BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

BAC BLANC 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice et du dictionnaire n'est pas autorisé.

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

Ce sujet comporte 10 pages numérotées de 1 à 10.

Le sujet est composé de 3 exercices indépendants.

Exercice 1 : (9pt) *Cet exercice porte sur les systèmes d'exploitation, les commandes UNIX, les structures de données (de type LIFO et FIFO) et les processus.*

"Linux ou GNU/Linux est une famille de systèmes d'exploitation open source de type Unix fondée sur le noyau Linux, créé en 1991 par Linus Torvalds. De nombreuses distributions Linux ont depuis vu le jour et constituent un important vecteur de popularisation du mouvement du logiciel libre."

Source : Wikipédia, extrait de l'article consacré à GNU/Linux.

"Windows est au départ une interface graphique unifiée produite par Microsoft, qui est devenue ensuite une gamme de systèmes d'exploitation à part entière, principalement destinés aux ordinateurs compatibles PC. Windows est un système d'exploitation propriétaire."

Source : Wikipédia, extrait de l'article consacré à Windows.

1) Expliquer succinctement les différences entre les logiciels libres et les logiciels propriétaires.
2) Expliquer le rôle d'un système d'exploitation.

On donne ci-dessous une arborescence de fichiers sur un système GNU/Linux (les noms encadrés représentent des répertoires, les noms non encadrés représentent des fichiers, / correspond à la racine du système de fichiers) :

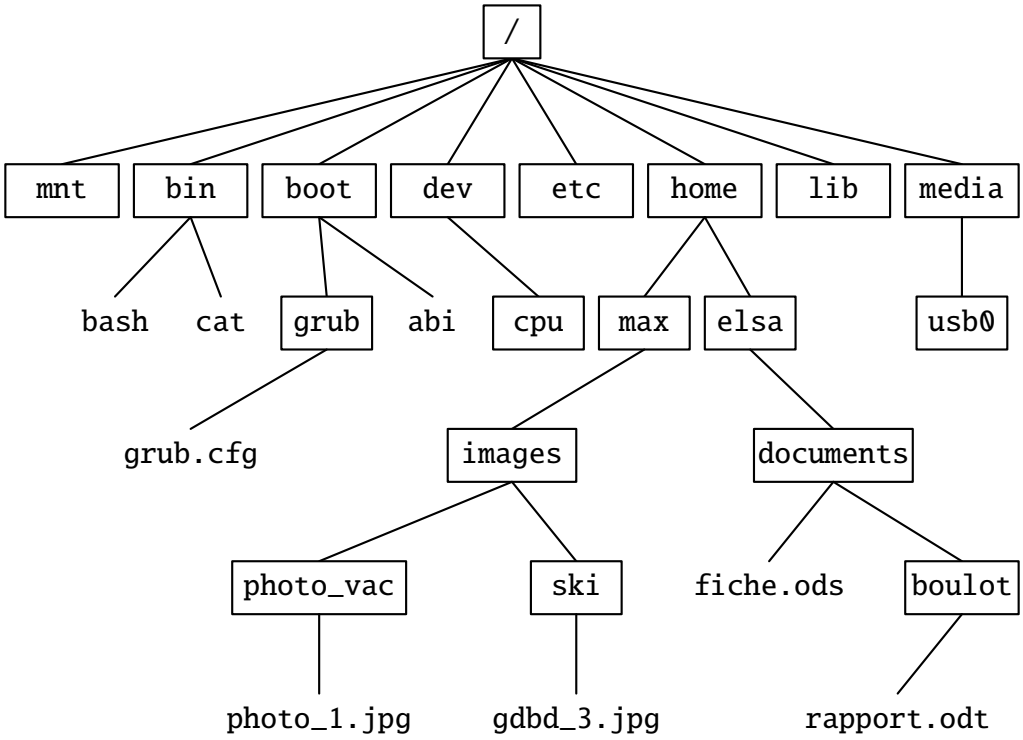


Figure 1 : Arborescence des fichiers

3) Indiquer le chemin absolu du fichier `rapport.odt`.
4) On suppose que le répertoire courant est le répertoire `elsa`. Indiquer le chemin relatif du fichier `photo_1.jpg`.

