Extraits des rapports et notices concrnant les oraux d'informatique de la filière MPI (hors X-ENS)

Oral pour Mines-Télécom

- Quelques minutes d'appropriation (sans prise de note) puis 30' de passage avec tableau 2 exercices à traiter
- Rapport du jury pour la session 2024 : pages 2 à 4

Oral pour CCINP

- 30' de préparation, 30' de passage avec tableau, machine et projecteur chaque élève doit traiter un exercice type A et un type B, soit un sur machine et l'autre non.
- Rapport du jury pour la session 2024 : pages 5 à 9
- Extrait du document fixant le cadre de l’épreuve orale d’informatique : page 10

TP CCMP

- 3h30 de TP avec plusieurs élèves en parallèle dans la même salle, avec un examinateur ou une examinatrice, + compte-rendu
- Rapport du jury pour la session 2024 : pages 11 à 20

TP Centrale

- proche du TP type Mines, mais la durée de l’épreuve est de 3 heures
- Rapport du jury pour la session 2024 : pages 21 à 25

4

BILAN DES COORDINATEURS DE L'ÉPREUVE
ORALE D'INFORMATIQUE

Galatée Hémery Vaglica et Samy Jaziri

L'épreuve orale consiste en la résolution d'un ou plusieurs exercices avec un temps d'appropriation du sujet de 15 minutes avant l'épreuve. Ce temps d'appropriation doit avant tout être utilisé par les candidats pour prendre connaissance des annexes et de l'ensemble du sujet.
Les sujets étaient composés de deux exercices sur des parties différentes du programme de première et de deuxième année.
Le premier exercice proposant souvent une application directe du cours et le second à la résolution guidée d'un problème. Les différents sujets sont de difficulté variable et la notation est ajustée selon la difficulté des exercices.

Le niveau moyen des candidats est bon dans la filière MPI au Concours Mines-Télécom. Même s'il y a toujours quelques exceptions, en grande majorité les candidats étaient globalement bien préparés à l'épreuve orale et ont su réagir aux remarques du jury et entrer en discussion avec lui.

Statistiques

FILIÈRE	NB CANDIDATS	MOYENNE	ECART-TYPE
MPI	361	11,81	3,966

Déroulement de l'épreuve

Le candidat se voit remettre le sujet et ses annexes 15 minutes avant le début de l'épreuve, dans une salle dédiée. Ce temps doit être mis à profit pour s'approprier les annexes, puis le sujet. Il est fondamental de lire l'intégralité des annexes et du sujet avant de commencer une phase optionnelle de résolution des questions. À noter que les candidats ne sont pas autorisés à prendre de notes durant ce moment d'appropriation.

À l'issue de ces 15 minutes, les candidats sont conduits jusqu'à leur salle d'interrogation. Le jury s'attend, à l'entrée dans la salle, à ce que le candidat soit familier avec l'ensemble du sujet, annexes comprises. Contrairement à ce que certains ont pensé, il n'est par contre pas attendu des candidats qu'ils aient déjà résolu certaines questions.

L'interrogation orale dure 30mn, incluant un temps incompressible de déplacements et formalités administratives.

En entrant dans la salle d'interrogation, le candidat remet à l'examinateur une pièce d'identité et la feuille d'émargement des examinateurs. Il est souhaitable que ces documents soient prêts à l'avance, tout temps passé à rechercher l'un d'entre eux au fond d'un sac va raccourcir le temps de l'interrogation.

C'est l'examinateur qui guide l'oral et, par défaut, il est attendu que le candidat aborde les différentes questions du sujet dans l'ordre. S'il est possible d'échanger avec le jury quant aux questions sur lesquelles le candidat se sent le plus à l'aise, seul l'examinateur peut autoriser le candidat à passer une ou plusieurs questions. L'examinateur peut, tout en informant clairement le candidat, considérer que ce dernier n'a pas été en capacité de résoudre une ou plusieurs questions s'il insiste pour passer à la suite.

Le jury possède une copie du sujet avec lui, il n'est pas nécessaire de relire l'intégralité de l'énoncé ou de présenter le sujet devant le jury si ce dernier a été bien compris lors de la phase d'appropriation. Si toutefois le candidat a un doute sur sa compréhension d'une partie de l'énoncé, il lui est conseillé d'en faire part au jury. S'il ne s'agit pas d'un manque de vocabulaire ou de connaissance lié au cours, le candidat ne sera pas pénalisé par cette démarche. Elle sera toujours préférable à la découverte, plus tard dans l'oral, d'un problème de compréhension.

Pendant l'oral, les candidats ont à leur disposition un tableau entier. Il est fortement conseillé aux candidats de profiter de cet espace et de bien l'organiser. Les schémas, graphes, arbres de preuves, etc. doivent être lisibles et pour cela les candidats ne doivent pas

hésiter à prendre de la place. Même si la majorité des candidats en sont conscients, le jury tient à rappeler qu'il ne faut rien effacer au tableau sans demander au préalable son aval. Enfin, il est important pour les candidats de trouver un bon équilibre entre ce qui doit être écrit au tableau et ce qu'il suffit de présenter à l'oral. Les preuves en particulier nécessitent une part de formalisation écrite et le jury se contentera rarement d'une simple description orale des grandes lignes.

À la fin de l'épreuve, le candidat rend le sujet à l'examinateur, efface le tableau et quitte la salle avec sa feuille d'émargement, sa carte d'identité et ses affaires.

Notation

Lors de l'épreuve orale sont évaluées, non seulement les connaissances en informatique, mais aussi le dynamisme et la capacité à interagir avec le jury, l'écouter et rebondir sur ses remarques et indications. Le jury déconseille fortement aux candidats de se murer dans le silence trop longtemps face à une question difficile. Tant qu'il ne fait pas d'erreur de cours ou de raisonnement grossière, le candidat ne peut que gagner à partager ses idées et ses pistes de résolution avec le jury lorsqu'il entre dans une phase de réflexion. Sur une question de cours dont le candidat aurait oublié la réponse, il est cette fois-ci préférable de répondre prudemment et ne pas essayer d'inventer la réponse, quittes à admettre son oubli. C'est au jury de déterminer la suite à donner à une telle réponse.
Le jury rappelle qu'en plus des connaissances et des capacités de résolution, la prestation orale est une part non négligeable de la note finale de l'examen. Rentre en compte, en particulier, l'attitude, le vocabulaire et l'élocution. Une attitude désinvolte ou désintéressée, un vocabulaire familier ou un manque de clarté dans l'explication sont autant de facteurs qui peuvent, s'ils sont répétés ou trop marqués, pénaliser le candidat. Il en va de même si le candidat tourne systématiquement le dos au jury ou regarde son sujet pendant la majorité de l'examen oral. En particulier, un candidat qui reste dos à l'examinateur face à son tableau en cachant ce qu'il écrit est pénalisé.

