Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 0

Travaux pratiques : modalités et réglages

Environnement

Lors des séances de TP dans les salles du rez-de-chaussée du bâtiment du site du Madrillet, vous travaillerez en priorité sur les machines mises à votre disposition et sous Ubuntu. Si vous souhaitez travailler sur votre propre machine, vous le ferez à vos risques et périls et devrez vous débrouiller.

Modalités

À la fin de chaque séance, vous devez impérativement déposer dans le cours Algorithmique 1 de la plateforme UniversiTICE une archive au format `zip` du dossier contenant, à sa racine ou dans ses sous-dossiers, les fichiers sources « `.c` », les fichiers sources « `.h` » et les fichiers `makefile`, et uniquement ceux-là, fournis, créés, modifiés ou utilisés à l'occasion de la ou des séances dévolues au traitement des exercices figurant sur la fiche. Vous pourrez par la suite déposer des versions améliorées.

À défaut de pouvoir utiliser un fichier `makefile` adéquat, vous pouvez créer l'archive de la manière suivante : cliquez droit sur l'icône du dossier contenant les fichiers, cliquez ensuite sur « Compresser… », sélectionnez « `.zip` », cliquez sur « Créer » puis sur « Fermer ».

Les enseignant·es pourront refuser de prendre en compte tout source qui n'aura pas été remis en forme par l'utilitaire de remise en forme de codes sources Uncrustify.

MÀJ 26-01 en +forme+ de codes (YG, SH)

Réglages de l'EDI Geany

Avant de lancer Geany, récupérez l'archive `algo1_src.zip` sur UniversiTICE, extrayez-en ses composants puis copiez dans votre dossier personnel `~/` le contenu du dossier `config/` [1].

Réglages partiels de l'EDI Geany en mode manuel

Les « préférences de l'utilisateur » décrites par le fichier de configuration `geany.conf` peuvent être effectuées à la main dans quatre boites de dialogue.

1) Dans la boite de dialogue « Éditer > Format > Envoyer la sélection vers > Définir les commandes personnalisées » : fixer le champ « Commande » de l'article dont le champ « ID » vaut 1 à « `bash -c "uncrustify -c $HOME/uncrustify072_c.cfg -lc -q"` » Si cet article n'existe pas déjà : « Ajouter ». Validez deux fois : à gauche de la commande puis « Valider ».

1. Le dossier `config/` est composé de cinq fichiers et trois sous-dossiers. Trois fichiers de configuration de l'EDI Geany : `geany.conf`, `keybindings.conf` et `filetypes.c`, un fichier configuration pour Uncrustify : `uncrustify072_c.cfg`, un script : `config.sh`. Les deux premiers, à trouver dans `config/.config/geany/`, sont à copier dans `~/.config/geany/`, le troisième, à trouver dans `config/.config/geany/filedefs/`, dans `~/.config/geany/filedefs/`, le quatrième, à trouver dans `config/`, dans `~/`, le cinquième, lui aussi dans `config/`, peut être copié dans `~/` ou ne pas être copié du tout. Attention : les dossiers et fichiers dont les noms commencent par un point sont cachés par défaut dans les fenêtres gestionnaire de fichiers ; si vous voulez qu'ils y soient affichés, « Affichage > Afficher les fichiers cachés » (« Affichage » sous Ubuntu : bouton avec trois traits horizontaux superposés) ou tapez ctrl+h. Pour réaliser la copie attendue très simplement : ouvrez un terminal dans le dossier `config/` et exécutez le script `config.sh` : « `./config.sh` » en ligne de commande. Si la configuration s'est correctement déroulée, vous verrez apparaitre un filet vertical rouge en colonne 80 à l'ouverture de Geany en lieu et place du filet vert, à peine visible, placé en colonne 72 et proposé par défaut.

2) Dans la boite de dialogue « Éditer > Préférences » :
Général > Divers > Divers : éteindre « Émettre un bip sur les erreurs ou lorsque la compilation est terminée » ;
Interface > Interface > Police > Éditeur : la fixer à « Liberation Mono Regular 10 » ;
Éditeur > Fonctionnalités > Fonctionnalités : fixer « Marqueur de commentaire » à la chaine vide ;
Éditeur > Indentation > Indentation : fixer « Largeur » à 2 ;
Éditeur > Indentation > Indentation : fixer « Type » à « Espaces » ;
Éditeur > Complétions > Complétions : allumer « Compléter automatiquement les symboles » ;
Éditeur > Complétions > Complétions : allumer « Compléter automatiquement tous les mots du document » ;
Éditeur > Complétions > Complétions : fixer « Caractères à taper... » à 3 ;
Éditeur > Affichage > Affichage : allumer « Afficher les guides d'indentation » ;
Éditeur > Affichage > Affichage : allumer « Afficher les espaces » ;
Éditeur > Affichage > Marqueur de longues lignes : fixer « Colonne » à 80 ;
Éditeur > Affichage > Marqueur de longues lignes : fixer « Couleur » à « rouge » (au milieu du panel proposé) ;
Fichiers > Enregistrement des fichiers : allumer « Nouvelle ligne à la fin du fichier » ;
Fichiers > Enregistrement des fichiers : allumer « Enlever les espaces et tabulations de fin » ;
Raccourcis > Raccourcis clavier > Format : cliquer sur l'action « Envoyer vers la commande personnalisée 1 », cliquer sur « Changer ». Dans la nouvelle boite de dialogue, taper ctrl+alt+a puis cliquer sur « Valider ».

3) Dans la boite de dialogue « Affichage » :
Changer le jeu de couleurs... : choisir « Alternate » ;
Afficher la barre d'outils : éteindre.

4) Dans la boite de dialogue « Construire > Définir les commandes de construction », rubrique « Commandes pour C », colonne « Commande », un source C étant en cours d'édition :
Ligne « Compiler » : fixer le champ à l'intersection à « `gcc -std=c2x -Wall -Wconversion -Werror -Wextra -Wfatal-errors -Wpedantic -Wwrite-strings -O2 "%f"` » ;
Ligne « Construire » : même chose, mais en ajoutant à la commande précédente les options « `-o "%e"` » entre « `-O2` » et « `"%f"` ».

Suite des réglages

Il est possible d'éditer sous l'EDI Geany un fichier source C ou `makefile` en double-cliquant sur son icône. Il faut pour cela modifier l'action par défaut qui est exécutée par un double clic.

Cliquez droit sur le premier fichier d'extension `.c` venu. Dans « Propriétés > Général > Ouvrir avec », signifiez que vous l'ouvrirez dorénavant — lui comme tous ceux de même extension — avec Geany. Vous ferez par la suite de même avec le premier fichier d'extension `.h` et le premier fichier `makefile` que vous rencontrerez.

