Rattrapages C

Code C

27 juin 2022

Version 1

Fabrice BOISSIER <fabrice.boissier@epita.fr>

Mark ANGOUSTURES <mark.angoustures@epita.fr>

Copyright

Table des matières

1 Consignes Générales

Les informations suivantes sont très importantes :

Le non-respect d'une des consignes suivantes entraînera des sanctions pouvant aller jusqu'à la multiplication de la note finale par 0.

Ces consignes sont claires, non-ambiguës, et ont un objectif précis. En outre, elles ne sont pas négociables.

N'hésitez pas à demander si vous ne comprenez pas une des règles.

Consigne Générale 0 : Vous devez lire le sujet.
Consigne Générale 1 : Vous devez respecter les consignes.
Consigne Générale 2 : Vous devez rendre le travail dans les délais prévus.

Consigne Générale 3 : Le travail doit être rendu dans le format décrit à la section Format de Rendu.

Consigne Générale 4 : Le travail rendu ne doit pas contenir de fichiers binaires, temporaires, ou d'erreurs (`*~`, `*.o`, `*.a`, `*.so`, `*#*`, `*core`, `*.log`, `*.exe`, binaires, ...).

Consigne Générale 5 : Dans l'ensemble de ce document, la casse (caractères majuscules et minuscules) est très importante. Vous devez strictement respecter les majuscules et minuscules imposées dans les messages et noms de fichiers du sujet.

Consigne Générale 6 : Dans l'ensemble de ce document, `login` correspond à votre login.

Consigne Générale 7 : Dans l'ensemble de ce document, `nom1-nom2` correspond à la combinaison des deux noms de votre binôme (par exemple pour Fabrice BOISSIER et Mark ANGOUSTURES, cela donnera `boissier-angoustures`).

Consigne Générale 8 : Dans l'ensemble de ce document, le caractère ␣ correspond à une espace (s'il vous est demandé d'afficher ␣␣␣, vous devez afficher trois espaces consécutives).

Consigne Générale 9 : Tout retard, même d'une seconde, entraîne la note non négociable de 0.

Consigne Générale 10 : La triche (échange de code, copie de code ou de texte, ...) entraîne **au mieux** la note non négociable de 0.

Consigne Générale 11 : En cas de problème avec le projet, vous devez contacter le plus tôt possible les responsables du sujet aux adresses mail indiquées.

Conseil : N'attendez pas la dernière minute pour commencer à travailler sur le sujet.

2 Format de Rendu

Responsable(s) du projet :	**Fabrice BOISSIER** `<fabrice.boissier@epita.fr>` **Mark ANGOUSTURES** `<mark.angoustures@epita.fr>`
Balise(s) du projet :	**`[RATT][C]`**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir/Assignment sur Teams
Nom du répertoire :	`login-RATT-C`
Nom de l'archive :	`login-RATT-C.tar.bz2`
Date maximale de rendu :	27/06/2022 12h00
Durée du projet :	2 semaines
Architecture/OS :	Linux - Ubuntu (x86_64)
Langage(s) :	C
Compilateur/Interpréteur :	**`/usr/bin/gcc`**
Options du compilateur/interpréteur :	**`-W -Wall -Werror -std=c99 -pedantic`**

Les fichiers suivants sont requis :

`AUTHORS`	contient le(s) nom(s) et prénom(s) de(s) auteur(s).
`Makefile`	le Makefile principal.
`README`	contient la description du projet et des exercices, ainsi que la façon d'utiliser le projet.

Un fichier **`Makefile`** doit être présent à la racine du dossier, et doit obligatoirement proposer ces règles :

`all`	*[Première règle]* lance la règle `binaries`.
`clean`	supprime tous les fichiers temporaires et ceux créés par le compilateur.
`dist`	crée une archive propre, valide, et répondant aux exigences de rendu.
`distclean`	lance la règle `clean`, puis supprime les binaires et bibliothèques.
`check`	lance l'éventuelle suite de tests.
`binaries`	compile chaque exercice et produit un exécutable

Votre code sera testé automatiquement, vous devez donc scrupuleusement respecter les spécifications pour pouvoir obtenir des points en validant les exercices. Votre code sera testé en générant un exécutable ou des bibliothèques avec les commandes suivantes :

```
./configure
make distclean
./configure
make
```

Suite à cette étape de génération, les exécutables ou bibliothèques doivent être placés à ces endroits :

login-RATT-C/is_even
login-RATT-C/my_calc
login-RATT-C/my_pintree
login-RATT-C/my_transpose

