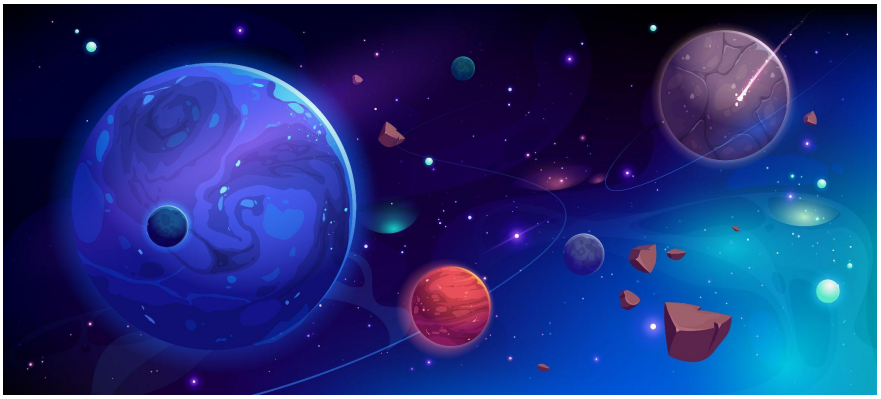
ASYNCONF

TOURNOI D'ALGORITHMIQUE ET DE PROGRAMMATION

Thème : Espace



Spécifications :

Les `[entrées]` sont des données variables, qui doivent être entrées via la console.

Les `{sorties}` sont des données qui doivent être affichées dans la console, dans le format défini par l'exercice.

Les principaux points qui seront évalués sont :

- Les fonctionnalités : le code doit marcher pour les tests donnés dans chaque exercice.
- La fiabilité : le code doit marcher pour des tests non donnés, mais conformes à la consigne.
- La robustesse : le code doit pouvoir marcher dans toutes les situations et prévoir les cas où des erreurs peuvent survenir.
- La lisibilité : le code doit être organisé, aéré et explicite.
- L'optimisation : le code doit effectuer l'exercice simplement, sans moyens sur ou sous-proportionnés.

D'autres critères seront pris en compte mais ceux-cis sont les principaux. Le total des points s'exprimera sous la forme d'une note sur 100 points, avec 20 points par exercice. Le détail des notes pourra être donné sur demande à la fin du tournoi.

Pour les exercices où c'est possible, un **set de données de test** vous est fourni. Cela vous permet de vérifier que votre programme fonctionne bien comme il est censé le faire.
Attention : lors de l'évaluation de votre programme, d'autres données que celles testées seront utilisées et, par conséquent, votre programme doit s'adapter à toutes les données qui pourront lui être fournies conformément à l'énoncé. La gestion des erreurs en cas d'entrée invalide sera valorisée.

Règles du tournoi :

- Le code doit être rendu sur un lien https://replit.com sur le Discord de l'Asynconf (Cliquer ici) dans le salon #📥 | déposer-son-projet.
  Toutes les informations dont vous avez besoin pour le rendu du projet sont disponibles en tant que message épinglé sur le salon.
- Le langage utilisé pour réaliser les différents exercices est libre, mais le langage de réalisation doit être le même pour tous les exercices.
- Le tournoi est individuel et ne doit donc pas être réalisé à plusieurs.
- L'utilisation d'internet afin de chercher des informations et/ou de l'aide est autorisée.
- Tout code identique trouvé sur deux projets vaudra une disqualification à la fois du copieur et du copié. Veillez à bien garder votre code secret durant la durée des épreuves, afin de ne pas vous faire disqualifier.
- Vous n'êtes pas obligés de faire tous les exercices. Si un exercice vous semble trop dur, ou que vous n'y arrivez pas, n'hésitez pas à passer à l'exercice suivant, mais veuillez préciser dans votre fichier principal les exercices que vous n'avez pas traités.
- Votre fichier main (le fichier exécuté par Repl.it) doit permettre de choisir l'exercice à exécuter (voir annexe à la fin du fichier).
- L'usage de tests unitaires sera valorisé. Attention cependant, les tests unitaires sont un plus et ne doivent pas remplacer les entrées console.
- L'usage de café afin de rester éveillé est autorisé, et recommandé ☕ (surtout pour l'exercice 5).