Raccourcis de l'EDI Geany

Voici ce que permettent certaines touches ou combinaisons de touches :

— f8 : vérifier la syntaxe du fichier d'extension `.c` ou `.h` en cours d'édition avec les options de compilation standards. En cas de succès pour un fichier `.c`, produire le fichier objet associé d'extension `.o`. En cas d'échec ou de succès pour un fichier `.h`, produire le fichier en-tête précompilé (*precompiled header file*) associé d'extension `.gch`[2] ;

2. Il vous appartient de supprimer vous-même les éventuels fichiers `.gch` qui figurent dans votre dossier avant que de créer l'archive à déposer.

— f9 : construire l'exécutable associé au fichier d'extension `.c` en cours d'exécution avec les options de compilation standards. Ne peut être utilisée que pour des fichiers complets, hors compilation séparée ;

— maj+f9 : construire l'exécutable décrit dans le fichier `makefile` qui figure dans le dossier du fichier d'extension `.c` ou `.h` ou encore du fichier `makefile` en cours d'édition. Le même résultat peut être obtenu en ligne de commande par « `make` » ou « `make all` » ;

— maj+ctrl+f9 – « clean » : même résultat que « `make clean` » ;

— ctrl+alt+a : remettre en forme par Uncrustify [3] de la partie du source C sélectionnée.

— ctrl+a – ctrl+alt+a : sélectionner tout le texte en cours d'édition (première combinaison) puis le remettre en forme par Uncrustify (deuxième). À n'utiliser que sur des fichiers sources C.

3. La remise en forme est intéressante mais n'est pas parfaite. De nouvelles versions du fichier de configuration `uncrustify072_c.cfg` pourront être fournies dans le courant du semestre.

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 1

Programmes sur chaines

Objectifs : introduction de quelques fonctions sur les chaines.

Prérequis : cours *Bases de la programmation impérative* ; capacité à se rendre à la B.U. pour consulter, par exemple, le KR, ou *to read C11 in the original*, ou encore *to use the Linux man command*.

Travail minimum : exercices 1 à 3.

Exercice 1

Écrivez en C, dans un fichier source de nom `str_divide.c`, un programme qui, à l'aide d'une boucle `while`, de la fonction `scanf` et d'une chaine de format utilisant le caractère `s` [4], lit les chaines de caractères d'une longueur maximale à fixer [5], ne comprenant aucun caractère d'espacement qui figurent sur l'entrée standard et les écrit les unes après les autres sur la sortie standard, précédées de leur numéro dans l'ordre de lecture.

Les contraintes suivantes doivent être respectées :

— la numérotation commencera à 1 (un) ;
— sur chaque ligne de texte produite ne figureront qu'une chaine et son numéro ;
— le numéro sera séparé de la chaine par une tabulation (`'\t'`).

Par exemple, si la longueur maximale est fixée à 8 et que l'entrée est :

```
The book assumes some familiarity with basic programming concepts like
 variables, assignment statements, loops, and functions.
```

la sortie sera :

```
1	The
2	book
3	assumes
4	some
5	familiar
6	ity
7	with
8	basic
9	programm
10	ing
11	concepts
12	like
13	variable
14	s,
15	assignme
16	nt
17	statemen
```

4. Voir Annexe B1.3 « Les entrées mises en forme » du KR par exemple.

5. Pour l'instant — et sauf à avoir déjà fait un tour en *Algorithmique 2* ou à avoir développé une fonction idoine —, vous devrez vous résigner à utiliser un nombre magique qui apparaitra à la fois dans la spécification de la longueur du tableau de caractères destiné à la mémorisation de la dernière chaine lue et dans celle de la largeur maximum du champ lu par `scanf`.

```
18⟶ts,
19⟶loops,
20⟶and
21⟶function
22⟶s.
```

le symbole « ⟶ » signifiant une tabulation dont la largeur est ici fixée à 8.

Exercice 2

Réalisez une copie du fichier précédent ; nommez-la `str_islongint.c`. Éditez ce nouveau fichier source.

Transformez le programme de sorte que, pour chacune des chaines lues, il ajoute en fin de ligne l'information « `value =` *a* » lorsque la chaine correspond à l'écriture en base 10 d'un entier codable sur le type **`long int`**, *a* étant la valeur de cet entier. Sinon, l'information ajoutée sera « `value out of range` » (débordement) si la chaine correspond bien à l'écriture en base 10 d'un entier mais que cet entier n'est pas codable sur le type **`long int`**, et « `illegal value` » sinon.

Contraintes supplémentaires :

— une tabulation sera insérée entre la chaine et l'information ;

— en cas d'information « `value =` *a* », il faudra recourir à une chaine de format utilisant la suite de caractères `ld` pour afficher le nombre ;

— il devra être fait appel à la fonction standard `strtol`[6] et à la variable standard `errno`[7] pour à la fois convertir la chaine lue en un entier du type **`long int`** et tester si cette conversion s'est correctement déroulée.

```
#include <stdlib.h>
long int strtol(const char *nptr, char **endptr, int base);
```

```
#include <errno.h>
errno
```

Par exemple, avec une longueur maximale égale à 8, si l'entrée est :

```
The C Programming Language 2nd Edition March 22 1988
```

la sortie sera :

```
1⟶The⟶illegal value
2⟶C⟶illegal value
3⟶Programm⟶illegal value
4⟶ing⟶illegal value
5⟶Language⟶illegal value
6⟶2nd⟶illegal value
7⟶Edition⟶illegal value
8⟶March⟶illegal value
9⟶22⟶value = 22
10⟶1988⟶value = 1988
```

Attention : 1) la longueur 8 utilisée dans l'exercice précédent et dans l'exemple ne permet pas de mettre en évidence les débordements puisque la norme exige que les valeurs minimale et maximale du type **`long int`** soient au minimum égales à −2 147 483 647 et 2 147 483 647 ; 2) l'exemple, avec ses « petits » 22 et 1988, ne permet donc pas de tester les débordements. Conseil : fixez la longueur maximale à 32. Exigence : testez des débordements.

6. Voir Annexe B5 « Les fonctions utilitaires » de l'ouvrage précédemment cité.
7. Voir Annexe B4 « Les fonctions mathématiques » et Annexe B5.

Exercice 3

Réalisez une copie du fichier précédent ; nommez-la `str_operclass.c`. Éditez ce nouveau fichier source.

Transformez le programme de sorte que, pour chacune des chaines lues, il ajoute cette fois en fin de ligne l'information « `operand =` *a* » si la chaine correspond à l'écriture en base 10 de l'entier *a* et que *a* est codable sur le type **`long int`**, « `value out of range` » en cas de débordement, « `operator` » si la chaine égale l'une des chaines `"ADD"`, `"MUL"` ou `"END"`, et, sinon, « `rejected form` ».

Contraintes supplémentaires :

— en cas de débordement ou de rejet, il devra être immédiatement mis fin à l'exécution du programme avec renvoi de la valeur `EXIT_FAILURE` ;

— pour comparer les chaines, vous ferez appel à la fonction `strcmp`[8].

