Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 0

Travaux pratiques : modalités et réglages

Environnement

LORS DES SÉANCES DE TP, 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 Algo1 de la plateforme UniversiTICE une archive au format `.tar` ou `.tar.gz` contenant des fichiers source `.c`, des fichiers sources `.h` et des fichiers `makefile`, et uniquement ceux-là, créés, modifiés ou utilisés durant la ou les séances dévolues au traitement des exercices figurant sur la fiche. Vous pourrez par la suite déposer des versions améliorées pendant une semaine.

Si vous avez travaillé à plusieurs, indiquez les noms et prénoms de chacun des membres du groupe au début de tous les fichiers modifiés.

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

Réglage de l'EDI Geany

AVANT DE LANCER GEANY : à partir du dossier `geany/` qui figure dans le dossier `algo1_src` de l'archive `algo1_src.tar` du cours Algo1, copiez dans votre dossier `~/.config/geany` les fichiers `geany.conf` et `filedefs/filetypes.c` (écrasez les fichiers existants le cas échéant).

Installation privée sous Windows 10

SI JAMAIS VOUS DEVIEZ, POUR D'OBSCURES RAISONS, vous résoudre à travailler la programmation en C sur votre propre machine et sous Windows :

Téléchargements

`http://www.geany.org/Download/Releases`, cliquez sur le lien `geany-`*version*`_setup.exe`.
`http://sourceforge.net/projects/mingw/`, cliquez sur le bouton Download.

Installation

1) Lancez `geany-`*version*`_setup.exe` et répondez favorablement à toute demande de validation sans rien modifier aux propositions.

2) Lancez `mingw-get-setup.exe` et répondez favorablement à toute demande de validation sans rien modifier aux propositions. Arrivé sous MinGW Installation Manager, sélectionnez les paquets `mingw32-base` et `msys-base` (dans Basic Setup). Appliquez les changements puis quittez le Manager.

3) Affichez les paramètres systèmes avancés (obtenus par exemple en cliquant sur Paramètres puis en recherchant « paramètres systèmes avancés »). Sous la fenêtre « Propriétés système », trouvez dans le tableau « Variables d'environnement » la variable `Path` en vous servant de l'ascenseur. Double-cliquez sur la ligne sur laquelle la variable est située, ajoutez

```
;C:\MinGW\bin;C:\MinGW\msys\1.0\bin
```

à la valeur de la variable. Validez.

4) Redémarrez pour que toutes ces modifications prennent effet.

Subtilités

1) Les fichiers de configurations de Geany sont situés dans votre propre dossier `AppData\Roaming\geany`.

2) L'installation du paquet `msys-base` et l'ajout du chemin `C:\MinGW\msys\1.0\bin` à la variable `Path` permet d'avoir accès à des versions Windows des commandes Linux `cat`, `ls`, `rm`, `tar`...

3) En lieu et place de la commande `make`, il faut utiliser `mingw32-make`.

4) Dans les fichiers `makefile` fournis, il faut ajouter l'extension `.exe` à la variable `executable`.

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 *Informatique : 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 : programmation de tous les exercices de la fiche.

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` [1], lit les chaines de caractères d'une longueur maximale à fixer [2], ne comprenant aucun caractère d'espacement et les affiche les unes après les autres, 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
```

1. Voir Annexe B1.3 « Les entrées mises en forme » du KR par exemple.
2. 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.
```

Exercice 2

Réalisez une copie du fichier précédent et nommez-la `str_islong.c`.

Transformez le programme de sorte que, pour chacune des chaines lues, il ajoute en fin de ligne l'information « `long =` *n* » lorsque la chaine correspond à l'écriture en base 10 d'un entier codable sur le type **`long int`**, *n* étant la valeur de cet entier. Dans le cas contraire, l'information ajoutée sera « `not long` ».

Contraintes supplémentaires :

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

— en cas d'information « `long =` *n* », 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`[3] et à la variable standard `errno`[4] 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 supérieure à 16, si l'entrée est :

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

la sortie sera :

```
1———⟩The———⟩not long
2———⟩C———⟩not long
3———⟩Programming———⟩not long
4———⟩Language———⟩not long
5———⟩2nd———⟩not long
6———⟩Edition⟩not long
7———⟩March—⟩not long
8———⟩22———⟩long = 22
9———⟩1988—⟩long = 1988
```

Attention : la longueur 8 utilisée à l'exercice précédent 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. Conseil : fixez la longueur à 32.

Exercice 3

Réalisez une copie du fichier précédent et nommez-la `str_operclass.c`.

Transformez le programme de sorte que, pour chacune des chaines lues, il ajoute cette fois en fin de ligne l'information « `operand =` *n* » si la chaine correspond à l'écriture en base 10 de l'entier

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

n et que *n* est codable sur le type `long int`, « `operator` » si la chaine égale l'une des chaines `"ADD"`, `"MUL"` ou `"END"`, et, sinon, « `rejected form` ».

Contraintes supplémentaires :

— en cas de rejet (*rejected*), il devra être mis fin au programme immédiatement avec renvoi de la valeur `EXIT_FAILURE` ;

— pour comparer les chaines, vous ferez appel à la fonction `strcmp`[5].

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

Par exemple, avec une longueur maximale supérieure à 16, si l'entrée est :

```
20 100 MUL 18 ADD END
UN ET UN, DEUX.
```

la sortie sera :

```
1⟶20⟶operand = 20
2⟶100⟶operand = 100
3⟶MUL⟶operator
4⟶18⟶operand = 18
5⟶ADD⟶operator
6⟶END⟶operator
7⟶UN⟶rejected form
```

5. 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 : cours *Informatique : Bases de la programmation impérative* ; fiche n° 1 de TP.
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 : $a\,b\,c\times d+\times e+$.

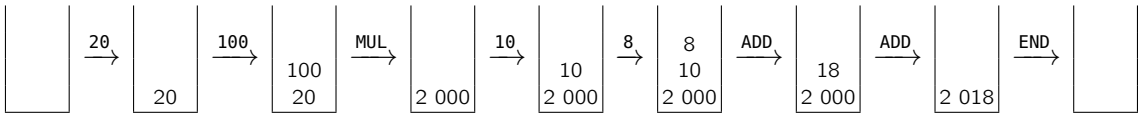
C'est la deuxième notation, également connue sous le nom de *notation polonaise inverse* qui nous intéresse ici : il s'agit d'évaluer toutes les expressions arithmétiques données en notation polonaise inverse qui figurent sur l'entrée. L'ensemble des lexèmes du langage de départ est réduit : des entiers ; 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 10 8 ADD ADD END
```

le programme à développer doit ainsi afficher 2 018.

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 `str_operclass.c`, résultat du travail sur la fiche précédente, et nommez-la `calc.c`.

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 (syntaxe, débordement de la pile, hauteur de la pile insuffisante), il devra être mis immédiatement fin au programme.

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.

Travail minimum : sur deux séances, l'un des exercices 1 ou 2, puis 3 ou 4, puis 5 ou 6 ; preuve manuscrite complète de la correction de deux des fonctions de troncature.

Exercice 1

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

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

en procédant selon les puissances croissantes de x. Contraintes : ne faites appel à aucune autre fonction ; utilisez au plus trois variables locales à la fonction.

Documentez la fonction de manière détaillée : fournissez sous forme de commentaires son assertion d'entrée, son assertion de sortie, l'invariant de boucle et la quantité de contrôle.

La fonction principale de votre programme doit, pour tous les couples (x, n) lus sur l'entrée standard, afficher la valeur renvoyée par la fonction de troncature. Comparez avec les valeurs données en fin de sujet pour n petit.

Exercice 2

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

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

Exercice 3

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

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

Exercice 4

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

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

Exercice 5

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

$$\sum_{j=0}^{n} \frac{(2j)!}{4^j\,(j!)^2\,(2j+1)} x^{2j+1}.$$

Exercice 6

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

$$1 + \sum_{j=1}^{n} \frac{(-1)^{j+1}(2j)!}{4^j\,(j!)^2\,(2j-1)}\,x^j.$$

Exercice 7

Produisez les tableaux de valeurs qui figurent en fin de sujet pour les fonctions de troncature que vous avez développées. Utilisez pour cela la bibliothèque mathématique standard, avec ses fonctions `log`[6] ou `log1p`[7], `atan`, `sin`, `cos`, `asin` et `sqrt`. Sous Linux, passez nécessairement l'option `-lm` (liaison avec la bibliothèque mathématique) à `gcc`.

Exercice 8

Application des exercices 2 et 5 : proposez des calculs de valeurs approchées de π.

Production d'un programme réunissant les fonctions de calcul de troncatures des exercices 1 à 6. Pour chacune des fonctions : sur la première ligne, une expression de la somme calculée ; sur la deuxième ligne, les mentions des contenus des colonnes des lignes suivantes ; sur les lignes suivantes, pour les valeurs de x signifiées dans la colonne de gauche, la valeur fournie par la fonction mathématique de référence, puis les troncatures pour diverses valeurs de n. Le nombre de chiffres après le point décimal représentables sans erreur d'arrondi pour le type réel flottant utilisé égale 15.

