

CS221 - TP2

Vincent MOUCADEAU - Rémi MAZZONE — 2A

30/11/2022

Table des matières

1	Introduction	2
2	Travail Préparatoire	2
2.1	Nombres impairs	2
2.2	Comptage de voyelles	2
2.3	Fichier transporteur into source	3
2.4	Makefile	3
3	Stéganographie	4
3.1	Une application un peu trop visible...	4
3.2	Une application un peu moins visible...	5
4	Conclusion	8

Listings

2.1	Code pour avoir les 100 premiers nombres impairs	2
2.2	Code pour compter le nombre de voyelles dans un fichier texte	2
2.3	Algorithme en français de ce processus	3
2.4	Makefile du projet	3
3.1	Code décodage fichier txt	4
3.2	Code pour lire un champ de la structure bitmap	5
3.3	Code pour décoder une image	5
3.4	Fonctions annexes	6
3.5	Encodage d'un jpg secret dans une image bmp	7
3.6	Fonction <code>set_bit_faible</code>	8
4.1	Fonction main	8

1 Introduction

Dans ce TP, nous allons travailler sur les fichiers et les applications qu'on peut y trouver dans le domaine de la stéganographie.

2 Travail Préparatoire

2.1 Nombres impairs

Listing 2.1: Code pour avoir les 100 premiers nombres impairs

```
1 void file_odd(int n) {
2     int i = 0, c = 0;
3     FILE *f = fopen("nombres_impairs.txt",
4         "w");
5     while (c < n) {
6         if (i % 2 != 0) {
7             fprintf(f, "%d\n", i);
8             c++;
9         }
10        i++;
11    }
12    fclose(f);
13 }
```

2.2 Comptage de voyelles

Listing 2.2: Code pour compter le nombre de voyelles dans un fichier texte

```
1 int file_count_vowels(char *filename) {
2     int count = 0;
3     char vowels[6] = {'a', 'e', 'i', 'o', 'u', 'y'};
4     int tab_len = sizeof(vowels)/sizeof(vowels[0]);
5     FILE *file = fopen(filename, "r");
6     char curr_char;
7
8     while (curr_char != EOF) {
9         curr_char = fgetc(file);
10        for(int i = 0; i < tab_len; i++) {
11            if(curr_char == vowels[i]) {
12                count++;
13                i = tab_len;
14            }
15        }
16    }
17    fclose(file);
18 }
```

```

18
19     return count;
20 }

```

2.3 Fichier transporteur into source

Listing 2.3: Algorithme en français de ce processus

```

1  input : fichier f_input
2  output : fichier f_output
3  ouvrir f_input
4  octet = 0
5  nouveau_char = ""
6  tant que f_input pas termine
7      si octet = 8
8          nouveau_char <- conversion bin to char de
              nouveau_char
9          ecrire nouveau_char dans f_output
10     sinon
11         evaluer le prochain caractere de f_input
12         si il est alphanumerique
13             si c'est une majuscule
14                 nouveau_char <- nouveau_char + '1'
15             sinon
16                 nouveau_char <- nouveau_char + '0'
17             fin si
18             octet++
19         fin
20     fin si
21 fin tant que

```

2.4 Makefile

Listing 2.4: Makefile du projet

```

1  main: main.o
2      gcc -o $@ $^ -g
3
4  main.o: main.c
5      gcc -c $< -g
6
7
8  clean:
9      rm -rf *.o main

```

3 Stéganographie

3.1 Une application un peu trop visible...

À partir du code rédigé en pseudo code dans la partie préparatoire (cf. 2.3), nous avons pu écrire le code correspondant en C. Le voici :

Listing 3.1: Code décodage fichier txt

```
1 void readable_txt(char *filename) {
2     FILE *f = fopen(filename, "r");
3
4     char *main = malloc(strlen(filename) + strlen("_source.txt") + 1);
5     char *second = malloc(strlen("_source.txt") + 1);
6     strcpy(main, filename);
7     strcpy(second, "_source.txt");
8     FILE *fout = fopen(strcat(main, second), "w");
9
10    char new_char[8];
11    int octet = 0;
12    int curr_char = 0;
13    while (curr_char != EOF) {
14        if(octet == 8) {
15            fputc(bit_to_int(new_char), fout);
16            octet = 0;
17        } else {
18            curr_char = fgetc(f);
19            if(isalpha(curr_char) != 0) {
20                if(isupper(curr_char)) {
21                    new_char[octet] = 1;
22                } else {
23                    new_char[octet] = 0;
24                }
25                octet++;
26            }
27        }
28    }
29    fclose(f);
30    fclose(fout);
31 }
```

