Projets de fin d'année
Sujets et Consignes

Année 2014-2015

Table des matières

Chapitre 1

Consignes

1.1 Introduction

Ce document regroupe l'ensemble des informations nécessaires à la réalisation du projet de fin d'année. Vous y trouverez les directives générales, les sujets (section **??**) les dates de remises de documents ainsi qu'en annexe, un guide et des consignes pour la réalisation du projet.

Formation des groupes : la plupart des projets sont prévus pour être réalisés en trinôme. (Le nombre d'étudiants peut être quatre, mais dans ce cas, il faut que les fonctionnalités proposées le justifient). De manière générale, la présence d'une interface graphique est recommandée, mais pour les étudiants dispensés de stage (le module vaut alors 6 ECTS) une interface graphique est obligatoire. Pour les étudiants ayant suivis l'option Logiciel Libre, le projet est l'occasion cette année de leur donner une note de mise en pratique des logiciels libres valant pour l'option. Il est donc conseillé à ces étudiants de se réunir dans un même groupe. (Il leur sera demandé en particulier de choisir une licence, et de mettre en place un git ou un svn pour le projet).

1.2 Déroulement des projets

Vous devez normalement choisir un sujet parmi ceux présentés ici. Mais si vous souhaitez faire une autre proposition, envoyez rapidement une page de description au format PDF à Nicolas Rolin et Catherine Recanati qui valideront ou non votre proposition.

Vous devez envoyer à Nicolas Rolin (`nicolas.rolin@lipn.univ-paris13.fr`) et Catherine Recanati (`catherine.recanati@lipn.univ-paris13.fr`) trois sujets intéressant votre groupe dans l'ordre de vos préférences, ainsi que la liste des membres de votre groupe et leurs adresses électroniques, et ceci **avant le 11 mars 2015, 17h**. Le sujet qui vous sera ensuite attribué vous sera indiqué par courrier électronique, avant le 22 mars 2015. (NB : Si vous ne fournissez pas une liste de trois sujets, il est possible que vous vous voyiez attribuer un tout autre sujet).

Le langage utilisé sera a priori le langage C ou Java. Des librairies graphiques (Gtk par exemple) sont autorisées pour l'interface graphique. En cas de doute sur le (ou les) langage(s) de dveloppement utilisé(s), envoyez un mail à vos deux encadrants.

Le **13 avril 2015 à 0h au plus tard**, vous enverrez un document de 15 pages (maximum) **au format PDF uniquement** (tout document dans un autre format sera ignoré). Ce premier document suivra les indications du Guide de ralisation des projets (annexe A) et inclura :

— une présentation générale et une analyse approfondie de votre projet (cf. **??**)
— une liste détaillée des fonctionnalités de votre logiciel (cf. **??**)
— l'architecture logicielle (les diagrammes de classes éventuels si vous programmez dans un langage objet) et les interactions entre modules et librairies en vue de l'implémentation sous forme d'un diagramme de dépendances entre les modules (les modules étant représentés par des boîtes et des flèches matérialisant les dépendances entre les boîtes). (cf. **??**)

Notez bien qu'aucune programmation n'est demandée à ce stade (il ne s'agit que de spécifications) **et qu'il est très important que vous respectiez les délais.**

Vous soignerez l'analyse du sujet et la présentation du document. Un guide de réalisation des projets et des indications sur les documents à rendre vous sont fournis en annexet **??** et **??**. Soignez en particulier la première page, et faites une table des matières paginée. La note finale tiendra compte de la qualité de ce premier rapport et des corrections qui lui seront apportées ultérieurement.

Le 10 mai 2015 au plus tard, vous aurez des retours sur votre document par courrier électronique. Les groupes en difficulté seront convoqués pour une séance de discussion informelle autour du rapport dans la semaine du 11 mai. Vous serez prévenu de ce rendez-vous par courrier électronique.

La soutenance finale du projet est prévue la **semaine du 1er juin** (vous serez informé par affichage et par mail). **La semaine qui précéde (semaine du 25 mai)**, vous devrez envoyer par mail à Nicolas Rolin et Catherine Recanati :

— un rapport d'au plus 25 pages (format PDF uniquement) décrivant l'ensemble de votre projet (vous pourrez partir du rapport d'analyse du projet envoyé initialement)
— un manuel utilisateur (comportant aussi un manuel d'installation) au format PDF également
— une archive `ZIP` ou tar.gz contenant le code commenté de vos programmes dans leur état actuel (tout autre format sera refusé - excepté une archive jar pour le code java)

NB : Le code pourra être modifié jusqu'au jour de la présentation et de la démonstration mais à vos risques et périls (conservez une version qui fonctionne). La note finale du projet tiendra compte du rendu de ce rapport (qualité des documents, qualité des commentaires du code, précision du manuel utilisateur et du manuel d'installation).

Vous effectuerez pour la soutenance elle-même une présentation collective (basée sur un fichier PowerPoint, OpenOffice ou pdf) et vous ferez une démonstration de votre logiciel en salle machine.

Le fichier de la présentation (format .pdf, .ppt ou .pptx uniquement) sera envoyé **deux jours avant la soutenance orale** pour que nous puissions l'installer par avance sur un même ordinateur. Par ailleurs, vous devez préparer soigneusement le scénario de votre démonstration et répéter dans l'environnement prévu pour cette démo. Même si votre logiciel ne fonctionne pas entièrement, vous pouvez montrer le fonctionnement correct de certaines fonctions ou modules dans une démonstration. La note finale tiendra compte de la qualité de la présentation, de celle de la démonstration, ainsi que du rendu logiciel.

Aide technique : Vous pouvez naturellement envoyer des messages électroniques pour avoir des éclaircissements sur votre sujet ou une aide sur un problème technique (ne restez pas trois semaines sur un bug sans réagir ! Mais ne venez pas non plus pour un probléme de compilation dû à une simple erreur de syntaxe). Pensez à d'abord envoyer un mail pour fixer un rendez-vous, et à amener votre code sous format électronique (de préférence sur clé USB, sinon disponible sur le réseau).

Chapitre 2

Liste des projets

2.1 Gestion de réferences bibliographiques en bibtex et HTML

Le projet consiste à écrire un logiciel permettant de gérer des références bibliographiques contenues dans un fichier au format bibtex. Le logiciel offrira les fonctionnalités suivantes :

- chargement d'un fichier bibtex
- ajout de références
- modification de références
- suppression de références.
- sauvegarde des modifications
- Extraction/recherche des références à partir de noms d'auteurs, de mots du titre, de noms de références
- génération d'un fichier HTML valide permettant d'afficher la liste des références dans le format standard :
  author, title, publisher, series, booktitle, volume(number), pages, adress, year, note, annote.
- interrogation à distance, grâce à une architecture client/serveur. (Obligatoire si plus de deux étudiants).

Une base de données de références bibliographiques contient des références qui peuvent référencer des livres, des articles de revues ou des articles de conférences. A chacun de ces types de référence correspondent plusieurs champs dans le format bibtex. Ces différents champs sont donnés dans les exemples ci-dessous :

- Exemple de référence au format bibtex, pour un livre :

```
@Book{Mallatbook,
  author =   {S. Mallat},
  title =   {A Wavelet Tour of Signal Processing},
  publisher =   {Academic Press},
  year =   {1998},
  OPTeditor =   {},
  OPTvolume =   {},
  OPTnumber =   {},
  OPTseries =   {},
  address =   {Boston},
  OPTedition =   {},
  OPTnote =   {},
  OPTannote =   {}
}
```

— Exemple de référence au format bibtex, pour un article de revue :

```
@Article{Temlyakov,
  author =   {V. Temliakov},
  title =   {Nonlinear methods of approximation},
  journal =   {FOCM},
  year =   {2008},
  OPTkey =   {},
  volume =   {3},
  number =   {1},
  pages =   {33-107},
  month =   {Feb.},
  OPTnote =   {},
  OPTannote =   {}
}
```

— Exemple de référence au format bibtex, pour un article de conférence :

```
@InProceedings{pati93OMP,
  author =   {Y. Pati and R. Rezaiifar and P. Krishnaprasad},
  title =   {Orthogonal Matching Pursuit: Recursive Function
                Approximation with Applications to Wavelet Decomposition},
  booktitle = {27 th Annual Asilomar Conference on Signals, Systems,
    and Computers},
  pages =   {40-44},
  year =   {93},
  OPTeditor =   {},
  volume =   {1},
  OPTnumber =   {},
  OPTseries =   {},
  OPTaddress =   {},
  publisher = {IEEE},
  OPTnote =   {},
  OPTannote =   {}
}
```

Dans les références ci-dessus, les champs commençant par OPT, sont optionnels et ne sont pas renseignés. Les références portent ici les noms "Mallatbook", "Temlyakov" et "pati93OMP".

NB :

— Vous pourrez utiliser les outils Flex et Bison pour analyser un fichier Bibtex.

— Un fichier chargé pourra contenir des références incomplètes

2.2 Gestion de cave à vin

Il s'agit d'écrire un logiciel permettant de gérer le stock d'une cave à vin. On peut imaginer que cette cave appartienne à un particulier ou à un marchand de vin. On caractérisera un vin par sa région, son domaine, son château, sa couleur, son année, son cépage. Une bouteille de vin sera caractérisée par le vin qu'elle contient et la taille de la bouteille. La cave contient des vins. On voudra donner pour un vin, le nombre de bouteilles, une année de maturité, une localisation (dans la cave), des commentaires (d'au plus 1000 caractères).

Le logiciel offrira les fonctionnalités suivantes :

— Chargement d'un fichier décrivant une cave

— modification des vins

— ajout de vins

— suppression de vins

— sauvegarde d'une cave
— Recherche et affichage des vins par nom, par année de maturité, par couleur, par cépage, région, etc. Plusieurs critères pourront être combinés.
— interrogation à distance, grâce à une architecture client/serveur si le cas envisagé est celui d'un magasin

NB :

— Le nombre de vins ne sera pas limité (vous utiliserez une structure de données dont la taille n'est pas fixée à l'avance).
— Vous pourrez utiliser les outils Flex et Bison pour analyser les fichiers stockant les informations relatives à/aux cave(s) à vins.

2.3 Jeu d'échecs en réseau

Ce projet devra permettre à un joueur de jouer une ou plusieurs parties à travers le réseau avec d'autres partenaires. Un serveur gérera les plateaux correspondant aux différentes parties. Les clients se connecteront au serveur soit pour jouer, soit pour assister, comme spectateur, au déroulement d'une ou plusieurs parties.

2.4 Allocation contiguë en mémoire centrale

Il s'agit de simuler l'arrivée de processus ("threads") venant demander un espace mémoire au système pour s'exécuter, et d' implanter des algorithmes classiques d'allocation mémoire à savoir "best fit", "first fit", et "worst fit ", en vue de permettre une comparaison de leurs performances. Le logiciel permettra de paramétrer la simulation, de la lancer, et de comparer les résultats obtenus par les différents algorithmes.

2.5 Ordonnancement de processus

On suppose que des processus (sous forme de "threads ") arrivent dans le système au hasard. On fera pour cela une simulation temporelle d'arrivée de processus demandant un temps de calcul au processeur. Implanter les différents algorithmes d'ordonnancement de ces processus vus en cours. Le logiciel permettra de lancer ces différents algorithmes et de comparer leur efficacité.

2.6 SGBD réparti

Dans ce projet, on souhaite gérer un SGBD réparti simplifié. Dans un SGBD réparti, les données ne sont pas stockées sur un seul serveur mais sur un ensemble de serveurs (on suppose qu'un enregistrement est stocké complètement sur un serveur). Lorsqu'un client fait une requête sur ce SGBD, on doit interroger l'ensemble des serveurs constituant le SGBD pour obtenir le résultat (on suppose des requêtes simples : SELECT FROM WHERE sans jointure). De même, lorqu'il fait une insertion ou une mise à jour, il faut déterminer sur quel serveur doit se faire l'opération. Pour résoudre ce problème, deux solutions architecturales sont envisageables :

1. un serveur central qui a la connaissance de tous les serveurs constituant la base
2. une architecture de type P2P.

Vous mettrez en place un mécanisme pour le stockage des enregistements sur les serveurs, et un mécanisme de cache en mémoire pour les enregistements les plus souvent accédés. Vous définirez un mini langage de requêtes pour ce SGBD.

2.7 Gestion d'arbres généalogiques

L'arbre généalogique d'une personne contient tous ses ascendants (parents, grands-parents, arrière-grands-parents, etc.) connus.

Le programme doit permettre à l'utilisateur d'entrer une personne et son arbre généalogique. Pour chaque personne, on peut donner son nom, son prénom, sa date et son lieu de naissance, et éventuellement sa date et son lieu de décès. Le programme doit permettre de saisir l'arbre généalogique d'une personne, puis de l'afficher et de le modifier. L'utilisateur devra aussi avoir la possibilité de sauvegarder un arbre généalogique dans un fichier et, bien sûr, de pouvoir afficher, modifier, ou compléter un arbre ainsi sauvegardé.

Le couple des parents d'une personne peut avoir d'autres enfants. On souhaite également pouvoir stocker les autres enfants d'un couple, ce qui amène les notions de frère et sœur.

Le logiciel pourra avoir la notion de frère et sœur, ainsi que d'autres notions de parenté et permettre de rechercher dans la base un frère, une sœur, un cousin, ou tout autre relation de parenté comprise par le logiciel.

Vous devrez également permettre d'effectuer des recherches par nom, prénom, date ou lieu de naissance, date ou lieu de décès, etc. sur des personnes connues de la base.

Vous pourrez utiliser les outils Flex et Bison pour analyser les fichiers stockant les informations relatives à un arbre généalogique.

2.8 Gestion d'une bibliothèque

Dans ce projet, vous développerez un logiciel de gestion de bibliothèque. Il devra offrir les fonctionnalités suivantes :

— stocker des références de livres (auteur, titre, ...)
— ajout, suppression et modification de livres
— Recherche d'un ouvrage selon un ou plusieurs critères
— Gestion des prêts

Deux interfaces de visualisation et de manipulation sont demandées : d'une part, une interface en ligne de commande, d'autre part une interface graphique.

