Projet : manipulation d'un calendrier composé d'événements
Semestre impair 2024

1 Introduction

Le but de ce projet est de manipuler un calendrier, composé d'événements en mettant en œuvre les compétences travaillées en CTD et TP. Il s'agit d'un modèle simplifié qui a pour seul objectif de vous faire progresser dans votre apprentissage du langage C.

1.1 Intégrité académique

Le travail doit être réalisé individuellement. Vous pouvez discuter du sujet avec vos camarades mais il est strictement interdit de copier-coller du code. L'équipe pédagogique utilise des outils de mesure de similarité de codes sources. Tout plagiat sera sévèrement sanctionné.

1.2 Évaluation

Le travail que vous allez produire sera évalué :

- à partir d'un dépôt sur Moodle que vous devrez effectuer au plus tard le jeudi 5 décembre 2024 à 23h59 (n'attendez pas le dernier moment...) ; quelle que soit la raison, si le dépôt est vide le jour de la remise à minuit alors vous aurez 0 ; il est donc conseillé de déposer votre projet avant, même s'il n'est pas fini (la version déposée doit toutefois pouvoir être compilée sans erreur pour être considérée comme valide) ;
- à partir d'un oral qui se déroulera :
  - le lundi 9/12 à 13h30 pour les groupes A11, A12, A21, A22, A31 et A32 ;
  - le lundi 9/12 à 10h00 pour les groupes B11, B12.

1.3 Barème

Le projet comporte trois niveaux de difficulté croissante, numérotés de 1 à 3. La réalisation complète et exacte de chaque niveau donne la possibilité d'obtenir une certaine note maximale, comme cela est indiqué dans le tableau suivant :

Niveau	Note maximale
1	8
2	16
3	20

Pour commencer un niveau, il faut avoir complètement réalisé le niveau précédent. Par exemple, si vous n'avez pas terminé le niveau 2 et si vous réalisez une partie du niveau 3, vous serez tout de même noté sur 16. Un niveau est considéré comme réalisé quand toutes ses fonctionnalités sont programmées et que le programme correspondant au niveau s'exécute sans erreur, avec une gestion correcte des allocations/désallocations mémoire.
Attention : si le projet que vous avez déposé sur Moodle ne peut pas être compilé sans erreur en utilisant la commande `make`, vous aurez automatiquement la note 0.

1.4 Dépôt sur Moodle

Vous déposerez sur Moodle un seul fichier archive au format ZIP qui doit être produit en suivant les directives suivantes.

1. Créez un répertoire dont le nom doit suivre le modèle :
   NOM-PRENOM-NUMETU
   où :
   - NOM, PRENOM et NUMETU doivent être remplacés, respectivement et dans cet ordre, par votre nom de famille en majuscules, votre prénom en majuscules et votre numéro d'étudiant (8 chiffres) ;
   - n'apparaît aucune lettre accentuée ;
   - n'apparaît aucun espace ni apostrophe, les éventuels espaces et apostrophes dans votre nom ou votre prénom devant être remplacés par des tirets `-` (signe moins).
2. Placez dans ce répertoire :
   - les fichiers fournis dans le sujet décrivant les interfaces de chaque module, c'est-à-dire les fichiers d'extensions `.h` (ceux-ci ne doivent en aucun cas être modifiés !) ;
   - les fichiers sources des modules de votre projet, c'est-à-dire les fichiers d'extensions `.c` ;
   - les fichiers contenant la fonction principale (`main`) de chaque niveau :
     - `testdate.c` et `testevenement.c` pour le niveau 1
     - `testcalendrier.c` pour le niveau 2
     - `testtrieurcalendrier.c` pour le niveau 3 ;
   - les fichiers de description des dépendances de nom `Makefile`n, où n vaut 1.1, 1.2, 2 et 3, si vous avez réalisé tous les niveaux, permettant, grâce à la commande `make -f Makefile`n, de produire l'exécutable correspondant ;
   - un fichier texte (obtenu avec un simple éditeur de texte) de nom `niveau-`n`.txt` où vous remplacerez n par le numéro du niveau que vous avez réalisé (1, 2 ou 3) ; ce fichier pourra être vide ou contenir d'éventuelles informations destinées à expliquer les choix que vous aurez faits.