© Crédits :

Ce sujet a été conjointement réalisé par RedsTom et Graven, et mis en forme par RedsTom en suivant le thème instauré par l'édition de 2021.

Sur ce,
Bonne chance à tous 🍀

Exercice 1 : Nommons les étoiles

Difficulté : Facile
Temps estimé : 20 min.

Contexte : L'ESA (European Space Agency) cherche un nouveau code pour nommer ses missions. Elle en a trouvé un et cherche à automatiser la création du nom de ses missions via un programme.

Énoncé : Vous disposez du set de données ci-dessous, à vous de deviner comment les techniciens de l'ESA ont réfléchi pour nommer leurs missions et de créer un programme pour pouvoir créer un code pour n'importe quelle mission à partir de ses étapes. Votre programme devra prendre en compte les cas où plusieurs étapes pourraient se confondre et adapter la notation en fonction.

Entrée : Plusieurs `[étapes]` vous sont fournies une par une. Lorsque l'utilisateur entre une étape vide, le programme considère que toutes les étapes ont été saisies.

Sortie : Le `{code mission}` de la mission.

Set de données de test :

Entrée	Sortie
étapes	code mission
Jupiter Terre	J6T4
Lune Terre Soleil	L3T4S5
Terre Mars Mercure	T4M3Me5
Pluton Mercure Terre Mars Calisto	P5M6T4Ma2C6

Exercice 2 : Mission Phantom 2064

Difficulté : Facile
Temps estimé : 45 mins.

Contraintes : L'utilisation des regex est interdite lors de cet exercice.

Contexte : Maintenant que vous avez déterminé le nom de vos missions, vous cherchez à savoir le prix de chacune d'elles afin de prévoir votre budget efficacement. Vous cherchez à calculer le prix d'un trajet avec un vaisseau donné. À l'aide des caractéristiques du vaisseau qui vous sont données, et de la durée estimée de la mission, déterminez le prix du trajet.

Rappel : $distance = vitesse \times temps$

Entrées :

- Les `[caractéristiques du vaisseau]` vous sont fournies sous le format « `name=[nom];speed=[vitesse]km/h;price=[prix]/km` » où `[nom]` (le nom du vaisseau), `[vitesse]` (la vitesse du vaisseau) et `[prix]` (le prix par kilomètre du vaisseau en euros) sont des données changeantes.
- La `[durée du trajet]` vous est fournie en jours

Sortie : Votre programme devra renvoyer le `{prix du trajet}` en euros (symbole affiché), arrondi au dixième.

Set de données de test :

Entrées		Sorties
caractéristiques du vaisseau	temps de trajet	prix du trajet
name=Crystal;speed=20000km/h;price=400/km	10	1920000000€
name=Atmos;speed=2045km/h;price=23/km	2	2257680€
name=CircleBurn;speed=178547km/h;price=3612/km	6	92867294016€
name=SpaceDestroyer;speed=98928423km/h;price=9294/km	12	264798939848256€

Exercice 3 : Gérez vos tâches

Difficulté : Moyen
Temps estimé : 1 heure.

Cet exercice est un peu spécial : en effet, contrairement aux autres, il n'est pas principalement basé sur de l'algorithmique mais vous propose plutôt un mini-projet à réaliser. Ainsi, vous n'aurez pas de set de test, mais un cahier des charges à respecter que vous interpréterez pour réaliser l'exercice à votre manière. Sur cet exercice seront principalement jugés la facilité d'utilisation et l'expérience globale du programme.

Contexte : Dans votre vaisseau, vous devez gérer les tâches des membres d'équipage. Dans ce but, vous voulez avoir accès à un programme en ligne de commande vous permettant d'assigner des tâches à différents membres.

Spécifications (ce que votre programme doit pouvoir faire) :

- Un système de comptes ayant des permissions (par exemple ajouter des tâches, les compléter, les supprimer, etc.) et ne pouvant exécuter que les commandes auxquelles il a droit. Un compte est identifié par un pseudo seulement. L'utilisateur par défaut a pour nom "Administrateur". Le seul utilisateur disposant de toutes les permissions est l'administrateur.
- Un système de commandes permettant d'interagir avec des actions dans le code. On doit pouvoir savoir en regardant la ligne de commande sur quel utilisateur on est connecté.

