/\*

Name :- Sk Sahil

Date :- 14/8/2022

Description :- Project **Steganography (encode.c)**

**\*/**

#include <stdio.h>

#include "encode.h"

#include "types.h"

#include <string.h>

#include "common.h"

/\* Function definition for check operation type \*/

OperationType check\_operation\_type(char \*argv[])

{

if (strcmp(argv[1], "-e") == 0)

return e\_encode;

if (strcmp(argv[1], "-d") == 0)

return e\_decode;

else

return e\_unsupported;

}

/\* Function Definitions \*/

/\* Get image size

\* Input: Image file ptr

\* Output: width \* height \* bytes per pixel (3 in our case)

\* Description: In BMP Image, width is stored in offset 18,

\* and height after that. size is 4 bytes

\*/

uint get\_image\_size\_for\_bmp(FILE \*fptr\_image)

{

uint width, height;

// Seek to 18th byte

fseek(fptr\_image, 18, SEEK\_SET);

// Read the width (an int)

fread(&width, sizeof(int), 1, fptr\_image);

printf("width = %u\n", width);

// Read the height (an int)

fread(&height, sizeof(int), 1, fptr\_image);

printf("height = %u\n", height);

// Return image capacity

return width \* height \* 3;

}

/\*

\* Get File pointers for i/p and o/p files

\* Inputs: Src Image file, Secret file and

\* Stego Image file

\* Output: FILE pointer for above files

\* Return Value: e\_success or e\_failure, on file errors

\*/

Status open\_files(EncodeInfo \*encInfo)

{

// Src Image file

encInfo->fptr\_src\_image = fopen(encInfo->src\_image\_fname, "r");

// Do Error handling

if (encInfo->fptr\_src\_image == NULL)

{

perror("fopen");

fprintf(stderr, "ERROR: Unable to open file %s\n", encInfo->src\_image\_fname);

return e\_failure;

}

// Secret file

encInfo->fptr\_secret = fopen(encInfo->secret\_fname, "r");

// Do Error handling

if (encInfo->fptr\_secret == NULL)

{

perror("fopen");

fprintf(stderr, "ERROR: Unable to open file %s\n", encInfo->secret\_fname);

return e\_failure;

}

// Stego Image file

encInfo->fptr\_stego\_image = fopen(encInfo->stego\_image\_fname, "w");

// Do Error handling

if (encInfo->fptr\_stego\_image == NULL)

{

perror("fopen");

fprintf(stderr, "ERROR: Unable to open file %s\n", encInfo->stego\_image\_fname);

return e\_failure;

}

// No failure return e\_success

return e\_success;

}

// Function definition for read and validate encode args

Status read\_and\_validate\_encode\_args(char \*argv[], EncodeInfo \*encInfo)

{

// If condition to check argv[2] is .bmp or not

if (strcmp(strstr(argv[2], "."), ".bmp") == 0)

{

encInfo->src\_image\_fname = argv[2];

}

else

return e\_failure;

if (strcmp(strstr(argv[3], "."), ".txt") == 0)

{

encInfo->secret\_fname = argv[3];

}

else

return e\_failure;

if (argv[4] != NULL)

encInfo->stego\_image\_fname = argv[4];

else

encInfo->stego\_image\_fname = "stego.bmp";

return e\_success;

}

// Function definition for check capacity

Status check\_capacity(EncodeInfo \*encInfo)

{

encInfo->image\_capacity = get\_image\_size\_for\_bmp(encInfo->fptr\_src\_image);

encInfo->size\_secret\_file = get\_file\_size(encInfo->fptr\_secret);

if (encInfo->image\_capacity > ((strlen(MAGIC\_STRING) + 4 + 4 + 4 + encInfo->size\_secret\_file) \* 8))

return e\_success;

else

return e\_failure;

}

// Function definition for getting file size

long get\_file\_size(FILE \*fptr)

{

fseek(fptr, 0, SEEK\_END);

return ftell(fptr);

}

// Function definition for copying 1st 54 bytes header file

Status copy\_bmp\_header(FILE \*fptr\_src\_image, FILE \*fptr\_dest\_image)

{

char str[54];

// Setting pointer to point to 0th position

fseek(fptr\_src\_image, 0, SEEK\_SET);

// Reading 54 bytes from beautiful.bmp

fread(str, 54, 1, fptr\_src\_image);

// Writing 54 bytes to str

fwrite(str, 54, 1, fptr\_dest\_image);

return e\_success;

}

// Function definition for encoding magic string

Status encode\_magic\_string(char \*magic\_string, EncodeInfo \*encInfo)

{

encode\_data\_to\_image(magic\_string, 2, encInfo->fptr\_src\_image, encInfo->fptr\_stego\_image, encInfo);

return e\_success;

}

// Function definition for Encode data to image

Status encode\_data\_to\_image(char \*data, int size, FILE \*fptr\_src\_image, FILE \*fptr\_stego\_image, EncodeInfo \*encInfo)

{

for (int i = 0; i < size; i++)