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

Par exemple, avec une longueur maximale égale à 32, si l'entrée est :

```
20 100 MUL 15 9 ADD ADD END
UN ET UN, DEUX.
```

la sortie sera :

```
1———⟩20———⟩operand = 20
2———⟩100———⟩operand = 100
3———⟩MUL———⟩operator
4———⟩15———⟩operand = 15
5———⟩9———⟩operand = 9
6———⟩ADD———⟩operator
7———⟩ADD———⟩operator
8———⟩END———⟩operator
9———⟩UN———⟩rejected form
```

8. Voir Annexe B3 « Les fonctions de traitement de chaines ».

Algorithmique 1 : méthodologie de la programmation impérative
Fiche de TP n° 2

Notation polonaise

Objectifs : utilisation d'un tableau.
Prérequis : enseignement *Informatique : Bases de la programmation impérative* ; fiche de TP n° 1.
Travail minimum : exercice 1.

À l'entrée « polonaise » du *Dictionnaire des Mathématiques* de Bouvier, George et Le Lionnais cité en référence dans le support de cours et d'exercices, on lit :

> *Notation polonaise.* — Permet l'écriture des formules logiques et algébriques sans parenthèses. Elle est due à Łukasiewicz. Par exemple le nombre $a(b+c)$ peut se noter $\times a+b\,c$ ou $a\,b\,c+\times$. Dans le premier cas, la notation polonaise est dite préfixée, dans le second cas, elle est dite postfixée. Certaines calculatrices électroniques de poche utilisent ce principe de notation.

Ce à quoi il faut ajouter que Łukasiewicz (1878-1956) est un logicien et philosophe polonais, d'où le qualificatif de la notation, et que la notation permet également de ne pas connaitre la priorité des opérateurs.

La notation « standard » de l'expression algébrique $a(bc+d)+e$ par exemple exige de savoir que le produit est prioritaire sur la somme : bc est prioritaire sur $c+d$; de même $a(bc+d)$ sur $(bc+d)+e$. Voici la même expression en notation polonaise préfixée : $+\times a+\times b\,c\,d\,e$, et postfixée : $a\,b\,c\times d+\times e+$.

C'est la deuxième notation, également connue sous le nom de *notation polonaise inverse* qui nous intéresse ici : il s'agit d'évaluer toutes les expressions arithmétiques données en notation polonaise inverse qui figurent sur l'entrée. L'ensemble des lexèmes du langage de départ est réduit : des entiers ; l'opérateur `ADD` pour l'addition ; l'opérateur `MUL` pour la multiplication ; l'opérateur `END` pour marquer la fin de l'expression. Le résultat du dernier exercice de la fiche précédente donc. Pour l'expression

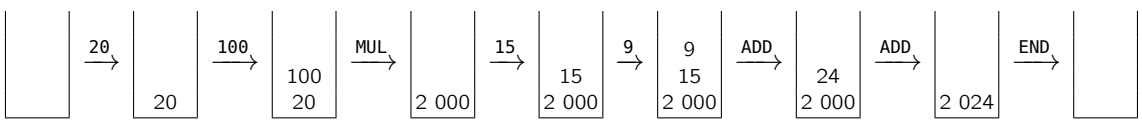
```
20 100 MUL 15 9 ADD ADD END
```

le programme à développer doit ainsi afficher

```
2024
```

Pour l'évaluation d'une expression, on utilise une *pile* d'entiers. Initialement, la pile est vide. Si un entier est lu, il est empilé (ajouté à la pile). Si un opérateur binaire est lu (`ADD` ou `MUL`), les deux derniers entiers empilés sont dépilés (retirés de la pile) et le résultat de l'opération associée appliquée à ces deux entiers est empilé. Si l'opérateur `END` est lu, le dernier entier empilé est dépilé et affiché.

Voici une illustration de l'état de la pile pour l'exemple donné plus haut :



Attention ! Il doit y avoir au moins deux entiers dans la pile lorsqu'un opérateur binaire est rencontré. Il ne doit y avoir qu'un seul entier dans la pile lorsque l'opérateur `END` est rencontré. La pile ne peut contenir qu'un nombre limité d'entiers.

Exercice 1

Réalisez une copie du fichier source `str_operclass.c`, résultat du travail sur la fiche précédente ; nommez-la `calc.c`. Éditez ce nouveau fichier source.

Transformez le programme pour qu'il assure l'évaluation de toutes les expressions arithmétiques en notation polonaise inverse lues sur l'entrée.

Vous implanterez la pile à l'aide d'un tableau d'entiers et d'un entier mémorisant la *hauteur* de la pile, c'est-à-dire le nombre d'entiers qui y sont empilés. En cas d'erreur (valeur entière lue non représentable sur le type `long int`, syntaxe incorrecte, débordement de la pile, hauteur de la pile insuffisante), un message approprié devra être affiché puis il devra être immédiatement mis fin à l'exécution du programme avec renvoi de la valeur `EXIT_FAILURE`.

Exercice 2

Si jamais les chaines `"ADD"`, `"MUL"` et `"END"` n'apparaissaient pas qu'une fois et une seule dans votre programme, utilisez des macroconstantes pour que cela soit le cas.

Exercice 3

Ajoutez d'autres opérateurs : `SUB` pour la soustraction, `QUO` pour le quotient de la division, `REM` pour son reste.

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 3

Troncatures de développements en série

Objectifs : mise en œuvre de l'exigence de documentation des programmes ; documentation détaillée.

Prérequis : logique de Hoare ; assertion d'entrée, assertion de sortie ; invariant de boucle, quantité de contrôle ; exercice 2.9 du support de cours et d'exercices.

Travail minimum : trois exercices au choix ; la preuve manuscrite complète de la correction de deux des fonctions de troncature traitées doit être rendue.

Les cinq exercices proposés dans la suite sont des prolongations à l'exercice 2.9 qui figure sur le support de cours et d'exercices avec, ici, l'exigence supplémentaire de programmation des calculs de troncature sous la forme de fonctions.

Pour chacune des fonctions mathématiques $f(x)$ et les troncatures $t(x, n)$ à un ordre dépendant d'un naturel n de leurs développements en série au voisinage de 0, il s'agit de :

— construire un algorithme correct de calcul de $t(x, n)$ qui procède selon les puissances croissantes de x. L'algorithme ne doit recourir qu'à des opérations arithmétiques et à aucune fonction (comme la factorielle ou l'élévation à une puissance par exemple). Il ne doit pas modifier ses données x et n. Il ne doit utiliser que quelques variables ;

— coder cet algorithme en C sous la forme d'une fonction à deux paramètres, `x` du type **`double`** et `n` du type **`int`**. Si la valeur de `n` est strictement négative, la fonction doit renvoyer une valeur nulle ;

— documenter ce codage de manière détaillée en fournissant sous la forme de commentaires l'assertion d'entrée de la fonction, son assertion de sortie, son invariant de boucle et sa quantité de contrôle ;

— inscrire ce codage dans un source afin de le tester selon le protocole suivant : pour toutes les valeurs flottantes x lues sur l'entrée standard, le source doit afficher sur la sortie standard les troncatures $t(x, n)$ pour n allant de 0 à 9 puis la valeur $f(x)$; l'affichage de chacune des troncatures est suivi d'une tabulation et de l'affichage de n ; les troncatures et la valeur $f(x)$ sont affichées sur des lignes séparées ; chacune des valeurs flottantes est affichée sur une largeur (minimale) de 13 caractères et avec une précision de 10 décimales. Les sources pouvant être testés de manière automatique, ce protocole doit être scrupuleusement respecté.

