Table des matières

1 Analyse

Problème : Affecter des températures (une maximale et l'autre minimale) pour chacun des jours des années comprises entre 1901 et 2099, ensuite transférer les données générées dans un fichier texte dans la structure sera lisible et claire par un éditeur de texte.

Ensuite, vient l'étape de la lecture, qui consiste a extraire les données stockées dans le fichier creé et les stocker dans une structure de données compatible, pour faciliter la lecture des températures et ainsi mieux exploiter les données du fichier, avec des fonctions d'interrogations.

En ce qui concerne l'interrogation on peut voir les données lues à partir du fichier, comme une table dans une base de données ayant les colonnes suivantes :

— `jour` : Le jour.

— `mois` : Le mois.

— `an` : L'année.

— `min` : La température minimum du jour.

— `max` : La tempréture maximum du jour.

En plus de ces informations qu'on peut onbtenir directement de la structure de donnée, on peut ajouter d'autres qui sont le résultat d'un calcule effectuer sur un an, mois ou jour donnée.

— `moy(jour)` : La température moyenne du jour.

— `min(mois)` : La température minimum du mois.

— `max(mois)` : La température maximum du mois.

— `moy(mois)` : La température moyenne du mois.

— `min(an)` : La température minimum de l'année.

— `max(an)` : La température maximum de l'année.

— `moy(an)` : La température moyenne de l'année.

Après qu l'utilisateur a entré les colonnes qu'il veut afficher, il doit saisir les conditions qui doivent etre vrai pour les lignes résultats. L'exemple suivant montre le déroulement du programme de lecture des données.

```
Entrez le nom du fichier  :temps.txt
COLONNES  :
    an mois jour min max
CONDITIONS  :
    jour est mardi et an = 2001 et min < 10
```

```
ou an = 1901 et mois = 1 et jour = 1 ;
1901 1 1 1 9
2001 1 2 -7 9
2001 1 9 0 8
2001 1 16 -4 8
2001 1 23 2 8
...
```

Afin de disposer d'un maximum d'information pour l'analyse et la résolution du problème posé, il est nécessaire d'avoir des connaissances au sujet de la structure du calendrier et des saisons.

- Une année normale est composée de 365 jours, une année bissextile est composée de 366 jours,
- Une année bissextile est toujours divisible par 4.
- Une année est composée de 12 mois.
- Chaque mois est composé d'un nombre de jours variant entre 28 et 31 ; les mois 1,3,5,7,8,10,12 ont 31 jours, les mois 4,6,9,11 ont 30 jours et le mois 2 a 29 jours dans le cas d'une année bissextile et 28 jours dans le cas normal.
- Un jour sera caractérisé par deux températures (la premiere représente la valeur minimale enregistrée pendant la journée, la seconde est celle de la valeur maximale).
- Une année est composée de 4 trimestres chaque trimestre est une saison dont la longueur est approximative à 90 jours.
- Lors de la generation des temperatures et pour rester dans le domaine de la logique (ne pas avoir des temperatures avec le temps d'un mois d'une saison quelconque) (ex : température max = 45 au mois de décembre.) chaque mois aura un interval de temperatures bien défini et bien compatible), en plus de la condition (min $<$ max).
- Pour l'affectation des intervals de temperatures possibles pour chaque mois, on vas commencer par affecter a chaque mois un nombre t, ainsi le min sera dans l'interval [t-12, t-2] et le max dans l'interval [t+2, t+12]. Ainsi on aura verifier les deux conditions de validité des données générées (des temperatures compatibles avec chaque saison et logiques(min $<$ max)).

Exemple 1 : Les temperatures min et max du jour 2000/01/12. pour le mois 1 on aurais défini t = 5. ainsi : min sera dans [-7, 3] min = 2. max sera dans [7, 17] max = 15.

Exemple 2 : Les temperatures min et max du jour 1982/07/09. pour le mois 1 on aurais défini t = 24. ainsi : min sera dans [12, 22] min = 15. max sera dans [26, 36] max = 33.

Exemple 3 : Les temperatures min et max du jour 1952/04/23. pour le mois 1 on aurais défini t = 11. ainsi : min sera dans [-1, 9] min = 8. max sera dans [13, 23] max =21.

2 Analyse fonctionnelle

2.1 Structures

2.1.1 Structure "Jour"

```
typedef struct
{
  char min; // température minimum
  char max; // température maximum
} Jour;
```

2.1.2 Les énumération

L'énumération colonne indique la colonne à afficher (an, mois, jour …).

```
typedef enum
{
  AN,
  MOIS,
  JOUR,
  MIN,
  MAX,
  MOY_JOUR,
  MOY_MOIS,
  MOY_AN,
  MIN_MOIS,
  MIN_AN,
  MAX_MOIS,
  MAX_AN,
  OU // Séparateur
} Colonne;
```

L'énumération Fonction indique l'opérateur utlisé dans une condition (`=`, `<`, `>`, et `est` pour comparer le nom d'un jour).

```
typedef enum {JOUR_EST, EGALE, INF, SUP} Fonction;
```

2.1.3 Structure “Param”

Cette structure repèsente un paramètre dans une condition. Et comme un paramètre peut être une chaine de caractères, un entier ou un réel. Nous avons mis la valeur du paramètre dans une union

```
typedef union
{
  char jour[9];
  float f;
  int i;
} UParam;

typedef enum {STR, INT, FLOAT} TypeParam;

typedef struct
{
  UParam val;
  TypeParam type;
} Param;
```

2.1.4 Structure “Condition”

Enfin nous arrivons à la structure condition qui reporèsente une condition saisie par l'utilisateur.

```
/** structure condition */
typedef struct
{
  Colonne col; // Colonne sur laquelle appliquer la condition
  Fonction f; // la fonction applique
  Param param; // les paramètres de la fonction
} Condition;
```

2.2 Fonctions

2.2.1 Fonction “longueur_mois”

A – Spécification des données :
la fonction reçois comme arguments deux entiers, le premier représente

l'année et le deuxième représente le mois.