L'utilisatrice Elsa ouvre une console (aussi parfois appelée terminal), le répertoire courant étant toujours le répertoire `elsa`. On fournit ci-dessous un extrait du manuel de la commande UNIX `cp` :

```
NOM
    cp - copie un fichier

UTILISATION
    cp fichier_source fichier_destination
```

5) Déterminer le contenu du répertoire `documents` et du répertoire `boulot` après avoir exécuté la commande suivante dans la console :
`cp documents/fiche.ods documents/boulot`

"Un système d'exploitation est multitâche (en anglais : multitasking) s'il permet d'exécuter, de façon apparemment simultanée, plusieurs programmes informatiques. GNU/Linux, comme tous les systèmes d'exploitation modernes, gère le multitâche."
"Dans le cas de l'utilisation d'un monoprocesseur, la simultanéité apparente est le résultat de l'alternance rapide d'exécution des processus présents en mémoire."

Source : Wikipédia, extraits de l'article consacré au Multitâche.

Dans la suite de l'exercice, on s'intéresse aux processus. On considère qu'un monoprocesseur est utilisé. On rappelle qu'un processus est un programme en cours d'exécution. Un processus est soit élu, soit bloqué, soit prêt.

6) Recopier et compléter le schéma ci-dessous avec les termes suivants :
élu, bloqué, prêt, élection, blocage, déblocage.

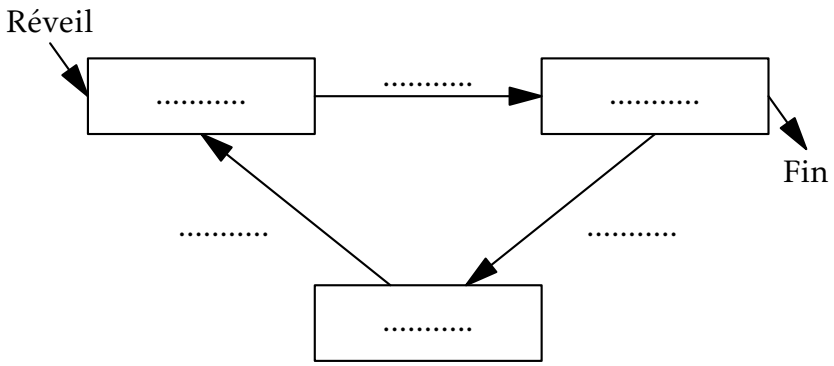


Figure 2 : Schéma processus

7) Donner l'exemple d'une situation qui contraint un processus à passer de l'état élu à l'état bloqué.

"Dans les systèmes d'exploitation, l'ordonnanceur est le composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur le processeur d'un ordinateur."

Source : Wikipédia, extrait de l'article consacré à l'ordonnancement.

L'ordonnanceur peut utiliser plusieurs types d'algorithmes pour gérer les processus. L'algorithme d'ordonnancement par "ordre de soumission" est un algorithme de type FIFO (First In First Out), il utilise donc une file.

8) Nommer une structure de données linéaire de type LIFO (Last In First Out).

