BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2021

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 2

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 15 pages numérotées de 1/15 à 15/15.

Le candidat traite au choix 3 exercices parmi les 5 exercices proposés.

Chaque exercice est noté sur 4 points.

EXERCICE 1 : Algorithmes de tri (4 points)

Cet exercice traite principalement du thème « algorithmique, langages et programmation ». Le but est de comparer le tri par insertion (l'un des algorithmes étudiés en 1ère NSI pour trier un tableau) avec le tri fusion (un algorithme qui applique le principe de « diviser pour régner »).

Partie A : Manipulation d'une liste en Python

1. Donner les affichages obtenus après l'exécution du code Python suivant.

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
notes[3] = 16
print(len(notes))
print(notes)
```

2. Écrire un code Python permettant d'afficher les éléments d'indice 2 à 4 de la liste notes.

1. *print(len(notes))* permet d'afficher la taille de la liste notes, soit : 8
   print(notes) permet d'afficher la liste : [8 , 7 , 18 , 16 , 12 , 9 , 17 , 3]

2. Pour afficher les éléments d'indice 2 à 4 de la liste notes, il suffit d'exécuter : notes[2 : 5]

Partie B : Tri par insertion

Le tri par insertion est un algorithme efficace qui s'inspire de la façon dont on peut trier une poignée de cartes. On commence avec une seule carte dans la main gauche (les autres cartes sont en tas sur la table) puis on pioche la carte suivante et on l'insère au bon endroit dans la main gauche.

1. Voici une implémentation en Python de cet algorithme. Recopier et compléter les lignes 6 et 7 surlignées (uniquement celles-ci).

```
1  def tri_insertion(liste):
2      """ trie par insertion la liste en paramètre """
3      for indice_courant in range(1,len(liste)):
4          element_a_inserer = liste[indice_courant]
5          i = indice_courant - 1
6          while i >= 0 and liste[i] > ..... :
7              liste[.....] = liste[.....]
8              i = i - 1
9          liste[i + 1] = element_a_inserer
```

On a écrit dans la console les instructions suivantes :

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
tri_insertion(notes)
print(notes)
```

On a obtenu l'affichage suivant : `[3, 7, 8, 9, 12, 14, 17, 18]`

1.

```
def tri_insertion(liste):
    """ trie par insertion la liste en paramètre """
    for indice_courant in range(1,len(liste)):
        element_a_inserer = liste[indice_courant]
        i = indice_courant - 1
        while i >= 0 and liste[i] > element_a_inserer :
            liste[i+1] = liste[i]
            i = i - 1
        liste[i + 1] = element_a_inserer
```

On s'interroge sur ce qui s'est passé lors de l'exécution de `tri_insertion(notes)`.

2. Donner le contenu de la liste `notes` après le premier passage dans la boucle `for`.
3. Donner le contenu de la liste `notes` après le troisième passage dans la boucle `for`.

2. Au 1er passage dans la boucle for, les différentes variables ont les valeurs suivantes :
   - indice_courant = 1
   - element_a_inserer = liste [1] = 7
   - i = 0
   - liste[i] = 8 et 8> 7 , donc la boucle while est exécutée : liste [1] = liste [0] = 8
   - liste [0] = 7

   Ainsi au 1er passage, la liste notes est égale à [7 , 8 , 18 , 14 , 12 , 9 , 17 , 3]

3. Au second passage, on :
   - indice_courant = 2
   - element_a_inserer = liste [2] = 18
   - i = 1
   - liste[1] < 18 et liste[0] < 18, donc la boucle while n'est pas exécutée
   - liste [2] = 18

   Ainsi au second passage, la liste notes est égale à [7 , 8 , 18 , 14 , 12 , 9 , 17 , 3]

   Au 3ième passage, on :
   - indice_courant = 3
   - element_a_inserer = liste [3] = 14
   - i = 2
   - liste[2] = 18 et 18 > 14 , donc : liste[3] = 18
   - liste[1] = 8 et 8 < 14 , donc la boucle while s'arrête
   - liste [2] = 14

   Ainsi au 1er passage, , la liste notes est égale à [7 , 8 , 14 , 18 , 12 , 9 , 17 , 3]

Partie C : Tri fusion

L'algorithme de tri fusion suit le principe de « diviser pour régner ».

> (1) Si le tableau à trier n'a qu'un élément, il est déjà trié.
>
> (2) Sinon, séparer le tableau en deux parties à peu près égales.
>
> (3) Trier les deux parties avec l'algorithme de tri fusion.
>
> (4) Fusionner les deux tableaux triés en un seul tableau. *source : Wikipedia*

1. Cet algorithme est-il itératif ou récursif ? Justifier en une phrase.

2. Expliquer en trois lignes comment faire pour rassembler dans une main deux tas déjà triés de cartes, la carte en haut d'un tas étant la plus petite de ce même tas ; la deuxième carte d'un tas n'étant visible qu'après avoir retiré la première carte de ce tas.
   À la fin du procédé, les cartes en main doivent être triées par ordre croissant.

