BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

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 13 pages numérotées de 1/13 à 13/13.

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

Chaque exercice est noté sur 4 points.

EXERCICE 1 (4 points)

Cet exercice porte sur les structures de données (pile).

La notation polonaise inverse (NPI) permet d'écrire des expressions de calculs numériques sans utiliser de parenthèse. Cette manière de présenter les calculs a été utilisée dans des calculatrices de bureau dès la fin des années 1960. La NPI est une forme d'écriture d'expressions algébriques qui se distingue par la position relative que prennent les nombres et leurs opérations.

Par exemple :

Notation classique	Notation NPI
3+9	3 9 +
8×(3+5)	8 3 5 + ×
(17+5)×4	17 5 + 4 ×

L'expression est lue et évaluée de la gauche vers la droite en mettant à jour une pile.

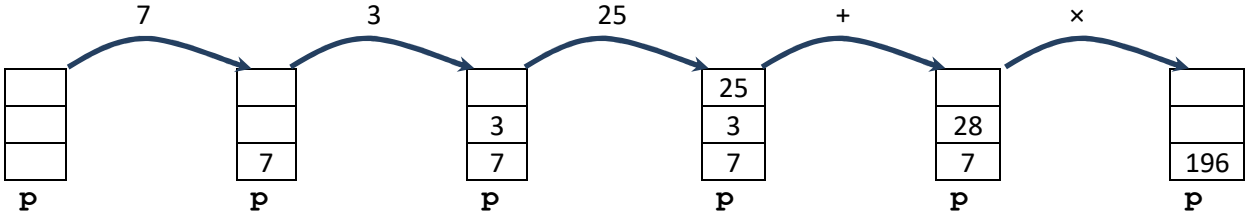
- Les nombres sont empilés dans l'ordre de la lecture.
- Dès la lecture d'un opérateur (+, -, ×, /), les deux nombres au sommet de la pile sont dépilés et remplacés par le résultat de l'opération effectuée avec ces deux nombres. Ce résultat est ensuite empilé au sommet de la pile.

A la fin de la lecture, la valeur au sommet est renvoyée.

Exemple : l'expression `7 3 25 + ×` qui correspond au calcul **7×(3+25)** s'évalue à 196 comme le montrent les états successifs de la pile créée, nommée **p** :

- On empile la valeur 7.
- On empile la valeur 3.
- On empile la valeur 25.
- On remplace les deux nombres du sommet de la pile (25 et 3) par leur somme 28.
- On remplace les deux nombres du sommet de la pile (28 et 7) par leur produit 196.

Schéma descriptif des différentes étapes d'exécution.



1. En vous inspirant de l'exemple ci-dessus, dessiner le schéma descriptif de ce que donne l'évaluation par la NPI de l'expression `12 4 5 × +` .

2. On dispose de la pile suivante nommée `p1` :

On rappelle ci-dessous les primitives de la structure de pile (LIFO : Last In First out) :

Fonction	Description
`pile_vide()`	Créé et renvoie une nouvelle pile vide
`empiler(p, e)`	Place l'élément **`e`** au sommet de la pile **`p`**.
`depiler(p)`	Supprime et renvoie l'élément se trouvant au sommet de **`p`**.
`est_vide(p)`	Renvoie un booléen indiquant si **`p`** est vide ou non.

On dispose aussi de la fonction suivante, qui prend en paramètre une pile `p` :

```
def top(p) :
    x = depiler(p)
    empiler(p, x)
    return x
```

On exécute la ligne suivante `temp = top(p1)` :

a. Quelle valeur contient la variable `temp` après cette exécution ?

b. Représenter la pile `p1` après cette exécution.

3. En utilisant uniquement les 4 primitives d'une pile, écrire en langage Python la fonction `addition(p)` qui prend en paramètre une pile p d'au moins deux éléments et qui remplace les deux nombres du sommet d'une pile p par leur somme. Remarque : cette fonction ne renvoie rien, mais la pile p est modifiée.

