INF123 - Examen
12 mai 2016

Durée : 2h.
Tout document interdit à l'exception du mémo bash non annoté.
Calculatrices, téléphones interdits.

Le barème est indicatif. Pour chaque question, une partie des points peut tenir compte de la présentation.
Toute réponse même incomplète sera valorisée à partir du moment où elle réalise au moins une partie de ce qui est demandé.
Les questions ne sont pas indépendantes, mais peuvent être traitées dans un ordre quelconque à condition d'avoir pris connaissance de l'ensemble du sujet. Dans tous les cas vous pouvez utiliser les fonctions demandées dans les questions précédentes même si vous n'avez pas réussi à les écrire.
Vous pouvez utiliser les fonctions de la bibliothèque standard (`strcmp`, `strcpy`, ...) vues en TD ou en TP.

Préambule

Dans cet examen, on vous demande de compléter (parties 1, 2, 3 et 5) et d'utiliser (partie 4) un programme C qui vous est partiellement fourni en annexe. Il y est fait référence dans la suite via les numéros de lignes. Il vous est conseillé d'en prendre connaissance, ainsi que de l'ensemble des questions avant de commencer à composer.

Le but du programme est de vérifier la correction syntaxique de phrases simples du français. Les mots des phrases appartiennent à différentes catégories : ce sont des articles, des noms, des adjectifs, des verbes (conjugués à la troisième personne du singulier), et des conjonctions de coordination.

• Partie 1. Les mots de chaque catégorie sont lus dans des fichiers (il y a un fichier pour les articles, un fichier pour les noms ...) et stockés dans des ensembles de mots (il y a un ensemble d'articles, un ensemble de noms ...). Cette lecture est effctuée par la fonction `initialiser` appelée lignes 101 à 105.

• Partie 2. Les phrases sont lues ligne par ligne dans un fichier (une phrase par ligne), puis chaque phrase est lue mot par mot.

• Partie 3. À chaque mot est associée sa *nature* qui correspond à sa catégorie c'est-à-dire à l'ensemble de mots auquel il appartient. Les *natures* des mots sont les entrées d'un automate reconnaisseur qui décide si la phrase est correcte ou non.

• Partie 4. Il s'agit d'écrire un script shell qui détecte les fichiers textes contenant des phrases incorrectes en utilisant le programme précédent.

• Partie 5. On traite le problème d'homonymie (un même mot peut avoir plusieurs *natures* différentes). Pour cela, on calcule l'intersection d'ensembles de mots.

Remarque : des constantes sont définies lignes 5 à 7 pour le nombre maximum de mots de chaque ensemble, la longueur maximum des mots et des lignes, et on supposera dans tout ce qui suit que ces constantes sont suffisamment grandes : **vous n'avez pas à traiter d'éventuels problèmes de débordement.**

Partie 1 : Ensembles de mots (∼ 5 points)

Les ensembles de mots sont mémorisés dans des variables de type `ens_mots` (lignes 16 à 19) :
- le champ `card` est le cardinal de l'ensemble,
- le champ `tab` est une séquence contiguë de chaînes de caractères entre les indices `0` et `card-1` inclus. Chaque chaîne de l'ensemble n'apparait qu'une fois, et la séquence n'est pas triée.

Question 1 :

Dessinez un exemple de valeur de la variable `noms` (déclarée ligne 21) représentant l'ensemble {crayon, chat, papillon, gendarme, potage, bateau}.

Question 2 :

Écrivez les fonctions :
- `init_vide` (lignes 23-25) qui initalise un ensemble vide.
- `est_vide` (lignes 27-29) qui retourne `1` si `e est vide`, `0` sinon.
- `chercher` (lignes 31-33) qui recherche le mot `m` dans l'ensemble `e` et qui retourne
  - l'indice où se trouve `m` s'il est présent dans `e`,
  - `-1` sinon.
- `ajouter` (lignes 35-37) qui ajoute le mot `m` à l'ensemble `e`, s'il n'est pas déjà présent.

Les ensembles de mots sont lus à partir de fichiers textes. Par exemple, la variable `noms` pourra être initialisée à partir du fichier dont le contenu est le suivant :

chat potage gendarme papillon crayon bateau

Question 3 :

Écrivez la fonction `initialiser` (lignes 39-41) qui initialise l'ensemble de mots `m` à partir des mots contenus dans le fichier de nom `nom_fich`. Si l'ouverture du fichier est impossible, un message d'erreur approprié est affiché et le programme se termine.

