INF203 - Examen terminal
Année 2025-2026

Durée : 2h.
Une feuille A4 manuscrite recto/verso autorisée.
Tout autre document, calculatrices et appareils électroniques interdits.
Pour chaque question, une partie des points peut tenir compte de la présentation.
Le barème est indicatif.

Toute réponse même incomplète sera valorisée à partir du moment où elle réalise au moins une partie de ce qui est demandé. Les questions sont relativement indépendantes, et dans tous les cas vous pouvez utiliser les scripts et les fonctions demandées dans les questions précédentes même si vous n'avez pas réussi à les écrire.

Par ailleurs, tout code ou algorithme *élégant*, en plus d'être correct, sera bonifié.

1 Copie d'un fichier ligne par ligne (6 points)

Question 1. (2 points) Écrivez en C une fonction

`void lire_ligne(FILE *f, char *ligne)`

avec les deux paramètres :

- `f` est un descripteur de fichier ouvert en lecture,
- `ligne` est une une chaîne de caractères,

dont le but est de lire une ligne complète de `f` et de la recopier dans `ligne` en remplaçant le caractère de fin de ligne par le caractère de fin de chaîne.

Solution:

```
void lire_ligne(FILE *f, char *ligne) {
  char c;
  int i=0;

  fscanf(f, "%c", &c);
  while (!feof(f) && c != '\n') {
    ligne[i] = c;
    i++;
    fscanf(f, "%c", &c);
  }

  ligne[i] = '\0';
}
```

Barème

- ½ : boucle de lecture fscanf/feof
- ½ : détection du `\n`
- ½ : copie dans `ligne`
- ½ : écriture du `\0`

Question 2. (3 points) À l'aide de la fonction `lire_ligne`, écrivez une fonction C

`void affiche_fichier(char * nom_fich)`

dont le but est d'afficher le contenu du fichier de nom `nom_fich` à l'écran, ligne par ligne. On suppose que les lignes du fichier comportent au plus `128` caractères.

Vous vérifierez que le fichier s'est ouvert correctement, et sinon vous afficherez un message d'erreur.

Solution:

```
void affiche_fichier(char *nom_fich) {
  char buffer[128];
  FILE *entree = fopen(nom_fich, "r");

  if (entree == NULL) {
    fprintf(stderr, "Fichier %s inaccessible\n", nom_fich);
  }

  while(!feof(entree)) {
    lire_ligne(entree, buffer);
      printf("%s\n", buffer);
  }
}
```

Barème

- — ½ : déclarations des variables (pas de char * !)
- — ½ : fopen
- — ½ : test à NULL
- — ½ : boucle de lecture (pas trop d'exigence, il faut juste penser à boucler)
- — ½ : appel à `lire_ligne`
- — ½ : écriture de la ligne

Question 3. (1 point) À l'aide de la fonction `affiche_fichier`, écrivez la fonction `main` d'un programme C qui prend un nom de fichier en argument, et affiche son contenu à l'écran.

Vous vérifierez que votre programme a reçu le bon nombre d'arguments, et sinon vous afficherez un message d'erreur.

Solution:

```
int main(int argc, char *argv[]) {
  if (argc != 2) {
    fprintf(stderr, "Usage : %s fichier\n", argv[0]);
  }

  affiche_fichier(argv[1]);

  return 0;
}
```

Barème

- — ½ : test sur `argc`
- — ½ : présence dans l'en-tête et utilisation correcte de `argv`

2 Traitement des lignes `#include` (6 points)

Dans cette partie, les fichiers traités sont des fichiers sources de programmes C. Ces fichiers peuvent comporter des lignes de la forme

`#include <librairie_standard.h>`

ou

`#include "fichier_utilisateur.h"`

Il s'agira, comme dans la partie 1, d'afficher le contenu du fichier à l'écran ligne par ligne, sauf pour les lignes commençant par `"#include"` : dans ce cas, à la place de la ligne, on affichera le contenu du fichier suffixé par `.h` auquel la ligne fait référence (`librairie_standard.h` ou `fichier_utilisateur.h`).

Les fichiers d'entête des librairies standards (ceux dont le nom est entre < et >) se trouvent dans le répertoire `/usr/include`, les autres dans le répertoire courant.