4. On considère que l'on dispose également d'une fonction `multiplication(p)` qui remplace les deux nombres du sommet d'une pile p par leur produit (on ne demande pas d'écrire cette fonction). Recopier et compléter, en n'utilisant que les primitives d'une pile et les deux fonctions `addition` et `multiplication`, la suite d'instructions (ci-dessous) qui réalise le calcul (3+5)×7 dont l'écriture en NPI est : 3 5 + 7 ×

```
p=pile_vide()
empiler(p,3)
```

EXERCICE 2 (4 points)

Cet exercice porte sur les bases de données.

Dans cet exercice, on pourra utiliser les mots clés suivants du langage SQL : `SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `INSERT INTO`, `UPDATE`, `VALUES`, `OR`, `AND`. Leur utilisation est rappelée en annexe 1 en fin de sujet.

La gestion d'un hôtel est faite à l'aide d'une base de données dont voici le schéma relationnel.

`Chambres (NumChambre, Prix, lits)`
`Clients (NumClient, Nom, Prenom, Telephone)`
`Reservations (NumRes, #NumClient, #NumChambre, DateArr, DateDep)`

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole `#`.

1. Expliquer pourquoi le couple `(NumClient, NumChambre)` ne pouvait pas servir de clé primaire pour la relation `Reservations`.

2.

a. Écrire une requête SQL donnant la liste des noms et prénoms des clients.

b. Écrire une requête SQL donnant le numéro de téléphone d'une cliente s'appelant "Grace Hopper".

3. Dans cette base de données, les dates `DateArr`, `DateDep` sont au format chaîne de caractères `'aaaa-mm-jj'`. Par exemple, la chaîne `'2020-10-01'` représente le 1er octobre 2020.

On peut alors les manipuler en utilisant la fonction `date()`. Par exemple :

```
SELECT *
FROM Reservations
WHERE date(DateArr) < date('2020-10-01')
```

permet de donner toutes les réservations dont la date d'arrivée est avant le 1 octobre 2020. Un client nommé "JOHN DOE" veut réserver une chambre pour la nuit du 28 décembre 2024.
Écrire une requête donnant la liste des numéros des chambres occupées à cette date.

4.

a. Le prix de la chambre 404 a changé et s'élève désormais à 75 euros. Quelle requête faut-il faire pour modifier ce prix dans la base de données ?

b. Ecrire une requête SQL affichant les numéros des chambres où a séjourné "Edgar Codd".

EXERCICE 3 (4 points)

Cet exercice porte sur la représentation binaire d'un entier relatif et les systèmes d'exploitation.

1. Codage des entiers naturels :

a. Combien de bits sont utilisés pour coder un entier naturel sur un octet ?

b. En déduire, le nombre de valeurs pouvant être codées sur un octet.

c. Donner un encadrement de ces valeurs.

2. Codage des entiers relatifs :
Pour effectuer la soustraction sur 8 bits : 65 – 58, un processeur effectue l'addition : 65 + (-58) où -58 est obtenu par la méthode du complément à deux.

a. Donner l'écriture en base 2 sur 8 bits du nombre 65.

b. Vérifier que 58 s'écrit 0011 1010 en base 2.

c. On rappelle le protocole du complément à deux pour coder un entier négatif sur 8 bits :

