DLST

UNIVERSITÉ Grenoble Alpes

INF203 - Examen terminal
Année 2023-2024

Durée : 2h.
Une feuille A4 manuscrite recto/verso autorisée, dictionnaire papier (non annoté) autorisé pour les étudiants étrangers uniquement.
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 Programmer la commande tr (C) (10 pts)

Dans cette partie nous avons l'objectif d'écrire un programme dont le comportement sera similaire à celui de la commande `tr`.

On rappelle que cette commande copie l'entrée standard sur la sortie standard en appliquant deux comportements distincts :

Mode "translate" : Si on l'appelle sous la forme `tr CHAINE1 CHAINE2`, chaque caractère présent dans `CHAINE1` est remplacé par le caractère de position correspondante dans `CHAINE2`

Mode "squeeze" : Si on l'appelle sous la forme `tr -s CHAINE`, les répétitions des caractères présents dans `CHAINE` sont remplacés par une occurrence unique

Par exemple, si le fichier `toto` contient les lignes

```
Belle journee pour passer des partielles
Il est vraiment cooooooool cet examen non ?
Je me suis endooooooormi sur la touche o de mon clavier
```

Alors la commande `tr abcd efgh < toto` affiche

```
Belle journee pour pesser hes pertielles
Il est vreiment gooooooool get exemen non ?
Je me suis enhooooooormi sur le toughe o he mon glevier
```

et la commande `tr -s ol < toto` affiche

```
Bele journee pour passer des partieles
Il est vraiment col cet examen non ?
Je me suis endormi sur la touche o de mon clavier
```

Important. Dans toute cette partie, l'utilisation des fonctions de la bibliothèque `string.h` est interdite : c'est à vous d'écrire vos propres fonctions de parcours des chaînes de caractères (on n'utilisera notamment pas les commandes `strlen`, `strcpy`, `strcmp`, `strchr`).

Fonctions utiles

On programme quelques fonctions qui nous permettront par la suite d'implémenter les deux modes de fonctionnement de `tr`.

1. (1½ points) Écrire une fonction `int verif_longueurs(char *chaine1, char *chaine2)` qui renvoie 1 si `chaine1` et `chaine2` sont de la même longueur, ou 0 sinon.

Solution:

```
int verif_longueurs(char *chaine1, char *chaine2){
  int i=0;
  while (chaine1[i] != '\0' && chaine2[i] != '\0')
    i++;

  return (chaine1[i] == chaine2[i]);
}
```

Barème

- ½ : parcours d'une chaîne jusqu'à `'\0'`
- ½ : parcours simultané des deux chaînes OU calcul explicite de deux longueurs
- ½ : valeur renvoyée correcte

2. (2 points) Écrire une fonction `int verif_unique(char *chaine1)` qui renvoie 1 si la chaîne `chaine1` ne contient que des caractères tous distincts, ou 0 sinon.

Solution:

```
int verif_unique(char *chaine1){
  int i=0;
  while (chaine1[i] != '\0') {
    int j=i+1;
    while (chaine1[j] != '\0') {
      if (chaine1[j] == chaine1[i])
        return 0;
      j++;
    }
    i++;
  }
  return 1;
}
```

Barème

- ½ : utilisation de deux indices
- ½ : boucles imbriquées de parcours de la chaîne
- ½ : détection correcte des doublons
- ½ : valeur 1 renvoyée en l'absence de doublons

3. (1 point) Écrire une fonction `int position(char c, char *chaine)` qui renvoie :
   - l'indice du caractère `c` dans `chaine` s'il est présent dans cette chaîne ;
   - -1 sinon.

Solution:

```
int position(char c, char *chaine){
  int i=0;
  while (chaine[i] != '\0') {
    if (chaine[i] == c)
      return i;
    i++;
  }
  return -1;
}
```

Barème

- ½ : arrêt du while en fin de chaîne et renvoi de -1
- ½ : arrêt du while sur `c` et renvoi de l'indice

Programmation du fonctionnement de la commande

À l'aide des fonctions précédemment écrites, on réalise le premier mode de fonctionnement de `tr` qui remplace certains caractères par d'autres. Vous pouvez utiliser ces fonctions même si vous n'avez pas réussi à les écrire.