Code code va donc parcourir le fichier transporteur et va écrire dans un nouveau fichier le message caché. Pour cela, il va lire le fichier transporteur caractère par caractère. Si le caractère lu est une lettre, il va regarder si c'est une majuscule ou une minuscule. Si c'est une majuscule, il va écrire un 1 dans un tableau, sinon il va écrire un 0. Il va continuer à faire cela jusqu'à ce qu'il ait lu 8 caractères (ce qui correspond donc à 1 octet car on aura inséré 8 bits dans le tableau). Ensuite, il va convertir la chaîne de caractères en un entier et l'écrire dans le fichier de sortie. Il va continuer à faire cela jusqu'à la fin du fichier principal et on aura donc fini de décoder le fichier !

3.2 Une application un peu moins visible...

On a donc 3 champs de structures dans le fichier bitmap.h : fichierEntete, imageEntete et couleur-Palette. Pour les lire et les afficher, on utilise le code suivant (l'exemple ne montre la procédure que pour un champ avec l'affiche de seulement quelques informations mais c'est la même chose pour les autres) :

Listing 3.2: Code pour lire un champ de la structure bitmap

```
1 FILE *f = fopen(filename, "r");
2 fichierEntete *header = malloc(sizeof(fichierEntete));
3 fread(header, 1, sizeof(fichierEntete), f);
4 printf("%d\n", header->tailleFichier);
5 printf("%d\n", header->offset);
6 fclose(f);
```

En partant de ces informations, on a donc pu imaginer le code qui nous sera nécessaire à décoder l'image. On va donc parcourir tous les pixels de l'image source (en commençant après tous les entêtes d'un fichier Bitmap) et on va en extraire le bit de poids faible à chaque fois. Ainsi, on va pouvoir former petit à petit l'image qui avait été cachée.

Listing 3.3: Code pour décoder une image

```
1 char *decode_bmp(char *filename) {
2 FILE *f = fopen(filename, "r");
3
4 char *main = malloc(strlen(filename) + strlen("_source.jpg") + 1);
5 char *second = malloc(strlen("_source.jpg") + 1);
6 strcpy(main, filename);
7 strcpy(second, "_source.jpg");
8 FILE *fout = fopen(strcat(main, second), "w");
9
10 fichierEntete *header = malloc(sizeof(fichierEntete));
11 imageEntete *imageHeader = malloc(sizeof(imageEntete));
12
13 fread(header, 1, sizeof(fichierEntete), f);
14 fread(imageHeader, 1, sizeof(imageEntete), f);
15 fseek(f, header->offset, SEEK_SET);
16
17 char pixel, bit[8];
18 int octet = 0;
19 for(int i = 0; i < imageHeader->tailleImage; i++) {
20     pixel = fgetc(f);
21     bit[octet] = get_bit_faible(pixel);
22     octet++;
23     if(octet == 8) {
24         fputc(bit_to_int(bit), fout);
25         octet = 0;
26     }
27 }
28
29 fclose(f);
```

```

30 fclose(fout);
31
32 printf("File decoded succefully\n");
33
34 return main;
35 }

```

Ce code utilise quelques fonctions annexes que nous avons codés afin de rendre cette partie plus compréhensible. Voici donc les extraits des fonctions nécessaires :

Listing 3.4: Fonctions annexes

```

1  char get_bit_faible(char octet) {
2      char mask = 1;
3      return mask & octet;
4  }
5
6  int puissance(int a, int b) {
7      int res = 1;
8      for(int i = 0; i < b; i++) {
9          res *= a;
10     }
11     return res;
12 }
13 int bit_to_int(char bit[8]) {
14     int tot = 0;
15     for(int i = 0; i < 8; i++) {
16         if(bit[i] == 1) {
17             tot += puissance(2, 7-i);
18         }
19     }
20     return tot;
21 }

```

À la suite de l'exécution de cette fonction, nous avons donc obtenu l'image suivante :



Figure 1: L'image cachée

En faisant une rapide recherche d'image inversée sur Google, on a découvert qu'il s'agit en fait de Jonathan Bruce Postel, dit Jon Postel, un célèbre informaticien qui a contribué à la création d'Internet.

