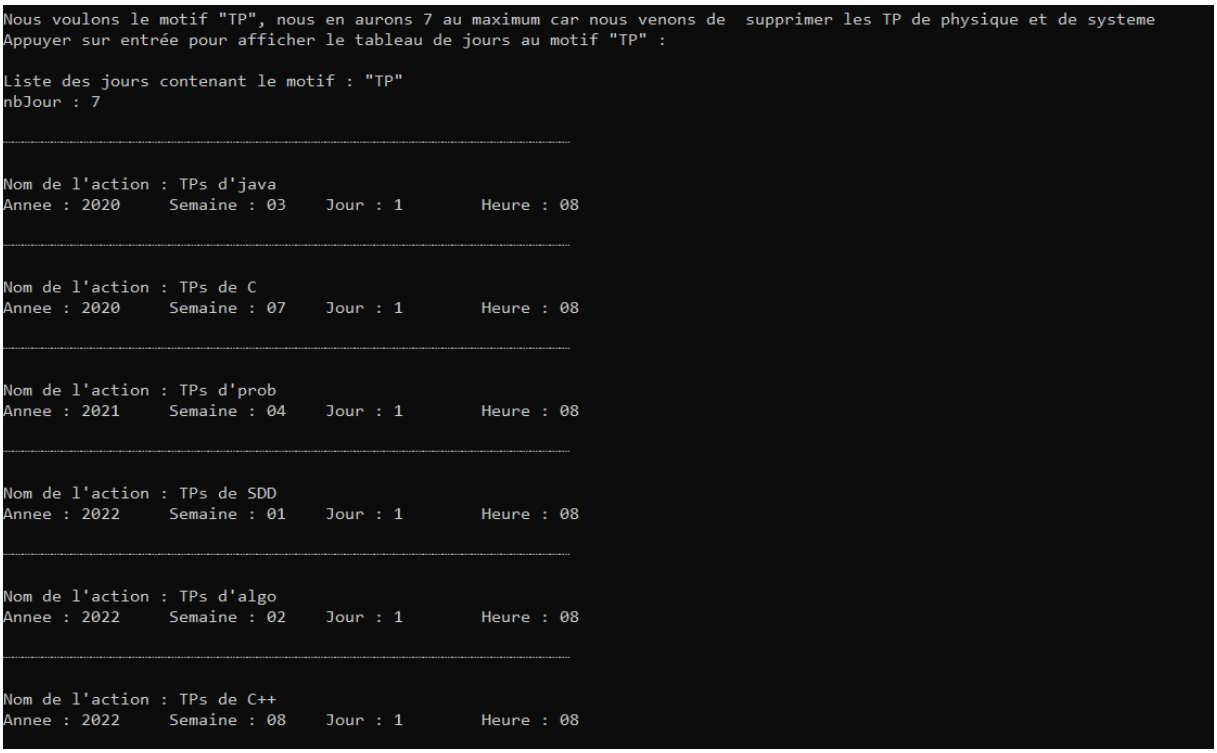


Figure 14 : Test de la recherche de motif « TP »

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.

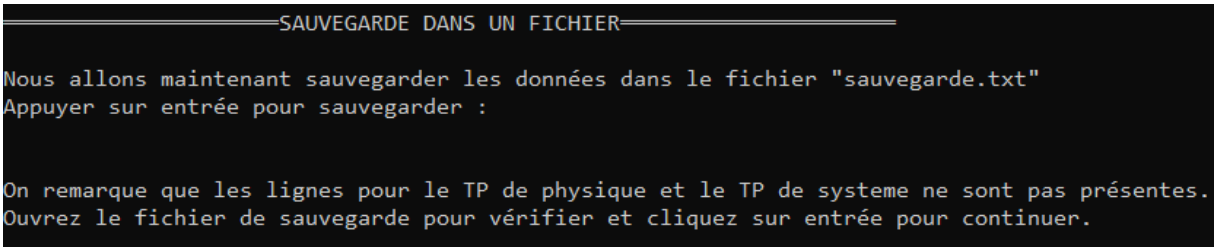


Figure 15 : Test de la sauvegarde

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

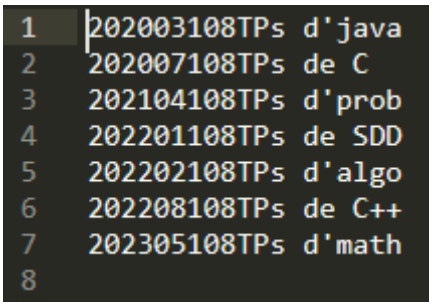


Figure 16 : Fichier « sauvegarde.txt »

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.