L'arborescence attendue pour le projet est la suivante :

login-RATT-C/
login-RATT-C/AUTHORS
login-RATT-C/README
login-RATT-C/Makefile
login-RATT-C/configure
login-RATT-C/check/
login-RATT-C/src/
login-RATT-C/src/is_even.c
login-RATT-C/src/my_calc.c
login-RATT-C/src/my_pintree.c
login-RATT-C/src/my_transpose.c

Vous ne serez jamais pénalisés pour la présence de makefiles ou de fichiers sources (code et/ou headers) dans les différents dossiers du projet tant que leur existence peut être justifiée (des makefiles vides ou jamais utilisés sont pénalisés, des fichiers sources hors du dossier sources sont pénalisés).
Vous ne serez jamais pénalisés pour la présence de fichiers de différentes natures dans le dossier `check` tant que leur existence peut être justifiée (des fichiers de test jamais utilisés sont pénalisés).

3 Aide Mémoire

Le travail doit être rendu au format ***.tar.bz2***, c'est-à-dire une archive **bz2** compressée avec un outil adapté (voir **`man 1 tar`** et **`man 1 bz2`**).
Tout autre format d'archive (zip, rar, 7zip, gz, gzip, ...) ne sera pas pris en compte, et votre travail ne sera pas corrigé (entraînant la note de 0).

Pour générer une archive *tar* en y mettant les dossiers *folder1* et *folder2*, vous devez taper :

```
tar cvf MyTarball.tar folder1 folder2
```

Pour générer une archive *tar* et la compresser avec GZip, vous devez taper :

```
tar cvzf MyTarball.tar.gz folder1 folder2
```

Pour générer une archive *tar* et la compresser avec BZip2, vous devez taper :

```
tar cvjf MyTarball.tar.bz2 folder1 folder2
```

Pour lister le contenu d'une archive *tar*, vous devez taper :

```
tar tf MyTarball.tar.bz2
```

Pour extraire le contenu d'une archive *tar*, vous devez taper :

```
tar xvf MyTarball.tar.bz2
```

Pour générer des exécutables avec les symboles de debug, vous devez utiliser les flags **`-g -ggdb`** avec le compilateur. N'oubliez pas d'appliquer ces flags sur *l'ensemble* des fichiers sources transformés en fichiers objets, et d'éventuellement utiliser les bibliothèques compilées en mode debug.

```
gcc -g -ggdb -c file1.c file2.c
```

Pour produire des exécutables avec les symboles de debug, il est conseillé de fournir un script **`configure`** prenant en paramètre une option permettant d'ajouter ces flags aux **`CFLAGS`** habituels.

```
./configure
cat Makefile.rules
CFLAGS=-W -Wall -Werror -std=c99 -pedantic
./configure debug
cat Makefile.rules
CFLAGS=-W -Wall -Werror -std=c99 -pedantic -g -ggdb
```

Dans ce sujet précis, vous ferez du code en C et des appels à des scripts shell qui afficheront les résultats dans le terminal (donc des flux de sortie qui pourront être redirigés vers un fichier texte).

4 Exercice 1 - is even

Nom du(es) fichier(s) :	**is_even.c**
Répertoire :	**login-RATT-C/src/is_even.c**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640

Objectif : Le but de l'exercice est d'afficher si un nombre est pair ou impair.

Vous devez écrire un programme qui prendra un paramètre, et affichera selon ce paramètre s'il est pair ou impair.
Si le nombre est pair, vous devez écrire le mot **even** dans le terminal et renvoyer 0.
Si le nombre est impair, vous devez écrire le mot **odd** dans le terminal et renvoyer 1.

```
$ ./is_even 3
odd
$ echo $?
1
$ ./is_even 6
even
$ echo $?
0
$ ./is_even 0
even
$ echo $?
0
```

Cas général

Si plusieurs ou aucun paramètre ne sont donnés, vous devez afficher le message d'erreur suivant, et renvoyer 255.

