Projet - Algorithmique et base de la programmation
ASIspell, un correcteur orthographique

Nicolas Delestre

1 Cahier des charges

L'objectif de ce projet est de développer un correcteur orthographique à l'image des programmes unix *ispell* et *aspell* [1]. Les fonctionnalités attendues de ce programme sont les suivantes.

1.1 Aider

Lorsqu'il est lancé sans option, ou avec l'option `-h`, ou avec une option non reconnue, le programme doit afficher une aide.

Par exemple :

```
./asispell
Aide de asispell :
  asispell -h : cette aide
  asispell -d dico : correction de l'entree standard a l'aide du
     dictionnaire dico
  asispell -d dico -f fic : completer le dictionnaire dico a l'aide
  des mots du fichier fic (un mot par ligne)
```

1.2 Compléter un dictionnaire

Il doit être possible de compléter un dictionnaire à l'aide des mots d'un fichier texte (un mot par ligne). Les options sont alors :

-d pour spécifier le nom du fichier correspondant au dictionnaire utilisé ;

-f pour spécifier le nom du fichier contenant les mots à ajouter.

L'exemple suivant ajoute les mots du fichier `dico-ref_asscii.txt` (téléchargeable sur moodle) au dictionnaire `francais.dico` :

```
./asispell -d francais.dico -f dico-ref-ascii.txt
```

1. l'encodage utilisé sera la norme ISO 8859-1 (l'ASCII étendu UNIX)

1.3 Corriger

À l'image du programme *ispell*, le programme doit pouvoir analyser (et proposer des corrections orthographiques si besoin) un texte qui lui est donné via l'entrée standard. Cette analyse est dépendante d'un dictionnnaire (option `-d`). Cette analyse sera produite sur la sortie standard avec un mot analysé par ligne. Deux cas de figure sont traités :

1. un mot est bien orthographié (présent dans le dictionnaire), le programme produit une étoile ('*') pour ce mot ;
2. un mot est mal orthographié, le programme produit un « et commercial » (&) suivi des informations suivantes :
   - le mot mal orthographié ;
   - le nombre de corrections proposés ;
   - le position du mot mal orthographié depuis le début du flux d'entrée, suivi de deux points
   - les corrections proposées séparées par un espace.

Voici un exemple d'utilisation :

```
echo "␣yn␣pett␣␣␣tset␣de␣corection␣orthographyque␣avec␣quelques␣
    fotes␣d'orthographe." | ./asispell -d francais.dico
& yn 7 1: yin yen un on in en an
& pett 2 4: petit peut
& tset 1 11: test
*
& corection 1 19: correction
& orthographyque 1 29: orthographique
*
*
& fotes 16 58: foutes fortes fontes fores foies fûtes fîtes fêtes
    votes rotes potes notes lotes dotes cotes botes
& d 11 64: dû dé du do dm dl dg de à y a
*
```

2 Comment corriger un mot ?

Pour proposer des corrections d'un mot mal orthographié (pour un dictionnnaire donné) le plus simple est de partir de ce mot et de lui appliquer des transformations qui le transforment en des mots proches. Si une transformation donne un mot bien orthographié alors ce mot est une correction possible. Voici quelques exemples de transformation :

- remplacement d'une lettre
- inversion de deux lettres consécutives ;
- suppression d'une lettre ;
- insertion d'une lettre ;
- décomposition d'un mot en plusieurs mots ;
- proposition d'un mot phonétiquement proche ;
- etc.

Dans l'exemple précédent, ce sont ces trois premières stratégies qui sont utilisées (c'est pour cela que le programme n'arrive pas à proposer une correction valable pour le mot « fote ».

L'inconvénient de cette méthode est qu'elle génère beaucoup de mots. Par exemple pour un mot de 5 lettres, la premières stratégies va générer $5 \times (26+16) - 5 = 205$ mots (26 lettres de l'alphabet et 16 versions accentuées). Il faut donc que être capable de vérifier si un mot est présent dans un dictionnaire de manière efficace (le fichier `dico-ref_asscii.txt` proposé contient 336531 mots).

3 Travail demandé

3.1 Analyse

On considère que l'on possède les types suivants :

Type Mode = {lecture,ecriture}

Nom:	FichierTexte	
Utilise:	**Chaine de caracteres**, Mode,**Caractere**,**Booleen**	
Opérations:	fichierTexte:	**Chaine de caracteres** $\rightarrow$ FichierTexte
	ourvir:	FichierTexte $\times$ Mode $\nrightarrow$ FichierTexte
	fermer:	FichierTexte $\nrightarrow$ FichierTexte
	estOuvert:	FichierTexte $\nrightarrow$ **Booleen**
	mode:	FichierTexte $\nrightarrow$ Mode
	finFichier:	FichierTexte $\nrightarrow$ **Booleen**
	ecrireChaine:	FichierTexte $\times$ Chaine $\nrightarrow$ FichierTexte
	lireChaine:	FichierTexte $\nrightarrow$ FichierTexte $\times$ Chaine
	ecrireCaractere:	FichierTexte $\times$ **Caractere** $\nrightarrow$ FichierTexte
	lireCaractere:	FichierTexte $\nrightarrow$ FichierTexte $\times$ **Caractere**
Sémantiques:	fichierTexte:	creation d'un fichier texte à partir d'un fichier identifié par son nom
	ouvrir:	ouvre un fichier texte en lecteur ou ecriture. Si le mode est ecritue et que le fichier existe, alors ce dernier est écrasé
	fermer:	ferme un fichier texte
	lireCaractere:	lit un caractère à partir de la position courante du fichier
	lireChaine:	lit une chaîne (jusqu'à un retour à la ligne ou la fin de fichier) à partir de la position courante du fichier
	ecrireCaractere:	écrit un caractère à partir de la position courante du fichier
	ecrireChaine:	écrit une chaîne suivie d'un retour à la ligne à partir de la position courante du fichier
Préconditions:	ouvrir(f):	non estOuvert(f)
	fermer(f):	estOuvert(f)
	finFichier(f):	mode(f)=lecture
	lireXX(f):	estOuvert(f) et mode(f)=lecture et non finFichier(f)
	ecrireXX(f):	estOuvert(f) et mode(f)=ecriture

L'analyse du cahier des charges nous permet d'identifier les TAD suivants :
- `Mot`
- `Dictionnaire`[2]
- `Correcteur orthographique`

Questions :

1. Présentez ces TAD à l'aide du formalisme vu en cours.
2. Proposez une analyse descendante des deux dernières fonctionnalités demandées.

3.2 Conception

3.2.1 Conception préliminaire

1. Donnez les signatures des fonctions et procédures permettant de manipuler les TAD précédents.
2. Donnez les signatures des fonctions et procédures issues de votre analyse descendante (fonctions et procédures métiers).

3.2.2 Conception détaillée

1. Proposez une implantation des TAD précédents ainsi que le corps de leurs fonctions et procédures les plus complexes.
2. Explicitez le corps des fonctions et procédures métiers.

3.3 Développement, tests unitaires et documentation

Développez le programme en C en testant au maximum vos procédures et fonctions à l'aide de l'API CUnit (la personne qui codera les tests unitaires ne doit pas être la personne qui implante les fonctions C). Vous générerez aussi une documentation de votre code à l'aide du logiciel Doxygen.

2. à ne pas confondre avec le TAD Dictionnaire vu en cours