```
somme((-1)^(j+1) * x^j / j ; j == 1 ... n)
          x     ln(1+x)       n = 1       n = 2       n = 3       n = 4
-0.25000000 -0.28768207 -0.25000000 -0.28125000 -0.28645833 -0.28743490
-0.20000000 -0.22314355 -0.20000000 -0.22000000 -0.22266667 -0.22306667
-0.15000000 -0.16251893 -0.15000000 -0.16125000 -0.16237500 -0.16250156
-0.10000000 -0.10536052 -0.10000000 -0.10500000 -0.10533333 -0.10535833
-0.05000000 -0.05129329 -0.05000000 -0.05125000 -0.05129167 -0.05129323
 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
 0.05000000  0.04879016  0.05000000  0.04875000  0.04879167  0.04879010
 0.10000000  0.09531018  0.10000000  0.09500000  0.09533333  0.09530833
 0.15000000  0.13976194  0.15000000  0.13875000  0.13987500  0.13974844
 0.20000000  0.18232156  0.20000000  0.18000000  0.18266667  0.18226667
 0.25000000  0.22314355  0.25000000  0.21875000  0.22395833  0.22298177

somme((-1)^j * x^(2j+1) / (2j+1) ; j == 0 ... n)
          x   arctan(x)       n = 0       n = 1       n = 2       n = 3
-0.25000000 -0.24497866 -0.25000000 -0.24479167 -0.24498698 -0.24497826
-0.20000000 -0.19739556 -0.20000000 -0.19733333 -0.19739733 -0.19739550
-0.15000000 -0.14888995 -0.15000000 -0.14887500 -0.14889019 -0.14888994
-0.10000000 -0.09966865 -0.10000000 -0.09966667 -0.09966867 -0.09966865
-0.05000000 -0.04995840 -0.05000000 -0.04995833 -0.04995840 -0.04995840
```

6. `log` est un faux ami, puisqu'il s'agit du logarithme népérien.
7. `log1p` est l'ami de circonstance, puisque s'agit du logarithme népérien de 1 plus (« `1p` ») l'argument passé à la fonction.

```
 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
 0.05000000  0.04995840  0.05000000  0.04995833  0.04995840  0.04995840
 0.10000000  0.09966865  0.10000000  0.09966667  0.09966867  0.09966865
 0.15000000  0.14888995  0.15000000  0.14887500  0.14889019  0.14888994
 0.20000000  0.19739556  0.20000000  0.19733333  0.19739733  0.19739550
 0.25000000  0.24497866  0.25000000  0.24479167  0.24498698  0.24497826

somme((-1)^j * x^(2j+1) / (2j+1)! ; j == 0 ... n)
          x      sin(x)       n = 0       n = 1       n = 2       n = 3
-0.25000000 -0.24740396 -0.25000000 -0.24739583 -0.24740397 -0.24740396
-0.20000000 -0.19866933 -0.20000000 -0.19866667 -0.19866933 -0.19866933
-0.15000000 -0.14943813 -0.15000000 -0.14943750 -0.14943813 -0.14943813
-0.10000000 -0.09983342 -0.10000000 -0.09983333 -0.09983342 -0.09983342
-0.05000000 -0.04997917 -0.05000000 -0.04997917 -0.04997917 -0.04997917
 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
 0.05000000  0.04997917  0.05000000  0.04997917  0.04997917  0.04997917
 0.10000000  0.09983342  0.10000000  0.09983333  0.09983342  0.09983342
 0.15000000  0.14943813  0.15000000  0.14943750  0.14943813  0.14943813
 0.20000000  0.19866933  0.20000000  0.19866667  0.19866933  0.19866933
 0.25000000  0.24740396  0.25000000  0.24739583  0.24740397  0.24740396

somme((-1)^j * x^(2j) / (2j)! ; j == 0 ... n)
          x      cos(x)       n = 0       n = 1       n = 2       n = 3
-0.25000000  0.96891242  1.00000000  0.96875000  0.96891276  0.96891242
-0.20000000  0.98006658  1.00000000  0.98000000  0.98006667  0.98006658
-0.15000000  0.98877108  1.00000000  0.98875000  0.98877109  0.98877108
-0.10000000  0.99500417  1.00000000  0.99500000  0.99500417  0.99500417
-0.05000000  0.99875026  1.00000000  0.99875000  0.99875026  0.99875026
 0.00000000  1.00000000  1.00000000  1.00000000  1.00000000  1.00000000
 0.05000000  0.99875026  1.00000000  0.99875000  0.99875026  0.99875026
 0.10000000  0.99500417  1.00000000  0.99500000  0.99500417  0.99500417
 0.15000000  0.98877108  1.00000000  0.98875000  0.98877109  0.98877108
 0.20000000  0.98006658  1.00000000  0.98000000  0.98006667  0.98006658
 0.25000000  0.96891242  1.00000000  0.96875000  0.96891276  0.96891242

somme((2j)! * x^(2j+1) / (4^j * (j!)^2 * (2j+1)) ; j == 0 ... n)
          x   arcsin(x)       n = 0       n = 1       n = 2       n = 3
-0.25000000 -0.25268026 -0.25000000 -0.25260417 -0.25267741 -0.25268013
-0.20000000 -0.20135792 -0.20000000 -0.20133333 -0.20135733 -0.20135790
-0.15000000 -0.15056827 -0.15000000 -0.15056250 -0.15056820 -0.15056827
-0.10000000 -0.10016742 -0.10000000 -0.10016667 -0.10016742 -0.10016742
-0.05000000 -0.05002086 -0.05000000 -0.05002083 -0.05002086 -0.05002086
 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
 0.05000000  0.05002086  0.05000000  0.05002083  0.05002086  0.05002086
 0.10000000  0.10016742  0.10000000  0.10016667  0.10016742  0.10016742
 0.15000000  0.15056827  0.15000000  0.15056250  0.15056820  0.15056827
 0.20000000  0.20135792  0.20000000  0.20133333  0.20135733  0.20135790
 0.25000000  0.25268026  0.25000000  0.25260417  0.25267741  0.25268013
```

```
1 + somme((-1)^(j+1) * (2j)! * x^j / (4^j * (j!)^2 * (2j-1)) ; j == 1 ... n)
          x    rac(1+x)       n = 1       n = 2       n = 3       n = 4
-0.25000000  0.86602540  0.87500000  0.86718750  0.86621094  0.86605835
-0.20000000  0.89442719  0.90000000  0.89500000  0.89450000  0.89443750
-0.15000000  0.92195445  0.92500000  0.92218750  0.92197656  0.92195679
-0.10000000  0.94868330  0.95000000  0.94875000  0.94868750  0.94868359
-0.05000000  0.97467943  0.97500000  0.97468750  0.97467969  0.97467944
 0.00000000  1.00000000  1.00000000  1.00000000  1.00000000  1.00000000
 0.05000000  1.02469508  1.02500000  1.02468750  1.02469531  1.02469507
 0.10000000  1.04880885  1.05000000  1.04875000  1.04881250  1.04880859
 0.15000000  1.07238053  1.07500000  1.07218750  1.07239844  1.07237866
 0.20000000  1.09544512  1.10000000  1.09500000  1.09550000  1.09543750
 0.25000000  1.11803399  1.12500000  1.11718750  1.11816406  1.11801147
```

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 4

Données temporelles et structures

Objectifs : utilisation de la construction structure ; passage de références ; compréhension des spécifications informelles d'une fonction ; initialisation de tableaux par liste de valeurs.

Prérequis : opérateurs de référence `&` et de déréférence `*` ; construction structure, mot-clé **`struct`**, opérateurs `.` et `->` ; opérateur `[]`.

Travail minimum : exercices 1 à 3.

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

Vous prendrez connaissance du contenu des fichiers en même temps que vous lirez les explications et les attendus.

Les deux fichiers fournis, `tod1.c` pour *time of day* version 1 et `day1.c` pour *day* version 1, demandent à être complétés : s'ils sont compilables, il n'est pas possible d'en obtenir immédiatement des exécutables. Leur découpage est le suivant :

— inclusions ;

— définition du nom d'un type servant au stockage de données temporelles ;

— spécifications et déclarations de fonctions sur ce type ;

— définition du type, une structure de trois champs dans le premier cas, une structure de quatre champs dans le deuxième ;

— fonction principale.

L'exercice 1 a pour but de compléter le source `tod1.c`. L'exercice 2 est un travail sur une copie du source complété. Même schéma ensuite pour les exercices suivants à partir du source `day1.c`.

Exercice 1

Définissez les fonctions de lecture `get_timeofday` et d'écriture `put_timeofday` des données temporelles du type **`timeofday`**. Testez.

Exemple :