À chaque processus, on associe un instant d'arrivée (instant où le processus demande l'accès au processeur pour la première fois) et une durée d'exécution (durée d'accès au processeur nécessaire pour que le processus s'exécute entièrement).
Par exemple, l'exécution d'un processus P4 qui a un instant d'arrivée égal à 7 et une durée d'exécution égale à 2 peut être représentée par le schéma suivant :

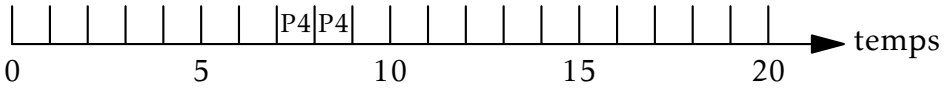


Figure 3 : Utilisation du processeur

L'ordonnanceur place les processus qui ont besoin d'un accès au processeur dans une file, en respectant leur ordre d'arrivée (le premier arrivé étant placé en tête de file). Dès qu'un processus a terminé son exécution, l'ordonnanceur donne l'accès au processus suivant dans la file.
Le tableau suivant présente les instants d'arrivées et les durées d'exécution de cinq processus :

5 processus		
processus	instant d'arrivée	durée d'exécution
P1	0	3
P2	1	6
P3	4	4
P4	6	2
P5	7	1

9) Recopier et compléter le schéma ci-dessous avec les processus P1 à P5 en utilisant les informations présentes dans le tableau ci-dessus et l'algorithme d'ordonnancement "par ordre de soumission".

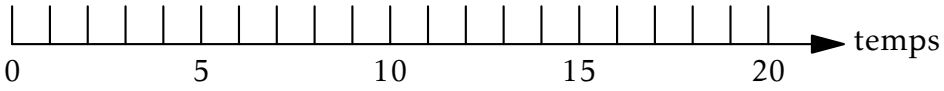


Figure 4 : Utilisation du processeur

On utilise maintenant un autre algorithme d'ordonnancement : l'algorithme d'ordonnancement "par tourniquet". Dans cet algorithme, la durée d'exécution d'un processus ne peut pas dépasser une durée Q appelée quantum et fixée à l'avance. Si ce processus a besoin de plus de temps pour terminer son exécution, il doit retourner dans la file et attendre son tour pour poursuivre son exécution.
Par exemple, si un processus P1 a une durée d'exécution de 3 et que la valeur de Q a été fixée à 2, P1 s'exécutera pendant deux unités de temps avant de retourner à la fin de la file pour attendre son tour ; une fois à nouveau élu, il pourra terminer de s'exécuter pendant sa troisième et dernière unité de temps d'exécution.

10) Recopier et compléter le schéma ci-dessous, en utilisant l'algorithme d'ordonnancement "par tourniquet" et les mêmes données que pour la question 9, en supposant que le quantum Q est fixé 2.

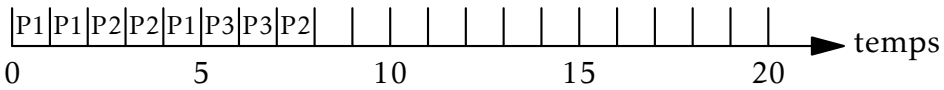


Figure 5 : Utilisation du processeur

On considère deux processus P1 et P2, et deux ressources R1 et R2.

11) Décrire une situation qui conduit les deux processus P1 et P2 en situation d'interblocage.

Exercice 2 : (17pt) *Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.*
Cet exercice est composé de 3 parties indépendantes.
On veut créer une application permettant de stocker et de traiter des informations sur des livres de science-fiction. On désire stocker les informations suivantes :

- l'identifiant du livre (`id`) ;
- le titre (`titre`) ;
- le nom de l'auteur (`nom_auteur`) ;
- l'année de première publication (`ann_pub`) ;
- une note sur 10 (`note`).

Voici un extrait des informations que l'on cherche à stocker :

Livres de science-fiction				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K. Dick	1953	9
8	Blade Runner	K. Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A
Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
dico_livres = {
    'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
    'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade Runner',
        'Les Robots', 'Ravage', 'Chroniques martiennes',
        'Dragon déchu', 'Fahrenheit 451'],
    'auteur' : ['Orwell', 'Herbert', 'Asimov',
        'K. Dick', 'K. Dick', 'Asimov', 'Barjavel',
        'Bradbury', 'Hamilton', 'Bradbury'],
    'ann_pub' : [1949, 1965, 1951, 1953,1968,
        1950, 1943, 1950, 2003, 1953],
    'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
}
a = dico_livres['note']
b = dico_livres['titre'][2]
```

1) Déterminer les valeurs des variables `a` et `b` après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None`.

