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` » puis cliquez sur « Créer ».

Les enseignant·es pourront refuser de prendre en compte toute archive de format ou de contenu inadéquat, ou tout source non mis en forme par l'utilitaire de remise en forme de codes sources Uncrustify.

Réglages de l'EDI Geany

Avant de lancer Geany, récupérez l'archive `algo1_cfg.zip` sur UniversiTICE, extrayez-en ses composants, ouvrez un terminal dans le dossier `config/` de l'archive décompressée et exécutez le script `config.sh` : « `./config.sh` » en ligne de commande.

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 > 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 pré-compilé (*precompiled header file*) associé d'extension `.gch`[1] ;

1. 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'édition 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 [2] 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.

2. La remise en forme est intéressante mais n'est pas parfaite. De nouvelles versions des fichiers de configuration `uncrustify..._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 C23 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` [3], lit les chaines de caractères d'une longueur maximale à fixer [4], 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
```

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

4. 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`[5] et à la variable standard `errno`[6] 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.

5. Voir Annexe B5 « Les fonctions utilitaires » de l'ouvrage précédemment cité.
6. 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`[7].

```c
#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 19 7 ADD ADD END
UN ET UN, DEUX.
```

la sortie sera :

```
1———⟩20———⟩operand = 20
2———⟩100———⟩operand = 100
3———⟩MUL———⟩operator
4———⟩19———⟩operand = 19
5———⟩7———⟩operand = 7
6———⟩ADD———⟩operator
7———⟩ADD———⟩operator
8———⟩END———⟩operator
9———⟩UN———⟩rejected form
```

MÀJ 29-01
4 15...
5 9... →
4 19...
5 7...
(TL)

7. 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 la seule expression

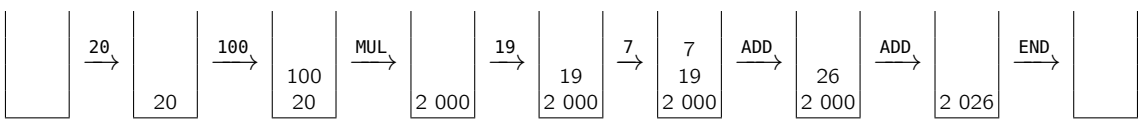
```
20 100 MUL 19 7 ADD ADD END
```

le programme à développer doit ainsi afficher

```
2026
```

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`.