Une fonction `fusionner` a été implémentée en Python en s'inspirant du procédé de la question précédente. Elle prend quatre arguments : la liste qui est en train d'être triée, l'indice où commence la sous-liste de gauche à fusionner, l'indice où termine cette sous-liste, et l'indice où se termine la sous-liste de droite.

3. Voici une implémentation de l'algorithme de tri fusion. Recopier et compléter les lignes 8, 9 et 10 surlignées (uniquement celles-ci).

```
1   from math import floor
2
3   def tri_fusion (liste, i_debut, i_fin):
4       """ trie par fusion la liste en paramètre depuis
5       i_debut jusqu'à i_fin """
6       if i_debut < i_fin:
7           i_partage = floor((i_debut + i_fin) / 2)
8           tri_fusion(liste, i_debut, .....)
9           tri_fusion(liste, ....., i_fin)
10          fusionner(liste, ....., ....., .....)
```

1. Dans cet algorithme de tri, on a une fonction *tri_fusion()*. Pour trier une liste on applique cette même fonction *tri_fusion()* sur les moitiés gauche et droite de la liste. Ainsi la fonction s'appelle elle-même. On est ainsi en présence d'un algorithme **récursif**.

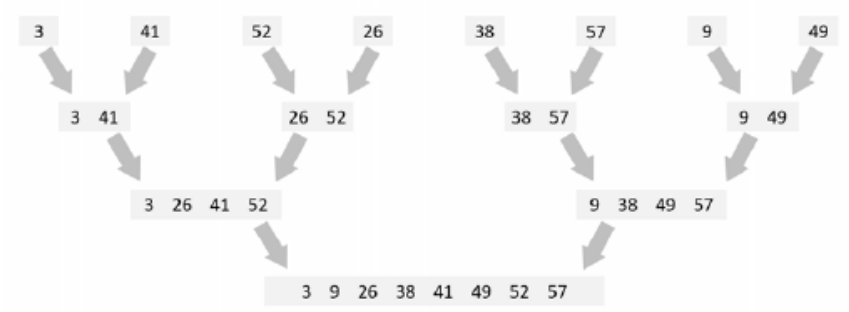
2. On tire les cartes du haut (les plus petites) de chacun des 2 tas. On prend la plus petite des 2 que l'on pose sur un 3ième tas. On retire aussitôt une carte et on recommence la comparaison.

3.
```
def tri_fusion (liste, i_debut, i_fin):
    """ trie par fusion la liste en paramètre depuis i_debut jusqu'à i_fin """
    if i_debut < i_fin:
        i_partage = floor((i_debut + i_fin) / 2)
        tri_fusion(liste, i_debut, i_partage)
        tri_fusion(liste, i_partage+1, i_fin)
        fusionner(liste, i_debut , i_partage , i_fin )
```

4. La première ligne du code permet d'importer la fonction *floor* de la bibliothèque *math* .

Partie D : Comparaison du tri par insertion et du tri fusion

Voici une illustration des étapes d'un tri effectué sur la liste [3, 41, 52, 26, 38, 57, 9, 49].



1. Quel algorithme a été utilisé : le tri par insertion ou le tri fusion ? Justifier.

2. Identifier le tri qui a une complexité, dans le pire des cas, en $O(n^2)$ et identifier le tri qui a une complexité, dans le pire des cas, en $O(n \log_2 n)$.
   Remarque : n représente la longueur de la liste à trier.

3. Justifier brièvement ces deux complexités.

1. Dans cette illustration, on a un tri par fusion.

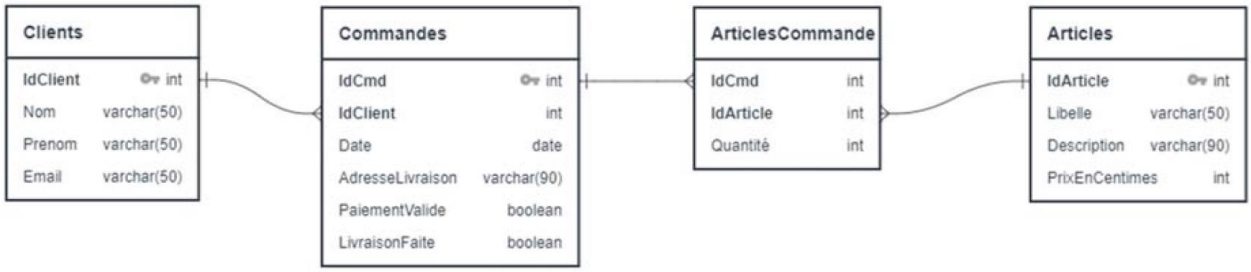
2. et 3. Dans le tri par insertion, on réalise un parcours de la liste. A l'intérieur de celui-ci, on en réalise un autre sur la partie gauche de la liste. On a ainsi une complexité en $O(n^2)$.