4. (2 points) Écrire une fonction `void translate(char *chaine1, char *chaine2)` qui applique le traitement translate sur chaque caractère de l'entrée standard et l'affiche sur la sortie standard.
   Vous utiliserez `feof(stdin)` pour décider quand cette fonction doit s'arrêter.

Solution:

```
void translate(char *chaine1, char *chaine2){
  char c;
  scanf("%c", &c);
  while (!feof(stdin)){
    int i = position(c, chaine1);
    if (i == -1)
      printf("%c", c);
    else
      printf("%c", chaine2[i]);
    scanf("%c", &c);
  }
}
```

Barème

- ½ : utilisation de `!feof(stdin)` dans le while
- ½ : lecture anticipée du prochain caractère par `scanf`
- ½ : appel à `position` et distinction des cas où `c` doit être remplacé ou non
- ½ : utilisation de l'indice renvoyé par `position`

À l'aide des fonctions précédemment écrites, on réalise le second mode de fonctionnement de `tr`, qui remplace plusieurs caractères successifs identiques par une seule occurrence.

5. (1½ points) Écrire une fonction `void squeeze(char *chaine)` qui parcourt l'entrée standard et recopie chaque caractère sur la sortie standard, **sauf si** ce caractère apparaît dans `chaine` et est identique au caractère précédent.
   Il est conseillé d'utiliser une variable pour mémoriser l'avant-dernier caractère lu.
   Vous utiliserez `feof(stdin)` pour décider quand cette fonction doit s'arrêter.

Solution:

```
void squeeze(char *chaine){
  char c, prev;
  scanf("%c", &c);
  prev = c+1;
  while (!feof(stdin)){
    if (c != prev || position(c, chaine) == -1)
      printf("%c", c);
    prev = c;
    scanf("%c", &c);
  }
}
```

Barème

- ½ : initialisation de la variable qui mémorise le caractère précédent (ou gestion à part du premier caractère)
- ½ : test correct pour décider s'il faut afficher le caractère ou non (attention aux négations !)
- ½ : mise à jour de la variable qui mémorise le caractère précédent

On intègre désormais les deux modes de fonctionnement de `tr` dans le programme principal.

6. (2 points) Écrire une fonction main qui :
   - vérifie qu'exactement 2 arguments ont été donnés au programme ;
   - si le premier argument est `-s`, applique le traitement "squeeze" à son entrée standard ;
   - sinon, vérifie que les 2 chaînes reçues en arguments ont la même longueur et que la première chaîne ne comporte que des caractères distincts, et applique le traitement "translate".

   Un message d'erreur sera affiché si les arguments sont inadaptés ou en nombre incorrect.

Solution:

```
int main(int argc, char *argv[]) {
  if (argc != 3) {
    fprintf(stderr, "Usage: %s -s SET\nor     %s SET1 SET2\n", argv[0], argv[0]);
    return 1;
  }

  if (argv[1][0] == '-' && argv[1][1] == 's' && argv[1][2] == '\0') {
    squeeze(argv[2]);
  }
  else if (verif_longueurs(argv[1], argv[2]) && verif_unique(argv[1])) {
    translate(argv[1], argv[2]);
  }
  else {
    fprintf(stderr, "Bad format for SET1 or SET2\n");
    return 2;
  }
  return 0;
}
```

Barème

- ½ : test sur argc
- ½ : test sur l'option `-s` et appel à `squeeze`
- ½ : tests à l'aide de `verif_longueurs` et `verif_unique`, et appel à `translate`
- ½ : structure générale du `main` pour distinguer correctement les différents cas

2 Acronymes (automate, C, Bash) (12 pts)

L'objectif de cet exercice est de créer un programme `acronyme.c` en langage C qui remplace toutes les occurrences de l'acronyme (à 2 lettres exactement) de vos prénom-nom (par exemple toutes les occurrences de " RC ", **espaces inclus**, par " Romain Couillet ") au sein d'un fichier `input` donné en argument et d'en donner le résultat dans un second fichier `output` également donné en argument. Pour cela, dans le même esprit que vu en cours, nous ferons usage d'un automate.