```
1  def titre_livre(dico, id_livre):
2      for i in range(len(dico['id'])):
3          if dico['id'][i] == ... :
4              return dico['titre'][...]
5      return ...
```

2) Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.
3) Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.
4) Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).
5) Écrire une fonction `livres_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Partie B

Dans cette partie, on utilise le paradigme orientée objet (POO). On propose deux classes : `Livre` et `Bibliotheque`.

```
1  class Livre:
2      def __init__(self, id_livre, titre, auteur, ann_pub, note):
3          self.id = id_livre
4          self.titre = titre
5          self.auteur = auteur
6          self.ann_pub = ann_pub
7          self.note = note
8      def get_id(self):
9          return self.id
10     def get_titre(self):
11         return self.titre
12     def get_auteur(self):
13         return self.auteur
14     def get_ann_pub(self):
15         return self.ann_pub
16
17 class Bibliotheque:
18     def __init__(self):
19         self.liste_livre = []
20     def ajout_livre(self, livre):
21         self.liste_livre.append(livre)
22     def titre_livre(self, id_livre):
23         for livre in self.liste_livre :
24             if ... == id_livre :
25                 return ...
26         return ...
```

6) Citer un attribut et une méthode de la classe `Livre`.

7) Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra renvoyer la note d'un livre.
8) On a déjà créé une "bibliothèque" de type `Bibliotheque` et qui s'appelle `biblio`. Écrire le programme permettant d'ajouter le livre Blade Runner dans `biblio` en utilisant les méthodes de la classe `Livre` et la classe `Bibliotheque` (voir le tableau en début d'exercice).
9) Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque`. Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table `livres`. Cette table `livres` contient toutes les données sur les livres. On obtient donc la table suivante :

livres				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K. Dick	1953	9
8	Blade Runner	K. Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

L'attribut `id` est la clé primaire pour la table `livres`.

10) Expliquer pourquoi l'attribut `auteur` ne peut pas être choisi comme clé primaire.
11) Donner le résultat renvoyé par la requête SQL suivante :

```
SELECT titre
FROM livres
WHERE auteur = 'K. Dick';
```

12) Écrire une requête SQL permettant d'obtenir les titres des livres écrits par Asimov publiés après 1950.
13) Écrire une requête SQL permettant de modifier la note du livre Ubik en la passant de 9/10 à 10/10.

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type INT ;
- `nom` de type TEXT ;
- `prenom` de type TEXT ;
- `annee_naissance` de type INT (année de naissance).

auteurs			
id	nom	prenom	annee_naissance
1	Orwell	George	1903
2	Herbert	Franck	1920
3	Asimov	Isaac	1920
4	K. Dick	Philip	1928
5	Bradbury	Ray	1920
6	Barjavel	René	1911
7	Hamilton	Peter	1960

La table `livres` est aussi modifiée comme suit :

livres				
id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	9	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

14) Expliquer l'intérêt d'utiliser deux tables (`livres` et `auteurs`) au lieu de regrouper toutes les informations dans une seule table.

15) Expliquer le rôle de l'attribut `id_auteur` de la table `livres`.

16) Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés après 1960.

17) Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance < 30;
```

Un élève décide de créer une application d'annuaire pour sa classe. On pourra retrouver, grâce à cette application, différentes informations sur les élèves de la classe : nom, prénom, date de naissance, numéro de téléphone, adresse email, etc.

18) Expliquer en quoi la réalisation de ce projet pourrait être problématique.

