TP3 : Arguments en ligne de commande et E/S

CSH : Initiation au C et au shell

★ 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 ***longueur_arg***, l'exécution de la commande
`longueur_arg 0 bonjour 2.56 adieu` produira la sortie :

```
l'argument no 0 contient 12 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: Les arguments de la ligne de commande sont des chaînes de caractères.

▷ **Question 1:** Écrivez un programme duppliquant un fichier *source* sous le nom *destination* (que vous demanderez à l'utilisateur). Pour la copie, vous utiliserez les fonctions `fprintf` et `fscanf` avec la chaine de formatage `"%c"`.

Réponse

La première chose à laquelle il faut les amener est un

```
while (!feof(IN)) {
  char c;
  fscanf(IN, "%c", &c);
  fprintf(IN, "%c", c);
}
```

C'est pas mal, ca marche presque, sauf que le dernier caractère du fichier source est écrit deux fois dans le fichier destination. C'est normal. Voici ce que dit la page man correspondante :

Remember fscanf returns EOF in case of errors or if it reaches eof. So you can check return value of fscanf instead of feof().

```
while (fscanf(in_fd, "%s", username) != EOF) {
  fprintf(out_fd, "%s\n", username);
}
```

Fin réponse

▷ **Question 2:** Modifiez votre programme pour qu'il prenne ses arguments (`source` et `destination`) 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 des fichiers concernés.

Réponse

L'un des trucs faux très souvent rencontré est quelque chose comme

```
char input[32];
if (argc<2) {
  scanf("%s",input);
} else {
  input=argv[1];
}
```

Ca ne peut évidement pas marcher car on tente d'affecter un tableau dans un autre, ce qui est interdit. Il faut soit changer input en un `char*` (mais là, attention à bien réserver la place nécessaire pour l'éventuel scanf, par exemple avec un malloc), soit utiliser `strcpy()` pour faire la copie d'argv dans input.

Fin réponse

★ **Exercice 3: Les arguments de la ligne de commande sont des entiers.**

Lisez le code source ***max2.c***, disponible dans le dépot. Il demande interactivement deux entiers à l'utilisateur et renvoie la valeur maximum lue.

On souhaite modifier ***max2*** pour lui fournir les données en arguments de la ligne de commande.

▷ **Question 1:** Pour réaliser cela, un programmeur pressé a modifié ***max2.c***, sous le nom ***max2_v2.c***. Examinez ce programme et compilez-le. Vous obtenez un message d'erreur qui signifie que les parties gauches et droites d'affectations ne sont pas du même type : en effet, `a` et `b` sont des entiers, `argv[1]` et `argv[2]` sont des adresses (pointeurs). Exécutez le code exécutable (qui est tout de même généré, car ces erreurs sont des *warnings*).

▷ **Question 2:** Notre programmeur pressé a fait une première tentative de correction en forçant une conversion de type, dans le fichier ***max2_v3.c***. Lisez-le, compilez-le et vérifiez que l'erreur de compilation ne se produit plus. Exécutez le programme obtenu ... et concluez.

▷ **Question 3:** Corrigez le programme en utilisant la fonction `atoi` (cf. `man atoi`).

★ **Exercice 4: Mesurer la complexité d'un fichier C.**

On souhaite avoir une mesure de la complexité de divers programmes C. Pour cela, plusieurs métriques sont utilisables, que nous allons explorer au fil des questions.

La première idée est de mesurer la longueur du texte. Il est assez courant d'annoncer le nombre de lignes composant un programme donné (Noyau linux 2.6 : 4 millions ; GCC : 2,5 millions ; Windows XP : 40 millions, ...).

▷ **Question 1:** Faites un programme C `complexite` prenant le nom d'un fichier en argument et comptant le nombre de lignes le composant (il faut compter les occurences du caractère '\n').

▷ **Question 2:** Modifiez votre programme pour ne pas tenir compte des lignes blanches (il faut utiliser un booléen réinitialisé à chaque ligne indiquant si on a vu un caractère autre que '\t' et ' ').

▷ **Question 3:** (facultative) Modifiez votre programme pour ne pas compter ce qui se trouve à l'intérieur de commentaires. On souhaite traiter à la fois le style C (`/* ... */`) et le style C++ (`// ... fin de ligne`). On utilisera un booléen indiquant si on se trouve actuellement dans un commentaire ou non.

Cette métrique de complexité est trompeuse car un long programme ne comportant qu'un enchaînement de commandes sans `if`, `for` ou autres est sans doute moins complexe qu'un programme plus court présentant un schéma d'exécution plus complexe.

▷ **Question 4:** Modifiez votre programme pour qu'il compte les occurences du caractère '{', qui est présent dans la plupart des structures syntaxiques du C.

▷ **Question 5:** Voici deux extensions possibles pour votre programme, à réaliser si vous avez le temps :

- Comptez séparément les occurences des différentes constructions syntaxiques du C.
- Faites en sorte que votre programme compte les lignes lorsqu'on lui passe l'argument supplémentaire `--long` tandis qu'il compte la complexité si on lui passe l'option `--complexe`.

★ **Exercice 5: Pour aller plus loin : les récipients en C.**

Réponse

Pas de panique sur cet exo, il est optionnel. Le résoudre est sans doute plus complexe que le projet de TOP. C'est donc pour occuper ceux qui s'ennuient les soirs chez eux. Renvoyez ceux que ca intéresse vers moi, on gèrera par mail.

Fin réponse

Reprenez problème des récipients vu en TD et en TP dans le module de TOP [1]. Implémentez une solution générique pour ce problème, sans limitation sur le nombre de récipients ni borne arbitraire sur la profondeur d'exploration. Utilisez l'algorithme le plus efficace vu pour ce problème, c'est-à-dire celui stockant chaque état déjà rencontrés sous forme d'un entier.

Réponse

Bon, le sujet de TOP est faux, et la façon de stocker dans un entier proposée ne marche pas. Faut que je corrige le problème de TOP, mais c'est une autre histoire.

Fin réponse

1. Ces sujets sont disponibles sur la page `http://www.loria.fr/~quinson/teach-prog.html`

▷ **Question 1:** Résolvez l'instance {5,3}→4. C'est à dire que l'on dispose de deux récipients de taille respective 5 et 3, et que l'on cherche à générer la quantité 4. Comme présenté dans le sujet de TOP, cette instance a une solution en 6 transvasements.

▷ **Question 2:** Résolvez les instances suivantes {8, 13, 21}→1 ; {13, 21, 34}→1 ; {21, 34, 55}→1 ; {34, 55, 89}→1 ; {55, 89, 144}→1 ; {89, 144, 233}→1. Pour les plus grosses de ces instances, un parcours en largeur de l'arbre des possibilités est bien plus efficace que la méthode de backtracking proposée dans le sujet de TOP.

▷ **Question 3:** Résolvez l'instance {1597, 2584, 4181}→1. Pour cela, il vous faudra sans doute observer les solutions obtenues à la question précédente, et «avoir une bonne idée»...

Réponse

On observe en effet que quand on donne ainsi des nombres successifs de la suite de fibonacci, seuls les deux premiers récipients sont utilisés dans la solution optimale. Si on donne à l'algo de résolution les trois valeurs de cette question, le problème explose en mémoire et ca ne converge pas. Si on supprime le récipient 4181, une implémentation un peu efficace trouve en moins d'une seconde la solution, qui compte 5166 transvasements (quand même).

Fin réponse

▷ **Question 4:** Peut-être existe-t-il un algorithme pour créer des instances de problèmes arbitrairement complexe ?

Réponse

Ce truc ressemble à une forme étendue du calcul du PGCD, je trouve. Y'a ptet moyen de trouver une idée. Je sais pas vraiment, une histoire avec beaucoup de taille de récipients premières entre elles, et la cible le pgcd d'un ensemble ?

Le fait que les valeurs de fibbo donnent des instances aussi complexes me bluffe un peu. Y'a quelque chose que je comprend pas là dedans.

Fin réponse