Vous pourrez utiliser les outils Flex et Bison pour analyser les fichiers stockant les informations relatives à une bibliothèque.

Des informations complémentaires détaillant le sujet sont fournies à la section ??.

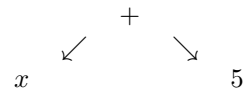
2.9 Matrices creuses

Nombre d'étudiants maximum : 2. Dans beaucoup de domaines on utilise des matrices de grande taille mais dont beaucoup de coefficients sont nuls. Ces matrices sont appelées *creuses*. On peut représenter ces matrices en ne stockant que la liste de leurs coefficients non nuls. Écrire un programme manipulant des matrices creuses. Ce logiciel doit permettre de saisir, de sauvegarder, de charger des matrices et d'effectuer des opérations courantes sur les matrices.

Des informations complémentaires détaillant le sujet sont fournies à la section ??.

2.10 Calcul formel

Une expression symbolique du type $x + 5$ peut être représentée par un arbre



Le logiciel permettra de saisir une expression (par exemple $x^2 + 3x + 8$), de l'évaluer pour une valeur de x et de la dériver. On désire aussi sauvegarder et récupérer des expressions. Les expressions seront représentées dans le programme sous forme d'arbres.

Des informations complémentaires détaillant le sujet sont fournies à la section **??**.

2.11 Polynômes

Un polynôme à coefficients réels est représenté par la liste de ses monômes (sans limitation de degré). Le logiciel devra permettre de faire toutes les opérations usuelles sur les polynômes (addition, soustraction, multiplication, division euclidienne, PGCD, PPCM, dérivation) et aussi permettre de sauvegarder et restaurer des polynômes précédemment définis.

Vous étudierez une variante où les coefficients sont dans $\mathbb{Z}/p\mathbb{Z}$ avec p premier. Vous pourrez également étendre les calculs aux polynômes de 2 variables quand cela a un sens.

Le logiciel devrait permettre d'entrer un polynôme de différentes manières (en entrant par exemple les coefficients de chaque monôme à l'aide d'une interface graphique), mais on veillera à avoir aussi une procédure d'entrée qui consiste à lire une chaîne de caractères représentant le polynôme. Pour cela on définira la syntaxe d'écriture d'un polynôme, et si la syntaxe de la chaîne entrée est incorrecte, on affichera un message d'erreur. Une possibilité de syntaxe est de représenter les exposants avec le caractère circonflexe. Ainsi le polynôme $3x^5 + 4x^2 + 7x - 1$ pourrait être entré à l'aide de la chaîne de caractères 3xˆ5 + 4xˆ2 + 7x -1. Avec ce type de syntaxe, l'ordre d'entrée des monômes serait quelconque.

On devra aussi pouvoir donner un nom à un polynôme et le sauvegarder avec son nom dans un fichier. On pourra aussi charger un fichier contenant un polynôme. Il pourrait aussi être intéressant de pouvoir charger ou sauvegarder des fichiers contenant plusieurs polynômes.

2.12 Simulation de circuits électroniques

Les circuits électroniques sont constitués de composants reliés entre eux par des entrées et des sorties. La simulation consiste à envoyer un signal à l'entrée du circuit (des 0 ou des 1) et à observer ce qui se passe à la sortie (encore des 0 ou des 1). Le logiciel devra permettre de créer, de sauvegarder et de charger des circuits électroniques.

Des informations complémentaires détaillant le sujet sont fournies à la section **??**.

2.13 Assembleur MAMIAS

MAMIAS est un langage assembleur élémentaire (pour une machine virtuelle) utilisé pour l'enseignement en premier cycle. Le logiciel lira un fichier source contenant un programme écrit en langage MAMIAS et générera un fichier binaire contenant le code machine de ce programme. On pourra alors exécuter le programme pas à pas.

Les informations détaillant le sujet sont fournies dans la section **??**.

2.14 Le jeu de la vie

Le jeu de la vie (créé par le mathématicen John Horton Conway) est basé sur l'évolution de populations cellulaires. La naissance et la mort d'une cellule y sont conditionnées par la présence suffisante ou excessive de cellules vivantes dans le voisinage de cette cellule. On considére généralement un espace plat, constitué d'une grille, dans laquelle chaque case a 8 voisins. Une case ne peut contenir qu'une seule cellule et les régles sont les suivantes : i) si le voisinage d'une case vide contient trois cellules, alors une cellule nait ; ii) si une cellule est entourée de plus de 4 cellules, elle meurt étouffée ; iii) si une cellule est entourée d'une cellule ou moins, elle meurt d'isolement.

On voudrait pouvoir simuler l'évolution d'une population de cellules placées sur une grille, avec des facilités de chargement et sauvegarde d'états. Ainsi, on devra pouvoir entrer une grille facilement pour tester le logiciel, l'idéal étant de pouvoir le faire interativement sur une grille avec la souris. (Mais on pourra aussi choisir de le faire par l'intermédiaire de fichiers textes éditables si l'on ne fournit pas d'interface graphique).

Le logiciel doit permettre de simuler l'évolution d'une population de cellules initialement placées sur une grille. Cette simulation devrait pouvoir être réalisée en différent mode : un mode pas à pas enchaînant les évolutions successives, ou un mode rapide. On doit également pouvoir à tout moment sauvegarder une grille ou en charger une nouvelle. On préparera pour la démonstration des exemples intéressants, préalablement sauvegardés dans des fichiers.

Des informations complémentaires sur le sujet sont fournies dans la section **??**.

2.15 Moteur d'indexation

Un moteur de recherche est composé de trois grandes parties : l'exploration (réalisée par le moissonneur ou le *crawler*), l'indexation, et la recherche.

On s'intéressera ici à la partie indexation. L'objectif est de construire un index d'une collection de documents. L'index construit est similaire à l'index d'un livre. Il contient notamment les termes (mots et groupes de mots) présents dans les documents et en indique la liste. Il doit être organisé de manière à faciliter la recherche, c'est-à-dire la récupération des documents pertinents pour un terme donné.

Le projet consiste à développer un moteur d'indexation suffisamment générique pour indexer des documents en fonction de termes, mais aussi en fonction de différents types de méta-données qui pourraient être associées aux documents. Il pourra gérer ainsi plusieurs index. On fournira également une interface minimale offrant la possibilité de soumettre des requêtes au format CQL (Common Query Language, protocole Z3940).

Documentation complémentaire :

— `http://fr.wikipedia.org/wiki/Moteur_de_recherche`
— `http://fr.wikipedia.org/wiki/Optimisation_pour_les_moteurs_de_recherche`
— `http://fr.wikipedia.org/wiki/Indexation`
— `http://zing.z3950.org/cql/`
— `http://www.ifla.org/IV/ifla72/papers/102-McCallum_trans-fr.pdf`

2.16 Master Mind

Le but du jeu est de découvrir une combinaison de pions de couleurs différentes (le nombre de pions et de couleurs peut varier en fonction du niveau). Le joueur qui cherche propose une combinaison de pions de couleur. Le programme lui indique alors combien de pions sont bien placés (bonne couleur à la bonne place) et combien

sont mal placés (bonne couleur, mais pas à la bonne place). Pour un exemple de jeu, on pourra consulter le site `http://www.litterales.com/jeux`

Ce jeu de Master Mind pourra se jouer à un ou deux joueurs. Dans la version mono-joueur, le choix de la combinaison à découvrir sera fait par le programme. Le but est alors de trouver 10 combinaisons en un minimum de propositions. Dans la version à deux joueurs, c'est l'adversaire qui propose une combinaison. A chaque tour, on échange les rôles et on additionne le nombre de coups qu'a utilisés le joueur pour trouver la bonne combinaison. Le perdant est le premier qui atteint 100. Une partie pourra être sauvegardée en cours de route et rechargée pour être poursuivie plus tard.

2.17 Thématiseur Général

Un thématiseur général est un logiciel prenant en entrée un document et retournant une liste de mots-clés (les thèmes) correspondant au contenu du document (par exemple : science, tourisme, humour, horoscope, etc.). Un document est un fichier texte, ici sans ponctuation : on considèrera qu'il s'agit d'une suite de mots (chade caractères sans espace) séparés par des espaces. Un thématiseur général fait appel à des thématiseurs spécialisés (un par thème). Un thématiseur spécialisé est initialisé pour un thème (ou mot-clé) particulier. Il a comme ressource une liste de mots correspondant au thème (par exemple pour le thème 'science' les mots 'informatique', 'physique', 'chimie', etc.). Sa fonction principale, étant donné un document, est de retourner le pourcentage de mots du document faisant partie de sa liste de mots thématique.

Des informations complémentaires sont fournies à la section **??**.

2.18 Serveur de petites annonces

Le projet consiste à écrire un gestionnaire de petites annonces sous la forme d'une architecture client/serveur. Le programme serveur stockera un ensemble d'annonces et permettra aux clients de consulter les annonces, d'en créer de nouvelles, ou d'en supprimer. Une annonce sera constituée d'un titre et d'un texte (les deux sous forme de chaîne de caractères) et d'un auteur (l'utilisateur ayant créé l'annonce). Le serveur permettra la connexion de programmes clients et ceux-ci pourront consulter la liste des titres des annonces, éditer le texte d'une annonce particulière, créer une nouvelle annonce, ou encore supprimer une annonce (mais on ne pourra supprimer une annonce que si on en est l'auteur). Le programme client aura une interface graphique agréable affichant en permanence la liste des titres des annonces et leur nombre. Des boutons permettront l'ajout, la suppression et l'édition d'une annonce sélectionnée par son titre. Un bouton de mise-à-jour permettra de recharger la liste des annonces disponibles au niveau du serveur.

Chapitre 3

Informations complémentaires

3.1 Gestion d'une bibliothèque

Le logiciel est destiné à la bibliothèque d'un petit collège de 12 classes contenant environ un millier de livres.

Il faudra gérer trois fichiers :
- le fichier des emprunteurs :
  - référence (né d'identifiant unique)
  - nom
  - prénom
  - classe
- le fichier des ouvrages
  - référence (né d'identifiant unique)
  - nom auteur
  - prénom auteur
  - titre
  - état : emprunté, rendu, en rayon, en réparation
  - référence d'emprunt
- le fichier des emprunts
  - référence (né d'identifiant unique)
  - référence livre
  - référence d'emprunteur
  - date d'emprunt
  - date limite d'emprunt

Il faut que l'on puisse emprunter, rendre, ajouter, retirer, modifier l'état d'un livre, et inscrire, désinscrire les emprunteurs.

Vous permettrer également :
- de gérer les retards de rendu d'ouvrages
- de rechercher un ouvrage

3.2 Matrices creuses

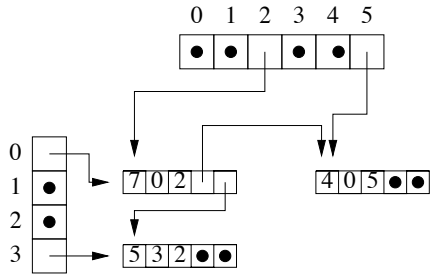
Les matrices creuses sont des matrices ayant beaucoup d'éléments nuls. Soit A une matrice creuse. Pour chaque élément non nul $a[i, j]$ on crée un enregistrement de cinq champs contenant

— la valeur,
— les indices i,j,
— deux pointeurs vers l'élément non nul suivant de de sa ligne et de sa colonne respectivement.

La matrice sera représentée par un tableau ligne et un tableau colonne de pointeurs téte de liste pour les lignes et les colonnes respectivement. Par exemple la matrice :

$$\begin{pmatrix} 0 & 0 & 7 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix}$$

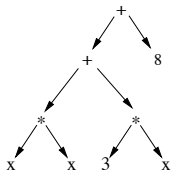
donnera la représentation :



Le logiciel permettra l'addition, la soustraction et la multiplication de matrices creuses. On pourra sauvegarder et restaurer des matrices creuses. Vous pourrez également remplacer les tableaux qui représentent la matrice par des listes.

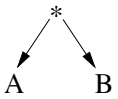
3.3 Calcul formel

Une expression symbolique du type $x^2 + 3x + 8$ peut être représentée par un arbre

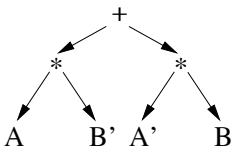


Dans un premier temps on se limitera aux 4 opérations de base. Le logiciel permettra de saisir une expression symbolique, de l'évaluer pour un valeur de x, et de la dériver. La notation la plus facile à lire pour un programme est la notation polonaise inversée (en vigueur sur certaines calculatrices comme les HP48). Elle consiste simplement à donner les arguments de la fonction avant la fonction elle même. Ainsi, $x^2 + 3x + 8$ est-il noté $x\ x\ \times\ 3\ x\ \times\ +\ 8\ +$ en notation polonaise inversée.

Pour dériver on utilise les formules de dérivation d'une somme, d'une différence, d'un produit et d'un quotient. Ainsi la formule $(ab)' = ab' + a'b$ se traduira par le fait que la dérivée de l'arbre



est l'arbre



Mais vous devrez aussi faire des simplifications élémentaires :

$$(3x + 1)' = 3 \times 1 + 0 \times x + 0 = 3 + 0 + 0 = 3$$

Vous inclurez les fonctionnalités suivantes :
— ajouter des constantes : π , e ...
— ajouter d'autres fonctions : sinus, log ...
— améliorer les simplifications.

3.4 Simulation de circuits électroniques

On voudrait simuler le fonctionnement d'un circuit électronique logique, c'est-à-dire donner les valeurs de sortie (0 ou 1) en fonction des valeurs données en entrée (0 ou 1).