```
$ ./tod1
17 004 58
17 h 4 min 58 s
$ ./tod1
17 5 62
Erreur : donnée heures, minutes ou secondes illégale
$
```

Vous respecterez les contraintes suivantes :

— il est interdit de modifier en quoi que ce soit le type défini ;

— il est interdit de modifier la fonction principale et donc, en particulier, d'accéder aux champs du type **`timeofday`** dans cette fonction principale ;

— la définition des fonctions de lecture et d'écriture est à faire après celle de la fonction principale ;

— il est interdit de ne pas respecter intégralement leurs spécifications ;

— il est obligatoire de définir des macroconstantes pour signifier les limites maximales des heures, minutes et secondes.

Exercice 2

Cette fois, il ne s'agit plus d'obtenir des données temporelles par lecture, mais de récupérer l'heure courante. Et pour ce faire, il va falloir recourir à deux types et à deux fonctions qui figurent dans l'en-tête standard `<time.h>` :

```
#include <time.h>
time_t
struct tm;
time_t time(time_t *timer);
struct tm *localtime(const time_t *timer);
```

Brièvement, le type **`time_t`** est capable de mémoriser le temps (ou, plus exactement, un moment sous forme numérique) tandis que la structure **`struct tm`** dispose de champs, tous du type **`int`**, qui donnent une version éclatée du temps. Parmi les champs de la structure **`struct tm`** :

— `tm_sec` pour les secondes après la minute, dans l'intervalle $[0, 60]$;
— `tm_min` pour les minutes après l'heure, dans l'intervalle $[0, 59]$;
— `tm_hour` pour les heures après minuit, dans l'intervalle $[0, 23]$;
— `tm_mday` (*day of the month*) pour le quantième, dans l'intervalle $[1, 31]$;
— `tm_mon` pour les mois depuis janvier, dans l'intervalle $[0, 11]$;
— `tm_year` pour les années depuis 1900 ;
— `tm_wday` (*day of the week*) pour les jours depuis dimanche, dans l'intervalle $[0, 6]$;
— `tm_yday;` (*day of the year*) pour les jours depuis le 1er janvier, dans l'intervalle $[0, 365]$.

La fonction `time` renvoie la meilleure approximation disponible du temps courant. Si ce temps n'est pas disponible, la fonction renvoie la valeur `(time_t) (-1)`. Si `timer` est différent de `NULL`, la valeur renvoyée est également affectée à l'objet pointé par `timer`.

La fonction `localtime` convertit le temps pointé par `timer` en une version éclatée. Elle renvoie un pointeur vers une structure contenant ce temps éclaté en cas de succès et `NULL` en cas d'échec.

Réalisez une copie du fichier `tod1.c` ; nommez-la `tod2.c`. Incluez l'en-tête `<time.h>`. Donnez comme nouvelle spécification à la première fonction :

```
//  get_timeofday : si le temps courant n'est pas disponible, ne modifie pas
//    l'objet *tptr et renvoie une valeur non nulle. Sinon, affecte à l'objet
//    *tptr les données temporelles heures, minutes et secondes du temps courant
//    et renvoie zéro
```

Remplacez le message d'erreur qui figure dans la fonction principale par :

```
    fprintf(stderr, "Erreur␣:␣temps␣indisponible\n");
```

Modifiez le corps de la fonction `get_timeofday` conformément à sa spécification (vous devriez pouvoir faire disparaitre les macroconstantes). Faites précéder le message dispensé par `put_timeofday` de `"Il␣est␣précisément␣"` et faites-le terminer par un point. Testez.

Exemple :

```
$ ./tod2
Il est précisément 21 h 29 min 49 s.
$ date ; ./tod2
mercredi 21 février 2018, 21:29:51 (UTC+0100)
Il est précisément 21 h 29 min 51 s.
$
```

La deuxième commande est l'enchainement de deux commandes : affichage de la date et de l'heure courante dans le format système par défaut, puis affichage de la seule heure courante suivant la norme en typographie française.

Exercice 3

Complétez le source `day1.c` selon les spécifications qui y figurent ; respectez les interdits énoncés dans l'exercice 1 tout en les adaptant. Donnez nécessairement du

```
#define TM_YEAR_BASE 1900
```

et utilisez la macroconstante ainsi définie. Les noms des jours et des mois devront être mémorisés dans des tableaux initialisés par liste de valeurs. Rappelons qu'en français les noms des jours et des mois sont des mots communs : leur première lettre n'est en majuscule que si le mot débute une phrase. Rappelons aussi que le premier quantième est suivi de « er » : `1er`. Testez.

Exemple :

```
$ ./day1
Nous sommes le mercredi 21 février 2018.
$ date ; ./day1
mercredi 21 février 2018, 22:20:02 (UTC+0100)
Nous sommes le mercredi 21 février 2018.
$
```

Exercice 4

Développez la version 2 de la commande précédente qui supporte qu'un paramètre figure sur la ligne de commande : si ce paramètre est numérique, la commande affiche, si cela est possible, la date du jour courant décalé de la valeur du paramètre.

Exemple :

```
$ ./day2
Nous sommes le mercredi 21 février 2018.
$ ./day2 trois nous irons au bois
Erreur : au plus un paramètre attendu
$ ./day2 deux
Erreur : paramètre de décalage illégal 'deux'
$ ./day2 2148000000
Erreur : paramètre de décalage illégal '2148000000'
$ ./day2 2
Nous serons le vendredi 23 février 2018.
$ ./day2 -20
Nous étions le jeudi 1er février 2018.
$ ./day2 -21
Nous étions le mercredi 31 janvier 2018.
$ ./day2 -22
Nous étions le mardi 30 janvier 2018.
$ ./day2 -158944
Nous étions le lundi 20 décembre 1582.
$
```

Recourez une fois encore à une fonction de l'en-tête `<time.h>` :

```
#include <time.h>
time_t mktime(struct tm *timeptr);
```

La fonction `mktime` est la réciproque de la fonction `localtime`. Attention : les champs `tm_wday` et `tm_yday` n'étant pas pris en compte pour cette conversion, il faudra agir sur le champ `tm_mday`.

Exercice 5

Passez à la version 3 qui accepte encore un autre paramètre optionnel spécifiant la langue. Exemple :