Conseil. Récupérez le source `algo1_src/tp/3/log1p_trunc.c` situé dans l'archive `algo1_src.zip` sur UniversiTICE. Prenez connaissance de son contenu. En particulier :

— de la déclaration de la fonction de troncature avec la spécification, située avant la fonction principale `main` ;

— de la définition de cette même fonction, sans plus la spécification mais avec l'invariant de boucle et la quantité de contrôle, situé après la fonction principale ;

— des dénominations différentes, ln et `log1p`. La première désigne la fonction mathématique « logarithme népérien ». La deuxième est une fonction standard du C :

```
#include <math.h>
double log1p(double x);
```

qui renvoie (une approximation réputée excellente) du logarithme népérien de 1 plus son argument, autrement dit de $\ln(1 + \texttt{x})$;

— de l'inclusion de l'en-tête standard `<math.h>`, précisément pour pouvoir utiliser `log1p` ;

— de l'accord du nom du source et du nom la fonction de troncature avec la dénomination C.

Attention. Pour pouvoir produire l'exécutable associé au source `log1p_trunc.c`, il est nécessaire de demander explicitement à l'éditeur de liens d'incorporer la bibliothèque mathématique. Pour cela, et pour cette séance de TP, ajoutez « `-lm` » au bout de la commande de construction de Geany (« Construire > Définir les commandes de construction » ou alt+c puis s, rubrique « Commandes pour C », champ à l'intersection de la ligne « Construire » et de la colonne « Commande »).

Exercice 1

Donnez le code d'une fonction C qui renvoie la troncature à l'ordre $2n+1$ du développement en série au voisinage de 0 de la fonction $x \mapsto \sinh(x)$ (sinus hyperbolique) :

$$\sum_{j=0}^{n} \frac{x^{2j+1}}{(2j+1)!}.$$

La fonction sinh figure dans `<math.h>` sous le nom `sinh`.

Exercice 2

Même chose pour la fonction $x \mapsto \arctan(x)$ et sa troncature à l'ordre $2n+1$:

$$\sum_{j=0}^{n} (-1)^j \frac{x^{2j+1}}{2j+1}.$$

La fonction arctan figure dans `<math.h>` sous le nom `atan`.

Exercice 3

Même chose pour la fonction $x \mapsto \cos(x)$ et sa troncature à l'ordre $2n$:

$$\sum_{j=0}^{n} (-1)^j \frac{x^{2j}}{(2j)!}.$$

La fonction cos figure dans `<math.h>` sous le nom `cos`.

Exercice 4

Même chose pour la fonction $x \mapsto \sin(x) + \cos(x)$ et sa troncature à l'ordre n :

$$\sum_{j=0}^{n} (-1)^{\lfloor j/2 \rfloor} \frac{x^j}{j!}.$$

Les fonctions sin et cos figurent dans `<math.h>` sous les noms `sin` et `cos`.

Exercice 5

Même chose pour la fonction $x \mapsto \sqrt{1+x}$ et sa troncature à l'ordre n :

$$\sum_{j=0}^{n} (-1)^j \frac{-1 \cdot 1 \cdot 3 \cdot \ldots \cdot (2j-3)}{2^j \cdot j!} x^j.$$

La fonction $\sqrt{\ }$ figure dans `<math.h>` sous le nom `sqrt`.

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 4

Objectif structures en tableaux

Objectifs : utilisation de l'agrégation structure et du chevauchement union.
Prérequis : initialisation d'un tableau par liste de valeurs ; agrégation structure et chevauchement union, mots-clés **`struct`** et **`union`**, opérateurs `.` et `->` ; syntaxe et sémantique de la fonction `main`, récupération des arguments figurant sur la ligne de commande.
Travail minimum : exercices 1 à 3.

Récupérez l'archive `data.zip` sur UniversiTICE et décompressez-la en plaçant le dossier `data/` au même niveau que tous vos dossiers de TP [9]. Pour la fiche courante, seul le texte `tintin.csv` est utilisé.

Copiez dans votre dossier dévolu à la fiche de TP courante le source situé dans le dossier `algo1_src/tp/4/`.

Vous prendrez connaissance du contenu des fichiers `tintin.csv` et `tintin_all.c` en même temps que vous lirez les explications et les attendus.

Le fichier `tintin.csv`, est au format CSV (*comma-separated values*). Il contient quelques informations sur quelques-uns des personnages des albums de Tintin. Plus précisément, chacune de ses lignes fait apparaitre trois rubriques : le nom du personnage, son ou ses prénoms, l'année de publication de l'album dans lequel il apparait pour la première fois.

Le fichier `tintin_all.c` est un source C en devenir. Tel qu'il est fourni, il ne contient que la propre spécification du source, la spécification et un début de déclaration d'un alias d'une structure anonyme, `character`, qui doit permettre de mémoriser les trois informations, la spécification et la déclaration d'une procédure d'affichage partiel des informations figurant dans un objet de (nom de) type `character`, `character_display_names`.

Il vous est demandé de commencer par compléter le source `tintin_all.c` — exercice 1 — en y incluant directement les données qui figurent dans le fichier `tintin.csv`, puis de composer d'autres sources — exercices suivants — qui iront questionner les informations mémorisées. Le chevauchement **`union`** n'est abordé que dans l'exercice final.

Exercice 1

Complétez le source `tintin_all.c` de manière à ce qu'il soit conforme à sa spécification.

Exemple partiel :

```
$ ./tintin_all et Milou ?
./tintin_all: No parameter is expected
Syntax: ./tintin_all
$ ./tintin_all
Abdallah Ben Kalish Ezab
Omar Ben Salaad
Peggy Alcazar
...
Stephan Szprinkoth
Ali Ben Mahmoud
```

9. Le dossier `data/` et son contenu sont amenés à resservir.

Vous respecterez les contraintes suivantes :

— la fonction principale `main` inclut la définition d'un tableau de `character` dont le contenu est obtenu à partir d'un copié-collé du contenu du fichier `tintin.csv` ;

— l'ordre des composants de ce tableau est celui des lignes du fichier `tintin.csv` ;

— la définition de la fonction `character_display_names` figure après celle de la fonction principale.

Exercice 2

Réalisez une copie du source `tintin_all.c`. Nommez-la `tintin_yearfirstapp.c`. Donnez comme spécification au nouveau source :

```
//  Utilisation :
//    tintin_yearfirstapp VALUE
//
//  Affiche les personnages d'albums de Tintin figurant dans une très petite
//    base de données qui ont fait leur première apparition dans un album paru
//    l'année égale à VALUE.
//
//  Renvoie EXIT_FAILURE à l'environnement d'exécution si argc est différent de
//    deux, si VALUE ne correspond pas à l'écriture d'un entier en base 10 ou si
//    cet entier n'est pas codable sur le type long int.
//  Renvoie EXIT_SUCCESS dans tous les autres cas.
```

Modifiez le source de manière à ce qu'il soit conforme à sa spécification.