   Vos fichiers sources doivent être indentés et contenir des commentaires permettant de bien comprendre votre travail. Aucun autre fichier (objets, exécutable, données, résultats, etc.) ne doit être présent dans le répertoire.
3. Créez une archive au format ZIP de ce répertoire ; cette archive doit porter le même nom que le répertoire auquel est ajoutée l'extension `.zip` ; pour produire cette archive, dans le terminal, vous pouvez vous placer dans le répertoire parent du répertoire contenant vos fichiers sources et utiliser la commande (␣ représente un caractère espace) :
   `zip␣`NOM-PRENOM-NUMETU`.zip␣`NOM-PRENOM-NUMETU`/*`

Respectez bien toutes ces consignes car des tests et des vérifications automatiques seront effectués sur votre projet.

1.5 Oral

Au moment de l'oral, vous devrez être prêt à :
- faire fonctionner votre programme ;
- répondre aux questions de l'enseignant ;
- effectuer des tests ou des modifications demandées par l'enseignant.

2 Consigne commune aux trois niveaux

Pour chaque niveau, les fichiers headers (.h) correspondant aux différents modules vous sont imposés. Les déclarations des fonctions publiques (leurs noms et leurs signatures), les définitions des structures et des types manipulés ne doivent pas être modifiés. Votre travail sera notamment évalué sur la base de tests automatiques, **la modification des headers rendra impossible leur compilation, et votre travail sera évalué avec la note 0**.

3 Niveau 1 : gestion de dates et définition d'un événement

3.1 Module `date`

Il s'agit d'écrire un module[1] `date`. Ce module permet de gérer une date en utilisant la structure de données **publique** suivante :

```
struct sDate {
    unsigned int annee;
    unsigned char mois;
    unsigned char jour;
    unsigned char heure;
    unsigned char minute;
};
```

Vous noterez que l'utilisation d'une structure de donnée publique n'est pas conforme aux bonnes pratiques présentées en cours. C'est un choix volontaire pour nous permettre d'évaluer votre capacité à faire la différence entre une définition publique (`sDate`) et une définition privée (`sEvenement` et `sCalendrier`).

Le module `date` devra comporter les fonctions suivantes :

- `int EstValide(const struct sDate)` :
  retourne 1 si la date passée en paramètre est valide, 0 si elle est invalide. Vous prendrez en compte la durée en jours des différents mois ainsi que des éventuelles années bissextiles (on rappelle que les années sont bissextiles si elles sont multiples de quatre, mais pas si elles sont multiples de cent, à l'exception des années multiples de quatre cents qui, elles, sont également bissextiles).
- `int Compare(const struct sDate reference, const struct sDate autreDate)` :
  compare deux dates par ordre chronologique, en retournant 0 si les deux dates sont égales, une valeur négative si la première est antérieure à la seconde, une valeur positive sinon.
- `int Appartient(const struct sDate date, const struct sDate debut, const struct sDate fin)` :
  vérifie si une date est comprise entre une date de début et une date de fin, et retourne 1 si la date est comprise entre les deux autres, 0 sinon. On considère que les bornes sont comprises dans la période.
- `int Chevauche(const struct sDate debut1, const struct sDate fin1, const struct sDate debut2, const struct sDate fin2)` :
  vérifie si deux intervalles (définis par leurs dates de début et de fin) se chevauchent, en retournant 1 si les intervalles se chevauchent, 0 sinon. On considère que les dates de début et de fin d'une période sont contenues dans cette période, ainsi un événement se terminant le 31 décembre à 23h59 est considéré comme chevauchant un événement débutant exactement à la même heure.

