TP de SDD : les listes chaînées

Chloé Berthold & Rémi Chassagnol

Février 2022

Dans ce TP, il est question de créer et de gérer une structure d'agenda à l'aide d'une liste chaînée à deux niveaux.

Table des matières

Première partie
Présentation générale

1 Description de l'objet du TP

Le but de ce TP est de créer et de gérer une structure d'agenda grâce à une liste chaînée à deux niveaux. La liste chaînée du premier niveau contient l'année et la semaine ainsi qu'une liste chaînée des actions à effectuer cette semaine-ci. Chaque action contient un nom, un jour et une heure. Pour gérer cet agenda, on propose de créer l'agenda correspondant à un fichier ayant une syntaxe précise, de sauvegarder un agenda dans un fichier avec cette même syntaxe, de récupérer toutes les actions dont le nom contient un certain motif dans une liste contigûe, ainsi que de supprimer une action connaissant l'année, la semaine, le jour et l'heure de cette action.

1.1 Choix effectués

Pour réaliser ce TP, nous avons fait quelques choix :

— Pour une année, une semaine, un jour et une heure donnés, il ne peut exister qu'une seule et unique action. Ainsi, si l'utilisateur tente d'ajouter une action pour une date à laquelle il existe déjà une action, l'ajout ne sera pas fait. Il lui faudra supprimer l'ancienne action puis ajouter la nouvelle s'il souhaite la remplacer.

— Un utilisateur ne peut pas ajouter d'action pour une année antérieure à l'année 2021. Nous avons estimé qu'il était nécessaire de vérifier l'année donnée et avons décidé d'empêcher de mettre des actions pour une date antérieure à l'année 2021 puisque cela ne présenterait pas un grand intérêt.

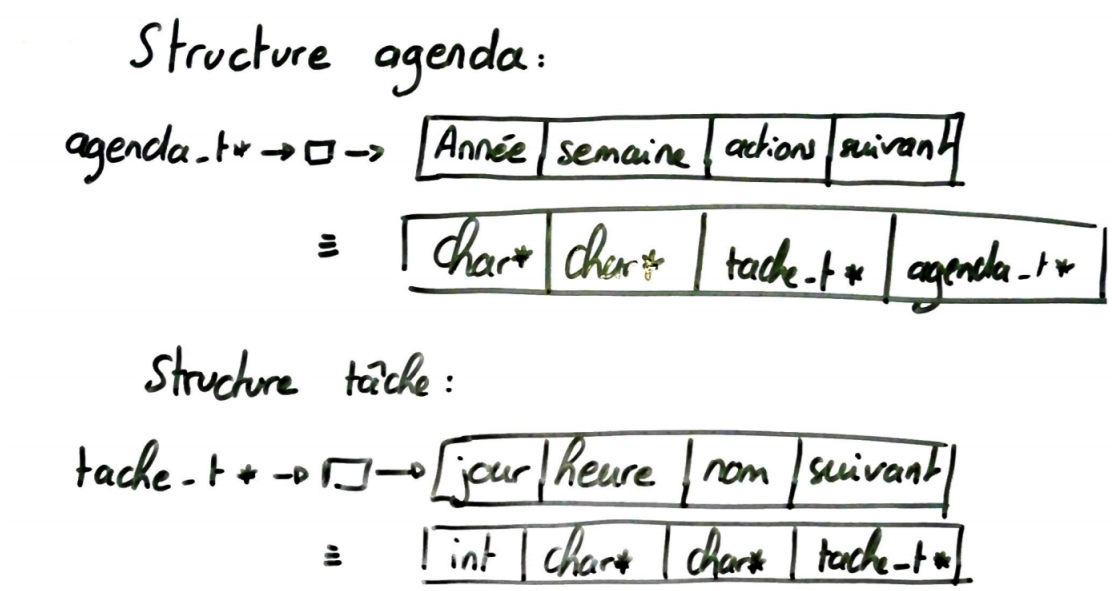
— Lors de la création de la liste contigue des actions contenant dans leur nom un certain motif, nous avons estimé qu'il ne pourrait pas y avoir plus de 30 actions remplissant le critère.

2 Schéma de la structure de donnée

Nous utilisons donc deux structures de données distinctes :

— Une structure *agenda* (nommée *agenda_t*) en liste chaînée contenant donc un tableau de caractères de 5 caractères pour l'année (quatre caractères pour l'année plus un caractère de fin de chaîne de caractère), un tableau de 3 caractères pour la semaine (trois caractères pour le numéro de semaine plus un caractère de fin de chaîne de caractère), un pointeur vers une liste d'actions ainsi qu'un pointeur vers l'élément suivant.

— Une structure *tache* (nommée *tache_t*) qui ici nous permet de gérer la liste des actions. Cette structure contient donc une itération d'une énumaration nommée *jour_t* pour le jour (qui peut donc être considéré comme un entier), un tableau de 3 caractères pour l'heure (deux caractères pour l'heure plus un caractère de fin de chaîne de caractère), un tableau de 11 caractères pour le nom de la tâche (10 caractères pour le nom plus un caractère de fin de chaîne de caractère) ainsi qu'un pointeur vers la tâche suivante.



3 Schéma des fichiers d'entrée

Les fichiers d'entrée sont composés de lignes de caracrètes. Chaque ligne correspond à une tâche de l'agenda et est construite ainsi :

- Les quatre premiers caractères sont des chiffres représentant l'année de l'action.
- Les deux suivants sont des chiffres représentant le numéro de la semaine de l'action pour l'année considérée.
- Le suivant est un chiffre correspondant au numéro du jour de l'action, soit 1 pour lundi jusqu'à 7 pour dimanche.
- Les deux suivants sont des chiffres correspondant à l'heure de l'action.
- Les dix derniers correspondent au nom donné à l'action. Si le nom est censé comporter moins de dix caractères, la ligne est remplie d'espaces pour compléter les dix caractères réservés.

4 Organisation du code source

Le code source est séparé en quatre fichiers :

- Le fichier *main.c* contient la fonction principale du programme ainsi que les sous-fonctions qui s'y rapportent, c'est-à-dire les fonctions relatives à la boucle principale et à ses options.
- Le fichier *agenda.c* regroupe les fonctions permettant de gérer la structure de données *agenda*, c'est-à-dire sa création et sa libération ainsi que les fonctions de manipulations telles qu'ajouter un élément à l'agenda, créer un agenda à partir d'un fichier, supprimer un élément, sauvegarder un agenda dans un fichier texte, afficher un agenda ainsi qu'une fonction de comparaison de deux éléments d'un agenda et les fonctions nécessaires à la création d'une liste contigue d'actions dont le nom contient un certain motif.
- Le fichier *tache.c* regroupe les fonctions permettant de gérer la structure de données *tache*, c'est-à-dire la création et libération d'une tâche ainsi que les fonctions permettant les manipulations de cette structure telles que l'ajout ou la suppression d'une tâche dans une liste de tâche, la comparaison de deux tâches et l'affichage d'une liste de tâches, qu'elle soit contigue ou chaînée, et la transformation d'une tâche en une chaîne de caractères.

