Algorithmique 2 : structures récursives, linéaires et binaires

Projet

Lignes répétées

Le projet consiste à écrire un programme en C dont le but est :

— si le nom d'un seul fichier figure sur la ligne de commande, d'afficher, pour chaque ligne de texte non vide possédant plusieurs occurrences dans le fichier, la suite strictement croissante des numéros de ligne auxquels elle se situe ;

— si au moins deux noms de fichiers figurent sur la ligne de commande, d'afficher, pour toutes les lignes de texte de contenus non vides et différents appartenant au premier fichier, leurs nombres d'occurrences dans ce premier fichier et dans les autres.

Les fichiers sont supposés être des fichiers texte. La longueur des lignes n'est pas limitée.

L'exécutable doit être nommé `lntracker`.

L'affichage se fait en colonnes sur la sortie standard. Les colonnes sont séparées par le caractère de tabulation horizontale `'\t'`. La première ligne affichée reprend les noms des fichiers. Pour les lignes suivantes :

— dans le premier cas, l'affichage se fait sur deux colonnes. La première colonne est réservée à la suite des numéros de ligne, la deuxième, au contenu de la ligne répétée. Les numéros de ligne sont séparés par une virgule ;

— dans le deuxième cas, l'affichage se fait sur une colonne de plus que le nombre de noms fichiers : figurent dans les premières colonnes les nombres d'occurrences, dans la dernière, le contenu de la ligne commune.

Exemples

Suivent des traces d'exécutions sur des fichiers qui figurent dans le dossier `cm/9/hashtbl_mge/`. Pour simplifier, c'est de ce dossier qu'est lancé l'exécutable.

Voici une illustration du premier cas :

```
$ ./lntracker hashtbl.c
hashtbl.c
14,89,112,116,154⟶    }
21,55,101⟶  size_t hkey;
23,31⟶};
37,44,99,130⟶    return NULL;
38,45,48,65,68,77,80,90,100,122,131,158⟶  }
52,70,95,125,138,143,159⟶}
104,118⟶    (*pp) -> value = value;
110,153⟶      }
124,137⟶  return value;
128,141⟶  clist **pp = hashtable_search(ht, key, NULL);
```

Et en voici une du second :

```
$ ./lntracker hashtbl.c hashtbl.h
hashtbl.c⟶hashtbl.h
```

Il n'y a pas d'erreur : rien de strictement commun à ces deux fichiers.

Deux résultats sensiblement différents et plus intéressants peuvent être obtenus si seuls les caractères alpha-numériques (test de caractère `isalnum`) qui constituent les lignes sont retenus. Voici ce que cela donne pour la première exécution :

```
$ ./lntracker hashtbl.c
hashtbl.c
12,105──────⟩else
21,55,101───⟩sizethkey
27,34───────⟩intcomparconstvoidconstvoid
37,44,99,109,115,130────⟩returnNULL
83,149──────⟩whilepNULL
85,150──────⟩clisttp
103,114,129─⟩ifppNULL
104,118─────⟩ppvaluevalue
121,136─────⟩htnentries1
124,137─────⟩returnvalue
128,141─────⟩clistpphashtablesearchhtkeyNULL
135,152─────⟩freet
```

Et voici pour la seconde :

```
$ ./lntracker hashtbl.c hashtbl.h
hashtbl.c───⟩hashtbl.h
1───────────⟩1──────────⟩constvoidvalue
2───────────⟩1──────────⟩intcomparconstvoidconstvoid
```

Mise en œuvre

Votre programme doit être écrit en C et doit supporter les options de compilation usuelles : `-std=c11`, `-Wall`, `-Wconversion`, `-Werror`, `-Wextra` et `-Wpedantic`. Vous pouvez ajouter toute autre option (`-O2` par exemple) qui ne contre pas les précédentes.

Il doit être correctement indenté (2 espaces), espacé (une espace après un mot-clé comme **`if`** ou **`while`**, une virgule, aucune espace après une parenthèse ouvrante et avant une parenthèse fermante, une espace de chaque côté d'un opérateur binaire...), ne doit comporter aucune tabulation. Aucune des lignes de texte qui y figurent ne doit excéder 80 caractères ; si une instruction longue figure sur plusieurs lignes, les lignes en excès sont double indentées par rapport à la première.

