Travaux pratiques de C avancé

Guillaume.Duc@enst-bretagne.fr
Stephanie.Even@ensmp.fr
Serge.Guelton@telecom-bretagne.eu
Ronan.Keryell@enstb.org
Majeure informatique, UV2 INF 446, TÉLÉCOM Bretagne
Id: TP.tex 3738 2009-05-11 13:19:11Z keryell

11 mai 2009

Résumé

Ces travaux pratiques sont (trop) longs et ne sont pas forcément à faire en entier. Chaque élève fera ce qu'il pourra en fonction de ses propres capacités.

1 Introduction

Chaque élève est sensé savoir lire, voire même de lire un peu plus loin...

Il faudra généralement faire un `Makefile` pour chaque fichier mais on le fera *minimal* en exploitant au maximum les valeurs par défaut. Même *sans* `Makefile`, pour compiler le programme `toto.c` taper :

```
make toto
```

Vous avez intérêt à instruire `make` que vous voulez faire du C99 par défaut avec

```
export CFLAGS=-std=c99
```

ou

```
setenv CFLAGS -std=c99
```

selon votre religion de *shell*.

Comme chaque module d'enseignement se doit d'avoir une note pratique, votre comportement lors du TP sera évalué et noté. Il faut donc persuader l'encadrant que vous êtes très bon(ne)... On ne demande pas de rendre quelque chose.

Les versions électroniques de ce document et du cours sont accessibles depuis `https://info.enstb.org/enseignement/tc/c_avance`, voire depuis Moodle.

Merci d'utiliser un environnement de développement raisonnable qui puisse vous aider à écrire du C plus rapidement ainsi que mettre au point plus facilement. Il faudra passer du temps en dehors des heures de TP à se configurer un environnement qui dépote :

- coloration syntaxique ;
- indentation automatique ;
- aide à la saisie de code ;
- aide à la compilation ;
- aide à la mise au point.

Si vous voulez utiliser Eclipse pour faire du C, penser à faire

```
SETUP ECLIPSE_IDE_C
```

et le menu `Project/Build All` pour compiler votre programme.

Pensez de temps en temps aux débogueurs qui sont là pour vous aider :

- pour la mise au point des problèmes d'utilisation mémoire, on utilisera
  `valgrind <votre programme>`
- pour le débogage plus profond, vous utiliserez un des systèmes proposés par votre environnement de développement intégré, ou à la main après avoir compilé avec l'option `-g`. Cela est fait à la compilation au niveau du `Makefile` en rajoutant un `CFLAGS=-g` ou en modifiant la variable de votre environnement sous votre *shell* `bash` avec un
  `export CFLAGS="-g -std=c99"`
  ou sous votre *shell* `tcsh` avec un
  `setenv CFLAGS "-g -std=c99"`
  par exemple.
  Ensuite, on lance le débogueur graphique avec
  `ddd <votre programme>`
  ou un plus traditionnel
  `gdb <votre programme>`
  voire si vous utilisez l'éditeur de texte Emacs
  `M-x gdb`

Dans la suite on peut demander de mesurer des performances pour comparer différentes solutions entre elles. On se basera sur l'exemple dans :
`~keryell/sources/produit_scalaire`.

2 Affichage des arguments d'une commande

Écrire un programme `a.c` qui affiche les arguments passés en ligne de commande.

Tester par exemple avec :

```
./a Ça marche à 100 % !
```

(évidemment il faudra éventuellement adapter pour que cela marche dans votre environnement).

Compléter le programme pour qu'il affiche en plus toutes les variables d'environnement (lire le manuel de `environ`). La bibliothèque C ayant plusieurs potentialité, pour avoir la variable `environ` et des fonctions styles `putenv()` et `getenv()`, on a intérêt à mettre *avant tout* `#include` un

```
#define _GNU_SOURCE
```

qui a pour effet de mettre en fonction toutes les possibilités de la bibliothèque C.