Un circuit électronique est un ensemble de composants connectés entre eux. Ces composants peuvent être des portes AND (2 entrées, 1 sortie), des portes OR (2 entrées, 1 sortie), des portes XOR (2 entrées, 1 sortie), des inverseurs NOT (1 entrée, 1 sortie), des entrées IN (0 entrée, 1 sortie) ou des sorties OUT (1 entrée, 0 sorties). Les tables de vérités de ces composants sont les suivantes :

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

NOT	0	1
	1	0

Les circuits ne seront pas édités dans le logiciel, mais chargés à partir d'un fichier. Une ligne dans un fichier décrit soit un composant (`composant NomComposant TypeComposant`), soit une liaison entre deux composants (`liaison NomComposant NéSortie NomComposant NéEntrée`), avec `TypeComposant`$\in$ {`AND,OR,NOT,IN,OUT`} et `NomComposant` un mot (suite de caractères sans espace).

Exemple de fichier de circuit :

```
composant   in1   IN
composant   in2   IN
composant   in3   IN
composant   out1  OUT
composant   and1  AND
composant   or1   OR
liaison     in1   1    and1  1
liaison     in2   1    and1  2
liaison     and1  1    or1   1
liaison     in3   1    or1   2
liaison     or1   1    out1
```

Vous inclurez les fonctionnalités suivantes :

— ajouter d'autres composants : noeud1-2 (noeud qui envoie sur ses deux sorties la même chose que ce qu'il a en entrée), multiplexeurs, ...
— produire des tables de vérité (tableaux dont les colonnes contiennent les valeurs en entrée et les valeurs en sortie)
— éditer un circuit avec le logiciel
— (et animer) les circuits (difficile)

3.5 Assembleur MAMIAS

MAMIAS est un langage assembleur pour machine virtuelle élémentaire utilisé pour l'enseignement en premier cycle. Le logiciel lira un fichier source contenant un texte de programme du type

```
def     x       31
def     y       30

        INIT    0
        CHARGE  x
        SAUTE   suite
        RANGE   y
suite : ADD     x
        ....
```

où les deux premières lignes définissent des 'variables', et dont les suivantes sont les instructions du programme dans une notation symbolique. Le logiciel générera un fichier binaire contenant le code de ce programme.

Vous écrirez aussi un outil de visualisation de la mémoire centrale permettant d'exécuter un programme MAMIAS pas à pas.

Description de MAMIAS.

Les instructions et les données sont codées dans des mots mémoire (**huit positions binaires**). Chaque mot mémoire est repéré par un entier $n \geq 0$ appelé son **adresse**, le mot mémoire d'adresse n est désigné par `(n)`. On dispose en plus d'un mot mémoire particulier appelé **accumulateur** et désigné par *Acc*.

Une instruction est un mot mémoire décomposable en $CodeArgument$
- *Code* : est le numéro de code de l'opération (les 3 positions binaires de gauche),
- *Argument* : est
  - ou bien une adresse `n`,
  - ou bien un entier[5] `x` (les 5 positions binaires de droite).

On peut donc coder 8 opérations (puisque Code est sur 3 bits), adresser 32 mots mémoire (de 0 à 31, puisque sur 5 bits) et utiliser des entiers signés compris entre -16 et 15.

Une donnée est un entier[8] codé dans un mot mémoire adressé, ou figurant dans l'accumulateur.

Pour faciliter la lecture, chaque code opération sera désignée par une "expression mnémonique" (que MAMIAS lui-méme n'est pas censé comprendre : il ne comprend que le code des 3 bits).

Les 8 opérations et l'effet des instructions correspondantes sont les suivants : ($\leftarrow$ est l'affectation)
- `INIT` (Code = 000) : `INIT` est `Acc` $\leftarrow$ `x`
  Note : `x` est un entier[5] qui se trouve traduit en entier[8] dans l'accumulateur, plus précisément :

  - l'instruction 0000*abcd* (`INIT` 0*abcd*) donne la valeur 0000*abcd* à l'accumulateur,
  - l'instruction 0001*abcd* (`INIT` 1*abcd*) donne la valeur 1111*abcd* à l'accumulateur.
- `CHARGE` (Code = 001) : `CHARGE n` est `Acc` ← `(n)`
- `RANGE` (Code = 010) : `RANGE n` est `(n)`← `Acc`
- `ET` (Code = 011) : `ET n` est `Acc`← `Acc` ∧ `(n)`
  où ∧ est l'opération de conjonction bit à bit (ou masquage) définie par $1 \wedge 1 = 1$, $1 \wedge 0 = 0 \wedge 1 = 0 \wedge 0 = 0$. Par exemple $10110001 \wedge 11010000 = 10010000$.
- `SAUTE` (Code = 100) : `SAUTE n` est "si `Acc` = 0 alors aller à l'adresse `(n)`"
- `ADD` (Code = 101) : `ADD n` est `(n)`← `(n) + Acc`
- `DEC` (Code = 110) : `DEC x` décale de `x` positions le contenu de `Acc`
  Ce décalage se produit vers la gauche si `x` est positif, et vers la droite s'il est négatif : les digits qui "sortent" du mot sont perdus !
  Par exemple, si `Acc` = 01110011 :
  - l'instruction 11000010 (`DEC2`) donne la valeur 11001100 à l'accumulateur,
  - l'instruction 11011101 (`DEC` −3) donne la valeur 00001110 à l'accumulateur.
- `STOP` (Code = 111) : `STOP` arréte l'exécution.
  Les cinq positions binaires de droite n'ont aucune signification.

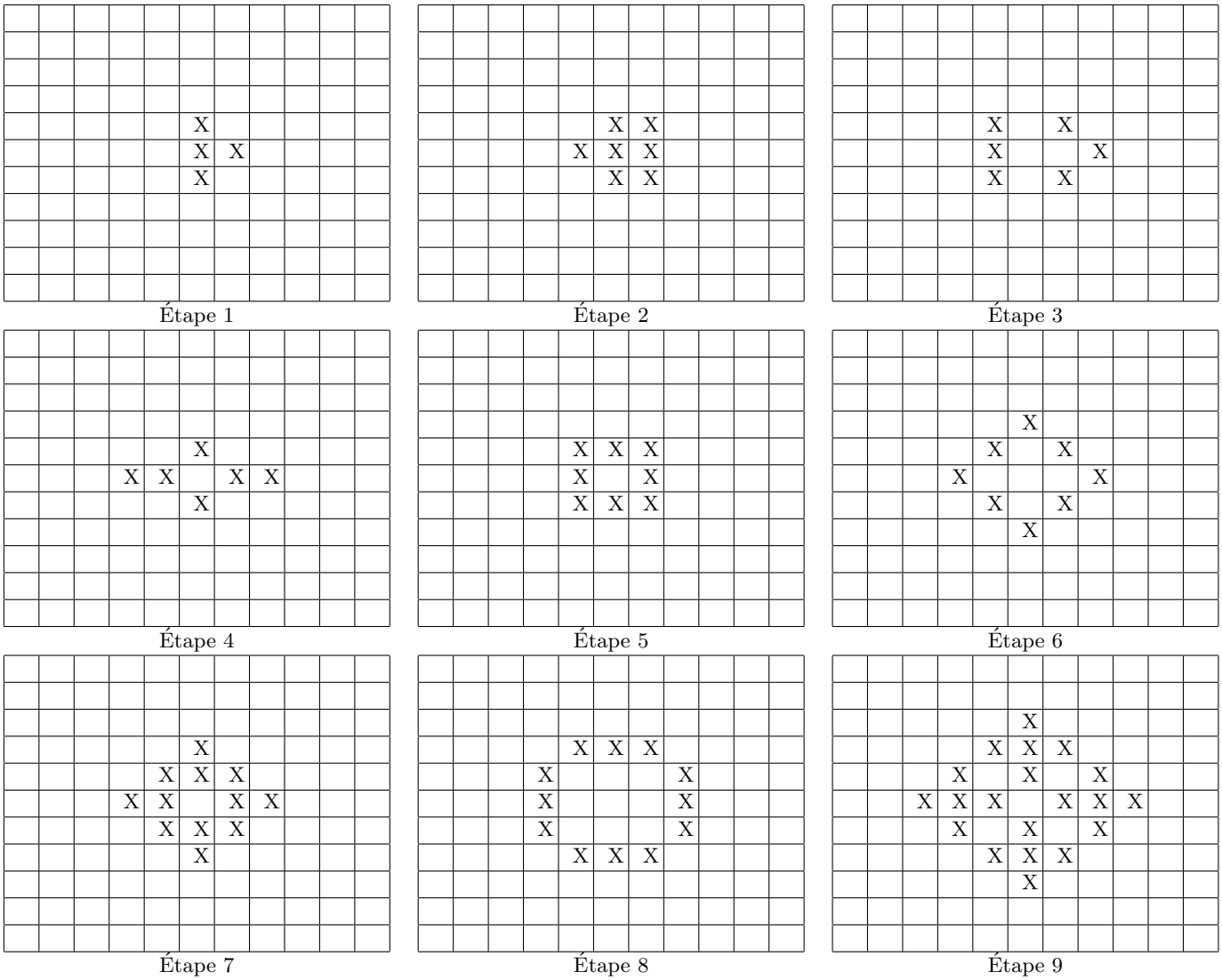
Sauf avis contraire (donné par l'opération `SAUTE`), les instructions s'exécutent dans l'ordre des adresses croissantes, à partir de 0.

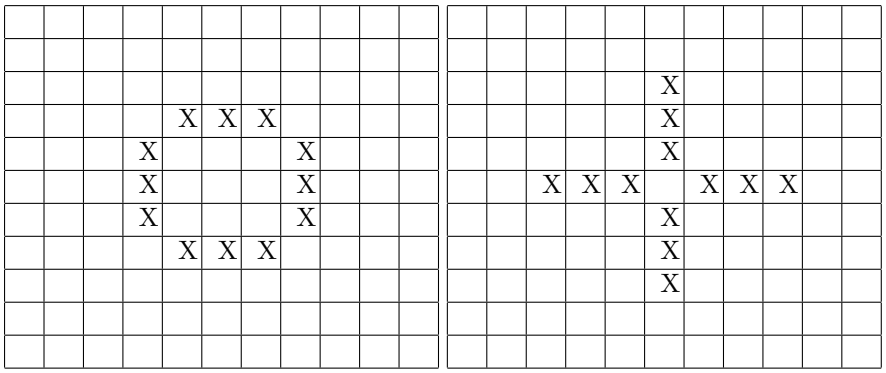
3.6 Le jeu de la vie

Le jeu de la vie (créé par le mathématicen John Horton Conway) est basé sur l'évolution de populations cellulaires. Les cellules sont disposées sur une grille. La naissance et la mort d'une cellule donnée sont conditionnées par la présence suffisante ou excessive de cellules vivantes dans son voisinage immédiat. On considère que l'espace est constitué d'une grille, dans laquelle chaque case a 8 voisins au maximum (moins pour les bords et les coins). Une case ne peut contenir qu'une seule cellule et les rde vie et mort sont les suivantes :

— si le voisinage d'une case vide contient trois cellules, alors une cellule nait ;
— si une cellule est entourée de plus de 4 cellules, elle meurt étouffée ;
— si une cellule est entourée d'une cellule ou moins, elle meurt d'isolement.

On voudrait pouvoir simuler l'évolution d'une population de cellules, avec des facilités de chargement/sauvegarde d'état.





Vous inclurez les fonctionnalités suivantes :

— considérer un espace torique (le haut et le bas se rejoignent, ainsi que la droite et la gauche)
— fixer les dimensions d'une grille
— éditer une configuration de grille
— choisir la vitesse de déroulement
— lancer une simulation
— permettre à l'utilisateur de modifier l'état du jeu pendant son déroulement
— sauvegarder/recharger une grille à partir d'un fichier

3.7 Thématiseur Général

Le logiciel de thématisation (thématiseur général) aura en entrée un document et récupèrera en provenance des thématiseurs spécialisés les pourcentages quíls obtiennent (c'est-à-dire les pourcentages de mots faisant partie de leur liste thématique de mots-clés associés). Il retourne alors la liste des thèmes (i.e. des thématiseurs) pour lesquels le pourcentage obtenu est supérieur à un certain seuil. Le thématiseur général peut ainsi parcourir la liste de tous les thématiseurs spécialisés pour découvrir les thèmes correspondant au document, mais l'utilisateur peut aussi sélectionner au préalable une liste de thèmes à parcourir (i.e. une liste de thématiseurs spécialisés à consulter) pour limiter la recherche des thèmes dans un sous-ensemble de thèmes initiaux.

Quelques idées pour vous permettre d'implémenter un jeu d'essais :

Thème astronomie : astronomie astronome soleil lune vénus saturne pluton neptune mercure galaxie planète planètes étoile étoiles satellites mars jupiter cosmique terre lunette télescope ciel lumière éclipse exploration astrophysique cosmogonie cosmos cosmographie longue-vue lorgnette astre comète nova filante météorite météore astérode aérolithe spoutnik lactée astral.

Thème science : science scientifique technologique médecine astronomie astrophysique chimie informatique mathématique biologie physique géologie paléontologie histologie géométrie empirisme logique empirique découverte scientifique cognition savoir expérimental algèbre algébrique formule preuve connaissances.

Thème horoscope : horoscope horoscopes poisson verseau bélier taureau gémeau cancer lion vierge balance scorpion sagittaire capricorne amour travail santé sensible coeur chance argent carrière célibataire soleil lune vénus mars saturne jupiter rencontre étoile ciel prédiction astrologie astrologue astrologues devin ésotérisme divination destin destinées signe signes astral ascendant décan.

Thème humour : rire histoires belges blondes blonde belge toto comique blague blagues comédie charade plaisanterie drôle galéjade canular farce farces hilarité pouffer esclaffer sourire rigoler amuser amuse amusant hilarant jeu bouffonnerie bouffon clown ironie malice noir anglais.

Document1 : Les astrologues sont de plus en plus consultés par les personnes fragiles lire leur horoscope les rassure et les amuse le ciel et les planètes ont toujours fasciné l homme et l astronomie n a pas su faire disparaître l astrologie

Document2 : Le rire est le propre de homme mais il y a des sens de l humour bien particulier j aime l humour anglais mais je déteste l humour des étudiants de médecine leurs farces sont souvent macabres et ne me font pas rire

Document3 : Je ne regarde jamais mon horoscope sur Internet je préfère y lire des blagues mes blagues préférées sont celles sur les blondes

Document4 : Une blonde entre dans une bibliothèque et demande un livre à la bibliothécaire de quel auteur interroge la bibliothécaire heu répond la blonde de vingt centimètres

Annexe A

Guide de réalisation des projets

A.1 Préliminaires

L'écriture d'un programme sur machine doit impérativement être précédée de l'analyse du problème et de la conception du logiciel. C'est la raison pour laquelle nous vous demandons de nous remettre un document avant de vous mettre à coder. Ce document nous permet de vous faire des remarques, et vous le corrigerez avant de vous mettre à la conception détaillée et au codage. La version corrigée sera ensuite incorporée au mémoire que vous nous remettrez en fin de projet.

