8 Épreuve d'Informatique – Filière MPI

Ce chapitre présente le compte rendu de la session 2025 de l'épreuve de TP d'informatique du CCMP. L'objectif de ce document est de présenter les modalités de l'épreuve, de présenter quelques statistiques sur celui-ci et des conseils pour que les candidats soient au mieux préparés les prochaines années.
Un des objectifs de ce document étant de présenter comment les candidats pourraient mieux se préparer, il va lister beaucoup de « défauts » mais le jury précise la qualité de la prestation des candidats. Tout comme en 2024, le jury a trouvé que les candidats de la session 2025 étaient bien préparés même si certains travers pourraient être corrigés.

8.1 Déroulement de l'épreuve.

Accueil des candidats.

Chaque session de l'épreuve du TP d'informatique commençait par un accueil des candidats où les examinateurs rappelaient les consignes générales :

- les candidats peuvent manger et boire dans les salles (mais en faisant très attention aux ordinateurs !) ; ils peuvent aller aux toilettes (en demandant avant !) ;
- pendant l'oral, les candidats peuvent n'avoir avec eux que pièce d'identité, convocation, stylos et éventuellement nourriture et boisson. S'ils ont d'autres affaires (comme des sacs, des téléphones ou des montres), ils peuvent les poser éteints dans un coin de la salle. Du brouillon et un compte rendu vierge leur sont fournis ;
- les candidats peuvent poser toutes leurs questions pendant l'oral et cela n'affecte pas la notation (mais le jury se réserve le droit de ne pas répondre) ;
- il est demandé aux candidats de tester leurs codes ;
- les candidats doivent rédiger un compte rendu (voir la section détaillée, plus bas) ;
- enfin les épreuves ayant lieu sur machine, il est possible d'avoir des pannes. Dans ce cas, les candidats doivent prévenir au plus vite les examinateurs qui résoudront le problème en compensant la perte de temps par du temps supplémentaire à la fin ou dans la notation. Il est attendu des candidats qu'ils enregistrent régulièrement.

Installation dans les salles.

Après l'introduction générale, les candidats sont ensuite répartis dans les diverses salles. Cette année, il y avait encore environ 4 candidats par salle (parfois moins, exceptionnellement plus) . Les candidats posent leurs sacs dans un coin de la salle puis s'installent à un des postes disponibles. Une nouveauté de cette année était que les candidats ont eu le droit à quelques minutes de familiarisation avec la machine.

Épreuve principale.

Les examinateurs donnent le top départ à une heure précise et à ce moment les candidats doivent charger une page web où le sujet est mis.
Même si le sujet est souvent fourni en intégralité dès le début de l'épreuve, le jury déconseille de le lire en entièreté car ce serait une perte de temps, en revanche ce n'est pas une mauvaise idée de lire quelques questions à l'avance (surtout au début de l'épreuve) pour mieux comprendre ce qui est attendu.
Au cours des 3h30, les examinateurs passent régulièrement voir les candidats. Les candidats qui ont une question peuvent appeler les examinateurs sans attendre que ceux-ci ne passent.
Certains candidats tentent de sauter certaines parties sans le demander à l'examinateur. Cette technique est à proscrire lors d'un oral, elle exhibe de toute façon les lacunes du candidat (peur de la technicité, impasse, ou autre).

Fin de l'épreuve.

L'épreuve est prévue pour durer 3h30. Quelques minutes avant la fin les examinateurs rappellent aux candidats l'imminence de la fin de l'épreuve pour que ceux-ci s'assurent que le compte rendu est bien finalisé et les examinateurs passent une dernière fois pour noter où en sont les candidats. Il n'est pas demandé aux candidats de mettre leurs fichiers à un endroit spécifique à la fin de l'épreuve.

Précisions quant à l'écriture du compte rendu.