Usage: ./is_even number

```
$ ./is_even 0 4
Usage: ./is_even number
$ echo $?
255
$ ./is_even
Usage: ./is_even number
$ echo $?
255
```

Cas d'erreur 1

Si le paramètre n'est pas un nombre, vous devez écrire le mot **error** dans le terminal et renvoyer 254.

```
$ ./is_even abcd
error
$ echo $?
254
$ ./is_even 42a
error
$ echo $?
254
```

Cas d'erreur 2

Si plusieurs paramètres non numériques sont donnés, c'est l'erreur sur le nombre de paramètres qui primera.

```
$ ./is_even 42 42a
Usage: ./is_even number
$ echo $?
255
$ ./is_even 42a 42
Usage: ./is_even number
$ echo $?
255
```

Cas d'erreur 2

5 Exercice 2 - Calculatrice

Nom du(es) fichier(s) :	**my_calc.c**
Répertoire :	**login-RATT-C/src/my_calc.c**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640

Objectif : Le but de l'exercice est de créer une mini calculatrice en C.

Vous devez écrire un programme qui prendra trois paramètres (deux nombres, puis l'opérateur), et affichera le résultat de l'opération désignée.
Vous devez implémenter les 5 opérations suivantes : l'addition (symbole **+**), la soustraction (symbole **-**), la multiplication (lettre **x**), la division (symbole **/**), et le reste de la division euclidienne (symbole **%**).
À la fin du calcul, votre programme doit renvoyer 0.

```
$ ./my_calc 1 3 +
4
$ echo $?
0
$ ./my_calc 144 362 -
-218
$ echo $?
0
$ ./my_calc 6 7 x
42
$ echo $?
0
$ ./my_calc 12 3 /
4
$ echo $?
0
$ ./my_calc 69 2 %
1
$ echo $?
0
```

Cas général

Si des paramètres manquent, vous devez écrire le message suivant, et renvoyer 1.

```
Not␣enough␣parameters.
Usage:␣./my_calc␣number␣number␣operator
Operator␣might␣be␣:␣+␣-␣x␣/␣%
```

```
$ ./my_calc
Not enough parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
1
$ ./my_calc 0 4
Not enough parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
1
```

Cas d'erreur 1 : pas assez de paramètres

Si des paramètres sont en trop, vous devez écrire le message suivant, et renvoyer 2.

Too␣much␣parameters.
Usage:␣./my_calc␣number␣number␣operator
Operator␣might␣be␣:␣+␣-␣x␣/␣%

```
$ ./my_calc 0 4 x 45
Too much parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
2
```

Cas d'erreur 2 : trop de paramètres

Si l'opérateur n'est pas placé après les nombres/en dernier/à la troisième position, vous devez indiquer le message suivant, et renvoyer 3.

Wrong␣parameters.
Usage:␣./my_calc␣number␣number␣operator
Operator␣might␣be␣:␣+␣-␣x␣/␣%

```
$ ./my_calc x x x
Wrong parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
3
$ ./my_calc 1 2 3
Wrong parameters.
Usage: ./my_calc number number operator
```

```
Operator might be : + - x / %
$ echo $?
3
$ ./my_calc 1 + 2
Wrong parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
3
```

Cas d'erreur 3 : pas les bons paramètres

Si le deuxième paramètre donné à la division est 0, vous devez renvoyer 4 et écrire le message d'erreur suivant.

`Division␣by␣0␣is␣forbidden.`

```
$ ./my_calc 1 0 /
Division by 0 is forbidden.
$ echo $?
4
```

Cas d'erreur 4 : division par 0

Si plusieurs des problèmes précédents sont rencontrés simultanément, vous devez les gérer dans cet ordre de priorité : le manque de paramètres est affiché en priorité, l'excès de paramètres est affiché en seconde priorité, le mauvais ordre/mauvaise nature de paramètres en troisième, et la division par 0 en dernier.

```
$ ./my_calc 1 0
Not enough parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
1
$ ./my_calc 1 0 0 /
Too much parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
2
$ ./my_calc 1 / 0
Wrong parameters.
Usage: ./my_calc number number operator
Operator might be : + - x / %
$ echo $?
3
```

```
$ ./my_calc 1 0 /
Division by 0 is forbidden.
$ echo $?
4
```

Cas d'erreurs : ordre des erreurs

Les paramètres donnés lors des tests/de la correction seront toujours des nombres ou des opérateurs. Vous n'avez pas à gérer les cas où des caractères sont donnés en paramètres.

ATTENTION

Il est formellement interdit d'utiliser le programme **bc** ou tout autre programme ou bibliothèque de calcul.

6 Exercice 3 - Transposition de matrice

Nom du(es) fichier(s) :	**my_transpose.c**
Répertoire :	**login-RATT-C/src/my_transpose.c**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640

Objectif : Le but de l'exercice est de transposer une matrice.