B – Spécification fonctionnelle :
une fois l'année et le mois sont saisis, la fonction "longueur_mois" effectue un test sur le nombre correspondant au numero du mois, pour les mois appartenant a l'interval $[1, 12]\backslash\{2\}$ leurs nombre de jours est affecté directement, suivant les valeurs (par rang du mois) {31, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}. Pour le mois numero 2, on procède a un deuxième test, celui de l'année, dans le cas ou elle est bissextile `an % 4 == 0` on affecte au moi de février 29 jours, dans le cas contraire 28 jours.

2.2.2 Fonction "ecrire_jour"

A – Spécification des données :
la fonction reçois comme arguments un fichier dans lequel seront remplies les données de température, et deux entiers le premier représente le jour et le deuxième représente le mois.

B – Spécification fonctionnelle :
une fois le nom du fichier cible, le jour et le mois sont saisis, la fonction "ecrire_jour" affecte a chaque mois un entier temp, dont la valeur sera (suivant le rang du mois-1) dans l'interval 5, 7, 10, 11, 15, 17, 24, 22, 16, 15, 10, 7. Ainsi pour chaque mois les valeurs min et max seront générées, inscrites et successivement comprises dans les intervals `[temp-12, temp-2]` et `[temp+2, temp+12]`, ceci avec les deux operations :

- `min = rand() % 10 + temp[mois - 1] - 12`
- `max = rand() \% -10 + temp[mois - 1] + 2.`

2.2.3 Fonction "lire _jour"

A – Spécification des données :
la fonction reçois comme arguments un fichier du quel sera faite la lecture des données de température, un tableau de type jours de taille 31 et deux entiers le premier représente le mois et le deuxième représente l'année.

B – Spécification fonctionnelle :
une fois le nom du fichier cible,le tableau des jours, le mois et l'année sont saisis, la fonction "lire_jour" procède à l'extraction des données correspondants aux temperatures des jours d'un mois défini, pour les mois $[1, 12]\backslash\{2\}$ le nombre de jours considérés est successivement (par rang du mois) 31, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, dans le cas du

mois de février le nombre de temperatures extraites sera 2*29 dans le cas d'une année bissextiles et 2*28 si non.

2.2.4 Fonction "ecrire_mois"

A – Spécification des données :
la fonction reçois comme arguments un fichier dans lequel seront remplies les données de température, et deux entiers le premier représente l'année et le deuxième représente le mois.

B – Spécification fonctionnelle :
après avoir reçu le nom du fichier d'enregistrement, l'année et le mois, la fonction "ecrire_mois" procède a la generation et l'enregistrement des temperatures de l'année cible en faisant appel a la fonction "ecrire_jour" pour chaque mois de l'année citée.

2.2.5 Fonction "lire_mois"

A – Spécification des données :
la fonction reçois comme arguments un fichier du quel sera faite la lecture des données de température, un tableau de type jour de 2 dimensions et l'année.

B – Spécification fonctionnelle :
après avoir reçu le nom du fichier de lecture, le tableau des jours et mois, et l'année cible, la fonction "lire_mois" procède au remplissage du tableau en faisant appel à la fonction "lire_jour" pour chaque ligne lu depuis le fichier, pour l'ensemble des lignes correspondants à chaque mois, jusque'à remplir les 12 mois de l'année cible.

2.2.6 Fonction "generation_temperatures"

A – Spécification des données :
la fonction reçois comme arguments un fichier dans lequel seront remplies les données de température, et deux entiers le premier représente l'année de debut de génération des températures et le deuxième représente l'année de fin de génération des températures.

B – Spécification fonctionnelle :
après avoir reçu le nom du fichier d'enregistrement et les deux années limites, la fonction "generation_temperatures" procède a la generation et l'enregistrement des temperatures des jours correspondant a (anFin-anDebut+1) années, et qui se situent entre les années anDebut et anFin. cela en faisant appel a la fonction "ecrire_mois" pour chaque année.

2.2.7 Fonction “lire”

A – Spécification des données :
la fonction reçois comme arguments un fichier du quel sera faite la lecture des données de température et un tableau de type jour de trois dimensions.

B – Spécification fonctionnelle :
après avoir reçu le nom du fichier de lecture et le tableau de remplissage, la fonction “lire” procède au remplissage du tableau par les temperatures a partir du fichier, ceci en faisant appel a la fonction “lire_mois” (anFin - anDebut + 1) fois pour chaque année entre anDebut et anFin.

2.2.8 Fonction “colonne_a_int”

A – Spécification des données :
La fonction reçoit comme paramètres : Les données lues et une colonne de type `enum Colonne`.

B – Spécification fonctionnelle :
Cette fonction convertit La colonne de type `enum Colonne`, à la valeur entière équivalente, en cherchant cette valeur dans le tableau contenant les données du fichier.

