

## TP2: Arguments en ligne de commande et E/S

### Module ArcSys

#### Objectifs pédagogiques:

- Lecture / écriture de fichier en C (E2, E3, E4)
- Paramètres en ligne de commande (E1, E2, E3)
- Syntaxe de base du C (E1, E2, E3, E4)
- Syntaxe du switch (E2)
- Curiosité et culture informatique (E2, E3, E4)

#### ★ Exercice 1: Échauffement

▷ **Question 1:** Écrivez un programme C acceptant un nombre variable d'arguments sur la ligne de commande et affiche pour chacun d'eux le nombre de caractères correspondant. Ainsi, si ce programme est compilé sous le nom `arglen`, l'exécution de la commande

```
arglen 0 bonjour 2.56 adieu
```

 produira la sortie :

```
l'argument no 0 contient 6 caractere(s)
l'argument no 1 contient 1 caractere(s)
l'argument no 2 contient 7 caractere(s)
l'argument no 3 contient 4 caractere(s)
l'argument no 4 contient 5 caractere(s)
```

*Indication:* On peut retrouver la longueur d'une chaîne de caractères avec la fonction `strlen()`, utilisable après avoir inclu le fichier d'entête `<string.h>`.

#### ★ Exercice 2: Leet Speak

▷ **Question 1:** Écrivez un programme affichant le contenu d'un fichier *source* (que vous demanderez à l'utilisateur) à l'écran. Vous utiliserez les fonctions `fscanf` et `printf` avec la chaîne de formatage `"%c"`.

▷ **Question 2:** Modifiez votre programme pour qu'il prenne son argument (*source*) depuis la ligne de commande grâce à `argv`. Si aucun argument n'est fourni en ligne de commande, il faut encore demander interactivement à l'utilisateur le nom du fichier concerné.

▷ **Question 3:** Modifiez votre programme pour qu'il change les majuscules en minuscules lors de l'affichage, sans changer les autres caractères. Un `man ascii` pourra être utile.

▷ **Question 4:** Modifiez votre programme pour qu'il change maintenant les 't' en '7', les 's' en '5', les 'i' en '1' et les autres changements indiqués sur la page wikipédia «Leet Speak».

#### ★ Exercice 3: Le radoteur

Le radoteur est un algorithme pour fabriquer des mots nouveaux à partir d'une liste de mots. Bien que très simple, il produit souvent des nouveaux mots qui *auraient pu* être dans la liste de départ. Lorsqu'on lui donne une liste de pays, il en fabrique de nouveaux : palombie syldavie bordurie kafiristan libzékistan. . .

Cet algorithme a été imaginé par Claude Shannon puis exploré et baptisé par Roland Moreno (chap. 6 de *Théorie du bordel ambiant*, 1990).

▷ **Question 1:** Implémentez cet algorithme, dont le pseudo-code est le suivant:

1. Choisir une lettre au hasard dans la liste de mot et l'écrire dans le mot en construction.
2. Chercher la prochaine occurrence de cette lettre dans la liste de mot.
3. Considérons la lettre juste après celle cherchée. On l'ajoute au mot en construction, et cela devient le nouveau motif à chercher en retournant à l'étape 2.

Par exemple, si l'on commence depuis le 'b' de "albanie", cela produit "bitie aze boswetourie".

albanie algérie allemagne angola antigua arabie saoudite argentine arménie australie autriche azerbaïdjan bahamas bahreïn bangladesh belgique bélize bénin bhoutan biélorussie birmanie bolivie bosnie botswana brésil bulgarie burkina burundi cambodge cameroun canada chili chine chypre colombie congo corée croatie cuba danemark djibouti égypte équateur érythrée espagne estonie états unis éthiopie fidji finlande france gabon gambie géorgie ghana grèce guatemala guinée guyana haïti hollande honduras hongrie inde indonésie iran iraq irlande islande israel italie jamaïque japon jordanie kazakstan kenya kirghizistan koweït laos lésotho lettonie liban libéria lybie liechtenstein lituanie luxembourg macédoine madagascar malaisie malawi maldives mali malte maroc maurice mauritanie . . .