   Dans un tri par fusion, on divise la liste en 2 jusqu'à arriver à des listes de taille1. Le nombre de division nécessaire est $log_2(n)$. On a en effet $2^{log_2(n)} = n$. Ensuite, on effectue à nouveau $log_2(n)$ fusions. Dans celles-ci, on effectue un tri classique qui équivaut à un parcours des 2 listes à fusionner. Globalement, la compléxité de ce tri par fusion est dans le pire des cas en $O(log_2(n))$.

EXERCICE 2 : Base de données d'une plateforme de vente en ligne (4 points)

Cet exercice traite principalement du thème « traitement de données en tables et bases de données ». Nous nous interrogerons dans cet exercice sur la modélisation et l'utilisation des données nécessaires aux fonctionnements de sites de vente en ligne.

Partie A : Modèle relationnel



Lecture :

- un symbole identifie une clé primaire
- un trait entre deux attributs indique qu'ils doivent partager les mêmes valeurs. Ces valeurs sont prises de manière unique dans la relation du côté et zéro, une ou plusieurs fois du côté à condition d'être présente du côté .

1. Donner les clés primaires des relations `Clients` et `Articles`.

1. La clé primaire de la table Clients est ***IdClient*** et celle de la table Articles est ***IdArticle***

2. Les commandes ci-dessous ont été utilisées pour créer ces deux relations. Donner le domaine (c'est-à-dire le type) des attributs `Email` et `Quantite`.

```
CREATE TABLE Clients (
IdClient INT PRIMARY KEY,
Nom VARCHAR(50),
Prenom VARCHAR(50),
Email VARCHAR(50)
);

CREATE TABLE ArticlesCommande (
IdCmd INT,
IdArticle INT,
Quantite INT,
PRIMARY KEY (IdCmd,IdArticle)
FOREIGN KEY(IdArticle) REFERENCES Articles(IdArticle)
);
```

2. L'attribut *Email* est une chaine de caractère de 50 caractère au maximum. L'attribut *Quantite* est un entier.

3. En vous inspirant des commandes ci-dessus, recopier et compléter la commande suivante qui permet de créer la relation `Commandes` en précisant sa clé étrangère.

```
CREATE TABLE Commandes (
IdCmd INT PRIMARY KEY,
IdClient INT,
Date DATE,
AdresseLivraison VARCHAR(90),
PaiementValide BOOLEAN,
LivraisonFaite BOOLEAN,
FOREIGN KEY(.....) REFERENCES ..... .....
(.....)
);
```

3. Pour créer la clé étrangère, on a en SQL : FOREIGN KEY (IdClient) REFERENCES Clients (IdClient)
Pour créer la clé primaire : PRIMARY KEY (IdCmd)

Partie B : Site web

La plateforme de vente en ligne possède un site web pour ses clients qui passent des commandes en remplissant un formulaire.

1. Expliquer pourquoi, lorsqu'un formulaire contient une quantité importante de données, il est préférable d'utiliser la méthode POST plutôt que GET.

2. Pour la validation du paiement, est-il préférable d'utiliser le protocole HTTP ou HTTPS ? Pourquoi ?

3. Expliquer l'intérêt de vérifier, avant la validation du formulaire, le format des informations saisies (par exemple qu'il n'y a pas de chiffre dans le nom ou qu'il y a bien un @ dans l'adresse de courriel).

1. La méthode GET inclus les données dans l'adresse URL. Les données envoyées deviennent visibles et d'autre part la taille de celles-ci est limitée. En utilisant la méthode POST, les données ne sont pas vues et leur taille n'est pas limitée.

2. Avec un protocole HTTPS, les données sont cryptées et sont ensuite envoyées au serveur de la même manière qu'avec un protocole HTTP.

3. Il est préférable de vérifier le format des données avant la validation du formulaire. L'envoi des données sur le serveur prend du temps. Un mauvais format de données entrainera un message d'erreur de la part du serveur. Il est préférable de corriger ces éventuelles erreurs directement sur le poste du client, par l'intermédiaire du navigateur.

Partie C : Requêtes SQL

1. Écrire une requête SQL permettant de récupérer l'identifiant et le libellé de tous les articles coûtant moins de 15 euros.

2. Expliquer ce que fait la requête SQL suivante.

```
SELECT u.IdClient, u.Email, v.IdCmd, v.AdresseLivraison
FROM Clients as u JOIN Commandes as v
ON u.IdClient = v.IdClient
WHERE v.PaiementValide = False;
```

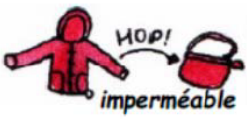
3. Écrire une requête SQL permettant de récupérer le libellé des articles de la commande 1345.