— Le fichier *util.c* regroupe toute fonction annexe, soit les fonctions d'affichages du menus, de messages, les fonctions de demandes d'informations, de traitement d'une chaîne de caractère ou de recherche d'un motif.

Deuxième partie

Algorithme de principe des fonctions

5 Fonctions du fichier main.c

5.1 Fonction main

```
1       int main(int argc, char **argv)
```

Le programme principal s'exécute comme suit :
Si le programme a été appelé avec l'argument *"import"* suivi d'un nom de fichier, il va récupérer l'agenda dans le fichier précisé (cf fonction *agendaViaFichier*). Sinon, elle ne fait rien et laisse l'agenda vide. Enfin, elle lance la boucle principale du programme sur l'agenda créé (cf fonction *run*).

5.2 Fonction run

```
1       void run(agenda_t *agenda)
```

La fonction run consiste en l'exécution de l'application. Tant que la variable *continuer* est différente de zéro, elle effectue le traitement suivant :

- Elle affiche affiche l'interface utilisateur (cf fonction *afficheMenu*).
- Après quoi, elle attend le choix de l'utilisateur qu'elle enregistre dans la variable *reponse*.
- Enfin, elle modifie l'agenda qui lui était passé en paramètre si besoin en fonction de la réponse utilisateur (cf fonction *gestionEvenements*).

A la fin de cette boucle, la fonction termine par libérer l'agenda qu'elle a utilisé (cf fonction *freelst*).

5.3 Fonction gestionEvenements

```
1       agenda_t *gestionEvenements(char entree, agenda_t *agenda, int *continuer)
```

Cette fonction permet de gérer les choix de l'utilisateur via le paramètre *entree* :

- Si l'entrée est "0", l'utilisateur a souhaité ajouter un élément à son agenda. Pour cela, la fonction récupère auprès de l'utilisateur les informations dont elle a besoin (cf fonction *demandeInfos* et fonction *demandeNom*), puis met à jour l'agenda de l'utilisateur avec le nouvel élément (cf fonction *ajouteAgenda*).
- Si l'entrée est "1", l'utilisateur a souhaité supprimer un élément. Ici aussi, la fonction a besoin d'informations (cf fonction *demandeInfos* ; ici, elle n'a pas besoin du nom de la tâche à supprimer). Enfin, elle met à jour l'agenda en conséquence (cf fonction *supprimeElt*).
- Si l'entrée est "2", l'utilisateur souhaite afficher son agenda (cf fonction *afficheAgenda*).
- Si l'entrée est "3", l'utilisateur souhaite sauvegarder son agenda dans un fichier. Pour cela, le programme demande le nom du-dit fichier à l'utilisateur avant de recopier l'agenda dans un fichier dont le nom lui est donné en paramètre (cf fonction *sauvFichier*).
- Si l'entrée est "4", l'utilisateur souhaite importer un agenda d'un fichier. Le programme demande donc ici aussi un nom de fichier à l'utilisateur. Si la fonction *demandeConfir-*

mation, qui informe simplement l'utilisateur que son agenda va être remplacé par celui extrait et lui demande une confirmation, retourne un nombre différent de 0, alors le programme libère l'ancien agenda (cf fonction *freelst*) avant d'extraire le nouvel agenda (cf fonction *agendaViafichier*).

— Si l'entrée est "5", l'utilisateur souhaite rechercher un motif dans le nom des tâches présentes dans son agenda. Pour cela, le programme demande le motif que veut trouver l'utilisateur qu'il stocke dans la variable *motif* avant de créer la liste contigue des tâche dont le nom contient le motif (cf fonction *creerListeContigue*). Après cela, elle affiche la liste des tâches récupérées jusqu'à ce que l'utilisateur appuie sur la touche 'Entrer'.(cf fonction *afficheListeContigue*).

— Si l'entrée est un autre caractère que ceux listés, alors l'utilisateur souhaite quitter l'application. Ainsi, la fonction affiche un message de fin avec la fonction *afficheMessageFin* avant de modifier la variable *continuer* en la mettant à 0 pour que le programme puisse sortir de la boucle principale (cf fonction *run*).

Finalement, après avoir traité le choix de l'utilisateur, la fonction termine par libérer le pointeur *deb* avant de retourner l'agenda, qu'il ait été modifié ou non.

6 Fonctions du fichier agenda.c

6.1 Fonction creeAgenda

```
1       agenda_t *creeAgenda(char annee[TAILLE_ANNEE], char semaine[TAILLE_SEMAINE])
```

Cette fonction permet l'allocation dynamique et l'initialisation des champs d'un nouvel agenda, c'est-à-dire l'année et la semaine qui lui sont passées en paramètres.
La fonction commence par allouer dynamiquement un élément de type *agenda_ t*. Si l'allocation s'est bien passée, elle recopie ses paramètres dans les champs correspondants de l'agenda nouvellement alloué et initialise ses pointeurs *suiv* et *actions* à NIL pour signifier que l'agenda est le dernier élément de la liste et qu'il n'y a pas d'actions pour cette année et cette semaine précises. Enfin, il retourne l'adress indirecte de l'agenda, que l'allocation se soit bien passée ou non.

6.2 Fonction freelst

```
1       void freelst(agenda_t **agenda)
```

Cette fonction permet la libération de la liste chaînée d'agendas commençant par l'agenda qui lui est passé en paramètre.
Pour cela, si l'agenda qui lui est donné est valide, elle définie un pointeur courant et un pointeur suivant. Le courant est initialisé sur l'adresse indirecte de l'agenda. Tant que le courant n'est pas null, c'est-à-dire qu'il n'a pas parcouru toute la liste chaînée, la fonction définit la variable suivant comme étant le champs suivant du courant, libère les actions du courant (cf fonction *freeTache*) avant de libérer le courant et de passer la valeur de la variable *suiv* au courant pour continuer le parcours. A la fin de cette boucle, la fonction se termine en modifiant le pointeur sur l'agenda pour qu'il soit invalide.

6.3 Fonction parseLigne

```
1       void parseLigne(char *s, char annee[TAILLE_ANNEE], char
            semaine[TAILLE_SEMAINE],
2               jour_t *jour, char heure[TAILLE_HEURE], char nom[TAILLE_NOM])
```

Cette fonction récupère les informations contenue par une ligne d'un fichier lors d'une importation.
Pour ce faire, elle fait à la fois avancer le pointeur dans la chaîne de caractères qui lui est donnée et remplir ses autres paramètres (cf fonction *parseSuiteNombre*). Pour chaque chaîne de caractères, elle ajoute le caractère " après l'appel à la sous-fonction. Enfin, pour le nom, elle se contente de recopier ce qu'il reste dans la chaîne de caractère *s*.