MÀJ 16-12 ajout de la double indentation

Les identificateurs des types, sous-programmes et paramètres doivent être correctement nommés. Vous éviterez toute duplication de code.

Si vous utilisez une partie interface ou implantation d'un module qui a été fournie à l'occasion du cours, vous ne devez en aucun cas la modifier. Si vous utilisez une partie interface ou implantation d'un module qui a été fournie à l'occasion d'une séance de travaux pratiques, vous ne pouvez la modifier qu'à la condition expresse que la modification ait été l'objet d'une commande figurant dans un sujet d'une fiche de travaux pratiques. Au besoin, créez vos propres modules. Dans l'ordre de préférence, vous utiliserez ou développerez des modules utilisant le type pointeur générique **`void *`**, la technique de généricification par paramétrisation et, en dernier lieu, des modules d'un seul type non pointeur générique.

Toutes les zones mémoires explicitement allouées devront être désallouées avant la fin de l'exécution, même en cas d'erreur.

L'option `--help` doit obligatoirement être supportée et doit afficher l'aide selon le format usuel. D'autres options peuvent être développées :

`-c TYPE` ou `--case=TYPE` : considérer les lignes après la transformation `TYPE`, de valeur `upper`, en majuscules, ou `lower`, en minuscules ;

`-f MOTIF` ou `--filter=MOTIF` : ne retenir des lignes que les caractères décrits par `MOTIF`, suite non vide de caractères choisis parmi `a`, lettres (correspondant au test de caractère `isalpha`), `c`, de contrôle (`iscntrl`), `d`, chiffres (`isdigit`), `n`, alpha-numériques (`isalnum`), `p`, de ponctuation (`ispunct`), `s`, espaces (`isspace`) ;

`-s` ou `--sort` : procéder à l'affichage dans l'ordre (`strcmp`) du contenu des lignes.

Modalités

Vous pouvez réaliser ce projet à deux, pas plus.

Votre projet sera compilable et exécutable sur les machines des salles de TP et sous Xubuntu.

Votre projet devra être rendu avec tous les fichiers source nécessaires, un fichier `makefile`, un rapport de développement, un jeu d'exemples illustrant le mieux possible ses fonctionnalités et performances. Le rapport sera fourni sous forme électronique (fichier au format `pdf`) qui précisera l'objet de chacun des modules, décrira l'implantation, commentera les exemples, explicitera les limites éventuelles du programme.

Vous remettrez votre projet au plus tard le **vendredi 6 janvier 2017 à 12 h** dans une archive au format `tar.gz` de nom *identifiant*`_projet.tar.gz`, où *identifiant* est l'identifiant de 8 caractères de l'un des membres du groupe. Cette archive sera envoyée en pièce jointe à un courriel à l'adresse de votre chargé de TP, de sujet « Algo2 projet ». Un seul envoi par groupe : l'éventuel autre membre du groupe sera mis en copie du message.

Vous soutiendrez le projet individuellement, oralement, sur une machine des salles de TP et sous Xubuntu.

La notation sera sensible à la propreté du code, à son homogénéité, aux performances de l'exécutable, à la clarté des explications fournies dans le rapport et lors de la soutenance, ainsi qu'à un passage sans encombre au grill `valgrind`.

Foire aux questions

▶ 1 On peut donc utiliser le module `hashtbl` qui figure dans le dossier `algo2_src/cm/9/hashtbl_mge/` ainsi que le module `slist` qui figure lui dans le dossier `algo2_src/cm/9/slist_mge/` ?

Oui. Et vous pouvez modifier `slist` en ajoutant la procédure `slist_sort` comme indiqué dans la fiche n° 8 de TP . Mais si vous envisagez de faire plus de modifications, créez votre propre module.

▶ 2 Si vous aviez à faire le projet, vous vous cantonneriez aux modules que vous nous avez fournis ?