Questions :

1. (3 points) Chaque entrée de l'automate correspondant à la lecture d'un unique caractère au sein du fichier `input`, réalisez graphiquement l'automate d'évolution de la lecture-écriture effectuée par `acronyme.c`.

   Prenez particulièrement soin

(i) de nommer les états de manière claire et distincte,
(ii) d'expliciter sur les liens entre états tout à la fois les entrées (les caractères lus dans `input`) et les sorties (les caractères ou chaînes de caractères écrites dans `output`).

Solution:

2. (2 points) Étant donné cet automate, donnez les tableaux
   (i) des transitions entre états,
   (ii) des sorties (associées à chaque lien état-entrée).

Solution:

3. (2 points) En reprenant le squelette suivant vu en cours (ou une version modifiée si vous le souhaitez)

```
#include <stdio.h>
#include <stdlib.h>
#define ... 1  // ici on pourra donner des raccourcis de noms
#define ... 2  // pour les états
#define ... 3
...

int transition(int etat_courant, char entree) {
    switch (etat_courant) {
        case ... :
            switch (entree) {
                case '...' : return ... ;
                case '...' : return ... ;
                ...
}

void f_sortie (int etat_courant, char entree, char sortie[]) {
    switch (etat_courant) {
        case ... :
            switch (entree) {
                case '...' :
                    sortie[0]= ... ;
                    sortie[1]= ... ;
                    ...
                    break ;
                case '...' :
                    ...
}
```

réalisez les deux fonctions de transition (`transition`) et de sortie (`f_sortie`) associées à l'automate.

Solution:

4. (2 points) Réalisez la fonction `main` du code `acronyme.c` qui devra successivement :
   (i) ouvrir les deux flux `FILE *input, FILE *output` des fichiers `input` et `output` donnés en argument, le premier en lecture, le second en écriture,
   (ii) lire l'ensemble des caractères contenus dans `input` et les traduire, grâce à l'automate, dans le fichier `output`,
   (iii) fermer les deux flux avant de sortir du programme.

Solution:

5. (1 point) Nous souhaitons désormais utiliser de manière étendue la fonction de traduction d'acronyme sur l'ensemble des fichiers texte d'un répertoire local contenant nos documents administratifs. Donnez tout d'abord la commande Bash qui permet de compiler le fichier `acronyme.c` en un exécutable `acronyme`, ainsi que la commande qui ne permet qu'à moi-même et à personne d'autre (même au sein de mes groupes d'utilisateurs) d'exécuter `acronyme`.

Solution:

6. (2 points) Proposez un script Bash `script_acronyme.sh` permettant d'appliquer la commande `acronyme` à l'ensemble des fichiers d'extension `.txt` (de type `fichier.txt`) d'un répertoire passé en argument et qui renommera ces fichiers sous le format étendu `_acronyme.txt` (typiquement `fichier_acronyme.txt`). Par exemple, si le dossier passé en argument est `./DocsAdmin` de contenu initial

```
DocsAdmin/
|-- signature.png
|-- important_et_serieux1.txt
|-- important_et_serieux2.txt
|-- important_et_serieux3.txt
```

suite à l'application de `script_acronyme.sh ./DocsAdmin`, son contenu final devient

```
DocsAdmin/
|-- signature.png
|-- important_et_serieux1_acronyme.txt
|-- important_et_serieux2_acronyme.txt
|-- important_et_serieux3_acronyme.txt
```

Solution: