LANGAGES DE PROGRAMMATION
IFT-3000
Hiver 2023

Travail pratique 1 - TRAVAIL INDIVIDUEL
Pondération de la note finale : 14.5%

À remettre, par le biais du site Web du cours, et avant 23h59, le dimanche 5 mars 2023

1 Énoncé

Ce travail pratique consiste à implanter en Ocaml une liste de fonctions utiles pour la gestion d'un bassin de cours universitaires et d'un ensemble de programmes ; il fait suite à 2 travaux pratiques réalisés dans le cadre du cours en hiver 2021. Plus précisément, les fonctions de ce Tp portent essentiellement sur la cohérence d'un programme universitaire donné (seules les 2 premières fonctions, «eq_pr» et «prerequis», n'ont pas de lien avec la cohérence d'un programme).

Pour réaliser ce travail, vous disposez d'une page Github qui comprend les éléments suivants :

1. Un fichier README.md, affiché automatiquement lors de l'ouverture de la page web, et qui résume les étapes à suivre pour réaliser le Tp ;
2. Un dossier «lib» comprenant les fichiers du Tp :
   (a) «tp1.mli» : définit la spécification des éléments du Tp, soit la signature des fonctions à programmer ; notez que ce fichier comprend toute la documentation nécessaire à la réalisation du Tp (vous n'avez pas à modifier ce fichier) ;
   (b) «tp1.ml» : correspond à la partie «implantation» du Tp ; c'est à ce niveau que vous devez programmer les différentes fonctions du Tp (le barême y est aussi précisé) ; notez que vous ne devez remettre que ce fichier (**tp1.ml**), une fois complété ;
   (c) «dune» : un fichier utilisé par l'outil *dune* (vous n'avez pas à modifier ce fichier) ;
3. Un dossier «test» comprenant un testeur pour le Tp :
   (a) «testeur.ml» : comprend la liste des cas de tests définis pour les différentes fonctions à implanter dans le cadre du Tp ;
   (b) «dune» : un fichier utilisé par l'outil *dune* (vous n'avez pas à modifier ce fichier) ;
4. Un dossier «gcp/lib» comprenant les fichiers correspondant aux travaux pratiques réalisés en hiver 2021 :
   (a) «gcp.mli» : comprend la spécification des différentes structures de données utilisées (qui seront utilisées dans le cadre de votre travail), ainsi que la signature des fonctions exportées par le module ;
   (b) «cours.ml» : comprend la définition d'une liste, *bcours*, comprenant la description de 220 cours (tous les cours définis dans nos programmes ainsi que leur chaine de prérequis sont présents dans ce fichier) ; à la fin du fichier, on retrouve aussi la définition d'une liste associant des abréviations à des noms de faculté, ainsi qu'une liste associant des abréviations à des noms de départements ou d'écoles ;
   (c) «programmes.ml» : comprend la définition de 3 de nos programmes : baccalauréat en informatique ; baccalauréat en génie logiciel ; et certificat en informatique ;
   (d) «dune» : un fichier utilisé par l'outil *dune* (vous n'avez pas à modifier ce fichier) ;
5. Un dossier «docs» comprenant la documentation générée automatiquement à partir des commentaires qui se trouvent dans les fichiers «gcp.mli» et «tp1.mli». Cette documentation est accessible via le lien suivant : https://bktari.github.io/ift3000-h23 ;

6. Un dossier «.vscode» comprenant un fichier *json* permettant de s'assurer que vous chargerez la bonne version de l'interpréteur sous *VSCode* ;
7. Un ensemble d'autres fichiers de configuration ; pour votre information :
   — «.gitignore» est utilisé par *Git* ;
   — «.ocamlformat» est utilisé par l'outil *ocamlformat* et permet de formatter votre code ; il comprend la version qui est censée être installée dans votre système ;
   — «dune-project» est utilisé par l'outil *dune* ;
   — «.ocamlinit» comprend des commandes qui seront automatiquement exécutées au lancement d'un interpréteur (*utop* ou *ocaml*) ; le but étant de charger automatiquement tous les fichiers produits par la compilation et l'édition de lien et d'ouvrir les différents modules (seul le module «Tp1» requerra de l'ouvrir explicitement : «open Tp1 ;;»).