{

fread(encInfo->image\_data, 8, 1, fptr\_src\_image);

encode\_byte\_to\_lsb(data[i], encInfo->image\_data);

fwrite(encInfo->image\_data, 8, 1, fptr\_stego\_image);

}

}

// Function definition for encode byte to lsb

Status encode\_byte\_to\_lsb(char data, char \*image\_buffer)

{

unsigned int mask = 0x80, i;

for (i = 0; i < 8; i++)

{

image\_buffer[i] = (image\_buffer[i] & 0xFE) | ((data & mask) >> (7 - i));

mask = mask >> 1;

}

}

// Function definition for encode secret file extn size

Status encode\_secret\_file\_extn\_size(int size, FILE \*fptr\_src\_image, FILE \*fptr\_stego\_image)

{

char str[32];

fread (str, 32, 1, fptr\_src\_image);

encode\_size\_to\_lsb(size, str);

fwrite(str, 32, 1, fptr\_stego\_image);

return e\_success;

}

// Function definition to encode size to lsb

Status encode\_size\_to\_lsb(int size, char \*image\_buffer)

{

unsigned int mask = 1 << 31, i;

for (i = 0; i < 32; i++)

{

image\_buffer[i] = (image\_buffer[i] & 0xFE) | ((size & mask) >> (31 - i));

mask = mask >> 1;

}

}

// Function definition to encode secret file extn

Status encode\_secret\_file\_extn(char \*file\_extn, EncodeInfo \*encInfo)

{

encode\_data\_to\_image(file\_extn, strlen(file\_extn), encInfo->fptr\_src\_image, encInfo->fptr\_stego\_image, encInfo);

return e\_success;

}

// Function definition for encode secret file size

Status encode\_secret\_file\_size(int size, EncodeInfo \*encInfo)

{

char str[32];

fread (str, 32, 1, encInfo->fptr\_src\_image);

encode\_size\_to\_lsb(size, str);

fwrite(str, 32, 1, encInfo->fptr\_stego\_image);

return e\_success;

}

// Function definition for encode secret file data

Status encode\_secret\_file\_data(EncodeInfo \*encInfo)

{

fseek(encInfo->fptr\_secret, 0, SEEK\_SET);

char str[encInfo->size\_secret\_file];

fread(str, encInfo->size\_secret\_file, 1, encInfo->fptr\_secret);

encode\_data\_to\_image(str, strlen(str), encInfo->fptr\_src\_image, encInfo->fptr\_stego\_image, encInfo);

return e\_success;

}

// Function definition for copying remaining data as it is

Status copy\_remaining\_img\_data(FILE \*fptr\_src, FILE \*fptr\_dest)

{

char ch;

while ((fread(&ch, 1, 1, fptr\_src)) > 0)

{

fwrite(&ch, 1, 1, fptr\_dest);

}

return e\_success;

}

// Function definition for do encoding called in main function

Status do\_encoding(EncodeInfo \*encInfo)

{

if (open\_files(encInfo) == e\_success)

{

printf("Open files is a success\n");

if (check\_capacity(encInfo) == e\_success)

{

printf("Check capacity is a success\n");

if (copy\_bmp\_header(encInfo->fptr\_src\_image, encInfo->fptr\_stego\_image) == e\_success)

{

printf("Copied bmp header successfully\n");

if (encode\_magic\_string(MAGIC\_STRING, encInfo) == e\_success)

{

printf("Encoded magic string successfully\n");

strcpy(encInfo->extn\_secret\_file, strstr(encInfo->secret\_fname, "."));

if (encode\_secret\_file\_extn\_size(strlen(encInfo->extn\_secret\_file), encInfo->fptr\_src\_image, encInfo->fptr\_stego\_image) == e\_success)

{

printf("Encoded secret file extn size\n");

if (encode\_secret\_file\_extn(encInfo->extn\_secret\_file, encInfo) == e\_success)

{

printf("Encoded secret file extn\n");

if (encode\_secret\_file\_size(encInfo->size\_secret\_file, encInfo) == e\_success)

{

printf("Encoded secret file size\n");

if (encode\_secret\_file\_data(encInfo) == e\_success)

{

printf("Encoded secret file data\n");

if (copy\_remaining\_img\_data(encInfo->fptr\_src\_image, encInfo->fptr\_stego\_image) == e\_success)

{

printf("Copied remaining data\n");

}

else

{

printf("Failed to copy remaining data\n");

return e\_failure;

}

}

else

{

printf("Failed to encode secret file data\n");

return e\_failure;

}

}

else

{

printf("Failed to encode secret file size\n");

return e\_failure;

}

}

else

{

printf("Failed to encode secret file extn\n");

return e\_failure;

}

}

else

{

printf("Failed to encoded secret file extn size\n");

return e\_failure;

}

}

else

{

printf("Failed to encode magic string\n");

return e\_failure;

}

}

else

{

printf("Failed to copy bmp header\n");

return e\_failure;

}

}

else

{

printf("Check capacity is a failure\n");

return e\_failure;

}

}

else

{

printf("Open files is a failure\n");

return e\_failure;

}

return e\_success;

}