Computer Network Laboratory

Assignment 7

Name: Ajay N
Link : https://github.com/Ajayneethikannan/CSN-361-Assignment-7
Enrollment Number: 17118007
Class: 3rd year, B.Tech CSE

Course: CSN-361

Three problems were given for this assignment. They are,

Question 1 :

Transmit a binary message (from a sender to a receiver) using socket programming in C and report whether the received msg is correct or not; using the following error detection algorithms:
1. Single Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)

Algorithms used :

1. Simple Parity check

Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :

=>1 is added to the block if it contains odd number of 1's, and

=>0 is added if it contains even number of 1's

2. Two-dimensional Parity check

Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns, then both are sent along with the data. At the receiving end these are compared with the parity bits calculated on the received data.

3. Checksum

1. In checksum error detection scheme, the data is divided into k segments each of m bits.
2. In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum.
3. The checksum segment is sent along with the data segments.
4. At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented. If the result is zero, the received data is accepted; otherwise discarded.

4. Cyclic redundancy check (CRC)

1. Unlike checksum scheme, which is based on addition, CRC is based on binary division.
2. In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number
3. At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
4. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

Data Structures used :

Socket creation:

sockfd: socket descriptor, an integer

struct sockaddr_in : structure to store internet addresses like IP address, port.

Bind:

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

bind function binds the socket to the address and port number specified in addr.

Accept:

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket,

sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.

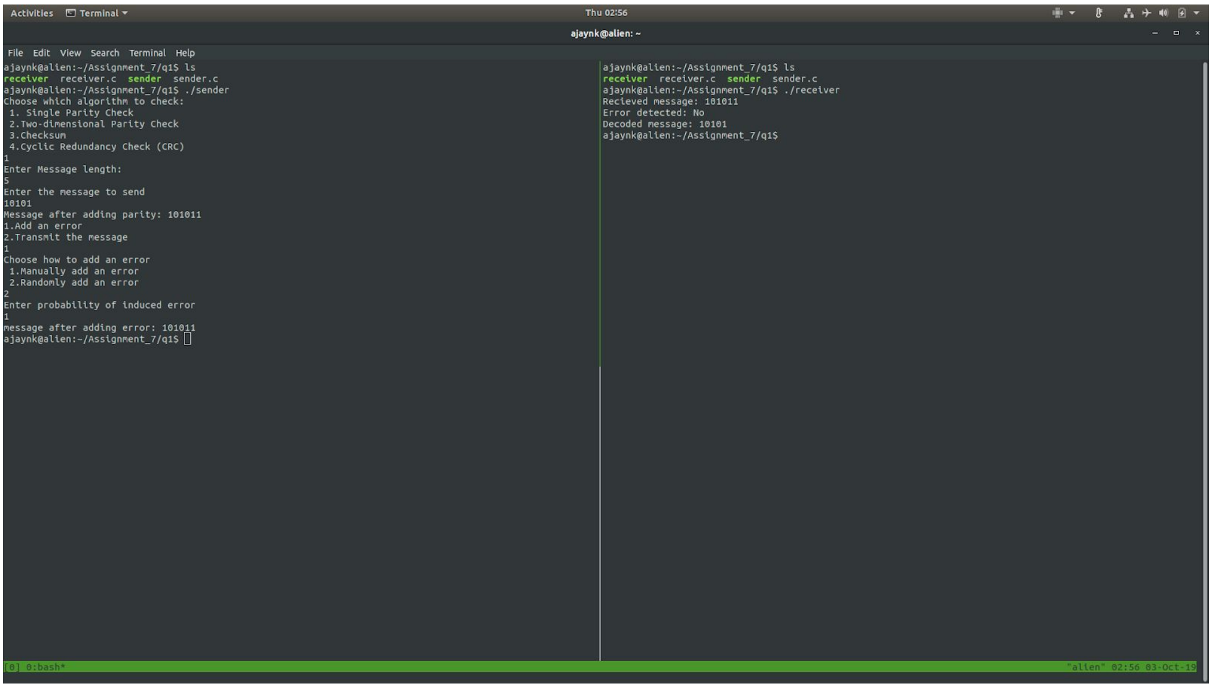
Socket connection:

same as that of server's socket creation

Connect:

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

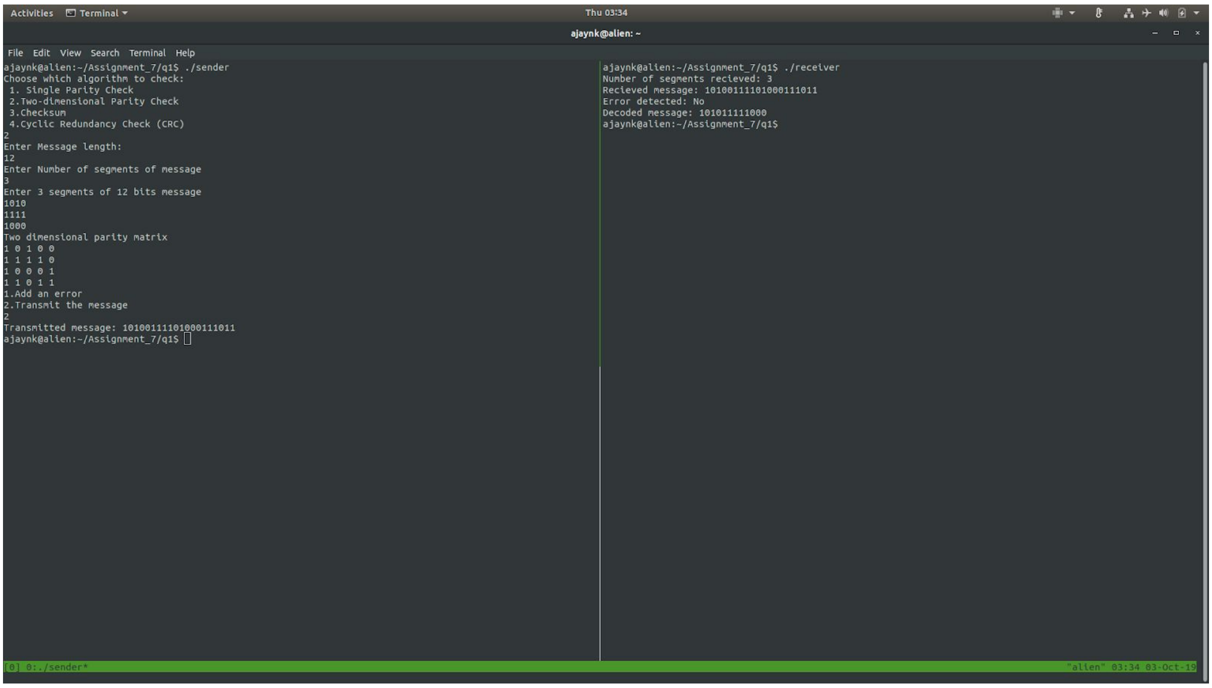
Screenshots :



```
ajaynk@alien:~/Assignment_7/q1$ ls
receiver  receiver.c  sender  sender.c
ajaynk@alien:~/Assignment_7/q1$ ./sender
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
1
Enter Message length:
5
Enter the message to send
10101
Message after adding parity: 101011
1.Add an error
2.Transmit the message
1
Choose how to add an error
 1.Manually add an error
 2.Randomly add an error
2
Enter probability of induced error
1
message after adding error: 101011
ajaynk@alien:~/Assignment_7/q1$
```

```
ajaynk@alien:~/Assignment_7/q1$ ls
receiver  receiver.c  sender  sender.c
ajaynk@alien:~/Assignment_7/q1$ ./receiver
Recieved message: 101011
Error detected: No
Decoded message: 10101
ajaynk@alien:~/Assignment_7/q1$
```

