Utilisation de l'outil informatique

Table des matières

CHAPITRE 1

Présentation

Si tu donnes un poisson à un homme, ce soir il sera rassasié ;
Si tu lui apprends à pécher, toute sa vie il sera rassasié.

1.1. Objectifs

L'objectif de ce cours est de vous amener au plus vite à l'autonomie face à l'outil informatique, de façon à ce que vous puissiez :

(1) Choisir en connaissance de cause les outils adaptés à vos besoins ;
(2) Progresser par vous-même en apprenant à utiliser de nouveaux langages, de nouveaux outils ;
(3) Comprendre les enjeux du monde informatique

Vaste programme ? Pas tant que ça en fait, à partir du moment où
- vous connaissez les concepts essentiels ;
- vous savez chercher l'information.

1.2. Organisation

Il y aura en tout 3 ou 4 TPs qui seront à rendre à dates régulières : je les regarderais pour voir quelles difficultés vous avez pu rencontrer et, le cas échéant, vous donner un coup de main. Ils ne seront pas notés. Certains de ces TPs sont longs, de façon à donner du grain à moudre à ceux qui ont déjà une certaine expérience. Essayez d'en faire le maximum, mais ne vous inquiétez pas outre mesure si vous n'arrivez pas à les terminer.

Dans les dernières séances, vous travaillerez sur un questionnaire et sur un projet de programmation que vous finirez dans les semaines qui suivent. Votre note sera établie en fonction de la qualité du document contenant vos réponses et sur ce projet. Comme les niveaux de chacun sont très hétérogènes, j'essayerais de tenir compte des progrès effectués autant que du niveau réel.

Les premiers TP seront effectués en java, sous `GNU/Linux`. Par la suite, vous pourrez utiliser le langage et l'environnement de programmation de votre choix, du moment que ce choix est motivé, et fait en connaissance de cause. Personnellement, mon environnement de programmation préféré est actuellement :

Système d'exploitation: `GNU/Linux`

Langages: `C++`, `MuPAD`, `perl`

Compilateur: gcc et consorts

Éditeur de texte: Emacs

Déboguer: gdb et consorts, sous emacs

Gestionnaire de version: `CVS`

Interpréteur de commande: `zsh`

Traitement de texte: LaTeX, LyX

Navigateur: `konqueror`

Moteur de recherche: `google`

Environnement graphique: `KDE`

Vous aurez deviné qu'en dehors de ces cours je suis aussi un défenseur des logiciels libres, mais ceci ne doit pas fausse"r votre réflexion propre.

CHAPITRE 2

Conventions pour les TPs et projets

2.1. Objectifs

Nous allons fixer un ensemble de règles et de conventions dans la façon d'écrire d'organiser les programmes que vous nous rendrez dans le cadre des TP et des projets d'informatique du DESS, pour les modules Outils Informatique, Remise à Niveau Informatique et Système et Réseaux. Les TP et projets qui ne suivent pas ces règles seront rejetés.

Le but de ces règles est d'une part d'obtenir des programmes plus faciles à relire, à modifier et à déboguer, et d'autre part de simplifier les échanges de programmes, entre vous, et entre vous et nous. L'objectif est avant tout de vous faire prendre de bonnes habitudes de programmation, ce qui vous fera gagner beaucoup de temps à l'usage.

Attention, nous n'affirmons certainement pas que ces règles sont uniques et les meilleures au monde ; certaines de ces conventions sont en effet purement arbitraires. Toute communauté de développeurs se créé, par nécessité, de telles conventions pour rendre possible le travail en équipe. Dans certain cas, ces règles sont précisément codifiées dans un document (en anglais un *Coding Standard*). Dans d'autres cas, il s'agit plutôt d'un folklore oral informel. Dans tous les cas, il est important au moment d'intégrer une équipe d'apprendre et de suivre les règles qui y ont cours. Ceci n'a d'ailleurs rien de bien spécifique à l'informatique : ■à Rome, agit comme les romains■. Bien entendu, il est toujours possible de proposer des améliorations de ces règles, une fois que l'on en a bien compris les motivations ! Aussi, n'hésitez-pas non plus à demander le pourquoi du comment de ces règles.

Certaines de ces règles nécessitent des outils que vous n'avez probablement jamais utilisés auparavant. Pas de panique, il n'y a rien de bien difficile, et on vous expliquera au fur et à mesure comment s'en servir. En particulier, on vous montrera comment automatiser la plupart des tâches.