6.4 Fonction ajouteAgenda

```
1       agenda_t *ajouteAgenda(agenda_t *agenda, char annee[TAILLE_ANNEE],
2                       char semaine[TAILLE_SEMAINE], jour_t jour,
3                       char heure[TAILLE_HEURE],
4                       char nom[TAILLE_NOM])
```

Cette fonction permet l'ajout d'un élément dans la liste triée contenant les éléments de l'agenda à partir des informations passées en paramètres.
La fonction parcourt la liste chaînée représentant l'agenda jusqu'à trouver un élément correspondant aux informations d'année et de semaine données en paramètre.

- Si elle trouve un tel élément, cela signifie qu'il existe déjà au moins une action pour la même année et semaine. Elle fait alors ajoute la tâche correspondant aux informations en paramètre à la liste de tâches de l'élément (cf fonction *ajouteTache*).
- Si elle ne le trouve pas, elle crée un nouvel élément d'agenda correspondant aux informations données en paramètre (cf fonction *creeAgenda*). Si cette création a bien eu lieu, elle l'ajoute dans l'agenda courant, à l'endroit où la recherche s'est arrêtée et fait ensuite crée la tâche correspondant aux informations données dans la liste de tâches de l'élément nouvellement créé (cf fonction *ajouteTache*).

Enfin, la fonction retourne le pointeur sur la tête de l'agenda qui a été modifié.

6.5 Fonction listeMotif

```
1       tache_t **listeMotif(agenda_t *agenda, tache_t **deb, char *motif)
```

Cette fonction trouve les jours où le nom de l'action correspond au motif donné et met l'adresse de la tâche correspondante dans le tableau *deb*.
Pour cela, la fonction parcoure l'agenda qui lui est donné en paramètre et pour chaque élément parcoure aussi sa liste d'actions. Pour chaque action, elle vérifie si le motif est dans son nom (cf fonction *motifCorrespond*). Si c'est le cas, la fonction ajoute dans le tableau l'adresse de la tâche et continue son parcours. Enfin, la fonction retourne l'adresse de fin de la liste contigue.

6.6 Fonction creerListeContigue

```
1       void creerListeContigue(agenda_t *agenda, tache_t ***deb, tache_t ***fin,
2                       char *motif)
```

Cette fonction génère la liste contigue des tâches dont le nom contient un motif donnée.
Elle commence par allouer un tableau de 30 tâches (on suppose qu'il ne peut y avoir plus de 30 tâches dont le nom contient le motif donné) et si cette allocation a fonctionné, elle crée la liste et récupère l'adresse de fin du tableau dans le pointeur *fin* (cf fonction *listeMotif*). Si l'allocation a échoué, la fonction affiche un message d'erreur.

6.7 Fonction agendaViafichier

```
1       agenda_t *agendaViafichier(char *nom)
```

Cette fonction permet de créer un agenda en extrayant des informations d'un fichier dont le nom est donné en paramètre.
La fonction ouvre d'abord le fichier dont le nom lui est donné en paramètre en lecture et, si l'ouverture a fonctionné, il le lit ligne par ligne grâce à un buffer jusqu'à l'avoir lu en entier. Pour chaque ligne lue, elle sépare les informations qu'elle contient (cf fonction *parseLigne*) et ajoute à un agenda qu'elle aura initialisé à vide l'élément correspondant aux informations (cf fonction *ajouteAgenda*. Une fois qu'elle a lu le fichier en entier, elle le ferme et retourne l'agenda créé et complété à partir des informations qu'il contient. Si l'ouverture n'a pas abouti, la fonction affiche un simple message d'erreur et retourne un pointeur null.

6.8 Fonction ecritFichier

```
1       void ecritFichier(FILE *f, agenda_t *agenda)
```

Cette fonction permet d'écrire les données d'un agenda dans un fichier déjà ouvert au préalable. Pour cela, elle vérifie que le fichier à bien été ouvert avant son appel.

- Si c'est le cas, la fonction parcout alors l'agenda qui lui est donné en paramètre
  - Lorsqu'elle trouve un élément, elle transforme ses données d'année et de semaine en une chaîne de caractère dans un buffer (cf fonction *agendaToString*). Après quoi, elle parcout la liste des actions qui est associé à cet élément.
    - Pour chaque action, elle complète le buffer avec la transformation en chaîne de caractère des données de jour, d'heure et de nom de la tâche (cf fonction *tacheToString*) avant d'écrire ce buffer dans le fichier ouvert.

6.9 Fonction sauvFichier

```
1       void sauvFichier(char *nom, agenda_t *agenda)
```

Cette fonction permet la sauvegarde d'un agenda dans un fichier dont le nom est donné en paramètre.
Elle commence par ouvrir le fichier en écriture. Si l'ouverture a bien eu lieu, elle y écrit les données de l'agenda (cf fonction *ecritFichier*) avant de le refermer. Si l'ouverture a échoué, elle se contente d'afficher un message d'erreur sans faire aucune autre manipulation.

6.10 Fonction supprimeElt

```
1       agenda_t *supprimeElt(agenda_t *agenda, char annee[TAILLE_ANNEE],
2                             char semaine[TAILLE_SEMAINE], jour_t jour,
3                             char heure[TAILLE_HEURE])
```

Cette fonction recherche dans un agenda l'action programmée pour l'année, la semaine, le jour et l'heure donnés en paramètre et la supprime si elle existe. Elle supprime l'élément de l'agenda correspondant à la semaine et à l'année si sa liste d'actions est vide à la fin de la suppression de l'action.
Ainsi, elle parcout l'agenda en faisant chaque fois une comparaison entre l'année donné en paramètre et l'année de l'élément considéré puis entre la semaine donnée en paramètre et la semaine de l'élément considéré (cf fonction *compAgendaElt*).

- Si la fonction a trouvé une correspondance, alors elle cherche et supprime l'action dans la liste d'actions de l'élément (cf fonction *supprimeTache*. Si à la fin de cette suppression la liste d'actions de l'élément est vide, la fonction supprime l'élément.
- Si la fonction n'a pas trouvé de correspondance, il n'existe pas de tâche correspondant aux informations données, elle ne fait donc rien.

Finalement, elle retourne le pointeur sur la tête de l'agenda, qu'il ait été modifié ou non.

6.11 Fonction compAgendaElt

```
1       int compAgendaElt(agenda_t *elt, char annee[TAILLE_ANNEE],
2                 char semaine[TAILLE_SEMAINE])
```

Cette fonction compare un élément de l'agenda à une année et à un numéro de semaine donnés en paramètre.
Elle compare d'abord les chaînes de caractères concernant l'année grâce à la fonction *strcmp*. Si elles sont égales, elle compare alors les semaines, toujours grâce à la fonction *strcmp*. Elle retourne le résultat de ces comparaisons, c'est-à-dire :

- 0 si les années et les semaines sont identiques.
- Une valeur positive si l'année de l'élément est supérieure à celle donnée en paramètre ou si les années sont identiques mais que la semaine de l'élément est supérieure à celle donnée en paramètre.
- Une valeur négative si l'année de l'élément est inférieure à celle donnée en paramètre ou si les années sont identiques mais que la semaine de l'élément est inférieure à celle donnée en paramètre.