```
$ ./day3
Nous sommes le mercredi 21 février 2018.
$ ./day3 fr
Nous sommes le mercredi 21 février 2018.
$ ./day3 en
Today is Wednesday, 21st February, 2018.
$ ./day3 2 fr
Nous serons le vendredi 23 février 2018.
$ ./day3 2 en
It will be Friday, 23rd February, 2018.
$ ./day3 -20
Nous étions le jeudi 1er février 2018.
$ ./day3 -20 en
It was Thursday, 1st February, 2018.
$
```

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 5

Relevés météorologiques (1) et fichiers texte

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` ; construction structure.
Travail minimum : au moins deux des exercices 1 à 3.

Récupérez le dossier `boos` situé dans le dossier `algo1_src/tp/` et placez-le dans le même dossier que votre dossier dévolu à la fiche de TP courante [8]. Figurent dans le dossier `boos` treize fichiers de la forme `boos`*nnnn*`.csv`. Ces treize fichiers sont des textes au format CSV (*comma separated value*), le séparateur de valeurs étant ici la tabulation. Ils contiennent des relevés météorologiques [9] de la station de Boos [10] des années 2005 à 2017, rangés dans l'ordre chronologique, du 1er janvier au 31 décembre.

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.

Attention : 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.

8. Le dossier `boos` et son contenu sont amenés à resservir.
9. Source des données : `http://www.meteociel.fr`.
10. La ville de Boos abrite le 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. Dans un premier temps, le terme « quelconque » devra être compris avec le sens « quelconque parmi les fichiers complets » ; l'extension aux deux autres fichiers, à savoir `boos2005.csv` et `boos2006.csv`, non complets, est l'objet de l'exercice 4. Chaque mesure affichée doit être suivie de son unité et, si elle est flottante, avec un chiffre après le point décimal.

Récupérez le fichier `meteocsv_sumrr.c` qui figure dans le dossier `algo1_src/tp/5` et mettez-le dans le dossier du TP courant. Prenez connaissance de son contenu : vous devrez vous en inspirer pour la suite, pour les descriptions des sous-programmes, leurs assertions d'entrée et de sortie, leurs invariants de boucle et quantités de contrôle, mais aussi pour la précision des unités des diverses quantités affichées. Compilez-le, exécutez-le sur des fichiers de relevés météorologiques.

Exercice 1

Écrivez un programme qui affiche la moyenne annuelle des températures minimales.

Exercice 2

Même chose pour le nombre de jours avec ensoleillement (ensoleillement non nul).

Exercice 3

Même chose pour le maximum annuel des températures minimales.

Exercice 4

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

Exercice 5

1) Programmez le calcul du minimum des températures minimales.

2) Programmez les calculs de la moyenne annuelle, du maximum annuel et du minimum annuel des températures maximales.

Exercice 6

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

— de gelée (température minimale négative) ;

— de forte gelée (température minimale inférieure ou égale à −5 °C) ;

— sans dégel (températures maximale et minimale négatives) ;

— 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 ensoleillement (ensoleillement non nul) ;

— avec précipitations (précipitations non nulles).

L'avant-dernier calcul est l'objet de l'exercice 2. Programmez les autres.

Exercice 7

Faites afficher un graphique, développé verticalement, sur lequel figurent les températures maximales et minimales. Prévoyez :

— le dessin de l'axe « 0 °C » ;

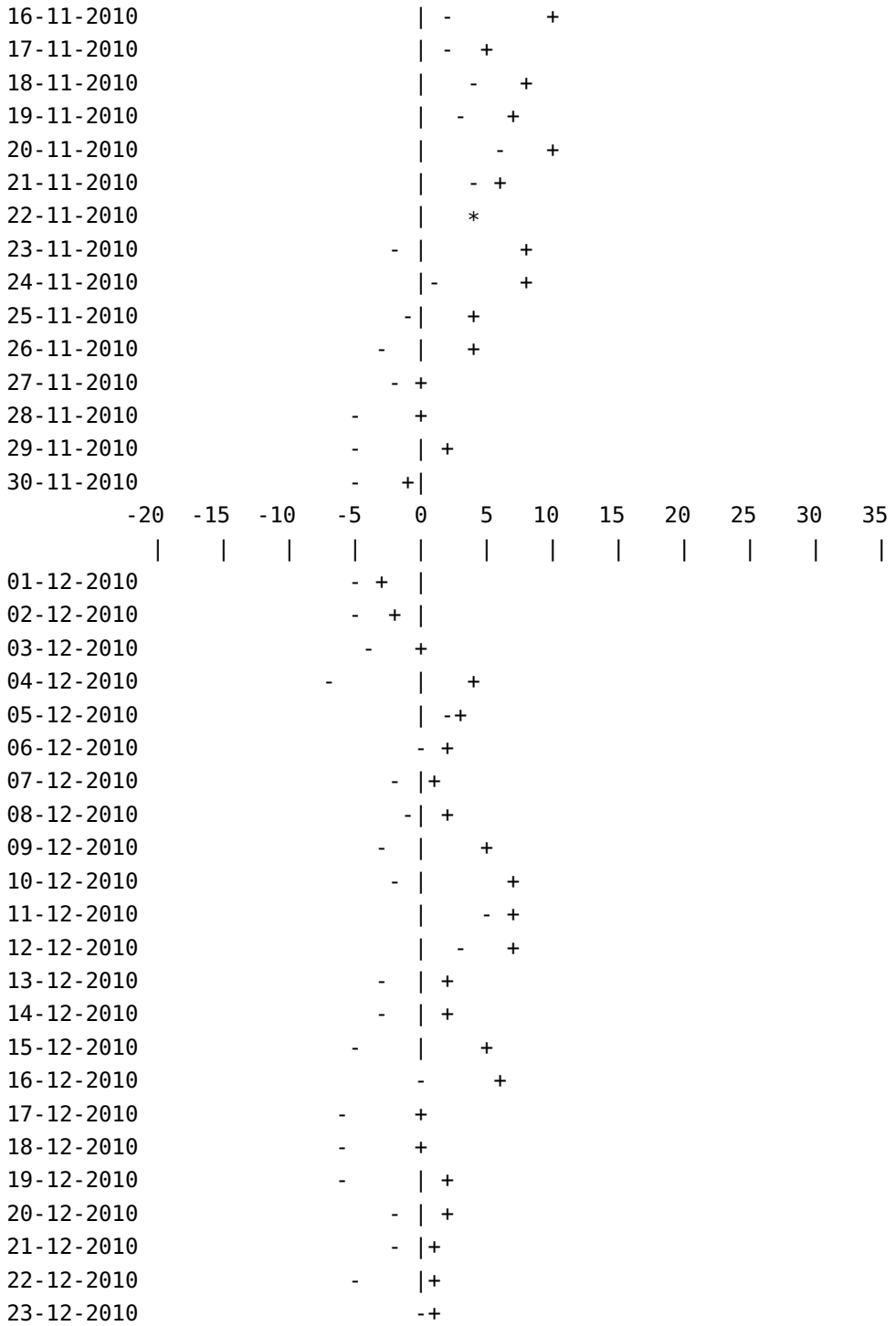
— une symbolique différenciée pour les températures maximales et minimales ;

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

— la possible confusion des affichages des températures maximales et minimales, avec une symbolique différente des précédentes ;

— le fait que, pour un jour donné, la température maximale puisse être strictement inférieure à la température minimale.

À titre d'exemple, figure ci-dessous un extrait d'un graphique obtenu à partir des données de l'année 2010. Notez : l'affichage de températures sur l'axe les 16-12-2010 et 17-12-2010 ; la confusion des affichages des températures maximales et minimales le 22-11-2010.



Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 6

Relevés météorologiques (2) et fichiers binaires homogènes

Objectifs : maitrise des fichiers homogènes ; expression des invariants de boucle liés à la lecture et à l'écriture.
Prérequis : fiche n° 5 de TP ; notion de flot binaire ; fonctions `fread` et `fwrite`.
Travail minimum : exercices 1 et 2.

Le but est ici de permettre travailler à terme sur des fichiers binaires homogènes de relevés météorologiques. Il faut pour cela commencer par obtenir les versions binaires homogènes des fichiers de relevés météorologiques fournis au format CSV.

Exercice 1

1) Écrivez un programme de production de fichiers binaires homogènes de relevés météorologiques à partir de fichiers de relevés météorologiques au format CSV. Le nom des fichiers cible sera obtenu en remplaçant le suffixe « `.csv` » du nom des fichiers source par « `.bin` ». Pour le calcul des noms des fichiers cible, il pourra être fait appel aux tableaux de longueur variable et aux fonctions `memcpy`, `memmove`, `strcpy`, `strncpy`, `strcat`, `strncat` ou `strlen` de l'en-tête standard `<string.h>`. Votre programme devra avoir une syntaxe du même genre que celle des programmes précédents :

```
$ ./meteocsv_to_bin --help
Utilisation : ./meteocsv_to_bin [FICHIER]...
Convertit les fichiers météorologiques FICHIERs du format CSV au format binaire.
```

Suggestion : adaptez le programme fourni à l'occasion de la fiche précédente ou celui que vous avez obtenu à l'issue de l'exercice 4 de la fiche n° 5.

2) Produisez les fichiers binaires associés aux fichiers CSV fournis. Renseignez-vous sur leurs tailles exactes [11]. Remarquez qu'elles sont le produit d'une certaine constante par 365 ou 366.

Exercice 2

Adaptez au cas des fichiers binaires homogènes l'un quelconque des programmes de calcul du total des précipitations, de la moyenne annuelle des températures minimales, du nombre de jours d'ensoleillement ou du maximum annuel des températures minimales fournis ou développés à l'occasion de la fiche n° 5. Testez.

Exercice 3

L'objectif est ici de développer des programmes de calcul soit de sommes, soit de moyennes, soit de maximums, soit de minimums, de l'une quelconque des variables *Tx*, *Tn*, *Rr* ou *W*. Le nom de la variable devra être le premier paramètre de la ligne de commande ; les autres paramètres seront les noms des fichiers sur lesquels le calcul doit être effectué.

Pour fixer les idées, voici l'aide affichée par un programme de calcul de sommes puis la trace de l'une des ses exécutions possibles :

```
$ ./meteobin_sum --help
Utilisation : ./meteobin_sum VAR [FICHIER]...
Affiche la somme de la variable VAR de chacun des fichiers météorologiques au
format binaire FICHIERs.
```

11. Par exemple à l'aide de la commande `ls -l` sous Linux.

