Rattrapages Projet S1

Code C et Python

27 juin 2022

Version 1

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 :	**Mark ANGOUSTURES** `<mark.angoustures@epita.fr>`
Balise(s) du projet :	**`[RATT][PROJ-S1]`**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir/Assignment sur Teams
Nom du répertoire :	`login-RATT-Projet-S1`
Nom de l'archive :	`login-RATT-Projet-S1.tar.bz2`
Date maximale de rendu :	27/06/2022 12h00
Durée du projet :	2 semaines
Architecture/OS :	Linux - Ubuntu (x86_64)
Langage(s) :	C et Python
Compilateur/Interpréteur :	**`/usr/bin/gcc /usr/bin/python3.8`**
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-Projet-S1/monnayeur

L'arborescence attendue pour le projet est la suivante :

login-RATT-Projet-S1/
login-RATT-Projet-S1/AUTHORS
login-RATT-Projet-S1/README
login-RATT-Projet-S1/Makefile
login-RATT-Projet-S1/configure
login-RATT-Projet-S1/check/
login-RATT-Projet-S1/src/
login-RATT-Projet-S1/src/monnayeur.c
login-RATT-Projet-S1/src/monnayeur.py

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 en Python***, 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 Projet - Monnayeur 2

Nom du(es) fichier(s) :	**monnayeur**
Répertoire :	**login-RATT-Projet-S1/src/monnayeur.[c|py]**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640

Objectif : Le but de l'exercice est de reproduire un monnayeur pouvant rendre la monnaie d'un distributeur automatique. Pour cela le programme prendra plusieurs paramètres, dont la somme totale insérée et le prix du produit demandé, et retournera le nombre de pièces rendues pour chaque valeur faciale.

Vous devez écrire un programme en C **et** un programme en Python dont les sources seront dans le même dossier. Vous pouvez bien évidemment séparer votre code en plusieurs fichiers (et cela est conseillé), mais votre fonction *main* doit se trouver dans le fichier indiqué dans la spécification en introduction de l'exercice.

Le programme doit tout d'abord prendre 10 paramètres dans cet ordre précis : la somme insérée, le prix visé, le nombre de pièces de chaque valeur faciale présentes dans les cylindres du monnayeur (soit huit types de pièces : 2€, 1€, 0.50€, 0.20€, 0.10€, 0.05€, 0.02€, 0.01€).

Un format partiellement anglais sera adopté dans ce projet : les centimes seront séparés des unités par un point " **.** ", et non par une virgule, mais les milliers ne seront pas séparés (exemple : *mille six cent quarante sept euros et vingt cinq centimes* s'écrit 1647.25€).

Seuls les nombres composés au maximum de deux chiffres après la virgule seront gérés.

Les zéros en trop devant ou derrière les nombres peuvent être omis : 002.1€ doit être compris comme 2.10€, c'est-à-dire 2 euros et 10 centimes.

Le programme retournera 0 après avoir écrit l'état des cylindres du monnayeur contenant chacune des 8 valeurs faciales des pièces dans cet ordre et ce format suivi d'un retour à la ligne :

2€␣1€␣0.50€␣0.20€␣0.10€␣0.05€␣0.02€␣0.01€

```
$ #            sum  cost 2E 1E 50 20 10  5  2  1
$ ./monnayeur  5.30  10  50 50 50 50 50 50 50 50
48 50 49 49 50 50 50 50
$ echo $?
0
$
$ # 4.70 rendus = 2 * 2E + 1 * 50c + 1 * 20c
$ # Etat des cylindres du monnayeur ecrit par le programme :
$ # 2E 1E 50 20 10  5  2  1
$ # 48 50 49 49 50 50 50 50
```

Cas général : insertion de 5.30€ pour un produit à 10€, en rendant 4.70€

Si vous avez assez de pièces à valeur faciale suffisament élevée, vous devez les préférer sur la petite monnaie (vous devez donc implémenter un algorithme dit *glouton* : l'algorithme préfère consommer les grosses valeurs en premier).

```
$ #            sum  cost 2E 1E 50 20 10  5  2  1
$ ./monnayeur  10    1   50 50 50 50 50 50 50 50
46 49 50 50 50 50 50 50
$ echo $?
0
```

Cas général : insertion de 10€ pour un produit à 1€, en rendant 9€

Si le monnayeur a assez d'argent en petites pièces, vous devez rendre la monnaie avec celles-ci.

```
$ #            sum  cost 2E 1E 50 20 10  5  2  1
$ ./monnayeur  10    1    0  3 10  0 10  0  0  0
0 0 0 0 0 0 0 0
$ echo $?
0
```

Cas général : insertion de 10€ pour un produit à 1€, en rendant 9€ en petites pièces

L'utilisateur peut indiquer des valeurs entières avec des zéros devant ou une virgule vide, mais vous devez absolument supprimer les zéros inutiles en sortie. Si un cylindre est vide lorsqu'il n'a plus aucune pièce, vous l'indiquerez avec un unique zéro.

```
$ #            sum  cost 2E 1E 50 20 10  5  2  1
$ ./monnayeur   1.   1    0 001 1 00 00 00 00 00
0 1 1 0 0 0 0 0
$ echo $?
0
```