6.12 Fonction agendaToString

```
1       void agendaToString(agenda_t *agendaElt, char buff[])
```

Cette fonction converti les données d'un élément de l'agenda en une chaîne de caractères.
Elle écrit tout d'abord l'année dans le buffer qui lui est donné en écrasant tout ce qui y était écrit, puis y accole la semaine de l'élément.

6.13 Fonction afficheAgendaElt

```
1       void afficheAgendaElt(agenda_t *agendaElt)
```

Cette fonction ne sert qu'à afficher un élément unique d'un agenda.
Pour cela, elle affiche son année et sa semaine avant d'afficher la liste des actions qui s'y rapportent (cf fonction *afficheTache*.

6.14 Fonction afficheAgenda

```
1       void afficheAgenda(agenda_t *agenda)
```

Cette fonction affiche un agenda complet.
Si l'agenda est vide, elle n'affiche qu'un panneau d'avertissement pour prévenir l'utilisateur que son agenda est vide. Sinon, elle parcout l'agenda entier et en affiche chaque élément un à un (cf fonction *afficheAgendaElt*). Elle affiche enfin un simple message recommandant à l'utilisateur d'appuyer sur la touche Entrer pour quitter l'affichage.

7 Fonctions du fichier tache.c

7.1 Fonction creeTache

```
1       tache_t *creeTache(char j, char heure[TAILLE_HEURE], char nom[TAILLE_NOM])
```

Cette fonction permet de créer une tâche contenant les informations données en paramètre.
Pour cela, elle tente d'allouer l'espace nécessaire à une tâche. Si cette allocation a fonctionné, elle copie les informations en paramètre dans les bons champs de la tâche nouvellement allouée et initialise son champs *suiv* à NULL. Elle retourne le pointeur sur la tâche nouvellement créée, que l'espace ait été allouée ou non.

7.2 Fonction freeTache

```
1       void freeTache(tache_t **tache)
```

Cette fonction libère la mémoire allouée pour une liste de tâches.
Si la liste qui lui est donnée n'est pas vide, elle la parcout jusqu'au dernier élément et les libère un à un grâce à un pointeur *suiv* et à un pointeur *cour*. Enfin, elle modifie le pointeur sur la tête qui lui a été donnée à NULL.

7.3 Fonction ajouteTache

```
1       tache_t *ajouteTache(tache_t *tache, jour_t jour, char heure[TAILLE_HEURE],
2                   char nom[TAILLE_NOM])
```

Cette fonction permet d'ajouter une tâche dans une liste triée de tâches s'il n'existe pas déjà une tâche au jour et à l'heure précisés.
Pour cela, elle parcout la liste des tâches en comparant chaque tâche aux données recherchées pour l'insérer au bon endroit (cf fonction *compTache*).

- Si elle trouve un élément correspondant au jour et à l'heure données, elle ne fait rien, l'ajout n'a pas lieu.
- Sinon, elle crée la tâche correspondante (cf fonction *creeTache*) et, si elle a bien été créée, elle l'ajoute à l'endroit où elle s'est arrêtée dans sa recherche.

Enfin, elle retourne le pointeur sur la tête de la liste de tâches qu'elle a modifié.

7.4 Fonction supprimeTache

```
1       tache_t *supprimeTache(tache_t *tache, jour_t jour, char heure[TAILLE_HEURE])
```

Cette fonction supprime la tâche programmée pour le jour et à l'heure donnés en paramètre dans une liste de tâches.
Elle parcout donc la liste en vérifiant pour chaque élément s'il correspond aux données (cf fonction *compTache*). Si la tâche à supprimer existe, elle le supprime de la liste et le libère. Enfin, elle retourne le pointeur sur la tête de la liste de tâches qu'elle a manipulée.

7.5 Fonction compTache

```
1       int compTache(tache_t *elt, char j, char heure[TAILLE_HEURE])
```

Cette fonction fait la comparaison entre une tâche donnée et des informations de jour et d'heure données en paramètre.
Pour comparer le jour, puisque ce sont de simples nombres, elle évalue le résultat du retranchement du jour en paramètre au jour de l'élément.

- Si ce résultat est 0, alors les deux jours sont identiques. Dans ce cas-ci, elle compare ensuite les semaines. Pour cela, elle parcout les deux chaînes de caractères en comparant d'abord le caractère des dizaines grâce à une soustraction puis celui des unités.
- Sinon, elle n'a pas besoin de comparer les heures.

Ainsi, la fonction retournera 0 si l'élément correspond aux données en paramètre, une valeur positive si la tâche est prévue pour plus tard que les données en paramètre et une valeur négative si elle est prévue plus tôt.

7.6 Fonction tacheToString

```
1       void tacheToString(tache_t *tache, char buff[])
```

Cette fonction converti les données d'une tâche en une chaîne de caractères sachant que le buffer qui lui est donné contient déjà une année et une semaine.
Pour cela, elle remplace le caractère nul par le jour de l'élément et ajoute à la suite le caractère nul. Ensuite, elle concatène l'heure de l'élément grâce à la fonction *strcat* et enfin le nom de la tâche.

7.7 Fonction afficheTache

```
1       void afficheTache(tache_t *taches)
```

Cette fonction affiche l'ensemble des tâche d'une liste de tâche dont le pointeur sur la tête lui est donnée en paramètre.
Elle parcout donc toute la liste et affiche, pour chaque élément, son jour, son heur et son nom.

7.8 Fonction afficheListeContigue

```
1       void afficheListeContigue(tache_t **deb, tache_t **fin)
```

Cette fonction affiche une liste contigue de tâches dont l'adresse de début et l'adresse de fin sont données en paramètre.

Elle parcout la liste contigue en faisant bouger le pointeur de début jusqu'à ce qu'il pointe au même endroit que le pointeur de fin. Pour chaque nouvel élément, elle affiche le jour, l'heure et le nom.

8 Fonctions du fichier util.c

8.1 Fonction parseSuiteNombres

```
1       char *parseSuiteNombres(char *s, char tab[], int n)
```

Cette fonction permet de récupérer une suite de n nombres dans une chaîne de caractères.
Elle parcourt sur n caractères la chaîne de caractères source qui lui est donnée en paramètre en modifiant le pointeur de cette chaîne pour ainsi 'consommer' les caractères lus et les inscrits dans les cases d'un tableau de caractères. Une fois qu'elle a lus n caractères où qu'elle a atteint la fin de la chaîne source, elle retourne le pointeur sur la chaîne source qu'elle a déplacé.

8.2 Fonction flushInput

```
1       void flushInput()
```

Cette fonction permet de vider le buffer d'input utilisateur.
Pour cela, la fonction récupère caractère par caractère les entrées restantes dans *stdin*, jusqu'à atteindre soit un saut de ligne, soit la fin du buffer d'input.