```
Les valeurs possibles de VAR sont les suivantes :
  Tx      température maximale
  Tn      température minimale
  Rr      précipitations
  W       ensoleillement
$ ./meteobin_sum Rr ../boos/*.bin
 698.6 mm ../boos/boos2005.bin
 777.1 mm ../boos/boos2006.bin
 950.2 mm ../boos/boos2007.bin
 834.0 mm ../boos/boos2008.bin
 773.4 mm ../boos/boos2009.bin
 810.0 mm ../boos/boos2010.bin
 819.1 mm ../boos/boos2011.bin
 933.9 mm ../boos/boos2012.bin
 765.5 mm ../boos/boos2013.bin
 899.4 mm ../boos/boos2014.bin
 657.0 mm ../boos/boos2015.bin
 782.5 mm ../boos/boos2016.bin
 788.9 mm ../boos/boos2017.bin
```

Afin de ne pas avoir à décliner la fonction de calcul (soit de somme, soit de moyenne, soit de maximum, soit de minimum) autant de fois qu'il y a de variables :

— introduisez un type énuméré décrivant les quatre variables ;

— redéfinissez le type `report` en mémorisant cette fois les quatre variables non plus individuellement mais regroupées à l'aide d'un tableau d'indices le type énuméré précédemment introduit ;

— si le nouveau type `report` ne correspondait pas à l'ancien (tailles ou alignements différents), donnez une nouvelle version du programme de conversion que celle développée à l'occasion de l'exercice 1 et convertissez à nouveau les fichiers ;

— donnez une nouvelle version à votre programme de calcul développé à l'occasion de l'exercice 2 en introduisant un paramètre supplémentaire du type énuméré défini plus haut à la fonction de calcul et en associant le paramètre « variable » fourni par l'utilisateur à l'une des valeurs du type énuméré.

Testez.

Exercice 4

Allez plus loin encore avec deux paramètres pour guider le calcul attendu. Pour fixer les idées :

```
$ ./meteobin_calc --help
Utilisation : ./meteobin_calc CALC SPEC [FICHIER]...
Affiche le résultat du calcul CALC relatif à SPEC appliqué à chacun des fichiers
météorologiques au format binaire FICHIERs.

Les valeurs possibles de CALC sont les suivantes :
  sum     somme
  avg     moyenne
  max     maximum
  min     minimum
  ndays   nombre de jours
```

```
Si CALC vaut « ndays », les valeurs possibles de SPEC sont les suivantes :
  W+      avec ensoleillement (W > 0 h)
  Rr+     avec précipitations (Rr > 0 mm)
  Tn-     de gelée (Tn <= 0 °C)
  Tn--    de forte gelée (Tn <= -5 °C)
  Txn-    sans dégel (Tx <= 0 °C et Tn <= 0 °C)
  Tx+     de chaleur (Tx >= 25 °C)
  Tx++    de forte chaleur (Tx >= 30 °C)
  Txn+    de canicule (Tx >= 30 °C et Tn >= 20 °C)

Sinon, les valeurs possibles de SPEC sont les suivantes :
  Tx      température maximale
  Tn      température minimale
  Rr      précipitations
  W       ensoleillement
$ ./meteobin_calc sum Rr ../boos/*.bin
 698.6 mm ../boos/boos2005.bin
 777.1 mm ../boos/boos2006.bin
 950.2 mm ../boos/boos2007.bin
 834.0 mm ../boos/boos2008.bin
 773.4 mm ../boos/boos2009.bin
 810.0 mm ../boos/boos2010.bin
 819.1 mm ../boos/boos2011.bin
 933.9 mm ../boos/boos2012.bin
 765.5 mm ../boos/boos2013.bin
 899.4 mm ../boos/boos2014.bin
 657.0 mm ../boos/boos2015.bin
 782.5 mm ../boos/boos2016.bin
 788.9 mm ../boos/boos2017.bin
$ ./meteobin_calc avg Tn ../boos/*.bin
   7.1 °C ../boos/boos2005.bin
   7.4 °C ../boos/boos2006.bin
   7.2 °C ../boos/boos2007.bin
   6.7 °C ../boos/boos2008.bin
   6.6 °C ../boos/boos2009.bin
   5.7 °C ../boos/boos2010.bin
   7.5 °C ../boos/boos2011.bin
   6.7 °C ../boos/boos2012.bin
   6.6 °C ../boos/boos2013.bin
   7.8 °C ../boos/boos2014.bin
   7.2 °C ../boos/boos2015.bin
   7.0 °C ../boos/boos2016.bin
   7.3 °C ../boos/boos2017.bin
$ ./meteobin_calc ndays Tn- ../boos/*.bin
 54 jours ../boos/boos2005.bin
 54 jours ../boos/boos2006.bin
 29 jours ../boos/boos2007.bin
 46 jours ../boos/boos2008.bin
```

```
48 jours ../boos/boos2009.bin
77 jours ../boos/boos2010.bin
23 jours ../boos/boos2011.bin
41 jours ../boos/boos2012.bin
59 jours ../boos/boos2013.bin
14 jours ../boos/boos2014.bin
28 jours ../boos/boos2015.bin
36 jours ../boos/boos2016.bin
39 jours ../boos/boos2017.bin
```

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 7

Outils pour fichiers texte

Objectifs : maitrise des fichiers texte.
Prérequis : notion de fichier texte ; fonctions `fgetc`, `fputc`, `fscanf`, `fseek` ; fonctions passées en paramètre.
Travail minimum : les exercices 1 à 3.

Copiez dans votre dossier dévolu à la fiche de TP courante le contenu du dossier `algo1_src/tp/7`. Les fichiers source qui figurent dans ce dossier demandent à être complétés.

Exercice 1

Complétez le fichier `textnl.c` selon les indications qui y figurent (cet exercice doit être vu comme une simple adaptation de la solution à l'exercice 5.4 qui figure sur le support de cours et d'exercices).

Testez. Comparez les résultats avec ce que propose la commande Linux `nl -b a`.

Exercice 2

Complétez le fichier `textwc.c` selon les indications qui y figurent (très simple dès lors que l'on a recours à l'*optional assignment-suppressing character* `*` placé *just after* le caractère `%` dans les chaines de format des fonctions de la famille `...scanf`).

Testez. Comparez les résultats avec ce que propose la commande Linux `wc`.

Exercice 3

Complétez le fichier `textcpbleep_alphaordigit.c` selon les indications qui y figurent (cet exercice doit être vu comme une adaptation de la solution à l'exercice 6.2 qui figure sur le support de cours et d'exercices).

Testez par exemple à partir du fichier `affaires.txt`.

Exercice 4

Complétez le fichier `textbleep_alphaordigit.c` selon les indications qui y figurent (cet exercice doit être vu comme une adaptation des solutions aux exercices 5.6 et 6.2 qui figurent sur le support de cours et d'exercices). Suggestion : définissez et utilisez la fonction

```
//  textbleep_char : lit un caractère depuis le flot contrôlé par l'objet
//    pointé par stream. En cas de succès et si ce caractère satisfait la
//    condition charcond, lui substitue le caractère bleep. Renvoie zéro en cas
//    de succès, une valeur non nulle en cas d'échec
int textbleep_char(FILE *stream, int bleep, int (*charcond)(int));
```

Testez par exemple à partir d'une copie du fichier `affaires.txt`.

Exercice 5

Faites une copie du source précédent nommée `textbleep.c`. Étendez le nouveau source de telle sorte que le paramètre `CLASS` puisse être n'importe lequel des suffixes `alnum`, `alpha`, `blank`... correspondant aux fonctions de test d'appartenance à une catégorie de caractères de l'en-tête standard `<ctype.h>`. Ne recourez pas à des copiés-collés pour obtenir une cascade d'alternatives : passez par un tableau dont les composants sont des structures à deux champs, le premier, l'étiquette de la catégorie, le deuxième, le pointeur vers la fonction associée, et une boucle.

Testez.

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 8

Suite de Collatz et fichiers binaires homogènes

Objectifs : compilation séparée et utilisation de la commande `make`.

Prérequis : notions de compilation séparée et d'utilisation de la commande `make` ; maitrise des fichiers homogènes.

Travail minimum : exercices 1 à 5.

En plus des objectifs annoncés, il s'agit de produire un petit jeu sur la suite de Collatz [12], ou, pour être plus exact, sur un préfixe de celle-ci à partir d'un germe donné. Le temps du jeu, ce préfixe est stocké dans un fichier binaire homogène sur un type entier non signé fixé. Une grande partie du jeu a déjà été écrite : il manque simplement l'implantation de quatre fonctions, dont l'une est d'une importance cruciale puisque le jeu ne peut démarrer sans qu'elle ait été exécutée et que son exécution ait été couronnée de succès.

C'est au niveau de l'implantation de ces quatre fonctions que vous devez intervenir. La trace d'un jeu d'essai est tout d'abord présentée afin de faire comprendre le but à atteindre.

L'exécutable associé au jeu est nommé `cgame`. Comme indiqué par l'aide, deux paramètres sont attendus :

```
$ ./cgame --help
Usage: ./cgame SEED LENGTH
Small game on the prefix of length LENGTH of the Collatz sequence of seed SEED.