À noter enfin que les sujets peuvent être de longueur et difficulté variables. Certains ne peuvent pas être terminés en 30 minutes, même pour les meilleurs candidats. Il est donc tout à fait possible d'obtenir une excellente note, voire la note maximale, sans avoir traité l'intégralité des questions. À l'inverse, aller jusqu'à la dernière question du sujet ne présume rien quant à la qualité de l'oral, le jury se réservant le droit de ne pas relever certaines erreurs.

REMARQUES ET CONSEILS

- Le jury tient tout d'abord à rappeler que l'épreuve orale d'informatique balaie tout le programme de MP2I et MPI. Il peut être demandé au candidat, d'une manière adaptée au format oral de l'épreuve, de présenter des algorithmes, d'étudier et d'écrire du code en C ou en OCaml, ou encore des requêtes en SQL. L'écriture de code en C et Ocaml s'en tient généralement à des questions courtes et raisonnable à traiter au tableau.
- Le jury note une amélioration dans la variété des définitions et des notations utilisées par les candidats par rapport à l'année précédente. Les candidats semblent comprendre et utiliser le vocabulaire et les notations du programme de façon plus systématique et cela doit continuer ainsi.
- Le jury conseille aux futurs candidats de s'appliquer à faire des démonstrations précises et rigoureuses. Les candidats doivent savoir distinguer une récurrence d'une induction et les poser clairement. Les hypothèses doivent être formulées et vérifiées dans le cas initial et pour l'hérédité.
- Trop de candidats, bien que souvent vifs par ailleurs, ne sont pas capables de restituer avec exactitude un algorithme ou un théorème de cours, ni de l'appliquer ou de l'utiliser correctement. Le jury rappelle aux candidats que les preuves des théorèmes sont aussi à connaître. Le lemme de l'étoile est très rarement (moins d'une dizaine de candidats) restitué ou utilisé correctement. Les réductions sont très rarement faites correctement, alors même qu'elles sont guidées ou très simples, et même lorsque le candidat sait ce qu'est une réduction et dans quel sens l'effectuer. La connaissance du cours prenant une part importante dans la note, le jury encourage vivement les candidats à retravailler les algorithmes au programme et les démonstrations classiques pour cet oral.

Questions de cours

Le jury fournit une liste non exhaustive de questions de cours qui sont apparues cette année dans les exercices pour faciliter le travail de révisions des futurs candidats :

1. Compresser une chaîne de caractère exemple avec l'algorithme de Lempel-Ziv-Welch.
2. Rappeler le théorème de Cook-Levin.
3. Définir NP-complet.
4. Montrer l'indécidabilité du problème de l'arrêt.
5. Rappeler le principe de l'algorithme de Kruskal, sa complexité temporelle et les structures de données utilisées pour une implémentation efficace.
6. Décrire un algorithme de tri efficace. Donner et justifier sa complexité temporelle.
7. Montrer que pour tout langage régulier, il existe une grammaire hors contexte qui l'engendre.
8. Construire un arbre de Huffman pour la compression d'une chaîne de caractère exemple.
9. Rappeler le fonctionnement de l'algorithme de Dijkstra à l'aide d'un pseudo-code.
10. Qu'est-ce qu'un tri topologique ?
11. Comment construire un automate reconnaissant l'intersection des langages reconnus par deux automates donnés ?
12. À quel besoin répond l'algorithme ID3 ? Décrire brièvement le principe.
13. Décrire un algorithme permettant de déterminer les composantes fortement connexes d'un graphe orienté.
14. Énoncer et démontrer le lemme de l'étoile.

MPI

CONCOURS COMMUN INP

RAPPORT DE L'ÉPREUVE ORALE D'INFORMATIQUE

PROPOS INTRODUCTIFS

Le sujet de l'oral d'informatique CCINP en spécialité MPI est constitué de deux exercices :

- un exercice de type A "théorique", noté sur 8 points et ne nécessitant pas l'utilisation d'un ordinateur. L'exercice A peut évaluer la syntaxe SQL mais ni la syntaxe C ni la syntaxe OCaml ;
- un exercice de type B "pratique", noté sur 12 points et consacré principalement à de la programmation en C ou en OCaml. Quelques questions d'analyse peuvent également être posées. Cet exercice fournit généralement un code compagnon que le candidat devra compléter, corriger ou modifier. Il est également possible que le candidat doive écrire intégralement le code demandé par l'énoncé.

L'oral, d'une durée totale d'une heure, se décompose en deux temps :

- 30 minutes de préparation du sujet sur feuille et machine, démarches administratives incluses ;
- 30 minutes de passage avec un examinateur, démarches administratives incluses.

Ce rapport rappelle les consignes principales, qui sont susceptibles d'évoluer. Se référer à la notice du Concours pour obtenir l'ensemble de l'information et au site du concours https://www.concours-commun-inp.fr/fr/epreuves/les-epreuves-orales.html rubrique MPI. Vous trouverez le cadre de l'épreuve (dont l'environnement technique sous format fichier .ova).
Également, restent disponibles les 12 « exercices 0 » qui furent proposés à la rentrée 2022-23, en amont de la première session Oraux 2023 de la filière MPI, afin que les candidats se préparent au mieux à cette nouvelle épreuve.

En complément à ce rapport, une sélection de sujets, commentés et partiellement corrigés, posés lors de la session 2024 est publiée, ainsi que l'archive correspondante contenant les codes compagnons, les codes corrigés et le fichier source LaTeX.

PRÉPARATION DU SUJET

La préparation s'effectue dans une salle dédiée. Plusieurs candidats préparent en même temps, dans des conditions favorisant la concentration, sous la surveillance de vacataires. À l'entrée de la salle de préparation, convocations et pièces d'identité sont vérifiées.

Pour préparer, le candidat dispose :

- d'un ordinateur disposant de l'environnement diffusé sur le site du concours à la rubrique MPI[1] et d'un clavier (configuration AZERTY par défaut),
- de feuilles de brouillon,
- des énoncés imprimés des deux exercices (types A et B),

[1] https://www.concours-commun-inp.fr/fr/epreuves/les-epreuves-orales.html

- d'une calculatrice,
- d'une clé USB contenant le code accompagnant l'exercice de type B.

Le candidat doit copier le code compagnon disponible sur la clé sur sa machine pour pouvoir le compléter / modifier, puis copier le code produit dans un répertoire dédié sur sa clé USB en fin de préparation. Les consignes quant aux modalités de préparation et notamment la manipulation des clés sont projetées dans la salle de préparation, de même qu'un chronomètre indiquant le temps de préparation restant. Ce dernier est fixé à 27 minutes, les 3 minutes restantes étant réservées aux formalités administratives et au déplacement vers les salles d'interrogation.