1. Pour chaque module que vous devez écrire, vous devez, sauf demande explicite du sujet, appliquer les principes vus en CTD concernant la programmation modulaire, notamment l'encapsulation des données et des traitements.

- `void YYYYMMDDTHHMM(char* destination, const struct sDate date)` :
  délivre en sortie, à l'adresse passée en paramètre par le pointeur `destination`, une chaîne de 13+1 caractères (avec le marqueur de fin de chaîne) correspondant à la représentation textuelle d'une date au format [2] `YYYYMMDDTHHMM` :
  Pour une date correspondant au 31 décembre 2024 à 23h59, si l'on passe à la fonction un pointeur vers une zone préalablement allouée de 14 caractères, la fonction écrit les 14 caractères suivant : `20241231T2359\0`

  On notera que l'allocation et la libération de la chaîne de caractère passée en paramètre à la fonction `YYYYMMDDTHHMM` est à la charge de l'appelant.

Remarque : dans tous les modules, il est possible d'écrire des fonctions « auxiliaires » qui ne seront pas publiques ; elles devront donc être de classe statique.

3.2 Test du module `date`

3.2.1 Compilation séparée

Créez le fichier source `testdate.c` contenant la fonction `main` afin de tester les différentes fonctions du module `date`.

Effectuez la compilation séparée grâce à l'utilitaire `make`. Pour cela, écrivez un fichier de description des dépendances `Makefile1.1` et lancez les compilations en tapant :

```
make -f Makefile1.1
```

L'exécutable ainsi produit doit être nommé `testdate`.

3.3 Module `evenement`

Il s'agit d'écrire un module `evenement`, faisant appel au module `date`, qui permet de gérer des événements caractérisés par un titre, une date de début et une date de fin. Ce module utilise la structure de données **privée** suivante :

```
struct sEvenement {
    char* titre;
    struct sDate debut;
    struct sDate fin;
};
```

et le type **public** suivant :

```
typedef struct sEvenement* tEvenement;
```

Exemple : la représentation mémoire d'un événement dont le titre serait `"test"`, la date de début est le 31 décembre 2024 à 20h00 et la date de fin le 31 décembre 2024 à 23h59 devra être celle qui est présentée sur la figure 1.

Le module `date` devra comporter les fonctions suivantes :

- `tEvenement CreerEvenement(const char* titre, struct sDate debut, struct sDate fin)` :
  Retourne un événement alloué dynamiquement à partir d'une chaîne de caractère allouée par l'appelant et deux dates. En cas d'erreur d'allocation, si les dates ne sont pas valides, ou bien si la date de fin est antérieure à la date de début, la fonction retourne `NULL`.

  Remarque : le titre passé en paramètre doit être recopié (par exemple en utilisant `strcpy`) dans une nouvelle zone allouée dynamiquement par cette fonction.