Dans l'épreuve de TP informatique le compte rendu sert principalement à suivre la progression des candidats durant l'épreuve ainsi qu'à résumer ce qui a été fait.
Quand les examinateurs passent voir les candidats, ils commencent généralement par lire le compte rendu, voient ce qui a changé depuis leur dernier passage puis, posent éventuellement des questions ou regardent ce que le candidat est en train de faire. Ceci permet d'économiser les interactions orales qui feraient perdre du temps au candidat mais aussi pourraient gêner les autres candidats dans la salle. Le compte rendu doit donc être plus détaillé qu'une succession de « Q 8 : faite » mais ce n'est pas non plus la peine de détailler les réponses autant que dans une copie écrite. En général, même les questions les plus compliquées ne requièrent pas d'écrire plus de 5 lignes.
Savoir remplir le compte rendu n'est pas une compétence attendue, le jury ne retire pas de points sur la façon dont les candidats remplissaient les compte rendus, en revanche il peut signaler à des candidats qu'ils peuvent être plus concis dans leurs réponses (pour ne pas perdre de temps) ou, au contraire, qu'il faut détailler plus leurs réponses quand ils trouve celles-ci insuffisantes. Les réponses fausses (mauvais calcul de complexité ou algorithme faux par exemple) sont prises en compte dans la notation. Voici quelques précisions sur la façon de remplir le compte rendu :

- pour toute question posée dans le sujet ou à l'oral, le jury attend une réponse écrite sur le compte rendu ;
- les sujets sont majoritairement donnés en format PDF avec des questions numérotés. Ce n'est donc pas la peine de recopier les questions, il suffit de donner le numéro de la question répondue ;

- certaines questions sont vraiment très simples (comme écrire une fonction qui somme deux vecteurs 2D) dans ce cas le candidat peut se contenter d'un « Q 1 : faite » écrit sur le compte rendu ;
- pour les questions qui n'attendent pas du code, on demande une réponse brève sur le compte rendu et de cette façon l'examinateur peut lire la réponse sans déranger le candidat (ou les autres candidats).
- enfin, pour les questions d'algorithmique ou de code non triviales, le jury attend une description générale de l'algorithme, une complexité (sans justifier) et les tests effectués. Par exemple, dans le cas où l'algorithme est un parcours de graphe, il convient d'expliquer si le parcours est en largeur ou en profondeur, dans quel graphe quand le graphe est implicite, etc. puis d'indiquer brièvement quels graphes ont été testés et que la complexité est $O(n+m)$ avec n le nombre de noeud et m le nombre d'arêtes. Le compte-rendu peut servir de "brouillon" pour préparer les tests qui sont parfois plus lisibles sous forme de dessin (par exemple pour un graphe) que sous forme de code.

8.2 Commentaires généraux sur la méthode de programmation.

Le jury a constaté que les défauts relevés dans les rapports des concours 2023 et 2024 étaient bien moins présents en 2025, mais certains défauts persistent.

Architecture du code.

Beaucoup de candidats voient chaque question comme un bloc unitaire et semblent avoir du mal à voir la combinaison de plusieurs sous-problèmes qui peuvent être implémentés dans plusieurs fonctions avec des tests.
De manière plus générale, le jury trouve que, même quand les candidats ont la bonne idée, ils ont du mal à décrire et donc à réfléchir sur leur solution avant de se lancer dans le code.
Cela pénalise souvent les candidats qui perdent du temps à ne pas tester chaque partie indépendamment quand il y a des bugs, qui écrivent souvent des bouts de code inutiles qu'ils doivent ensuite retravailler et parfois qui se rendent compte qu'au bout de 15 minutes que leur solution ne marche pas. Pour toutes les questions non triviales, le jury conseille de passer quelques minutes à élaborer rapidement la solution (par exemple sous forme de pseudo code ou juste en donnant les grandes étapes) puis à réfléchir à comment simplifier la solution ou la décomposer en plusieurs sous-problèmes.
Tout ceci fait que les candidats sont en majorité bien plus à l'aide sur les sujets très guidés avec beaucoup de questions intermédiaires, et sont parfois déroutés sur les sujets plus ouverts.

Tests.