I - Système de tâches

- Une commande "ajouter" qui demande un nom, une description, et un assigné (ou plusieurs) à la tâche. La personne assignée doit avoir un compte existant.
- Une commande "retirer" qui permet de retirer une tâche à partir de son nom (accessible uniquement à l'administrateur).
- Une commande "compléter" qui permet de compléter une tâche à partir de son nom, et qui n'est accessible que pour la personne à qui est assignée la tâche, où l'administrateur
- Une commande "liste" qui affiche, pour l'administrateur, toutes les tâches et leur état, et pour les autres utilisateurs, toutes les tâches qui leurs sont assignées et leur état.
- Une commande "vider" qui permet de supprimer toutes les tâches. Cette commande n'est accessible qu'à l'administrateur

II - Système de comptes

- Une commande "ajouter-compte" qui demande un nom de compte, et les permissions associées et crée le compte. Cette commande n'est accessible qu'à l'administrateur.
- Une commande "supprimer-compte" qui demande un nom de compte et qui supprime le compte demandé. Cette commande n'est accessible qu'à l'administrateur.
- Une commande "connecter" qui demande un nom qui est accessible à tous les comptes et qui permet de se connecter sur le compte de l'utilisateur précisé.

Exercice 4 : La supernova

Difficulté : Moyen
Temps estimé : 30 min.

Contraintes : Vous ne devrez pas utiliser de méthodes de tri fournies par votre langage ou une bibliothèque tierce lors de la réalisation de cet exercice. Vous pouvez cependant implémenter vous-même un tel algorithme (quicksort, bubblesort, etc.)

Contexte : Vous entrez dans un système solaire inconnu et votre tableau de bord vous indique la liste des planètes dans un format inconnu. Vous savez seulement que votre ordinateur est conçu pour décoder et encoder du base64.

Énoncé : Votre mission est de déterminer les planètes du système solaire et de les afficher par distance à l'étoile croissante.

Entrée : Vous disposez du `[code secret du système]` et cherchez à le décoder pour en afficher les informations.

Sortie : Les {planètes du système}, chacune séparée par un saut de ligne sous le format :

```
Nom : [nom]
Taille : [taille]km
Masse : [masse] tonnes
Distance à l'étoile : [distance]km
```

Set de données de test :

Entrées	Sorties
code secret du système	planètes du système
WwogICAgewogICAgICAgICJuYW1lIjogIlNpbG9wcCIsCiAgICAgICAgInNpemUiOiAxNDkyNCwKICAgICAgICAiZGlzdGFuY2VUb1N0YXIiOiA5MDI0ODQ1MiwKICAgICAgICAibWFzcyI6IDE5NDUzMgogICAgfSwKICAgIHsKICAgICAgICAibmFtZSI6ICJBc3RyaW9uIiwKICAgICAgICAic2l6ZSI6IDE1MjAwMCwKICAgICAgICAiZGlzdGFuY2VUb1N0YXIiOiAxNDkzMDIsCiAgICAgICAgIm1hc3MiOiAyMTk0CiAgICB9LAogICAgewogICAgICAgICJuYW1lIjogIlZhbGVudXMiLAogICAgICAgICJzaXplIjogMjkwNDUwLAogICAgICAgICJkaXN0YW5jZVRvU3RhciI6IDIwOTQ4NTkzNDU1LAogICAgICAgICJtYXNzIjogMTk1MjkzCiAgICB9Cl0=	Nom : Astrion Taille : 152000km Masse : 2194 tonnes Distance à l'étoile : 149302km Nom : Silopp Taille : 14924km Masse : 194532 tonnes Distance à l'étoile : 90248452km Nom : Valenus Taille : 152000km Masse : 2194 tonnes Distance à l'étoile : 149302km

Exercice 5 : Attaque de météorite

Difficulté : Difficile
Temps estimé : 2 heures

Contraintes : Aucune bibliothèque spécialisée ne doit être utilisée lors de cet exercice. L'algorithme doit être écrit par vous et vous seul.