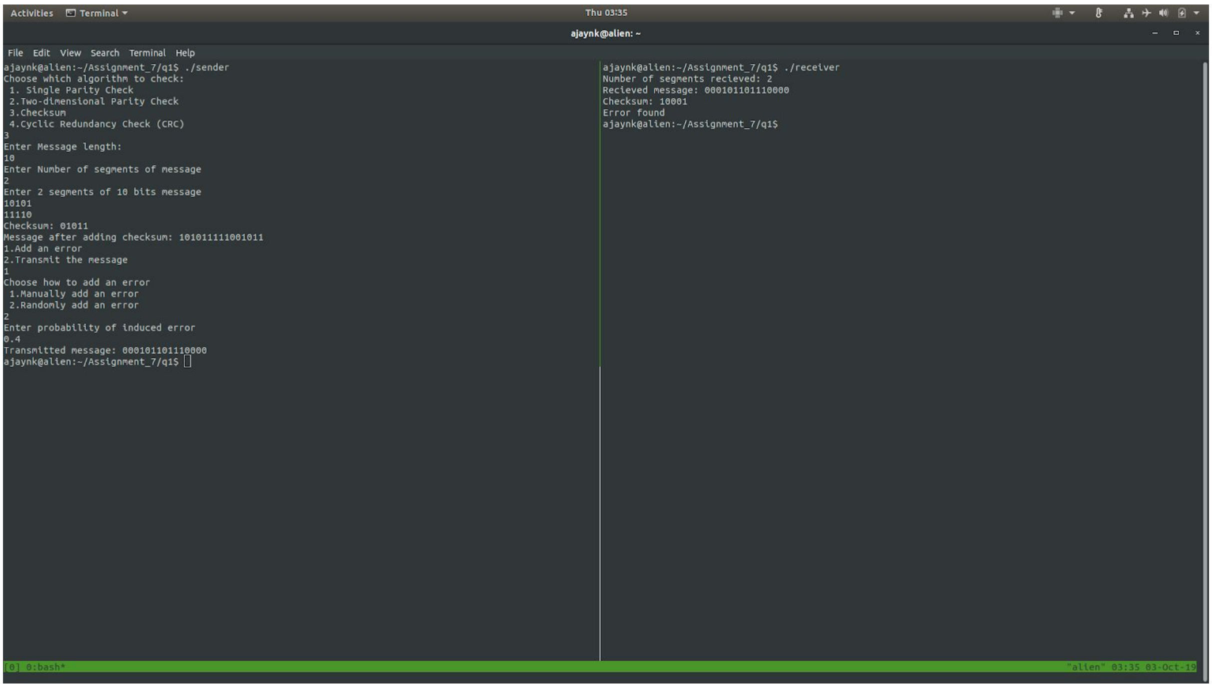
```
[0] 0:bash*                                                    "alien" 02:56 03-Oct-19
```



```
ajaynk@alien:~/Assignment_7/q1$ ./sender
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
2
Enter Message length:
12
Enter Number of segments of message
3
Enter 3 segments of 12 bits message
1010
1111
1000
Two dimensional parity matrix
1 0 1 0 0
1 1 1 1 0
1 0 0 0 1
1 1 0 1 1
1.Add an error
2.Transmit the message
2
Transmitted message: 10100111101000111011
ajaynk@alien:~/Assignment_7/q1$
```

```
ajaynk@alien:~/Assignment_7/q1$ ./receiver
Number of segments recieved: 3
Recieved message: 10100111101000111011
Error detected: No
Decoded message: 101011111000
ajaynk@alien:~/Assignment_7/q1$
```