En fin de préparation, les candidats sont accompagnés jusqu'aux salles d'interrogation par un vacataire.

PASSAGE DU CANDIDAT

Le candidat rentre dans la salle qui lui est indiquée, fournit à l'examinateur sa feuille de passage et sa pièce d'identité puis signe la feuille d'émargement que lui présente l'examinateur. Le chronomètre est lancé à ce moment, pour un temps de passage effectif de 27 minutes.

Le candidat dispose d'un ordinateur prêt, dans la même configuration que celui qu'il a utilisé dans la salle de préparation. Un vidéoprojecteur permet de présenter son travail à l'examinateur. Le candidat est invité à copier le code présent sur sa clé sur la machine mise à disposition.

Le candidat commence par présenter tout ce qu'il a préparé. Il débute par l'exercice de son choix et peut, à tout moment, changer d'exercice et/ou revenir sur l'exercice précédemment abordé. Il dispose pour sa présentation d'un tableau et de son code source projeté. L'examinateur demandera que le code produit soit compilé / exécuté / interprété. L'examinateur peut donner des indications quant au temps restant et, le cas échéant, indiquer au candidat sur quelles questions se concentrer.

En fin d'épreuve, après la fin du temps imparti, le candidat remet à l'examinateur les énoncés des deux exercices, ses brouillons et la clé contenant le code produit. Il ferme sur l'ordinateur les fenêtres de son code, efface le tableau avant de récupérer sa pièce d'identité et sa feuille de passage signée par l'examinateur.

CRITÈRES D'ÉVALUATION

Sont pris en compte dans l'évaluation :

- la maîtrise des éléments du programme relatifs aux deux exercices,
- la maîtrise de l'environnement technique et des outils mis à disposition,
- la pertinence de la réflexion,
- l'interactivité lors de la présentation des résultats et des tests (exercice de type B),
- la réactivité aux éventuels conseils et indications de l'examinateur,
- la qualité de la prestation orale.

BILAN

Sur 684 candidats admissibles (dont 104 grands admissibles), 526 se sont présentés aux épreuves orales. La moyenne des notes obtenues est de 11,50 avec un écart-type de 3,48.

Les examinateurs ont proposé des exercices couvrant l'ensemble du programme des deux années de MPI. En particulier, les thématiques suivantes ont été abordées : la récursivité, la représentation de types, les structures de données (listes, tableaux, piles, files, tas, etc.), les arbres, les graphes, les algorithmes d'approximation, les algorithmes probabilistes, les stratégies algorithmiques (diviser pour

régner, programmation dynamique, retour sur trace, etc.), l'algorithmique des textes, la concurrence, la décidabilité, la NP-complétude, les langages réguliers, les automates finis, les grammaires non contextuelles, les algorithmes d'apprentissage, l'algorithmique des jeux, la déduction naturelle ou encore les bases de données.

Le langage des exercices de type B était imposé : environ 50 % des exercices devaient être traités en OCaml et 50 % en C. La syntaxe SQL a, quant à elle, été évaluée via certains exercices de type A. Tous les exercices qui ont été donnés ont été entièrement ou quasiment entièrement résolus par au moins un candidat.

Les examinateurs ont pu voir d'excellentes prestations et sont satisfaits du niveau global des candidats. La majorité des candidats maîtrisent les langages OCaml et C de manière satisfaisante. Les candidats ont dans l'ensemble été très bien préparés à cette épreuve aux modalités exigeantes.

CONSEILS AUX CANDIDATS

Nous souhaitons souligner les points suivants, à l'attention des futurs candidats.

Conseils quant à la forme de l'épreuve

- Certains candidats se sont présentés devant l'examinateur sans avoir leur convocation et pièce d'identité à disposition et ont ainsi perdu du temps à fouiller leur sac pour les retrouver. Nous rappelons que les formalités administratives sont incluses dans les 30 minutes de temps de passage.

- Savoir éjecter proprement une clé fait partie des compétences attendues d'un élève de MPI. Nous rappelons donc qu'il ne faut pas arracher une clé USB sans précaution, en particulier alors que des documents qui y sont présents sont encore utilisés.

- Certains candidats ont mal copié le code qu'ils avaient produit pendant la préparation sur leur clé (ou ont arraché leur clé causant des erreurs d'écriture). La responsabilité leur en incombe et ils ont dû reprendre l'exercice B depuis le début pendant le passage.

- Un sujet est composé de deux exercices et il n'est pas possible de faire de transfert de points entre les deux. Un candidat qui réussit brillamment l'exercice A et n'aborde pas l'exercice B ne pourra avoir une note supérieure à 8/20.

- Il est recommandé aux candidats de commencer par annoncer, en début d'épreuve, les questions des deux exercices qui ont été traitées en préparation. Ceci permet à l'examinateur d'aider les candidats à gérer leur temps et de les interroger sur l'ensemble des questions abordées pendant la phase de préparation.

- L'examinateur connaît le sujet, il n'est donc pas nécessaire pour le candidat de le réintroduire.

- Il n'est pas toujours judicieux de commencer par l'exercice A. De plus, il est possible de passer d'un exercice à l'autre pendant le passage, voire de ne pas traiter toutes les questions dans l'ordre (ou même d'en sauter quelques-unes).

- Le candidat peut aborder des questions de code (exercice de type B) non traitées en préparation pendant le passage en les programmant devant l'examinateur sur la machine de la salle ou en présentant ses idées pour réaliser une telle implémentation au tableau.

- Il est nécessaire de bien lire les énoncés et en particulier celui de l'exercice B. Un nombre non négligeable de bons candidats a perdu du temps avec une lecture trop rapide et incorrecte des types imposés par un énoncé, voire en réimplémentant des fonctions qui étaient explicitement fournies dans le code compagnon, voire sans lire du tout un des énoncés.

- Il est conseillé aux candidats de bien prendre connaissance des deux exercices : certains énoncés sont difficiles à traiter à l'oral sans y avoir un peu réfléchi en amont.

- Un nombre faible mais non négligeable de candidats a été dans l'incapacité de compiler / exécuter / interpréter leur code en OCaml arguant qu'ils ne savaient utiliser aucun des outils mis à disposition. L'environnement utilisé était pourtant connu bien en amont des épreuves (voir https://www.concours-commun-inp.fr/fr/epreuves/les-epreuves-orales.html, rubrique MPI).

- Les examinateurs encouragent les candidats à utiliser le `Makefile` proposé pour les programmes en C, dès la phase de préparation. Lors du passage à l'oral, l'examinateur peut demander aux candidats d'utiliser ce `Makefile` à l'aide de la commande `make safe`, qui permet à l'examinateur de vérifier certains aspects de la gestion de la mémoire par le candidat. Rappelons que le contenu du `Makefile` ainsi que les consignes d'utilisation, qui sont rappelées au début de chaque exercice de type B utilisant le langage C, sont disponibles et connus en amont des épreuves.