Partie 2 : Lignes (∼ 5 points)

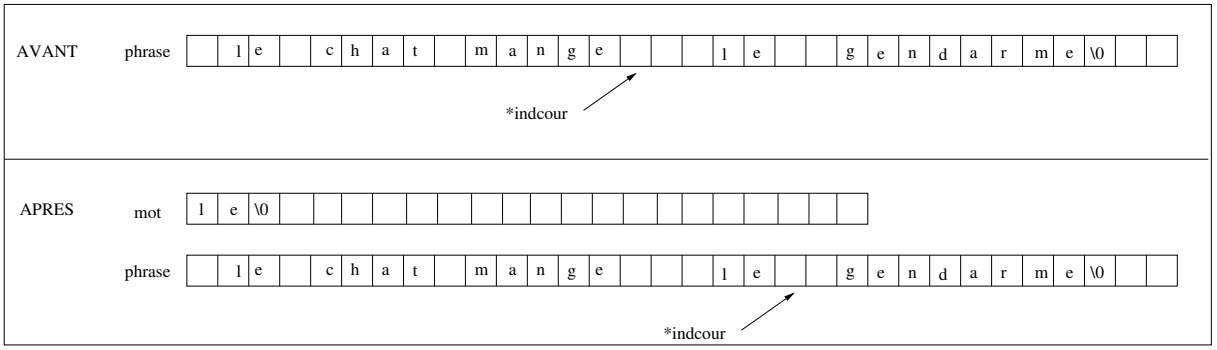
Question 4 :

Écrivez la fonction `lire_ligne` (lignes 43-46) dont la spécification est la même que la fonction `lire_ligne` de l'interpréteur : `f` étant un descripteur de fichier ouvert en lecture, la fonction lit et stocke dans `phrase` la prochaine ligne de ce fichier sous la forme d'une chaîne de caractères, c'est-à-dire en remplaçant le `'\n'` par `'\0'`. Elle renvoie `0` si la fin de fichier est atteinte sans qu'aucun caractère n'ait été lu, `1` sinon.

Nous appelons *phrases* les chaînes de caractères obtenues après lecture d'une ligne par la fonction `lire_ligne`. Une phrase est constituée de séquences de lettres appelées *mots* séparées par un ou plusiers espaces. Il n'y a pas d'autres caractères. On suppose que **les phrases comportent au moins un mot, et il n'y a pas d'espace après le dernier mot.**

La fonction `prochain_mot` (lignes 59-61) est appelée avec comme paramètre une chaîne de caractères `phrase` obtenue par la fonction `lire_ligne`. Elle écrit dans son paramètre `mot` le prochain mot de `phrase` après l'indice `*ind_cour`, et met à jour cet indice en le plaçant après le mot lu.
Plus précisément :
- avant l'appel : `*ind_cour` est l'indice du début d'un mot de `phrase` ou d'une séquence d'espaces précédant un mot,
- après-l'appel :
  - le mot a été stocké dans `mot` sous la forme d'une chaîne de caractères (c'est-à-dire avec un `'\0'` à la fin.
  - `*ind_cour` est l'indice du caractère qui suit le mot,
  - `phrase` n'a pas été modifiée.

Les schémas ci-dessous montrent 2 exemples d'exécution de `prochain_mot` :



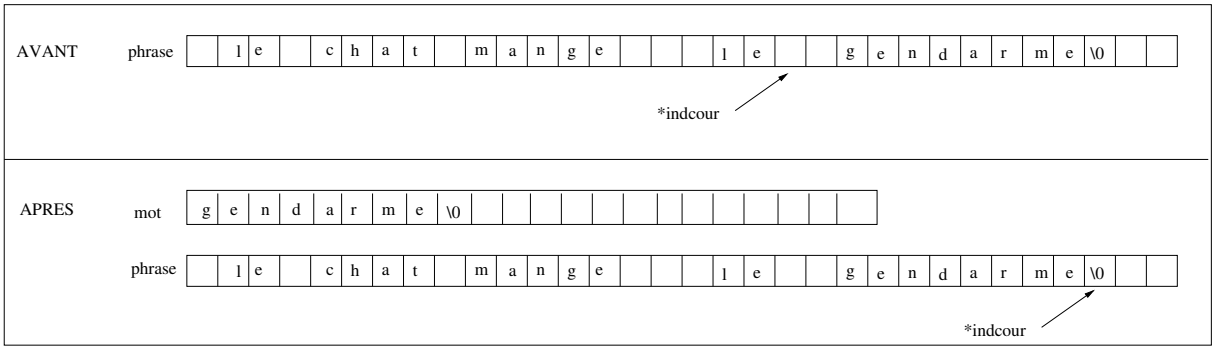


Figure 1 : deux exemples d'exécution de `prochain_mot`

Question 5 :

Écrivez la fonction `prochain_mot`.