2.2. Les règles

À part la toute première séance, vous utilisez systématiquement le logiciel CVS pour gérer les versions de vos programmes.

Vous rendrez vos TPs et projets sous la forme d'une archive au format `.tar.gz` de votre répertoire de travail, que vous recopierez dans le répertoire `~nthiery/TP/nomDuTP/`. Le répertoire de travail lui-même sera architecturé comme suit :

```
nomDuTP-Nom.Prénom-Version/
   README:     Descriptif de l'archive
   Makefile:   Descriptif des règles de compilation
```

```
rapport.html ou rapport.pdf: Compte rendu
bla.java:   Sources du programme
...
```

- `nomDuTP` est de la forme OI-TP1, RN-TP5, SR-TP3 pour respectivement les TPs d'Outil Informatique, de Remise à Niveau et de Système et Réseaux.
- Le fichier README contient la date, une description courte du contenu de l'archive, une liste des fichiers avec description courte de leur rôle respectifs.
- S'il y a plusieurs auteurs, le fichier `AUTHORS` contient la liste des auteurs. Note : pas plus de trois personnes par groupes !
- Sauf exception, le Makefile contient ce qu'il faut pour que :
  - `make` ou `make all` lance la compilation du (des) programme(s) ;
  - `make demo` lance une démonstration du (des) programme(s) ;
  - `make check` lance la batterie de tests ;
  - `make doc` fabrique la documentaion html avec javadoc ;
  - `make clean` nettoie les fichiers temporaires.

  Dans le cas contraire, le fichier `README` spécifie explicitement comment faire à la main les actions correspondantes.
- Chaque fichier source contient un entête précisant la révision, le nom de l'auteur et une description succincte.
- Chaque fonction et classe est accompagnée d'une documentation succincte. Pour les fonctions java, cette documentation est au format javadoc. Cette documentation décrit les préconditions éventuelles, le résultat, la liste des exceptions que la fonction peut renvoyer, ainsi que quelques exemples d'utilisation. Si l'algorithme utilisé n'est pas trivial, la documentation inclue des explications ou des références.
- Pour chaque fonction, il y a une série de tests dans la batterie de tests. Ces tests doivent au moins inclure les cas extrêmes triviaux, quelques cas simples, et quelques cas plus élaborés.
- Le code est indenté de façon systématique et uniforme.
- Sauf dans les cas triviaux (par exemple `i` pour un compteur d'une boucle simple), les variables ont des noms longs et significatifs. Si le nom n'est pas complètement explicatif, ou s'il y a des contraintes supplémentaires sur la variable, cela est documenté à l'endroit où la variable est définie.
- Les invariants de boucle et les assertions non triviales seront documentées, et testée avec assert.
- Les constantes sont déclarées de manière groupées, par exemple en entête de fichier.
- La pollution de l'espace de nom principal doit être limité au maximum : pas de variables globales, utilisation de package, ...
- Lorsqu'une analyse des résultats obtenus est demandée, ou lorsque vous le jugez utile, vous pouvez joindre un compte rendu succinct, au format `HTML` ou `PDF.`
- Pour ceux qui veulent aller plus en avant, je conseille fortement d´écrire tous les commentaires en anglais. Je n'adore pas particulièrement l'anglais, mais si vous étiez amenés à échanger vos programmes avec des programmeurs étrangers, il sera probablement plus facile pour eux de lire de l'anglais que du français. Par contre, autant que faire ce peux, la documentation utilisateur doit être disponible en français.

CHAPITRE 3

Mardi 30/09/2003, mercredi 1/10/2003 : TP1, bases d'UNIX, CVS

À rendre pour le mercredi 01/10/2003.

3.1. Le shell, ou comment faire du lego sous UNIX

– Tout programme a une entrée standard, une sortie standard et une sortie d'erreur
– Exemples : echo, cat, grep, perl, tee, locate, more / less, sort, wc

```
perl -n -e '/bla/ and print'
perl -n -e '/^ (\d+) \s+ (\w+) /x and print "$2 $1\n"'
```

– On peut combiner ces briques de base en redirigeant les entrées et sorties dans des fichiers avec `<`, `>`, `>!`, `>>`, `2>`, ou dans des tuyaux avec `|`.
– Pour les détails, voir les manuels de ces commandes (man echo, man cat), et les références !