Votre projet devra être réalisé en fonction des besoins et caractéristiques des futurs utilisateurs du logiciel. Vous devez concevoir le logiciel afin qu'il soit facilement utilisable pour les personnes auxquelles il est destiné. Ainsi il peut s'agir d'un enfant ou d'un adulte, de quelqu'un qui ne connait rien à l'informatique ou au domaine exploré, ou au contraire d'un expert du domaine ou d'un expert en informatique. Il est donc important de savoir à qui le logiciel est destiné, car cela détermine la forme de l'interface utilisateur et les fonctionnalités proposées.

Parallèlement, on doit déterminer quelle est la tâche effectuée par cet utilisateur potentiel. Le but du logiciel, c'est ce à quoi il sert, ce qu'il permet de faire, sa ou ses fonctions principales. Les ergonomes distinguent ici deux niveaux de description : celui de la tâche que permet d'accomplir le logiciel (par exemple, écrire une lettre pour un traitement de texte, jouer aux échecs pour un jeu d'échecs, gérer une bibliothàque, etc.), et celui des fonctionnalités proposées par le logiciel pour permettre d'accomplir la tâche (par exemple, mettre en gras, justifier à droite, définir des marges, etc. pour un logiciel de traitement de texte, ou se connecter en réseau, jouer une partie, pour un jeu d'échecs). Le niveau de la tâche est assez général et orienté "utilisateur", et celui des fonctionnalités est plus fin et plus orienté "programmation". Pour bien fixer tout cela, nous vous demandons de nous rendre un premier document contenant une présentation et une analyse approfondie de votre projet, ainsi qu'une liste détaillée des fonctionnalités du logiciel. La section qui suit vous donne des indications sur le contenu de cette première partie. La section suivante concerne la conception et vous aidera à rédiger la seconde partie concernant l'architecture logicielle qui vous est également demandée dans le premier document à remettre.

A.2 Analyse approfondie et spécifications fonctionnelles

Cette étape de la description du projet doit permettre de déterminer ce que le logiciel à développer devra permettre de faire (le quoi ?). A cette étape, on ne se préoccupe pas de savoir comment le logiciel permettra de réaliser ces fonctionnalités (on ne se préoccupe donc pas de l'interface utilisateur), ni bien entendu de comment ces fonctionnalités seront programmées. Cela signifie, en particulier, que cette partie ne ne doit pas dépendre ou mentionner des aspects d'implantation comme les caractéristiques d'un langage de programmation ou d'une machine. On ne se préoccupe à cette étape que des services que le logiciel doit rendre à l'utilisateur. En principe, c'est seulement après cette première étape que le choix du langage de programmation et de l'environnement logiciel et matériel sera effectué.

A.2.1 Analyse approfondie et analyse des besoins

L'analyse approfondie est consacrée à l'explicitation du contexte d'utilisation du logiciel et des besoins des utilisateurs. C'est la raison pour laquelle on parle d'analyse des besoins. Cette analyse répondra aux questions suivantes :

Quel est l'objectif du logiciel : à quoi doit-il servir, quel est le problème qu'il doit résoudre ? L'analyse commence par exposer ce sur quoi porte le logiciel, et ce qu'il permet de faire (la ou les tâches qu'il permet d'exécuter). En particulier, il faudra introduire les objets (ou entités abstraites) faisant partie du domaine sur lequel porte le logiciel (par exemple, des fichiers, des utilisateurs, et des arborescences de répertoires pour un gestionnaire de fichiers ; des pièces, des joueurs, des parties, des coups et des plateaux pour un jeu d'échecs ; des vins, des bouteilles et des étagères pour une cave à vin ; des expressions algébriques pour un logiciel de calcul formel, etc.). Il est important ici d'introduire et de fixer le vocabulaire lié au domaine.

Conseil : pour décrire ce que fait votre logiciel, vous pourrez partir de l'énoncé du sujet et d'un premier texte (de vous) décrivant le logiciel que vous souhaitez faire. Soulignez-y les mots qui vous semblent importants et essayez d'isoler de cette manière les entités (objets ou concepts clés) qui font partie de ce petit monde (=le domaine de l'application). Un programme est toujours la simulation d'un petit univers. Vous devez dans cette première phase de conception, lister les objets faisant partie de cet univers en vous aidant de la terminologie. Cela vous permet de mieux cerner ce que fait votre logiciel et vous permettra plus tard de choisir les structures de données ou les objets qui feront partie de votre programme. Les entités de votre domaine correspondront normalement à des noms, et les actions pouvant être exécutées à des verbes. Une fois cette analyse de texte effectuée, vous pourrez réécrire un texte de présentation du logiciel plus clair et plus précis, car vous en aurez fixé le vocabulaire.

A quel type d'utilisateur s'adresse le logiciel ? quel est son âge et son niveau de compétence dans le domaine couvert par le logiciel ? Ces informations seront essentielles pour la conception de l'interface utilisateur et le choix des fonctionnalités qui seront offertes.

Quel est le contexte d'utilisation du logiciel ? A quelle(s) occasion(s), l'utilisateur se servira-t-il du logiciel ? Quelles sont les différents types d'utilisation du logiciel ? A titre d'exemple, il arrive souvent qu'un logiciel nécessite une phase d'initialisation ou de configuration durant laquelle seules certaines fonctionnalités seront utilisées (et ces fonctionnalités ne seront que très rarement utilisées dans le contexte courant). Souvent, les différents contextes

d'utilisation correspondent à des types d'utilisateurs différents, par exemple, des utilisateurs novices ou des utilisateurs expérimentés.

Déterminer précisément les entrées et les sorties du logiciel : l'utilisateur devra parfois fournir des données au logiciel (les entrées) pour qu'il puisse accomplir sa tâche. Vous devez déterminer sous quelle forme l'utilisateur disposera de ces données afin de choisir la procédure d'entrée des données la plus adaptée. De la même façon, vous devez déterminer quelle est la manière la plus appropriée pour présenter les résultats du programme (les sorties) à l'utilisateur, cela dépendra en particulier de l'utilisation que fera l'utilisateur des résultats produits par le logiciel.

Caractéristiques quantitatives : vous devez déterminer la taille et la quantité de données que l'utilisateur doit pouvoir traiter avec le logiciel. Indiquer aussi s'il existe des limitations.

A.2.2 Liste des fonctionnalités du logiciel

En prenant en compte les points précédents, vous devez déterminer précisément ce que le programme permet de faire, c'est-à-dire établir la liste exhaustive de ses fonctionnalités. C'est la partie la plus importante de cette étape, et c'est celle qui nécessitera le plus gros travail de rédaction. Chaque fonctionnalité doit être décrite sans pour autant entrer dans des détails d'implantation ou de configuration matérielle. Il ne faut pas non plus expliciter comment les fonctionnalités seront proposées à l'utilisateur par l'interface utilisateur. Pour bien décrire les fonctionnalités, il faut au contraire faire abstraction de l'interface utilisateur. Une bonne manière pour vous de faire abstraction de l'interface utilisateur est d'imaginer que le logiciel n'a pas d'interface graphique et ne permet d'interagir avec l'utilisateur que dans une fenêtre de type console dans laquelle il pourrait taper des commandes. On essayera aussi dans la mesure du possible de présenter la liste des fonctionnalités du logiciel en les regroupant par sections, éventuellement hiérarchisées.

Pour chaque fonctionnalité, il faudrait idéalement aussi prévoir une procédure de test. Il s'agit ici de préparer la phase de test en décrivant des procédures et éventuellement des données qui permettront, une fois la programmation effectuée, de vérifier que la fonctionnalité est correctement implantée. (Ces tests pourront aussi être utilisés lors de la démonstration finale pour la soutenance du projet). Ici, vous pouvez dresser un tableau de validation indiquant pour chaque fonctionnalité les valeurs problématiques et l'état du test de validation (à faire, en cours, ou validé). Ce tableau de validation vous sera demandé en phase de conception mais ne fera pas nécessairement partie du premier document remis.

A.3 Conception

L'activité de conception consiste à enrichir la description du logiciel de détails d'implantation afin d'aboutir à une description très proche d'un programme. Elle se déroule en deux étapes : l'étape de conception architecturale et l'étape de conception détaillée. Cette seconde étape peut éventuellement donner lieu à une révision de la première.

A.3.1 Conception architecturale

L'étape de conception architecturale a pour but de décomposer le logiciel en composants simples et indépendants les uns des autres (ces composants sont aussi appelés modules) afin de simplifier l'activité de programmation qui suivra. Cette décomposition doit être faite de sorte que chaque module puisse être réalisé de

manière totalement indépendante des autres, par des programmeurs différents. La personne qui, à l'étape du codage, programmera un module doit pouvoir effectuer sa tâche sans avoir besoin de savoir comment les autres modules ont été, sont ou seront programmés.

C'est aussi à cette étape que l'on doit faire les choix contraignant l'implantation :

Configuration matérielle : sur quel type de machine fonctionnera le logiciel ? Votre logiciel nécessite-t-il la présence de périphériques particuliers (écran graphique, souris, imprimante, scanner, lecteur de CD-ROM, etc.) ?

Configuration logicielle : système d'exploitation, compilateur, bibliothèques de fonctions. Pour chacun de ces logiciels, vous devrez préciser le numéro de version.

Réutilisation de logiciels existants : Dans le cas particulier où vous faites une nouvelle version d'un logiciel déjà opérationnel, il faut choisir entre reprendre et modifier le logiciel ou le refaire complètement.

Choix de l'environnement de développement : l'environnement de développement est l'ensemble des logiciels que vous utiliserez pour la programmation (par exemple gcc, java ou eclipse). Il comprend, au minimum, un compilateur, un éditeur de texte et un débogueur. Ces logiciels peuvent ou non être accessibles à partir d'une unique application.

Pour réaliser le découpage en modules, on part des fonctionnalités qui ont été définies précédemment. Pour chaque fonctionnalité, on définit précisément son interface avec le reste du logiciel, c'est-à-dire l'ensemble des informations dont elle aura besoin pour effectuer sa tâche, et celles qu'elle pourra modifier. Puis on la décompose en modules plus simples, dont on précise la fonction réalisée et l'interface avec le logiciel. Ce processus de décomposition est réitéré jusqu'à obtenir des modules aussi simples que possible.

Lors de ce travail vous devez donner un nom le plus explicite possible à chaque module et aux données qu'il utilise ou produit.

Un autre but important de cette décomposition est de mettre en évidence des modules possédant des caractéristiques suffisamment proches pour être fusionnés en un même module plus général (cette factorisation réduira d'autant le travail de programmation). Dans le même ordre d'idée, lorsque des bibliothéques logicielles sont disponibles, le découpage en modules devra, autant que possible, permettre la réutilisation de modules déjà existant dans ces bibliothèques.

En même temps que ce découpage, vous devez faire apparaître les dépendances entre modules : quel module utilise quel autre module ? Le résultat de ce travail sera exprimé sous forme d'un graphe de dépendances : chaque module est représenté par une boîte et les dépendances sont représentées par des flèches entre ces boîtes.

Pour décrire l'architecture du programme, on fournira donc dans le premier document remis un découpage en modules (en choisissant soigneusement leurs noms). On donnera une brève description des modules et on les représentera sur un graphe de dépendances. Remarque : Il se peut qu'il existe plusieurs programmes, et une architecture particulière du système si ce dernier utilise plusieurs composants logiciels, comme par exemple, une architecture client/serveur. Mais on ne fait pas en réalité de distinction entre programme et module, et on représentera cette architecture globale sur le même diagramme en indiquant les dépendances des différents programmes (considérés comme des modules) par des flèches.

On aura en particulier un découpage classique si le logiciel possède une interface graphique, ou s'il utilise une base de données. Dans le cas où l'on a une interface graphique, on prévoie généralement un module destiné à l'interface. En effet le monde de l'interface est un monde à part de celui du domaine du logiciel. Il est

peuplé d'éléments comme des boutons ou des menus, des boîtes de dialogue, etc. La construction et l'agencement de ces éléments interactifs seront donc effectués dans le module de l'interface graphique, et ce dernier sera relié au reste de l'application par des appels de fonctions.

De même, si l'on souhaite utiliser une base de données, on pourra regrouper dans un même module tout ce qui concerne la gestion de cette base de données. Ce module interagira alors avec le reste de l'application en permettant l'interrogation et la modification de la base de données.

A.3.2 Conception de l'interface utilisateur

Les différentes fonctionnalités du logiciel seront accessibles via l'interface utilisateur. Mais une même fonctionnalité peut être accessible de différrentes manières : par exemple, par l'intermédiaire d'un menu et par l'intermédiaire d'un bouton, ou encore, via une combinaison de touches particulières (raccourci de commande) et par l'intermédiaire d'une barre d'outils. En outre, l'interface utilisateur n'est pas nécessairement une interface graphique avec clavier et souris. Néanmoins dans la plupart des cas, on pourra donner une version préliminaire de l'interface en présentant les différents écrans devant lesquels pourra se trouver l'utilisateur. Bien que la description de l'interface utilisateur ne figure pas dans le premier document que vous devez nous rendre, vous devrez, avant de vous lancer dans la conception détaillée et le codage, produire les documents suivants :

Réalisation d'une version préliminaire de l'interface : les différents écrans de l'interface utilisateur doivent être décrits, ainsi que la manière dont ils s'enchaînent en fonction des choix de l'utilisateur.

