TME2 : Langage C: étude de cas sur les tables de hachage

Objectif

Pour la majorité d'entre-vous, vous connaissez déjà le C, mais certains ne le connaissent que superficiellement. Nous devons essayer de mettre tout le monde au niveau, en vous faisant étudier un petit programme. L'objectif de ce programme est double :

1. Il doit d'une part vous permettre de faire une auto-évaluation de vos connaissances des outils de developpement C en vous posant des questions auxquelles vous devriez savoir répondre. Si ce n'est pas le cas, vous **devez** trouver les réponses dans les documentations (man, web), ou auprès de vos camarades.
2. Il vous offre un modèle de programme, avec makefile et man pour vos futurs développements.

Pour réaliser une application en C, vous devez savoir:

- Ecrire un programme C en respectant des conventions d'écriture.
- Compiler en plusieurs fichiers objet et en constituant une librairie.
- Décrire un makefile.
- Debugger en utilisant gdb ou xgdb.
- Faire des mesures de performances avec gprof.
- Ecrire un man sur l'outil.

Travail demandé

- Vous devez commencer par copier sur votre compte le répertoire :

```
cp -rp /users/enseig/encadr/cao/tme2 ~/cao/tme2
```

- Ce répertoire contient un programme utilisant une table de hachage.
- Le travail consiste:
    1. à répondre aux questions portant sur le code fourni. Les questions sont sur cette page. Vous rédigerez un compte rendu informatique pour vous même avec les réponses.
    2. à programmer des évolutions du programme:
- L'évaluation sera individuelle et orale au début du tme3.
- Commencez par lire le programme en entier et faîtes le tourner pour comprendre son fonctionnement.
- Répondez ensuite aux questions et faîtes les évolutions demandées.

Evolution du programme

Le programme fourni compte le nombre de mots d'un fichier texte et indique le nombre de mots présents et le

nombre de mots différents. Votre programme devra indiquer pour chaque mot:

- le nombre d'occurences
- les numéros de lignes où il est présent

Vous donnerez également des statistiques sur l'usage des tables de hachage:

- taux de remplissage.
- moyenne du nombre de comparaisons nécessaire lors de la recherche d'un mot

Description des sources fournies

- `Makefile ..... description du processus de construction de l'exécutable.`
- `main.c, main.h ..... programme principal source et déclaration.`
- `count.c, count.h ..... algorithme de parcours d'un fichier texte en vue de comptage.`
- `hte.c, hte.h ..... fonction de création des tables de hachage et déclaration de toutes fonctions de gestion.`
- `dico.c, dejavu.c, namealloc.c .. fonctions de gestion des tables pour trois types d'usage`
- `man1/tool.1 ..... fichier au format man`

Questions

Le processus de construction : Makefile

1. Completez la liste des dépendances lignes 24 à 28.
2. Réécrivez les commandes en utilisant les variables automatiques : `$@ $< $^`
    - `$@` : désigne la cible d'une règle.
    - `$<` : désigne le premier fichier de la liste des sources d'une règle.
    - `$^` : désigne la liste des sources d'une règle.
3. Donnez une raison à la definition des commandes et paramètres au début du Makefile
4. A quoi servent les options -p, -g, -wall, -werror, -ansi ?
5. Comment demander l'optimisation maximale du compilateur ?
6. L'option -p est présente dans LDFLAGS et CFLAGS, pourquoi n'est-ce pas le cas de -g ?
7. Que fait la regle indent ? quelle est la signification des flags utilisés par le programme indent ?

```
 1 # Definition des commandes
 2 CC      = gcc
 3 AR      = ar
 4 RM      = rm
 5 INDENT  = indent
 6
 7 # Definition des parametres
 8 LDFLAGS = -p
 9 CFLAGS  = -g -p -Wall -ansi -Werror
10 ARFLAGS = -r
11 IDFLAGS = -gnu  -bli0 -npsl -l90
12
13 # Definition de la liste des librairies necesaires a l'edition de lien
14 LDLIBS  = -L. -lhte
15
16 .PHONY: all clean realclean
17
18 stat : main.o count.o libhte.a
19         $(CC) $(LDFLAGS) main.o count.o -o stat $(LDLIBS)
```

```
20
21 libhte.a : hte.o dico.o dejavu.o namealloc.o
22         $(AR) $(ARFLAGS) libhte.a hte.o dico.o dejavu.o namealloc.o
23
24 main.o:
25 count.o:
26 hte.o:
27 dejavu.o:
28 namealloc.o:
29
30 all: clean stat
31
32 clean:
33         $(RM) *.o *.a *.out *~ 2> /dev/null || true
34
35 realclean: clean
36         $(RM) stat 2> /dev/null || true
37
38 indent:
39         $(INDENT) $(IDFLAGS) *.c *.h
```