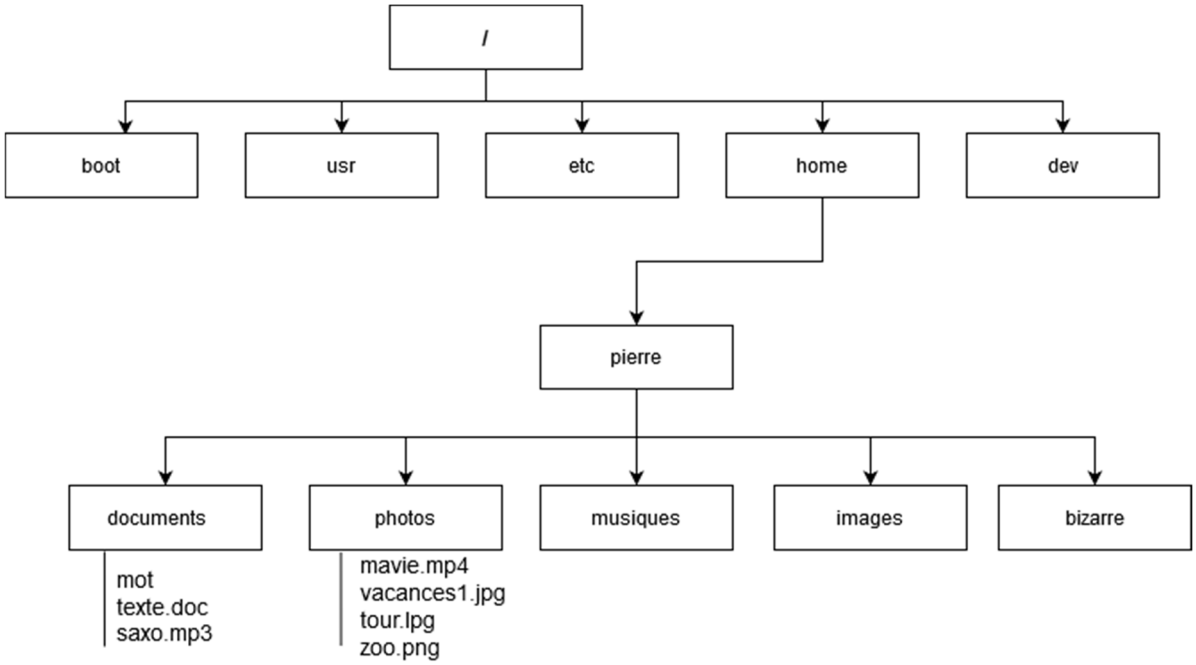
- Coder la valeur absolue du nombre en base 2 (Par exemple, la valeur absolue de -5 est 5).
- Compléter éventuellement l'octet à gauche avec des 0.
- Echanger tous les bits 0 en 1 et réciproquement.
- Additionner le nombre 00000001.

Déterminer l'écriture de -58 sur 8 bits en suivant le protocole ci-dessus.

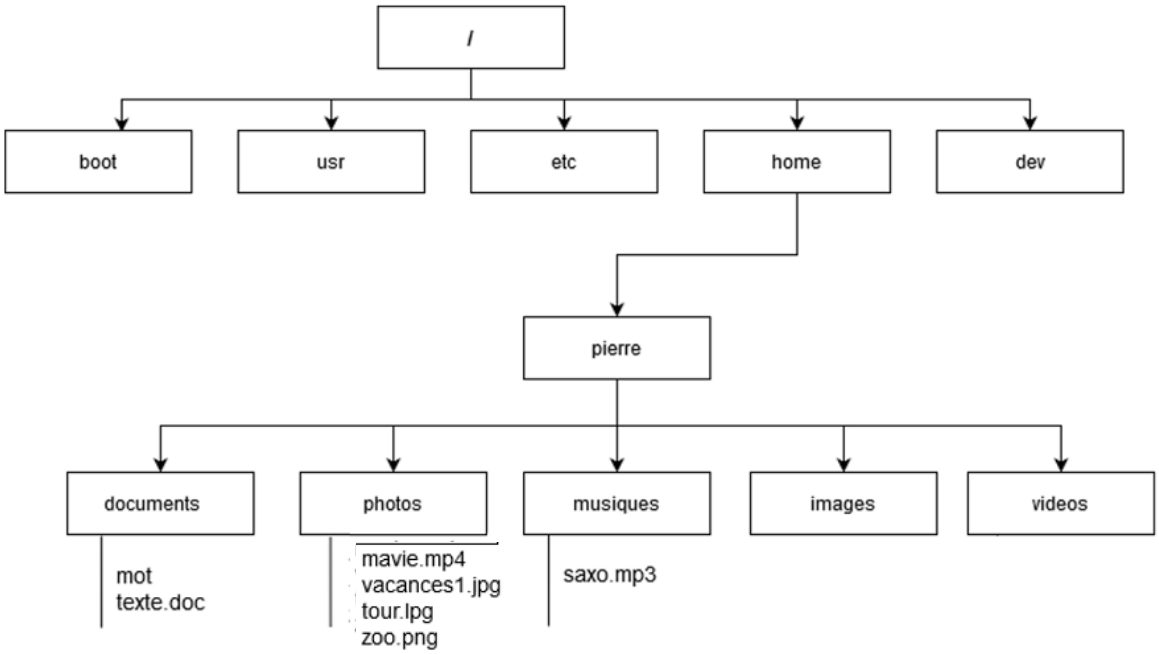
d. Effectuer la soustraction de 65 – 58 en binaire telle que ferait le processeur.