Maintenant que nous avons décodé l'image dans la question précédente, nous allons faire la procédure inverse : nous allons recréer l'image que nous venons de décoder. Dans la question précédente, nous avons pris l'image transporteur.bmp et nous en avons décodé l'image jon_postel.jpg. Désormais, nous allons nous servir de l'image originel.bmp dans laquelle nous allons encoder l'image jon_postel.jpg pour reconstituer le fichier transporteur.bmp.

Voici le code correspondant

Listing 3.5: Encodage d'un jpg secret dans une image bmp

```
1 FILE *fsource = fopen(file_source, "r");
2 FILE *fsecret = fopen(file_secret, "r"); // file extrait de
   transporteur.bmp avec decode_bmp() -> transporteur.bmp_source.
   jpg
3
4 char *remake = malloc(strlen(file_secret) + strlen("_remake.bmp")
   + 1);
5 char *second = malloc(strlen("_remake.bmp") + 1);
6 strcpy(remake, file_secret);
7 strcpy(second, "_remake.bmp");
8 FILE *ftransp = fopen(strcat(remake, second), "w"); // nouveau
   fichier
9
10 char temp = fgetc(fsecret);
11
12 fichierEntete *header = malloc(sizeof(fichierEntete));
13 imageEntete *imageHeader = malloc(sizeof(imageEntete));
14
15 fread(header, 1, sizeof(fichierEntete), fsource); // c'est l'
   inverse entre le 1 et le sizeof !!
16 fwrite(header, 1, sizeof(fichierEntete), ftransp);
17
18 fread(imageHeader, 1, sizeof(imageEntete), fsource);
19 fwrite(imageHeader, 1, sizeof(imageEntete), ftransp);
20
21 int new_bmp_size = 0;
22 char octet_source;
23 int bin_secret[8];
24
25 unsigned char char_secret = 0;
26 char new_octet_in_src = 0;
27
28 //Get file length
29 fseek(fsecret, 0, SEEK_END);
30 long fileLen = ftell(fsecret);
31 fseek(fsecret, 0, SEEK_SET);
32
33 for(int counter = 0; counter < fileLen; counter += 1) {
```

```

34     char_secret = fgetc(fsecret);
35     int_to_bin(char_secret, bin_secret);
36     for(int i = 0; i < 8; i++) {
37         octet_source = fgetc(fsource);
38         new_octet_in_src = set_bit_faible(octet_source, bin_secret
39             [7-i]);
40         fputc(new_octet_in_src, ftransp);
41     }
42     for(int i = 0; i < 8; i++) {
43         bin_secret[i] = 0;
44     }
45 }
46 fclose(fsource);
47 fclose(fsecret);
48 fclose(ftransp);
49
50 printf("Secret file '%s' successfully encoded in '%s' with '%s' as
51     original file\n\n", file_secret, remake, file_source);
52
53 return remake;

```

Listing 3.6: Fonction set_bit_faible

```

1 char set_bit_faible(char octet, char bit) {
2     char mask = 1;
3     if(bit == 1) {
4         return octet | mask;
5     } else {
6         return octet & ~mask;
7     }
8 }

```

On a donc bien réussi à faire ce que l'on voulait et pour vérifier, on peut se réserver de la fonction de décodage pour décodé le fichier que l'on vient d'encoder et vérifier que l'on obtient bien l'image secrète !

4 Conclusion

Nous avons donc terminé ce TP, voici le code inséré dans la fonction main pour faire toutes les actions d'un coup :

Listing 4.1: Fonction main

```

1 file_odd(100);
2
3 char *filename;
4 printf("Fichier voyelles : vowels.txt\n");
5 int counter = file_count_vowels("./vowels.txt");

```



```

6  printf("Le nombre de voyelle dans le fichier est : %d\n\n",
    counter);
7
8  readable_txt("utils/transporteur.txt");
9
10 char *secret_file = decode_bmp("utils/transporteur.bmp");
11 char *file_encode = encode_file("utils/originel.bmp", secret_file)
    ;
12
13 decode_bmp(file_encode);

```

Nous avons beaucoup apprécié faire ce TP qui nous a appris beaucoup de techniques de stéganographie. Il a été très enrichissant car nous n'avons pas l'habitude de manipuler les fichiers à ce point en lisant seulement certains octet afin de créer ou des nouveaux fichiers encodés ou décodés par exemple donc c'était une bonne expérience !

Nous pouvons donc maintenant nous envoyer des images secrètes avec nos propres fonctions d'encodage et ça, c'est cool !