1. Pour récupérer les attributs *IdArticle* et *Libelle* de la table *Articles* pour lesquels *PrixEnCentimes* < 1500 , on exécute la requête : SELECT *IdArticle* , *Libelle* FROM *Articles* WHERE *PrixEnCentimes* < 1500

2. La requête donnée permet de récupérer les attributs *IdClient* et *Email* de la table *Clients* , ceux *IdCmd* et *AdresseLivraison* de la table *Commandes* pour lesquels l'attribut *PaiementValide* de la table *Commandes* a la valeur *False* .

3. SELECT a.Libelle FROM Articles AS a
JOIN ArticlesCommandes AS ac ON a.IdArticle = ac.IdArticle
WHERE ac.IdCmd = 1345

4. On suppose que l'attribut `IdArticle` de la table `Articles` est auto-incrémenté et ne doit donc pas être précisé lors de l'ajout d'un nouvel article. Écrire une requête SQL permettant d'ajouter dans la base ce nouvel article.



« Cet imperméable se replie en forme de pochette. »

Prix : 9,99 euros

4. La requête SQL est :

INSERT INTO Articles (Description , PrixEnCentimes) VALUES ('Cet imperméable se replie en forme de pochette' , 999)

Partie D : Adaptation du modèle relationnel

Le propriétaire du site souhaite une adaptation du modèle relationnel afin de :

- comptabiliser le stock pour chaque article
- pouvoir mémoriser, pour chaque client, une adresse de livraison par défaut de ses commandes.

1. Préciser, pour chaque relation que vous jugez nécessaire de modifier, les attributs ajoutés ainsi que leurs domaines.

2. À l'arrivée d'une nouvelle commande d'un client, l'algorithme de mise à jour de la base de données ci-dessous est exécuté.
   Indiquer l'erreur présente dans cet algorithme.

```
Algorithme nouvelle_commande (commande)
Début
   Pour chaque article de la commande Faire
      Si Quantité ≤ Stock Alors
         Stock ← Stock – 1
      Sinon
         annuler l'achat de cet article
      FinSi
   FinPour
Fin
```

1. Pour comptabiliser le stock de chaque article, on peut ajouter à la table Articles un attribut *NbReste* qui donne le nombre d'articles restants dans le stock. Son type serait INT.
   Pour mémoriser pour chaque client une adresse de livraison par défaut, on peut rajouter à la table Clients un attribut Adresse de type VARCHAR(100) ou TEXT.

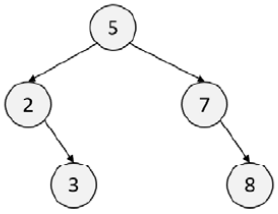
2. La ligne Stock = Stock – 1 devrait plutôt être : Stock = Stock – Quantité

EXERCICE 3 : Arbre binaire de recherche (4 points)

Cet exercice traite principalement du thème « algorithmique, langages et programmation » et en particulier les arbres binaires de recherche. La première partie aborde les arbres en mode débranché via l'application d'un algorithme sur un exemple. La suivante porte sur la programmation orientée objet. La dernière partie fait le lien avec les algorithmes de tri.

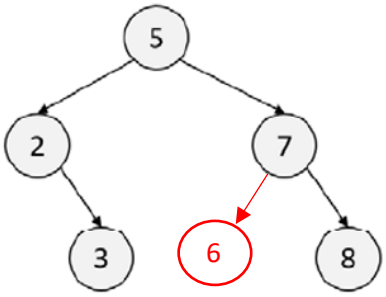
Partie A : Étude d'un exemple

Considérons l'arbre binaire de recherche ci-dessous.



1. Indiquer quelle valeur a le nœud racine et quels sont les fils de ce nœud.
2. Indiquer quels sont les nœuds de la branche qui se termine par la feuille qui a pour valeur 3.
3. Dessiner l'arbre obtenu après l'ajout de la valeur 6.