3. Un disque dur contient l'arborescence A ci-dessous et doit finalement contenir l'arborescence B ci-après. Sachant que le dossier en cours est le dossier home.

Arborescence A



Arborescence B



Vous trouverez, en annexe 2 (en fin de sujet), une liste de quelques commandes linux. Quelles commandes Linux faut-il saisir pour :

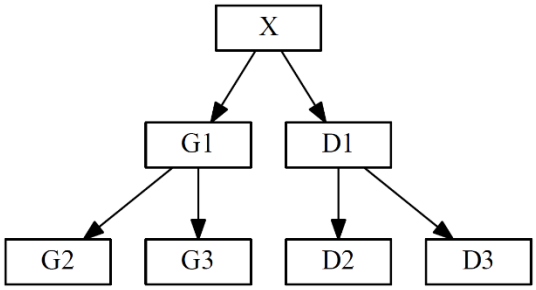
a. déplacer le fichier "`saxo.mp3`" du dossier "`documents`" vers le dossier "`musiques`" ?

b. renommer le dossier "`bizarre`" en dossier "`videos`" ?

EXERCICE 4

Cet exercice porte sur les arbres binaires de recherche.

Un arbre binaire est soit vide, soit un nœud qui a une valeur et au plus deux fils (le sous-arbre gauche et le sous-arbre droit). Dans la représentation ci-contre,



`X` est un nœud
`G1` est le fils gauche de `X`
`D1` est le fils droit de `X`

Un arbre binaire de recherche est ordonné de la manière suivante :

Pour **chaque** nœud,

- les valeurs de **tous les nœuds** du sous-arbre gauche sont **strictement inférieures** à la valeur du nœud
- les valeurs de **tous les nœuds** du sous-arbre droit sont **supérieures ou égales** à la valeur du nœud

Ainsi, par exemple, toutes les valeurs des nœuds `G1`, `G2` et `G3` sont strictement inférieures à la valeur du nœud `X` et toutes les valeurs des nœuds `D1`, `D2` et `D3` sont supérieures ou égales à la valeur du nœud `X`.

Au fur et à mesure que les billets d'une loterie sont vendus, les numéros inscrits sur les billets sont insérés dans un arbre binaire de recherche. Chaque nœud de l'arbre est un couple, la 1ère valeur correspond au numéro du billet et la seconde est une référence permettant de connaître le lieu de vente du billet.

Par exemple au couple (45,'AZ60') correspond le billet n°45 vendu par le commerçant 'AZ60'. Seul le numéro du billet est utilisé pour positionner le nœud dans l'arbre.

1.

a. Dessiner l'arbre binaire de recherche dont la racine est (45,'AZ60') et dans lequel on insère dans cet ordre les nœuds (70,'AZ60'), (22,'AZ60'), (65,'BB54'), (58,'BC25') et (67,'BC25')

b. Pour construire **la liste triée** de tous les numéros de billets vendus, préciser quel type de parcours de cet arbre faut-il programmer parmi les propositions ci-dessous :

- un parcours en largeur ;
- un parcours en profondeur infixe ;
- un parcours en profondeur préfixe ;
- un parcours en profondeur suffixe.

La fonction `filsgauche(a)` retourne le sous arbre gauche du nœud `a` et `null` si le nœud `a` n'a pas de fils gauche. Il en est de même pour la fonction `filsdroit(a)` avec le fils droit.

2. On rappelle que la taille d'un arbre est le nombre de nœuds. Recopier et compléter les [.....] de l'algorithme suivant pour que la fonction taille(a) renvoie la taille d'un arbre a.

```
fonction taille(a)
     si a est null
        alors renvoyer [.....]
     sinon
        renvoyer 1 + [.....] + [.....]
```

3. Si `a` n'est pas `null`, la fonction `billet(a)` retourne la première valeur du nœud `a` (c'est-à-dire le numéro du billet) et `reference(a)` retourne la deuxième valeur du nœud `a` (c'est-à-dire le lieu de vente du billet `billet(a)`).

a. Que fait la fonction récursive écrite en pseudo-langage suivante, où le paramètre `a` est un nœud et `n` un nombre entier ?