SEED range: 1 - 4294967295
LENGTH range: 0 - 1000000

  --help  display this help and exit

$
```

Les termes de la suite, dont son germe, sont codés sur un type entier non signé. La borne maximale de la plage affichée pour le germe en dépend. La longueur maximale du préfixe est volontairement limitée : il s'agit de ne pas saturer l'espace disque.

Lançons le jeu avec des valeurs raisonnables :

```
$ ./cgame 3 10
Size of natural type: 4 bytes.
Maximum value of natural type: 4294967295.
Seed: 3.
Length: 10.
Commands:
  h         print this help
  q         quit this game
  p         print the sequence <C(seed, n)>
  t INDEX   print the term C(seed, INDEX)
```

12. La suite de Collatz est définie dans l'exercice 2.7 du support de cours et d'exercices

```
  i VALUE   print the smallest index n such that C(seed, n) = VALUE

?
```

Le jeu commence par afficher une aide qui fournit quelques informations et indique les commandes disponibles. Le symbole « `?` » qui suit ici l'affichage de l'aide est l'invite de l'interprète de commande du jeu. Mais si le jeu a bien voulu afficher l'aide, c'est qu'il a réussi à générer le fichier `tmp.bin` qui stocke le préfixe de la suite de Collatz. S'il n'y était pas parvenu, voici ce qu'il aurait affiché :

```
$ ./cgame 3 10
Creation of file 'tmp.bin' aborted.
Try './cgame --help'.
$
```

Supposons le fichier `tmp.bin` créé. Le jeu propose d'obtenir des informations sur les termes du préfixe :

— affichage de tous les termes, commande `p` ;
— affichage d'un seul terme à partir de son indice dans la suite, commande `t` ;
— recherche d'une valeur dans le préfixe et, si cette valeur est égale à l'un des termes, affichage le plus petit indice pour lequel l'égalité est détectée, commande `i`.

Effectuons quelques tests puis quittons le jeu :

```
? p
  0 3
  1 10
  2 5
  3 16
  4 8
  5 4
  6 2
  7 1
  8 4
  9 2
? t2 t0 t1 t6 t2017
= 5
= 3
= 10
= 2
*** An error occured while reading the file or index out of range
? i5 i3 i10 i1 i7
= 2
= 0
= 1
= 7
*** An error occured while reading the file or term not found
? q

May the Cforce be with you!

$
```

Récupérez le dossier `algo1_src/tp/8/`. Cinq fichiers figurent dans ce dossier :

— `main.c` : fichier contenant la fonction principale et réalisant l'interface du jeu ;

— `nat.h` : fichier en-tête définissant le nom de type `nat` codant les entiers non signés ainsi que les macroconstantes `NAT_MAX`, valeur maximale du type, `NAT_PRI` et `NAT_SCN`, spécifications de format à utiliser pour les fonctions des familles `printf` et `scanf` ;

— `fnat.h` : fichier en-tête contenant les déclarations spécifications des quatre fonctions évoquées plus haut sur les fichiers binaires homogènes de `nat` ;

— `fnat.c` : fichier devant contenir au final les définitions complètes des quatre fonctions déclarées dans le fichier `fnat.h` ;

— `makefile` : fichier permettant de produire à l'aide de la commande `make` les fichiers objets et l'exécutable de nom `cgame` à partir des fichiers précédents.

Pour travailler correctement sous Geany et donc bénéficier d'un environnement de développement correct :

— éditez les trois fichiers `nat.h`, `fnat.h` et `fnat.c`, le premier pour obtenir que le nom de type `nat` bénéficie de la coloration syntaxique propre aux types non standard, le deuxième pour avoir sous les yeux des spécifications des quatre fonctions, le troisième parce qu'il s'agit du fichier à compléter ;

— ne cherchez pas à « compiler » le fichier `fnat.c`, que ce soit en allant chercher la commande dans le menu, en utilisant son raccourci clavier ou en cliquant sur un bouton ;

— que ce soit pour vérifier la syntaxe ou construire l'exécutable, utilisez la commande « Make » du menu ou son raccourci clavier, Maj+F9.

Précisions :

— si vous construisez l'exécutable sans rien avoir modifié aux fichiers fournis puis que vous lancez le jeu avec les deux paramètres requis, vous obtiendrez l'affichage indiquant l'impossibilité de création du fichier `tmp.bin` tel que montré plus haut ;

— vous ne devez modifier en aucune façon les fichiers `main.c`, `nat.h` et `fnat.h` ;

— vous n'êtes autorisé à modifier le fichier `make` qu'à l'exercice 5 [13] ;

— au moment de rendre votre travail, exécutez en ligne de commande et dans le dossier dévolu à la fiche de TP courante la commande `make tar`. C'est l'archive produite que vous déposerez sur la plateforme UniversiTICE.

Exercice 1

Implantez la fonction `fnat_open_collatz` sans vous soucier des éventuels débordements sur le type de nom `nat` utilisé pour coder les termes de la suite de Collatz. Construisez. Testez (en ligne de commande) : observez dans une fenêtre ad hoc la création du fichier `tmp.bin` au lancement du jeu puis sa suppression une fois quitté le jeu.

Exercice 2

Implantez la fonction `fnat_print`. Construisez. Testez.

Exercice 3

Implantez la fonction `fnat_index_to_value`. Construisez. Testez.

Exercice 4

Implantez la fonction `fnat_value_to_index`. Construisez. Testez.

Exercice 5

Éditez le fichier `makefile`. Supprimez l'option `-Wno-unused-parameter`. Construisez.

13. Sauf à ce que vous travailliez sous Windows auquel cas il vous sera nécessaire de modifier le contenu de la variable `executable` en l'initialisant à `cgame.exe`.

Exercice 6

Revenez sur les problèmes de débordement laissés de côté à l'exercice 1 : si un débordement survient, la fonction doit dorénavant renvoyer une erreur.

Exercice 7

Testez avec d'autres types entiers non signés : ajoutez pour cela l'option `-DNATWIDTH=8`, `-DNATWIDTH=16` ou `-DNATWIDTH=64` à la variable `CFLAGS` du fichier `makefile` (pour que l'option soit prise en compte, enchainez les commandes `make clean` et `make`). Apportez les éventuelles améliorations nécessaires.

MÀJ 30-03
`NATSIZE` → `NATWIDTH` (V. F.)

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 9

Polynômes

Objectifs : maitrise d'une implantation des polynômes ; arithmétique des pointeurs.
Prérequis : notions basiques sur les polynômes ; méthode de Horner ; exercice 1.
Travail minimum : exercices 2 à 4.

En plus des objectifs annoncés, il s'agit de produire un petit jeu sur les polynômes à coefficients du type `double`. Une grande partie du jeu a déjà été écrite : il manque simplement l'implantation de quelques fonctions d'importance variable.

C'est au niveau de l'implantation de ces fonctions que vous devez intervenir.

L'exécutable associé au jeu est nommé `pgame`. Comme indiqué par l'aide, aucun paramètre n'est attendu :

```
$ ./pgame --help
Usage: ./pgame
Small game on polynomes.

  --help  display this help and exit

$
```

Lançons le jeu :

```
$ ./pgame
Polynome memories are letters from A to H.
In the following list of commands, R and S mean any of but different memories.

Commands:
  h       print this help
  q       quit this game
  a R S   advance R by S
  c R S   initialize R by copying S
  d R     replace R by its derivative
  e R     evaluate R for a list of values ended by a new-line
  m R S   multiply R by S
  n R     initialize R to null
  o R     replace R by its opposite
  p R     print R
  s R     initialize R with a list of values ended by a new-line

?
```

Le jeu commence par afficher une aide qui fournit quelques informations et indique les commandes disponibles. Initialement, seules les commandes `h`, `q`, `c`, `p` et `s` sont fonctionnelles. Voici une invocation à ces quatre dernières :

```
? sA -1 0 1 2
? sB 0 3 -1
? cCA pA pB pC
[A] = 2 X^3 + X^2 - 1
[B] = -X^2 + 3 X
[C] = 2 X^3 + X^2 - 1
? q

May the Pforce be with you!