1. Le nœud racine a la valeur 5 . Ses fils ont comme valeur 2 et 7 .
2. Une branche est une suite de nœuds qui vont de la racine à une feuille. Les nœuds de la branche qui se termine par la feuille qui a pour valeur 3 ont comme valeur 5 et 2 .
3. La contrainte des ABR est que les nœuds gauches ont une valeur inférieure à celle des nœuds droits. Ainsi en ajoutant un nœud de valeur 6, on obtient l'arbre ci-contre :



Partie B : Implémentation en Python

Voici un extrait d'une implémentation en Python d'une classe modélisant un arbre binaire de recherche.

```
class ABR:
    """Implémentation d'un arbre binaire de recherche (ABR)"""
    def __init__(self, valeur=None):
        self.valeur = valeur
        self.fg = None
        self.fd = None

    def estVide(self):
        return self.valeur == None

    def insererElement(self, e):
        if self.estVide():
            self.valeur = e
        else:
            if e < self.valeur:
                if self.fg:
                    self.fg.insererElement(e)
                else:
                    self.fg = ABR(e)
            if e > self.valeur:
                if self.fd:
                    self.fd.insererElement(e)
                else:
                    self.fd = ABR(e)
```

1. Expliquer le rôle de la fonction `__init__`.

2. Dans cette implémentation, expliquer ce qui se passe si on ajoute un élément déjà présent dans l'arbre.

1. La fonction __init__ est le constructeur de la classe. Il permet d'initialiser les attributs valeur, fg et fd des instances de cette classe.

2. La méthode *insererElement()* est exécutée de manière récursive. L'objet de la classe ABR sera modifiée uniquement si on arrive sur une feuille. Si on ajoute un élément déjà présent dans l'arbre, les tests *e < self.valeur* et *e > self.valeur* seront faux tous les 2 en arrivant sur une feuille. Ainsi le processus récursif s'arrêtera sans qu'aucune modification de l'arbre n'ait été réalisée.

3. Recopier et compléter les lignes de code surlignées ci-dessous permettant de créer l'arbre de la partie A.

```
arbre = ABR(....5....)
arbre.insererElement(2)
arbre.insererElement(....3....)
arbre.insererElement(7)
arbre.insererElement(....8....)
```

Partie C : Tri par arbre binaire de recherche

On souhaite trier un ensemble de valeurs entières distinctes grâce à un arbre binaire de recherche. Pour cela, on ajoute un à un les éléments de l'ensemble dans un arbre initialement vide. Il ne reste plus qu'à parcourir l'arbre afin de lire et de stocker dans un tableau résultat les valeurs dans l'ordre croissant.

1. Donner le nom du parcours qui permet de visiter les valeurs d'un arbre binaire de recherche dans l'ordre croissant.

2. Comparer la complexité de cette méthode de tri avec celle du tri par insertion ou du tri par sélection.

3. Le parcours **infixe** trie la liste. Il consiste à parcourir de manière récursive l'ABR en commençant par le sous-arbre gauche, puis le nœud, puis le sous-arbre droit. Ce procédé de parcours conduit à visiter les nœuds dans l'ordre croissant.
4. La complexité d'un tri par insertion est en $O(n^2)$. En effet, il s'agit de parcourir la liste une première fois et pour chacun de ces parcours, on a en moyenne $n/2$ comparaison.
   Le tri par sélection a également une complexité en $O(n^2)$.

   Pour construire un ABR à partir d'une liste de n valeurs, pour chaque élément de la liste, il faut en moyenne $n/2$ comparaisons. Pour parcourir ensuite cet ABR de manière récursive, on a un parcours de liste.
   Globalement, la complexité est proportionnelle à $n \times \frac{n}{2} + n = \frac{n}{2}(n+1)$. La complexité est ainsi en $O(n^2)$ dans le pire des cas.

EXERCICE 4 : Au cœur des machines (4 points)

Cet exercice traite principalement du thème « architectures matérielles, systèmes d'exploitation et réseaux ». Cet exercice mobilise des connaissances sur le routage et sur l'évolution des architectures des machines.

Partie A : Routage dans un réseau informatique

1. Expliquer pourquoi le protocole TCP-IP prévoit un découpage en paquets et une encapsulation des fichiers transférés d'un ordinateur à un autre via Internet.

2. On souhaite modéliser un réseau informatique par un graphe pondéré pour identifier le chemin optimal pour un paquet.
   a. Préciser ce que représentent les sommets et les arêtes du graphe.
   b. Préciser si le protocole RIP utilise le nombre de sauts ou le délai de réception comme poids des arêtes.

1. Le protocole TCP-IP permet de transmettre des volumes de données très importants en les découpant en paquets de taille limitée. C'est un gage de fiabilité de la transmission. Dans chaque paquet sont encapsulés les données à transférer et les informations permettant de réaliser le transfert de manière indépendante à travers l'internet.
2. On peut modéliser un réseau informatique par un graphe pondéré.
   a. Les sommets du graphe sont des routeurs, les arêtes des moyens physiques de communication
   b. Le protocole RIP utilise le nombre de sauts comme pois des arêtes.

Partie B : Système d'exploitation

Un système d'exploitation doit assurer la gestion des processus et des ressources.