Rédaction d'une version préliminaire du manuel utilisateur : les fonctionnalités doivent être suffisamment bien décrites pour prévoir précisément comment le logiciel pourra être utilisé. Le manuel utilisateur ne vous est pas demandé initialement, mais vous devrez le fournir dans la documentation finale. Il sera rédigé du point de vue de l'utilisateur et pourra être constitué en partie de la description des différents écrans. Pensez donc lors de la rédaction de votre document de description de l'interface à garder le point de vue de l'utilisateur.

Mais avant de vous lancer dans la conception de l'interface utilisateur, il vous est vivement recommandé de lire le document qui figure dans l'annexe **??**.

A.3.3 Conception détaillée et Planning prévisionnel

L'étape de conception détaillée fournit pour chaque composant (ou module) une description de la manière dont les fonctions du composant sont réalisées et une description précise de la manière dont les autres modules pourront interagir avec lui. Pour que vous puissiez travailler en parallèle sur plusieurs modules, il faut en effet définir soigneusement pour chaque module les structures de données et les fonctions dont les autres modules auront besoin. C'est ce que l'on appelle l'interface logicielle.

C'est à cette étape que vous devez définir et nommer les structures de données qui seront utilisées dans le programme. Pour les principales structures de données vous devez donner une justification précise des choix effectués.

Lorsque le logiciel utilise des fichiers, c'est à cette étape que vous devez décrire très précisément le format de ces fichiers. Il s'agit de donner tous les détails nécessaires afin que chaque module utilisant ces fichiers puisse être programmé indépendamment des autres.

Pour les modules les plus importants, vous devez de plus donner un algorithme *en français* (et non pas dans un langage de programmation). A cette étape, il ne s'agit pas de commencer la programmation, mais seulement *de préparer* le travail de programmation en précisant la manière dont les fonctionnalités seront réalisées.

Vous devez ensuite établir un planning prévisionnel indiquant la liste des tâches à réaliser, la date et le début de chaque tâche, qui dans le groupe la réalise, en veillant à ce que l'enchaînement des tâches dans le temps soit correct : il faut réaliser d'abord les tâches nécessaires à la réalisation d'autres tâches. On ordonnera ou parallélisera en particulier les tâches de programmation des différents modules, mais il ne faudra pas oublier de lister aussi les tâches de documentation, d'apprentissage ou de recherche d'informations, ainsi que les tâches de rédaction ou de préparation de la démonstration. L'intégration des différents modules (par exemple l'intégration de l'interface graphique) ou l'intégration d'un module de base de données, réserve souvent des surprises et demande parfois des corrections. Il est donc important de réserver aussi du temps pour la phase d'intégration des différents modules.

Si vous en avez le temps, vous reprendrez le tableau de validation suggéré à la fin de la section **??**, en présentant cette fois, pour chaque module la liste des fonctionnalités à implanter, les valeurs d'entrées normales avec les résultats attendus (jeu de test) en précisant les cas particuliers qui peuvent se présenter, et l'état actuel de la validation (pas fait : les tests n'ont pas été passés ; en cours : les tests sont en cours, mais la mise au point n'est pas terminée, il reste des bogues ; OK : les tests ont été passés avec succès). La colonne "validation" sera remplie au fur et à mesure de la mise au point, après le codage, et restera donc vide pour le moment.

A.4 Codage

C'est seulement à cette étape que commence la programmation. Si les étapes précédentes ont été passées correctement chaque module peut être programmé indépendamment des autres et le travail de programmation pourra être réparti entre les différents participants du projet. Cette phase de programmation doit suivre certaines règles :

- se conformer strictement à la description du module qui a été faite lors de la conception détaillée (fonctionnalités à réaliser et interface avec le reste du logiciel) : tout écart à cette règle peut remettre en cause la réalisation de l'ensemble du projet et réduire à néant tout le travail déjà effectué sur les autres modules.
- adopter des standards de programmation (cf. annexe) pour le nommage des différents objets du langage (fichiers, type, variables, constantes, fonctions, procédures...), pour l'écriture des commentaires, pour la présentation du code (en particulier pour l'indentation), pour le traitement des erreurs (détection systématique des erreurs pouvant être causées par les appels systèmes, format des messages d'erreurs, ...).

Tests et mise au point

Le programme est exécuté sur les données de test et les résultats obtenus sont comparés avec ceux attendus. Lorsque les résultats obtenus diffèrent des résultats attendus, il faut corriger le programme et faire repasser tous les tests. La phase de tests est considérée comme terminée lorsque la version déboguée du logiciel passe tous les tests avec succès.

Annexe B

Conception de l'interface graphique

B.1 Distinction Interface/Application

Avant de réfléchir à la conception de l'interface, il convient de bien cerner cette notion. Pour l'organisation du code, on fait toujours, dit-on, une séparation nette entre l'interface et l'application. La justification, outre la modularité, est qu'il existe différents systèmes de fenêtrages (Macintosh, X Window et Windows). L'application doit donc être écrite de manière indépendante pour que seul le code de l'interface ait besoin d'être réécrit s'il faut porter l'application dans un autre environnement.

La partie application, c'est le coeur du logiciel, ce à quoi il sert, ce qu'il permet de faire, sa ou ses fonctions principales. Les fonctionnalités proposées pour accomplir les tâches de l'utilisateur constituent finalement l'essentiel de l'application, et le travail de l'informaticien est d'abord de les définir, puis de les implémenter. Mais il n'est pas toujours facile de les distinguer de l'interface. L'interface guide l'utilisateur dans la mise en oeuvre des fonctionnalités pour l'aider à réaliser sa tâche. L'interface ne consiste donc pas uniquement en un dispositif permettant la mise en oeuvre des fonctionnalités. Des notions liées à la tâche peuvent y apparaître.

La définition de l'interface passe par l'utilisation des dispositifs d'entrées/sorties. Ainsi, les interfaces utilisateurs ne sont pas nécessairement des interfaces graphiques utilisant un clavier et une souris. Les premiers ordinateurs avaient des interfaces non interactives, qui utilisaient des cartes perforées. Il y a trente ans, les logiciels n'utilisaient en général qu'une entrée (le clavier) et les sorties se faisaient (à l'écran ou sur imprimante) sous forme d'impression de caractères. Ces interfaces ont été appelées alpha-numériques, car elles ne manipulaient que des nombres ou des caractères. Aujourd'hui la plupart des interfaces sont dites graphiques, et elles impliquent un dispositif d'entrées/sortie de type écran/clavier/souris.

Le code de l'application est constitué des structures représentant les objets du monde sur lequel porte l'application (les fichiers et les répertoires pour un gestionnaire de fichiers, etc.), et des structures utilisées pour organiser ces données (structures d'attributs, listes, tables de hachages, BD, entités plus abstraites comme des modèles de données, etc.). C'est un univers sur lequel on définit les opérations et transformations permettant d'implémenter les fonctionnalités du logiciel.

La programmation objet vous permet de décrire facilement un univers d'objets (et vous donne divers avantages comme l'héritage, l'encapsulation, l'abstraction sur les procédures et les données, etc.), mais si vous programmez dans un autre langage, vous faites tout de même la même chose : vous représentez le monde de l'application et vous définissez les structures de données et les procédures nécessaires à la mise

en oeuvre des fonctionnalités du logiciel.

Le code de l'interface graphique représente un autre monde, celui des écrans actuels. Ce monde est complexe mais très conventionnel et a pour but de gérer l'interaction avec l'utilisateur. Il est constitué d'entités comme les fenêtres, les menus, les boutons, les ascenseurs, les boîtes de dialogues, etc., mais aussi d'entités plus abstraites comme les gestionnaires d'affichage, les modèles ou les vues d'objets complexes (par exemple modèles et vues de tables ou de listes en Java). La structuration du code est imposée par la bibliothèque graphique utilisée. Si vous utilisez Java, vous ferez une modélisation objet et suivrez le modèle de programmation par événement basé sur les écouteurs (modèle par délégation). Avec les toolkits X11 ou les librairies graphiques, vous serez amené également à suivre le modèle de programmation préconisé par les concepteurs de la librairie (pour Xt, les Callbacks par exemple). Ces modèles prévoient plus ou moins bien la communication entre l'interface et l'application, et on aura parfois des problèmes d'intégration si les langages de programmation utilisés dans l'application et l'interface sont différents. Quoiqu'il en soit, en cas d'interface graphique, on prévoira un module séparé pour la programmation de l'interface.

B.2 Conception de l'interface graphique

On dit toujours qu'une interface graphique doit être claire, simple et facile d'emploi. Nous avons ici cherché à isoler quelques règles à suivre pour réaliser cet objectif. La lecture de cette section vous permettra de focaliser sur les points importants à prendre en compte dans la spécification de vos écrans - points qui vous sembleront peut-être évidents, mais qui échappent pourtant souvent aux débutants.

B.2.1 Règles de conception

Nous avons essayé de lister des régles de conception d'une interface graphique assez générales. Dans les sections suivantes, nous en développerons quelques unes.

Règle 1 : Définir les termes utilisés. Pour la clarté de l'interface, il faut utiliser la terminologie du domaine de l'application. Il faut fixer les termes apparaissant dans l'interface avec soin. Cette tâche prend du temps mais est très importante. Ces termes apparaîtront dans les titres, les menus et les cadres. On réutilisera ensuite ces mêmes termes dans la documentation utilisateur et si possible dans le code. Cela permettra une meilleure compréhension pour l'utilisateur et le programmeur.

Règle 2 : Définir les icônes et les symboles. De même que les éléments textuels doivent être précisément définis, les icônes, symboles et autres objets graphiques apparaissant dans l'interface doivent être choisis avec soin et en accord avec les futurs utilisateurs.

Règle 3 : S'inspirer des interfaces existantes. L'interface Macintosh a été inspirée par celle d'une machine graphique initialement conçue par Rank Xerox. Elle aura elle même beaucoup inspiré Windows. Aujourd'hui, il y a une certaine standardisation qui s'effectue entre les différents systèmes, et des conventions s'instaurent à partir du moment où elles sont reprises par plusieurs grands logiciels.

Règle 4 : S'appuyer sur les standards. Pour que l'interface soit simple et claire, utilisez des conventions déjà connues de l'utilisateur. On procèdera ainsi pour le choix de certains termes figurant dans les menus, le choix des icônes, et pour la disposition générale des fenêtres et des objets graphiques.

On utilisera aussi les boîtes prédéfinies comme les sélecteurs de fichiers ou les boîtes de dialogue standards.

Règle 5 : Concevoir l'interface pour le novice et pour l'expert. En fait, il faut distinguer plusieurs types d'utilisateurs, car on peut être novice ou expert de la tâche, et novice ou expert de l'application. Pour le novice de l'application, le parcours de la barre de menu devra donner un panorama des possibilités du logiciel. Pour l'expert de l'application, on offrira des raccourcis claviers et des barres d'icônes, ou d'autres moyens (menus courts/menus longs, possibilités de configuration de l'interface).

Règle 6 : Guider l'utilisateur. L'utilisateur doit à chaque instant savoir ce qu'il est en train de faire (s'il vient de lancer une commande, si cette commande est en cours ou terminée, etc.) et pouvoir visualiser les résultats de ses commandes. On parle ici de *feed-back* (retour) utilisateur. Pour réaliser ce *feed-back*, on peut utiliser toutes sortes de moyens : des clignotements, des grisés, des animations, l'entraînement des icônes à l'écran, le tracé de rectangles pointillés pour sélectionner des zones, la sélection d'objets par de l'inverse vidéo (ou faire apparaître des poignées), des curseurs ou pointeurs particuliers (montres, etc.), des barres de défilement pour montrer l'écoulement du temps, etc. Il est important également que l'utilisateur puisse visualiser l'effet de ses commandes (mises à jour de vues, fenêtres de statuts ou de messages). L'utilisateur doit également savoir à chaque instant ce qu'il peut faire. On veillera à griser les commandes inaccessibles (textes ou icônes), indiquer par un beep sonore les actions interdites (textes non éditables), souligner à l'aide d'icônes l'état d'un document édité (sauvegardé, modifié), etc. Pour expliquer l'invalidation d'une commande, on affichera des messages d'erreurs. Ces messages apparaîtront dans des boîtes au centre de l'écran ou dans une petite fenêtre spécialisée (par exemple en bas de la fenêtre principale). Pour les boîtes de messages, on utilisera dans la mesure du possible les boîtes de dialogue toutes prêtes fournies par la bibliothèque utilisée.

Règle 7 : Donner le contrôle à l'utilisateur. L'utilisateur doit toujours avoir la main. S'il lance une tâche, il doit aussi pouvoir l'interrompre. Un bouton Annuler d'une boîte de lancement inutilisable quand le curseur de souris est un sablier a de quoi l'énerver à juste titre. Il faut également veiller à ce que les actions explicites, comme la demande d'une sauvegarde, soient effectives (par exemple, pas de sauvegarde en mémoire locale pour optimiser le programme). Il est également souhaitable que l'utilisateur puisse revenir en arrière, et annuler l'effet de sa dernière commande.

Règle 8 : Gérer les erreurs. Il faut prévoir une gestion des erreurs concernant la saisie de données, avec éventuellement des corrections automatiques, ou des valeurs par défaut.

Règle 9 Soigner la mise en page. Nous développerons quelques idées sur la mise en page dans les paragraphes suivants. Mais il n'est pas toujours évident d'analyser pourquoi une interface est belle et pourquoi une autre ne l'est pas.

Règle 10 : Regrouper spatialement les éléments. La clarté de l'interface provient en grande partie de la disposition géométrique des éléments. Le regroupement d'objets dans des zones spécifiques doit correspondre à un regroupement conceptuel. Cette règle est fondamentale. Elle s'applique partout dans l'interface : dans les barres de menus, dans les barres d'icônes, dans l'organisation de l'interface principale et des boîtes de dialogue. On regroupe ensemble les commandes correspondant à un groupe de fonctionnalités (portant par exemple sur des objets de même type), et l'on regroupe ensemble les objets de l'interface permettant la mise en oeuvre d'une même fonctionnalité (par exemple les boutons sur lesquels ils déclenchent des actions).