Vous respecterez les contraintes suivantes :

— la fonction standard `strtol` de `<stdlib.h>` doit être utilisée ;

— les deux cas « valeur illégale » et « dépassement de capacité » doivent être traités. Les messages d'erreur associés doivent être dispensés ;

— l'en-tête `<errno.h>` doit donc être inclus et la variable `errno` utilisée.

Exemple :

```
$ ./tintin_yearfirstapp
./tintin_yearfirstapp: One and only one parameter is expected
Syntax: ./tintin_yearfirstapp VALUE
$ ./tintin_yearfirstapp 1 2 3
./tintin_yearfirstapp: One and only one parameter is expected
Syntax: ./tintin_yearfirstapp VALUE
$ ./tintin_yearfirstapp VALUE
./tintin_yearfirstapp: Illegal value
$ ./tintin_yearfirstapp 1934-1939
./tintin_yearfirstapp: Illegal value
$ ./tintin_yearfirstapp 125376564754788560970
./tintin_yearfirstapp: Value out of range
$ ./tintin_yearfirstapp 1934
Allan Thompson
Roberto Rastapopoulos
$ ./tintin_yearfirstapp "1939"
Igor Wagner
Blanca Castafiore
```

Exercice 3

Réalisez une copie du source `tintin_all.c` ou du source `tintin_yearfirstapp.c`. Nommez-la `tintin_innames.c`. Donnez comme spécification au nouveau source :

```
//  Utilisation :
//    tintin_innames VALUE
//
//  Affiche les personnages d'albums de Tintin figurant dans une très petite
//    base de données dont le nom ou le(s) prénom(s) qui contient VALUE.
//  Lorsque VALUE est la chaine vide, affiche tous les personnages figurant dans
//    la base de données.
//
//  Renvoie EXIT_FAILURE à l'environnement d'exécution si argc est différent de
//    deux.
//  Renvoie EXIT_SUCCESS sinon.
```

Modifiez le source de manière à ce qu'il soit conforme à sa spécification.

Vous respecterez les contraintes suivantes :

— la fonction standard `strstr` de `<string.h>` doit être utilisée ;

— le cas de la chaine vide ne doit pas donner lieu à un traitement particulier.

Exemple partiel :

```
$ ./tintin_innames
./tintin_innames: One and only one parameter is expected
Syntax: ./tintin_innames VALUE
$ ./tintin_innames un deux trois
./tintin_innames: One and only one parameter is expected
Syntax: ./tintin_innames VALUE
$ ./tintin_innames VALUE
$ ./tintin_innames lo
Alonzo Perez
Arturo Benedetto Giovanni Giuseppe Pietro Archangelo Alfredo Cartoffoli
Roberto Rastapopoulos
$ ./tintin_innames "ca"
Peggy Alcazar
Blanca Castafiore
$ ./tintin_innames ""
Abdallah Ben Kalish Ezab
Omar Ben Salaad
Peggy Alcazar
...
Stephan Szprinkoth
Ali Ben Mahmoud
```

Exercice 4

Réunissez les fonctionnalités des deux sources précédents dans le source `tintin.c`. Faites également en sorte que plusieurs valeurs (questions) puissent figurer sur la ligne de commande et que la validité de toutes les valeurs soit testée avant que de lancer les recherches dans le tableau.

Exemple :

```
$ ./tintin
./tintin: At least one parameter is expected
Syntax: ./tintin VALUE...
$ ./tintin RG 1907 1983 123434625465780679089680 Jo Zette Jocko
./tintin: Value out of range: '123434625465780679089680'
$ ./tintin 1937 le Ra to 1956
--- characters appearing for the first time in 1937
Ramon Bada
Alonzo Perez
Basil Bazaroff
Rodrigo Tortilla
--- characters whose first or last names contain "le"
Marc Charlet
--- characters whose first or last names contain "Ra"
Ramon Bada
Roberto Rastapopoulos
Ramon Zarate
--- characters whose first or last names contain "to"
Arturo Benedetto Giovanni Giuseppe Pietro Archangelo Alfredo Cartoffoli
Roberto Rastapopoulos
--- characters appearing for the first time in 1956
Arturo Benedetto Giovanni Giuseppe Pietro Archangelo Alfredo Cartoffoli
Alfredo Topolino
Seraphin Lampion
Stephan Szprinkoth
```

Il pourra être intéressant d'introduire l'alias :

```
//  parameter : alias d'une structure anonyme permettant de mémoriser soit un
//    entier de type long int, champ number, soit un pointeur vers une chaine de
//    caractères, champ string. Le champ category mémorise le type de valeur
//    mémorisée, NUMBER pour le premier cas, STRING pour le deuxième.
typedef struct {
  enum {
    NUMBER, STRING
  } category;
  union {
    long int number;
    const char *string;
  } u;
} parameter;
```

puis de déclarer et utiliser dans la fonction principale un tableau de `parameter` de longueur `argc`. Première utilisation : initialisation à partir des paramètres figurant sur la ligne de commande, l'initialisation prenant fin dès lors que l'un des paramètres n'est pas valide. Deuxième utilisation : recherches dans le tableau. Par souci de lisibilité, introduisez les deux fonctions suivantes :

```
//  parameter_initialize : initialise l'objet pointé par p à partir de la chaine
//    de caractères pointée par s. Si la chaine pointée par s correspond à
//    l'écriture en base 10 d'un entier mais que cet entier ne peut être codé
//    sur le type long int, la fonction ne modifie par l'objet pointé par p et
//    renvoie un pointeur vers une chaine de caractères signifiant une erreur.
//    La fonction renvoie sinon NULL.
const char *parameter_initialize(parameter *p, const char *s);
```

```
//  parameter_execute : affiche sur la sortie standard tous les personnages
//    d'une liste qui valident un test associé à l'objet pointé par p. La liste
//    consiste en la suite des n premiers composants d'un tableau dont q est
//    l'adresse du premier composant. Si l'objet pointé par p correspond à un
//    entier, le test est validé lorsque l'année de première apparition du
//    personnage est égale à cet entier. Sinon, l'objet pointé par p correspond
//    à une chaine de caractères et le test est validé lorsque le nom ou le(s)
//    prénom(s) du personnage contiennent cette chaine. L'affichage de tout
//    personnage est réalisé par un appel à la fonction character_display_names.
//    L'affichage débute par l'envoi sur la sortie standard d'une ligne de texte
//    rappelant la teneur du test.
void parameter_execute(parameter *p, const character *q, size_t n);
```

Si vous parvenez à répondre à la demande de plusieurs valeurs sur la ligne de commande, la spécification de votre source devrait être la suivante :

```
//  Utilisation :
//    tintin VALUE...
//
//  Pour chaque VALUE figurant en paramètre sur la ligne de commande, affiche
//    les personnages d'albums de Tintin figurant dans une très petite base de
//    données qui satisfont à une requête associée à VALUE.
//  Selon que VALUE correspond à l'écriture d'un entier en base 10 et, dans ce
//    cas, que cet entier est codable sur le type long int ou que VALUE est une
//    chaine de caractères, affiche tous les personnages qui, dans le premier
//    cas, ont fait leur première apparition dans un album paru l'année égale à
//    l'entier ou dont, dans le deuxième, le nom ou les prénom(s) contiennent la
//    chaine.
//  Lorsque VALUE est la chaine vide, affiche tous les personnages figurant dans
//    la base de données.
//
//  Renvoie EXIT_FAILURE à l'environnement d'exécution si argc est inférieur ou
//    égal à l'unité ou si l'une des VALUEs correspond à l'écriture d'un entier
//    en base 10 mais que cet entier n'est pas codable sur le type long int.
//  Renvoie EXIT_SUCCESS dans tous les autres cas.
```

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 5

Relevés météorologiques et fichiers textes

Objectifs : maitrise de l'entrée ; expression des invariants de boucle liés à la lecture.
Prérequis : notion de flot texte ; fonctions `fopen`, `fclose`, `fscanf` et `feof` ; agrégation structure.
Travail minimum : exercices 1 à 3.

Figurent dans le dossier `data/` [10] vingt fichiers de la forme `boos`*nnnn*`.csv`. Ces vingt fichiers sont des textes au format CSV, le séparateur de valeurs étant ici la tabulation. Ils contiennent des relevés météorologiques [11] de la station de Boos [12] des années 2004 à 2023, rangés dans l'ordre chronologique, du 1er janvier au 31 décembre.

À partir de l'année 2007, chacune des lignes fait apparaitre sept rubriques :

— le numéro de l'année ;

— le numéro du mois ;

— le numéro du jour dans le mois (quantième) ;

— la *température maximale du jour*, notée *Tx* par les météorologistes, définie comme la température maximale, exprimée en degrés Celsius, relevée entre 6 h TU (temps universel) et 6 h TU le lendemain ;

— la *température minimale du jour*, notée *Tn*, définie comme la température minimale, exprimée en degrés Celsius, relevée entre 18 h TU la veille et 18 h TU ;

— les *précipitations du jour*, quantité notée *Rr*, c'est-à-dire la quantité de pluie, de neige ou de grêle, exprimée en mm d'eau, tombée entre 6 h TU et 6 h TU le lendemain ;

— l'*ensoleillement du jour*, quantité notée *W*, défini comme le nombre d'heures de soleil entre 6 h TU et 6 h TU le lendemain.

Les trois premières rubriques sont exprimées sous forme entière, tandis que les quatre dernières le sont sous forme flottante, avec le point comme séparateur décimal.

Pour fixer les idées, voici les trois premières et les trois dernières lignes de 2008 (extraites du fichier `boos2008.csv` donc), l'extrémité des flèches symbolisant les taquets de tabulation :

```
2008⟩1⟩1⟩5.9⟩3.1⟩0⟩0
2008⟩1⟩2⟩1.8⟩-2.1⟩0⟩0.6
2008⟩1⟩3⟩3.6⟩-2.8⟩2.4⟩0
...
2008⟩12⟩29⟩1.3⟩-6.6⟩0⟩5.7
2008⟩12⟩30⟩1.9⟩-5.2⟩0⟩0
2008⟩12⟩31⟩4.6⟩-1⟩0⟩4.5
```

On y comprendra, par exemple, que le maximum des températures maximales du jour pour toute l'année 2008 a été d'au moins 5,9 °C, que les précipitations ont été d'au moins 2,4 mm et que l'ensoleillement a été d'au moins 10,8 h.

La rubrique ensoleillement du jour n'a été renseignée qu'à partir du 17-04-2006 : avant cette date, seules les six premières rubriques figurent dans les fichiers.

10. Voir fiche de TP précédente.
11. Source des données : `http://www.meteociel.fr`.
12. La ville de Boos abrite un site météorologique pour Rouen et sa région.

Il vous est demandé d'écrire plusieurs programmes C dont l'objet est de dénombrer, sommer, moyenner, calculer un extrémum ou représenter graphiquement certaines données lues dans l'un quelconque des fichiers dont le nom figure sur la ligne de commande. Dans un premier temps, le terme « quelconque » devra être compris avec le sens « quelconque parmi les fichiers complets » ; l'extension aux autres fichiers, à savoir ceux des années antérieures à 2007, non complets, est l'objet de l'exercice 4.

Placez le fichier `meteocsv_sum_rr.c` qui figure dans le dossier `algo1_src/tp/5/` dans le dossier du TP courant. Prenez connaissance de son contenu. Produisez l'exécutable associé. Lancez-le sur des fichiers de relevés météorologiques.

Vous devez coller à ce source pour la suite : découpage, spécifications, invariants de boucle, format d'affichage des diverses mesures, messages d'erreur. Pour les exercices 1 à 4, chaque mesure affichée doit, si elle est flottante, l'être avec un chiffre après le point décimal, être suivie d'une espace, de son unité, d'une tabulation, du nom du fichier et d'une fin de ligne.

Exercice 1

Créez le source `meteocsv_avg_w.c` et faites en sorte qu'il permette l'affichage de la moyenne de l'ensoleillement *W*. Prévoyez le cas du fichier vide, avec un message d'erreur à l'avenant : `"Empty file"` (un fichier vide, `empty.txt`, figure opportunément dans le dossier `data/`).

Exercice 2

Une journée est dite *de forte chaleur* lorsque sa température maximale *Tx* est supérieure ou égale à 30 °C.

Créez le source `meteocsv_ndays_txpp.c` et faites en sorte qu'il permette l'affichage du nombre de journées de forte chaleur.

Exercice 3

Créez le source `meteocsv_max_tx.c` et faites en sorte qu'il permette l'affichage du maximum des températures maximales *Tx*. Prévoyez, là encore, le cas du fichier vide.

Exercice 4

Trouvez une parade pour que les sources précédents — ainsi que ceux qui suivraient — tournent correctement sur les vingt fichiers.

Exercice 5

Développez un programme qui affiche un graphique, développé verticalement, sur lequel figurent les températures maximales *Tx* et minimales *Tn*. Prévoyez :

— le paramétrage de la commande : les bornes minimale et maximale des températures qui doivent figurer sur le graphique, l'échelle précisant le nombre de caractères par degré, le pas de la graduation ;

— le dessin de l'axe « 0 °C » lorsque 0 appartient à l'intervalle allant de la borne minimale à la borne maximale ;

— l'affichage des graduations au début de chaque mois ;

— une symbolique différenciée pour les températures maximales *Tx* et minimales *Tn* ;

— l'affichage possible du symbole d'une température sur l'axe ;

— la possible confusion des affichages des températures maximale *Tx* et minimale *Tn* pour un jour donné, avec une symbolique différente des précédentes ;

— le fait que la température maximale *Tx* puisse être strictement inférieure à la température minimale *Tn* pour un jour donné [13] ;

— le dépassement des bornes minimale et maximale.