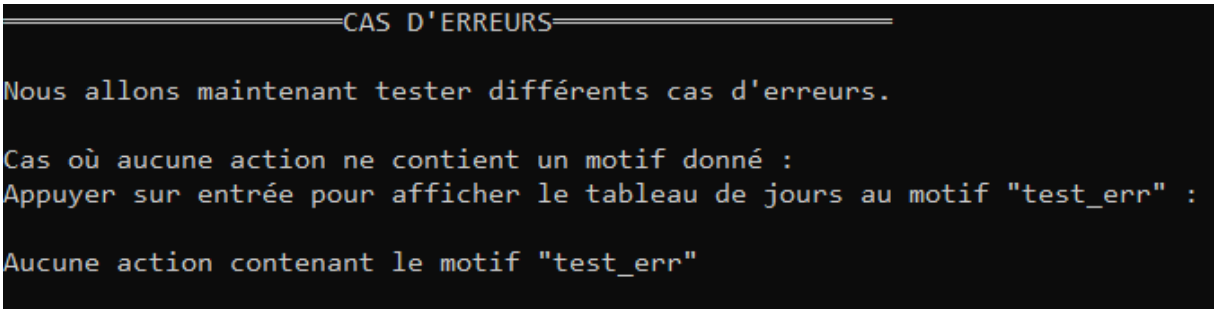


Figure 17 : Tests des cas d'erreurs pour le motif

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 » ».