J'ai fait le projet et j'ai utilisé les modules `hashtbl` et `sllist`. Je n'ai pas modifié le premier. J'ai modifié le second en lui ajoutant la procédure `slist_sort`. En plus du fichier contenant la fonction principale, j'ai créé un module dévolu à la gestion de tableaux d'entiers du type **`long int`** et un autre à la gestion de la ligne de commande et des options.

Votre fichier qui contient la fonction principale doit faire voir la façon dont vous utilisez les différents modules. Il ne devrait contenir aucune allocation explicite (appel à `malloc`). Ni de déclaration de structures.

▶ 3 Peut-on supposer que la longueur des lignes est limitée ?

Par `LONG_MAX` ! Donc : non. En revanche, vous pouvez faire le choix ne mémoriser que des préfixes d'une longueur limitée des lignes : cela fera partie des limites de votre programme. Si tel est votre choix, il faudra l'indiquer au moins par un message d'erreur approprié du style « seul le préfixe de longueur *p* de la ligne numéro *n* est mémorisé » et vraiment couper la ligne, c'est-à-dire ignorer les caractères situés après cette limite et avant la fin de ligne `'\n'`.

▶ 4 Que doit afficher l'option `--help` ?

L'aide selon le format usuel. Si votre programme ne prend en compte que cette option, il doit rejeter toutes les autres en signifiant quelles lui sont inconnues ou qu'il ne peut les traiter. Les options courtes sont introduites par le tiret suivi d'un seul caractère, les longues, de deux tirets suivis d'une suite de caractères de longueur au moins un. `--help` est une option longue sans argument.

L'aide doit être envoyée sur la sortie standard. La demande d'aide doit mettre fin au programme avec le renvoi de `EXIT_SUCCESS`. Tout message d'erreur doit être envoyé sur la sortie erreur et sanctionnée d'une fin de programme avec renvoi de `EXIT_FAILURE`. Tout message d'avertissement doit être envoyé sur la sortie erreur.

▶ 5 On est obligé d'introduire des options en plus de l'obligatoire `--help` ?

En aucun cas. Mieux vaut un programme sans option hormis `--help`, avec du hachage utilisé à bon escient, une maitrise complète des ressources mémoire (aucune fuite de mémoire rapportée par `valgrind`, une bonne modularité et un rapport clair et complet, qu'un programme fouillis, illisible, avec plein d'options et un rapport creux. N'oubliez pas que votre activité de programmation s'adresse à ceux qui devront la lire.

▶ 6 Des exemples de tests à faire ?

De manière générale, comparez vos résultats avec ceux des autres.

Commencez par des fichiers source C. Enchainez sur les deux petits fichiers `bobdylan_iwysb.txt` et `thebeatles_iwy.txt` qui figurent dans l'archive `textes_tests.tar.gz` à trouver sur la plateforme UniversiTICE. Continuer avec de plus gros. Pour `lesmiserables.txt`, vous pourrez observer que « `C'est la faute a Voltaire,` » par quatre fois, aux lignes 11 421, 11 426, 11 431 et 11 438, alors que « `C'est la faute a Rousseau.` » une fois de moins, aux lignes 11 423, 11 428 et 11 433.

Pour tester des limites maintenant.

Si vous voulez tester la capacité de votre exécutable `lntracker` à supporter de longues lignes et avez deux fois 2,7 Go libres sur votre disque, 1) produisez l'exécutable associé au texte :

```c
#include <stdlib.h>
#include <stdio.h>

#define MAX 2000

int main(void) {

  for (long int k = 0; k < MAX; ++k) {
    for (long int j = k * k; j > 0; --j) {
      putchar('0' + rand() % 10);
    }
    putchar('\n');
  }
  return EXIT_SUCCESS;
}
```

2) exécutez-le en redirigeant la sortie standard vers un fichier, 3) exécutez votre `lntracker` avec le nom de ce fichier en un puis deux exemplaires sur la ligne de commande en redirigeant la sortie standard vers autre fichier et 4) analysez les deux productions.

