TD/TP 5 : Chaînes de caractères encore, pointeurs

Gilles Peskine

08 novembre 2002

1 Vrai ou faux

Dans chacun des fragments de programme suivants, indiquer si le test à la fin est vrai ou faux. Il y a des pièges...

```
char *a = "coucou"; char *b = a; if (a == b) ...
char *a = "coucou"; char *b = "coucou"; if (a == b) ...
char a[7] = "coucou"; char b[7] = a; if (a == b) ...
char a[7] = "coucou"; char b[7] = "coucou"; if (a == b) ...
char a[7] = "coucou"; char *b = a; if (a == b) ...
char a[7] = "coucou"; char *b = a + 3; if (a[1] == b[1]) ...
char a[7] = "coucou"; char *b = a + 3; if (&a[1] == &b[1]) ...
char a[7] = "coucou"; char *b = a + 3; if (&a[3] == b) ...
char a[7] = "coucou"; char *b = a + 3; if (&a[5] == b + 5) ...
char *a = "coucou"; char *b = "coucou"; if (a = b) ...
char a[7] = "coucou"; char b[7] = "coucou"; if (a = b) ...
```

2 Ligne de commande

Il existe deux manières de déclarer la fonction `main` :

- `int main ()`, ce qui est équivalent à `int main (void)`.
- `int main (int argc, char * * argv)`. Dans ce cas, `argv` est un pointeur sur un tableau de chaînes de caractères. `argv[0]` est le nom du programme, et les éléments suivants `argv[1]`, `argv[2]`, etc., sont les arguments passés sur la ligne de commande du programme, jusqu'à `argv[argc-1]`. Enfin `argv[argc]` est un pointeur nul (`NULL`).

(a) Écrire un programme qui affiche les arguments qu'on lui passe, un par ligne. Par exemple :

```
[toto@lenteja tp04]$ ./affiche_arguments bonjour tout le monde
bonjour
tout
le
monde
```

(b) Écrire un programme qui suppose que tous ses arguments sont des nombres entiers écrits sous forme décimale et qui calcule et affiche leur somme. Par exemple :

```
[toto@lenteja tp04]$ ./somme 3 42 25 18
88
```

(c) L'en-tête `stdlib.h` définit deux valeurs entières `EXIT_SUCCESS` et `EXIT_FAILURE` ; par convention, pour indiquer qu'un programme échoue, on lui fait renvoyer `EXIT_FAILURE`, tandis qu'on lui fait renvoyer 0 ou `EXIT_SUCCESS` s'il réussit. Adapter le programme du *(b)* pour qu'il détecte si un de ses arguments n'est pas un nombre entier. Dans ce cas, ne rien afficher, mais faire retourner au programme la valeur `EXIT_FAILURE`.

3 Comparaison de deux chaînes

Écrire une fonction qui compare deux chaînes de caractères. La fonction devra renvoyer 1 si les chaînes sont identiques et 0 sinon. Par exemples :
`chaines_identiques("coucou", "coucou")` $\Longrightarrow$ `1`
`chaines_identiques("coucou", "bonjour")` $\Longrightarrow$ `0`
`chaines_identiques("cou", "coucou")` $\Longrightarrow$ `0`

4 Lire et écrire des caractères

Voici quelques un des objets et fonctions déclarés dans `stdio.h`.[1]

```
FILE *stdin;    /* l'entrée standard : par défaut, le clavier */
FILE *stdout;   /* la sortie standard : par défaut, l'écran */
FILE *stderr;   /* la sortie pour les messages d'erreur : par défaut, l'écran */
int fputc(int c, FILE *f);  /* écrire un caractère c sur f */
int putchar(int c);         /* fputc(c, stdout) */
int fgetc(FILE *f);         /* lire un caractère depuis f */
int getchar(void);          /* getc(stdin) */
int fputs(char *s, FILE *f);    /* écrire une chaîne de caractères sur f */
int puts(char *s);              /* fputs(s, stdout); putchar('\n'); */
char *fgets(char *s, int n, FILE *f); /* lecture d'une ligne */
```

S'il n'y pas d'erreur, les fonctions `fputc` et `putchar` renvoient le caractère écrit, qui est toujours positif ; les fonctions `fputs` et `puts` renvoient un nombre positif ou nul ; les fonctions `fgetc` et `getchar` renvoient le caractère lu ; la fonction `fgets` renvoie son argument `c`. S'il y a une erreur, la fonction `fgets` renvoie `NULL` ; les autres fonctions renvoient la valeur `EOF`, qui est forcément strictement négative.

Les fonctions de sortie sont faciles à utiliser ; il faut juste faire attention au fait que `puts` ajoute un saut de ligne mais pas `fputs`. En revanche, les fonctions d'entrée ont des pièges dont il faut se méfier.

Pour des raisons historiques, sur certains systèmes (notamment Unix), le type `char` a des valeurs négatives, et la constante `EOF` vaut -1, qui est un `char`. La fonction `getchar` (ou `fgetc`) renvoie une valeur positive même si on entre un caractère négatif ; mais si on stocke cette valeur dans une variable de type `char`, elle dépasse la valeur maximale d'un `char`, et ceci donne une valeur négative.