Règle 11 : Veiller à la cohérence. Par exemple, si vous avez plusieurs façons de déclancher une commande, la procédure déclanchée doit être exactement la même. (ça vous évite aussi de dupliquer le code). En Java, on pourra utiliser les Actions pour déclencher les mêmes commandes dans la barre d'icônes et dans la barre de menu. Il faut également avoir une utilisation cohérente des fontes et des couleurs. Pour que la documentation soit correcte, il faudra également utiliser les termes apparaissant dans l'interface de manière cohérente avec le manuel utilisateur, et si possible, retrouver ces mêmes termes dans le code.

Nous allons pour finir indiquer comment appliquer certaines de ces règles.

B.2.2 Définir les termes utilisés

Dans les barres de menus, on unifiera la syntaxe des termes utilisés : soit tout nominal, soit tout verbal. Cette remarque s'applique aussi aux titres des boîtes de dialogues. N'hésitez pas non plus à préciser un terme par un complément ou un adjectif. Le titre d'un menu doit recouvrir sémantiquement les différents items du menu. C'est un titre de rubrique. On veillera également à ne pas mélanger le français et l'anglais. Prévoir si vous avez le temps deux versions, en utilisant des tables pour les termes et les messages d'erreurs. Une fois fixée la terminologie, on évitera l'utilisation de synonymes dans la documentation. Par exemple, si on utilise le terme Couper dans un menu, on gardera ce terme partout, et on évitera les synonymes comme Supprimer, Effacer, etc. Cela évite des ambigutés et supprime le risque d'incohérences.

B.2.3 Soigner la mise en page

Utilisez a priori une disposition standard : une fenêtre principale avec en haut une barre de menu et une barre d'icônes ; en bas, un ascenseur éventuel et/ou une petite fenêtre pour les messages ; à droite, un ascenseur.

Mais il y a d'autres dispositions possibles. En particulier, les fenêtres à panneaux multiples (réajustables par l'utilisateur), ou des fenêtres à onglets (faciles à implémenter en Java). On peut également choisir de démarrer le logiciel avec une barre d'icônes au lieu d'une fenêtre principale. Cette solution est préconisée quand le logiciel est complexe et possède de nombreux types de fenêtres.

Faciliter la navigation

L'utilisateur doit pouvoir se repérer dans les différents écrans du logiciel. Savoir où il en est dans sa tâche, ce qu'il est en train de faire et comment revenir à une étape antérieure. Si votre disposition est standard, le problème ne se pose pas vraiment. Mais si vous devez afficher de nombreuses fenêtres, vous devez organiser la cohérence graphique de vos fenêtres et la navigation entre les fenêtres. Vous avez plusieurs solutions classiques :

- Utiliser des fenêtres à onglets.
- Utiliser une vue des fenêtres permettant de naviguer entre elles. Cette vue peut se présenter comme une liste de boutons représentant les fenêtres, comme une fenêtre menu, ou comme un arbre si l'organisation du logiciel s'y prête.
- Utiliser les menus Modes ou Fenêtres de la barre de menu.

Créer des zones et les nommer

Mettez des titres bien pensés aux fenêtres. En particulier, donnez toujours un titre aux boîtes de dialogue. Groupez les éléments opérant sur des objets de même type (ou concernant un même groupe de fonctionnalités) dans une même région. Procédez ainsi dans les boîtes de dialogue, mais aussi dans les barres de menus, les menus, les barres d'icônes, et utilisez des séparateurs ou des cadres titrés pour faire ressortir ce découpage.

Organiser les boîtes de dialogues

Encadrez et donnez des titres aux différentes zones d'une boîte de dialogue complexe. Pour cela utilisez des cadres (ou bords avec titre en Java). N'hésitez pas à ce qu'il y ait une certaine redondance entre les titres des zones et les labels utilisés dans la boîte. Vous pouvez également scinder l'espace de la fenêtre horizontalement (avec des séparateurs ou des panneaux réajustables) pour séparer les zones groupant des fonctionnalités différentes. Ainsi, si l'on veut organiser la spécification de certains paramètres, on les présentera par exemple sous forme de cases à cocher ou de champs étiquetés dans une même zone de la boîte avec un titre rappelant la fonctionnalité mise en oeuvre (par exemple, le titre Paramètres). On n'oubliera pas également de donner un titre général à la boîte (comme Spécifier les paramètres). Les boutons qui apparaissent habituellement dans les boîtes sont les boutons Ok, Annuler, Valider, Appliquer, Enregistrer, et Aide. Notez qu'il y a des emplacements standards pour ces boutons.

Les boutons doivent être placés de sorte qu'on voit immédiatement ce sur quoi ils portent. L'encadrement des zones permet de délimiter la portée des actions sur ces boutons. Ainsi, le bouton Supprimer situé près d'une liste dans une zone encadrée contenant la liste portera naturellement sur les items sélectionnés de la liste.

Un dernier point sur les boîtes : il ne faut pas qu'il y ait trop d'enchaînements de boîtes, car cela ennuie l'utilisateur (le cot d'accessibilité est trop élevé). Si l'utilisateur doit configurer beaucoup d'éléments, pensez plutôt à des boîtes complexes, ou revoyez la façon dont vous organisez l'accès aux fonctionnalités du logiciel.

Règles de mise en page :

Aligner les objets. Par exemple, on alignera les bords gauches des labels étiquetant des champs textuels.

Laisser des marges suffisantes. Ne collez pas vos étiquettes ou vos champs de textes aux bords des zones qui les contiennent.

Eviter les espaces vides. Cette règle s'applique à la conception des boîtes de dialogue. Une zone de travail vide n'est, bien entendu, pas gênante.

Minimiser le nombre de lignes verticales. Par lignes verticales, j'entends les lignes virtuelles qui apparaissent quand on aligne verticalement les bords gauche ou droit d'un groupe d'objets. Ces objets sont alors alignés sur une ligne (virtuelle) verticale que l'oeil perçoit. Cette ligne prend appui sur les bords alignés des objets. Ainsi, dans l'exemple précédent, si on aligne les bords gauches des labels et les bords gauches des champs de texte situés en vis-à-vis, on fera apparaître deux lignes verticales . Ne pas multiplier ces lignes consistera ici à aligner également d'autres objets de la même fenêtre sur ces lignes, comme par exemple, des boutons situés en dessous, ou le bord d'une zone encadrée située plus bas dans la boîte. Cela revient à aligner les différents objets parfois même au-delà des séparateurs de zones.

Placer toujours les mêmes choses aux mêmes endroits. C'est pour cela que l'on a choisi de griser les commandes inaccessibles dans les menus, plutôt que de les faire disparaître dynamiquement, car les items ne se seraient pas trouvés toujours à la même place dans le menu et cela aurait perturbé l'utilisateur. Dans cet esprit, uniformisez la mise en page des différentes boîtes ou fenêtres en plaçant les boutons, titres, etc. aux mêmes endroits. Adoptez un certain point de vue sur la manière dont l'utilisateur doit par exemple configurer des paramètres et maintenez une certaine cohérence entre vos boîtes. En particulier, n'hésitez pas dans le code à utiliser les mêmes panneaux si vos boîtes ont des parties communes.

B.2.4 Cohérence des fontes et des couleurs

Les fontes doivent normalement être spécifiées par l'utilisateur. Mais vous pouvez permettre à l'utilisateur de choisir sa fonte, et réserver l'italique ou le gras de la même fonte pour un usage particulier.

Néanmoins, il doit y avoir une cohérence dans l'usage des fontes de votre logiciel. On donnera un sens à leur usage. Par exemple, on pourra utiliser une fonte particulière pour les items de menu et faire apparaître le titre du menu dans une autre fonte (par exemple, en gras).

Surtout, ne pas multiplier les fontes. Si vous voulez plusieurs fontes, utilisez la même fonte dans différentes tailles, avec éventuellement du gras ou de l'italique. Car, outre l'effet général de confusion, rien n'est plus laid qu'un mélange de fontes.

Les couleurs de la même façon devront être utilisées de manière cohérente. On donnera un sens à l'usage d'une couleur. Par exemple, on pourra mettre en relief les lignes d'erreur dans un éditeur de programmes en leur affectant une certaine couleur, une autre étant réservée aux mots clés.

Le choix des couleurs est supposé aussi être du ressort de l'utilisateur. Vous pouvez cependant fixer l'usage d'une couleur, et laisser l'utilisateur choisir cette couleur. Il faudra cependant veiller dans certains cas à ce que les couleurs choisies, en particulier pour des couleurs de fonte et d'arrière plan, aient un contraste suffisant. Le contraste idéal est celui du blanc et du noir. Si l'on veut mettre de la couleur, on peut utiliser une couleur claire à la place du blanc et écrire en noir, ou à l'inverse, prendre une couleur foncée pour le fond et écrire en blanc.

Annexe C

Le mémoire du projet

Lors de la réalisation du projet, chaque étape a donné lieu à la rédaction de documents. Ces différents documents ont du être révisés au fur et à mesure que la réalisation avançait. Le document final du projet reprend l'essentiel de ces documents dans leur version finale, y ajoute certaines informations complémentaires, un manuel utilisateur et un manuel d'installation.

C.1 Présentation

Sa présentation est la même que tous les documents intermédiaires. Le mémoire doit être tapé à l'aide d'un traitement de texte. Il doit comporter une première page avec :

- un titre (mentionnant le sujet traité)
- le nom de(s) (l')auteur(s) et leur(s) courrier(s) électronique(s)
- la formation et l'année
- le nom des enseignanst encadrants auquel le document est destiné
- la date à laquelle cette version du mémoire a été terminée ou un numéro de version

Toutes les pages doivent être numérotées et le mémoire doit comporter une table des matières.

C.2 Plan du mémoire

Le mémoire proprement dit doit comporter les chapitres suivants :

1. L'énoncé du problème et son analyse approfondie, afin de décrire entièrement le projet à réaliser. Les problèmes (théoriques ou pratiques) sous-jacents à l'énoncé, la liste des fonctionnalités demandées ou nécessaires, les contraintes explicites ou non, l'environnement matériel et logiciel du projet, doivent être détaillés ; les choix effectués doivent être présentés avec leur justification.
2. La conception architecturale du logiciel, afin de définir les constituants du logiciel. Le projet est découpé en modules, chaque module regroupe des procédures ou fonctions. Cette partie du document comprend l'architecture logicielle (la découpe en modules et le graphe de dépendances).
3. La conception détaillée :
   - les paramètres et variables principaux, qui sont nommés et décrits,
   - la liste des en-têtes de procédures ou fonctions classés par module puis ordre alphabétique,

- la liste des données partagées et leur structure classées par ordre alphabétique,
- la liste des fichiers comportant les données ou les programmes avec leur structure,
- l'esquisse des algorithmes cruciaux.

4. Un manuel utilisateur en deux parties :
   - le manuel d'installation, qui donne :
     - la liste des fichiers nécessaires pour exécuter le logiciel (librairies, utilitaires, programme source, exécutable) en les commentant ; ne pas oublier de préciser les numéros de version de système ou de compilateur ; tout l'environnement nécessaire à l'exécution doit être décrit (on fournira éventuellement les adresses internet où les trouver) ;
     - les fichiers constituant le logiciel et la façon de les installer sur sa machine. (On donnera par exemple un fichier makefile et on dira précisément quelles commandes taper pour l'exécuter).
   - le manuel utilisateur proprement dit, qui décrira :
     - comment lancer le logiciel ;
     - comment l'utiliser : quelles sont les fonctionnalités possibles, quelles contraintes doivent être respectées, comment se déroule le dialogue utilisateur-logiciel (éventuellement, donner les écrans de l'interface utilisateur), que se passe-t-il en cas d'erreur (liste des messages d'erreur), comment quitter le logiciel (procédure normale et en cas d'erreur). Une bonne manide procéder pour cela est de faire un premier chapître présentant une session typique d'un utilisateur, puis de décrire explicitement ensuite toutes les fonctionnalités possibles du logiciel dans les chapîtres suivants.
5. Le planning prévisionnel remis avec la conception détaillée et le planning réel,
6. Le tableau de validation complété,
7. Le cas échéant, les fichiers de données avec lesquels vous avez testé votre logiciel ;
8. Une liste de références bibliographiques et de sites internet consultés. Les références bibliographiques sont les ouvrages (livres, articles, ...) que vous avez utilisés pour l'analyse du problème et la réalisation de votre projet. Ne faites apparaître que les ouvrages que vous avez directement consultés. De même, ne mentionnez que les sites qui vous ont été utiles.

L'ensemble de la documentation devra être rendu par mail, au format pdf, avec un fichier d'archive pour le code (format zip ou tar.gz uniquement).

Annexe D

Évaluation du projet

L'évaluation du projet sera faite au vu de :
— la qualité technique de la réalisation (en particulier l'adéquation du logiciel au problème posé, les fonctionnalités proposées et l'interface utilisateur)
— la qualité du mémoire (présentation et contenu, corrections effectuées après les remarques des encadrants)
— la présentation orale
— la qualité de la démonstration du logiciel
— l'organisation du travail en groupe.

La présentation orale

Vous devez présenter un bilan du projet dans une présentation PowerPoint (ou un fichier de transparents au format pdf). Cette présentation peut être collective et vous vous passerez alors tour-à-tour la parole. Vous commencerez bien sûr par présenter le sujet. Vous pouvez ensuite reprendre des spécifications du premier rapport comme la liste des fonctionnalités proposées ou des schémas de spécifications fonctionnelles, ainsi que l'architecture globale du logiciel (par exemple le diagramme de dépendances entre les modules). Faites ensuite un bilan en indiquant brièvement ce qui a été finalement développé (mais ne développez pas plus la présentation du logiciel car ce sera fait dans la démonstration de l'après-midi).

Vous pouvez aussi parler de l'organisation du travail (répartition et planning). Enfin, vous pouvez indiquer les problèmes rencontrés et dire comment ils ont pu être résolus pour mettre en avant votre travail, et indiquer les possibilités d'amélioration du logiciel. Vous conclurez par un bilan (collectif ou individuel) sur ce que vous aura apporté le projet sur le plan technique et humain.