Tester. Rajouter une variable d'environnement `POURCENT` avec la valeur `%`.

3 Afficher des paramètres de votre ordinateur

On se propose d'afficher de manière graphique l'évolution de quelques paramètres de votre ordinateur, tels que la température de composants de votre ordinateur.

Regarder ce qui se trouve dans le répertoire système « magique » `/proc` dont le contenu évolue en temps réel, en particulier `/proc/acpi/thermal_zone` si la mesure de température existe, ou encore la charge de la machine dans `/proc/loadavg`, le nombre d'interruptions dans `/proc/interrupts`, le champ `MemFree:` dans `/proc/meminfo`, le nombre d'octets passant sur les interfaces réseau dans `/proc/net/dev`, etc. Lire le manuel de `/proc` si vous voulez plus de détails.

Écrire un programme qui analyse un des fichiers qui vous intéresse dont vous avez passé le nom en paramètre avec une période en seconde que vous passerez en paramètre sur la ligne de commande.

Concrètement, le programme sera composé d'une boucle infinie qui effectuera :

- ouverture du fichier de nom passé en paramètre avec `fopen()` ;
- lire les informations qui vous intéressent avec un ou plusieurs `fscanf()` ;

– afficher l'information à l'écran avec `printf()` ;
– fermer le fichier avec `fclose()` ;
– attendre le temps indiqué par la période avec `sleep()`

On sortira du programme en tapant `^C`.

On testera évidemment les possibilités d'erreur lors de l'exécution. On utilisera la fonction `perror()` pour afficher de manière sympathique les erreurs le cas échéant.

Pour les élèves les plus avancés : on testera si on ne peut pas optimiser le programme en utilisant `fseek()` pour relire au début du fichier d'information à chaque fois plutôt que de perdre du temps à fermer et ouvrir à chaque itération[1].

Modifier le programme pour calculer une valeur moyenne du paramètre ou sa valeur par unité de temps si cela a un sens et l'afficher en même temps.

Enfin modifiez votre programme pour démarrer l'outil d'affichage graphique de donnée GnuPlot et lui envoyer les données à afficher et le télécommander. Utiliser la fonction `popen()` pour ce faire. L'idée est de demander à GnuPlot de retracer la fenêtre à chaque fois en rechargeant les données et en réaffichant[2].

GnuPlot permet d'afficher des données et des fonctions de plein de manières mais on va l'utiliser de manière très simple pour afficher des donnée séquentielles numériques entrées sur l'entrée standard. Il faut générer pour chaque affichage un flux de style :

```
plot '-' with lines
```
y_0
y_1
y_2
$\vdots$
y_n
`e`

où la lettre `e` marque la fin des données. Cela aura pour effet de tracer les points (i, y_i). On peut aussi entrer les données de la forme $x_i \quad y_i$, superposer plusieurs courbes avec `plot '-','-','-'`, etc.

4 Chenillard logiciel

Le club SeL a besoin d'un nouveau chenillard pour animer ses projecteurs mais n'a plus trop les moyens pour investir... Il s'agit donc de développer le logiciel d'animation. ☺

L'idée est de sortir sur un port parallèle des signaux binaires qui commanderont les projecteurs, chaque projecteur étant relié à un bit d'un nombre binaire. Bon, comme ici on n'a pas le matériel pour, on va afficher les choses à l'écran.

On va utiliser les opérations binaires sur les entiers pour faire des animations :
– les décalages permettent de décaler les motifs lumineux ;
– la négation permet d'inverser un motif visuel.

On rappelle qu'on peut saisir des nombres directement en binaire en C extension GNU avec la notation commençant par `0b`.