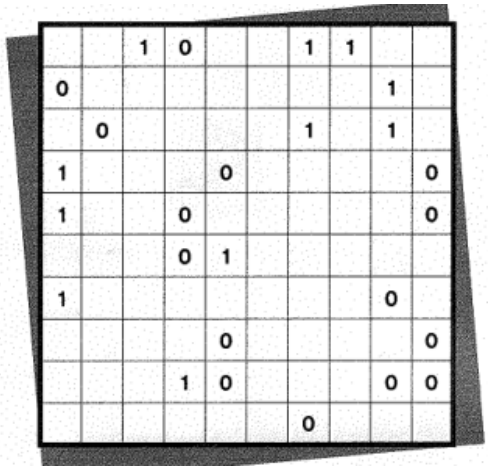
```
fonction mystere(a, n)
  si a est null
    alors renvoyer Faux
  sinon, si billet(a) vaut n
    alors renvoyer Vrai
  sinon
    renvoyer mystere(filsgauche(a)) OU mystere(filsdroit(a))
```

b. La fonction précédente est applicable à tout arbre binaire. L'arbre construit à la première question est un arbre binaire de recherche. Recopier et compléter le [.....] de la ligne 6 de l'algorithme ci-dessous pour améliorer la fonction `mystere` en tenant compte de cette remarque.

```
1. fonction mystereABR(a, n)
2.      si a est null
3.         alors renvoyer Faux
4.      sinon, si billet(a) vaut n
5.         alors renvoyer Vrai
6.      sinon si [.....]
7.        renvoyer mystereABR(filsgauche(a))
8.      sinon
9.         renvoyer mystereABR(filsdroit(a))
```

EXERCICE 5 (4 points)

Cet exercice porte sur les algorithmes et la programmation Python.



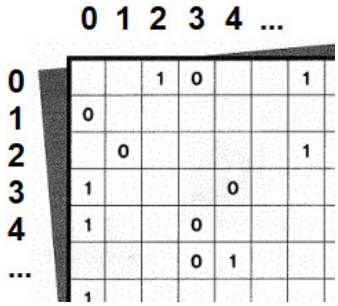
Dans le jeu du **TAKAZU**, on dispose d'une grille de 10 lignes et 10 colonnes contenant des zéros et des uns. L'objectif est de compléter les cases blanches en respectant les règles ci-dessous :

- **[REGLE1]** : chaque ligne et chaque colonne doivent contenir autant de 0 que de 1.
- **[REGLE2]** : les lignes ou colonnes identiques sont interdites.
- **[REGLE3]** : il ne doit pas y avoir plus de deux 0 ou plus de deux 1 placés à la suite, ni dans le sens vertical, ni dans le sens horizontal.

Dans cet exercice, on suppose que les valeurs de chaque ligne sont stockées dans une liste en Python, et toutes les lignes sont à leur tour placées dans une liste **globale** notée `grille`.

Ainsi, dans la situation représentée ci-contre, `grille[0][2]` vaut 1 et `grille[0][3]` vaut 0.

On décide aussi de coder par -1 toutes les cases blanches, c'est-à-dire celles dont on ne connait pas encore la valeur. Dans notre exemple, au départ `grille[0][1]` vaut -1.



1. Ecrire une fonction `autre(x)` qui renvoie 1 si `x` vaut 0 et 0 si `x` vaut 1.

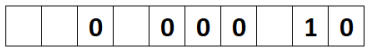
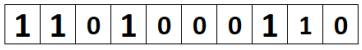
2. D'après la première règle, si une ligne contient 5 zéros, les cases blanches de cette ligne sont forcément des uns. De même si une ligne contient 5 uns, les cases blanches sont forcément des zéros.

a. Ecrire le code d'une fonction `nbValeurs(li, v)` dont les paramètres sont `li`, le numéro de la ligne de la grille et `v` la valeur que l'on souhaite compter et qui retourne la nombre de fois où la valeur `v` est présente sur la ligne `li`.