Partie 3 : Automate reconnaisseur (∼ 5 points)

La *nature* d'un mot est donnée par l'ensemble de mots (de type `ens_mots`) auquel il appartient.
Considérons par exemple les ensembles de mots suivants et la *nature* correspondante :

Ensemble	Nature
{bateau, chat, potage, crayon, gendarme, papillon}	`NOM`
{mange, regarde, espere, dessine}	`VERBE`
{le, la, du, les}	`ARTICLE`

Figure 2

Si un mot n'appartient à aucun ensemble, il a par convention la nature `INCONNU`. Avec les ensembles ci-dessus, c'est par exemple le cas du mot *lait*.

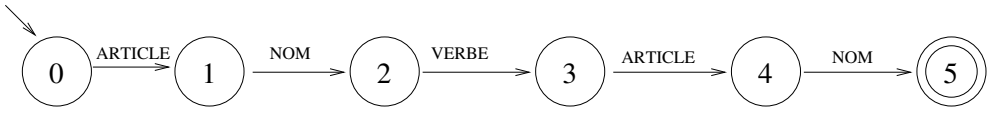


Figure 3

Dans l'automate ci-dessus, les entrées des transitions sont des *natures* de mots. Pour reconnaître si une phrase est correcte :

- on part de l'état initial,
- successivement pour chaque mot, on effectue la transition associée à sa *nature*.
- la phrase est correcte si l'état final est atteint à la fin de la phrase.

Exemple : En ne considérant que les mots des ensembles de la figure 2 :
• Soit la phrase

le chat mange du gendarme

Elle correspond à la séquence de *natures* :

`ARTICLE NOM VERBE ARTICLE NOM`

qui est reconnue par l'automate. La phrase est donc correcte.

• Soit la phrase

chat mange du le gendarme

Elle correspond à la séquence de *natures* :

`NOM VERBE ARTICLE ARTICLE NOM`

qui n'est pas reconnue par l'automate. La phrase n'est donc pas correcte.

• Soit la phrase

le lait mange du chat

Elle correspond à la séquence de *natures* :

```
ARTICLE INCONNU VERBE ARTICLE NOM
```

qui n'est pas reconnue par l'automate. La phrase n'est donc pas correcte.

Remarque : l'automate ne tient pas compte des accords en genre ou en nombre. Par exemple, la phrase *la chat mange les gendarme* est considérée comme correcte.

Question 6 :

Dessinez l'automate obtenu en ajoutant un nouvel état de numéro 6 vers lequel vont toutes les transitions qui ne sont pas dessinées sur l'automate de la figure 3, en particulier les transitions étiquetées par `INCONNU`. **Indication :** l'état 6 est un état puits, c'est-à-dire que toutes les transitions issues de l'état 6 mènent à l'état 6.

On souhaite maintenant reconnaître des phrases pouvant comporter des adjectifs et des conjonctions de coordinations :

- les noms peuvent être précédés ou suivis d'un nombre quelconque d'adjectifs.
- une phrase peut être constituée de propositions reliées par des conjonctions de coordination.

Par exemple, en considérant les ensembles d'adjectifs et de conjonctions suivants (en plus des ensembles de noms, verbes et articles de la figure 2) :

Ensemble	Nature
{petit, grand, beau, fort, intelligent, gros, bleu}	`ADJECTIF`
{mais, ou, et, donc, or, ni, car}	`CONJONCTION`

Figure 4

on pourra maintenant reconnaître comme correctes des phrases telles que :

le petit gros gendarme fort dessine le beau petit chat intelligent donc le grand papillon mange du potage bleu

correspondant à la séquence de *natures* :

```
ARTICLE ADJECTIF ADJECTIF NOM ADJECTIF VERBE ARTICLE ADJECTIF NOM ADJECTIF CONJONCTION
                    ARTICLE ADJECTIF NOM VERBE ARTICLE NOM ADJECTIF
```

Question 7 :

En considérant l'ensemble d'entrées {`ARTICLE, ADJECTIF, NOM, VERBE, CONJONCTION, INCONNU`}, complétez l'automate pour tenir compte des adjectifs et des conjonctions. Veillez à ce que de chaque état soit issue une et une seule transition pour chaque entrée possible.