- ▷ **Question 2:** Testez votre implémentation avec des listes de nom de fleurs, de médicaments ou de plantes, que vous constituerez depuis wikipedia ou ailleurs. Mélangez les champs lexicaux.
- ▷ **Question 3: (optionnelle)** Mélangez des proverbes (*Après la pluie, rien d'impossible ; L'union vient en mangeant ; veni, vedi, j'y reste*), des petites annonces (*Etudiant ch. dame sensible avec remorque*) ou des partitions de musique.
- ▷ **Question 4: (optionnelle)** Au lieu de chercher et recopier une seule lettre à la fois, utilisez deux lettres pour chaque. Par exemple, après le groupe "ar" de **argentine**, on sélectionnera le "mé" de **arménie**. On peut faire varier le nombre de lettres cherchées et le nombre de lettres recopiées. La fonction `getopt(3)` est pratique pour doter votre programme de paramètres en ligne de commande.
- ▷ **Question 5: (optionnelle)** Pour de meilleurs résultats, on peut compliquer l'algorithme en le découpant en deux phases. Lors d'une phase d'apprentissage, on mesure dans la liste de mots la probabilité de trouver chacune des lettres après un groupe de deux lettres données. Par exemple, après "in", on peut trouver dans la liste complète des pays les lettres suivantes : e (10 occurrences), a, m, r, s (1 occurrence chacune) ou une fin de mot (2 occurrences). On utilise ensuite ces probabilités dans un second temps pour générer les mots. Pour plus d'informations, voir "Markov name generator" dans un moteur de recherche.

★ **Exercice 4: Rien ne sert de courir (optionnel).**

La tortue est un animal ayant des mouvements limités et sur un terrain plat. Elle possède une position (`x` et `y` de type `double`) et une direction (un entier `dir` où 0 signifie nord, 1 est, 2 sud et 3 ouest).

▷ **Question 1: Déplacements**

Programmer les trois fonctions suivantes :

- `void go(double d)` qui fait avancer la tortue de  $d$  dans la direction courante;
- `void right()` qui change la direction de la tortue de 90 degrés vers la droite sans la déplacer;
- `void left()` qui change la direction de la tortue de 90 degrés vers la gauche sans la déplacer.

Pour dessiner le parcours de la tortue, on va utiliser du postscript.

▷ **Question 2: Affichage**

Écrire un programme qui :

- affiche `!postscript` sur une ligne
- à chaque appel de `go`, affiche une ligne de la forme `x1 y1 moveto x2 y2 lineto stroke` avec  $x_1$  et  $y_1$  les coordonnées de départ et  $x_2$  et  $y_2$  les coordonnées d'arrivée du `go`
- à la fin, affiche `showpage` sur une ligne.

Puis, rediriger la sortie du programme vers un fichier avec une extension `.ps` et l'ouvrir avec le programme `gv` ou `evince` par exemple.

On sait que la tortue commence avec `x = 297`, `y = 419`, et `dir = 0`.

▷ **Question 3: Le premier déplacement**

Écrire un programme qui fait faire à la tortue un carré de 10 unités de côté et qui l'affiche dans un fichier postscript.

Après d'intenses recherches, on a réussi à caractériser le mouvement de la tortue : la tortue a un mouvement récursif. Pour  $n$  entier, elle évolue selon la fonction `chemin(n)` :

- si  $n \leq 0$ , avancer de 2 unités;
- sinon, tourner à droite, faire `chemin(n-1)`, tourner à gauche et faire `chemin(n-1)`.

▷ **Question 4: Tortue récursive**

Programmer la tortue pour qu'elle effectue son chemin avec  $n = 12$ .

Quand la tortue est fatiguée, son chemin est différent :

- si  $n \leq 0$ , avancer de 2 unités;
- sinon, faire `chemin(n-1)`, tourner à gauche, faire `chemin(n-1)`, tourner à droite, faire `chemin(n-1)`, tourner à droite, faire `chemin(n-1)`, tourner à gauche et faire `chemin(n-1)`.

▷ **Question 5: Tortue fatiguée**

Programmer le chemin de la tortue fatiguée pour  $n = 4$ .