8.3 Fonction motifCorrespond

```
1       int motifCorrespond(char *motif, char nom[TAILLE_NOM])
```

Cette fonction recherche dans une chaîne de caractères donnée un motif donné en paramètre.
Pour cela, elle parcourt la chaîne de caractère caractère par caractère et compare à chaque fois le motif à la chaîne de caractère débutant au caractère parcouru et de la même taille que le motif. La boucle s'arrête si le motif a été trouvé ou s'il ne reste plus assez de caractères dans la chaîne de caractères pour pouvoir la comparer au motif. Elle retourne True si le motif a été trouvé et False sinon.

8.4 Fonction demandeInfos

```
1       void demandeInfos(char annee[TAILLE_ANNEE], char semaine[TAILLE_SEMAINE],
2               char heure[TAILLE_HEURE], int *jour)
```

Cette fonction n'est utilisée que pour demander les informations d'année, de semaine, de jour et d'heure à l'utilisateur.
La fonction demande d'abord à l'utilisateur une année sur quatre caractères et n'accepte qu'une année supérieure à l'année 2020. Une fois que l'utilisateur a saisi une année correcte, elle demande cette fois-ci une semaine comprise enre 1 et 53. Ensuite, elle demande un jour entre 1 pour lundi et 7 pour dimanche et enfin une heure entre 0 et 24. Au fur et à mesure, elle remplit les champs qui lui ont été donnés en paramètre.

8.5 Fonction demandeNom

```
1       void demandeNom(char nom[TAILLE_NOM])
```

Cette fonction permet de demander à l'utilisateur un nom d'action.
La fonction demande donc un nom sur 10 caractères qu'elle stocke dans la chaîne de caractères qui lui a été donnée en paramètre et vide le buffer d'input (cf fonction *flushInput*).

8.6 Fonction demandeConfirmation

```
1       char demandeConfirmation()
```

Cette fonction demande à l'utilisateur une confirmation lorsqu'il tente d'importer un agenda à partir d'un fichier et qu'il remplacera son agenda actuel. Si l'utilisateur répond 'y', la fonction retourne True, sinon elle retourne False.

8.7 Fonction afficheMenu

```
1       void afficheMenu()
```

Cette fonction sert à afficher un menu utilisateur dont les numéros correspondent à des choix possibles.

8.8 Fonction afficheMessageFin

```
1       void afficheMessageFin()
```

Cette fonction affiche un message de fin lorsque l'utilisateur quitte l'application.

8.9 Fonction clear

```
1       void clear()
```

Cette fonction fait un appel système à la fonction clear du terminal pour assurer un affichage dynamique et organisé de l'application losque le programme n'est pas compilé en mode 'DEBUG'.

Troisième partie

Compte rendu d'exécution : jeux de tests

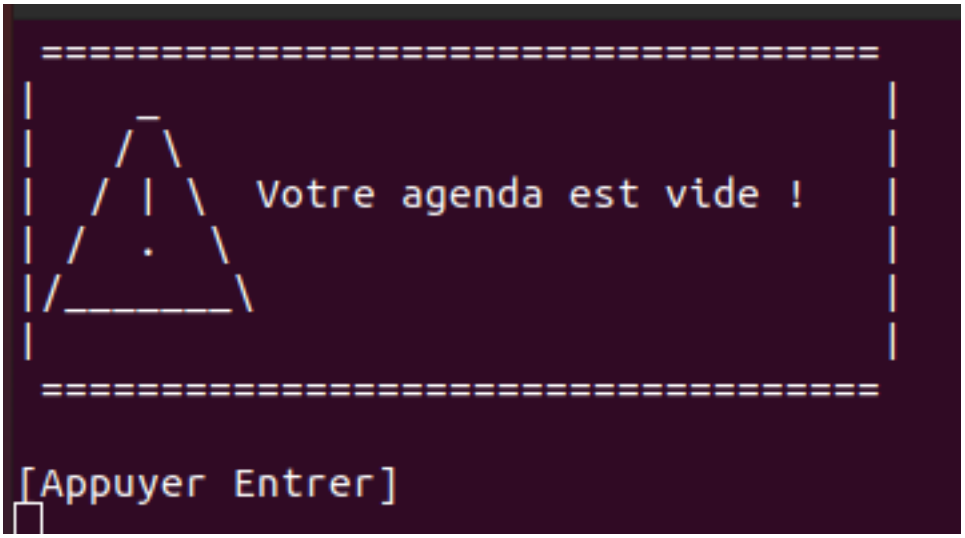
Les tests ont été effectués en mode 'DEBUG', c'est-à-dire que la variable 'DEBUG' du fichier *util.h* a été mise à 1 avant la compilation du programme, ce qui permet de garder l'affichage de l'exécution entière du programme.

9 Tests pour la fonction d'importation

Pour cette fonction (cf fonction *agendaViaFichier*), la liste des cas est la suivante :

9.1 Cas du fichier vide

Si l'utilisateur tente d'importer un agenda à partir d'un fichier vide, la fonction retourne l'agenda suivant :



c'est-à-dire que la fonction ne modifie pas l'agenda vide.

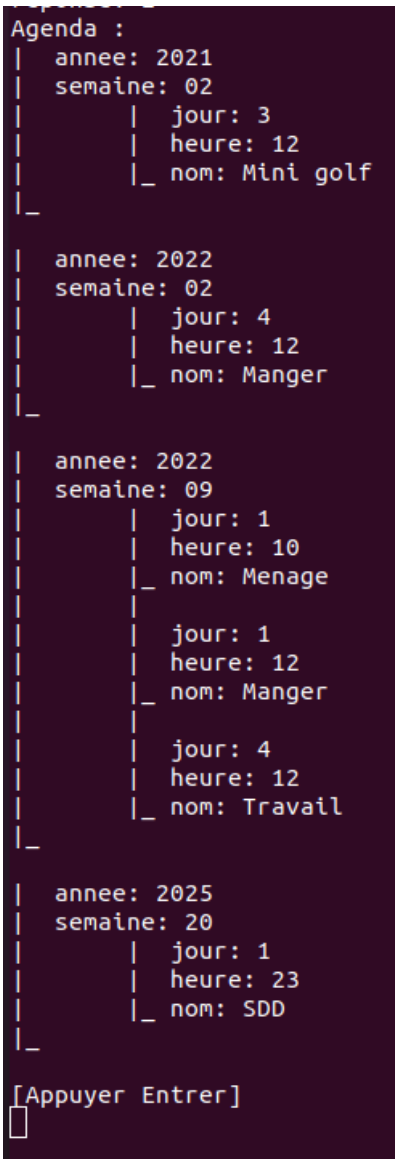
9.2 Cas d'un fichier non vide

9.2.1 Cas d'un fichier trié chronologiquement

Nous utilisons un fichier construit comme suit pour ce test :

```
202102312Mini golf
202202412Manger
202209110Menage
202209112Manger
202209412Travail
202520123SDD
```

La fonction crée l'agenda suivant :