2. inspiré de l'ISO 8601

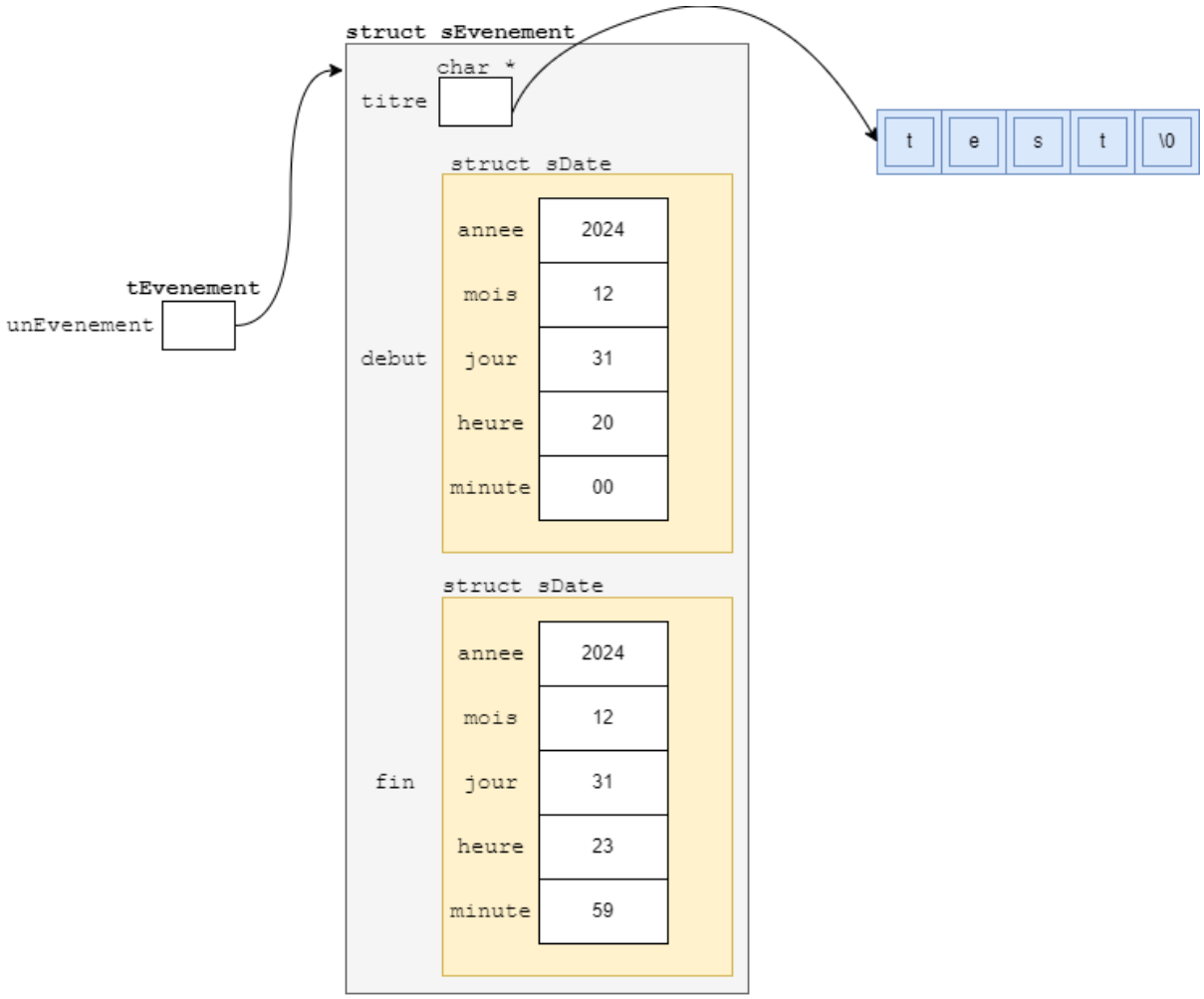

FIGURE 1 – *Représentation mémoire d'un événement "test" le 31 décembre 2024 commençant à 20h et se terminant à 23h59.*

- `char* Titre(tEvenement evenement)` :
  retourne l'adresse du premier caractère du titre de l'événement passé en paramètre (qui sera supposé non `NULL`).
- `struct sDate Debut(tEvenement evenement)` :
  retourne la date de début d'un événement passé en paramètre (supposé non `NULL`).
- `struct sDate Fin(tEvenement evenement)` :
  retourne la date de fin d'un événement passé en paramètre (supposé non `NULL`).
- `void DetruitEvenement(tEvenement* pEvenement)` :
  libère la mémoire allouée pour un événement (y compris son titre) et force la valeur de l'événement dont l'adresse est passée en paramètre à `NULL`.
- `void AfficheEvenement(tEvenement evenement)` :
  affiche à l'écran les informations relatives à un événement. Cette fonction affiche une seule ligne, terminée par un retour à la ligne, composée du texte `"TITRE du YYYYMMDDTHHMM au YYYYMMDDTHHMM"`.

