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 ».