3.1.1. Références. Il existe sur internet pleins de bons sites de référence sur l'utilisation d'UNIX. Voir par exemple, en français, `http://www.eleves.ens.fr/tuteurs/`, `http://lea-linux.org` (et en particulier `http://lea-linux.org/admin/shell.php3`), ou `http://www.newtolinux.org.uk/tutorials/shell_fr.shtml`.

3.2. Utilisation de CVS (Concurrent Version System)

3.2.1. Introduction. Voici un texte de ■propagande■ que j'avais écrit aux US pour un tutoriel que j'y avais donné sur CVS :

Have you ever made a modification to a document, and later wondered if it was not better before ? Have you ever broken a working program, without knowing how to undo your changes ? Have you ever told your sponsor "That's really too bad, we cannot make you a demonstration. We are right now in an experimental phase, and our program is not working". Or simply, have you ever destroyed accidentally an article just before the deadline (of course it was Sunday, and there was no system administrator around to get a backup) ?

If you answered yes to any of those questions, spending one hour right now to learn how to use CVS might save you many hours and headaches in the future. CVS means Concurrent Version System ; it is a version control system which can record the history of your files (usually, but not always, source code). CVS only stores the differences between versions, instead of every version of every file you've ever created. CVS also keeps a log of who, when and why changes occurred, among other aspects.

3.2.2. Utilisation.

3.2.2.1. *Configuration de cvs.* On commence par définir la variable d'environnement CVSROOT qui indique à CVS l'emplacement de son archive : ici on a choisi le sous-répertoire CVSROOT du répertoire utilisateur. La façon de faire dépends du système :

– Sous windows : `set cvsroot=D:\myhome\CVSROOT` (sous Windows)
– Sous UNIX, avec sh/bash/zsh/...: `export CVSROOT=$HOME/CVSROOT`
– Sous UNIX, avec csh/...: `setenv CVSROOT $HOME/CVSROOT`

En fait, sur les machines Linux de Gerland, `cvs` devrait déjà être configuré convenablement ! Vous pouvez faire `echo $CVSROOT` pour vérifier.

La toute première fois, il faut initialiser l'archive avec : `cvs init`

3.2.2.2. *Gestion des modules.* Pour créer un nouveau module, le plus simple est de créer un répertoire avec le nom du module dans l'archive avec : `mkdir $CVSROOT/essai`. C'est la seule fois où vous modifierez l'archive directement, le reste du temps c'est `cvs` qui s'en chargera ! En fait, même là on serait sensé utiliser `cvs` pour créer le module avec la commande `cvs import`, mais nous ne rentrerons pas dans ces détails.

Pour utiliser ce module, il faut en créer une copie de travail avec : `cvs checkout essai` (en bref `cvs co essai`). Maintenant vous pouvez travailler dans le répertoire `essai` comme dans tout répertoire. Vous remarquerez qu'il contient un sous-répertoire CVS qui contient des informations utiles pour `cvs`. Normalement on a jamais besoin d'y toucher ! Lorsque l'on a fini son travail, c'est une bonne habitude de détruire la copie de travail après l'avoir archivé. Cela se fait avec : `cvs release -d essai` depuis le répertoire parent.

3.2.2.3. *Gestion des fichiers d'un module et archivage.* Allez dans la copie de travail de `essai`, et créez-y quelques fichiers texte avec votre éditeur préféré.

Très rapidement, la copie de travail sera encombré de fichiers temporaires qui n'ont pas besoin d'être archivés (`machin.class`,...). Il faut donc indiquer à `cvs` la liste des fichiers à archiver. Cela se fait avec la commande : `cvs add fichier` ; vous pouvez donner un descriptif au fichier avec `cvs add -m'Source de la classe Bidule' Bidule.java`. La même commande sert à ajouter des sous-répertoires. Si, plus tard, un fichier devient inutile, vous pouvez l'indiquer à `cvs` avec la commande `cvs remove fichier`. Il faut avoir détruit le fichier au préalable !

Aussi souvent que vous le souhaitez, vous pouvez demander à `cvs` d'archiver la version courante d'un fichier avec : `cvs commit fichier` (en bref `cvs ci fichier`). `cvs` vous demande alors de rentrer un descriptif des modifications. Vous pouvez archiver tout les fichiers dans le répertoire courant d'un seul coup avec : `cvs commit` (en bref `cvs ci`).