Pour les questions d'algorithmique non triviales, il était attendu des candidats qu'ils testent leurs programmes. Cette consigne était donnée avant l'oral et les examinateurs le rappelaient régulièrement aux candidats. Il n'est pas attendu de faire des tests complets qui pourraient presque garantir une absence de bugs mais simplement vérifier qu'il n'y a pas d'erreur manifeste. Cette demande de tester les programmes remplit plusieurs objectifs qui permettent :

- de valider la compétence qui figure au programme de la filière MPI ;
- aux candidats d'avoir confiance dans ce qu'ils écrivent. Certains candidats semblent penser que tester un programme est une perte de temps voire un aveu de faiblesse. Le jury préfère toujours un candidat prudent qui teste son code à un candidat trop sûr de lui qui ne le teste pas, même quand le code est correct. Un candidat qui refuse de tester un code que le jury sait faux, fait une mauvaise impression.

Il est évident que la quantité de tests à effectuer dépend de la complexité du code de la difficulté d'écrire des tests et de la précision des tests. Une fonction qui renvoie une information booléenne (si un graphe est connexe ou non par exemple) a plus de chances de renvoyer la mauvaise réponse au hasard qu'une fonction qui renvoie quelque chose de précis (par exemple si cela affiche pour chaque composante la liste des nœuds). De la même manière une fonction très simple (par exemple la somme de deux vecteurs 2D) a moins de chances d'être fausse que l'implémentation d'un algorithme compliqué (p.ex. une file à priorité).
Voici plusieurs conseils pour les candidats :

- il est important de calculer à l'avance le résultat des tests. À plusieurs reprises, le jury a vu des candidats écrire un unique test compliqué et prendre le résultat de leur fonction comme étant le standard pour leur test alors que la fonction renvoyait un résultat faux ;
- des candidats utilisent un éditeur pour leur code et testent avec un shell (comme `utop`) et donc leurs tests "disparaissent" et ils perdent beaucoup de temps à les réécrire pour chaque petit changement de leur fonction, ce qui pousse les candidats à n'écrire que peu de tests. Le jury ne déconseille pas d'utiliser `utop` mais recommande d'écrire les tests dans le code principal ;
- il est parfois possible de tester plusieurs fonctions en même temps quand les questions sont données à l'avance. S'il est demandé, par exemple, de faire des fonctions somme, produit puis évaluation pour des polynômes, on peut tester très rapidement chaque fonction puis faire des tests qui combinent ces trois fonctions en testant que $((P+Q)*R)(42) = (P(42)+Q(42))*R(42)$ pour plusieurs polynômes P, Q et R.

Gestion des bugs.

Les outils adaptés au débuggage (comme `ocamldebug` ou `trace` en OCaml et `gdb` en C) sont peut-être un peu compliqués pour les candidats mais voici quelques conseils faciles à mettre en œuvre pour s'attaquer méthodiquement au débuggage :

- parfois un premier problème engendre un second plus visible (par exemple un calcul renvoie le mauvais résultat et à cause de cela on fait un accès hors des cases du tableau). Il faut bien penser à détecter le moment où le premier problème apparaît. Pour cela la programmation défensive et l'utilisation d'assert permet de gagner du temps. De la même manière, il est souvent intéressant de chercher à simplifier l'exemple où l'algorithme buggue, car il est plus simple de suivre le déroulé de l'algorithme ;

- les candidats pensent souvent à faire des printf mais pas toujours à l'utilisation des assert. Même si le assert est moins informatif et donc qu'il est parfois plus utile de débugguer en utilisant aussi printf, lire des dizaines de lignes prend plus de temps qu'un simple assert... il faut savoir combiner les deux ;
- quand on manipule une structure de données un peu compliquée (comme un tas ou un arbre rouge-noir) il peut être intéressant de faire une fonction qui vérifie que cette structure a le format attendu (pour un tas ou un arbre rouge-noir, par exemple, la fonction de vérification est assez simple comparé au reste de l'algorithme). Lors de la phase de test on peut par exemple vérifier la structure après chaque modification (ce qui renforce la confiance en la correction de l'algorithme car les tests testent bien mieux). Il peut être utile d'utiliser une variable `debug` pour déclencher ou non ces tests ;
- quand le candidat écrit une fonction qui attend un argument qui a une forme précise, il ne faut pas hésiter à mettre un assert pour le vérifier ;
- quand un programme a plusieurs fonctions et sous-fonctions on peut tester chaque fonction indépendamment ;
- enfin, il est presque toujours recommandé d'utiliser les warnings du compilateur et les options de compilation recommandées en C sont `-Wall -Wextra` et `-fsanitize=address` (mais un candidat qui n'a jamais testé ces options risque d'être décontenancé par les warnings).