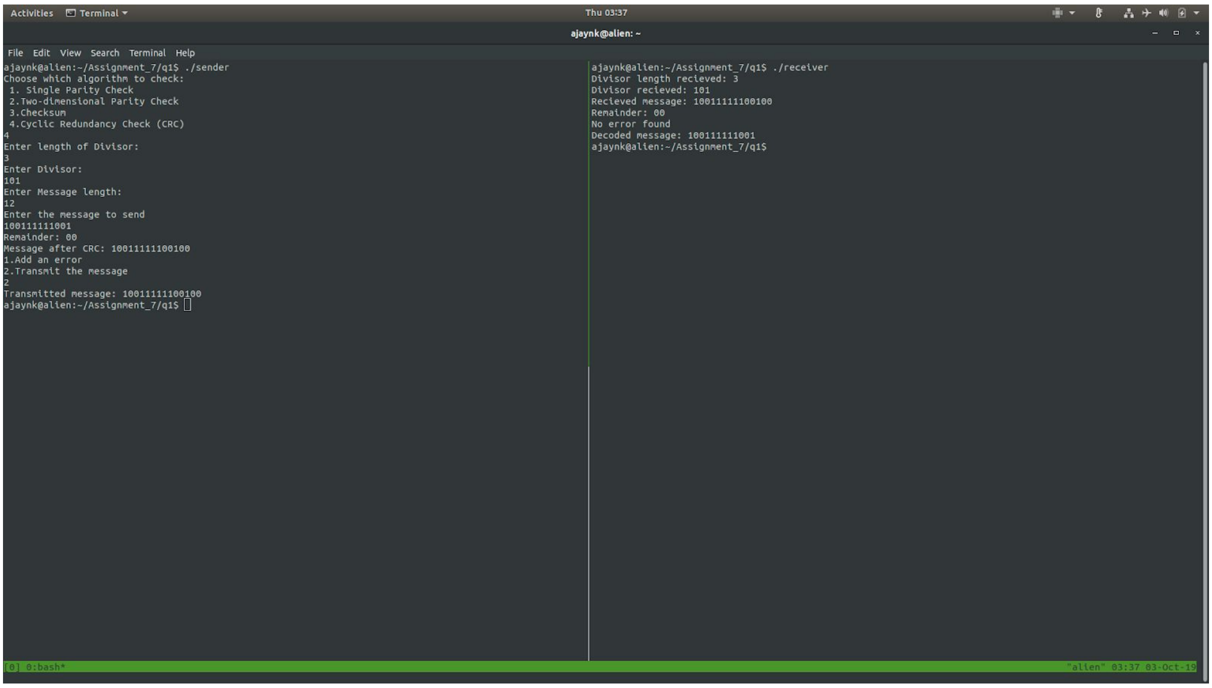
```
[0] 0:./sender*                                              "alien" 03:34 03-Oct-19
```



```
ajaynk@alien:~/Assignment_7/q1$ ./sender
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
3
Enter Message length:
10
Enter Number of segments of message
2
Enter 2 segments of 10 bits message
10101
11110
Checksum: 01011
Message after adding checksum: 101011111001011
1.Add an error
2.Transmit the message
1
Choose how to add an error
 1.Manually add an error
 2.Randomly add an error
2
Enter probability of induced error
0.4
Transmitted message: 000101101110000
ajaynk@alien:~/Assignment_7/q1$
```

```
ajaynk@alien:~/Assignment_7/q1$ ./receiver
Number of segments recieved: 2
Recieved message: 000101101110000
Checksum: 10001
Error found
ajaynk@alien:~/Assignment_7/q1$
```

```
[0] 0:bash*                                                  "alien" 03:35 03-Oct-19
```



```
ajaynk@alien:~/Assignment_7/q1$ ./sender
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
4
Enter length of Divisor:
3
Enter Divisor:
101
Enter Message length:
12
Enter the message to send
100111111001
Remainder: 00
Message after CRC: 10011111100100
1.Add an error
2.Transmit the message
2
Transmitted message: 10011111100100
ajaynk@alien:~/Assignment_7/q1$
```

```
ajaynk@alien:~/Assignment_7/q1$ ./receiver
Divisor length recieved: 3
Divisor recieved: 101
Recieved message: 10011111100100
Remainder: 00
No error found
Decoded message: 100111111001
ajaynk@alien:~/Assignment_7/q1$
```

```
[0] 0:bash*                                                    "alien" 03:37 03-Oct-19
```

Question 2:

Transmit a binary message (from a sender to a receiver) using socket programming in C. Using Hamming code to detect and correct errors in the transmitted message.

Algorithm used :

1. First we get the length of the dataword to be sent to the server from the client.
2. After taking the data word as the input, we calculate the number of redundancy bits we need to add, to form the necessary code word, by using the fomula,

$$2^m - 1 - m >= k$$

where m is the number of redundant bits, k is the size of the data word.
3. We add the redundant bits in the places whose numerical values are in powers of 2.
4. We calculate the values of the redundant bits, done as follows,
For 1st checkbit (1st position), we add the values of 1, 3, 5, 7 th bits and so on, mod 2.

For 2nd checkbit (2nd position), we add the values of 2, 3, 6, 7 th bits and so on, mod 2.
For 3rd checkbit (4th position), we add the values of 4-7, 12-15 th bits and so on, mod 2.
5. We can get some errors by flipping some bits, according to the user's needs.
6. Since the dmin is constant for all sizes of data words, and is 3, the maximum error which we can detect is 2, and the maximum error which we can correct is 1.
7. The server receives the codeword, and calculates the values of all the check bits again.
8. The numerical value of the checkbit, gives the position of the error.
9. If it is 0, then no error has been detected.
10. For non zero values, we can flip the bit at that position and get the correct codeword.
11. We then extract the data word from the codeword by neglecting the characters at positions which are powers of 2.