Question 4. (2,5 points) Écrivez une fonction

`void extraire(char ligne[], char nom_header[], char *type)`

dont la spécification est la suivante :

- la chaîne `ligne` reçue en argument commence par `"#include"` .
- la valeur des paramètres `nom_header` et `type` sera modifiée par la fonction,
- la fonction copie le nom du fichier `.h` dans la chaîne `nom_header`,
- la fonction écrit le caractère `'"'` ou `'<'` selon les cas dans `*type`.

Exemples :

ligne	nom_header	*type
`#include <stdio.h>`	`stdio.h`	<
`#include "prog.h"`	`prog.h`	"

Pour mieux comprendre l'usage de la fonction `extraire`, vous pouvez regarder à la question suivante comment elle sera utilisée.

Solution:

```
void extraire(char ligne[], char nom_header[], char *type) {
  int i=10, j=0;
  *type = ligne[9];
  while (ligne[i] != '"' && ligne[i] != '>') {
    nom_header[j] = ligne[i];
    i++;
    j++;
  }
  nom_header[j] = '\0';
}
```

Barème

- ½ : récupération du type " ou <
- ½ : copie de `ligne` dans `nom_header`
- ½ : gestion du décalage d'indices entre les deux chaînes
- ½ : arrêt après le nom du fichier d'en-tête
- ½ : écriture du marqueur de fin de chaîne

Question 5. (2 points) Complétez la fonction `inclure_fichier` suivante. Cette fonction doit afficher à l'écran le contenu du fichier `.h` à inclure en utilisant la fonction `affiche_fichier` décrite à la question 2.

```
void inclure_fichier(char ligne[]) {
  /* ligne est une chaine de caractères qui commence par #include */
  char nom_fich[128] ;
  char nom_complet[128] ;
  char type ;
  extraire(ligne, nom_fich, &type) ;
  /*** A COMPLETER ***/
}
```

Solution:

```
void inclure_fichier(char ligne[]) {
  char nom_fich[128] ;
  char nom_complet[128] ;
  char type ;
  extraire(ligne, nom_fich, &type) ;
```

```
  if (type == '<') {
    strcpy(nom_complet, "/usr/include/");
  } else {
    strcpy(nom_complet, "./");
  }

  strcat(nom_complet, nom_fich);
  affiche_fichier(nom_complet);
}
```

Barème

- — ½ : test sur le type
- — ½ : utilisation de `strcpy` ou équivalent
- — ½ : utilisation de `strcat` ou équivalent
- — ½ : appel correct à `affiche_fichier`

Question 6. (1,5 points) Modifiez la fonction `affiche_fichier` écrite dans la question 2 afin d'écrire le contenu d'un fichier à l'écran en remplaçant les lignes qui commencent par `"#include"` par les contenus des fichiers à inclure.

Indication : pour savoir si une ligne commence par `"#include"`*, utilisez la fonction* `strncmp` *décrite en Annexe.*

Solution:

```
void affiche_fichier(char *nom_fich) {
  char buffer[128];
  FILE *entree = fopen(nom_fich, "r");

  if (entree == NULL) {
    fprintf(stderr, "Fichier %s inaccessible\n", nom_fich);
  }

  while(!feof(entree)) {
    lire_ligne(entree, buffer);
    if (strncmp(buffer, "#include", 8) == 0) {
      inclure_fichier(buffer);
    } else {
      printf("%s\n", buffer);
    }
  }
}
```

Barème

- — ½ : utilisation de `strncmp` ou équivalent
- — ½ : if/else
- — ½ : appel à `inclure_fichier`

3 Programmation bash : Comparaison du comportement de deux programmes (3 points)

On souhaite vérifier que deux programmes ***prog1*** et ***prog2*** ont le même comportement. Plus exactement, chacun de ces programmes prend un nom de fichier en argument, et produit un affichage à l'écran. Il s'agit donc de comparer les sorties produites par ces deux programmes pour un ensemble de fichiers de test.

Question 1. (1 point) Écrivez une commande ou une suite de commandes permettant de savoir si les sorties produites par ***prog1*** et ***prog2*** avec le fichier ***fich.test*** sont les mêmes (il ne s'agit pas de *lire* ces sorties sur l'écran).