Sous peine d'écrire un programme qui croit que le caractère -1 marque la fin d'un fichier, il faut utiliser `getchar` de la manière suivante :

```
int x;
x = getchar();
if (x == EOF) { /* c'est la fin du fichier */ }
else { char c = x; /* on utilise c normalement */ }
```

[1] Il y a quelques petites imprécisions dans cette table, mais ce bas de page est trop petit pour les expliquer.

Le piège de la fonction `fgets` est plus simple à comprendre, il faut juste s'en souvenir. Cette fonction lit et place dans la zone mémoire pointée par `s` au plus $n - 1$ caractères, suivis d'un caractère nul pour marquer la fin de la chaîne. De plus, `fgets` ne lit jamais plus d'une ligne. Si `fgets` s'arrête parce qu'elle arrive à la fin de la ligne, alors la chaîne `s` résultante se termine par un saut de ligne ; mais si `fgets` s'arrête parce qu'elle a lu le maximum de caractères, alors `s` ne contient pas de saut de ligne.

Exercices Pour mettre en pratique les fonctions `getchar` et `fgets`, écrire deux programmes qui comptent les lignes entrées sur `stdin`, l'un au moyen de `getchar`, puis l'autre au moyen de `fgets`. Pour le deuxième, on pourra dans un premier temps supposer que les lignes ne font pas plus de 100 caractères.

Remarque pratique Sous Unix ou sous Windows, pour exécuter le programme `monprog` avec le fichier `blabla.txt` comme entrée standard, on utilise la syntaxe suivante (redirection) :

```
monprog <blabla.txt
```

On peut aussi rediriger la sortie standard d'un programme dans un fichier :

```
monprog >sortie.txt
```

5 Comptage de mots

Écrire un programme qui compte les mots du fichier passé sur son entrée standard. Un mot est une séquence de caractères qui ne sont pas des caractères d'espacement.

Pour savoir si un caractère est un espacement, on utilisera la fonction `int isspace(int c)`, déclarée dans l'en-tête `ctype.h`. Cette fonction renvoie une valeur vraie si `c` est un caractère d'espacement, 0 sinon.

Il y a une complication avec `isspace`, du même type qu'avec `getchar` : cette fonction attend en fait un caractère non signé (et elle définie aussi sur la valeur `EOF` et renvoie 0). Il faut donc l'utiliser comme ceci :

```
char c = ...;
if (isspace((unsigned char) c)) ...
```

Remarque : il existe d'autres fonctions dans le même genre, notamment `isalpha` qui teste si un caractère est une lettre, `islower` et `isupper` qui reconnaissent les lettres minuscules et majuscules respectivement, `isdigit` qui reconnaît les chiffres (de `'0'` à `'9'`).

6 Dessiner du texte

Le but de cet exercice est d'écrire un programme qui prend des arguments sur sa ligne de commande et les affiche, un par ligne, et centrés. Par exemple :

```
[toto@lenteja tp04]$ ./centre-texte "Bonjour a" "tout le monde" "Comment ca va ?"
                                     Bonjour a
                                   tout le monde
                                  Comment ca va ?
```

(a) Écrire un tel programme. On supposera que la fenêtre dans laquelle le texte est affiché a 80 colonnes.

(b) Pour obtenir un résultat plus joli, utiliser la bibliothèque `graphics`. Vous aurez en particulier besoin des fonctions `gr_draw_string` et `gr_text_size`.

7 Graphe d'une fonction (le retour)

Écrire une fonction qui dessine le graphe d'une fonction numérique. Le prototype de cette fonction sera

```
void dessine_graphe(double (*f)(double));
```

8 Manipulations de textes

Nous avons couvert les bases de C. Voici quelques exercices supplémentaires pour ceux qui veulent s'entraîner. Les énoncés sont volontairement donnés sous une forme un peu moins précise (moins scolaire).

Note générale : pour les programmes qui doivent manipuler un texte, lire le texte de départ sur `stdin` et écrire le texte modifié sur `stdout`. Pour l'instant, vous pouvez supposer que les lignes ne dépassent pas 1000 caractères (nous verrons bientôt comment passer outre cette limitation).

(a) Écrire un programme qui « normalise » les blancs dans un texte. Ceci signifie remplacer toute séquence de un ou plusieurs blancs (espaces et tabulations) par un unique caractère espace, supprimer les blancs en début et en fin de ligne, supprimer les lignes vides...

(b) Écrire la fonction `fgets` à l'aide de la fonction `getc`.

(c) Écrire un programme qui « reformate » les lignes trop longues, c'est-à-dire qui passe à la ligne avant les mots qui dépassent à droite.

(d) Écrire un programme qui supprime tous les commentaires d'un programme C. Attention aux coups tordus comme `"/*"` (qui n'est pas un commentaire) et `/*/*/` (qui est un commentaire bien formé).

(e) Écrire une fonction qui effectue le traitement de C des chaînes de caractères. Par exemple, la chaîne

```
\"coucou\"\nbonjour
```

devra être transformée en

```
"coucou"
bonjour
```

On supposera déjà alloué un tableau de longueur suffisante pour stocker le résultat.

(f) Écrire une fonction qui effectue la transformation inverse. Il y a trois caractères « dangereux » qu'il faut remplacer par une séquence d'échappement : le guillemet double `"`, la barre oblique inverse `\`, et le saut de ligne.