- Le test des fonctions demandées par l'exercice B fait partie de l'évaluation et l'examinateur peut demander à voir le résultat de ces tests pendant le passage. Pour les exercices de type B en OCaml, les candidats qui interprétaient à la volée et au fur et à mesure leur code étaient avantagés sur ce point par rapport à ceux qui le compilaient, ces derniers devant effectivement se doter de fonctions d'affichage, ce qui peut leur prendre du temps (par exemple pour afficher une valeur de type `int option list`).

- Il a été constaté chez certains candidats une vitesse de frappe très lente, en particulier lors de l'utilisation de l'ordinateur pendant le passage à l'oral. Cela n'a pas été pénalisé, mais ces candidats se sont retrouvés mécaniquement désavantagés. Il est à noter que les claviers des machines sont par défaut en configuration AZERTY et que les candidats doivent y être habitués ou bien savoir en changer.

Conseils quant au fond de l'épreuve

- Les fonctionnalités des langages au programme ainsi que tout ce qui se trouve dans la rubrique "éléments de syntaxe devant être reconnus et utilisables après rappels" peuvent être librement utilisés par les candidats. Si un candidat utilise une construction des langages en dehors de ce cadre, il pourra lui être demandé d'expliquer comment répondre à la question avec uniquement les outils du programme. L'examinateur n'interdit pas, mais ne souhaite en aucun cas encourager, l'utilisation d'éléments hors programme : tous les sujets peuvent être résolus avec le fragment au programme des langages et utiliser des constructions alternatives expose le candidat à des questions qui peuvent lui faire perdre du temps sans que sa réponse ne soit davantage valorisée.

- De nombreux candidats ont été mis en difficulté par des d'erreurs et des bogues dans leurs programmes en C, davantage qu'en OCaml. S'il n'y a pas le temps dans le cadre d'une épreuve aussi courte d'utiliser des solutions de débogage complexes, il est essentiel que les candidats pensent et sachent mettre en œuvre des solutions simples pour trouver l'origine des comportements incorrects observés.

- La syntaxe SQL est généralement maîtrisée mais de nombreux candidats ont eu de grandes difficultés avec la modélisation de problèmes et le modèle entité-association. Même si aucun

formalisme n'est spécifié par le programme, des diagrammes entités-association simples font partie des attendus que les candidats doivent savoir mettre en œuvre. Il en va de même pour les schémas relationnels et leur lien avec le modèle entité-association.

- De nombreux candidats oublient de prendre en compte des cas de bases vides pour les objets construits par induction (arbres, listes, etc.).

- On observe encore de trop nombreuses confusions entre listes et tableaux en OCaml avec des tentatives non pertinentes d'accéder à un élément d'une liste via son indice.

- La structure d'arbre binaire de recherche est mal connue et parfois confondue avec celle des tas binaires. La notion d'arbres binaires de recherche équilibrés a posé de nombreux problèmes aux candidats.

- Même si elle n'est pas explicitement au programme, la fonction `Printf.printf` en OCaml a un fonctionnement très similaire à celui de la fonction `printf` en C et est particulièrement pratique pour un oral, que ce soit comme outil simple de débogage ou pour valider simplement un programme par un affichage d'exemples. Il est conseillé aux candidats de s'être familiarisés à son usage pour pouvoir l'utiliser avec aisance lorsque l'énoncé en rappelle les modalités d'usage et les codes de formatage.

- L'usage de l'opérateur OCaml @ pour ajouter un élément en tête d'une liste est vivement déconseillé. Dans ce cas précis, elle ne permet généralement au candidat que de montrer un manque de maîtrise et de compréhension de la syntaxe OCaml.

- De nombreux candidats oublient que la pile est de taille limitée et qu'un trop grand nombre d'appels récursifs peut la faire déborder.

- Un algorithme aussi essentiel et central que le parcours de graphe n'est souvent pas maîtrisé.

Les connaissances et les compétences générales[1] attendues sont :

- connaître et maîtriser son cours, les concepts associés et les langages de programmation (*OCaml* et *C*) et de requête (*SQL*),

- maîtriser les définitions et les différents algorithmes du cours,

- s'approprier l'information et le contexte d'un problème,

- savoir hiérarchiser ses connaissances des techniques de l'informatique,

- proposer ou mettre en œuvre des méthodes pour la résolution d'un problème,

- mener à son terme la démarche de résolution du problème par les questions posées,

- concevoir une solution, un algorithme ou une structure de données,

- vérifier la pertinence des réponses ou justifier (preuve, validation de code, …),

- pouvoir faire part des directions prises pendant la préparation,

- réagir aux questions de l'examinateur et savoir mettre à profit ses indications,

- exposer clairement, précisément et concisément ses raisonnements et ses résultats.

Les connaissances et les compétences attendues spécifiques à l'exercice A sont :

- modéliser un point de vue avec les objets conceptuels de l'informatique (table, graphe, arbre, automate, …) ou le spécifier avec un formalisme,

- savoir appliquer sur papier un algorithme sur un exemple simple,

- savoir écrire des requêtes en SQL,

- savoir démontrer de manière théorique des énoncés informatiques ; par exemple, la terminaison, la correction (partielle ou non), la complexité d'un algorithme, des propriétés sur des structures de données, etc.

Les connaissances et les compétences attendues spécifiques à l'exercice B sont :

- savoir exploiter un environnement technique (ordinateur et logiciel),

- développer ou amender un algorithme ou une structure de données dans un langage de programmation (C ou OCaml selon la nature de l'exercice),

- pratiquer une programmation sûre par une validation formelle ou expérimentale rigoureuse,

- établir une validation de code et écrire un jeu de test,

- savoir appliquer la programmation défensive et discuter l'invalidité des données d'un programme.

[1] Ces éléments ne se veulent pas exhaustifs. Se référer au programme CPGE *Informatique – MP2I-MPI, 2021*

8 Épreuve d'Informatique – Filière MPI

Ce chapitre présente le compte rendu de la session 2024 de l'épreuve de TP d'informatique.
Son objectif est de présenter les modalités du TP et de donner des conseils pour que les candidats puissent au mieux préparer les prochaines années.
Beaucoup de « défauts » sont listés, mais le jury précise qu'il a été satisfait de la prestation des candidats. En 2024, le jury a trouvé que les candidats étaient mieux préparé qu'en 2023, année où le jury avait déjà trouvé que le niveau était bon, à la fois sur le programme et les technique que sur la manipulation des outils.