13. Mesures partielles sur une plage de 24 h parfois dues à… des conditions météorologiques défavorables.

Pour fixer les idées, voici l'aide affichée par un exécutable qui répond au problème, des extraits de traces d'exécutions puis des commentaires sur ces extraits :

```
$ ./meteocsv_grapher_txtn
Syntax: ./meteocsv_grapher_txtn TMIN TMAX FOCUS GRADUATION [FILE]...
$ ./meteocsv_grapher_txtn -10 35 1 5 ../data/boos2010.csv
         -10   -5    0    5   10   15   20   25   30   35
           |    |    |    |    |    |    |    |    |    |
01-01-2010           - +
02-01-2010       -   |  +
03-01-2010       -   | +
04-01-2010   -      +|
05-01-2010   -       | +
06-01-2010      - +  |
07-01-2010 -      +  |
08-01-2010   -     + |
09-01-2010     -    +|
10-01-2010         -+|
11-01-2010         - +
12-01-2010      -    +
13-01-2010         - |   +
...
21-11-2010           |   - +
22-11-2010           |   *
23-11-2010         - |       +
24-11-2010           |-      +
25-11-2010          -|   +
26-11-2010        -  |   +
27-11-2010         - +
28-11-2010      -    +
29-11-2010      -    | +
30-11-2010      -   +|
         -10   -5    0    5   10   15   20   25   30   35
           |    |    |    |    |    |    |    |    |    |
01-12-2010      - +  |
...
30-12-2010           -+
31-12-2010         - +
$ ./meteocsv_grapher_txtn -10 35 1 5 ../data/boos2005.csv
...
31-01-2005           |   -   +
         -10   -5    0    5   10   15   20   25   30   35
           |    |    |    |    |    |    |    |    |    |
01-02-2005           |    -  +
...
11-02-2005           |       -  +
12-02-2005           |        +-
13-02-2005           |-    +
...
```

```
$ ./meteocsv_grapher_txtn -10 20 2 5 ../data/boos2012.csv
...
         -10        -5         0         5        10        15        20
           |         |         |         |         |         |         |
01-02-2012         -          +|
02-02-2012     -             + |
03-02-2012  -                  |+
04-02-2012        -          + |
05-02-2012   -        +        |
06-02-2012   -               + |
07-02-2012<            +       |
08-02-2012   -           +     |
09-02-2012  -                 +|
10-02-2012 -              +    |
11-02-2012<               +    |
12-02-2012<                    |   +
13-02-2012                  -  |        +
...
$ ./meteocsv_grapher_txtn -10 35 1 5 ../data/boos2020.csv
...
31-07-2020           |                -                  >
         -10   -5    0    5   10   15   20   25   30   35
           |    |    |    |    |    |    |    |    |    |
01-08-2020           |                  -       +
02-08-2020           |           -           +
03-08-2020           |           -         +
04-08-2020           |       -                 +
05-08-2020           |           -                   +
06-08-2020           |             -                    +
07-08-2020           |                -                   >
08-08-2020           |                     -           +
09-08-2020           |                     -              >
10-08-2020           |                  -               +
11-08-2020           |                    -               >
12-08-2020           |                  -                +
...
```

Notez : dans la première trace pour l'année 2010, l'affichage de températures maximales ou minimales sur l'axe les 01-01, 11-01, 12-01, 30-12 et 31-12 ainsi que la confusion des affichages des températures maximale et minimale le 22-11 ; dans la deuxième pour 2005, une température maximale strictement inférieure à la température minimale le 12-02 ; dans la troisième pour 2012, des températures en deçà de la borne minimale les 07-02, 11-02 et 12-02, ainsi qu'un « focus » porté à deux caractères par degré ; dans la dernière pour 2020, des températures en delà de la borne maximale les 31-07, 07-08, 09-08 et 11-08.

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 6

Relevés météorologiques et fichiers binaires homogènes

Objectifs : maitrise des fichiers homogènes ; expression des invariants de boucle liés à la lecture et à l'écriture.
Prérequis : fiche de TP n° 5 ; notion de flot binaire ; fonctions `fread` et `fwrite`.
Travail minimum : exercices 1 à 3.

Le but premier est ici de permettre par la suite de pouvoir exploiter des données météorologiques issues de fichiers binaires homogènes. Il faut pour cela commencer par obtenir les versions binaires homogènes des fichiers de relevés météorologiques fournis au format CSV.

Un travail préliminaire sur les noms des fichiers est proposé à l'exercice 1. L'exercice 2 est dévolu à la production des fichiers binaires homogènes. L'exercice 3 permet de tester sur les fichiers binaires des calculs précédemment déclinés sur les fichiers au format CSV. Les deux exercices suivants proposent d'aller plus loin.

Copiez dans votre dossier dévolu à la fiche de TP courante les deux sources C situés dans le dossier `algo1_src/tp/6/`.

Exercice 1

Complétez le source `chsuff_trial.c` de manière à ce qu'il soit conforme à sa spécification.

Déclarez la « zone mémoire suffisamment grande » comme un tableau de longueur variable. Recourez aux fonctions `memcpy` et `strlen` de l'en-tête standard `<string.h>`.

Exemple, en supposant l'exécutable lancé dans le dossier de TP courant, avec pour seuls fichiers présents dans ce dossier les deux sources C fournis et l'exécutable :

```
$ ls
chsuff_trial  chsuff_trial.c  meteocsv_create_bin.c
$ ./chsuff_trial
Syntax: ./chsuff_trial SUFF REPL [FILE]...
$ ./chsuff_trial ".c" "" *
./chsuff_trial: Invalid suffix for file 'chsuff_trial'
chsuff_trial.c⟶chsuff_trial
meteocsv_create_bin.c⟶meteocsv_create_bin
$ ./chsuff_trial ".c" "_dot_c" *
./chsuff_trial: Invalid suffix for file 'chsuff_trial'
chsuff_trial.c⟶chsuff_trial_dot_c
meteocsv_create_bin.c⟶meteocsv_create_bin_dot_c
$ ./chsuff_trial "" "something" *
chsuff_trial⟶chsuff_trialsomething
chsuff_trial.c⟶chsuff_trial.csomething
meteocsv_create_bin.c⟶meteocsv_create_bin.csomething
```

Exercice 2

Complétez le source `meteocsv_create_bin.c` de manière à ce qu'il soit conforme à sa spécification. À titre indicatif, la chaine désignée par la macroconstante `REPL` a été fixée à `".bin"` pour les exemples dont il est fait état dans la suite. Si vous avez réalisé l'exercice 4 de la fiche n° 5, remplacez l'actuelle définition de la fonction `freport_csv_get` par la vôtre.

Produisez les fichiers binaires associés aux fichiers CSV fournis. Renseignez-vous sur leurs tailles exactes [14]. Remarquez que, sauf erreur, elles sont le produit d'une certaine constante par 365 ou 366.