2.2.9 Fonction “tester_condition”

A – Spécification des données :
La fonction reçoit comme paramètres : Une condition, les donées, et une date (an/mois/jour).

B – Spécification fonctionnelle :
Teste si la condition est vraie pour la date donnée.

2.2.10 Fonction “test_jour”

A – Spécification des données :
Cette fonction reçoit le tableau contenant les données lues, une date, et une liste de conditions.

B – Spécification fonctionnelle :
La fonction “test_jour” parcourt la liste des conditions et les teste l’une après l’autre sur la date donnée à l’aide de la fonction “tester_condition”

2.2.11 Fonction “parcourir”

A – Spécification des données :
Cette fonction reçoit le tableau contenant les données lues à partir du fichier, une liste des conditions et une liste des colonnes saisies par l'utilisateur

B – Spécification fonctionnelle :
La fonction “test_parcourir” parcourt tous les jours stockés dans le tableau des données et affiche ceux pour lesquels la fonction “tester_jour” retourne 1.

Lors de l'affichage la fonction parcourt la liste des colonnes et affiche seulement celles éxistantes dans la liste.

3 Dossier de programmation

3.1 Objectifs

Les deux objectifs majeurs de ce programme sont :

— Générer une base de donnée de températures, s'étallant sur une période de 199 années, en les insérant dans un fichier.

— Récupérer les données concernant les températures des jours (de l'année 1901 à 2099) depuis un fichier, et leurs appliquer des fonctions d'interrogation (Max, Min, Moyenne…) pour avoir des conclusions et des résultats plus complexes.

3.2 Type de données

— dans la première étape qui est la génération des températures, la donnée est un couple (année, année). Qui signifie l'année du début et l'année de fin de génération des températures.

— dans la deuxième étape qui est l'extraction des données depuis le fichier, dans un premier lieu on va remplir notre structure avec les températures disponibles, ainsi le type de donnée sera un couple (année, année). Or une fois la structure est remplie le type de données dans ce cas peut varier en fonction de la fonction d'interrogation utilisée ; un triplet (année, mois, jour) pour la fonction qui donne les températures min et max d'un jour quelconque. un couple (année, mois) pour la fonction qui donne la moyenne de température pour un mois quelconque. une donnée (année) pour la fonction qui détermine le mois le plus chaud (ou froid) d'une année quelconque.

3.3 Initialisation des donées

Comme il y a deux étape de gestion des données dans ce projet, il y aura automatiquement deux méthodes distinctes d'initialisation des variables :

— dans la première étape les variables min et max seront initialisées de façon aléatoire avec quelques conditions pour garantir leurs validité.

— dans la deuxième étape la variable tableau sera initialisé depuis un fichier (précédemment crée).

3.4 Fonction de base

Dans le programme nous auront besoin de fonction modulo 4 et rand, pour determiner si l'année est bissextile ou non et générer les données concernant les température de façon aléatoire.

Definition des fonctions MOD4 et rand()% en language C

```
int MOD4(int valeur)
{
  faie la division euclidienne de valeur sur 4;
  retourner le reste de cette division qui sera dans 'linterval [0, 3];
}

int rand()%valeur
{
  générer un entier quelconque;
  faire sa division euclidienne sur valeur;
  retourner le reste de la division précédente,
  qui sera un entier dans 'lintervalle [0, valeur-1];
}
```

3.5 Programmation des fonctions

```
longueur_mois,
ecrire_jour,
ecrire_mois,
generation_temperatures,
lecture,
parcourir,
tester_condition;
```

```
fonction longueur_mois(an, mois entier) :entier
{
retourner (29 si (mois==2 et MOD4(an)==0),
           28 si (mois==2 et MOD4(an)!=0),
           30+(((mois * 9) / 8) & 1) sinon);
}

fonction ecrire_jour(fichier Fichier, jour, mois entier) :procédure
{
  temp[] = {5, 7, 10, 11, 15, 17, 24, 22, 16, 15, 10, 7} :entier;
  min = rand() % 10 + temp[mois - 1] - 12 :entier;
  max = rand() % -10 + temp[mois - 1] + 2 :entier;
  écrir les valeurs de min et max dans le fichier fichier;
}

fonction ecrire_mois(fichier Fichier, an, mois entier) :procédure
{
  jour entier;
  écrir la valeur du mois dans la fichier;
  pour(jour allant de 1 à longueur_mois(an, mois)
    écrire les températures min et max dans le fichier;
}

fonction generation_temperatures
(fichier Fichier, anDebut, anFin entier) :procédure
{
  déclarer an et mois entier;
  pour(an allant de anDebut à anFin)
    écrire le nom de 'lannée dans le fichier;
    pour(mois allant de 1 a 12)
      écrire les températures des jours du mois dans le fichier;
}

fonction lecture
(fichier char, jour[anFin-anDebut+1][mois][jour] :Jour) :procédure
{
  ouvrir le fichier dont le nom est "fichier" en mode lecture;
  si le fichier n'est pas vide'
   tanquon a pas atteint la fin du fichier
   {
       lire 'lannée;
```

```
       lire les mois;
   }
  sinon
    retourner un message d'erreur;
}

fonction parcouir
(ans[ANS][12][31] :Jour, conditions ListeConditions, cols ListeChaines)
{
  pour an allant de ANDB à ANFN
    pour mois allant de 1 à 12
      pour jour allant de 1 à longueur_mois(an, mois)
        si test_jour(ans, conditions, an, mois, jour)
              pour toutes les colonnes dans cols
                 afficher la valeur de colonne pour (an, mois, jour);
              fin pour
        fin si
      fin pour
    fin pour
  fin pour
}

fonction tester_condition
(an[ANS][12][31] :Jour, cond Condition, an, mois, jour  :Entier)
{
        switch(cond.fonction)
        {
          case EGALE  :return cond.col == cond.param;
          case SUP  :return cond.col > cond.param;
          case INF  :return cond.col < cond.param;
          case EST  :return cond.col == nom_jour(cond.param);
        }
}
```