Vous devez écrire un programme nommé **my_transpose** qui prendra en paramètre un fichier contenant une matrice, et affichera dans le terminal la version transposée.

```
$ cat file1.txt
123
456
789
$ cat file2.txt
0123
4567
$ cat file3.txt
01
23
45
67
$ cat empty.txt
$ cat file0.txt
0
```

Exemples de fichiers d'entrée

```
$ ./my_transpose file1.txt
147
258
369
$ ./my_transpose file2.txt
04
15
26
37
$ ./my_transpose file3.txt
0246
1357
$ ./my_transpose file0.txt
0
```

Cas général

Deux cas d'erreur doivent être gérés avant tout affichage : si la matrice est vide (ou n'existe pas), et si la matrice contient autre chose que des nombres.
Si la matrice est vide ou que le fichier n'existe pas, il faut indiquer le message suivant et retourner 1.

Empty_matrix

```
$ ./my_transpose empty.txt
Empty matrix
$ echo $?
1
$ rm -f non-existent
$ ./my_transpose non-existent
Empty matrix
$ echo $?
1
```

Cas d'erreur 1

Si la matrice contient des caractères autres que des nombres, il faut indiquer le message suivant et renvoyer 2.

Incorrect_matrix

```
$ cat file_error.txt
123
abc
789
$ ./my_transpose file_error.txt
Incorrect matrix
$ echo $?
2
```

Cas d'erreur 2

7 Exercice 4 - Sapin

Nom du(es) fichier(s) :	**my_pintree.c**
Répertoire :	**login-RATT-C/src/my_pintree.c**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640

Objectif : Le but de l'exercice est d'afficher un sapin en ASCII art, dont la taille varie selon le paramètre donné.

Vous devez écrire un programme qui prendra un paramètre, et affichera selon ce paramètre un sapin. Dans le cas général, affichez le sapin, et renvoyez 0.

```
$ ./my_pintree 1
 /\
/  \
____
 ||
$ echo $?
0
$ ./my_pintree 4
    /\
   /  \
  /    \
 /      \
/        \
__________
    ||
$ echo $?
0
$ ./my_pintree 5
     /\
    /  \
   /    \
  /      \
 /        \
/          \
____________
     ||
$ echo $?
0
```

Cas général

De façon précise, voici les spécifications pour les cas 1, 4, et 5 :

```
 /\     1 espace / 0 espace  \
/  \    0 espace / 2 espaces \
____    4 _
 ||     1 espace 2 |
```

Cas général 1

```
    /\          4 espaces / 0 espace  \
   /  \         3 espaces / 2 espaces \
  /    \        2 espaces / 4 espaces \
 /      \       1 espace  / 6 espaces \
/        \      0 espace  / 8 espaces \
__________      10 _
    ||          4 espaces 2 |
```

Cas général 4

```
     /\         5 espaces / 0 espace   \
    /  \        4 espaces / 2 espaces  \
   /    \       3 espaces / 4 espaces  \
  /      \      2 espaces / 6 espaces  \
 /        \     1 espace  / 8 espaces  \
/          \    0 espace  / 10 espaces \
____________    12 _
     ||         5 espaces 2 |
```

Cas général 5

Plusieurs cas spéciaux sont à prendre en compte.

Tout d'abord, dans le cas où le paramètre 0 est donné, vous retournerez 0 et afficherez :

```
$ ./my_pintree 0
/\
||
$ echo $?
0
```

Cas 0

Si aucun ou plus de 1 paramètre est donné, vous devrez afficher le message d'erreur suivant, et retourner 1 :

`Usage:␣./my_pintree␣number`

```
$ ./my_pintree
Usage: ./my_pintree number
$ echo $?
1
$ ./my_pintree 0 4
Usage: ./my_pintree number
$ echo $?
1
```

Cas d'erreur

Si un paramètre texte est donné, il sera interprété comme 0 :

```
$ ./my_pintree blob
/\
||
$ echo $?
0
```

Cas texte

Si plusieurs paramètres sont donnés, qu'ils soient texte ou nombre, vous devez afficher le message d'erreur suivant et retourner 1.

```
$ ./my_pintree plop plop
Usage: ./my_pintree number
$ echo $?
1
$ ./my_pintree 3 plop
Usage: ./my_pintree number
$ echo $?
1
```

Cas texte d'erreur