Quand archiver ? Idéalement :

– Il n'y a dans l'archive que des versions stables (en particulier qui passent tous les tests),
– Entre deux versions successives, il n'y a qu'une seule modification (cette modification peut toucher plusieurs fichiers !)

C'est pas toujours facile à faire, et il peut arriver que l'on ait besoin de faire des compromis.

3.2.2.4. *Utilisation de l'archive.* C'est bien d'archiver des versions, mais encore faut-il pouvoir utiliser cette archive !

Vous pouvez demander les informations sur un fichier avec `cvs log fichier`. Pour demander les différences par rapport à une ancienne révision : `cvs diff -r1.2 fichier`. C'est une des commandes les plus importantes de `cvs` : si vous avez fait des modifications et vous remarquez qu'il y a un nouveau bogue, vous avez immédiatement la liste exacte des modifications ; cela permet de localiser très finement les sources potentielles du bogue.

À tout moment, vous pouvez revenir à une ancienne révision avec : `cvs update -r1.2 fichier`. C'est utile pour faire une démonstration lorsque la version courante est cassée ! Il est possible d'apporter des modifications à une telle ancienne version, mais cela demande de créer une nouvelle branche de révisions ; nous ne rentrerons pas dans ces détails ici. Pour revenir à la toute dernière version, tapez : `cvs update -A fichier`. Si vous voulez juste consulter une ancienne révision vous pouvez aussi faire : `cvs update -r1.2 -p fichier | less`.

3.2.2.5. *Tags.* Lorsqu'une version vous paraît importante, vous pouvez lui donner un nom avec la commande `cvs tag essai-1_0`. Vous pouvez ensuite utiliser ce nom dans toutes les commandes au lieu du numéro de révision. C'est d'autant plus pratique que ce numéro de révision varie d'un fichier à l'autre. Par exemple, vous pouvez créer une copie de travail fraîche avec la version 1.0 de votre logiciel avec : `cvs checkout -r essai-1_0 essai`.

3.2.2.6. *Pour aller plus loin.* Ce ne sont que les commandes essentielles de `cvs`. Il y a beaucoup d'autres commandes qui sont bien utiles. En particulier, un des atouts de `cvs` est de faciliter le travail de plusieurs personnes sur le même projet. Je recommande de consulter, par exemple, les références suivantes :

Documentation en ligne: `info cvs`

Sites web de CVS: `http://www.nongnu.org/cvs/`

CVS pour windows: `http://www.wincvs.org`

CVS pour windows en ligne de commande: `http://www.hpcc.ecs.soton.ac.uk/hpci/tools/cv`

Tutoriels :`http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/cvs/`**:**
`http://www.loria.fr/~molli/cvs/cvs-tut/cvs_tutorial_toc.html`

3.2.2.7. *Comment est-ce que cela marche ?* Soyez curieux, allez voir le contenu des répertoires `CVS` et de l'archive dans `CVSROOT` !

3.3. Sujet du TP

Créez deux fichiers `bla1` et `bla2` avec votre éditeur favori (par exemple emacs). Utilisez la commande `cat` pour concaténer les deux fichiers, et mettre le résultat dans `bla`. Rajoutez une ligne contenant `coucou` à la fin fichier `bla`, (indication : utiliser `>>`).

Récupérez un gros fichier de texte (cf. par exemple sur http ://www.promo.net/pg/ `http://www.promo.net/pg/`). Comptez le nombre de lignes et de mots dans le texte ; comptez le nombre d'occurrences d'un mot clef de votre choix (indication : utilisez `grep` et `wc` ; on pourra se contenter de compter le nombre de lignes contenant le mot clef). Regardez le contenu du fichier avec la commande `less` (indication : h donne accès à l'aide de `less`) ; à l'intérieur de `less`, utilisez `/mot` pour recherchez les mots clefs.

Explorez le système de fichier. Par exemple, visualisez le contenu du fichier `/etc/passwd`, listez tous les programmes dans `/usr/local/bin`, ou recherchez toutes les images au format gif du système (indication : utilisez `locate`).

Suivez le tutoriel `http://lea-linux.org/dev/shell_script.php3` pour écrire de petits scripts.