Il faudra développer :
– une procédure affichant à l'écran l'état de tous les bits d'un entier sous forme de caractère `*` (si 1) ou espace (si 0) ;
– initialiser l'entier à une valeur et faire évoluer avec des décalages ou autre tout en affichant le résultat à chaque pas de temps ;

[1] En fait il semble que la difficulté réside dans le fait que le contenu dynamique du fichier n'est pas vu comme tel par la bibliothèque du C : le fichier n'est pas relu physiquement au niveau du système, seule la valeur déjà dans le tampon est renvoyée. Il faut donc faire un `fflush()` avant le `fscanf()`, voire pour les plus aventuriers de supprimer le tampon via `setvbuf()`.

[2] Ne pas oublier de faire des `fflush()` pour être sûr de bien pousser les informations quand on en a vraiment besoin.

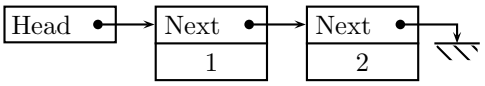

FIG. 1 – Principe d'une liste chaînée d'entiers.

– mettre un système à boucle d'attente active[3] ou basée sur l'appel système `nanosleep()` pour pouvoir varier la vitesse de l'animation (qu'on pourra faire évoluer dynamiquement). Pour pouvoir utiliser cette fonction qui a disparu de la bibliothèque du C avec C99 il faut avoir un :
```
#define __USE_POSIX199309
#include <time.h>
```
en début de fichier (le `#define` doit être avant tout autre `#include`).

Inventer de nouveaux motifs d'animation (symétrie...), voire un chenillard en 2D si vous êtes véloces.

Réaliser un compteur de GRAY et comparer l'aspect visuel par rapport à un compteur normal (`n ^ (n >> 1)` pour un code de GRAY).

5 Bibliothèque de listes chaînées

Lorsque l'on programme, il est très souvent nécessaire de stocker les données que l'on manipule dans des conteneurs comme par exemple, des tableaux, des listes, des dictionnaires, des ensembles, etc. Nous allons nous intéresser ici à la réalisation d'une petite bibliothèque permettant de stocker des données dans une structure souvent utilisée, une liste chaînée[4]

5.1 Liste d'entiers

Vous trouverez sur la figure 1 le principe de base d'une liste chaînée d'entiers. Chaque nœud de la liste contient deux éléments : un pointeur vers le nœud suivant dans la liste et l'élément (donc ici un entier) que l'on veut stocker à cette position dans la liste.

Notons qu'un identifiant de (début de) liste est simplement une variable de type pointeur vers un nœud.

Écrivez une petite bibliothèque permettant de créer et de manipuler des listes chaînées d'entiers. Vous pourrez notamment écrire des fonctions permettant :

– d'ajouter un élément entier en début d'une liste (cette **procédure**[5] sera utilisée pour construire ensuite plusieurs des fonctions suivantes),
– d'imprimer les éléments de la liste passée en argument,
– d'obtenir le nombre d'éléments d'une liste passée en argument,
– d'ajouter un élément à une position donnée dans une liste (on peut réutiliser la première fonction),
– de supprimer un élément situé à une position donnée,
– d'appeler une fonction (donnée en paramètre via un pointeur de fonction) pour chaque élément d'une liste,
– de dupliquer tout le contenu d'une liste (on peut utiliser la fonction précédente à l'occasion).

[3] Les fameux bogomips...

[4] Dans la vraie vie on utiliserait des bibliothèques toutes faites style Glib ou probablement on programmerait en C++ avec la STL ou Boost qui fournissent plein de modèles de conteneurs et d'algorithmes, mais ici le but est développer du code et de sensibiliser à la problématique et aux pointeurs. ☺

[5] C'est à dire une fonction qui renvoie un `void`...