$
```

Récupérez le dossier `algo1_src/tp/9/`. Six fichiers figurent dans ce dossier :

— `main.c` : fichier contenant la fonction principale et réalisant l'interface du jeu ;

— `double_base.h` et `double_base.c` : petit module permettant la lecture et l'affichage de valeur du type **`double`** ainsi qu'un affichage spécialisé des monômes ;

— `double_poly.h` : fichier en-tête qui contient : 1) la définition des deux macroconstantes `POLY_DEGREE_MAX` et `POLY_DEGREE_INF`, la première fixant le degré maximum des polynômes, la seconde, le degré du polynôme nul ; 2) la définition du (nom de) type `poly`, structure permettant la mémorisation des polynômes de degré inférieur ou égal à `POLY_DEGREE_MAX` et du polynôme nul ; 3) les déclarations de neuf fonctions documentées sur objets du type `poly` ;

— `double_poly.c` : partie implantation du fichier précédent ;

— `makefile` : fichier permettant de produire à l'aide de la commande `make` les fichiers objets et l'exécutable de nom `pgame` à partir des fichiers précédents.

C'est le fichier `double_poly.c` que vous devez compléter et modifier : seules quatre des neuf fonctions sont complètement implantées.

Exercice 1

Prenez connaissance du contenu des fichiers `double_poly.h` et `double_poly.c`. Comprenez de quoi il retourne.

Exercice 2

Donnez corps à la fonction `poly_opposite` en ayant recours à la notation usuelle avec indices et crochets des composants des tableaux. N'omettez ni les invariants de boucle ni les quantités de contrôle. Construisez. Testez.

Exercice 3

Même chose pour la fonction `poly_evaluate`.

Exercice 4

Donnez une version alternative à la fonction `poly_evaluate` qui utilise l'arithmétique des pointeurs : faites une copie de la version précédente ; mettez l'originale en commentaire (balises `/*` et `*/` ou **`#if`** `0` *fin-de-ligne* et **`#endif`** *fin-de-ligne*) ; travaillez avec un pointeur de **`double`**.

Exercice 5

Enchainez avec les fonctions `poly_derivative`, `poly_add` et `poly_multiply`. Après les avoir testées, supprimez l'option `-Wno-unused-parameter` qui figure dans le fichier `makefile`.

Algorithmique 1 : méthodologie de la programmation impérative

Fiche de TP n° 10

Relevés météorologiques (3) et algorithmes de recherche

Objectifs : maitrise du passage des fonctions en paramètre ; maitrise de l'arithmétique des pointeurs.
Prérequis : fiche n° 6 de TP ; pointeurs et fonctions ; pointeurs et tableaux ; type pointeur générique `void *` ; polymorphisme ; algorithmes de recherche présentés en cours.
Travail minimum : l'un des exercices 1, 2, 3 ou 5.

Récupérez le dossier `algo1_src/tp/10/`.

Produisez l'exécutable du fichier source `meteobin_misc_idx.c` qui figure dans ce dossier (sous Geany, commande « Construire » du menu ou son raccourci clavier, F9 ; le fichier `makefile` ne sert qu'à l'exercice 5, `make`, et lors du rendu du travail effectué, `make tar`). Exécutez-le en ligne de commande sur un ou plusieurs fichiers météorologiques au format binaire (produits de la fiche n° 6 de TP).

Les résultats affichés sont essentiellement le fruit de l'action des trois fonctions d'identificateurs `report_frostnight_count`, `report_frost_left_search` et `report_mintn_left_search`, dont les spécifications et déclarations figurent dans les premières lignes du fichier source, et les définitions, dans les dernières lignes. Ces définitions sont une adaptation à la fois immédiate et simpliste des algorithmes du cours : utilisation d'indices de tableaux ; aucune réutilisation possible du code.

Votre travail consiste précisément à faire évoluer ces codes vers le polymorphisme : exercices 1, 2 et 3, de plans identiques [14]. Puis à envisager d'autres calculs que ceux du fichier source originel : exercice 4. Ou encore la création et l'utilisation d'un module de fonctions de recherche sur les tableaux : exercice 5.

Exercice 1

Travail sur `report_frostnight_count`.

1) Passez à une version avec variable de boucle de type pointeur sur `report`.

2) En vous basant sur le code obtenu, dérivez-en la définition de la fonction de spécification et prototype :

```
//  report_cond_count : renvoie le nombre de journées qui satisfont la condition
//    cond comptées dans le tableau sur le type report dont l'adresse du premier
//    composant est base et dont la longueur est nmemb
size_t report_cond_count(const report *base, size_t nmemb,
    bool (*cond)(const report *));
```

En conséquence, la définition de la fonction `report_frostnight_count` doit désormais être :

```
size_t report_frostnight_count(const report *base, size_t nmemb) {
  return report_cond_count(base, nmemb, report_frostnight_cond);
}
```

où `report_frostnight_cond` est une (très courte) fonction à définir.

3) Transformez la fonction `report_cond_count` en la fonction de spécification et prototype :

14. Il n'est pas interdit de mener de front les trois exercices : cela permet de bien ancrer chacune des étapes dans trois configurations différentes. Il n'est pas non plus interdit de griller les trois premières étapes de certains de ces exercices. Il n'est pas non plus interdit de passer directement à l'exercice 5. Mais il faut au minimum terminer l'un des quatre exercices mentionnés.

```
//  cond_count : renvoie le nombre de composants d'un tableau qui satisfont une
//    condition. base est l'adresse du premier composant du tableau, nmemb, le
//    nombre de composants du tableau, size, la taille des composants, cond, un
//    pointeur vers la condition
size_t cond_count(const void *base, size_t nmemb, size_t size,
    bool (*cond)(const void *));
```

et, conjointement, la définition de la fonction `report_frostnight_count` en :

```
size_t report_frostnight_count(const report *base, size_t nmemb) {
  return cond_count(base, nmemb, sizeof(report),
      (bool (*)(const void *)) report_frostnight_cond);
}
```

4) Supprimez toute trace de la fonction `report_frostnight_count` : spécification, prototype, définition, et remplacez son appel dans la fonction `main` par un appel direct à `cond_count`.

Exercice 2

Travail sur `report_frost_left_search`.

1) Passez à une version avec variable de boucle de type pointeur sur `report`.

2) En vous basant sur le code obtenu, dérivez-en la définition de la fonction de spécification et prototype :

```
//  report_cond_left_search : recherche la première occurrence d'une journée
//    satisfaisant une condition dans un tableau sur le type report. base est
//    l'adresse du premier composant du tableau est base, nmemb, sa longueur,
//    cond, le pointeur vers la condition. Renvoie l'adresse du composant trouvé
//    en cas de succès, NULL en cas d'échec
report *report_cond_left_search(const report *base, size_t nmemb,
    bool (*cond)(const report *));
```

En conséquence, la définition de la fonction `report_frost_left_search` doit désormais être :

```
report *report_frost_left_search(const report *base, size_t nmemb) {
  return report_cond_left_search(base, nmemb, report_frost_cond);
}
```

où `report_frost_cond` est une (très courte) fonction à définir.

3) Transformez la fonction `report_cond_left_search` en la fonction de spécification et prototype :

```
//  cond_left_search : recherche la première occurrence d'un composant d'un
//    tableau qui satisfait une condition. base est l'adresse du premier
//    composant du tableau, nmemb, le nombre de composants du tableau, size, la
//    taille des composants, cond, un pointeur vers la condition. Renvoie
//    l'adresse du composant trouvé en cas de succès, NULL en cas d'échec
void *cond_left_search(const void *base, size_t nmemb, size_t size,
    bool (*cond)(const void *));
```

et, conjointement, la définition de la fonction `report_frost_left_search` en :

```
report *report_frost_left_search(const report *base, size_t nmemb) {
  return cond_left_search(base, nmemb, sizeof(report),
      (bool (*)(const void *)) report_frost_cond);
}
```

4) Supprimez toute trace de la fonction `report_frost_left_search` : spécification, prototype, définition, et remplacez son appel dans la fonction `main` par un appel direct à `cond_left_search`.

Exercice 3

Travail sur `report_mintn_left_search`.

1) Passez à une version avec variable de boucle de type pointeur sur `report`.

2) En vous basant sur le code obtenu, dérivez-en la définition de la fonction de spécification et prototype :

```
//  report_min_left_search : recherche la première occurrence d'une journée
//    minimum au sens d'une fonction de comparaison dans un tableau sur le type
//    report. base est l'adresse du premier composant du tableau est base,
//    nmemb, sa longueur, cond, le pointeur vers la condition. Renvoie NULL si
//    nmemb vaut zéro, l'adresse du composant trouvé sinon
report *report_min_left_search(const report *base, size_t nmemb,
    int (*compar)(const report *, const report *));
```

En conséquence, la définition de la fonction `report_mintn_left_search` doit désormais être :