4 Code source

4.1 Programme de génération des données

4.1.1 Fichier “main.c”

```
/**
```

```
  * Programme de génération des température
  * Réalisé par  :Youssef Bouhjira et Mohamed Ayoub El Midaoui
  */
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <time.h>

/** Entrées  :
  *   an  :L'année
  *   mois  :Le mois
  * Sorties  :
  *   longeur  :Le nombre de jour dans le mois
  */
int longueur_mois(int an, int mois)
{
  assert(1 <= mois && mois <= 12);
  return mois == 2  ? an % 4 ? 28  :29  :30 + (((mois * 9) / 8) & 1);
}

/** Entrées  :
  *   fichier  :Un fichier
  *   jour  :un jour
  *   mois  :Le mois dont lequel se trouve le jour jour.
  * Description  :
  *   Génere les températures min et max d'un jour selon son mois
  */
void ecrire_jour(FILE *fichier, int jour, int mois)
{
  const int temp[] = {5, 7, 10, 11, 15, 17, 24, 22, 16, 15, 10, 7};
  // temp - 12 <= min <= temp - 2
  int min = rand() % 10 + temp[mois - 1] - 12;
  // temp + 2 <= max <= temp + 12
  int max = rand() % -10 + temp[mois - 1] + 2;
  fprintf(fichier, "\t\t%d  :%d,%d\n", jour, min, max);
}

/** Entrées  :
  *   fichier  :Un fichier
  *   annee  :L'année
  *   mois  :Le mois
```

```
  * Description  :
  *   Ecrit toutes les température du mois
  */
void ecrire_mois(FILE *fichier, int an, int mois)
{
  int jour;
  fprintf(fichier, "\t%d  :\n", mois);
  for(jour = 1; jour <= longueur_mois(an, mois); ++jour)
    ecrire_jour(fichier, jour, mois);
}

/** Entrées  :
 *    fichier  :Le fichier
 *    anDebut  :L'an de début
 *    anFin  :L'an de fin
 * Description  :
 *    Ecrit les température de tous les années entre anDebut et anFin
 */
void generation_temperatures(FILE *fichier, int anDebut, int anFin)
{
  int an, mois;
  for(an = anDebut; an <= anFin; ++an)
    {
      fprintf(fichier, "%d  :\n", an);
      for(mois = 1; mois <= 12; ++mois)
        ecrire_mois(fichier, an, mois);
    }
}

int main()
{
  // initialisation du générateur de nombres aléatoire
  srand(time(NULL));

  // Lire le nom du fichier
  char nomFichier[100];
  printf("Entrez le nom du fichier  :");
  scanf("%[^\n]s");

  // Ouverture du fichier
  FILE *fichier = fopen(nomFichier, "w+");
```

```
  if(fichier)
    {
      generation_temperatures(fichier, 1901, 2099);
      fclose(fichier);
    }
  else
    printf("Impossible d'ouvrir le fichier.\n");

  return 0;
}
```

4.2 Programme de lecture des données

4.2.1 Fichier “main.c”

```
/**
  * Lecture du fichier de températures
  * Réalisé par  :Youssef Bouhjira et Mohamed Ayoub El Mioui
  **/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "lecture.h"
#include "requete.h"

/**
 * La fonction principale
 */
int main(void)
{
  // Lire le nom du fichier
  char nomFichier[100];
  printf("Entrez le nom du fichier  :");
  scanf("%[^\n]s", nomFichier);

  // Lire le fichier
  Jour ans[ANS][12][31];
  lecture(nomFichier, ans);

  interrogation(ans);
```

```
  return 0;
}
```

4.2.2 Fichier “structs.h”

```
#ifndef STRUCTS_H
#define STRUCTS_H

/** Liste chainée de chaine de caractères */
typedef struct ListeChaines
{
  char val[100]; // Valeur
  struct ListeChaines *suiv;
} ListeChaines;

typedef enum
{
  AN,
  MOIS,
  JOUR,
  MIN,
  MAX,
  MOY_JOUR,
  MOY_MOIS,
  MOY_AN,
  MIN_MOIS,
  MIN_AN,
  MAX_MOIS,
  MAX_AN,
  OU // Séparateur
} Colonne;

typedef enum {JOUR_EST, EGALE, INF, SUP} Fonction;


typedef union
{
  char jour[9];
  float f;
  int i;
} UParam;
```

```
typedef enum {STR, INT, FLOAT} TypeParam;

typedef struct
{
  UParam val;
  TypeParam type;
} Param;

/** structure condition */
typedef struct
{
  Colonne col; // Colonne sur laquelle appliquer la condition
  Fonction f; // la fonction appliquée
  Param param; // les paramètres de la fonction
} Condition;

/** Liste chainée de conditions */
typedef struct ListeConditions
{
  Condition val;
  struct ListeConditions* suiv;
} ListeConditions;

#endif // STRUCTS_H
```

