Polytech Nice Sophia
SI3 / 2021-2022
Erick Gallesio

Programmation Procédurale

Feuille 6

Pointeurs

Chaînes, pointeurs sur fonction & arguments

1 Fonctions str...

Réécrire les fonctions suivantes à l'aide de pointeurs `(char *)` :

- `char *strcpy(char *dest, const char *src);`
- `size_t strlen(const char *s);`
- `char *strdup(const char *s);`
- `char *strchr(const char *s, int c);`

Consulter le manuel Unix pour verifier leur comportement respectif.

2 Pointeur sur fonction: La fonction map

Écrire la fonction `map` qui applique une fonction sur un tableau d'entiers. Cette fonction prend en paramètres:

- une fonction unaire `fct` ,
- un tableau d'entiers et
- la taille du tableau.

La fonction `fct` , quant à elle, prend un entier en paramètre et retourne un entier.

Soit le tableau `tab` , tel que

```
tab = {1, 2, 3 , 4, 5};
```

alors, l'appel `map(carre, tab, 5)` (où `carre` est une fonction qui renvoie le carré de son paramètre) modifie le tableau `tab` en

```
tab = {1, 4, 9 , 16, 25};
```

Note:

Pour tester votre programme, vous écrirez une fonction qui affiche le contenu de votre tableau. **Votre fonction d'affichage pourra être écrite à l'aide de la fonction `map` !**

3 Pointeur sur fonction: La fonction iterate

Écrire une fonction `iterate` qui calcule l'itération d'une fonction à deux arguments sur un tableau d'entiers. La fonction `iterate` prend en paramètre:

- une fonction `f` ,
- un tableau d'entiers `tab` et
- la taille du table `tab` .

La fonction `f` prend deux entiers en paramètre et retourne un entier.

Si le tableau contient les valeurs x_0, x_1, x_2, …, x_n, la fonction `iterate` retourne la valeur $f(f(\ldots f(f(x_0, x_1), x_2), \ldots), x_n)$.

Si la taille du tableau est respectivement 1, 2, 3 ou 4 la fonction retourne respectivement les valeurs

- x_0,
- $f(x_0, x_1)$,
- $f(f(x_0, x_1), x_2)$ et
- $f(f(f(x_0, x_1), x_2), x_3)$.

Comment utiliser cette fonction `iterate` pour calculer,

- le maximum du tableau `tab` ,
- la somme des éléments de `tab` ,
- le produit des éléments de `tab` ,

4 Commande echo

Ecrire la commande echo de Unix en gérant l'option `-r` qui permet, si elle est présente, d'afficher les arguments de la commande à l'envers (cette option n'existe pas normalement dans le `echo` d'Unix).

On rappelle que le nombre et la valeur des arguments d'un programme sont accessibles au travers des 2 premiers paramètres de la fonction `main` .

Exemple:

```
$ ./echo abc def
abc def
$ ./echo -r abc def
cba fed
$ echo abc def -r
abc def -r
$
```

5 Analyseur d'options Unix

Ecrire la commande `option` qui analyse les options et les paramètres passés sur la ligne de commande et qui les affiche sur la sortie standard. On se comportera ici suivant les conventions Unix habituelles:

- une option courte commence par un tiret (`'-'`) et utilise un seul caractère.
- plusieurs options courtes peuvent être concaténées (*e.g.* `-abc` est équivalent à l'utilisation des options `-a` , `-b` et `-c`)
- une option longue commence par un double tiret (`'--'`)
- les options sont placées avant les paramètres
- la fin des options est marquée par la présence du premier paramètre ou par la rencontre de la séquence spéciale `'--'`

Exemples:

```
$ options -xdf -y - -kj toto -z
Option courte: -x
Option courte: -d
Option courte: -f
Option courte: -y
Option courte: -k
Option courte: -j
Argument: toto
Argument: -z

$ options -ab --ab foo -ab
Option courte: -a
Option courte: -b
Option longue: --ab
Argument: foo
Argument: -ab

$ options -a -- -a a
Option courte: -a
Argument: -a
Argument: a
```

6 Commande cat

En utilisant l'analyseur d'options de la question précédente, écrire une version simplifiée de la commande `cat` . Ce programme ne gérera que les options `'-n'` et `'-E'` de la commande standard (Cf. manuel).

Rajouter ensuite l'option `'-h'` (et sa version longue `--help`) permettant d'afficher de l'aide sur la commande.

Note

Pour simplifier, votre fonction travaillera seulement sur le fichier standard d'entrée.

7 Retour sur la commande echo

On veut ajouter deux options à la commande `echo` précédente:

- l'option `-O` (pour `One`) qui affiche un mot par ligne
- l'option `-h` (mais aussi `--help`) qui affiche de l'aide et termine le programme

Par rapport à l'exercice précédent, on va essayer ici d'avoir une structure un peu plus «propre» et d'éviter l'utilisation de variables globales. On va pour cela définir le type `options` suivant:

```c
typedef struct {
  char sep;           // valeur du spérateur ' ' par défaut et '\n' si option -O
  int reverse;        // 1 si option -r et 0 sinon.
} options;
```

Écrire la fonction

```c
char **option_analysis(char **argv, options *opt);
```

qui initialise la structure `options` et avance sur toutes les options du programme. Lorsque cette fonction se termine, elle renvoie l'adresse du premier paramètre utile.

Avec ces conventions, votre programme (et tout programme qui traite des options) peut donc s'écrire de la façon suivante:

```
int main(int argc, char *argv[]) {
  options opt;

  for (argv = option_analysis(argv, &opt); *argv; argv++) {
     // Traiter le paramètre *argv
     ...
  }
  return 0;
}
```