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 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 répondre ou pas) ;
- il est demandé aux candidats de tester leurs codes et la notation prend en compte ces tests ;
- les candidats doivent rédiger un compte rendu ;
- 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 maximum 4 candidats par salle. Les candidats posent leurs sacs dans un coin de la salle puis s'installent à un des postes disponibles. Les candidats ont eu le droit à quelques minutes de familiarisation avec la machine.
Épreuve principale.
Vient ensuite l'épreuve. 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 présenté.
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. Ceux qui ont une question peuvent appeler les examinateurs sans attendre que ceux-ci ne passent.

Attention, certains candidats tentent de sauter certaines parties sans 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 exactement 3h30. Quelques minutes avant la fin les examinateurs rappellent l'imminence de la fin de l'épreuve pour que les candidats s'assurent que le compte rendu est bien à jour et les examinateurs passent une dernière fois pour noter où en sont les candidats. Il n'est pas demandé de mettre leurs fichiers à un endroit spécifique à la fin de l'épreuve.

Précisions quant aux tenues vestimentaires.

De nombreuses matières du Concours Commun Mines Ponts insistent dans les rapports sur la tenue correcte exigée. Entre le stress sur 3h30 d'épreuve et les salles peu ou pas climatisées les candidats peuvent avoir chaud... Les candidats peuvent venir en tenue légère si des températures chaudes sont prévues et ceux qui viennent en costume-cravate peuvent retirer la veste ou la cravate pendant l'épreuve.

Précisions quant à l'écriture du compte rendu.

À l'inverse des autres TPs où le compte rendu a un rôle central dans le suivi des expériences et la compétence de remplissage d'un compte rendu est attendue, 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, en fonction de cela, 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 remplissent 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 celle-ci insuffisante. Les réponses fausses (mauvais calcul de complexité ou algorithme faux par exemple) peuvent être prises en compte dans la notation mais si le jury demande de compléter une réponse trop courte ou lorsqu'il demande à être plus concis, cela n'est pas pris 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 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 "Q1 : 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). Le jury insiste sur le fait que les candidats n'ont pas à écrire des réponses longues, si l'examinateur trouve la réponse trop laconique, il peut toujours demander au candidat de préciser ;
- 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 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 le rapport du concours 2023 étaient moins présents en 2024. Cette partie va néanmoins va beaucoup répéter le contenu du rapport 2023 car ces défauts persistent chez certains candidats.

Architecture du code.

Beaucoup de candidats voient chaque question comme un bloc unitaire et semblent avoir du mal à voir une question comme la combinaison de plusieurs sous-problèmes qui peuvent être implémentés dans plusieurs fonctions avec des tests pour chaque. 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. Ce temps perdu sera souvent regagné ensuite...

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. Il serait intéressant d'être en mesure, à la fin des années de CPGE, de prendre l'initiative sur un problème complexe. Plusieurs sujets étaient assez ouverts et les jurys continueront de poser des sujets ouverts les années prochaines, ces sujets posent particulièrement problèmes aux élèves qui ont du mal à prendre des initiatives ou à avoir une posture réflexive par rapport à ce qu'il faudrait coder.

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 :

- cela permet aux examinateurs de valider la compétence test qui figure au programme de la MPI, notamment dans le fait d'écrire des tests pertinents qui essaient de couvrir les différents cas possibles ;
- cela permet ensuite aux examinateurs de vérifier plus facilement la correction des programmes écrits par les candidats. En effet, les examinateurs lisent souvent rapidement les programmes écrits par les candidats et les tests (s'ils sont bien écrits) renforcent la confiance dans la correction ou non du code ;
- enfin les tests permettent aux candidats d'avoir confiance (de façon justifiée !) 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 toute particulièrement mauvaise impression. Les candidats devraient savoir que tout programmeur peut faire des bugs et c'est encore plus vrai pour des candidats sortants de MPI qui n'ont que peu d'expérience.

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).
Il est parfois utile de réfléchir à comment écrire des tests rapidement ou intelligemment. Si on ne peut pas se contenter de tester un algorithme que sur des exemples triviaux, ces exemples peuvent tout de même servir de test car ils testent bien les cas de base. S'il est demandé, par exemple, d'écrire une fonction qui décompose un graphe en ses composantes connexes, on peut facilement tester sur 5 graphes de la façon suivante : on a 2 graphes triviaux (un graphe avec 2 nœuds, connectés ou non) puis 3 graphes plus compliqués (par exemple un graphe à 8 nœuds et sa copie avec une puis deux arrêtes en plus dans la copie ce qui fait passer le graphe de 3 à 1 composante connexe).
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. Si, par exemple, il faut tester des algorithmes de graphe, on peut définir des graphes dans le code principal et les utiliser à chaque question pour tester. Il est aussi conseillé d'utiliser des assert, de cette façon on garde une trace des tests utilisés (pour les montrer au jury), on peut plus facilement faire beaucoup de tests d'un coup et ces tests sont facilement copiables d'une question à la suivante ;

– enfin, 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 plus compliqués 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 chercher 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 c'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. Il ne faut pas hésiter à mettre des dizaines d'assert dans un code ;

– 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 (par exemple un entier positif ou un tableau trié), il ne faut pas hésiter (quand c'est pertinent) à mettre un assert pour le vérifier et ainsi détecter plus rapidement les bugs plus loin dans le programme (si cette fonction est réutilisée) ;

– 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. 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.

Cette année le concours n'a pas utilisé de machine virtuelle mais une image était fournie sur le site du concours et l'environnement du concours était assez proche en terme de configuration. Nous encourageons vivement les candidats à tester la machine virtuelle proposé 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.

Cette année, les candidats avaient quelques minutes avant le début de l'épreuve où ils pouvaient utiliser 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.

Le jury essaie de fournir, dans la mesure du possible, les logiciels libres et populaires, et on a vu de nombreux logiciels utilisés comme VScodium, emacs, gedit, nano, vim. Le jury continuera à proposer un large choix d'éditeurs de texte ainsi que plusieurs outils adaptés (des "modes" ocaml dans ces éditeurs et des shells adaptés comme `utop`). Cela étant, il est dommageable que beaucoup de candidats ne savent programmer qu'avec un outil très spécifique et sont déroutés à l'oral lorsque l'environnement ne leur propose pas ou que ce n'est pas compatible avec un sujet.

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. De la même manière, il était recommandé dans un sujet de ne pas taper manuellement l'entrée à chaque test mais de rediriger l'entrée standard (la syntaxe `./monProgramme < monFichier` était donnée dans le sujet) et très peu d'élèves ont utilisé cette commande... qui était pourtant là pour aider les candidats !

Pour finir rappellons que les candidats vont passer un certain temps dans le terminal. Sans que le jury n'attende une grande familiarité avec l'outil, certains candidats gagneraient beaucoup de temps en connaissant 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 envie de 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`.