Solution:

```
prog1 fich.test > sortie1.test
prog2 fich.test > sortie2.test
diff sortie1.test sortie2.test
```

Barème

- — ½ : utilisation de redirection
- — ½ : utilisation de diff

Question 2. (2 points) Écrivez un script shell qui compare les sorties produites par ***prog1*** et ***prog2*** sur un ensemble de fichiers de tests donnés en argument à ce script. Pour chaque test, un affichage sera produit : `OK` si les sorties sont les mêmes, `pas OK` dans le cas contraire.

Solution:

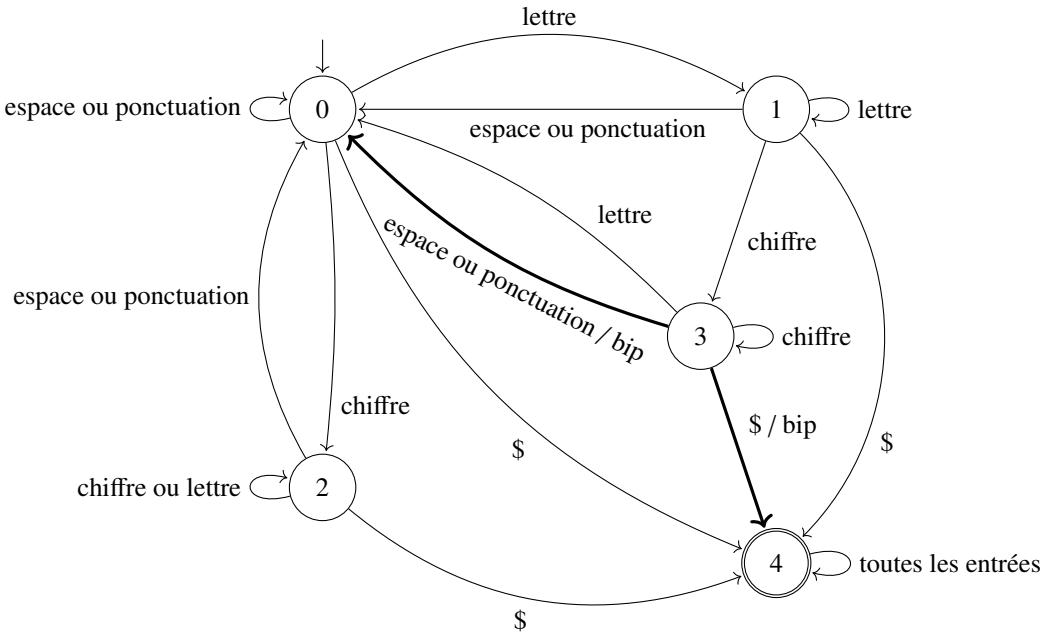
```
#!/bin/bash

for ENTREE in $*
do
  prog1 $ENTREE > sortie1.test
  prog2 $ENTREE > sortie2.test
  diff sortie1.test sortie2.test > /dev/null
  if [ $? -eq 0 ]
  then
    echo "test $ENTREE : OK"
  else
    echo "test $ENTREE : pas OK"
  fi
done
```

Barème

- — ½ : boucle for
- — ½ : utilisation de `$*`
- — ½ : utilisation de `$?` ou directement de diff comme test
- — ½ : if/then/else

4 Automate (5 points)



Les symboles d'entrée de cet automate sont des caractères qui peuvent être soit des lettres (majuscules ou minuscules), soit des chiffres décimaux, soit des espaces, soit des caractères de ponctuation, soit le caractère `$`. Pour certaines de ses transitions (indiquées en gras sur la figure), il émet un signal de sortie noté `bip`.

Question 1. (1 point) Donnez pour cet automate deux séquences d'entrée composées de 5 caractères, l'une résultant en l'émission du signal de sortie `bip`, l'autre non. Dans les deux cas, le dernier caractère mènera à l'état final (`4`).