Remarque : Vous pouvez répondre aux questions suivantes (8 et 9) même si vous n'avez pas su répondre à la question 7 : utilisez alors l'automate de la question 6.

Question 8 :

Représentez la fonction de transition de votre automate par un tableau à deux dimensions dont les lignes sont indexées par les numéros des états (de `0` à `6`), et les colonnes par les entrées.

L'automate est simulé dans la fonction `analyse` (lignes 63-78). L'automate est représenté ainsi :

- les états sont les entiers `0, 1, 2, ...`,
- l'état initial est l'état 0,
- les entrées sont les constantes définies lignes 9 à 14,
- la fonction de transition est le tableau `transition` déclaré et initialisé dans la fonction `analyse` (lignes 65-67) : le premier indice est l'état courant, le second est l'entrée (l'initialisation en C de ce tableau ne vous est pas demandée).

De plus, la fonction `nature_mot` (lignes 48-57) calcule la nature d'un mot.

Question 9 :

La fonction `analyse` renvoie `1` si la chaîne `phrase` en argument est reconnue par l'automate, `0` sinon. À l'aide des fonctions `prochain_mot` et `nature_mot`, complétez cette fonction.

Partie 4 : Shell (∼ 2 points)

Soient **Articles**, **Noms**, **Adjectifs**, **Verbes**, et **Conjonctions** les fichiers contenant respectivement les ensembles d'articles, noms, adjectifs, verbes et conjonctions. Soit **analyser** le nom de l'exécutable obtenu à partir du source donné en annexe et complété dans les parties 1 à 3.

Question 10 :

Écrivez un script shell qui prend en arguments un ou plusieurs noms de fichiers contenant des phrases et qui affiche pour chacun d'eux :

- `les phrases du fichier <nom_du_fichier> sont correctes`
  si toutes les phrases du fichier son correctes
- `le fichier <nom_du_fichier> contient au moins une phrase incorrecte`
  sinon.

De plus, le script affichera un message d'erreur si aucun argument ne lui est fourni, ou si un des arguments n'est pas le nom d'un fichier existant.
Indications : Si toutes les phrases du fichier analysé sont correctes, le programme **analyser** ne produit pas d'affichage. Sinon, une ligne est affichée par l'instruction de la ligne 113. On rappelle (cf memo-bash) que le test `[ -n <chaîne> ]` vaut vrai si la chaîne n'est pas vide.

Partie 5 : Intersection d'ensembles (∼ 3 points)

Considérons les ensembles de noms, d'adjectifs et de verbes suivants :

Ensemble	Nature
{chat, papillon, gendarme, potage, rose, porte}	`NOM`
{petit, grand, gros, beau, fort, intelligent, rose}	`ADJECTIF`
{mange, regarde, espere, dessine, porte}	`VERBE`

Figure 5

Ainsi, il est possible qu'un même mot soit à la fois `NOM` et `ADJECTIF` (comme *rose*) ou à la fois `NOM` et `VERBE` (comme *porte*)

Question 11 :

Écrivez une fonction de profil
`void intersection(ens_mots *A, ens_mots *B, ens_mots *C)` (lignes 80-82)
qui construit dans `C` l'intersection des ensembles `A` et `B`.
Indications

- Utilisez les fonctions sur les ensembles de mots de la partie 1.
- Les ensembles ne sont pas triés. Un algorithme possible est le suivant : pour chaque élément de `A`, on cherche s'il est dans `B`, si oui, on l'ajoute à `C`.

Question 12 :

Complétez la fonction `main` en signalant par un message si les ensembles des noms et des adjectifs, ou si les ensembles des noms et des verbes, ont une intersection non vide (on ne teste pas d'autres intersections d'ensembles). Ne recopiez pas toute la fonction `main`, indiquez les modifications apportées à l'aide des numéros de lignes.

Bonus : intersection de séquences triées (∼ 2 points)

Par abus de langage, on dira qu'un ensemble représenté par une variable `E` de type `ens_mots` est trié si le champ `tab` de `E` est un tableau trié. On suppose ici que les ensembles de mots sont triés selon l'ordre lexicographique. On souhaite calculer l'intersection de deux ensembles triés, et on veut que le résultat soit trié.