1. Dans ce contexte, expliquer et illustrer par un exemple ce qu'est une situation d'interblocage (deadlock).

2. Citer des mécanismes permettant d'éviter ces situations.

1. Une situation d'interblocage se produit lorsque des processus se bloquent car ils sont en attente l'un de l'autre.
2. Les interblocages peuvent être évités si certaines informations sont connues à l'avance lors des allocations de ressources. Pour chaque allocation de ressources, le système regarde s'il va entrer dans un état « non sûr », c'est-à-dire un état qui pourrait engendrer un interblocage. Le système ne répond favorablement qu'aux requêtes qui mènent à des états « sûrs ».

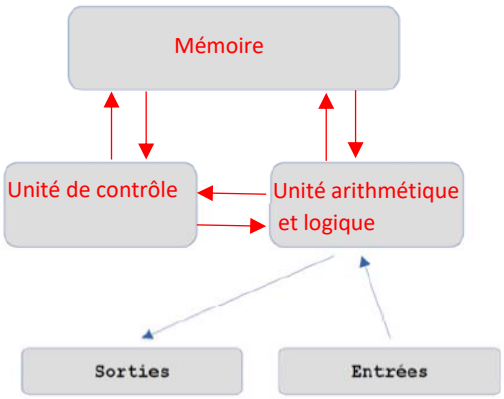
Partie C : Architectures matérielles

Architecture Von Neumann

*« L'architecture dite **architecture de Von Neumann** est un modèle pour un ordinateur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données demandées ou produites par le calcul. De telles machines sont aussi connues sous le nom d'ordinateur à programme enregistré. »* source : Wikipédia

Elle décompose l'ordinateur en 4 éléments : l'unité de contrôle *(appelé aussi unité de commande)*, l'unité arithmétique et logique *(UAL)*, la mémoire et les entrées-sorties. Les deux premiers éléments sont rassemblés dans le processeur *(CPU en anglais pour Control Processing Unit)*.

1. Recopier et compléter le schéma de cette architecture ci-dessous en faisant apparaître les communications entre les différents éléments.



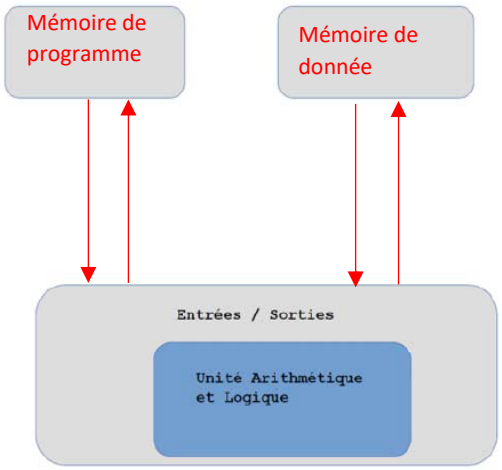
2. Dans quel(s) élément(s) sont situés le « compteur de programme » *(CP ou IP en anglais pour Instruction Pointer)* et le « registre d'instruction » *(RI ou IR en anglais pour Instruction Register)*. Préciser leurs rôles.

2. Le compteur programme et le registre d'instruction sont situés dans le processeur. Le registre « compteur de programme » CP contient l'adresse de la prochaine instruction à exécuter. Le « registre d'instruction » RI contient l'instruction en cours d'exécution.

Architecture de Harvard

*« L'**architecture de type Harvard** est une conception qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts. […] L'architecture Harvard est souvent utilisée dans les processeurs numériques de signal (DSP) et les microcontrôleurs. »* source : Wikipédia

3. Recopier et compléter le schéma de cette architecture ci-dessous et faire apparaître les communications entre les différents éléments.



4. Expliquer ce qu'est une mémoire morte et une mémoire vive. Expliquer brièvement pourquoi, dans les microcontrôleurs, la mémoire programme est une mémoire morte.

3. Les mémoires mortes sont des mémoires non volatiles, ce qui n'est pas le cas des mémoires vives.
Dans les microcontrôleurs qui sont souvent utilisés sur des systèmes embarqués, la mémoire programme est une mémoire morte car il est un composant autonome, capable d'exécuter le programme contenu dans sa mémoire morte dès qu'il est mis sous tension.

Partie D : Système sur puce

*« Un "**système sur une puce**", souvent désigné dans la littérature scientifique par le terme anglais "**system on a chip**" (d'où son abréviation **SoC**), est un système complet embarqué sur une seule puce ("circuit intégré"), pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue. »* source : Wikipédia

1. Citer un des avantages d'avoir plusieurs processeurs.

2. Expliquer pourquoi les systèmes sur puces intègrent en général des bus ayant des vitesses de transmission différentes.

3. Citer un des avantages d'un circuit imprimé de petite taille.

4. Citer un des inconvénients de cette miniaturisation.

1. Les ordinateurs multiprocesseurs permettent un parallélisme de tâches, où un processus peut être exécuté sur chaque processeur.
2. La vitesse des processeurs augmente constamment, celle des mémoires dans une moindre mesure et au regard de cette évolution, la vitesse des circuits d'entrées/sorties change relativement peu. Cette différenciation des vitesses s'est marquée progressivement et l'architecture des bus a dû s'adapter pour que les communications entre les composants restent possibles.