Solution:

Barème

— ½par séquence répopndant à la question

On initialise une variable entière `n` à `0`, et on lance la simulation de l'automate avec la séquence d'entrée suivante :

```
Mercredi 6 mai 2026, 09h00min ... onzeheure0min : session1 de INF203 !!! $
```

À chaque `bip`, la variable `n` est incrémentée de `1`.

Question 2. (1 point) Lorsqu'on arrive dans l'état final (`4`)

— que vaut `n` (pour ce texte d'entrée spécifiquement) ?

— de manière générale, qu'énumère `n` pour tout texte d'entrée ?

Solution:

Barème

— ½ : n=2

— ½ : le nombre de séquences de la forme « une ou plusieurs lettres, puis un ou plusieurs chiffres, puis une espace ou ponctuation »

Question 3. (3 points) En utilisant les fonctions et déclarations fournies, complétez le programme de l'annexe C (écrivez `transition`, `sortie` et `main` sur votre copie) pour simuler l'automate à partir d'un séquence d'entrée lue au clavier.

Indication. Remarquez que certaines sorties et transitions peuvent être mises en commun : on gagne beaucoup à ne pas énumérer les cas de manière exhaustive.

Solution:

Barème

- 1 : codage des transitions
- 1 : codage des sorties
- 1 : boucle principale

Dans chaque cas, on pourra prévoir ½ point pour le fait que l'automate est correctement traduit, et ½ pour la qualité du code (tableau correctement initialisé, switch ou if bien construit, etc.)

Annexe A : Extraits du manuel en ligne de `strcmp` et de `strcat`

```
NAME
       strcmp, strncmp - Comparaison de deux chaînes

SYNOPSIS
       #include <string.h>

       int strcmp(const char *s1, const char *s2);

       int strncmp(const char *s1, const char *s2, size_t n);

DESCRIPTION
       La fonction strcmp() compare les deux chaînes s1 et s2. Elle renvoie un
       entier négatif, nul, ou positif, si s1 est respectivement inférieure,
       égale ou supérieure à s2.

       La fonction strncmp() est identique sauf qu'elle ne compare que
       les n (au plus) premiers caractères de s1 et s2.

.....................................................................................

NAME
       strcat - concaténer deux chaînes

SYNOPSIS
       #include <string.h>

       char *strcat(char *dest, const char *src);

DESCRIPTION
       La fonction strcat() ajoute la chaîne src à la fin de la chaîne dest
       en écrasant le caractère nul ('\0') à la fin de dest, puis en ajoutant
       un nouveau caractère nul final. Les chaînes ne doivent pas se chevaucher,
       et la chaîne dest doit être assez grande pour accueillir le résultat.

.....................................................................................
```

Annexe B : Extrait du memo bash

Variables spéciales définies par le shell

`$?`	code de retour de la dernière commande (0 si ok)
`$0`	nom du script
`$1` à `$9`	les éventuels 9 premiers arguments passés au script
`$#`	nombre d'arguments
`$*`	liste des arguments (à partir de `$1`)

Annexe C : squelette de programme pour la question 4.3

```
#define LETTRE 0
#define CHIFFRE 1
#define PONCTUATION 2
#define ESPACE 3
#define DOLLAR 4

int lire_nature_entree() {
        char c ;
        int nature ;
        scanf("%c", &c) ;
        if (isalpha(c)) nature = LETTRE ;
        else if (isspace(c)) nature = ESPACE ;
        else if (isdigit(c)) nature = CHIFFRE ;
        else if (c=='$') nature = DOLLAR ;
        else nature = PONCTUATION ;
        return nature ;
}

// Transition :
// ----------
// A COMPLETER
// - soit sous la forme d'un tableau de type
//      int transition[5][5] = { ... };
// - soit sous la forme d'une fonction de forme
//      int transition(int etat, int nature_entree){ ... };

// Sortie :
// ------
int sortie(int etat, int nature_entree) {
        int bip =0 ;
        //
        // A COMPLETER
        //
        return bip ;
}

// Main :
// ----
int main() {
        int n ;
        int etat_courant, etat_suivant ;
        int nature_entree ;

        n = 0 ;
        etat_courant = 0 ;
        while (etat_courant != 4) {
                //
                // A COMPLETER
                //
        }
        printf("Nombre de bips = %d\n", n) ;
        return 0 ;
}
```