  Exemple : l'événement de titre `"test"`, avec pour date de début le 1/10/2024 à 12h, et une date de fin le 1/10/2024 à 15h sera affiché en produisant exactement la ligne :
  `test du 20241001T1200 au 20241001T1500`

3.4 Test du module `evenement`

3.4.1 Compilation séparée

Créez le fichier source `testevenement.c` contenant la fonction `main` afin de tester les différentes fonctions du module `evenement`.

Effectuez la compilation séparée grâce à l'utilitaire `make`. Pour cela, écrivez un fichier de description des dépendances `Makefile1.2` et lancez les compilations en tapant :

```
make -f Makefile1.2
```

L'exécutable ainsi produit doit être nommé `testevenement`.

3.4.2 Aide à la mise au point

Afin de vérifier que vos programmes n'ont pas de « fuites » mémoire, c'est-à-dire gèrent correctement l'allocation dynamique et la libération des zones en mémoire, un outil d'analyse dynamique peut vous aider. Sous Linux, vous pouvez utiliser Valgrind [3]. Par exemple, avec l'exécutable `testevenement`, lancez l'analyse avec Valgrind en tapant [4] :

```
valgrind ./testevenement
```

et lisez le résultat pour savoir s'il a détecté des fuites (« *leaks* »).

Sous MacOS, tapez la commande :

```
leaks --atExit -- ./testevenement
```

et vérifiez le nombre de « *leaks* » à la fin de l'affichage produit.

Il existe également des outils (*code sanitizers* [5]) qui détectent, au moment de l'exécution du programme, les erreurs d'accès à la mémoire comme le débordement d'un tableau. Sur les machines des salles de TP, quand vous invoquez `gcc`, vous pouvez ajouter l'option `-fsanitize=address` qui permettra l'affichage d'informations en cas d'erreur d'accès à la mémoire lors de l'exécution d'un programme.

3. https://fr.wikipedia.org/wiki/Valgrind
4. Si des paramètres devaient être passés à l'exécutable depuis la ligne de commande, il suffirait de les placer normalement après le nom de l'exécutable.
5. https://en.wikipedia.org/wiki/Code_sanitizer

4 Niveau 2 : calendrier

Un calendrier est une liste chaînée d'événements, stockés dans leur ordre d'insertion.

4.1 Module `calendrier`

Il s'agit d'écrire un module `calendrier`, faisant appel au module `evenement`, qui permet de gérer une liste d'événements en utilisant une liste chaînée dynamique. La structure de données **privée** est la suivante :

```
struct sCalendrier {
    tEvenement pEvenement;
    tCalendrier pSuivant;
};
```

et le type **public** est le suivant :

```
typedef struct sCalendrier* tCalendrier;
```

Un calendrier vide est représenté par un pointeur `NULL`.
Exemple : la représentation mémoire d'un calendrier constitué de deux événements, s'ils sont ajoutés dans le calendrier dans l'ordre `noel`, puis le `reveillon`, devra être celle qui est présentée sur la figure 2 (le contenu des `struct sDate` a été simplifié sur le schéma, ainsi que la représentation des chaînes de caractères).