3. Avantages : miniaturisation, consommation d'énergie résuite
4. Inconvénients : Echauffement plus important

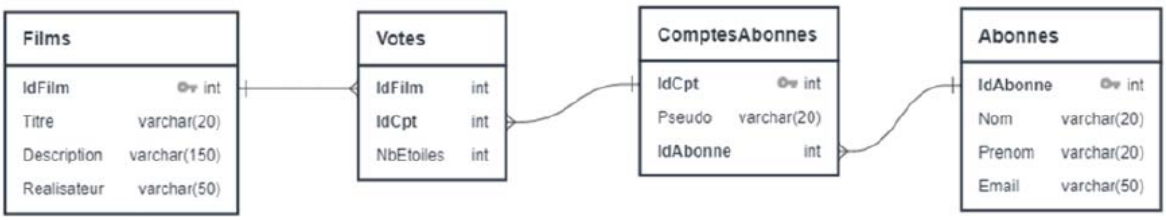
EXERCICE 5 : Vidéos à la demande (4 points)

Cet exercice traite principalement du thème « traitement de données en tables et bases de données ».

Les fournisseurs de VOD (vidéos à la demande) permettent à leurs abonnés d'avoir plusieurs comptes (afin que chaque membre de la famille puisse avoir son espace client rattaché à un compte unique pour la famille). Ils proposent également un service de votes et de conseils.

Partie A : Modèle relationnel

Voici le modèle relationnel qui sera utilisé dans cet exercice :



Lecture :

- un symbole identifie une clé primaire
- un trait entre deux attributs indique qu'ils doivent partager les mêmes valeurs. Ces valeurs sont prises de manière unique dans la relation du côté et zéro, une ou plusieurs fois du côté à condition d'être présente du côté .

1. Donner les clés primaires des relations `Films` et `Abonnes`.

1. Les clé primaires des tables Films et Abonnes sont respectivement IdFilm et IdAbonne .

2. Le code SQL suivant a été utilisé pour créer la relation `Films`. Donner le domaine (c'est-à-dire le type) des attributs `IdFilm` et `Description`.

```
CREATE TABLE Films (
IdFilm INT AUTO_INCREMENT PRIMARY KEY,
Titre VARCHAR(20),
Description VARCHAR(150),
Realisateur VARCHAR(50)
);
```

2. IdFilm est un entier (INT) et Description est une chaine de caractère composé au maximum de 150 caractères (VARCHAR 150)

3. Préciser la clé primaire et la clé étrangère de la relation `ComptesAbonnes`.

3. Sur la table *CompteAbonnes*, on a *IdCpt* qui est la clé primaire et *IdMabonne* qui est une clé étrangère de la table Abonnes

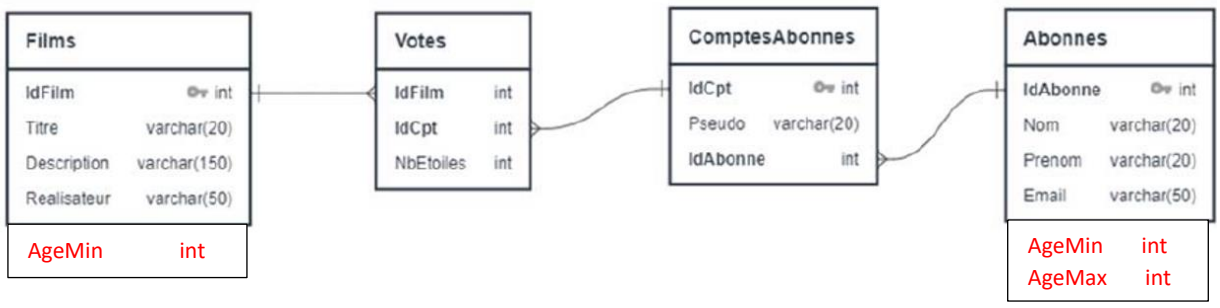
L'entreprise gérant le service de VOD demande une adaptation du modèle relationnel.

4. Elle souhaite pouvoir stocker les acteurs principaux de chaque film. Préciser les modifications à apporter.

4. Pour pouvoir stocker les acteurs principaux de chaque film, il est nécessaire de créer un attribut Acteurs dans la table Films. Le type de cet attribut sera une chaine de caractère (VARCHAR).

5. Elle souhaite pouvoir associer une tranche d'âge (-12 ans, 13-15 ans, 16-18 ans, +18 ans) à chacun des comptes d'un abonné et à attribuer, à chaque film, un âge minimum à avoir avant de le visionner. Préciser les modifications à apporter.