Si vous voulez tester sa capacité à gérer les suites de numéros de lignes, que vous ayez choisi de les coder par des listes dynamiques simplement chainées ou des tableaux extensibles, et avez 1 Go libres, même manipulation mais avec :

```c
#include <stdlib.h>
#include <stdio.h>

#define MAX 1000000

```

```
6  int main(void) {
7    for (long int k = 0; k < MAX; ++k) {
8      putchar('0' + rand() % 10);
9      putchar('\n');
10   }
11   return EXIT_SUCCESS;
12 }
```

et un seul exemplaire du nom de fichier sur la ligne de commande.

Si vous voulez tester sa capacité à compter et avez 2 Go libres, ajoutez trois zéros à la fin de la macroconstante du dernier texte (passant ainsi du million au milliard), procédez de même mais avec deux exemplaires (ou plus) du nom de fichier.

► 7 C'est une blague, une légende, une rumeur, le 0/20 si la moindre erreur est produite par `gcc` à la compilation ?

C'est la note juste pour un travail de programmation.

0/20 de même pour la moindre tabulation dans un fichier source C, `.c` ou `.h`. Jouez du Ctrl+H en cas de souci avant de rendre votre projet.

MÀJ 16-12 ajout de la grille d'évaluation

En dehors des cas rédhibitoires — les précédents, le plagiat, la méconnaissance du projet présenté —, la grille d'évaluation est :

catégorie	*points*
nom de l'archive correct	0,5
archive propre	0,5
makefile correct	0,5
option `--help` correcte au regard de ce qui est développé par l'exécutable	1
gestion correcte de la ligne de commande	1
test `valgrind` probant	1,5
développement abouti de la fonctionnalité pour un fichier	1
format de sortie correct pour la fonctionnalité pour un fichier	0,5
développement abouti de la fonctionnalité pour deux fichiers ou plus	1
format de sortie correct pour la fonctionnalité pour deux fichiers ou plus	0,5
« codage », dont indentation correcte, uniformité du style, lisibilité, non redondance, absence de nombres magiques, complexités satisfaisantes	7
rapport, dont justification du choix des structures de données, illustration, descriptif des modules développés et spécification des fonctions, explicitation des limites, problèmes rencontrés lors du développement	5

► 8 Sur la ligne de commande, c'est d'abord les options puis ensuite les noms de fichiers ?

MÀJ 16-12 ajout de la question-réponse 8

Oui si vous précisez que c'est comme cela que fonctionne votre exécutable.

Non si vous observez le fonctionnement des commandes Linux et que vous voulez vous en rapprocher : 1) si une option qui propose de dispenser une information puis de terminer l'exécution (aide, version) figure sur la ligne de commande et qu'il s'agit de la première de la sorte, elle est exécutée ; 2) les autres options sont ensuite repérées, traitées ; 3) ce qui n'est pas option est nom de fichier. D'autre part, lorsqu'ils sont utilisés seuls, les préfixes `-` et `--` des options courtes et longues indiquent que le paramètre qui suit sur la ligne de commande est un nom de fichier ; cela permet de prendre en compte les fichiers dont les noms commencent précisément par ces préfixes. Enfin, il doit être autorisé que seul un préfixe des options longues suffise s'il n'y a pas ambigüité : on peut très souvent taper `--h`, `--he`, `--hel` à la place de `--help` par exemple ; mais, toujours par

exemple, `--h` est une option ambigüe pour la commande `ls` puisqu'elle dispose de `--help`, `--hide`, `--hide-control-chars` et `--human-readable`.

À vous de voir ce qui, pour vous, est *human feasible*...

▶ 9 C'est quoi de déroulement d'une soutenance ?

MÀJ 16-12 ajout de la question-réponse 9

Ça peut être très rapide si l'examinateur comprend que vous êtes là simplement parce que vous avez réussi à persuader un autre d'ajouter votre nom pour former un binôme. Sinon, l'examinateur peut commencer par vous demander d'illustrer la structure de données au tableau sur un exemple simple. L'examinateur tentera ensuite, sous son propre compte, de produire l'exécutable du projet envoyé puis de l'exécuter sur des exemples qu'il aura sous la main ou que vous aurez joints (pas trop gros les exemples, pour ne pas saturer nos messageries, et pertinents). Il épluchera avec vous certaines parties de votre code, vous interrogera dessus.

— *Quoi d'autre ?*