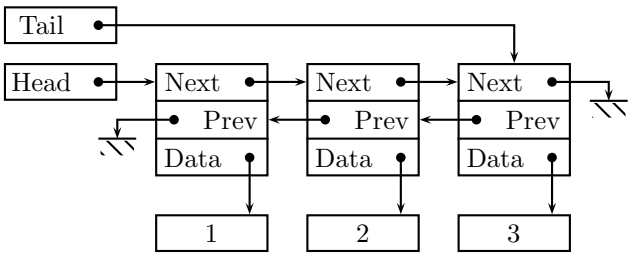

Fig. 2 – Principe d'une liste doublement chaînée.

Écrivez également un petit programme permettant de tester les différentes fonctions de votre bibliothèque. Vous ferez particulièrement attention à la gestion de la mémoire dans les fonctions de votre bibliothèque.

5.2 Généralisation

Comme il n'y a pas que les entiers dans la vie, et que vous n'avez bien entendu pas envie de coder de nouveau votre bibliothèque pour chaque nouveau type de donnée que vous inventez, modifiez votre bibliothèque pour qu'elle soit le plus générique possible.

Pour réaliser cette opération, vous pouvez, au lieu de stocker un entier dans chaque nœud de la liste, y stocker un simple pointeur vers les données. Vous réaliserez deux versions de cette bibliothèque, une première version où c'est le programme utilisant votre bibliothèque qui aura la charge de la gestion de la mémoire où sont stockés les éléments de votre liste (la gestion des structures décrivant les nœuds de votre liste restant à la charge de votre bibliothèque), et une seconde où votre bibliothèque aura seule la responsabilité de la gestion mémoire (elle réalisera, par exemple, une copie des éléments à ajouter dans la liste, retournera une copie d'un élément stocké et non l'élément lui-même, etc.).

On pourrait aussi faire une version où on utilise un type indéfini[6] à la fin de la structure de liste. Dans ce cas il faudra fournir la taille de cette partie à la fonction d'allocation d'un élément de la liste.

5.3 Listes doublement chaînées

Les listes chaînées sont souvent utilisées mais elles imposent un parcours dans un sens unique de la liste. Il existe des listes doublement chaînées permettant le parcours de la liste dans les deux sens. La figure 2 décrit le principe de ces listes. Comme on voit qu'une liste est identifiée par 2 paramètres, le début et la fin, on créera aussi un nouveau type de structure pour rassembler ces 2 information dans un même objet.

Créez une nouvelle bibliothèque, en vous basant sur celles que vous venez d'écrire, permettant de manipuler des listes doublement chaînées, ainsi qu'un programme de test utilisant cette bibliothèque.

5.4 Conclusion

Si vous voulez tout savoir sur les listes chaînées et comment les programmer en C, vous pouvez vous trouverez des documents détaillés aux adresses suivantes : `http://cslibrary.stanford.edu/103/` et `http://cslibrary.stanford.edu/105/`. Bien entendu, vous n'avez pas trop intérêt

[6] Une nouveauté du C99, vous vous souvenez ?

à regarder ces documents durant ce TP car cela diminuerait votre temps d'expérimentation très « pédagogie active »...

Au fait, l'allocateur mémoire `malloc()/free()` utilise dans votre dos des listes chaînées pour découper des gros bouts de mémoire en petits bouts que vous demandez...

Enfin, les listes sont un type de base fondamental dans le langage LISP puisqu'elles sont utilisées pour représenter les données et les... programmes ! Donc si vous utilisez Emacs, vous faites un usage intensif de listes. ☺ Si vous utilisez Google aussi, avec sa technologie `map/reduce`.

6 Manipulation mémoire et binaire

6.1 Bouger des octets

Écrire une fonction qui prend 3 paramètres et copie une zone d'octets de taille donnée à un autre endroit. Mesurer les performances pour des tailles significatives.

Faire la même chose en utilisant le type de donnée le plus gros possible pour optimiser le transfert.

Les hackers pourront utiliser des instructions en assembleur à la SSE4, ou mieux utiliser les types spéciaux et intrinsèques trouvés dans les extensions de GCC.