Data structures used :

Socket creation:

sockfd: socket descriptor, an integer

struct sockaddr_in : structure to store internet addresses like IP address, port.

Bind:

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

bind function binds the socket to the address and port number specified in addr.

Accept:

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket,

sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.

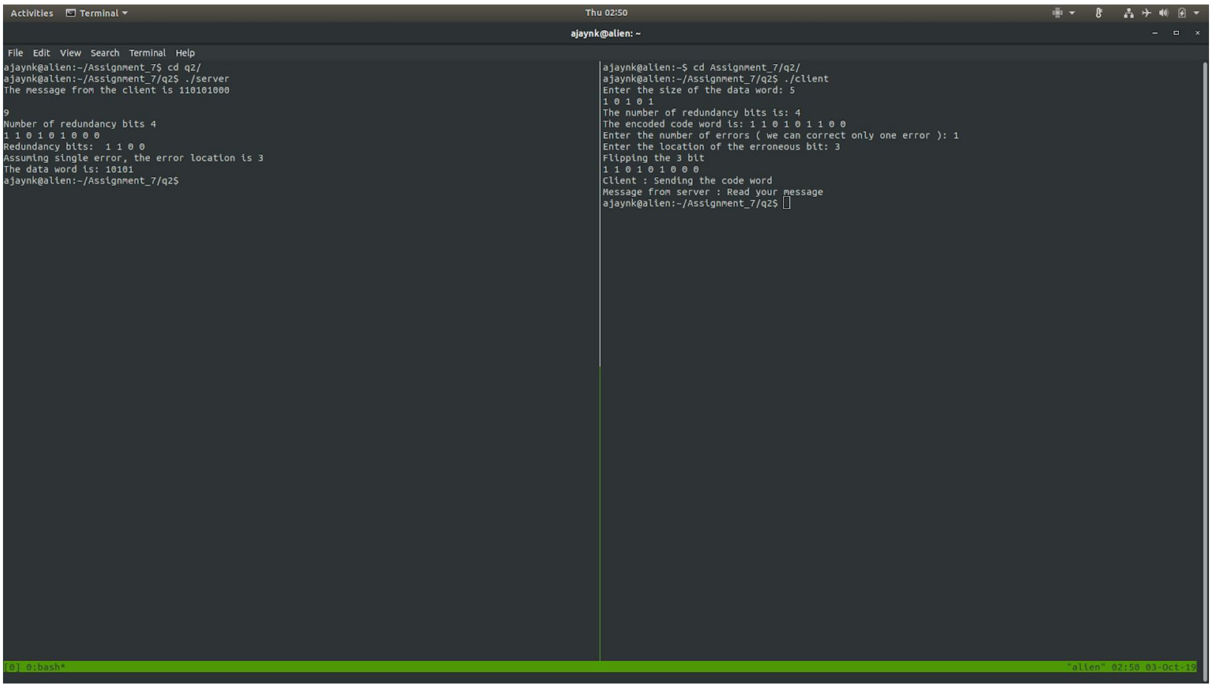
Socket connection:

same as that of server’s socket creation

Connect:

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Screenshot :



```
ajaynk@alien:~/Assignment_7$ cd q2/
ajaynk@alien:~/Assignment_7/q2$ ./server
The message from the client is 110101000

9
Number of redundancy bits 4
1 1 0 1 0 1 0 0 0
Redundancy bits:  1 1 0 0
Assuming single error, the error location is 3
The data word is: 10101
ajaynk@alien:~/Assignment_7/q2$
```

```
ajaynk@alien:~$ cd Assignment_7/q2/
ajaynk@alien:~/Assignment_7/q2$ ./client
Enter the size of the data word: 5
1 0 1 0 1
The number of redundancy bits is: 4
The encoded code word is: 1 1 0 1 0 1 1 0 0
Enter the number of errors ( we can correct only one error ): 1
Enter the location of the erroneous bit: 3
Flipping the 3 bit
1 1 0 1 0 1 0 0 0
Client : Sending the code word
Message from server : Read your message
ajaynk@alien:~/Assignment_7/q2$
```

```
[0] 0:bash*                                                    "alien" 02:50 03-Oct-19
```

Question 3 :

Write a C++ program to compress a message (non-binary, can be anything like a text message or a code like hexadecimal, etc.) using the following data compression algorithm:

1. Huffman 2. Shannon-Fano

Algorithms used :

HUFFMAN CODING

There are mainly two major parts in Huffman Coding

1) Build a Huffman Tree from input characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

2) Traverse the Huffman Tree and assign codes to characters.