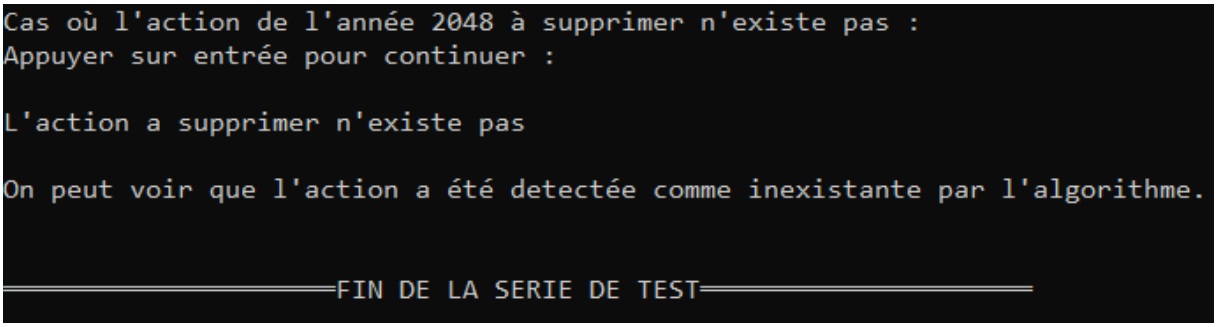
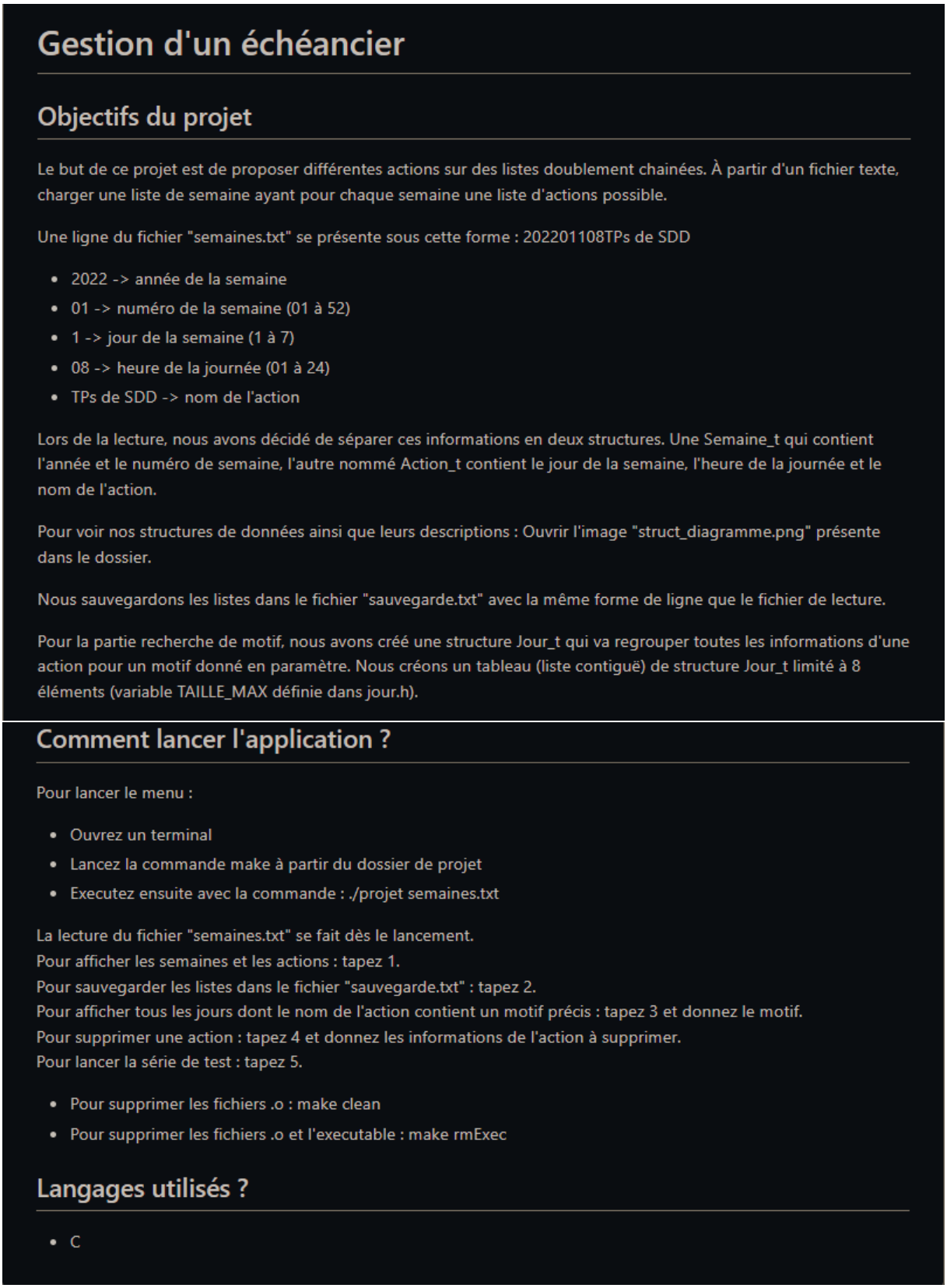


Figure 18 : Tests des cas d'erreurs pour la suppression d'action

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 ».

Vous pouvez essayer vous-même ces fonctionnalités en exécutant le projet dans un terminal. Référez-vous au README pour trouver les instructions à suivre.

4. Annexes



Gestion d'un échéancier

Objectifs du projet

Le but de ce projet est de proposer différentes actions sur des listes doublement chainées. À partir d'un fichier texte, charger une liste de semaine ayant pour chaque semaine une liste d'actions possible.

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

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.

Pour voir nos structures de données ainsi que leurs descriptions : Ouvrir l'image "struct_diagramme.png" présente dans le dossier.

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

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) de structure Jour_t limité à 8 éléments (variable TAILLE_MAX définie dans jour.h).

Comment lancer l'application ?

Pour lancer le menu :

- Ouvrez un terminal
- Lancez la commande make à partir du dossier de projet
- Executez ensuite avec la commande : ./projet semaines.txt

La lecture du fichier "semaines.txt" se fait dès le lancement.
Pour afficher les semaines et les actions : tapez 1.
Pour sauvegarder les listes dans le fichier "sauvegarde.txt" : tapez 2.
Pour afficher tous les jours dont le nom de l'action contient un motif précis : tapez 3 et donnez le motif.
Pour supprimer une action : tapez 4 et donnez les informations de l'action à supprimer.
Pour lancer la série de test : tapez 5.

- Pour supprimer les fichiers .o : make clean
- Pour supprimer les fichiers .o et l'executable : make rmExec

Langages utilisés ?

- C

Figure 19 : Captures d'écrans du README