La démonstration du logiciel

C'est lors de la démonstration que la qualité technique de la réalisation sera évaluée. Il faut donc préparer et répéter la démonstration soigneusement.

Pour cela :

1. définir des objectifs : ce que l'on veut montrer (si le code n'est pas complètement intégré vous pouvez prévoir de montrer le fonctionnement de fonctions isolées)
2. préparer un scénario d'utilisation permettant de satisfaire ces objectifs ; cela peut nécessiter l'utilisation de fichiers préparés d'avance

3. répéter et minuter votre démonstration en situation réelle, c'est-à-dire dans la configuration matérielle et logicielle qui sera celle de la véritable démonstration (en particulier, si vous avez réalisé le projet chez vous, il est très probable que vous devrez y apporter des modifications pour qu'il fonctionne correctement à l'Institut Galilée) et préparer vos commentaires oraux

Vous devez mettre en valeur ce que vous avez fait, et, seulement à la fin, expliquer les erreurs résiduelles et les fonctionnalités non opérationnelles. Les problèmes de gestion de groupe que vous pouvez avoir rencontrés ne doivent pas apparaître lors de la démonstration (ils doivent avoir été évoqués bien avant, au moment où ils se sont produits, avec votre enseignant).

Annexe E

Standards de programmation de GNU

Traduction partielle des “GNU Coding Standards” rédigés par Richard Stallman et disponible é `http://gnu.via.ecp.fr/prep/standards_toc.html`

E.1 Comportement commun à tous les programmes

Ce chapitre décrit comment écrire des logiciels robustes. Il décrit aussi des standards généraux pour les messages d'erreur, l'interface ligne de commande, et la manière dont les bibliothéques de fonctions doivent se comporter.

E.1.1 Ecrire des programmes robustes

Evitez les limites arbitraires sur la taille des structures de données, y compris la taille des noms de fichiers, des lignes, des fichiers et des symboles. Pour cela, utilisez l'allocation dynamique pour toutes les structures de données. Dans la plupart des utilitaires Unix, les lignes longues sont tronquées sans avertissement. Ce n'est pas toutefois pas acceptable.

Les utilitaires lisant des fichiers ne doivent pas éliminer les caractéres NUL ou tout autre caractère, *y compris ceux dont le code est inférieur à 0177*. Les seules exceptions admises concernent les utilitaires qui sont destinés spécifiquement à servir d'interface à certains types d'imprimantes qui ne peuvent pas gérer ces caractères.

Testez toujours le code d'erreur renvoyé par un appel système. Tout message d'erreur résultant de l'échec d'un appel système doit contenir le texte de l'erreur système (obtenu à l'aide de `perror` ou d'un équivalent) ainsi que le nom du programme dont l'appel système est issu et, le cas échéant, le nom du fichier. Afficher seulement “`cannot open foo.c`” ou “`stat failed`” n'est pas suffisant.

Testez tout appel à `malloc` ou `realloc` pour voir s'il renvoie zéro. Testez `realloc` même si vous diminuez la taille du bloc ; avec un système qui arrondit les tailles de bloc à une puissance de 2, `realloc` peut rendre un bloc différent si vous demandez moins d'espace mémoire.

Sous Unix, `realloc` peut détruire le bloc mémoire s'il renvoie zéro. Le `realloc` de GNU ne présente pas cette bogue : s'il échoue, le bloc original est inchangé. Vous pouvez faire l'hypothése que cette bogue a été corrigée. Si vous désirez exécuter votre programme sous Unix, et souhaitez éviter tout problème dans ce cas, vous pouvez utiliser le `malloc` de GNU.

Vous devez vous attendre à ce que `free` modifie le contenu du bloc qui a été

libéré. Tout ce que vous devez récupérer de ce bloc, doit être récupéré avant d'appeller `free`.

Si `malloc` échoue dans un programme non interactif, faites-en une erreur fatale (i.e. avec terminaison immédiate de l'exécution du programme). Pour un programme interactif (un programme qui lit des commandes de l'utilisateur), il est préférable de faire terminer la commande prématurément et de retourner à la boucle de lecture de commande. Cela permet à l'utilisateur de tuer d'autres processus pour libérer de la mémoire virtuelle et de réessayer la commande aprés.

Utilisez `getopt_long` pour décoder les arguments, à moins que la syntaxe des arguments en rende l'utilisation déraisonnable.

Quand de la mémoire statique (`static`) doit être modifiée lors de l'exécution d'un programme, utilisez du code C explicite pour l'initialiser. Réservez le mécanisme d'initialisation à la déclaration de C pour les données qui ne changeront pas.

Essayez d'éviter les références à des structures de données de bas niveau de Unix (telles que répertoires, `utmp` ou disposition de la mémoire du noyau Unix), car il est peut probable que ce soit compatible avec d'autres implantations d'Unix. A titre d'exemple, si vous avez besoin des noms de tous les fichiers d'un répertoire, utilisez `readdir` ou toute autre interface de haut niveau. Dans ce cas la compatibilité sera assurée par GNU.

Dans les tests d'erreurs qui détecte des conditions "impossibles", contentez vous de faire terminer le programme prématurément. En général, cela ne sert à rien d'afficher quelque message que ce soit. Ces tests indiquent la présence de bogues. Qui que ce soit voulant corriger ces bogues devra lire le code source et utiliser un débogueur. Expliquez donc l'erreur à l'aide de commentaires dans le code source. Les données intéressantes seront dans des variables qui pourront être examinées facilement avec le débogueur.

N'utilisez pas un compteur d'erreurs comme code de retour d'un programme. *Cela ne marche pas* car les valeurs de code de retour des programmes sont limitées à une représentation sur 8 bits (valeurs de 0 à 255). Il est possible qu'une seule exécution d'un programme produise 256 erreurs ; si vous essayez de renvoyer 256 comme code de retour, le processus père verra un code de retour égal à 0, et il supposera que le programme a réussi. Si vous créez des fichiers temporaires, testez la variable d'environnement `TMPDIR` ; si cette variable est définie, utilisez le répertoire spécifié à la place de `/tmp`.

E.1.2 Bibliothèques de fonctions

Voici maintenant quelques conventions de nommage concernant les bibliothèques de fonctions, afin d'éviter les conflits de nom.

Choisissez pour chaque bibliothèque un préfixe de plus de deux caractéres. Tout nom de fonction ou variable externe doit commencer par ce préfixe. De plus, chaque membre de la bibliothéque ne doit contenir qu'un symbole externe. Cela signifie habituellement que chacun doit être placée dans un fichier source distinct.

Une exception peut-être faite lorsque deux symboles externes sont toujours utilisés ensemble de sorte qu'aucun programme raisonnable n'en utiliserait un sans utiliser l'autre ; dans ce cas, ils peuvent aller tous les deux dans le même fichier.

Les symboles externes qui ne sont pas des points d'entrée documentés pour l'utilisateur (et ne lui sont donc pas destinés) doivent avoir des noms commençant par '_'. Ils doivent eux aussi contenir le préfixe choisi pour la bibliothèque afin de prévenir des conflits avec d'autres bibliothèques. Ils peuvent être placés dans les mêmes fichiers avec des points d'entrée pour l'utilisateur.

E.1.3 Formattage des messages d'erreur

Les messages d'erreurs des programmes non interactifs doivent ressembler à cela :
programme :*nom-fichier-source* :*num-ligne* : *message*
quand il existe un nom de fichier source approprié, ou à cela :
programme : *message*
quand il n'y a pas de fichier source pertinent.

Pour un programme interactif (un programme qui lit des commandes d'un terminal), il est préférable de ne pas inclure le nom du programme dans le message d'erreur. L'endroit adéquat pour indiquer quel programme s'exécute, est l'invite (prompt) ou un emplacement spécifiquement réservé dans l'agencement de l'écran. (Quand le même programme s'exécute en prenant son entrée à une autre source qu'un terminal, il n'est pas interactif et devra donc écrire les messages d'erreur dans le style non interactif).

La chaîne de caractéres message ne doit pas commencer par une majuscule quand elle suit un nom de programme ou de fichier. En outre, elle ne doit pas se terminer par un point.

Les messages d'erreur des programmes interactifs ainsi que les autres messages comme les messages d'utilisation (usage), doivent commencer par une majuscule mais ne doivent pas se terminer par un point.

E.1.4 Standards pour l'interface ligne de commande

Faites en sorte que le comportement d'un logiciel ne dépende pas du nom utilisé pour lancer la commande. Il est parfois utile de faire un lien vers un logiciel en utilisant un nom différent, et cela ne doit pas modifier ce qu'il fait.

Au lieu de cela, utilisez des options d'exécution ou des options de compilation ou bien les deux pour sélectionnez parmi les différents comportements alternatifs.

De la même façon, ne rendez pas le comportement d'un programme dépendant du type de périphérique de sortie avec lequel il est utilisé. L'indépendance par rapport aux périphériques est un principe important de conception des systèmes ; ne le compromettez pas dans le seul but d'éviter à quelqu'un de taper une option par ci par là.

Si vous pensez qu'un comportement est plus adéquat quand la sortie est dirigée vers un terminal, et qu'un autre est plus pertinent lorsque la sortie est dirigée vers un fichier ou un tube (pipe), alors, habituellement, le mieux est de choisir comme comportement par défaut celui qui est le plus pertinent avec un terminal de sortie et de proposer une option pour l'autre comportement.

La compatibilité nécessite que certains programmes soient dépendants du type de périphérique de sortie. Il serait désastreux que `ls` ou `sh` ne le fasse pas de la manière à laquelle tous les utilisateurs s'y attendent. Dans certains cas, nous ajoutons au programme une version alternative préférée ne dépendant pas du type de périphérique de sortie. Par exemple, nous fournissons un programme appellé `dir`, très proche de `ls` excepté que son format de sortie par défaut est toujours multi-colonne.

Il est préférable de suivre les principes proposés par POSIX pour les options de la ligne de commande d'un programme. La manière la plus simple de le faire est d'utiliser `getopt` pour les extraire de la ligne de commande. Notez que, normalement, la version GNU de `getopt` autorisera des options n'importe où parmi les arguments à moins que l'argument spécial '`--`' ne soit utilisé. Ce n'est pas ce que spécifie POSIX ; c'est une extension GNU.

Définissez des options à noms longs, équivalentes aux options en une lettre de style Unix. Nous espérons de cette manière rendre GNU plus convivial. C'est facile à mettre en œuvre avec la fonction GNU `getopt_long`.

L'un des avantages des options à noms longs est qu'elles peuvent être consistantes d'un programme à l'autre. Par exemple, les utilisateurs doivent pouvoir s'attendre à ce que l'option ¡¡ verbeuse ¿¿ de tout programme GNU en possédant une s'écrive exactement '`--verbose`'.

Habituellement, il est préférable que les noms de fichier donnés comme arguments ordinaires ne puissent être que des fichiers d'entrée ; tout fichier de sortie devra être spécifié à l'aide d'options (de préférence '`-o`' ou '`--output`'). Même si, pour des raisons de compatibilité, vous autorisez des noms de fichiers de sortie comme arguments ordinaires, essayez d'offrir en plus la possibilité de les spécifier à l'aide d'une option. Cela conduira é une plus grande consistance entre les utilitaires GNU et réduira le nombre de spécificités dont les utilisateurs devront se rappeler.

Tout programme doit supporter deux options standards : '`--version`' et '`--help`'.

E.1.5 `--version`

Tout programme appelé avec cette option doit afficher sur la sortie standard des informations sur son nom, sa version, son origine et son statut légal, ensuite de quoi le programme se termine normalement. Les autres options et arguments doivent être ignorés dès que cette option est reconnue et le programme ne doit pas assurer son fonctionnement normal. La première ligne est censée pouvoir être analysée facilement par un programme ; le numéro de version est placé juste après le dernier espace. De plus, elle contient le nom canonique de ce programme sous le format suivant :

```
GNU Emacs 19.30
```

Le nom du programme doit être une chaîne de caractères constante ; *ne la calculez pas* à partir de `argv[0]`. L'idée est de donner le nom standard ou canonique du programme, pas son nom de fichier. Il existe d'autres procédés pour retrouver le nom exact du fichier trouvé par l'intermédiaire de `PATH` pour une commande donnée. Si le programme est une partie annexe d'un package plus important, mentionnez le nom du package entre parenthèses de la manière suivante :

```
emacsserver (GNU Emacs) 19.30
```

Si le package a un numéro de version différent du numéro de version du programme, vous pouvez mentionner le numéro de version du package juste avant la parenthèse fermante. S'il est nécessaire de mentionner les numéros de version des bibliothèques qui sont distribuées séparément du package qui contient ce programme, vous pouvez le faire en affichant une ligne supplémentaire d'information sur la version pour chaque bibliothèque que vous voulez mentionner. Pour ces lignes, utilisez le même format que pour la première ligne. Ne mentionnez pas toutes les bibliothèques utilisées par le programme "juste par complétude", cela produit un fouillis inutile. Ne mentionnez les numéros de version des bibliothèques que si vous trouvez que c'est très important pour vous pour le déboguage. La ligne suivant la ou les lignes de numéro de version doit être une notice de copyright. Si plusieurs notices de copyright sont nécessaires, mettez-les chacune sur des lignes séparées. Ensuite, doit suivre une brève déclaration sur le fait que le programme est un logiciel libre, et que les utilisateurs sont libres de le copier et le modifier sous certaines conditions. Si le programme est couvert par la GNU GPL, dites le ici. Mentionnez aussi qu'il n'y a aucune garantie dans les limites permises par la loi. Vous pouvez terminer le message affiché par une liste des principaux auteurs du programme, afin de leur en attribuer le crédit. Voyez ci-aprés un exemple d'affichage respectant ces règles :

```
GNU Emacs 19.34.5
Copyright (C) 1996 Free Software Foundation, Inc.
GNU Emacs comes with NO WARRANTY,
to the extent permitted by law.
```

```
You may redistribute copies of GNU Emacs
under the terms of the GNU General Public License.
For more information about these matters,
see the files named COPYING.
```

Bien entendu, vous devez adapter cela à votre programme en y mettant les informations correctes concernant l'année, le détenteur du copyright, le nom du programme, le cadre autorisé pour la redistribution et en modifiant les termes employés autant que nécessaire. Dans cette notice de copyright, il n'est nécessaire de mentionner que l'année la plus récente à laquelle des modifications ont été apportées ; il n'est aucunement nécessaire de lister les années concernant les modifications correspondant aux versions précédentes. Vous n'avez pas à mentionner le nom du programme dans ces notices, puisqu'il a déjà été mentionné à la première ligne.