Environnement de développement.

Nous encourageons vivement les candidats à tester la machine virtuelle proposée sur le site du concours, ne serait-ce que 10 min, pour ne pas être déstabilisé face à l'environnement qu'ils rencontreront le jour du concours. Pour la session 2026, la principale évolution prévue est la mise à jour pour la nouvelle version stable de debian (Trixie) avec comme principal changement le passage d'Ocaml en version 5.3.
Juste avant le début de l'épreuve, les candidats avaient quelques minutes pour se familiariser avec la machine. Certains candidats, pris par le stress commencent à taper des bouts de code en C ou en OCaml, le jury n'est pas persuadé que cela les aide beaucoup d'autant que les candidats ne savent à ce moment pas quel sera le langage ni si un patron de code est fourni.
Par exemple, lorsque rappelée, l'utilisation de `ocamlopt` ou de `ocamlc` peut se révéler laborieuse et le manque d'aisance fait perdre du temps aux candidats. À l'inverse, certains candidats ne savent utiliser OCaml qu'en compilant. Dans certains sujets, utiliser un shell (comme `utop` ou même `ocaml` utilisé en mode interactif) est un grand atout : on peut tester rapidement des fonctions, quand le code manipule des types sommes récursifs, on peut les afficher sans faire de fonction d'affichage ad-hoc, etc. Au passage, rappellons l'existence de `#use "fichier.ml"` qui permet de renvoyer à l'évaluation l'intégralité du fichier là où un copier-coller peut être assez lent.
Pour finir rappellons que les candidats vont passer un certain temps dans le terminal. Sans que le jury n'évalue directement la familiarité avec l'outil, certains candidats perdent beaucoup de temps en ne connaissant pas les raccourcis de base comme flèche du haut pour remonter dans l'historique, ctrl+R pour rechercher, ctrl+C pour arrêter un processus, etc.

Nom des variables, fonctions et commentaires.

Les examinateurs examinant le code des candidats, il est attendu des candidats qu'ils produisent des codes lisibles. Le jury n'attend pas des candidats des choses élaborées ou le suivi de conventions particulières mais simplement que les candidats utilisent des noms de fonctions ou de variables pertinents et des commentaires aux endroits qui le nécessitent.
Quand le nom de la fonction et de ses arguments ne suffisent pas à comprendre son objectif, ou si la fonction est longue avec plusieurs blocs qui accomplissent plusieurs choses différentes, le jury attend aussi un commentaire rapide qui décrit ce que fait le bout de code considéré.
Pour des variables locales à une fonction, les candidats peuvent se contenter de noms simples de variables mais pour les variables globales, pour les noms de fonction et éventuellement pour les variables nommant les arguments de ces fonctions, il est demandé de donner des noms qui précisent ce qu'elles représentent. Le jury n'a pas à chercher quels sont les rôles respectifs de `aux`, `aux1`, `aux2` et `aux1bis` ni ce que signifient les arguments `a`, `b` et `c` tous de type `int`.

Documentation.

Le jury fournissait une documentation accessible via le navigateur. La documentation de référence en C et OCaml étaient disponibles ainsi qu'une "cheatsheet" de sqlite3.
Les noms et prototypes des fonctions de la bibliothèque standard qui étaient nécessaires dans certains sujets (pour ouvrir des fichiers ou prendre des mutex) étaient généralement directement rappelés dans les sujets soit par de la documentation soit par des exemples mais il ne faut pas que ça empêche les candidats d'aller lire la documentation qui contient souvent des exemples d'utilisation.
Pour l'édition 2025 du concours, les principales documentations fournies pour OCaml et C étaient des copies locales de
`https://ocaml.org/manual/4.13/index.html` et `https://en.cppreference.com/w/c.html`.