Récupérez l'archive suivante, et décompressez-la (indication : utilisez les commandes `gunzip` et `tar`).

`http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/OI-TP1-Nicolas.Thiery-1.0.tar.gz`

Vous obtenez un répertoire de la forme `OI-TP1-Nicolas.Thiery`. Entrez dedans, et essayez les différentes commandes `make`, `make demo`, `make check`, `make doc`. Utilisez votre navigateur préféré pour visualiser les fichiers `.html` créés. De même, utilisez `kdvi` (par exemple) pour visualiser les fichiers `.dvi`, et `gv` pour les fichier `.ps`, et `.pdf`. Enfin, utilisez `make clean`, pour faire le nettoyage.

Toutes ces commandes utilisent le fichier `Makefile`. Regardez son contenu, et essayez de reproduire les commandes qu'il contient à la main.

Maintenant, vous allez personnaliser ce répertoire. Faites en une copie sous le nom `OI-TP1-Prenom.Nom-1.0` (OI pour Outil Informatique) dans votre répertoire principal (Indication : utiliser `cp -p`). Utiliser votre éditeur préféré pour modifier le contenu des fichiers `Makefile` et `README` de façon conforme au cahier des charges. Faites aussi quelques modifications aux fichiers LaTeX (avec votre éditeur), et LyX (avec `lyx`).

Vous allez passer ce répertoire sous `CVS`, afin de pouvoir archiver les différentes versions au fur et à mesure. Créez un module `DESS`, et demandez en à `CVS` une copie de travail dans `~/DESS`. Déplacez le répertoire dans cette copie de travail sous le nom `OI-TP1`. Enregistrez le avec `cvs add`, ainsi que tous les fichiers importants dedans. Enfin utilisez `cvs commit`, pour tout archiver. Travaillez encore un peu dessus, archivez la nouvelle version, utilisez `cvs diff` pour voir les différences, etc.

Quand tout est fini, utilisez tar et gzip pour fabriquer une archive `OI-TP1-<Prenom>.<Nom>-<Version>.tar.gz` de votre répertoire de travail.

Enfin copiez cette archive dans le répertoire `~nthiery/TP/OI-TP1/`.

Deux petites notes :

(1) Comme indiqué dans les conventions, le répertoire contenu dans l'archive doit aussi s'appeler `OI-TP1-<Prenom>.<Nom>-<Version>`;

   (a) Pour les prochains TP, vous pourrez simplement utiliser `make dist` pour créer cette archive `.tar.gz.`

CHAPITRE 4

Mercredi 01/10/2003 : TP2, java et outils de programmation

À rendre pour le 06/10/2003.

Récupérez l'archive suivante, et décompressez-la avec les commandes `gunzip` et `tar`:`http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/OI-TP2-Daniel.Tounissoux-Nicolas.`
Entrez dans le répertoire créé, et essayez les différentes commandes `make`, `make demo`, `make check`, `make doc`. Le sujet du TP est dans le fichier `sujet.pdf`.

Correction :

`http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/OI-TP2-Daniel.Tounissoux-Nicolas.Thie`

Références :
- Cours Java de Daniel Tounissoux `http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/Cours-Ja`
- Site de SUN sur Java, avec des tutoriaux, ... `http://java.sun.com/`
- Documentation java, en local `file:/usr/java/docs/api/index.html`
- Documentation de la bibliothèque java, en local `file:/usr/java/docs/api/index.html`

CHAPITRE 5

Mardi 07/10/2003 : TP3, java et objets

À rendre pour le 14/10/2003.

http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/OI-TP3-Daniel.Tounissoux-Nicolas.Thie

Ce TP classe s'appuie sur les utilitaires standard de java, et en particulier java.util.ArrayList. Voir la documentation de référence. Voilà un micro résumé des commandes java qui vous seront utiles :

```
import java.util.ArrayList;
l=new ArrayList() // crée un nouveau tableau
l.add(1); // rajoute un élément à la fin
l.size(); // la taille du tableau
l.get(0); // le premier élément du tableau
l.set(0,new Integer(3)); // rentre la va-
leur 3 dans le premier élément du tableau
i=new Integer(3);
j=new Integer(4);
i.compareTo(j); // renvoie un nombre négatif
j.compareTo(i); // renvoie un nombre positif
i.compareTo(i); // renvoie 0
```

CHAPITRE 6

Semaine du 28 octobre au 1 novembre : TP4 : tutoriel

Dans la vraie vie, vous aurez régulièrement besoin d'apprendre par vous-même de nouveaux langages de programmation, de nouveaux logiciels. L'objectif de ce TP est de vous entraîner à cela. Vous allez choisir un langage de programmation que vous ne connaissez pas (`perl`, par exemple), chercher un tutoriel sur le web, et le suivre. Pour le 6 novembre, vous rendrez comme d'habitude une archive .tar.gz contenant les programmes que vous aurez écrit.