E.1.6 `--help`

Cette option doit afficher sur la sortie standard une documentation brève sur la manière d'appeler le programme, ensuite de quoi le programme se termine normalement. Les autres options et arguments doivent être ignorés dès que cette option est reconnue et le programme ne doit pas assurer son fonctionnement normal. Vers la fin de cet affichage, il doit y avoir une ligne expliquant où envoyer un mail pour signaler un bogue. Cette ligne doit respecter le format suivant :

`Report bugs to` *`mailing-address`*`.`

E.1.7 Utilisation de la mémoire

Si vous n'utilisez que quelques mégas de mémoire, ne vous préoccupez pas de faire des efforts pour réduire l'utilisation de la mémoire. Par exemple, si, pour de toutes autres raisons, il n'est pas envisageable en pratique de travailler sur des fichiers de plus de quelques mégaoctets, il est raisonnable de charger entièrement les fichiers d'entrée en mémoire pour travailler dessus.

Cependant, pour des programmes comme `cat` ou `tail`, qui opèrent utilement sur de très grands fichiers, il est important d'éviter d'utiliser une technique qui limiterait artificiellement la taille des fichiers qu'ils pourraient traiter. Si un programme travaille ligne par ligne et doit s'appliquer à des fichiers quelconques fournis par l'utilisateur, il ne doit conserver qu'une ligne en mémoire, car cela n'est pas très difficile à mettre en œuvre et que les utilisateurs voudront être en mesure de travailler sur des fichiers d'entrée plus grands que ceux qui pourraient tenir en mémoire.

Si votre programme crée des structures de données complexes, faites le en mémoire et faites terminer le programme prématurément (fatal error) si `malloc` retourne zéro.

E.2 Faire le meilleur usage de C

Ce chapitre fournit des conseils sur comment faire le meilleur usage du langage C lors de l'écriture de logiciels GNU.

E.2.1 Formater votre code source

Il est important de placer en colonne zéro les accolades ouvrantes qui marquent le début du corps d'une fonction C, et d'éviter de placer toute autre accolade ouvrante, parenthése ouvrante ou crochet ouvrant en colonne zéro. Différents outils recherchent les accolades ouvrantes en colonne zéro pour déterminer les débuts de

fonctions C. Ces outils ne fonctionneront pas si vous ne formater pas votre code source de cette manière.

De même, pour les définitions de fonctions, il est important de faire commencer le nom de la fonction en colonne zéro. Cela aide les gens dans la recherche de définitions de fonctions et peut aussi aider certains outils à les reconnaître. Le formatage correct est donc :

```
static char *
concat (s1, s2)        /* Name starts in column zero here */
char *s1, *s2;
{                     /* Open brace in column zero here */
  ...
}
```

ou bien, si vous voulez utiliser ANSI C, formater la définition de cette manière :

```
static char *
concat (char *s1, char *s2)
{
  ...
}
```

En ANSI C, si les arguments ne tiennent pas correctement sur une ligne, scinder la liste des arguments de la manière suivante :

```
int
lots_of_args (int an_integer, long a_long, short a_short,
              double a_double, float a_float)
...
```

Pour le corps de la fonction, nous préférons un code formaté comme ceci :

```
if (x < foo (y, z))
  haha = bar[4] + 5;
else
  {
    while (z)
      {
        haha += foo (z, z);
        z--;
      }
    return ++x + bar ();
  }
```

Il est plus facile de lire un programme quand il y a des espaces avant les parenthèses ouvrantes et après les virgules. Plus particuliérement après les virgules.

Lorsque vous scinder une expression sur plusieurs lignes, scindez la avant un opérateur, pas après. Ceci est la bonne manière de faire :

```
if (foo_this_is_long && bar > win (x, y, z)
    && remaining_condition)
```

Essayez d'éviter d'avoir deux opérateurs de précédences différentes au même niveau d'indentation. Par exemple, n'écrivez pas ceci :

```
mode = (inmode[j] == VOIDmode
        || GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])
        ? outmode[j] : inmode[j]);
```

Au lieu de cela, utilisez des parenthèses supplémentaires de sorte que l'indentation illustre l'imbrication :

```
mode = ((inmode[j] == VOIDmode
         || (GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])))
        ? outmode[j] : inmode[j]);
```

Insérez des parenthèses supplémentaires, de sorte que Emacs indente le code correctement. A titre d'exemple, l'indentation suivante peut sembler correcte si vous la faites à la main, mais Emacs sabotera votre beau travail :

```
v = rup->ru_utime.tv_sec*1000 + rup->ru_utime.tv_usec/1000
    + rup->ru_stime.tv_sec*1000 + rup->ru_stime.tv_usec/1000;
```

L'ajout de quelques parenthèses résoudra le problème :

```
v = (rup->ru_utime.tv_sec*1000 + rup->ru_utime.tv_usec/1000
     + rup->ru_stime.tv_sec*1000 + rup->ru_stime.tv_usec/1000);
```

Formatez les structures de contrôle do-while comme ceci :

```
do
  {
    a = foo (a);
  }
while (a > 0);
```

Utilisez des caractéres de saut de page (control-L) pour séparer votre programme en pages à des endroits logiques (mais pas à l'intérieur d'une fonction). La longueur des pages n'a aucune importance, il n'est pas nécessaire qu'elle corresponde à la longueur d'une page imprimée. Une ligne contenant un saut de page ne doit rien contenir d'autre.

E.2.2 Commenter votre travail

Tout programme doit commencer par un commentaire disant brièvement à quoi il sert. Exemple : '`fmt - filter for simple filling of text`'.

Dans un programme GNU, écrivez les commentaires en anglais, l'anglais est la langue que pratiquement tous les programmeurs de tous les pays peuvent lire. Si vous n'écrivez pas bien anglais, écrivez vos commentaires en anglais aussi bien que vous le pouvez puis demandez à d'autres personnes de vous aider à les réécrire. Si vous ne pouvez pas écrire de commentaires en anglais, trouvez quelqu'un pour travailler avec vous et traduire vos commentaires en anglais.

Pour chaque fonction, mettez un commentaire décrivant ce que fait la fonction, quelle sorte d'arguments elle prend ainsi que ce que signifient et à quoi sont utilisées les différentes valeurs possibles des arguments. Il n'est pas nécessaire de répéter avec des mots la signification des déclarations C des arguments, si un type C est utilisé de manière habituelle. Si il y a quelque chose de non standard dans son utilisation (comme un argument de type `char *` qui est en réalité l'adresse du second caractère d'une chaîne de caractères et non celle du premier), où que l'une des valeurs possibles ne fonctionnerait pas de la manière à laquelle on peut s'attendre (tel qu'un fonctionnement correct non garantie pour des chaînes de caractères contenants des sauts de lignes), alors vous devez obligatoirement le mentionner.

Expliquez aussi la signification de la valeur retournée, si elle en a une.

Afin que les commandes Emacs sur les phrases fonctionnent, mettez deux espaces après la fin d'une phrase dans vos commentaires. Ecrivez des phrases complètes et

mettez une majuscule au premier mot. En revanche, si un identificateur en minuscules vient en début de phrase, ne lui mettez pas de majuscule ! Changer une des lettres en fait un identificateur différent. Si vous n'aimez pas commencer une phrase par une lettre minuscule, écrivez la phrase autrement.

Le commentaire d'une fonction est plus claire si vous utilisez le nom des arguments pour parler des valeurs des arguments. Le nom de variable lui-même doit être en minuscule, mais écrivez en capitales quand vous parlez de la valeur plutôt que de la variable elle-même. Ainsi, "the inode number NODE_NUM" plutôt que "an inode".

Normalement, redonner le nom de la fonction dans le commentaire qui la précède n'a aucun intérêt, puisque le lecteur peut le voir par lui-même. Il peut néanmoins y avoir une exception lorsque le commentaire est si long que la fonction elle-même pourrait ne pas apparaître à l'écran.

Il doit aussi y avoir un commentaire pour chaque variable statique (`static`), comme :

```
/* Nonzero means truncate lines in the display;
   zero means continue them.  */
int truncate_lines;
```

Tout '`#endif`' doit avoir un commentaire associé, excepté dans le cas de structures conditionnelles courtes (juste quelques lignes) et sans imbrication. Le commentaire doit donner la condition de la structure conditionnelle qui se termine, ainsi que sa signification. Tout '`#else`' doit avoir un commentaire décrivant la condition et la signification du code qui le suit. Par exemple :

```
#ifdef foo
  ...
#else /* not foo */
  ...
#endif /* not foo */
```

en revanche, écrivez les commentaires de cette manière pour un '`#ifndef`' :

```
#ifndef foo
  ...
#else /* foo */
  ...
#endif /* foo */
```

E.2.3 Utilisation propre des constructions C

Déclarez explicitement tous les arguments des fonctions. Ne les omettez pas sous prétexte que ce sont des `int`.

Les déclarations de fonctions externes et de fonctions apparaissant plus loin dans le fichier source doivent toutes être placées à un endroit vers le début du fichier (quelque part avant le première définition de fonction du fichier) ou bien dans un fichier header (avec suffixe `.h`). Ne mettez pas de déclarations `extern` é l'intérieur des fonctions.

Evitez de réutiliser toujours les mêmes variables locales (avec des noms comme `tmp`) pour des valeurs différentes à l'intérieur d'une même fonction. Pour éviter cela, il est préférable de déclarer une variable locale distincte pour chaque utilisation distincte, et de lui donner un nom porteur de sens. Ce n'est pas uniquement pour rendre les programmes plus simples à comprendre, mais aussi parce que cela facilite l'optimisation faite par les bons compilateurs. Vous pouvez aussi déplacer

la déclaration de chaque variable locale dans le plus petit bloc incluant toutes ses utilisations. Cela rend le programme encore plus propre.

N'utilisez pas de variables locales ou de paramètres masquant des identificateurs globaux.

Ne déclarez pas plusieurs variables en une déclaration s'étendant sur plusieurs lignes. Au lieu de cela, commencez une nouvelle déclaration à chaque ligne. Par exemple, au lieu de ceci :

```
int    foo,
       bar;
```

écrivez soit cela :

```
int foo, bar;
```

soit cela :

```
int foo;
int bar;
```

(Si ce sont des variables globales, chacune doit être précédée d'un commentaire.)

Quand vous avez un `if-else` imbriqué dans un autre `if`, mettez toujours des accolades autour du `if-else`. Ainsi, n'écrivez jamais ceci :

```
if (foo)
  if (bar)
    win ();
  else
    lose ();
```

écrivez toujours cela :

```
if (foo)
  {
    if (bar)
      win ();
    else
      lose ();
  }
```

Si vous avez un `if` imbriqué dans un `else`, ou bien écrivez `else if` sur une ligne comme ceci,

```
if (foo)
  ...
else if (bar)
  ...
```

avec la partie `then` indentée comme la partie `then` précédente, ou bien écrivez le `if` imbriqué entre accolades comme cela :

```
if (foo)
  ...
else
  {
    if (bar)
      ...
  }
```

Ne déclarez pas à la fois une structure et des variables ou des définitions de types dans la même déclaration. Au lieu de cela, déclarez la structure séparément et utilisez la ensuite pour déclarer les variables ou les types.

Essayez d'éviter les affectations à l'intérieur des conditions des `if`. Par exemple, n'écrivez pas ceci :

```
if ((foo = (char *) malloc (sizeof *foo)) == 0)
  fatal ("virtual memory exhausted");
```

écrivez cela à la place :

```
foo = (char *) malloc (sizeof *foo);
if (foo == 0)
  fatal ("virtual memory exhausted");
```

Ne rendez pas le programme inutilement compliqué, juste pour satisfaire les exigences de `lint`. Ne mettez jamais de cast pour un `void`. Un zéro sans cast convient à la perfection pour une constante pointeur nulle, sauf lors d'un appel de fonction à nombre d'arguments variables.

E.2.4 Choix des identificateurs de variables et de fonctions

Dans un programme, les noms des variables globales et des fonctions constituent une forme de commentaire. En conséquence, plutôt que d'utiliser des noms concis, cherchez plutôt des noms qui donnent des informations utiles sur la signification de la variable ou de la fonction. Dans un programme GNU, de la même manière que pour les commentaires, les identificateurs doivent être en anglais.

Les noms de variables locales peuvent être plus courts, car ils sont utilisés uniquement dans un seul contexte où des commentaires expliquent leur rôle.

Utilisez des caractères underscore pour séparer les mots dans un identificateur, afin que les commandes sur les mots de Emacs puissent leur être appliquées utilement. Tenez vous en aux lettres minuscules ; réservez les majuscules aux constantes définies par macros ou `enum` et aux préfixes qui obéissent à une convention uniforme.

Vous pouvez, par exemple, utiliser des noms comme `ignore_space_change_flag` ; mais n'utilisez pas de nom comme `iCantReadThis`. Les variables indiquant si des options de la ligne de commande ont été spécifiées doivent être nommées d'après la signification de l'option et non d'après la lettre correspondant à l'option. Un commentaire doit indiquer à la fois la signification de l'option et sa lettre. Par exemple :

```
/* Ignore changes in horizontal whitespace (-b).  */
int ignore_space_change_flag;
```

Quand vous voulez définir des noms avec des valeurs constantes entières, utilisez `enum` plutôt que '`#define`'. GDB reconnaît les constantes énumérations.

Utilisez des noms de fichier de 14 caractères ou moins, afin d'éviter des problèmes avec les plus anciennes versions du système System V. Vous pouvez utilisez le programme `doschk` pour tester cela. `doscheck` teste aussi les conflits de nom potentiels si les fichiers étaient chargés sur un système de fichier MS-DOS – libre à vous de vous en soucier ou non.