Au sujet du bassin de cours À l'instar des Tps réalisés en hiver 2021, dans ce Tp, un bassin de cours est représenté par une liste de cours ; chaque cours est composé d'une paire d'éléments : le premier élément correspond au numéro de cours (une chaine de caractères qui, typiquement, comprend, séparés par un tiret, un sigle composé de 3 caractères, et un nombre composé de 4 chiffres) ; le deuxième élément correspond à la description du cours. Cette dernière correspond à un enregistrement qui comprend différents attributs (ou champs), comme le titre du cours, le nombre de crédits, ses préalables, etc. En particulier, on retrouve un attribut nommé «equiv» qui comprend éventuellement le numéro d'un cours jugé équivalent (par ex., dans l'attribut «equiv» des cours "IFT-2008" et "GLO-2100", on retrouve respectivement les valeurs "GLO-2100" et "IFT-2008" ; ces 2 cours sont ainsi considérés équivalents ce qui permet, pour certains traitements (essentiellement en hiver 2021), d'avoir des résultats cohérents). De même, on retrouve, dans la description d'un cours, un attribut «dom» qui permet de préciser les domaines de savoir traités dans un cours. Mis à part cet attribut, les autres attributs se retrouvent dans la description officielle «UL» de chaque cours, comme par exemple celle d'IFT-3000 : https://www.ulaval.ca/etudes/cours/ift-3000-langages-de-programmation.

Pour les autres structures de données définies dans la partie «spécification» du module Gcp (fichier «gcp.mli»), leur définition est assez claire ; notez cependant les quelques remarques suivantes :

— le type algébrique *credit* permet de distinguer les crédits associés à un cours régulier des crédits qu'on associe à un stage (cette distinction a notamment été utilisée dans la fonction «total_cr» - voir l'implantation de cette fonction dans le fichier gcp.ml) ;
— le type *session* comprend un constructeur («Annee») qui permet d'éviter de préciser les 3 sessions d'une année ;
— de même, le type *fe* comprend un constructeur («FEX») qui permet d'éviter de préciser toutes les formules d'enseignement ;
— le type *prealables* permet de représenter les préalables d'un cours ; par exemple, le préalable du cours "GIF-3101", soit «ET [CP "GIF-1003" ; CRE 57]», précise que pour pouvoir suivre ce cours, il faut avoir réussi "GIF-1003" et avoir réussi 57 crédits ; de même, le préalable associé au cours "GLO-4001", soit «ET [OU [CP "IFT-1004" ; CP "GLO-1901"] ; OU [CP "STT-1000" ; CP "STT-1900" ; CP "STT-2920"]]» précise qu'il faudra avoir réussi à la fois un des 2 cours "IFT-1004" et "GLO-1901", et un des 3 cours "STT-1000", "STT-1900" et "STT-2920".

Au sujet des programmes d'études Dans le fichier «gcp.mli», figure la définition d'un programme d'études, selon le règlement des études (comme indiqué ci-dessous, quelques simplications ont tout de même été apportées) ; ainsi, un programme est défini comme un tuple comprenant les éléments suivants :

— le type du programme, le titre du diplôme, du grade ou de l'attestation, ainsi qu'une liste de sessions d'admission (à l'instar des Tp réalisés en hiver 2021, dans ce Tp, vous n'aurez pas à manipuler ces 3 premières valeurs du tuple) ;
— les deux valeurs qui suivent correspondent aux règles définissant les activités de formation obligatoires, ainsi que celles correspondant aux autres exigences («cours optionnels») ; chacune de ces valeurs correspond à une paire comprenant le nombre de crédits requis (resp. pour la partie «obligatoire» et la partie «optionnelle»), ainsi qu'une liste de règles ; chaque règle comprend un nom et une liste de cours ou de motifs (*pattern*) de cours ; on distingue les cours obligatoires (constructeur «CoursOB»), qui correspondent à des cours ayant un sigle et un numéro, des cours au choix (dans un intervalle de crédits n_1 et n_2) qui

peuvent soit correspondre, comme pour les cours obligatoires, à des cours (ayant un sigle et un numéro), soit correspondre à une liste de cours ou de motifs de cours à éviter (exclus) ;
— puis on retrouve une liste de concentrations (définies avec un titre, un nombre total de crédits à atteindre et une liste de règles à respecter).