Question 13 :

a) La fonction `intersection` de la question 11 répond-elle au problème (c'est-à-dire, si `A` et `B` sont triés, est-ce que `C` l'est) ? Justifiez votre réponse

b) En tirant parti du fait que `A` et `B` sont triés, écrivez une nouvelle fonction C `intersection_triee` (lignes 84-86) plus efficace.

On rappelle que la comparaison selon l'ordre lexicographique des chaînes de caractères s'effectue avec la fonction `int strcmp(char ch1[], char ch2[])` dont le résultat est négatif, nul ou positif selon que `ch1` est avant `ch2`, égal à `ch2`, ou après `ch2` dans l'ordre lexicographique.

Annexe

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NB_MAX_MOTS 1024
#define LG_MAX_MOTS 64
#define LG_MAX_LIGNE 1024

#define ARTICLE 0
#define ADJECTIF 1
#define NOM 2
#define VERBE 3
#define CONJONCTION 4
#define INCONNU 5

typedef struct{
        char tab[NB_MAX_MOTS][LG_MAX_MOTS] ;
        int card ;
        } ens_mots ;

ens_mots articles, adjectifs, noms, verbes, conjonctions ;

void init_vide(ens_mots *e) {
/* A COMPLETER - Q2 */
}

int est_vide(ens_mots *e) {
/* A COMPLETER - Q2 */
}

int chercher(ens_mots *e, char m[]) {
/* A COMPLETER - Q2 */
}

void ajouter(ens_mots *e, char m[]) {
/* A COMPLETER - Q2 */
}

void initialiser(ens_mots *e, char nom_fich[]) {
/* A COMPLETER - Q3 */
}

/* meme spec que dans l'interpreteur */
int lire_ligne(FILE *f, char phrase[]) {
/* A COMPLETER - Q4 */
}

int nature_mot(char mot[]) {
        int nature ;
        if (chercher(&articles, mot)!=-1) nature = ARTICLE ;
        else if (chercher(&adjectifs, mot)!=-1) nature = ADJECTIF ;
        else if (chercher(&noms, mot)!=-1) nature = NOM ;
        else if (chercher(&verbes, mot)!=-1) nature = VERBE ;
        else if (chercher(&conjonctions, mot)!=-1) nature = CONJONCTION ;
        else nature=INCONNU ;
        return nature ;
}

void prochain_mot(char phrase[], int *ind_cour, char mot[]) {
/* A COMPLETER - Q5 */
}

int analyse(char phrase[]) {
        /* initialisation du tableau transition (non demandee) */
        int transition[...][...] ={
                ...
                } ;
        int i=0 ;
        char mot[LG_MAX_MOTS] ;
        int etat_cour=0 ;
        int etat_suiv ;
        int entree ;
```

```
73
74          while (phrase[i]!='\0'){
75                  /* A COMPLETER - Q9 */
76                  }
77          return (etat_cour==/*A COMPLETER - Q9 */) ;
78  }
79
80  void intersection(ens_mots *A, ens_mots *B, ens_mots *C) {
81  /* A COMPLETER - Q11 */
82  }
83
84  void intersection_triee(ens_mots *A, ens_mots *B, ens_mots *C) {
85  /* A COMPLETER - Q13 */
86  }
87
88  int main(int argc, char *argv[]) {
89          FILE *f ;
90          char phrase[LG_MAX_LIGNE] ;
91          int ok_ligne=1 ;
92          int ok_syntaxe ;
93
94          if (argc !=7) {
95                  fprintf(stderr, "syntaxe %s fichier_articles fichier_noms
96                          fichier_adjectifs fichier_verbes fichier_conjonctions
97                          fichier_phrases\n", argv[0]) ;
98                  exit(1) ;
99                  }
100
101         initialiser(&articles, argv[1]) ;
102         initialiser(&noms, argv[2]) ;
103         initialiser(&adjectifs, argv[3]) ;
104         initialiser(&verbes, argv[4]) ;
105         initialiser(&conjonctions, argv[5]) ;
106
107         f=fopen(argv[6], "r") ;
108         while(ok_ligne) {
109                 ok_ligne=lire_ligne(f, phrase) ;
110                 if (ok_ligne) {
111                         ok_syntaxe=analyse(phrase) ;
112                         if (!ok_syntaxe) {
113                                 printf("la phrase [ %s ] n'est pas correcte\n", phrase) ;
114                         }
115                 }
116
117         }
118         return 0 ;
119 }
```