

CSCI291 Lab 4

PROGRAMMING FOR ENGINEERS LAB

Name: Ahmed Ismail | ID: 8541747 | Course Title: CSCI291 | Date: 29/11/2024



UNIVERSITY
OF WOLLONGONG
IN DUBAI

Contents

Question 1: File Processing	2
Algorithm Description	2
Pseudo-code	2
Implementation.....	3
Error Handling	3
File Handling	3
Testing Evidence	3
Test Inputs.....	3
Test Results.....	4
Question 2: Steganography using 4-bit LSB algorithm	5
embedding process:	5
Extraction Process:.....	5
Memory Management.....	5
File Processing.....	5
Bitwise Operations	5
Error Handling	6
Pseudo-code	6
Test Evidence	7
Cover Image: "baboon.pgm"	7
Secret Image: "farm.pgm"	7
Stego Image: "stego_image_bin.pgm"	8
Extracted Secret Image: "extracted_secret.pgm"	8
Question 3: C++ Basic Programming	8
Purpose:	8
Key Formatting Manipulators Used:	9
Parameters:.....	9
Test Evidence:.....	9
GitHub Page	9

Question 1: File Processing

ALGORITHM DESCRIPTION

The algorithm is designed to:

1. Read a text file with an unknown number of float values
2. Handle potential data corruption (alphabetical characters mixed with floats)
3. Create a new file with only valid float values
4. Count and display the number of invalid values

PSEUDO-CODE

FUNCTION main():

Initialize input and output file pointers

Initialize invalid count to 0

Initialize line buffer

Open input file (data.txt)

IF file not opened successfully THEN

Display error message

Exit program

Open output file (dataOut.txt)

IF output file not created successfully THEN

Close input file

Exit program

WHILE not end of input file DO

Read a line from input file

FOR each token in line DO

IF token can be converted to float THEN

Write float to output file

ELSE

Increment invalid count

END IF

END FOR

END WHILE

Check for hardware processing errors

Close both files

Display total invalid values count

Exit program

IMPLEMENTATION

1. Used `fgets()` to read lines, handling potential line-based input
2. Utilized `strtok()` to split lines into tokens
3. Implemented `sscanf()` for float conversion and validation
4. Employed error handling for file operations
5. Tracked invalid value count

Error Handling

- Checked file opening success
- Implemented hardware failure detection using `ferror()`

File Handling

- Input file: "data.txt" (read mode)
- Output file: "dataOut.txt" (write mode)

TESTING EVIDENCE

Test Inputs

- Source file: data.txt

```
data.txt X C Lab4_Q1.c
data.txt
1 A 10.0 12.0 14.0 B 20.0 22.0 24.0 C 30.0 32.0 34.0 D 40.0 42.0 44.7 E
```

Figure 1 Data file for input

Test Results

- Total Invalid Values: 5

```
printf("\n\nTotal invalid values found: %d\n", totalInvalidValues);
```

Figure 2 Terminal view after running program

- Output file: dataOut.txt

```
data.txt C Lab4_Q1.c dataOut.txt X
dataOut.txt
1 10.000000
2 12.000000
3 14.000000
4 20.000000
5 22.000000
6 24.000000
7 32.000000
8 34.000000
9 40.000000
10 42.000000
11 44.700001
12
```

Figure 3 Output data

Question 2: Steganography using 4-bit LSB algorithm

EMBEDDING PROCESS:

1. Take the 4 most significant bits of the cover image pixel
2. Take the 4 most significant bits of the secret image pixel
3. Replace the 4 least significant bits of the cover image pixel with the secret image's 4 most significant bits

EXTRACTION PROCESS:

1. Extract the 4 least significant bits of the stego image
2. Shift these bits left by 4 positions to reconstruct the secret image pixel
3. Set the 4 least significant bits to zero

MEMORY MANAGEMENT

- Used malloc() for dynamic memory allocation
- Allocated memory for:
 - Cover image pixels
 - Secret image pixels
 - Stego image pixels
 - Extracted secret image pixels
- Ensured proper memory deallocation using free()

FILE PROCESSING

- Read input images in text PGM format using readPGMText()
- Wrote stego image in binary PGM format using writePGMBinary()
- Wrote extracted secret image in text PGM format using writePGMText()

BITWISE OPERATIONS

- Used bitwise AND (&) to clear/preserve bits
- Used bitwise OR (|) to combine pixel values
- Used bitwise shift (>>, <<) to manipulate bit positions

ERROR HANDLING

- Checked for:
 - Memory allocation failures
 - File open/read/write errors
 - Matching image dimensions
- Used perror() to provide detailed error messages
- Implemented proper error return values for functions

PSEUDO-CODE

Function embedLSB(coverPixels, secretPixels, width, height):

FOR each pixel i from 0 to (width * height - 1):

1. Clear 4 least significant bits of cover image pixel

$\text{coverPixels}[i] = \text{coverPixels}[i] \& 0xF0$

2. Extract 4 most significant bits of secret image pixel

$\text{secretNibble} = (\text{secretPixels}[i] \& 0xF0) \gg 4$

3. Combine cover and secret pixel

$\text{coverPixels}[i] = \text{coverPixels}[i] | \text{secretNibble}$

Function extractLSB(coverPixels, outputPixels, width, height):