À l'instar des Tps réalisés en hiver 2021, les notions de «profil», «majeure» et «mineure» ne sont pas traitées dans ce Tp.

2 Lien avec les travaux de l'hiver 2021

Dans le cadre de ce travail, relativement à ce qui a été fait en 2021, vous allez surtout utiliser :
— la liste *bcours* définie dans le fichier «cours.ml» ; elle est notamment utilisée par le testeur ; vous pourrez aussi l'utiliser, au niveau de l'interpréteur, pour tester certaines des fonctions que vous allez implanter ;
— de même pour les programmes *b_ift*, *b_glo* et *c_ift*, définis dans «programmes.ml» ;
— certaines fonctions exportées par le module «Gcp» ; à titre d'information, voici les seules fonctions que nous retrouvons dans le corrigé du travail et qui proviennent des travaux de l'hiver 2021 : «ret_descr» et «respecte_motif». Évdemment, vous pouvez utiliser autant de fonctions que voulu parmi celles exportées par le module *Gcp* ;
— les structures de données définissant les cours et leurs attributs ainsi que les programmes et leurs attributs.

Plus précisement, la présence du module *Gcp* vous permet de manipuler les structures de données (*cours*, *programme*, etc.) du Tp sans même avoir débuté l'implantation des fonctions du Tp ; il vous suffit d'utiliser les fonctions déjà présentes et fournies par ce module (à la fin du fichier «gcp.ml» figure un ensemble d'exemples d'utilisation de ces fonctions). Aussi, le code source du module étant fourni, vous pourrez ainsi plus rapidement aborder votre Tp et la programmation fonctionnelle en OCaml. Finalement, notez que comme indiqué dans le fichier «tp1.mli», pour toutes les fonctions pouvant soulever une exception, cette dernière le sera via une fonction invoquée du module *Gcp* (dans le cas, bien sûr, que vous faites appel minimalement à des fonctions de ce module (comme c'est le cas dans le corrigé - voir ci-dessus)).

3 Consigne à respecter

Mis à part l'utilisation des exceptions (même si, une nouvelle fois, vous n'êtes pas censé en utiliser car l'appel à certaines fonctions du module *Gcp* fera en sorte que des exceptions seront soulevées), ce Tp devra être programmé en utilisant que la partie «fonctionnelle» du langage OCaml (fonctions d'ordre supérieur, filtrage par motifs et récursion) ; donc, vous ne pouvez utiliser les références, ni les structures de contrôle (matières liées à la partie «impérative» du langage).

4 Démarche à suivre

Comme indiqué dans le fichier «README.md», il s'agit essentiellement de compléter une première fonction, puis de la tester grâce au testeur, ou via vos propres tests à l'aide d'un interpréteur ; puis de passer à la prochaine fonction jusqu'à terminer votre Tp.

Au niveau de la correction, nous allons nous assurer que :

1. le contenu du fichier «tp1.ml» que vous allez remettre respecte ce qui est spécifié dans «tp1.mli» ;
2. votre travail passe les différents tests (nous utiliserons d'autres cas de tests) ;
3. vos différentes fonctions sont correctement implantées (revue de code).

Notez qu'on utilisera la version du fichier «tp1.mli» remise avec l'énoncé (de même pour «gcp.mli» et «gcp.ml»).

5 À remettre

Il faut remettre un fichier .zip contenant uniquement le fichier "tp1.ml" complété. Le code doit être clair et bien structuré [1].

1. Suggestion : http://ocaml.org/learn/tutorials/guidelines.html

6 Remarques importantes

Plagiat : Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances. Tous les cas seront référés à la direction de la Faculté.

Travail remis en retard : Tout travail remis en retard sera refusé et obtiendra la note 0 ; désolé, aucune tolérance car le corrigé sera rendu public, aussitôt la date limite de remise passée. La remise doit se faire par la boite de dépôt du TP1 dans la section «Évaluation et résultats».

Barème : Au niveau du code à implanter, le barème est précisé dans le fichier "tp1.ml". Notez cependant que :

- (-10pts), si vous ne remettez pas un unique fichier nommé "tp1.ml".
- (-10pts), si votre code ne compile pas, ou n'est pas conforme à sa spécification.
- tout code en commentaire ne sera pas corrigé.

Bon travail.