4.2.3 Fichier “lecture.h”

```
#ifndef LECTURE_H
#define LECTURE_H

#include "date.h"

/* Les structures */
typedef struct
{
  char min; // température minimum
  char max; // température maximum
} Jour;

/* Les fonctions */
```

```
/** Entrées  :
  *    fichier  :Le fichier
  *    jours  :résultat de la lecture
  *    mois
  * Description  :
  *   Lit les température min et max de tous les jours
  * d'un mois et les enregistre dans le tableau jours.
  */
void lire_jours(FILE* fichier, Jour jours[31], int mois, int an)
{
  int j; // jour courant
  for(j = 1; j <= longueur_mois(an, mois); ++j)
    fscanf(fichier, "%*d  :%d,%d\n", (int*) &jours[j - 1].min,
        (int*) &jours[j - 1].max);
}

/** Entrées  :
  *    fichier  :Le fichier
  *    mois  :résultat de la lecture
  * Description  :
  *   Les 12 mois appartir de la position actuelle dans
  * le fichier et enregistre le résultat dans mois.
  */
void lire_mois(FILE* fichier, Jour mois[12][31], int an)
{
  int m; // mois courant
  for(m = 1; m <= 12; ++m)
    {
      while(fgetc(fichier) != '\n'); // sauter la ligne du mois
      lire_jours(fichier, mois[m - 1], m, an);
    }
}

/** Entrées  :
  *    nomFichier  :Le nom du fichier
  *    ans  :résultat de la lecture
  * Description  :
  *   Lit le fichier nommé nomFichier et l'enregistre
  * son contenu dans Jour.
```

```
  */
void lecture(char *nomFichier, Jour ans[ANS][12][31])
{
  // Ouvrir le fichier
  FILE *fichier = fopen(nomFichier, "r");
  if(fichier)
    {
      while(!feof(fichier))
        {
          // Lire l'année
          int an;
          fscanf(fichier, "%d  :\n", &an);

          // Lire les mois
          lire_mois(fichier, ans[an - ANDB], an);
        }
    }
  else
    {
      printf("Impossible d'ouvrir le fichier  :");
      perror(NULL);
    }
}

#endif // LECTURE_H
```