FOR each pixel i from 0 to (width * height - 1):

1. Extract 4 least significant bits and shift left

$\text{secretNibble} = (\text{coverPixels}[i] \& 0x0F) \ll 4$

2. Store extracted pixel with zero least significant bits

$\text{outputPixels}[i] = \text{secretNibble}$

TEST EVIDENCE

Cover Image: "baboon.pgm"

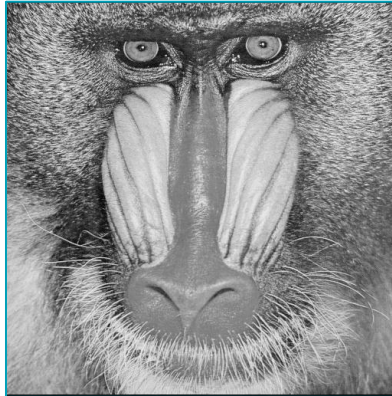


Figure 4 Cover image

Secret Image: "farm.pgm"

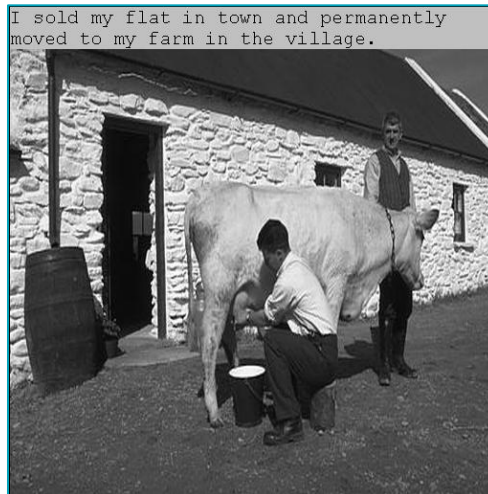


Figure 5 Secret Image to be embeded

Stego Image: "stego_image_bin.pgm"

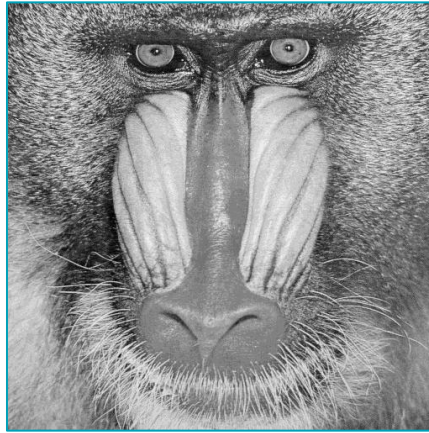


Figure 6 Stego Image

Extracted Secret Image: "extracted_secret.pgm"

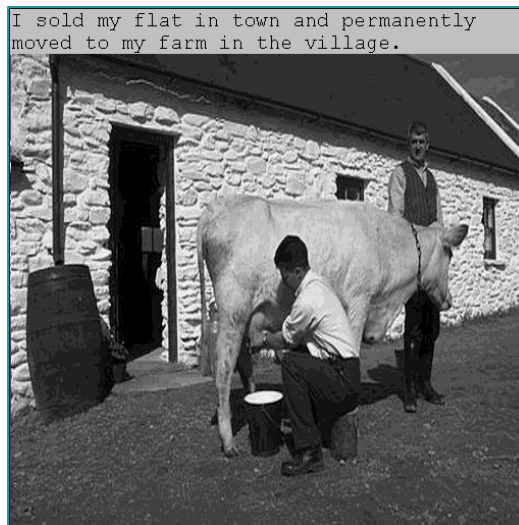


Figure 7 Extracted secret image from stego

Question 3: C++ Basic Programming

Purpose:

The `print_real()` function is designed to print a floating-point number with specific formatting requirements:

- Print the number in fixed-point notation
- Use a specified output field width

- Control the number of decimal places
- End with a new line

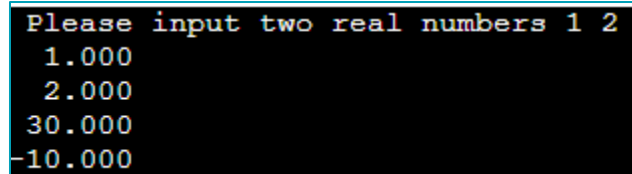
Key Formatting Manipulators Used:

1. `fixed`:
 - Forces the output to use fixed-point notation
 - Prevents scientific notation
2. `setprecision(precision)`:
 - Sets the number of digits to be displayed after the decimal point
3. `setw(fieldspace)`:
 - Sets the total width of the output field
 - Right-aligns the number within the specified field width

Parameters:

- `number`: The floating-point value to be printed
- `fieldspace`: Total width of the output field
- `precision`: Number of decimal places to display

Test Evidence:



```
Please input two real numbers 1 2
1.000
2.000
30.000
-10.000
```

Figure 8 Results for testing 1 and 2 input

GitHub Page

https://github.com/AIsmail17/Lab4_CSCI291.git