IN 101 - TD 11
2 décembre 2011

Corrigé

ENSTA ParisTech

Solution 1 : Chiffrement à flot

1.a] Voici à quoi ressemble la fonction `main` qui lit des arguments à l'aide d'options `-k`, `-i` et `-o`. Les arguments peuvent être dans n'importe quel ordre sur la ligne de commande et on peut définir des arguments par défaut (`stdin` et `stdout` pour l'entrée et la sortie).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(int argc, char* argv[]) {
6    int i, key=-1;
7    char *input_name=NULL, *output_name=NULL;
8    FILE *input, *output;
9
10   for (i=1; i<argc; i++) {  // on boucle sur tous les arguments
11     if (strcmp(argv[i],"-k") == 0) {
12       i++;    // la clef vient juste après le -k
13       key = atoi(argv[i]);
14     } else if (strcmp(argv[i],"-i") == 0) {
15       i++;   // le fichier d'entrée après -i
16       input_name = argv[i];
17     }  else if (strcmp(argv[i],"-o") == 0) {
18       i++;   // le fichier de sortié après -o
19       output_name = argv[i];
20     }   // on ne fait rien avec les arguments non reconnus
21   }
22
23   if (key == -1) {    // la clef est un argument obligatoire
24     fprintf(stderr, "Utilisation:\n
25                      %s -k key [-i input_file] [-o output_file]\n", argv[0]);
26     return 1;
27   }
28   if (input_name == NULL) {
29     input = stdin;    // par défaut, on lit sur stdin
30   } else {
31     if (!(input = fopen(input_name, "r"))) {
32       fprintf(stderr, "Impossible d'ouvrir '%s' en lecture.\n", input_name);
33       return 2;
34     }
35   }
36   if (output_name == NULL) {
37     output = stdout;  // par défaut on écrit sur stdout
38   } else {
39     if (!(output = fopen(output_name, "w"))) {
40       fprintf(stderr, "Impossible d'ouvrir '%s' en écriture.\n", output_name);
```

```
41        return 2;
42      }
43    }
44
45    // ...
46
47    fclose(input);
48    fclose(output);
49    return 0;
50  }
```

1.b] La fonction de chiffrement est ensuite très simple :

```
1  void encrypt(int key, FILE* input, FILE* output) {
2    srand(key);           // on initialise avec la clef
3    char c;
4    while (1) {
5      c = fgetc(input);   // on lit un caractère en entrée
6      if (feof(input)) {  // si le fichier d'entrée est terminé
7        break;            // on arrête la boucle
8      }
9      c ^= rand() & 255;  // on chiffre le caractère
10     fputc(c, output);   // on l'écrit en sortie
11   }
12 }
```

Solution 2 : Analyse de fréquence des lettres

Pour trouver les fréquences des lettres on utilise un tableau de 256 compteurs que l'on incrémente à chaque caractère lu.

```
1  int main(int argc, char* argv[]) {
2    int i, c;
3    char *input=NULL, *output=NULL;
4    FILE *input_file, *output_file;
5    int* freqs;
6
7    for (i=1; i<argc; i++) {
8      if (strcmp(argv[i],"-i") == 0) {
9        i++; input = argv[i];
10     } else if (strcmp(argv[i],"-o") == 0) {
11       i++; output = argv[i];
12     }
13   }
14
15   if (input == NULL) {
16     input_file = stdin;
17   } else {
18     if (!(input_file = fopen(input, "r"))) {
19       fprintf(stderr, "Impossible d'ouvrir '%s' en lecture.\n", input);
20       return 2;
21     }
```

```
22   }
23
24   if (output == NULL) {
25     output_file = stdout;
26   } else {
27     if (!(output_file = fopen(output, "w"))) {
28       fprintf(stderr, "Impossible d'ouvrir '%s' en écriture.\n", output);
29       return 2;
30     }
31   }
32
33   // tableau de compteurs initialisés à 0
34   freqs = (int*) calloc(256, sizeof(int));
35   // c est un int: permet de distinguer -1 (fin de fichier) du caractère ÿ (255)
36   while ((c = fgetc(input_file)) != -1) {
37     freqs[c]++;
38   }
39   fclose(input_file);
40
41   for(i=0; i<256; i++) {
42     if (freqs[i] != 0) {  // on écrit dans le fichier de sortie
43       fprintf(output_file, "'%c' %d\n", i, freqs[i]);
44     }
45   }
46   fclose(output_file);
47   return 0;
48 }
```

Solution 3 : Codage de Huffman

3.a] Voici la fonction pour construire l'arbre :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct node_st {
5    char let;
6    double freq;
7    struct node_st *zero, *un;
8  } node;
9
10 node* build_tree(FILE* freq_file) {
11   int i,min,n;
12   char c;
13   double f;
14   node *tmp1, *tmp2;
15   node** tab = (node**) malloc(256*sizeof(node*));
16
17   // on lit les fréquences dans freq_file et on crée des nouveaux node dans tab
18   n=0;        // n est le nombre d'arbres dans le tableau
19   while(1) {  // on lit les fréquences comme double (%lf)
20     i=fscanf(freq_file, "'%c' %lf\n", &c, &f);
21     if (i == 2) { // on vérifie que fscanf a bien lu 2 éléments
22       tab[n] = (node*) malloc(sizeof(node));
```

```
23        tab[n]->let = c;  // on initialise chaque case du nouveau node
24        tab[n]->freq = f;
25        tab[n]->zero = NULL; // c'est une feuille : pas de fils
26        tab[n]->un = NULL;
27        n++;
28      } else {  // si fscanf n'a pas réussi, on arrête de lire
29        break;
30      }
31    }
32
33    // toutes les feuilles de l'arbre sont créées, reste à construire l'arbre
34    while (n > 1) {
35      min = 0;    // on extrait le "min"
36      for (i=1; i<n; i++) {
37        if (tab[i]->freq < tab[min]->freq) {
38          min = i;
39        }
40      }
41      tmp1 = tab[min];     // on bouche le trou libéré par le min
42      tab[min] = tab[n-1]; // avec le dernier élément du tableau
43      n--;
44
45      min = 0; // on extrait un deuxième "min"
46      for (i=1; i<n; i++) {
47        if (tab[i]->freq < tab[min]->freq) {
48          min = i;
49        }
50      }
51      tmp2 = tab[min];
52      tab[min] = tab[n-1];
53
54      tab[n-1] = (node*) malloc(sizeof(node));   // on fusionne les 2 "min"
55      tab[n-1]->freq = tmp1->freq + tmp2->freq;  // dans un nouvel arbre
56      tab[n-1]->zero = tmp1;
57      tab[n-1]->un = tmp2;
58    }
59    tmp1 = tab[0]; // c'est l'arbre de Huffman !
60    free(tab);     // ne jamais oublier le free
61    return tmp1;
62  }
```