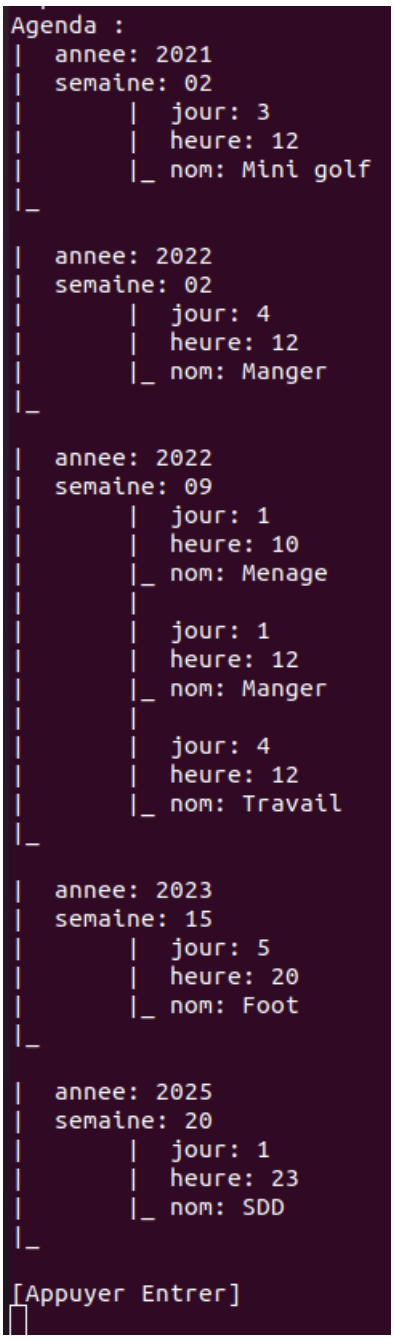


9.2.2 Cas d'un fichier non trié

Nous utilisons un fichier construit comme suit pour ce test :

```
202209412Travail
202520123SDD
202202412Manger
202209112Manger
202315520Foot
202102312Mini golf
202209110Menage
```

La fonction crée l'agenda suivant :

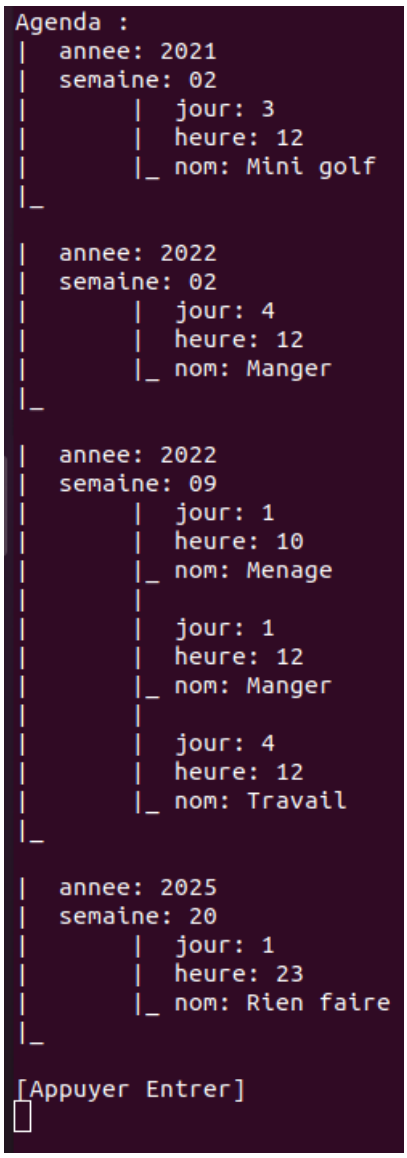


9.3 Cas d'un fichier ayant des lignes redondantes

Si, dans le fichier, il existe plusieurs lignes contenant les mêmes informations temporelles (c'est-à-dire les mêmes années, semaines, jours et heures), la fonction ne garde que la première itération des informations temporelles. Nous utilisons un fichier construit comme suit pour ce test :

```
202102312Mini golf
202202412Manger
202209110Menage
202209112Manger
202209412Travail
202520123Rien faire
202520123SDD
```

La fonction crée alors l'agenda suivant :



10 Tests pour la fonction de création de la liste contigue

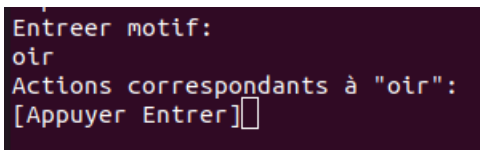
Pour cette fonction (cf fonction *creerListeContigue*), la liste des cas est la suivante :

10.1 Cas où aucun élément ne correspond au motif

Soit le fichier importé :

```
202102312Mini golf
202202412Manger
202209110Menage
202209112Manger
202209412Travail
202520123Rien faire
```

On choisit le motif "oir". La fonction affiche alors le tableau suivant :



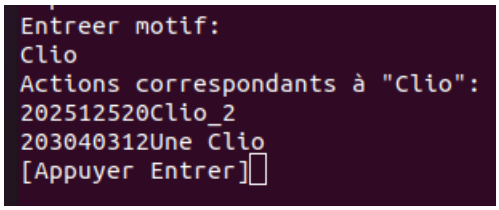
C'est-à-dire un tableau vide.

10.2 Cas où un ou plusieurs éléments correspondent au motif

Soit le fichier importé :

```
202512520Clio_2
202103121Mustang
203040312Une Clio
202101520Peugeot
202842109Ferrari
```

On choisit le motif "Clio". La fonction affiche alors le tableau suivant :