Pour la semaine prochaine, répondez aux questions marquées d'une étoile (*) dans le questionnaire.

CHAPITRE 7

Projet : Implantation d'un joueur automatique de Hex

Première version à rendre pour le mercredi 26 novembre. Version définitive pour le 10 décembre.

L'objectif de ce projet est d'implanter des joueurs automatiques pour le jeu de Hex.

On demande au minimum d'implanter les joueurs suivants :
- Méthode gloutone avec plusieurs fonctions d'évaluations (distance ou autre) ;
- Méthode min/max (recherche exhaustive) pour trouver une stratégie gagnante ;
- Méthode alpha/beta pour trouver un coup raisonnable

Pour aller plus loin :
- Implantation d'autres fonctions d'évaluation pour alpha/beta, en particulier par ■ mesure de résistance ■.
- Implantation d'autres méthodes.

Le squelette du projet, contenant entre autres une interface graphique, est fourni :
`http://lapcs.univ-lyon1.fr/~nthiery/DESS/Notes/Exemples/OI-Projet-Nicolas.Thiery-1.0.tar.gz`

CHAPITRE 8

Questionnaire

Première version à rendre pour le mercredi 26 novembre, version définitive pour le 10 décembre.

La suite de ce document est composé de questions, groupées par thèmes. Cette liste est susceptible d'évoluer. Beaucoup vous paraîtront au premier abord difficile, voire absconses. Ne vous fiez pas à votre première réaction ; en vous documentant, et avec un petit coup de main de temps en temps, vous devriez être capable d'y apporter une réponse *personnelle*. J'insiste sur *personnelle*. Selon vos goûts, vos sensibilités, certaines des questions peuvent avoir des réponses différentes, l'important étant que vos réponses soient *étayées*. Typiquement, chaque réponse devrait prendre quelques lignes et comporter un exemple simple et une ou plusieurs références.

J'encourage très fortement le travail en grand groupe, afin de partager le travail de recherche, de vous entraider, et de débattre les questions. Mais une fois ce travail de fond en commun effectué, l'objectif est que vous apportiez vous-même une réponse personnelle et détaillée à chacune des questions. Vous me rendrez donc un document pour au maximum deux personnes. Techniquement, ce document sera composé d'une archive `.tar.gz` contenant un (ou plusieurs) fichier(s) `HTML`. Ne lésinez pas sur les références et les exemples dans vos réponses : ce document est avant tout pour vous ; j'espère bien qu'il vous resservira !

Certains concepts ne sont pas évidents ! Si vous avez épuisé vos ressources sans trouver suffisamment d'éléments pour étayer votre propre réflexion, je me ferais un plaisir de vous guider. Alors, profitez de ma présence !

8.1. Langages de programmation

(1) Listez dix langages de programmations que vous jugez intéressants. Pour chacun d'entre eux, préciser de manière brève le domaine d'application, le type de langage, les caractéristiques principales, les points forts et faibles, et toute autre information que vous jugerez utile (bonnes références, tutoriels, mini-historique, ...).

(2) Sélectionnez cinq de ces langages, et faites un tableau récapitulatif indiquant leurs fonctionnalités. On pourra par exemple convenir d'un système de notation : 0 : fonctionalité absente, 1 : fonctionnalité possible mais au prix de contorsions, ... 5 : fonctionnalité parfaitement supportée. Pour chaque fonctionnalité non triviale, écrivez un petit programme qui l'illustre.