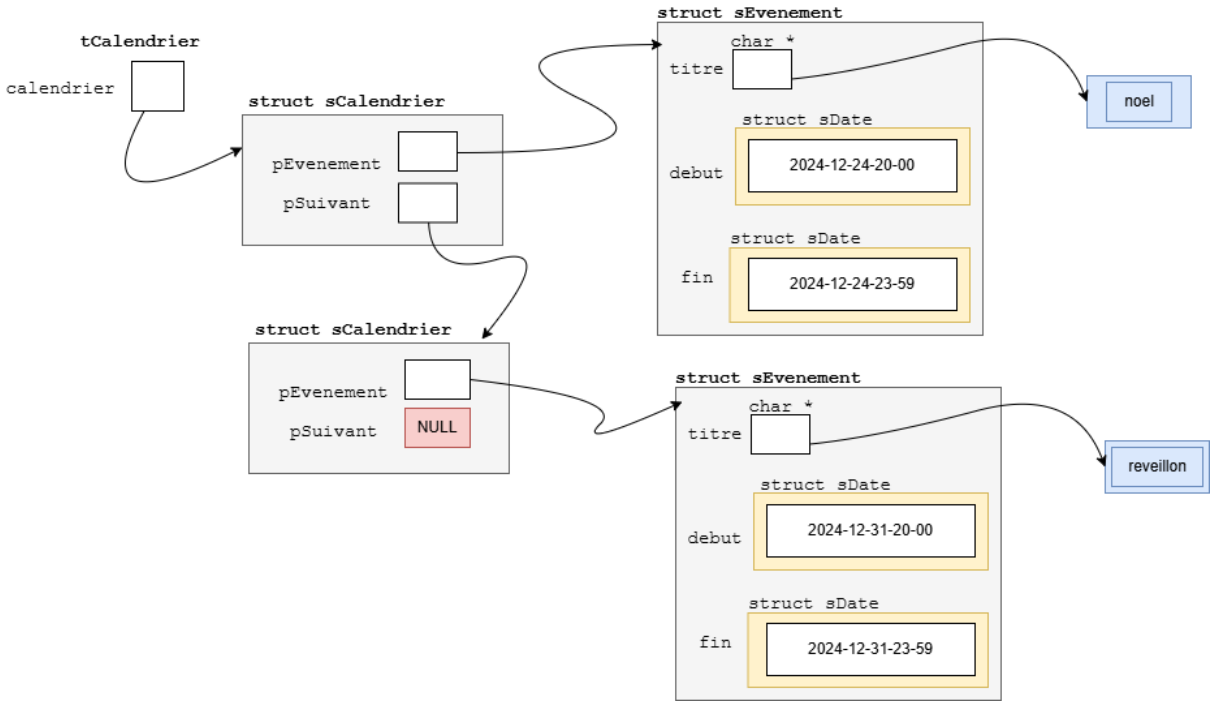

FIGURE 2 – *Représentation mémoire d'un calendrier de deux événements.*

Le module `calendrier` devra comporter les fonctions suivantes :

- `int AjouterEvenement(tCalendrier* pCalendrier, tEvenement evenement)` :
  ajoute un événement à un calendrier s'il ne contient pas déjà un événement portant le même titre

et si les dates de début et de fin du nouvel événement ne chevauchent pas un autre événement du calendrier. La fonction doit retourner 0 si l'ajout a été effectué, -1 sinon.
Remarque : dans le cas de l'ajout à un calendrier vide, le pointeur qui représente le calendrier est modifié.

- `tEvenement PremierEvenement(tCalendrier calendrier)` :
  retourne le premier événement d'un calendrier, celui-ci étant supposé ne pas être vide.
- `tCalendrier SuiteDuCalendrier(tCalendrier calendrier)` :
  étant donné un calendrier non vide, retourne l'adresse de la deuxième cellule de la liste chaînée qui constitue un calendrier.
- `void DetruitCalendrier(tCalendrier *calendrier)` :
  libère les ressources affectées à un calendrier et force le pointeur correspondant à `NULL`.
- `void AfficheCalendrier(tCalendrier calendrier)` :
  affiche un calendrier sur la sortie standard :
  - si le calendrier est vide, l'affichage doit être exactement :
    `Calendrier vide`
  - sinon, l'affichage est constitué d'une ligne pour chaque événement, dans l'ordre d'insertion.
- `int SupprimeEvenementsPeriode(tCalendrier* pCalendrier, struct sDate debut, struct sDate fin)` :
  supprime et libère tous les événements d'un calendrier qui chevauchent (au sens du module `date`) une période définie par un date de début et une date de fin. Dans le cas où tous les événements ont été supprimés, le calendrier doit être modifié pour correspondre à un calendrier vide. La fonction retourne le nombre d'événements ayant été supprimés.
- `int ExportCalendrier(tCalendrier calendrier, const char* fichier)` :
  écrit dans le fichier de texte de désignation `fichier` le calendrier `calendrier` au format iCal[6] afin de pouvoir l'importer dans un gestionnaire de calendrier ou de le visualiser, par exemple avec `online-ics-feed-viewer`[7].La fonction retourne 0 si l'export a réussi, -1 sinon (quelle que soit la raison).
  Voici le résultat d'un export du calendrier constitué de quatre événements le 1/10/2024 entre 8h et 11h59 :

```
BEGIN:VCALENDAR
VERSION:2.0
BEGIN:VEVENT
SUMMARY:Test1
DTSTART:20241001T080000
DTEND:20241001T085900
END:VEVENT
BEGIN:VEVENT
SUMMARY:Test2
DTSTART:20241001T090000
DTEND:20241001T095900
END:VEVENT
BEGIN:VEVENT
SUMMARY:Test3
DTSTART:20241001T100000
DTEND:20241001T105900
END:VEVENT
```