Cas général : écriture des résultats en sortie avec simplification

Si l'utilisateur insère 0€ et demande un produit à 0€, vous effectuerez la transaction en renvoyant 0 et en décrivant l'état du monnayeur en supprimant les zéros inutiles.

```
$ #            sum  cost 2E 1E 50 20 10  5  2  1
$ ./monnayeur   0    0    0 001 1 42 01 00 00 00000
0 1 1 42 1 0 0 0
$ echo $?
0
```

Cas 0

4.1 Cas d'erreur 1

S'il y a trop ou pas assez de paramètres, vous devez afficher le message d'erreur suivant pour le programme C, et renvoyer 255.

Usage: ./monnayeur sum cost 2E 1E 50 20 10 5 2 1

```
$ ./monnayeur
Usage: ./monnayeur sum cost 2E 1E 50 20 10 5 2 1
$ echo $?
255
$ ./monnayeur 42
Usage: ./monnayeur sum cost 2E 1E 50 20 10 5 2 1
$ echo $?
255
$ ./monnayeur 1 2 3 4 5 6 7 8 9 10 11
Usage: ./monnayeur sum cost 2E 1E 50 20 10 5 2 1
$ echo $?
255
```

Cas d'erreur 1 (C)

En Python, vous devrez avoir le même comportement, mais écrire le message d'erreur suivant :

Usage: monnayeur.py sum cost 2E 1E 50 20 10 5 2 1

```
$ python3.8 monnayeur.py
Usage: monnayeur.py sum cost 2E 1E 50 20 10 5 2 1
$ echo $?
255
```

Cas d'erreur 1 (Python)

4.2 Cas d'erreur 2

Si les paramètres ne respectent pas la syntaxe décrite plus haut (les deux premiers paramètres doivent être des nombres composés de chiffres éventuellement séparés par une virgule suivie de deux chiffres au plus pour les centimes, et les paramètres suivants doivent être des entiers), alors il faut écrire le message d'erreur suivant, et renvoyer 254.

Syntax␣error␣in␣parameters.
-␣Two␣first␣parameters␣must␣be␣numbers␣separated␣by␣a␣dot
-␣Significant␣digits:␣2
-␣Eight␣next␣parameters␣must␣be␣integers

```
$ ./monnayeur 42,1 10  50 50 50 50 50 50 50 50
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
$ ./monnayeur 42 10.150  50 50 50 50 50 50 50 50
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
$ ./monnayeur 42 10  50.5 50 50 50 50 50 50 50.32
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
```

Cas d'erreur 2 (C et Python)

L'erreur sur le nombre de paramètres prime sur l'erreur de syntaxe.

```
$ ./monnayeur 42,1
Usage: ./monnayeur sum cost 2E 1E 50 20 10 5 2 1
$ echo $?
255
```

Cas d'erreur 1 et 2

4.3 Cas d'erreur 3

Si la valeur insérée est trop faible, vous devez afficher le message d'erreur suivant, et renvoyer 253.

`Not␣enough␣money.`

```
$ ./monnayeur 2 10  50 50 50 50 50 50 50 50
Not enough money.
$ echo $?
253
$ ./monnayeur 0 1  50 50 50 50 50 50 50 50
Not enough money.
$ echo $?
253
```

Cas d'erreur 3 (C et Python)

L'erreur de syntaxe prime sur le montant trop faible.

```
$ ./monnayeur 2,10 10  50 50 50 50 50 50 50 50
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
$ ./monnayeur 2 10,1  50 50 50 50 50 50 50 50
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
$ ./monnayeur 2 10  50 50 50 50 50 50 50 50,
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
```

Cas d'erreur 2 et 3 (C et Python)

4.4 Cas d'erreur 4

Si le monnayeur n'a pas assez de pièces en réserve pour rendre la monnaie, vous devez afficher le message d'erreur suivant, et renvoyer 252.

Not␣enough␣money␣to␣give␣back␣change.

```
$ ./monnayeur 8.10 10  0 1 0 0 0 0 0 0
Not enough money to give back change.
$ echo $?
252
```

Cas d'erreur 4 (C et Python)

L'erreur de syntaxe prime sur la monnaie manquante.

```
$ ./monnayeur 8,10 10  0 1 0 0 0 0 0 0
Syntax error in parameters.
- Two first parameters must be numbers separated by a dot
- Significant digits: 2
- Eight next parameters must be integers
$ echo $?
254
```

Cas d'erreur 2 et 4 (C et Python)

Afin de ne pas vous perdre lors de la gestion des nombres à virgules, vous pouvez tout à fait choisir de considérer que les centimes sont les plus petites unités (ainsi, 1 centime sera comptabilisé comme 1, et 1 € sera comptabilisé comme 100).