```
report *report_mintn_left_search(const report *base, size_t nmemb) {
  return report_min_left_search(base, nmemb, report_tn_compar);
}
```

où `report_tn_compar` est une (très courte) fonction à définir.

3) Transformez la fonction `report_min_left_search` en la fonction de spécification et prototype :

```
//  min_left_search : recherche la première occurrence d'un composant d'un
//    tableau qui est minimum au sens d'une fonction de comparaison. base est
//    l'adresse du premier composant du tableau, nmemb, le nombre de composants
//    du tableau, size, la taille des composants, compar, un pointeur vers la
//    fonction de comparaison. Renvoie NULL si nmemb vaut zéro, l'adresse du
//    composant trouvé sinon
void *min_left_search(const void *base, size_t nmemb, size_t size,
    int (*compar)(const void *, const void *));
```

et, conjointement, la définition de la fonction `report_mintn_left_search` en :

```
report *report_mintn_left_search(const report *base, size_t nmemb) {
  return min_left_search(base, nmemb, sizeof(report),
      (int (*)(const void *, const void *)) report_tn_compar);
}
```

4) Supprimez toute trace de la fonction `report_mintn_left_search` : spécification, prototype, définition, et remplacez son appel dans la fonction `main` par un appel direct à `min_left_search`.

Exercice 4

En vous reportant à l'exercice 6 de la fiche n° 5 de TP pour la définition des termes météorologiques :

1) En plus du nombre de jours avec gelée, faites afficher le nombre de jours de chaleur.

2) En plus de la première occurrence d'une journée sans dégel, faites afficher la première occurrence d'une journée de chaleur.

3) En plus de la première occurrence d'une journée de température minimale minimum, faites afficher la première occurrence d'une journée de température maximale maximum, sans en passer par la définition de la fonction polymorphe `max_left_search`.

À titre indicatif, voici ce que vous pourriez obtenir pour les années 2010 :

```
$ ./meteobin_misc ../boos/*201?.bin
2010 n. jours Tn <= 0    77
2010 n. jours Tx >= 25   36
2010 prem. Tx/Tn <= 0    04-01 Tx/Tn = -0.6/-8.2
2010 prem. Tx >= 25      29-04 Tx = 26.4
2010 prem. min. Tn       07-01 Tn = -9.6
2010 prem. max. Tx       08-07 Tx = 33.2
2011 n. jours Tn <= 0    23
2011 n. jours Tx >= 25   25
2011 prem. Tx/Tn <= 0    29-01 Tx/Tn = -0.2/-2.8
2011 prem. Tx >= 25      23-04 Tx = 25.5
2011 prem. min. Tn       04-04 Tn = -7.1
2011 prem. max. Tx       27-06 Tx = 34.2
2012 n. jours Tn <= 0    41
2012 n. jours Tx >= 25   25
2012 prem. Tx/Tn <= 0    01-02 Tx/Tn = -0.7/-5.8
2012 prem. Tx >= 25      24-05 Tx = 27.1
2012 prem. min. Tn       12-02 Tn = -12.1
2012 prem. max. Tx       18-08 Tx = 33.1
2013 n. jours Tn <= 0    59
2013 n. jours Tx >= 25   51
2013 prem. Tx/Tn <= 0    18-01 Tx/Tn = -2.6/-6.6
2013 prem. Tx >= 25      14-04 Tx = 25.0
2013 prem. min. Tn       13-03 Tn = -7.0
2013 prem. max. Tx       21-07 Tx = 32.9
2014 n. jours Tn <= 0    14
2014 n. jours Tx >= 25   18
2014 prem. Tx/Tn <= 0    ***
2014 prem. Tx >= 25      06-06 Tx = 25.7
2014 prem. min. Tn       29-12 Tn = -5.5
2014 prem. max. Tx       18-07 Tx = 33.0
2015 n. jours Tn <= 0    28
2015 n. jours Tx >= 25   36
2015 prem. Tx/Tn <= 0    ***
2015 prem. Tx >= 25      15-04 Tx = 26.4
2015 prem. min. Tn       20-01 Tn = -4.3
2015 prem. max. Tx       01-07 Tx = 37.9
2016 n. jours Tn <= 0    36
2016 n. jours Tx >= 25   34
2016 prem. Tx/Tn <= 0    29-12 Tx/Tn = -0.9/-2.5
2016 prem. Tx >= 25      06-06 Tx = 25.1
2016 prem. min. Tn       20-01 Tn = -7.1
2016 prem. max. Tx       24-08 Tx = 35.9
2017 n. jours Tn <= 0    39
2017 n. jours Tx >= 25   41
2017 prem. Tx/Tn <= 0    02-12 Tx/Tn = -0.2/-2.5
2017 prem. Tx >= 25      16-05 Tx = 27.7
2017 prem. min. Tn       21-01 Tn = -7.3
2017 prem. max. Tx       21-06 Tx = 36.0
```

Exercice 5

1) Faites une copie de votre fichier de travail (celui qui contient la fonction `main`) ; nommez-la `meteobin_misc.c`.

2) Créez votre propre module de fonctions sur les tableaux : `array`.

a) Créez le fichier `array.h`. Déplacez-y les spécifications et prototypes des fonctions polymorphes `cond_count`, `cond_left_search` et `min_left_search`. Ajoutez pour la forme le mot-clé `extern` devant chacun des prototypes. Insérez ensuite les directives

```
#ifndef ARRAY__H
#define ARRAY__H

#include <stddef.h>
#include <stdbool.h>
```

au début du fichier et

```
#endif
```

à la fin du fichier.

b) Créez le fichier `array.c`. Déplacez-y les définitions des trois fonctions. Insérez ensuite la directive

```
#include "array.h"
```

au début du fichier.

3) Dans le fichier `meteobin_misc.c`, insérez la directive

```
#include "array.h"
```

à la suite des directives d'inclusion des en-têtes standard.

4) Construisez à l'aide du fichier `makefile`.

Algorithmique 1 : méthodologie de la programmation impérative
Fiche de TP n° 11

Relevés météorologiques (4) et algorithmes de tri

Objectifs : maitrise du passage des fonctions en paramètre ; maitrise de l'arithmétique des pointeurs.
Prérequis : fiche n° 10 de TP ; pointeurs et fonctions ; pointeurs et tableaux ; type pointeur générique `void *` ; algorithmes de tri présentés en cours.
Travail minimum : exercices 1 à 3.

Récupérez le dossier `algo1_src/tp/11/`.

Produisez l'exécutable du fichier source `meteobin_print.c` qui figure dans ce dossier (sous Geany, commande « Construire » du menu ou son raccourci clavier, F9 ; le fichier `makefile` ne sert qu'à l'exercice 3, `make`, et lors du rendu du travail effectué, `make tar`). Exécutez-le en ligne de commande sur un ou plusieurs fichiers météorologiques au format binaire (produits de la fiche n° 6 de TP).

Les relevés annuels sont affichés dans l'ordre chronologique, sans aucune modification de l'ordre dans lequel les relevés journaliers y figurent. Ils devront par la suite être affichés dans des ordre différents.

Exercice 1

Faites appel, à l'endroit indiqué dans le fichier source, à la procédure `qsort` de l'en-tête standard `<stdlib.h>` pour faire afficher les relevés annuels dans l'ordre croissant des *Tx*. Pour ce faire, vous devrez définir une fonction du type

```
int (*)(const report *ptr1, const report *ptr2)
```

qui, conformément au standard, renvoie un entier du type `int` strictement négatif, nul ou strictement positif selon que la mesure *Tx* du relevé pointé par `ptr1` est strictement inférieure, égale ou strictement supérieure à celle du relevé pointé par `ptr2`.

Exercice 2

Modifiez la fonction précédemment définie (ou définissez-en une autre) pour faire en sorte que l'affichage soit dorénavant dans l'ordre croissant des *Tx*, clé primaire, puis dans celui des *Tn*, clé secondaire.

Exercice 3

Implantez l'algorithme du tri par sélection du minimum ou celui du tri par insertion croissante en vous alignant sur le prototype de la procédure `qsort`. Testez en faisant appel à cette implantation en lieu et place de `qsort` dans votre solution à l'exercice 2.

Il vous sera nécessaire d'apporter au préalable une solution au problème de l'échange du contenu de deux zones mémoires de même taille `size` repérées par des pointeurs génériques `void *` (voir exercice 10.5 du support de cours et d'exercices).

Exercice 4

Implantez les deux autres algorithmes de tri du cours selon les mêmes modalités.

Exercice 5

Complétez votre module de fonctions sur les tableaux, `array`, introduit dans l'énoncé de l'exercice 5 de la fiche n° 10, en y ajoutant vos fonctions de tri.