4.2.4 Fichier “requete.h”

```
/**
  * fichier  requete.h
  * Contient les fonctions relatives à l'interrogation des données
  */

#ifndef REQUETE_H
#define REQUETE_H

#include <stdlib.h>
#include <string.h>

#include "lecture.h"
#include "date.h"
```

```c
#include "structs.h"


/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  *    mois  :Le mois
  *    jour  :Le jour
  * Sorties  :
  *   La température moyenne de la date donnée.
  */
float moy_jour(Jour ans[ANS][12][31], int an, int mois, int jour)
{
  Jour j = ans[an - ANDB][mois - 1][jour - 1];
  return ((float) (j.min + j.max)) / 2;
}


/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  *    mois  :Le mois
  * Sorties  :
  *   La température moyenne du mois.
  */
float moy_mois(Jour ans[ANS][12][31], int an, int mois)
{
  float moy = 0;
  int j, longueur = longueur_mois(an, mois);
  for(j = 0; j <= longueur; ++j) moy += moy_jour(ans, an, mois, j);
  return moy / longueur;
}

/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  *    mois  :Le mois
  * Sorties  :
  *   La température maximum du mois.
  */
int max_mois(Jour ans[ANS][12][31], int an, int mois)
```

```
{
  int max = ans[an - ANDB][mois - 1][0].max, j;
  for(j = 2; j < longueur_mois(an, mois); ++j)
    {
      int maxJour = ans[an - ANDB][mois - 1][j - 1].max;
      if(max < maxJour) max = maxJour;
    }
  return max;
}


/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  *    mois  :Le mois
  * Sorties  :
  *   La température minimum du mois
  */
int min_mois(Jour ans[ANS][12][31], int an, int mois)
{
  int min = ans[an - ANDB][mois - 1][0].min, j;
  for(j = 2; j < longueur_mois(an, mois); ++j)
    {
      int minJour = ans[an - ANDB][mois - 1][j - 1].min;
      if(min > minJour) min = minJour;
    }
  return min;
}

/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  *    mois  :Le mois
  *    jour  :Le jour
  * Sorties  :
  *   La température moyenne de la date donnée
  */
float moy_an(Jour ans[ANS][12][31], int an)
{
  float moy = 0;
  int m;
```

```
  for(m = 1; m <= 12; ++m) moy += moy_mois(ans, an, m);
  return moy / 12;
}


/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  * Sorties  :
  *   La température minimum l'année
  */
int min_an(Jour ans[ANS][12][31], int an)
{
  int min = min_mois(ans, an, 1), m;
  for(m = 2; m <= 12; ++m)
    {
      int minMois = min_mois(ans, an, m);
      if(min > minMois) min = minMois;
    }
  return min;
}


/** Entrées  :
  *    ans  :Les données
  *    an  :L'an
  *    mois  :Le mois
  *    jour  :Le jour
  * Sorties  :
  *   La température maximum de la date donnée
  */
int max_an(Jour ans[ANS][12][31], int an)
{
  int max = max_mois(ans, an, 1), m;
  for(m = 2; m <= 12; ++m)
    {
      int maxMois = max_mois(ans, an, m);
      if(max < maxMois) max = maxMois;
    }
  return max;
}
```

```
/** Entrées  :
 *    col  :Une variable de type Colonne contenant la donnée sur laquelle
 *          sera appliquée la condition
 *    ans  :Les données
 *    an  :L'année
 *    mois  :Le mois
 *    jour  :Le jour
 * Sorties  :
 *   le numéro du jour ou une température min ou max d'un
 * jour, d'un mois ou d'une année
 */

int colonne_a_int(Colonne col, Jour ans[ANS][12][31], int an,
int mois, int jour)
{
  switch (col) {
    case JOUR :return jour;
    case AN :return an;
    case MOIS :return mois;

    case MIN :return ans[an - ANDB][mois - 1][jour - 1].min;
    case MAX  :return ans[an - ANDB][mois - 1][jour - 1].max;

    case MIN_MOIS :return min_mois(ans, an, mois);
    case MIN_AN :return min_an(ans, an);

    case MAX_MOIS :return max_mois(ans, an, mois);
    case MAX_AN :return max_an(ans, an);

    default :assert(0); return 0;
    }
}

/** Entrées  :
 *    col  :Une variable de type Colonne contenant la donnée sur laquelle
 *          sera appliquée la condition
 *    ans  :Les données
 *    an  :L'année
 *    mois  :Le mois
 *    jour  :Le jour
```

```
 * Sorties  :
 *   la température moyenne d'un jour, d'un mois ou d'une année
 */

float colonne_a_float(Colonne col, Jour ans[ANS][12][31], int an,
int mois, int jour)
{
  switch (col)
    {
    case MOY_JOUR :return moy_jour(ans, an, mois, jour);
    case MOY_MOIS :return moy_mois(ans, an, mois);
    case MOY_AN :return moy_an(ans, an);
    default :// erreur
      assert(0);
      return 0;
    }
}

/** Entrées  :
 *    cond  :Une variable de type Condition
 *    ans  :Les données
 *    an  :L'année
 *    mois  :Le mois
 *    jour  :Le jour
 * Sorties  :
 *   return 1 si la condition proposée est vérifiée, 0 sinon
 */

int tester_condition(Condition cond, Jour ans[ANS][12][31], int an,
int mois, int jour, int *ok)
{
  switch (cond.col)
    {
    // FLOAT
    case MOY_AN :
    case MOY_JOUR :
    case MOY_MOIS :
      {
        float val = colonne_a_float(cond.col, ans, an, mois, jour);
        switch (cond.f)
          {
```

```
          case EGALE :return val == cond.param.val.f;
          case INF :return val < cond.param.val.f;
          case SUP :return val > cond.param.val.f;
          default :// erreur
            *ok = 0;
            return 0;
          }
        break;
      }
      // INT
    default :
      {
        int val = colonne_a_int(cond.col, ans, an, mois, jour);
        switch (cond.f)
          {
          case EGALE :return val == cond.param.val.i;
          case INF :return val < cond.param.val.i;
          case SUP :return val < cond.param.val.i;
          case JOUR_EST :
            return !strcmp(nom_jour(an, mois, val), cond.param.val.jour);
          default :
            *ok = 0;
            return 0;
          }
      }
    }
}

/** Entrées  :
 *    ans  :Les données
 *    an  :L'année
 *    mois  :Le mois
 *    jour  :Le jour
 *    conditions  :Une variable de type ListeConditions
 * Sorties  :
 *   return 1 si l'un des groupes de conditions est vérifié, 0 si non
 */

int test_jour(Jour ans[ANS][12][31], int an, int mois, int jour,
ListeConditions *conditions, int *ok)
{
```

```
  ListeConditions *lst ; // itérateur pour la liste
  int res = 1; // resultat

  for(lst = conditions; lst; lst = lst->suiv)
    {
      switch (lst->val.col) {
        case OU :
          //si les conditions précédentes sont vérifiées, on arrete
          if(res) return 1;
          //si non on passe aux conditions suivantes
          else res = 1;
          break;
        default :
          res = res && tester_condition(lst->val, ans, an, mois, jour, ok);
          if(!(*ok)) return 0;
        }
    }
  return res;
}

/** Entrées  :
 *    ans  :Les données
 *    cols  :Une variable de type ListeChaines
 *    conditions  :Une variable de type ListeConditions
 *    ok  :Une variable de type entier (pour vérifier les erreurs)
 * Déscription  :
 *    renvoyer sur l'ecran l'ensemble des données vérifiant les conditions
 * précisées
 */

void parcourir(Jour ans[ANS][12][31], ListeConditions *conditions,
ListeChaines *cols, int *ok)
{
  int an, mois, jour;

  for(an = ANDB; an <= ANFN; an++) // an
    {
      for(mois = 1; mois <= 12; ++mois) // mois
        {
          for(jour = 1; jour < longueur_mois(an, mois); ++jour) // jour
            {
```

```
if(test_jour(ans, an, mois, jour, conditions, ok)) // tester
  {
    ListeChaines *colonneCourante = cols;
    // parcourir les colonnes et écrire ceux séléctionnées
    while(colonneCourante)
      {
        if(!strcmp(colonneCourante->val, "an"))
          printf("%d ", an);

        if(!strcmp(colonneCourante->val, "mois"))
          printf("%d ", mois);

        if(!strcmp(colonneCourante->val, "jour"))
          printf("%d ", jour);

        if(!strcmp(colonneCourante->val, "min"))
          printf("%d ",
                 ans[an - ANDB][mois - 1][jour - 1].min);

        if(!strcmp(colonneCourante->val, "max"))
          printf("%d ",
                 ans[an - ANDB][mois - 1][jour - 1].max);