8.3 Commentaires liés au programme.

Il est important de bien maîtriser et de savoir implémenter rapidement tous les algorithmes de base (parcours en largeur et en profondeur, tas, etc.), mais il faut aussi connaître toutes les définitions et tous les algorithmes du programme. Il est très dangereux de faire une impasse complète sur un point du programme.
Le jury a remarqué que certains candidats (même parmi les bons) ont une connaissance limitée voire inexistante de certains traits du langages pourtant explicitement au programme. Un exemple : certains candidats ne savent pas rattraper une exception en Ocaml (soit parce qu'ils ne connaissent pas `try` et `with` soit parce qu'ils ne savent pas qu'il faut que les types du blocs `try` et `with` soient compatibles). Rappellons que de la documentation sur les langages étant fournie et les candidats qui ont oublié certains points de syntaxe peuvent souvent s'en sortir seuls.

8.4 Commentaires liés au langage SQL.

Cette année, nous avons de nouveau eu des candidats qui se déclaraient totalement incompétents en SQL et souhaitaient sauter cette partie. L'impasse en SQL est fortement déconseillée et est pénalisée.

La plupart des candidats ont réussi à résoudre les questions de SQL mais certains le font beaucoup plus rapidement que d'autres. Nous conseillons de s'entraîner à résoudre rapidement des problèmes de SQL.
Cette année encore, tous les sujets qui manipulaient du SQL utilisaient le moteur de requête SQLite3 (dont une brève documentation était fournie) mais le jury se réserve le droit d'utiliser d'autres moteurs de requêtes dans le futur. Il n'est pas attendu de connaissance spécifique aux légères variations qui peuvent exister entre les divers moteurs de requêtes et en particulier il n'est pas attendu des candidats qu'ils connaissent les options de chaque client SQL. Le jury fournissait la commande à lancer et indiquait qu'il était recommandé de taper les commandes `.header on` et `.mode column` pour avoir des résultats de commandes plus lisibles.
Enfin le jury conseille de connaître la construction `COUNT(DISTINCT v)` car elle peut souvent simplifier l'écriture des requêtes (qui pouvaient s'écrire autrement dans tous les sujets posés).

8.5 Commentaires liés au langage C.

Le jury a été favorablement surpris par la bonne maîtrise du langage C. Voici quelques précisions pour une meilleure préparation.

Gestion de la mémoire.

Certains candidats ont du mal avec `malloc`. Dans les erreurs récurrentes que les candidats ont commises : oubli de `sizeof` (et donc mémoire allouée trop petite), tentatives d'appels à `malloc` en dehors de toute fonction (pour des variables globales), quelques candidats qui ne savent pas allouer un tableau 1D, d'autres, plus nombreux, qui ont des problèmes avec les tableaux 2D (que ce soit des tableaux de tableaux ou des tableaux linéarisés).

Compilation.

Le jury recommande fortement aux candidats de compiler avec les options `-Wall -Wextra` et `-fsanitize=address` car cela permet d'attraper diverses erreurs et facilite le débuggage. Quand il n'y a qu'un seul fichier à compiler, le jury déconseille de compiler en deux étapes (fichier objet puis exécutable) car cela fait perdre du temps aux candidats, et ce d'autant plus que certains candidats retapent entièrement chaque commande dans le shell (plutôt que de rechercher avec ctrl+r ou de relancer une commande précédente avec flèche du haut). Attention à l'utilisation de `-fsanitize=address` si cette option permet de détecter des bugs, elle affichera une sortie qui ressemble à une erreur en cas de mémoire non libérée, il est préférable que les candidats aient l'habitude de ces options.

Libération de la mémoire et valeurs de retour des fonctions de la bibliothèque.

Le jury ne demande pas de libérer la mémoire et il ne faut pas que les candidats perdent du temps à le faire (sauf demande explicite dans le sujet ou par l'examinateur). De la même manière la vérification des valeurs de retour des fonctions (comme fopen) n'est pas attendue.
Le jury n'a pas pris en compte la qualité du code (pertinence des noms de variables et fonctions, commentaires, tests et assert, lisibilité) dans la note. Les candidats qui écrivent du code de mauvaise qualité rendent très difficile l'aide de l'examinateur à débugguer du code voire empêche les candidats eux-même de débugguer.

8.6 Commentaires liés au langage OCaml.

La maîtrise du langage OCaml par les candidats est assez bonne mais le jury formule quelques conseils et remarques pour les candidats.