3.b] Ces deux fonctions permettent maintenant de trouver les correspondances caractère/séquence en parcourant l'arbre. La première fonction récursive est appelée par la deuxième qui ne sert qu'à simplifier l'appel.

```
1  // conv_tab contient les correspondances, H est la racine du sous-arbre courant
2  // prefix est la chaîne de caractères menant au sous-arbre courant
3  // length est la longueur de prefix (pour ne pas avoir à utiliser strlen)
4  void fill_table(char** conv_tab, node* H, char* prefix, int length) {
5    if (H->zero == NULL) {
6      // on a une feuille, on écrit la correspondance
7      conv_tab[H->let] = prefix;
8    } else {
9      // on est sur un noeud interne, on lance des appels récursifs
```

```
10      char* prefix0 = (char*) malloc(length+2);
11      char* prefix1 = (char*) malloc(length+2);
12      sprintf(prefix0, "%s0", prefix);
13      sprintf(prefix1, "%s1", prefix);
14      free(prefix); // on libère prefix qui ne sert plus
15      fill_table(conv_tab, H->zero, prefix0, length+1);
16      fill_table(conv_tab, H->un, prefix1, length+1);
17    }
18  }
19
20  char** build_table(node* H) {
21    // on alloue un tableau pour 256 chaînes de caractères, NULL au départ
22    char** conv_tab = (char**) calloc(256,sizeof(char*));
23    char* vide = calloc(1, 1);      // chaîne de caractères vide (premier préfixe)
24    fill_table(conv_tab, H, vide, 0); // on lance la fonction récursive
25    return conv_tab;
26  }
```

3.c] Voici enfin le `main` qui lit les arguments et code ou décode en fonction.

```
1  int main(int argc, char* argv[]) {
2    int i, c;
3    char *freqs=NULL, *input=NULL, *output=NULL;
4    FILE *freq_file, *input_file, *output_file;
5    int decode = 0;
6
7    for (i=1; i<argc; i++) {
8      if (strcmp(argv[i],"-f") == 0) {
9        i++; freqs = argv[i];
10     } else if (strcmp(argv[i],"-i") == 0) {
11       i++; input = argv[i];
12     } else if (strcmp(argv[i],"-o") == 0) {
13       i++; output = argv[i];
14     } else if (strcmp(argv[i],"-d") == 0) {
15       decode = 1;
16     }
17   }
18
19   if (freqs == NULL) {
20     fprintf(stderr, "Utilisation:\n
21     %s [-d] -f frequences [-i input_file] [-o output file]\n", argv[0]);
22     return 1;
23   } else {
24     freq_file = fopen(freqs, "r");
25     if (!freq_file) {
26       fprintf(stderr, "Impossible d'ouvrir '%s' en lecture.\n", freqs);
27       return 2;
28     }
29   }
30   if (input == NULL) {
31     input_file = stdin;
32   } else {
33     input_file = fopen(input, "r");
34     if (!input_file) {
```

```
35        fprintf(stderr, "Impossible d'ouvrir '%s' en lecture.\n", input);
36        return 2;
37      }
38    }
39    if (output == NULL) {
40      output_file = stdout;
41    } else {
42      output_file = fopen(output, "w");
43      if (!output_file) {
44        fprintf(stderr, "Impossible d'ouvrir '%s' en écriture.\n", output);
45        return 2;
46      }
47    }
48
49    // on construit l'arbre de Huffman
50    node* Huff = build_tree(freq_file);
51
52    if (decode) { // on décode
53      node* cur;
54      // on lit l'input catractère par caractère en descendant dans l'arbre
55      cur = Huff; // on part de la racine
56      while ((c=fgetc(input_file)) != -1) {
57        if (c == '0') {
58          cur = cur->zero;
59        } else {
60          cur = cur->un;
61        }
62        // on teste si on est dans une feuille
63        if (cur->zero == NULL) { // si oui, on écrit le caractère correspondant
64          fputc(cur->let, output_file);
65          cur = Huff;
66        }
67      }
68    } else {  // on code
69      // on construit le table de correspondance caractère -> chaîne binaire
70      char** conv_tab = build_table(Huff);
71      while (1) {
72        c = fgetc(input_file);
73        if (feof(input_file)) {
74          break;
75        }
76        fprintf(output_file, "%s", conv_tab[c]);
77      }
78    }
79    fclose(freq_file);
80    fclose(input_file);
81    fclose(output_file);
82    return 0;
83  }
```