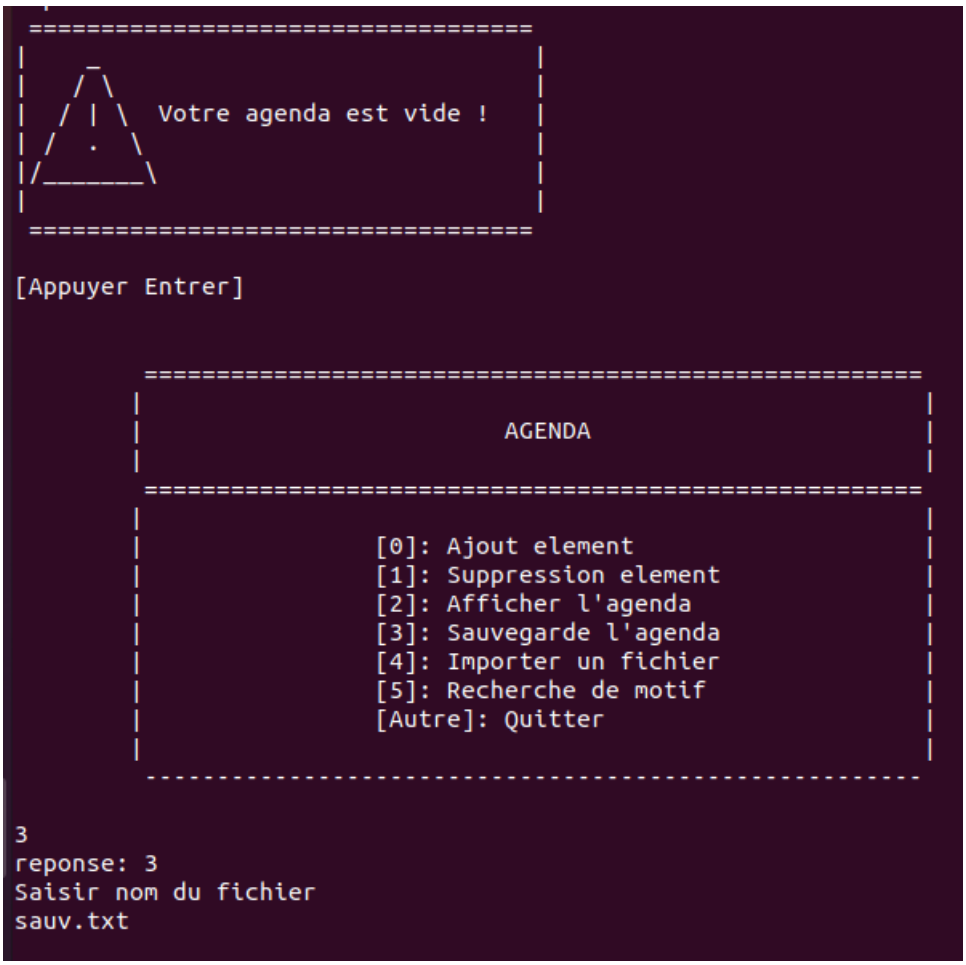


11 Tests pour la fonction de sauvegarde

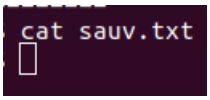
Pour cette fonction (cf fonction *sauvFichier*), la liste des cas est la suivante :

11.1 Cas d'un agenda vide

Pour l'exécution suivante :

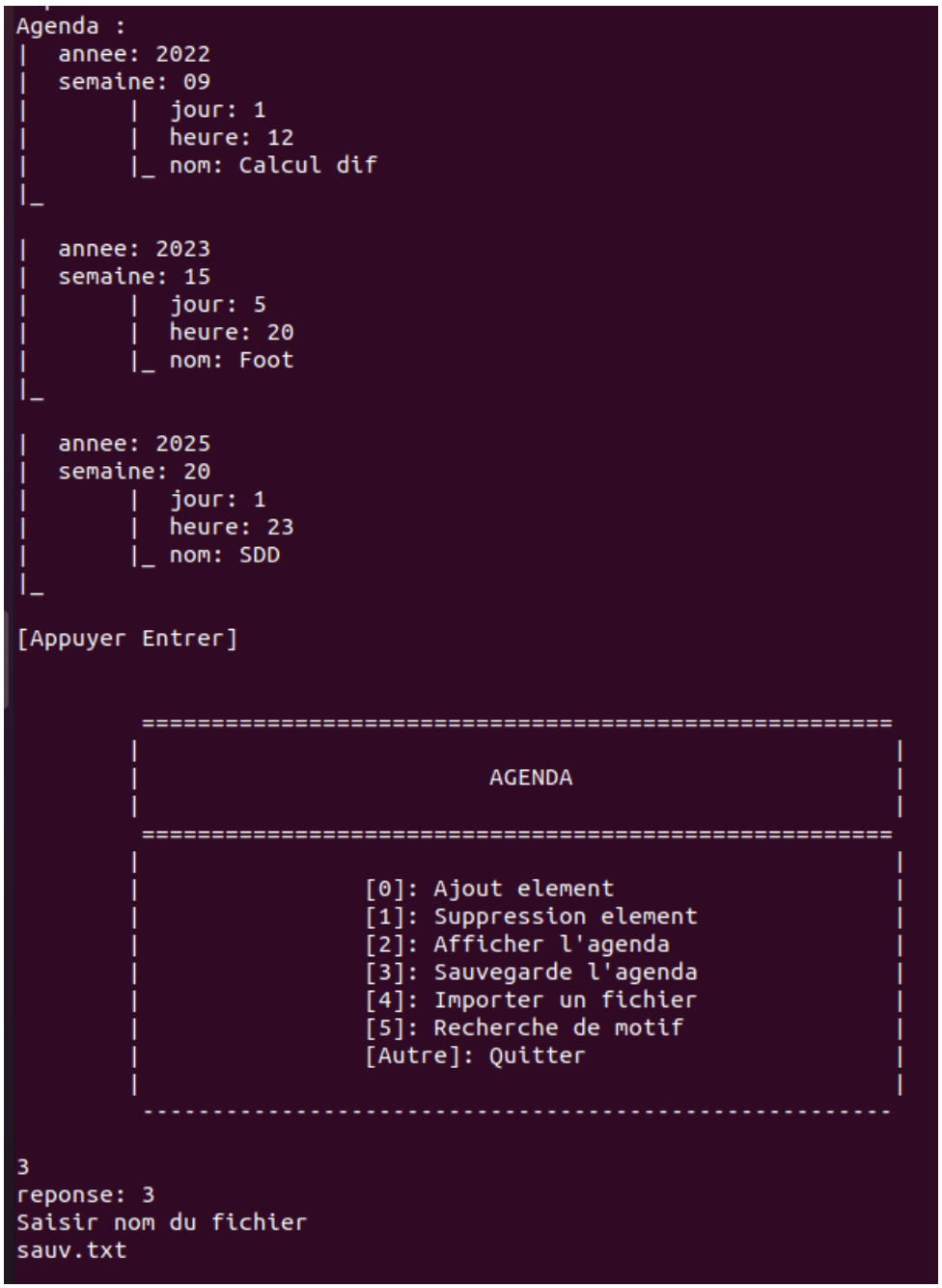


une commande cat du fichier sauv.txt nouvellement créé affiche un fichier vide :



11.2 Cas d'un agenda non vide

Soit l'agenda et l'exécution suivante :



Un appel de la commande cat sur le fichier sauv.txt affiche :

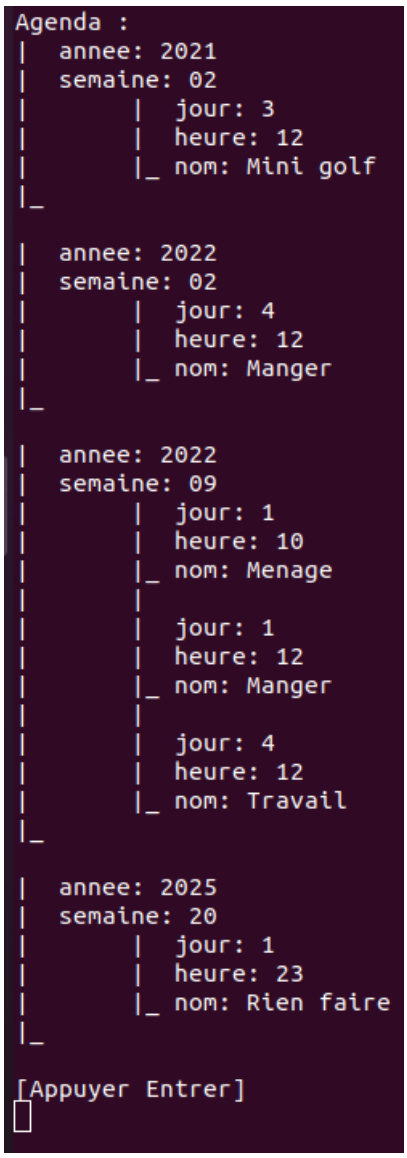
```
202209112Calcul dif
202315520Foot
202520123SDD
```