        if(!strcmp(colonneCourante->val, "moy(jour)"))
          printf("%f ", moy_jour(ans, an, mois, jour));

        if(!strcmp(colonneCourante->val, "moy(mois)"))
          printf("%f ", moy_mois(ans, an, mois));

        if(!strcmp(colonneCourante->val, "max(mois)"))
          printf("%d ", max_mois(ans, an, mois));

        if(!strcmp(colonneCourante->val, "min(mois)"))
          printf("%d ", min_mois(ans, an, mois));

        if(!strcmp(colonneCourante->val, "moy(an)"))
          printf("%f ", moy_an(ans, an));

        if(!strcmp(colonneCourante->val, "max(an)"))
          printf("%d ", max_an(ans, an));
```

```
                    if(!strcmp(colonneCourante->val, "min(an)"))
                      printf("%d ", min_an(ans, an));

                    colonneCourante = colonneCourante->suiv;
                  }
                printf("\n");
              }
            }
        }
    }
}

/** Entrées  :
 *    liste  :Une variable de type ListeChaines
 *    str  :Une variable de type chaine de caractéres
 * Déscription  :
 *    ajouter une donnée de type chaine de caractéres a la fin d'une liste
 * chainée
 */

void liste_chaines_ajout_fin(ListeChaines **liste, char *str)
{
  ListeChaines *nouv = calloc(1, sizeof(ListeChaines));
  strcpy(nouv->val, str);
  if(!(*liste)) *liste = nouv;
  else
    {
      ListeChaines *courant = *liste;
      while(courant->suiv) courant = courant->suiv;
      courant->suiv = nouv;
    }
}

/** Entrées  :
 *    liste  :Une variable de type ListeChaines
 *    str  :Une variable de type chaine de caractéres
 * Déscription  :
 *    ajouter une donnée de type Condition a la fin d'une liste
 * chainée de type ListeConditions
 */
```

```
void liste_conditions_ajout_fin(ListeConditions **liste, Condition cond)
{
  ListeConditions *nouv = calloc(1, sizeof(ListeConditions));
  nouv->val = cond;
  if(!(*liste)) *liste = nouv;
  else
    {
      ListeConditions *courant = *liste;
      while(courant->suiv) courant = courant->suiv;
      courant->suiv = nouv;
    }
}

/** Sorties  :
 *    liste  :Une variable de type ListeChaines
 * Déscription  :
 *    ajouter une donnée de type Condition a la fin d'une liste
 * chainée de type ListeConditions
 */

ListeChaines *lire_colonnes()
{
  while(getchar() != '\n');

  // Colonnes
  printf("COLONNES  :\n    ");
  char colonne[100];

  ListeChaines *listeColonnes = NULL;
  char c = 'A';
  int i = 0;
  while(c != '\n')
    {
      c = getchar();
      /*si le caractére lu n'est ni espace ni la fin de la chaine, le mettre
        dans la liste*/
      if(c != ' ' && c != '\n') colonne[i++] = c;
      else//sinon
        {
          colonne[i] = '\0';//marquer la fin de la chaine
          i = 0;
```

```
          //l'ajouter a la fin de la liste
          liste_chaines_ajout_fin(&listeColonnes, colonne);
        }
    }
  return listeColonnes;
}

/**
 * Entrées  :
 *    chaine  :Une chaine de caractéres de type chaine de longueur 100
 *    ok  :Une variable de type entier
 * Sorties  :
 *    Colonne  :Une variable de type Colonne
 * Déscription  :
 *    renvoie le contenu d'une chaine de caractéres
 */

Colonne chaine_a_colonne(char chaine[100], int *ok)
{
  if(!strcmp(chaine, "an")) return AN;
  if(!strcmp(chaine, "mois")) return MOIS;

  if(!strcmp(chaine, "jour")) return JOUR;
  if(!strcmp(chaine, "min")) return MIN;
  if(!strcmp(chaine, "max")) return MAX;
  if(!strcmp(chaine, "moy(jour)")) return MOY_JOUR;

  if(!strcmp(chaine, "moy(mois)")) return MOY_MOIS;
  if(!strcmp(chaine, "min(mois)")) return MIN_MOIS;
  if(!strcmp(chaine, "max(mois)")) return MAX_MOIS;

  if(!strcmp(chaine, "moy(an)")) return MOY_AN;
  if(!strcmp(chaine, "max(an)")) return MAX_AN;
  if(!strcmp(chaine, "min(an)")) return MIN_AN;

  // La chaine entrée est invalide
  *ok = 0;
  return 0;
}

/**
```

```
 * Entrées  :
 *    chaine  :Une chaine de caractéres de type chaine de longueur 4
 *    ok  :Une variable de type entier
 * Sorties  :
 *    Fonction  :Une variable de type Fonction
 * Déscription  :
 *    renvoie le contenu d'une chaine de caractéres
 */

Fonction chaine_a_fonction(char chaine[4], int *ok)
{
  if(!strcmp(chaine, "est")) return JOUR_EST;
  if(!strcmp(chaine, "=")) return EGALE;
  if(!strcmp(chaine, "<")) return INF;
  if(!strcmp(chaine, ">")) return SUP;

  // La chaine entrée est invalide
  *ok = 0;
  return 0;
}

/**
 * Entrées  :
 *    chaine  :Une chaine de caractéres de type chaine de longueur 9
 *    cond  :Une variable de type Condition
 * Déscription  :
 *    remplie la variable cond avec le contenu de chaine, selon le type
 * du parametre
 */

void chaine_a_param(char chaine[9], Condition *cond)
{
  switch (cond->col)
    {
    // jour -> STR pour est , et INT pour EGALE
    case JOUR ://dans le cas de JOUR
      switch (cond->f)
        {
        case JOUR_EST ://si la fonction est JOUR_EST
          strcpy(cond->param.val.jour, chaine);//affecter le jour
          cond->param.type = STR;//affecter le type de condition
```

```
          break;
        default  :
          cond->param.type = INT;
          cond->param.val.i = atoi(chaine);
        }
      break;

      // FLOAT
    case MOY_AN :
    case MOY_JOUR :
    case MOY_MOIS :
      cond->param.val.f = atof(chaine);
      cond->param.type = FLOAT;
      break;

      // INT
    default :
      cond->param.type = INT;
      cond->param.val.i = atoi(chaine);
    }
}

/**
 * Entrées  :
 *    ok  :Une variable de entier (pour vérifier les erreurs)
 * Sorties  :
 *    Une variable de type ListeConditions
 * Déscription  :
 *    retourne une variable de type ListeConditions, contenant des données
 * (conditions) imposées par l'utilisateur
 */

ListeConditions *lire_conditions(int *ok)
{
  printf("CONDITIONS  :\n    ");
  char mot[10];

  char c = 'a';
  int pos = 0;
  Condition courant = {0} ; // Condition courante;
  ListeConditions* conditions = NULL;
```

```
int posCondition = 0; // position dans la structure condition

while(c != ';')
  {
    c = getchar();

    // ajouter c au mot
    if(c != ' ' && c != '\n' && c != ';') mot[pos++] = c;
    else // fin du mot
      {
        mot[pos] = '\0';
        pos = 0;

        // Ajout a la condition courante
        if(posCondition == 0) // colonne
          {
            courant.col = chaine_a_colonne(mot, ok);
            posCondition++;
          }
        else if(posCondition == 1) // fonction
          {
            courant.f = chaine_a_fonction(mot, ok);
            posCondition++;
          }
        else
          {
            // fin de la condition
            if(strcmp(mot, "et") == 0 || c == '\n')
              {
                if(c == '\n')
                  chaine_a_param(mot, &courant);

                liste_conditions_ajout_fin(&conditions, courant);
                posCondition = 0; // nouvelle condition

                if(c == '\n') // Ou
                  {
                    printf("ou ");
                    Condition separateur;
                    separateur.col = OU;
                    liste_conditions_ajout_fin(&conditions, separateur);
```

```
                    }
                }
              else if(c == ';')
                {
                  liste_conditions_ajout_fin(&conditions, courant);
                }
              // Lecture des paramètres
              else
                chaine_a_param(mot, &courant);
            }
        }
    }
  return conditions;
}

/**
 * Entrées  :
 *    ans  :Les données
 * Déscription  :
 *    Définir les colonnes et les conditions et faire le parcour des
 * données, en se basant sur ces conditions
 */

void interrogation(Jour ans[ANS][12][31])
{
  int ok = 1;
  ListeChaines *colonnes = lire_colonnes();
  ListeConditions* conditions =  lire_conditions(&ok);

  parcourir(ans, conditions, colonnes, &ok);
}

#endif // REQUETE_H
```