Exercice 3

Adaptez au cas des fichiers binaires homogènes l'un quelconque des programmes de calcul du total des précipitations, de la moyenne de l'ensoleillement, du nombre de journées de forte chaleur ou du maximum des températures maximales fournis ou développés à l'occasion de la fiche n° 5.

Faites disparaitre la désormais inutile fonction `freport_csv_get`.

Testez.

Exercice 4

Les calculs classiques de dénombrement en météorologie sont les calculs des nombres de jours :

— *de gelée* : température minimale inférieure ou égale à 0 °C ;
— *de forte gelée* : température minimale inférieure ou égale à −5 °C ;
— *sans dégel* : température maximale inférieure ou égale à 0 °C ;
— *de chaleur* : température maximale supérieure ou égale à 25 °C ;
— *de forte chaleur* : température maximale supérieure ou égale à 30 °C ;
— *de canicule* : conjonction d'une forte chaleur et d'une température minimale supérieure ou égale à 20 °C ;
— *avec précipitations* : précipitations non nulles ;
— *avec ensoleillement* : ensoleillement non nul.

Développez une calculatrice capable de produire ces dénombrements sur les fichiers météorologiques au format binaire. Le type du dénombrement attendu doit être spécifié par le premier paramètre de la calculatrice.

Exemple :

```
$ ./meteobin_ndays
Syntax: ./meteobin_ndays SPEC [FILE]...

Display the result of countings with the SPEC specialisation to each of the
meteorological files in binary format FILEs.

Possible values for SPEC are:
  tn-     frost day: Tn <= 0 °C
  tn--    day of heavy frost: Tn <= -5 °C
  tx-     no-thaw day: Tx <= 0 °C
  tx+     hot day: Tx >= 25 °C
  tx++    very hot day: Tx >= 30 °C
  txn+    day of heatwave: Tx >= 30 °C and Tn >= 20 °C
  rr+     day with precipitation: Rr > 0 mm
  w+      day with sunshine: W > 0 h

With fewer parameters than required, display this help.
$ ./meteobin_ndays tx++ ../data/*200*.bin
     3 day(s)⟶../data/boos2004.bin
     4 day(s)⟶../data/boos2005.bin
    10 day(s)⟶../data/boos2006.bin
     1 day(s)⟶../data/boos2007.bin
```

14. Par exemple à l'aide de la commande « `ls -l` ».

```
     2 day(s)⟶../data/boos2008.bin
     5 day(s)⟶../data/boos2009.bin
$ ./meteobin_ndays w+ ../data/*200*.bin
./meteobin_ndays : Missing data in file '../data/boos2004.bin'
./meteobin_ndays : Missing data in file '../data/boos2005.bin'
./meteobin_ndays : Missing data in file '../data/boos2006.bin'
   263 day(s)⟶../data/boos2007.bin
   293 day(s)⟶../data/boos2008.bin
   313 day(s)⟶../data/boos2009.bin
```

Attention : il ne s'agit pas d'écrire autant de fonctions agissant sur les fichiers que de dénombrements différents ; il s'agit, essentiellement, de ne développer qu'une seule fonction de parcours de fichiers qui prend en paramètre une condition sur un relevé.

Exercice 5

Allez plus loin encore en utilisant deux paramètres pour guider le calcul attendu.

Exemple. Notez au passage l'introduction d'une nouvelle mesure, *Ta*, obtenue par calcul et jugée représentative de la température moyenne du jour :

```
$ ./meteobin
Syntax: ./meteobin CALC SPEC [FILE]...

Display the result of the CALC calculation with the SPEC specialisation to each
of the meteorological files in binary format FILEs.

Possible values for CALC are:
  ndays   number of days
  sum     sum
  avg     average
  max     maximum
  min     minimum

When CALC is 'ndays', possible values for SPEC are:
  tn-     frost day: Tn <= 0 °C
  tn--    day of heavy frost: Tn <= -5 °C
  tx-     no-thaw day: Tx <= 0 °C
  tx+     hot day: Tx >= 25 °C
  tx++    very hot day: Tx >= 30 °C
  txn+    day of heatwave: Tx >= 30 °C and Tn >= 20 °C
  rr+     day with precipitation: Rr > 0 mm
  w+      day with sunshine: W > 0 h

Otherwise, possible values for SPEC are:
  tx      daily maximum temperature Tx
  tn      daily minimum temperature Tn
  ta      daily average temperature (Tx + Tn) / 2
  rr      daily precipitation Rr
  w       daily sunlight W

With fewer parameters than required, display this help.
```

```
$ ./meteobin ndays tn-- ../data/*200*.bin
     1 day(s)⟶../data/boos2004.bin
     7 day(s)⟶../data/boos2005.bin
     4 day(s)⟶../data/boos2006.bin
     4 day(s)⟶../data/boos2007.bin
     2 day(s)⟶../data/boos2008.bin
    14 day(s)⟶../data/boos2009.bin
$ ./meteobin sum rr ../data/*200*.bin
 728.3 mm⟶../data/boos2004.bin
 698.6 mm⟶../data/boos2005.bin
 777.1 mm⟶../data/boos2006.bin
 950.2 mm⟶../data/boos2007.bin
 834.0 mm⟶../data/boos2008.bin
 773.4 mm⟶../data/boos2009.bin
$ ./meteobin avg w ../data/*200*.bin
./meteobin : Erroneous data in file '../data/boos2004.bin'
./meteobin : Erroneous data in file '../data/boos2005.bin'
./meteobin : Erroneous data in file '../data/boos2006.bin'
   3.6 h⟶../data/boos2007.bin
   3.9 h⟶../data/boos2008.bin
   4.4 h⟶../data/boos2009.bin
$ ./meteobin max tx ../data/*200*.bin
  31.3 °C⟶../data/boos2004.bin
  32.1 °C⟶../data/boos2005.bin
  34.8 °C⟶../data/boos2006.bin
  30.6 °C⟶../data/boos2007.bin
  31.7 °C⟶../data/boos2008.bin
  35.0 °C⟶../data/boos2009.bin
$ ./meteobin avg ta ../data/*200*.bin
  10.7 °C⟶../data/boos2004.bin
  10.9 °C⟶../data/boos2005.bin
  11.2 °C⟶../data/boos2006.bin
  11.2 °C⟶../data/boos2007.bin
  10.7 °C⟶../data/boos2008.bin
  10.9 °C⟶../data/boos2009.bin
```

Exercice 6

Allez plus loin encore en utilisant trois paramètres pour guider le calcul attendu, sur les fichiers binaires comme CSV.

Exemple, avec extrait (début) de l'aide uniquement.

```
$ ./meteo
Syntax: ./meteo FRMT CALC SPEC [FILE]...

Display the result of the CALC calculation with the SPEC specialisation to each
of the meteorological FILEs in FRMT format.

Possible values for FRMT are:
  csv     CSV tab-delimited
  bin     binary
```