Notons au passage que les candidats passent souvent autant de temps à relire ou débugguer leur code qu'à l'écrire, il est donc probable que le temps économisé à mettre des variables à une seule lettre soit

souvent perdu dans la relecture qui est plus difficile.

8.3 Commentaires liés au programme.

Il est difficile de se prononcer sur la compréhension du programme par les candidats car, bien que l'intégralité du programme a été traité dans les sujets posés, chaque point du programme n'était généralement traité que dans un petit nombre de sujets et chaque sujet n'était posé qu'à un petit nombre de candidats.

Par rapport à 2023, le jury a rencontré moins de candidats qui avaient fait des impasses complètes sur un pan du programme. Le conseil du jury est qu'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 qu'il faut aussi connaître toutes les définitions et tous les algorithmes du programme (sans forcément les avoir implémentés ou en maîtriser toutes les subtilités). Il est très dangereux de faire une impasse complète sur un point du programme car un sujet entier peut porter sur un point du programme.

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 et un polycopié de C.

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.

8.4 Commentaires liés au langage SQL.

Cette année, la plupart des candidats arrivaient à résoudre les questions de SQL mais certains le font beaucoup plus rapidement que d'autres et donc la distinction entre candidats se fait plus sur la vitesse à résoudre les questions. Nous conseillons donc aux candidats de s'entraîner à résoudre rapidement des problèmes de SQL.

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, de nouveau, été favorablement surpris par la maîtrise du langage C qu'ont en moyenne les candidats. 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. Comme décrit plus haut dans la section gestion des bugs, le jury recommande fortement aux candidats de compiler avec les options `-Wall -Wextra` et `-fsanitize=address` car cela permet d'attraper de 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).

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

Comme annoncé dans le rapport 2023, le jury ne peut pas facilement vérifier que la mémoire allouée est bien libérée et donc, même si ce n'est pas une bonne pratique de programmation, 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.

À l'inverse, le jury a pris en compte, pour une petite partie de la note, la qualité du code (pertinence des noms de variables et fonctions, commentaires, tests et assert, lisibilité) et continuera à la prendre en compte pour les prochaines années.

8.6 Commentaires liés au langage OCaml.

La maîtrise du langage OCaml par les candidats est assez bonne mais le jury a tout de même 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 bien les connaître permet souvent d'écrire du code plus court donc plus rapide à écrire et plus simple à relire. Le jury a aussi peu vu 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 donne un véritable avantage.

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 et qu'un candidat propose de répondre avec `Set`, il serait probablement demandé au candidat de recommencer sans (ou alors il n'obtiendrait qu'une partie des points sur cette question). Ce cas ne s'est pas présenté, mais, si un candidat veut utiliser de telles fonctions avancées, il convient de 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 pourraient rapidement les ré-implémenter (et que ce n'était pas le cœur du problème posé). Le jury ne conseille pas spécialement l'apprentissage de ces fonctions car il ne pense pas que l'utilisation de ces fonctions apportent un véritable avantage aux candidats. 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 et les candidats vraiment très à l'aise avec a bibliothèque standard gagnent du temps.

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 (ou même parfois connaître) la complexité de cette opération ! Même quand les candidats utilisent une fonction de la bibliothèque dans un algorithme, ils doivent être capables de donner la complexité de l'algorithme.

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. Au contraire, c'est plutôt un bon réflexe et comme lire la documentation n'est pas forcément évident et il est donc conseillé d'avoir déjà de l'expérience avec la documentation OCaml. En effet, des sujets peuvent nécessiter l'utilisation de bibliothèques, auxquels cas les candidats auront à lire la documentation.

Utilisation du tri de la bibliothèque.

Le cas du tri est un peu particulier. Les candidats pensent souvent à des algorithmes qui utilisent des tris même quand ceux-ci ne sont pas strictement nécessaires. C'est un réflexe algorithmique louable que de se ramener à un problème plus simple en utilisant un algorithme connu mais cela pose un problème : `List.sort` n'est pas au programme (et donc pas toujours connu), le tri fusion (assez facile à implémenter) n'est pas explicitement au programme et l'implémentation d'un tas est un exercice long et difficile pour beaucoup de candidats. Bien qu'il ne soit pas au programme et qu'il ne rentre pas dans la case "rapidement implémentable", le jury autorise l'utilisation et conseille fortement de connaître `List.sort` car il est difficile pour le jury de donner des points à l'implémentation d'un tri qui n'était pas requis... et certains perdent beaucoup de temps à implémenter un simple tri fusion.

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 2025.

Le format de l'édition 2025 devrait être très similaire à celui de 2024, la principale évolution souhaitée par le jury est l'utilisation d'outils qui vérifient le code des candidats de façon automatique à l'aide de tests. Aucune connaissance sur de tels systèmes de test ne sera nécessaire mais les élèves qui veulent s'entraîner à ce type d'épreuves peuvent utiliser des plateformes comme France-IOI.

Travaux pratiques d'informatique

Présentation de l'épreuve

L'épreuve de travaux pratiques d'informatique en MPI est une épreuve d'algorithmique et de programmation, avec utilisation d'un ordinateur, d'une durée de 3 heures. À partir d'un sujet imposé, elle demande de traiter sur machine des questions de programmation, d'effectuer des choix de modélisation et d'aborder certains aspects théoriques de l'informatique, tout en communiquant très fréquemment avec le jury.

L'objectif de l'épreuve est d'évaluer les capacités de programmation, la maitrise des méthodes classiques du programme, les capacités de modélisation, d'abstraction et d'inventivité ainsi que l'application de bonnes pratiques que l'on est en droit d'attendre de futurs ingénieurs.

Organisation de l'oral

Les candidats disposent d'un ordinateur fourni par le concours, configuré avec un environnement de travail adapté aux sujets demandés. Pour la session 2024, l'environnement choisi était à nouveau, comme en 2023, la distribution GNU/Linux Ubuntu 22.04, construite de façon analogue à l'environnement Pronaos (https ://gitlab.com/agreg-info/clef-agreg/).

Avant le début de l'épreuve, le jury a laissé plusieurs minutes aux candidats afin de prendre en main librement cet environnement de travail. Pendant ce temps, qui n'entre pas en compte dans l'évaluation, il a été possible de poser toutes les questions nécessaires aux membres du jury. Le jury compte reconduire ce temps de 10 minutes de prise en main de l'environnement lors des sessions à venir.

Les sujets ont été fournis durant l'épreuve en version papier ainsi qu'en version numérique au format PDF. Les sujets étaient tous accompagnés de fichiers auxiliaires, comprenant notamment :

— des fichiers sources, complets ou à compléter selon les cas ;

— des fichiers de données à exploiter ;

— des scripts d'aide à la compilation.