   (a) Interpréteur et/ou compilateur
   (b) Portabilité
   (c) Programmation impérative

(d) Fonctions, procédures et programmation fonctionnelle :
  (i) Récursivité
  (ii) Les fonctions sont-elles des objets de premier niveau ?
  (iii) Fonctions anonymes
  (iv) Fermetures, générateurs
  (v) Passage de paramètre par valeur et/ou par référence
  (vi) Variables locales à portée lexicale ou dynamique
  (vii) Macros, fonctions *inline*
  (viii) Optimisation partielle

(e) Typage, polymorphisme et programmation orientée objet :
  (i) Surcharge d'opérateurs, de fonctions
  (ii) Typage statique et/ou dynamique
  (iii) Programmation orientée objet
  (iv) Classes abstraites ; méthodes virtuelles
  (v) Classes paramétrées
  (vi) Création au vol de nouvelles classes
  (vii) Les classes sont-elles des objets de premier niveau ?
  (viii) Possibilité d'implanter de nouvelles structures de données aussi efficaces que celles fournies par le système

(f) Programmation logique
(g) Programmation littéraire
(h) Assertions, préconditions et postconditions
(i) Sémantique de copie et/ou de référence
(j) Glaneur de cellules (*garbage collector*)
(k) Librairie standard :
  (i) Structures de données classiques (listes chaînées, piles, files, vecteurs dynamiques, tables d'associations, ensembles).
  (ii) Calcul numérique
  (iii) Calcul symbolique
  (iv) Statistiques
  (v) Graphiques
  (vi) Expressions régulières
  (vii) Système et réseau

(l) Communauté :
  (i) Tutoriaux, sites d'introduction, listes de diffusions
  (ii) Sites pour échanger du code
  (iii) Licence d'utilisation

(3) Quelles sont les améliorations les plus importantes, de votre point de vue, dans Fortran 90 vis-à-vis des anciennes versions ?

8.2. Concepts et paradigmes de programmation

(1) Citez dix algorithmes classiques. Classez, selon vos goûts, les cinq plus importants.
(2) Que dit, en gros, la thèse de Curry-Howard ?
(3) De qui est la citation ■Beware of bugs in the above code ; I have only proved it correct, not tried it■ ?

8.3. Outils et méthodes de développement

(1) Quelles sont, selon Larry Wall, les 3 plus importantes qualités d'un programmeur ? Pourquoi ?
(2) Quels sont les avantages et inconvénients d'un environnement de programmation intégré ?
(3) Quels sont les avantages et inconvénients des composants visuels et non-visuels (cf. par exemple Borland Delphi, C++ builder, ou bien les javabeans). Pour certaines situations (ex. créer une unité qui effectue une régression multiple générique), est il plus intéressant ou plus restrictif de créer un composant visuel/non-visuel ?

8.4. écriture de documents

(1) Faut-il mettre les accents aux majuscules dans un texte tapé en français ? Où trouver ce genre d'information ?
(2) Pourquoi séparer la forme et le fond d'un document ?

8.5. Société

(1) Quel critères utiliser pour dire qu'un langage de programmation, ou un format de fichier (par ex. word, pdf, mp3) est un *standard* ? Est-ce que cela a une importance ?
(2) Quels sont les principes essentiels de la nétiquette ?
(3) Qui a cassé l'énigme *Enigma* ?
(4) Qu'est-ce qu'une FAQ, une RFC ?
(5) Quels sont les grands précurseurs de l'informatique avant 1940 ?
(6) Quel est le point commun entre Gödel, Escher et Bach ?

8.5.1. Droit du logiciel.

(1) Qu'est-ce qu'un logiciel libre ? (*free software*, *open source software)*
(2) Lorsque vous développez un logiciel, est-ce que le choix d'une licence est important ?
(3) Ai-je le droit de vendre un logiciel sous GPL ; peut-on utiliser des applications commerciales sous GNU/Linux ?
(4) Y-a-t'il une différence entre un programme du domaine public, et un programme sous licence libre ?
(5) Quel peut bien être l'intérêt pour une entreprise privée de développer un logiciel sous licence libre ?

(6) Pour ou contre les brevets sur les logiciels ?

(7) Pour ou contre l'usage de la cryptographie ?

8.5.2. Modèles de développement.

(1) Indiquez, parmi les logiciels suivants, ceux qui ont un modèle de développement de type ■*Cathédrale*■ ou de type ■*Bazar*■ :
   (a) Emacs
   (b) Linux
   (c) Word
   (d) Scilab

(2) L'homme-an est-il un mythe ?

(3) Que signifie ■Given enough eyeballs, every bug is shallow■ ?