6. https://fr.wikipedia.org/wiki/ICalendar
7. https://larrybolt.github.io/online-ics-feed-viewer/

```
BEGIN:VEVENT
SUMMARY:Test4
DTSTART:20241001T110000
DTEND:20241001T115900
END:VEVENT
END:VCALENDAR
```

Remarque : vous noterez que le format des dates n'est pas exactement celui proposé par la fonction `YYYYMMDDTHHMM`, une adaptation (mineure) sera nécessaire pour que l'export soit un fichier correctement formaté, sans pour autant introduire de régressions dans le module `date`.

4.2 Test du module `calendrier`

Créez le fichier source `testcalendrier.c` contenant la fonction `main` afin de tester les différentes fonctions du module `calendrier`. Effectuez la compilation séparée grâce à l'utilitaire `make`. Pour cela, écrivez un fichier de description des dépendances `Makefile2` et lancez les compilations en tapant :
`make -f Makefile2`
L'exécutable ainsi produit doit être nommé `testcalendrier`.

5 Niveau 3 : tri d'un calendrier par ordre chronologique selon les dates de début des événements

5.1 Fonction de tri

L'objectif du niveau 3 est d'implanter une unique fonction complémentaire :

- `void TriCalendrier(tCalendrier calendrier)` :
  trie par ordre croissant des dates de début les événements d'un calendrier non vide.

Contrainte :

- L'implantation de cette fonction doit être réalisée en utilisant la fonction `qsort` du module `stdlib` (voir le CTD 13 : slide 15).

Indices :

- La fonction `qsort` est capable de trier un tableau (quel que soit le type de ses éléments) sous réserve qu'on lui fournisse un pointeur vers une fonction de comparaison adéquate.
- Cette partie demande d'être capable de passer d'un `tCalendrier` à un tableau de `tEvenement` puis de réorganiser le `tCalendrier` selon le tableau une fois celui-ci trié.

5.2 Test de la fonction `TriCalendrier`

Créez le fichier source `testtrieurcalendrier.c` contenant la fonction `main` afin de tester la fonction de tri. Effectuez la compilation séparée grâce à l'utilitaire `make`. Pour cela, écrivez un fichier de description des dépendances `Makefile3` et lancez les compilations en tapant :
`make -f Makefile3`
L'exécutable ainsi produit doit être nommé `testtrieurcalendrier`.

6 Pour aller plus loin : lecture d'un fichier `iCal`

La lecture d'un fichier `iCal` bien formé ne pose pas de difficultés particulières. En revanche la lecture d'un fichier malformé (par exemple avec des balises `BEGIN:VEVENT` manquantes ou surnuméraires) demande de la rigueur, et un peu d'outillage pour gérer les contextes (en introduisant par exemple

une pile de `BEGIN:VEVENT`, ou bien avec un automate à états). Bien que hors barème, n'hésitez pas à proposer vos solutions !