Exercice 3 : (4pt) *Cet exercice est consacré aux arbres binaires de recherche et à la notion d'objet.*

1) Voici la définition d'une classe nommée `ArbreBinaire`, en Python :

```
class ArbreBinaire:
    """ Construit un arbre binaire """

    def __init__(self, valeur):
        """ Crée une instance correspondant
        à un état initial """
        self.valeur = valeur
        self.enfant_gauche = None
        self.enfant_droit = None

    def insert_gauche(self, valeur):
        """ Insère le paramètre valeur
        comme fils gauche """
        if self.enfant_gauche is None:
            self.enfant_gauche = ArbreBinaire(valeur)
        else:
            new_node = ArbreBinaire(valeur)
            new_node.enfant_gauche = self.enfant_gauche
            self.enfant_gauche = new_node

    def insert_droit(self, valeur):
        """ Insère le paramètre valeur
        comme fils droit """
        if self.enfant_droit is None:
            self.enfant_droit = ArbreBinaire(valeur)
        else:
            new_node = ArbreBinaire(valeur)
            new_node.enfant_droit = self.enfant_droit
            self.enfant_droit = new_node

    def get_valeur(self):
        """ Renvoie la valeur de la racine """
        return self.valeur

    def get_gauche(self):
        """ Renvoie le sous arbre gauche """
        return self.enfant_gauche

    def get_droit(self):
        """ Renvoie le sous arbre droit """
        return self.enfant_droit
```

a) En utilisant la classe définie ci-dessus, donner un exemple d'attribut, puis un exemple de méthode.

b) Après avoir défini la classe ArbreBinaire, on exécute les instructions Python ci-contre.
Donner les valeurs associées aux variables a et c après l'exécution de ce code.

```
r = ArbreBinaire(15)
r.insert_gauche(6)
r.insert_droit(18)
a = r.get_valeur()
b = r.get_gauche()
c = b.get_valeur()
```

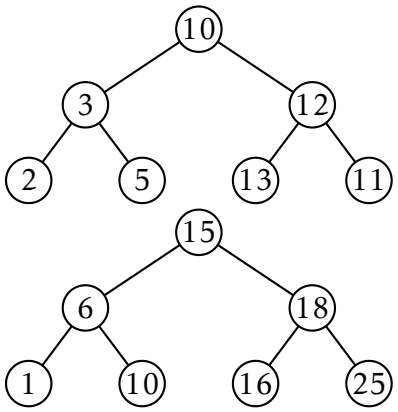
On utilise maintenant la classe `ArbreBinaire` pour implémenter un arbre binaire de recherche.
On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel :

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de `x`, alors il faut que `y.valeur <= x.valeur`.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur >= x.valeur`.

2) On exécute le code Python suivant. Représenter graphiquement l'arbre ainsi obtenu.

```
racine_r = ArbreBinaire(15)

racine_r.insert_gauche(6)
racine_r.insert_droit(18)

r_6 = racine_r.get_gauche()
r_6.insert_gauche(3)
r_6.insert_droit(7)

r_18 = racine_r.get_droit()
r_18.insert_gauche(17)
r_18.insert_droit(20)

r_3 = r_6.get_gauche()
r_3.insert_gauche(2)
```

3) On a représenté ci-contre un arbre. Justifier qu'il ne s'agit pas d'un arbre binaire de recherche. Redessiner cet arbre sur votre copie en conservant l'ensemble des valeurs {2,3,5,10,11,12,13} pour les nœuds afin qu'il devienne un arbre binaire de recherche.



4) On considère qu'on a implémenté un objet `ArbreBinaire` nommé A représenté ci-contre.

On définit la fonction `parcours_infixe` suivante, qui prend en paramètre un objet `ArbreBinaire` T et un second paramètre `parcours` de type liste.

```
1 def parcours_infixe(T, parcours):
2     """ Affiche la liste des valeurs de l'arbre """
3     if T is not None:
4         parcours_infixe(T.get_gauche(), parcours)
5         parcours.append(T.get_valeur())
6         parcours_infixe(T.get_droit(), parcours)
7     return parcours
```

Donner la liste renvoyée par l'appel suivant : `parcours_infixe(A, [])`.