le programme principal : main

- A quoi servent les lignes 1, 2 et 11 ?
- Pourquoi inclure `stdio.h` ici ?

```
 1 #ifndef _MAIN_H_
 2 #define _MAIN_H_
 3
 4 #include <stdio.h>
 5
 6 struct mainarg_s
 7 {
 8     FILE *INFILE;
 9     FILE *OUTPUTFILE;
10     char VERBOSE;
11 };
12 extern struct mainarg_s MAINARG;
13
14 #endif
```

1. A quoi sert chaque include ?
2. Pourquoi a-t-on un fichier main.h ?
3. Expliquez le fonctionnement de la fonction getopt (`man 3 getopt`)
   Ajoutez l'option -h qui affiche l'usage du programme et un petit texte de description du comportement (très court, c'est juste pour l'exercice).
   Vous ajouterez plus tard l'option -s qui demande les statistiques d'usage de la tables de hachage.
4. A quoi sert l'appel de return a la fin de la fonction main() ?
5. Pourquoi y-a-t-il exit() a la fin de la fonction usage() ?
6. Qu'est ce qu'un appel systeme, en voyez-vous dans ce fichier, si oui lesquels, citez en d'autres,
7. Quelle precaution doit on prendre lors de leur utilisation ?
8. Ou sont definiies les fonctions standards ?
9. Qu'est-ce qu'un filtre unix ?
10. Que faut-il faire pour transformer ce programme en filtre ?

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <getopt.h>
4 #include "main.h"
5 #include "count.h"
```

```
 6 #include "hte.h"
 7
 8 struct mainarg_s MAINARG;
 9
10 void usage (char *command)
11 {
12     printf ("\nStatistique Diverses\n");
13     printf ("Usage  : %s
14             [-v] [-o OutFile] InFile\n", command);
15     printf ("-v          verbose mode\n");
16     printf ("-o OutFile  output file (stdout by default)\n\n");
17     exit (EXIT_FAILURE);
18 }
19
20 void getarg (int argc, char **argv)
21 {
22     extern char *optarg;
23     extern int optind;
24     char option;
25
26     MAINARG.OUTPUTFILE = NULL;
27     while ((option = getopt (argc, argv, "vo:")) != EOF)
28         switch (option)
29         {
30         case 'v':
31             MAINARG.VERBOSE = 1;
32             fprintf (stderr, "Verbose mode\n");
33             break;
34
35         case 'o':
36             if (MAINARG.VERBOSE)
37                 fprintf (stderr, "Fichier de sortie : %s\n", optarg);
38
39             MAINARG.OUTPUTFILE = fopen (optarg, "w");
40             if (MAINARG.OUTPUTFILE == NULL)
41             {
42                 fprintf (stderr, "%s: %s: \n", argv[0], optarg);
43                 perror ("fopen");
44                 exit (EXIT_FAILURE);
45             }
46             break;
47
48         case '?':
49         default:
50             usage (argv[0]);
51         }
52     if ((optind + 1) != argc)
53     {
54         usage (argv[0]);
55     }
56     else
57     {
58         if (MAINARG.VERBOSE)
59             fprintf (stderr, "Fichier d'entrée : %s\n", argv[optind]);
60
61         MAINARG.INFILE = fopen (argv[optind], "r");
62         if (MAINARG.INFILE == NULL)
63         {
64             fprintf (stderr, "%s: %s ", argv[0], argv[optind]);
65             perror ("fopen");
66             exit (EXIT_FAILURE);
67         }
68         if (MAINARG.OUTPUTFILE == NULL)
69             MAINARG.OUTPUTFILE = stdout;
70     }
71 }
```

```
72
73 int main (int argc, char **argv)
74 {
75     getarg (argc, argv);
76     count (MAINARG.INFILE);
77     fclose (MAINARG.OUTPUTFILE);
78     return EXIT_SUCCESS;
79 }
```