4.2.5 Fichier “date.h”

```
/**
  * fichier  :date.h
  * Contient les fonctions relatives aux dates
  */
#ifndef DATE_H
```

```
#define DATE_H

#define ANDB 1901 // L'année de début
#define ANFN 2099 // L'année de fin
#define ANS 2099 - 1901 + 1 // Le nombre d'année

/** Entrées  :
  *   an  :L'année
  *   mois  :Le mois
  * Sorties  :
  *   Le nombre de jour dans le mois
  */
int longueur_mois(int an, int mois)
{
  assert(1 <= mois && mois <= 12);
  return mois == 2  ? an % 4 ? 28  :29  :30 + (((mois * 9) / 8) & 1);
}

/** Entrées  :
  *    mois  :Un mois de l'année
  *    an  :Une année
  * Sorties  :
  *   Le nombre de jour de janvier au mois "mois".
  */
int jours_mois(short int mois, int an)
{
  int nombreJours = 0, m ;
  for(m = mois - 1 ; m >= 1 ; --m) nombreJours += longueur_mois(an, m);
  return nombreJours ;
}

/** Entrées  :
  *   an  :L'année
  *   mois  :Le mois
  *   jour :Le jour
  * Sorties  :
  *   Le nom de la semaine correspondant à la datte donnée.
  */
char* nom_jour(int an, int mois, int jour)
{
  int jours_an = (an - ANDB) * 365 + (an - ANDB) / 4 ;
```

```
  int indice =  (jours_an + jours_mois(mois, an) + jour - 1) % 7;

  switch(indice)
    {
    case 0 : return "mardi";
    case 1 : return "mercredi";
    case 2 : return "jeudi";
    case 3 : return "vendredi";
    case 4 : return "samedi";
    case 5 : return "dimanche";
    default :return "lundi";
    }
}

#endif // DATE_H
```