12 Tests pour la fonction de suppression

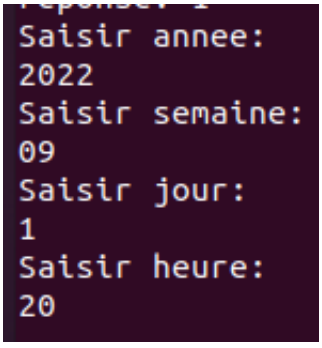
Pour cette fonction (cf fonction *supprimeElt*), la liste des cas est la suivante :

12.1 Cas où l'élément recherché n'existe pas

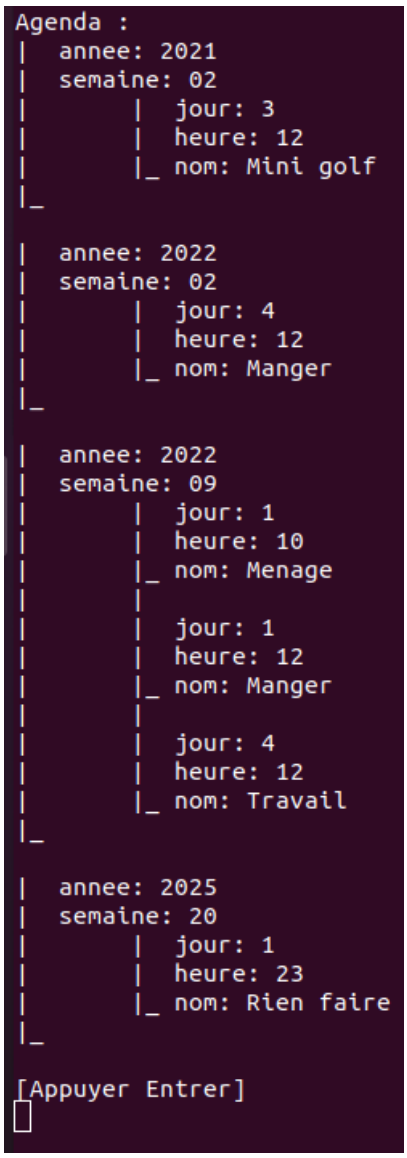
Ce cas concerne aussi un agenda vide. Soit l'agenda suivant :



On tente de supprimer l'élément dont l'année est 2022, le numéro de semaine 09, le jour 1 et l'heure 20.



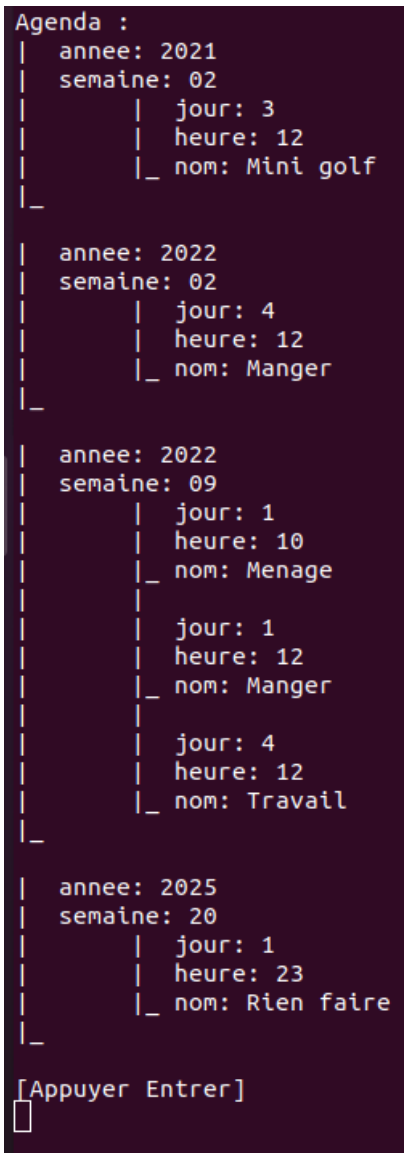
La fonction modifie l'agenda et, à l'affichage, il devient :



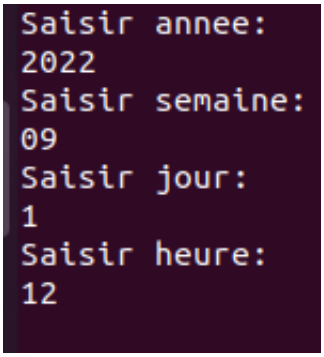
C'est-à-dire qu'il n'a pas été modifié.

12.2 Cas où l'élément recherché existe

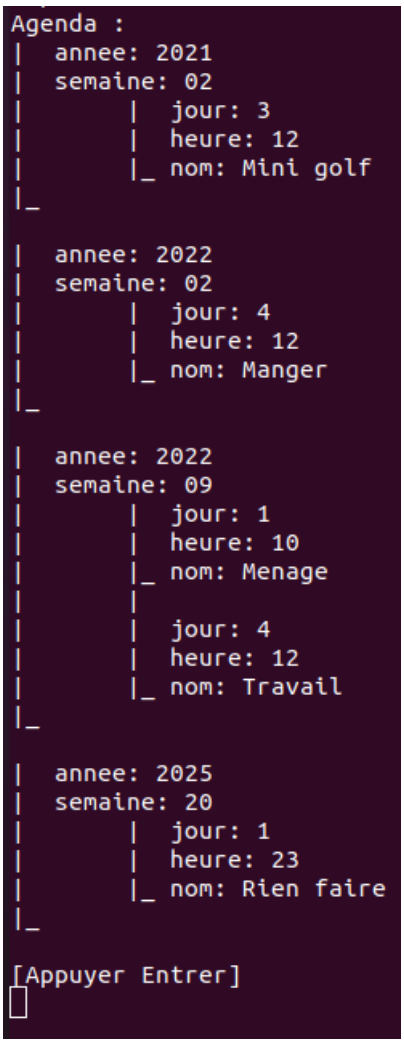
Soit l'agenda suivant :



On tente de supprimer l'élément dont l'année est 2022, le numéro de semaine 09, le jour 1 et l'heure 12.



La fonction modifie l'agenda et, à l'affichage, il devient :



L'action "Manger" correspondant à ces informations temporelles a bien été supprimée de l'agenda.