5. Les modifications suivantes sont à apporter :
   - Création attribut AgeMin (int) sur la table Films
   - Création attribue TrancheAge sur la table Abonnes



Partie B : Requêtes SQL

1. Écrire une requête SQL permettant de récupérer l'identifiant et le pseudo de tous les comptes rattachés à l'abonné 237

2. Expliquer ce que fait la requête SQL suivante.

```
SELECT AVG(NbEtoiles)
FROM Votes
WHERE IdFilm = 1542;
```

3. Expliquer ce que fait la requête SQL suivante.

```
SELECT u.IdFilm, u.Titre, v.NbEtoiles
FROM Films as u JOIN Votes as v
ON u.IdFilm = v.IdFilm
WHERE v.IdCpt = 508
ORDER BY v.NbEtoiles DESC;
```

4. Écrire une requête SQL permettant de modifier le pseudo du compte 508 pour lui attribuer la valeur « Champion ».

1. La requête SQL est : SELECT IdCpt , Pseudo FROM ComptesAbonnes
   WHERE IdAbonn = 508

2. La requête donnée permet de retourner la moyenne des valeurs de l'attribut *NbEtoiles* de la table *Votes* pour les enregistrements correspondants au film dont l'attribut *IdFilm* à la valeur 1542.

3. La requête retourne la liste des attributs *IdFilm* et *Titre* de la table *Films*, l'attribut *NbEtoiles* de la table *Votes* pour les enregistrements dont la valeur de l'attribut *IdCpt* de la table *Votes* est égal à 508. Cette liste est triée par la valeur de *NbEtoiles* par ordre décroissant.
   On obtient ainsi le résultat des votes des abonnés liées au compte 508, triés par ordre décroissant.

4. Cette requête de modification est :
   UPDATE *CompteAbonnes* SET *pseudo* = 'Champion' WHERE *IdCpt* = 508

Partie C : Conseils de films

Dans cette partie, nous utiliserons un module Python dont l'interface est fournie ci-dessous.

Module `ConsultationBaseVOD.py`	
`podiumCompte(IdCpt)`	Prend en paramètre un identifiant de compte et renvoie la liste des films les mieux notés par l'utilisateur de ce compte (*au maximum 10 films*) par ordre décroissant de nombre d'étoiles.
`spectateurs(listeFilms)`	Prend en paramètre une liste de films et renvoie une liste contenant les identifiants des comptes sur lesquels tous ces films ont été visionnés.
`nbEtoiles(IdFilm, IdCpt)`	Prend en paramètre un identifiant de film et un identifiant de compte. Renvoie le nombre d'étoiles attribuées à ce film par cet utilisateur.

1. Expliquer ce que fait la fonction ci-dessous.

```
def distance(IdCpt1, IdCpt2, listeFilms):
    assert (type(listeFilms) is list) and (len(listeFilms) !=
0)
    somme_ecarts = 0
    for film in listeFilms:
        somme_ecarts += abs(nbEtoiles(film, IdCpt1) - \
                            nbEtoiles(film, IdCpt2))
    return somme_ecarts / len(listeFilms)
```

1- Cette fonction détermine, pour chacun des identifiants de films contenus dans la liste *listeFilms*, la différence entre la notation attribuée par l'identifiant de compte 1 et celle attribuée par l'identifiant de compte 2. Cette fonction retourne la moyenne de ces écarts.

Ainsi, si les 2 comptes ont attribués exactement les mêmes nombres d'étoiles, la valeur renvoyée sera nulle. Si par contre, les notations sont très différentes, la valeur retournée sera élevée.

2. Traduire en Python la fonction `conseilsFilms` suivante qui permet, à partir d'un identifiant de compte, de conseiller des films proches des goûts de l'utilisateur de ce compte.

Fonction conseilsFilms (IdCpt)

Créer une liste vide nommée « conseils »

Récupérer la liste des films les mieux notés par l'utilisateur du compte passé en paramètre

Récupérer la liste des spectateurs ayant visionné ces films

Pour chacun de ces spectateurs

Si la distance avec l'utilisateur du compte passé en paramètre est inférieure à 10

Ajouter à la liste « conseils » les films préférés de ce spectateur (maximum 3)

Renvoyer la liste « conseils »

Remarque : Les répétitions sont autorisées dans la liste « conseils ».

```
def conseilsFilms(IdCpt) :
    conseils = [ ]
    listeFilms = podiumCompte(IdCpt)
    ListeCpt = spectateurs(listeFilms)
    for id in ListeCpt :
        d = distance(IdCpt,id,listeFilms)
        if d < 10 :
            liste = podiumCompte(id)
            i = 0
            while i < 3 and i < len(liste) :
                conseil.append(liste[i])
                i += 1
    return conseils
```