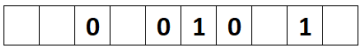
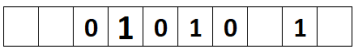
Sur l'exemple

```
>>>nbValeurs(0,1)
3
```

b. En déduire une fonction `regle1(li)` dont le paramètre `li` indique le numéro de la ligne à remplir et qui modifie l'objet `grille` en remplissant de 0 ou de 1 à partir de la première règle. Par exemple si la 6ème ligne est

 après l'appel `regle1(6)`, la ligne devient  . S'il n'y a pas assez de 0 ou de 1, la fonction ne modifie rien.

3. L'extrait de code ci-dessous effectue la recherche de deux cases identiques séparées par une blanche. Dans ce cas, on peut déterminer la valeur de la case blanche.

Par exemple, si la ligne n°4 est  , il y a deux zéros séparés par une case blanche donc la case blanche contient nécessairement un 1 sinon la troisième règle n'est pas respectée. On complètera donc en 

Recopier et compléter les ..... de la fonction `regle3(li)` dont le paramètre `li` indique le numéro de la ligne étudiée et qui modifie l'objet grille en utilisant la remarque précédente.

```
def regle3(li)
   for col in range(…):
     if grille[li][col] == grille[li][col+2] and ……:
             grille[…] = autre[…]
```

4. On suppose pour cette question que toutes les cases blanches ont été complétées. On veut déterminer si toutes les lignes sont bien différentes pour respecter la seconde règle. Chaque ligne constituée de 0 et de 1 peut être considérée comme un nombre écrit en base 2.

Ecrire le code d'une fonction `convert(L)` dont le paramètre est une liste `L` de 10 bits (0 ou 1) et renvoie la valeur décimale correspondant à la ligne `L`.

Par exemple `convert([0,0,0,0,0,1,1,1,1,1])` doit renvoyer `31`

5. On considère que l'on a stocké les valeurs obtenues à l'aide de la fonction `convert` dans une nouvelle liste `V`. On souhaite s'assurer qu'il n'y a pas de doublon.

Proposer en langage naturel l'algorithme d'une fonction dont le paramètre est une liste d'entiers et qui renvoie un booléen (VRAI ou FAUX) indiquant si cette liste possède des doublons.

ANNEXE 1 – LANGAGE SQL

- **Types de données**

CHAR(t) VARCHAR(t) TEXT	Texte fixe de t caractères. Texte de t caractères variables. Texte de 65 535 caractères max.
INT	*Nombre entier de* -2^{31} à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé).
FLOAT	Réel à virgule flottante.
DATE DATETIME	Date format AAAA-MM-JJ. Date et heure format AAAA-MM-JJHH:MI:SS.

- **Quelques exemples de syntaxe SQL :**

  - Insérer des enregistrements :

```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```

  - Modifier des enregistrements :

```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```

  - Supprimer des enregistrements :

```
DELETE FROM Table WHERE Selecteur
```

  - Sélectionner des enregistrements :

```
SELECT attributs FROM Table WHERE Selecteur
```

  - Effectuer une jointure :

```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE
Selecteur
```

ANNEXE 2 – COMMANDES LINUX

Extrait des commandes de base linux

`ls` *permet d'afficher le contenu d'un répertoire*
`cd` *se déplacer dans l'arborescence (ex : `cd repertoire1`)*
`cp` *créer une copie d'un fichier (ex : `cp fichier1.py fichier2.py`)*
`mv` *déplacer ou renommer un fichier ou un répertoire (ex : `mv fichier.txt doss`)*
`rm` *effacer un fichier ou un répertoire (ex : `rm mon_fichier.mp3`)*
`mkdir` *créer un répertoire (ex : `mkdir nouveau`)*
`cat` *visualiser le contenu d'un fichier*
`chmod` *modifier les permissions d'un fichier ou d'un dossier. Pour un fichier, le format général de l'instruction est :*

```
chmod droits_user droits_group droits_other nom_fichier
```

où `droits_user`, `droits_group` et `droits_other` indiquent respectivement les droits de l'utilisateur, du groupe et des autres et peuvent être :

```
+ ajouter
- supprimer
r read
w write
x execute
```

Ex : `chmod rwx +r -x script.sh`