Pour cette session, ces fichiers ont été mis à disposition dans un dossier de travail spécifique à chaque candidat, dont l'accès est déverrouillé grâce à un code donné en début d'épreuve par le jury. Les candidats pouvaient travailler directement dans ce dossier, il ne leur était demandé aucun transfert de fichier.

Les candidats sont responsables des sauvegardes de leur travail, qu'ils doivent effectuer à intervalles réguliers.

Les sujets sont organisés en une suite de questions, à traiter généralement de façon linéaire. Ces questions sont réparties en trois catégories :

— des questions de programmation ;

— des questions à préparer pour une présentation orale ;

— des questions de rédaction à réaliser sur un compte-rendu.

Le concours fournit toujours les feuilles et les brouillons utiles à l'épreuve. Les candidats doivent se munir du matériel d'écriture usuel (stylos, crayons, gomme, règle). Aucun autre matériel n'est autorisé. Pour certaines applications numériques demandées par les sujets, les candidats ont convenablement réussi à utiliser les fonctions de calcul des ordinateurs.

Les sujets proposés suivent tous le format des épreuves en vigueur depuis la session 2023.

Analyse globale des résultats

Les prestations des candidats sont dans leur très large majorité d'excellente qualité. Le jury constate une hausse sensible du niveau par rapport à la session 2023, que ce soit sur les compétences en programmation ou sur les compétences théoriques.

Le niveau observé en programmation est très satisfaisant. Les programmes produits respectent un grand nombre de critères de qualité (organisation, présentation, conventions de nommage) et la syntaxe des langages est bien maitrisée.

Les connaissances de cours sont également globalement complètes, mais le jury a constaté çà et là plusieurs faiblesses récurrentes. Le jury sur le fait que le contenu du cours de MP2I et de MPI doit être connu, au point d'être restitué efficacement. Les sujets continueront de proposer des questions de cours, sans lequel il est vraiment difficile d'aborder correctement les questions de programmation qui suivent. Ces questions de cours ont été la source de différences marquées dans l'évaluation des candidats.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Programme des épreuves orales

Les sujets posés permettent d'évaluer l'ensemble des notions présentes dans les programmes d'informatique des classes de MP2I et de MPI. Chaque sujet porte sur une ou plusieurs parties spécifiques de ces programmes. L'ensemble de tous les sujets couvrait globalement l'intégralité du programme des deux années.

Les langages évalués sont `C`, `OCaml` et `SQL`. Les sujets peuvent utiliser, selon la pertinence par rapport au thème abordé, un seul, deux ou trois langages (mais jamais `SQL` seul). La très grande majorité des sujets évalue au minimum deux langages. Le langage à utiliser pour chaque question est quasiment toujours imposé, mais certaines parties ont parfois été laissées librement au choix des candidats.

Évaluation des épreuves orales

Les sujets proposés restent volontairement longs. Les candidats les plus efficaces ont pu tout de même aborder la quasi totalité du contenu des sujets proposés. Les sujets comportent tous des parties d'application immédiate du cours, assez guidées, destinées à évaluer les compétences de base en programmation et les connaissances de cours, ainsi que des parties d'ouverture favorisant la prise d'initiative et permettant d'évaluer l'autonomie et la capacité de prise de recul des candidats.

L'évaluation de l'épreuve ne consiste cependant pas exclusivement, et loin de là, en une mesure du plus grand nombre de questions traitées. Le jury a valorisé non seulement l'efficacité en terme de programmation mais également l'intérêt porté à la discipline de programmation (compiler fréquemment, s'assurer que le code fonctionne, proposer spontanément des tests). L'épreuve de travaux pratiques d'informatique doit mener à un programme qui fonctionne effectivement. Le jury préfère des prestations avec un peu moins de questions traitées mais dans lesquelles les programmes écrits sont correctement compilés et soigneusement testés.

Les candidats étaient, pour cette session encore, bien préparés à l'épreuve. Ceci a permis lors des épreuves d'avoir des échanges entre le jury et les candidats de très haute qualité. Le jury a constaté que dans la très grande majorité des cas, les compétences visées par les programmes de MP2I et MPI étaient acquises, que les aspects théoriques étaient maitrisés et que la technique était manipulée avec efficacité. Les candidats

ont parfaitement démontré leurs aptitudes de futurs ingénieurs en informatique. Le jury adresse à nouveau ses félicitations à l'ensemble des candidats.

Évolution du format de l'épreuve

Le format de l'épreuve a permis d'évaluer correctement le niveau des candidats, et il n'est pas prévu d'évolution majeure pour la session 2025.

Le jury envisage de limiter la rédaction de questions sur le compte-rendu, qui servira uniquement de support pendant l'interaction orale. Les questions à noter sur le compte-rendu seront relues directement pendant la séance, et ne seront pas évaluées après l'épreuve. Le jury ne demande pas de rédaction très poussée comme on pourrait en trouver sur une copie à l'écrit, d'éventuelles imprécisions pouvant être levées par la discussion orale. Les candidats disposeront toujours de feuilles de brouillon ainsi que d'un support papier afin de préparer leurs réponses et d'indiquer certains résultats demandés.

Publication des sujets

Afin de renforcer la préparation à cette épreuve, le jury publie un nouvel échantillon de sujets qui ont été donnés lors de cette session.

Maitrise du cours

Le jury considère que la maitrise du cours demeure un élément essentiel : il encourage les candidats à bien apprendre le cours afin de traiter rapidement et efficacement les questions qui s'y rapportent. Les questions de cours ont souvent été discriminantes, d'autant plus que certains manques de connaissances ont été bloquants pour traiter l'approche proposée ensuite par le sujet. Dans de tels cas, le jury a été contraint de faire des rappels de cours très importants, ce qui se ressent nécessairement sur la note obtenue. À titre d'exemple, une très grande majorité de candidats n'est pas capable d'expliquer correctement en quoi consiste une table de hachage, ne parvenant pas à reconnaitre ni à compléter les implémentations proposées.

Le jury a constaté plusieurs confusions, où des candidats confondent des algorithmes de recherche de motifs avec des algorithmes de compression, ou bien confondent les méthodes algorithmiques usuelles (diviser pour régner, programmation dynamique, etc.) sans en connaitre vraiment le principe.

Le jury invite les candidats à faire preuve de précision dans l'utilisation des noms d'algorithmes au programme, dans une optique d'efficacité de la communication. La simple confusion de noms n'a cependant pas été sanctionnée quand il était clair que les candidats maitrisaient l'algorithme demandé et savaient le décrire précisément. À contrario, redonner simplement en tant que mot-clé le nom d'un algorithme sans en connaitre le principe n'a pas été valorisé.