Contexte : Durant votre trajet, vous traversez un champ d'astéroïdes. Malheur ! Vous n'arrivez pas à manoeuvrer vous-même. Vous cherchez donc à écrire un pilote automatique qui déterminerait toutes les positions successives que votre vaisseau doit prendre afin d'arriver à votre destination.

Énoncé : Votre vaisseau se place sur un plan 2d numéroté par des lettres horizontalement (A étant la première lettre, Z étant la lettre maximale possible), et par des nombres verticalement (1 étant le premier nombre possible). Vous cherchez le chemin vous permettant de vous déplacer de votre position actuelle à l'arrivée. Note : il n'y a qu'un seul chemin possible entre le vaisseau et l'arrivée.

Entrées : On vous fournit le `{plan de la zone}` ligne par ligne, et on considère qu'une ligne marque la fin du plan. Toutes les lignes ont la même taille. Les « `_` » sont des cases libres où le vaisseau peut circuler. Le « `X` » marque la position du vaisseau. Un « `O` » marque une météorite. Le« `V` » marque la destination du vaisseau. Le plan de la zone offre forcément un chemin possible.

Sorties : Votre programme retournera une `{liste des positions}` successives que votre vaisseau devra occuper pour sortir de ce champ d'astéroïdes. (position originale et destination comprises).

Set de données de test :

Entrées	Sorties
`plan du champ d'astéroïdes`	`positions successives`

`O___O_OO__OO__VO_O_O`
`__O___O_OOO_OO_____O`
`OO___O___OOO_OOOOO_O`
`__OO__X__OO_O___O__O`
`_OO___OO______O___OO`
	`G4;H4;I4;I5;J5;K5;L5;M5;N5;N4;O4;P4;P5;Q5;R5;R4;`
	`S4;S3;S2;R2;Q2;P2;O2;O1`

`X___OO__O___O`
`__O__OOOO____`
`OO_O__OO__O__`
`_OO_O____OO_O`
`__O_O__O_OOVO`
	`A1;B1;C1;D1;D2;E2;E3;F3;F4;G4;H4;I4;I3;J3;J2;K2;`
	`L2;L3;L4;L5`

Annexe : Guide pour du code propre.

Dans le but de faciliter le travail de l'équipe de correction, tout programme rendu doit l'être avec un fichier principal permettant de choisir l'exercice à exécuter. Nous vous proposons donc ci-dessous un pseudo-code vous permettant de, sans trop d'effort, réaliser ce fichier principal dans presque tous les langages.
Avant de commencer cependant nous allons vous donner quelques recommandations sur la manière d'organiser votre projet sur repl.it (ce guide est notamment utile pour les développeurs python ou javascript) :

- Faites un dossier par exercice nommés : "exo1", "exo2", "exo3", "exo4" et "exo5"
- Mettez un fichier "main" ou "index" par dossier d'exercice où vous mettrez votre code. Si votre exercice nécessite plusieurs fichiers, vous pourrez ainsi tous les placer dans le dossier correspondant.
- Mettez chaque exercice dans une fonction à l'intérieur de son fichier

  À faire :

  ```
  /exo1/main [pseudo-code]
  ```
  ```
  fonction exo1() {
      // Code de l'exercice
  }
  ```

  À ne pas faire :

  ```
  /exo1/main [pseudo-code]
  ```
  ```
  // Code de l'exercice
  ```

Vous pouvez maintenant éditer le fichier généré par défaut par repl.it et implémenter le pseudo-code suivant dans votre langage (selon le langage, votre code pourra être plus ou moins éloigné du pseudo-code).

```
Fichier principal [pseudo-code]
```
```
importer "exo1/main" en tant que exo1;
importer "exo2/main" en tant que exo2;
importer "exo3/main" en tant que exo3;
importer "exo4/main" en tant que exo4;
importer "exo5/main" en tant que exo5;

fonction main() {
    afficher "Quel exercice voulez-vous exécuter ? ";
    variable numero_exercice = demander un numéro;

    en fonction du numero_exercice {
        si 1 alors exo1.exo1();
        si 2 alors exo2.exo2();
        si 3 alors exo3.exo3();
        si 4 alors exo4.exo4();
        si 5 alors exo5.exo5();

        par défaut envoyer une Erreur();
    }
}
```