Utilisation simple de la bibliothèque standard.

Peu de candidats utilisent les fonctions "de base" sur la manipulation de listes (comme `filter`, `map`, `iter`, `exists`) qui sont au programme. Le jury ne pénalise pas leur non-usage mais les connaître permet souvent d'écrire du code plus court donc plus rapide à écrire et plus simple à relire. Le jury a aussi vu très peu d'utilisation des tables de hachage (par exemple pour détecter des doublons).

Utilisation avancée de la bibliothèque.

Seule une petite partie de la bibliothèque standard OCaml est au programme et seule cette partie est exigible mais cela ne veut pas dire que les candidats doivent s'interdire toute autre fonction. Les candidats ayant une installation standard d'OCaml avec la documentation complète, le jury souhaite leur laisser accès à ces bibliothèques pour ne pas pénaliser ceux qui en ont l'habitude mais il ne souhaite pas non plus que la connaissance des fonctionnalités avancées de la bibliothèque avantage certains candidats.

Il y a, par exemple, dans la bibliothèque standard les `Set`. Ces `Set` peuvent servir d'arbres binaires équilibrés ou de files de priorité. Si une question demande d'écrire un arbre binaire de recherche équilibré, un candidat peut répondre avec `Set`. Si un candidat veut utiliser de telles fonctions avancées, il convient d'abord demander à l'examinateur.

Le jury a constaté l'utilisation de nombreuses fonctions de la bibliothèque hors programme comme `List.mapi`, `List.assoc`, `Array.of_list` ou même `String.split_on_char` et le jury a considéré toutes ces utilisations acceptables du moment que les candidats pouvaient rapidement les ré-implémenter. Le jury ne conseille pas spécialement l'apprentissage de ces fonctions qui seraient données par le Jury en cas de besoin. Cela étant, connaître, par exemple, la structure de liste associative ou l'idée de décomposer un problème de parsing en un découpage de la chaîne en token pour ensuite traiter chaque token peuvent aider les candidats.

Attention tout de même à l'utilisation de la bibliothèque. De multiples candidats qui ont un style de programmation impératif et utilisent beaucoup, par exemple, `List.nth` sans faire attention à la complexité de cette opération ! Quand les candidats utilisent une fonction de la bibliothèque dans un algorithme, ils doivent être capables de donner la complexité de l'algorithme.

Des sujets peuvent nécessiter l'utilisation de bibliothèques, auxquels cas les candidats auront à lire la documentation. Certains candidats ont utilisé la documentation pour voir ce qui existait. Par exemple, pour un exercice de parsing certains ont regardé la documentation du module `String`. Tant que les candidats n'utilisent pas des fonctions trop avancées, le jury n'a aucun de problème avec cette pratique. Lire la documentation n'est pas forcément évident et il est donc conseillé d'avoir déjà de l'expérience avec la documentation OCaml.

Style de programmation.

Certains candidats connaissent les `ref` mais pensent qu'il ne faut surtout pas les utiliser en vertu d'une programmation fonctionnelle pure. La pureté du programme écrit n'étant pas notée, c'est dommageable pour les candidats qui perdent du temps à cause de cela ; par exemple, en voulant utiliser une boucle `for` mais sans `ref` ou pour faire un calcul simple sur un tableau.
À l'inverse, certains candidats ne programment qu'en style impératif, et c'est parfois plus compliqué. Par exemple, si le sujet demande d'écrire une fonction qui somme deux nombres en base B représentés sous forme de listes et que les candidats utilisent des boucles `for` et `List.nth` alors cela risque d'avoir un impact sur la lisibilité, la concision, voire la complexité du code résultant.
Pour toutes ces raisons, le jury rappelle que le style de programmation est libre mais conseille d'être capable d'un peu de souplesse sur ce style de programmation en s'adaptant au sujet (par exemple en utilisant plutôt des fonctions récursives sur les listes et plutôt des boucles et des `ref` sur les tableaux).

8.7 Évolutions envisagées pour l'édition 2026.

Aucune évolution importante n'est envisagée pour l'édition 2026 mais il est possible que des logiciels aient été mis à jour (passage de Debian Bookworm à Trixie).