Préparation technique

L'environnement de travail du concours permet d'évaluer l'intégralité des compétences prévues par les programmes de MP2I et MPI. Le jury constate que tous les candidats semblaient, pour cette session, mieux préparés à cet environnement. Ils ont su en maitriser tous les aspects sans aucune difficulté.

Le jury a fourni, avant l'épreuve et pendant l'épreuve, les indications nécessaires aux candidats pour faire fonctionner cet environnement, en particulier sur des points non exigibles des programmes de MP2I et MPI (aide à l'invocation des compilateurs, rappels sur le système de modules en `OCaml` lorsque le sujet fournissait plusieurs modules, utilisation des scripts de compilation fournis, aide à l'accès aux outils de détection d'erreurs tels que l'address sanitizer, l'activation des avertissements du compilateur, etc.).

Afin de maintenir le bon niveau de préparation constaté cette année, *le jury encourage de nouveau toutes les formations à proposer à leurs élèves, durant leur scolarité, un environnement de travail adapté au programme de MP2I et MPI*, permettant au minimum d'aborder les points suivants :

— l'accès à un shell dans un terminal permettant d'invoquer les compilateurs mais également d'autres commandes, et de manière générale permettant de se familiariser avec le fonctionnement d'une ligne de commandes ;

— le fonctionnement d'un système POSIX, en particulier quant à la gestion des processus, des fils d'exécution, des flux standard et leur redirection, éventuellement avec des tubes ;

— la familiarisation avec des outils de compilation usuels (`make` ou `dune`). Leur usage fera toujours l'objet d'un rappel et aucune compétence spécifique n'est attendue, mais avoir déjà rencontré ces outils permet une plus grande efficacité.

Le jury rappelle l'indication suivante des programmes de MP2I et MPI : « Bien que ces notions soient indépendantes du système d'exploitation, le système Linux est le plus propice pour introduire les éléments de ce programme. »

Remarques concernant `OCaml`

Le jury maitrise la bibliothèque standard du langage mais il n'exige pas que les candidats utilisent d'eux-mêmes des fonctions de haut niveau, par exemple celles provenant du module `List`. L'évaluation est par exemple indifférente à l'utilisation, qui n'est ni valorisée ni sanctionnée, de `List.fold_left` (et autres fonctions similaires). Le jury est surtout attentif à la clarté du code produit, à sa correction et à la maitrise de ce code par les candidats.

Il est pertinent de réfléchir quelques instants à la manière de concevoir un code, pour éviter que le résultat soit inutilement compliqué et soit source d'erreurs difficiles à détecter et corriger.

Certains sujets invitaient à utiliser dans un même programme des aspects fonctionnels et des aspects impératifs du langage, ce qui a déstabilisé plusieurs candidats ou bien a conduit à la rédaction de codes très compliqués. Le jury rappelle qu'un `match` ou un `if` sont des expressions comme les autres en OCaml et qu'il est parfaitement admis d'utiliser ces constructions partout où une expression est permise par le langage (membre droit d'un `let`, corps d'une boucle). Il est cependant impératif de bien connaitre les priorités des constructions et des opérateurs et de savoir parenthéser correctement (avec les mots-clés `begin-end` ou avec des parenthèses). Le jury conseille de systématiquement introduire ce parenthésage en présence conjointe de la construction `if then else` et de l'opérateur point-virgule (`;`), les erreurs de priorité à ce niveau ont posé des problèmes à de très nombreux candidats, engendrant souvent une importante perte de temps.

Les candidats doivent savoir proposer un programme `OCaml` complet, qui compile dans son intégralité et pas uniquement ligne par ligne dans un REPL. Une aide à la compilation a systématiquement été fournie. Le jury accepte si nécessaire le double point-virgule (`; ;`) pour séparer les phrases, même si ce dernier ne fait pas partie du langage à proprement parler ; il le propose aux candidats quand cela aide parfois à mieux cerner certaines erreurs de syntaxe difficiles à situer d'après le message du compilateur. Sur ce dernier point, le jury attire l'attention des candidats sur le fait que `;` est un opérateur binaire et non pas un terminateur d'instruction.

Les candidats doivent savoir identifier les erreurs de syntaxe dans leurs programmes, celles-ci se situent souvent bien avant la ligne à laquelle est indiquée l'erreur.

Remarques concernant `C`

La gestion de la mémoire est un aspect maitrisé par de nombreux candidats, mais les oublis d'allocations avec `malloc` et surtout les oublis de libération avec `free` ont été trop fréquents. Il est toujours pertinent de s'interroger sur la politique d'allocation de la mémoire, en particulier lorsque le sujet invite à programmer une petite API manipulant une structure de données, dont on se sert ensuite. Avoir une politique claire d'allocation *et de libération* est essentiel.

Lors de la préparation des candidats, il peut être utile d'indiquer comment compiler un programme avec `gcc -g -Wall -fsanitize=address` et d'interpréter la sortie en cas d'erreur de segmentation ou de libération oubliée. La compilation séparée a fait l'objet de rappels, mais le jury souligne que celle-ci doit être reconnue par les candidats qui sont donc supposés être un minimum familiers avec ce procédé.

Remarques concernant `SQL`

Le niveau de maitrise est très hétérogène parmi les candidats interrogés. Il est globalement satisfaisant. Le jury attire cependant l'attention sur le fait que plusieurs candidat semblent avoir fait l'impasse sur ce langage. Ces candidats ont tenté de construire des requêtes à partir d'exemples extraits de la documentation mise à disposition sur les machines. Il en a résulté des prestations décevantes :

— cette stratégie a permis de construire les requêtes les plus simples, au prix d'une énorme perte de temps ;

— elle conduit fréquemment à utiliser des éléments hors programme et non compris des candidats. En particulier, certains candidats ont tenté d'utiliser en `SQLite` des fonctions trouvées dans le manuel de `MariaDB`, sans comprendre pourquoi le moteur refusait la requête.

Les sujets proposent des requêtes de niveaux différents. Le jury s'attend à ce que les requêtes les plus faciles puissent être traitées par tous les candidats, notamment en ce qui concerne :

— l'usage des fonctions d'agrégation avec ou sans `GROUP BY` ;

— l'usage de jointures ;

— l'usage de `LIMIT` et `OFFSET`.

Le jury félicite les candidats qui ont réussi à construire des requêtes pour quelques questions particulièrement alambiquées.

Conclusion

Le jury a été globalement été très satisfait par l'ensemble des prestations des candidats. Ces derniers ont été très bien préparés aux modalités de cette épreuve. Le jury tient à nouveau à féliciter l'ensemble des candidats ainsi que leurs enseignants pour leur engagement.

Pour les sessions suivantes, il attire l'attention sur l'importance de la compréhension et de la maitrise du cours, sur la nécessaire prise de recul quant aux notions abordées pendant l'année et sur l'intérêt d'une pratique très régulière sur machine tout au long du cursus.