6.2 Bouger des bits

Faire la même chose avec des champs de bits (opération classique de type BITBLT utilisée lors des affichages d'objets graphiques dans les mémoires d'écran).

6.3 Crible d'Eratosthène

On désire trouver tous les nombres premiers entre 0 et N. L'idée est de constater qu'un nombre est premier si, et seulement si, il n'est pas divisible par tous les nombres premiers inférieurs à lui.

Écrire un programme qui utilise un tableau de N booléens pour coder le fait qu'un nombre est premier ou pas. Au départ on marque tous les nombres *a priori* premiers et on coche successivement tous les multiples de nombres premiers de plus en plus gros[7].

Écrire plusieurs versions optimisées en vitesse, stockage...

Dans la vraie vie il faudrait utiliser des bibliothèques de calcul en précision étendue...

7 Nombres flottants

7.1 Paramètres de base

Trouver la valeur maximale représentable en nombre flottant simple précision. Trouver la valeur positive minimale représentable en nombre flottant simple précision.

Comparer la vitesse de calcul de somme d'éléments dénormalisés avec des nombres « normaux ». On fera par exemple une boucle suffisamment intensive qui additionne ϵ dans une variable.

7.2 Calcul de fractales

Cette partie permet de découvrir à la fois le concept mathématique des fractales et une bibliothèque d'affichage graphique avec un générateur d'interface.

On utilisera le générateur d'interface graphique `glade` et l'ensemble de bibliothèques graphiques GTK+.

Ce sera aussi l'occasion de comparer les calculs flottants aux calculs en virgule fixe.

On fera un programme qui utilise les nombres flottants du C99 puis les nombres réels pour faire les mêmes calculs. Comparer les performances des 2 approches et argumenter.

[7] À quel nombre premier peut-on s'arrêter au fait ?

On peut faire de jolis dessins de fractales avec pas mal de procédés (études de convergence de résolution de méthode de NEWTON ou de suite sur des quaternions) mais on va se simplifier la vie en se restreignant à 2 suites sur des nombres complexes.

L'idée est d'associer le plan complexe à l'écran graphique de l'ordinateur et d'associer une couleur à une propriété d'une suite définie dans le plan complexe.

L'idée est d'étudier la convergence d'une suite mathématique. Comme c'est un problème indécidable informatiquement dans le cas général, on va regarder si au bout d'un certain temps la valeur de la suite sort d'une boule. On utilise le nombre d'itération pour sortir de cette boule pour choisir la couleur d'un point.

7.2.1 Ensemble de Mandelbrot

On étudie la divergence de la suite

$$z_{n+1} = z_n^2 + z_{x,y}$$

avec $z_0 = 0$ et $z_{x,y}$ le nombre complexe formé canoniquement à partir du pixel considéré à l'écran dans le plan complexe.

On pourra varier les plaisirs en changeant la fonction : prendre une autre puissance, du cosinus complexe, des fonctions hyperboliques, etc. On est borné par l'imagination mais cela permet de développer son côté artiste mathématicien et informaticien. ☺

7.2.2 Ensemble de Julia

On prend une fonction duale à celle de l'ensemble de MANDELBROT

$$z_{n+1} = z_n^2 + z_c$$

avec $z_0 = z_{x,y}$ le nombre complexe formé canoniquement à partir du pixel considéré à l'écran dans le plan complexe et z_c une constante qui permet d'influencer l'ensemble de JULIA.

7.2.3 La même chose en virgule fixe

Programmer des fonctions de calcul en virgule fixe pour réécrire les algorithmes précédents[8]. On pourra utiliser le mot-clé `inline`.

[8] C'est une méthode que Ronan KERYELL avait utilisé pendant sa thèse lorsqu'il développait un ordinateur parallèle qui n'avait pas encore le flottant pleinement opérationnel. L'intérêt, outre technique, est que cela donne des fractales de style « impressionniste » artistiquement très intéressantes ! ☺