SHANNON FANO CODING

The steps of the algorithm are as follows:

1. Create a list of probabilities or frequency counts for the given set of symbols so that the relative frequency of occurrence of each symbol is known.

2. Sort the list of symbols in decreasing order of probability, the most probable ones to the left and least probable to the right.

3. Split the list into two parts, with the total probability of both the parts being as close to each other as possible.

4. Assign the value 0 to the left part and 1 to the right part.

5. Repeat the steps 3 and 4 for each part, until all the symbols are split into individual subgroups.

Data Structures used :

1. ofstream: Stream class to write on files

2. ifstream: Stream class to read from files

3. fstream: Stream class to both read and write from/to files.

4. Node- a struct tree node

5. getNode- Function to allocate a new tree node

6. comp-Comparison object to be used to order the heap

7. encode-traverse the Huffman Tree and store Huffman Codes in a map.

8. Qsort -sort ptable according to probabilities

9. decode-traverse the Huffman Tree and decode the encoded string

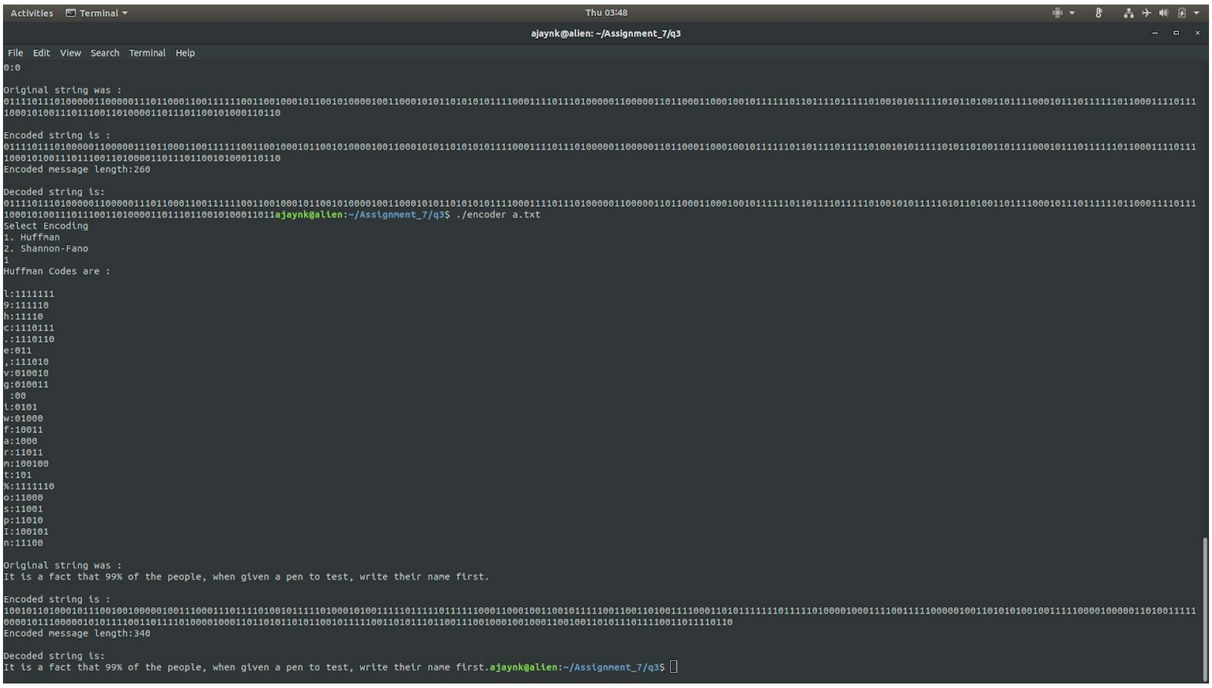
10. buildHuffmanTree - Builds Huffman Tree and decode given input text

11. ptable- table of probability

12. EncShannon-Calculating sum of probabilities at specified interval

Screenshots :

Huffman

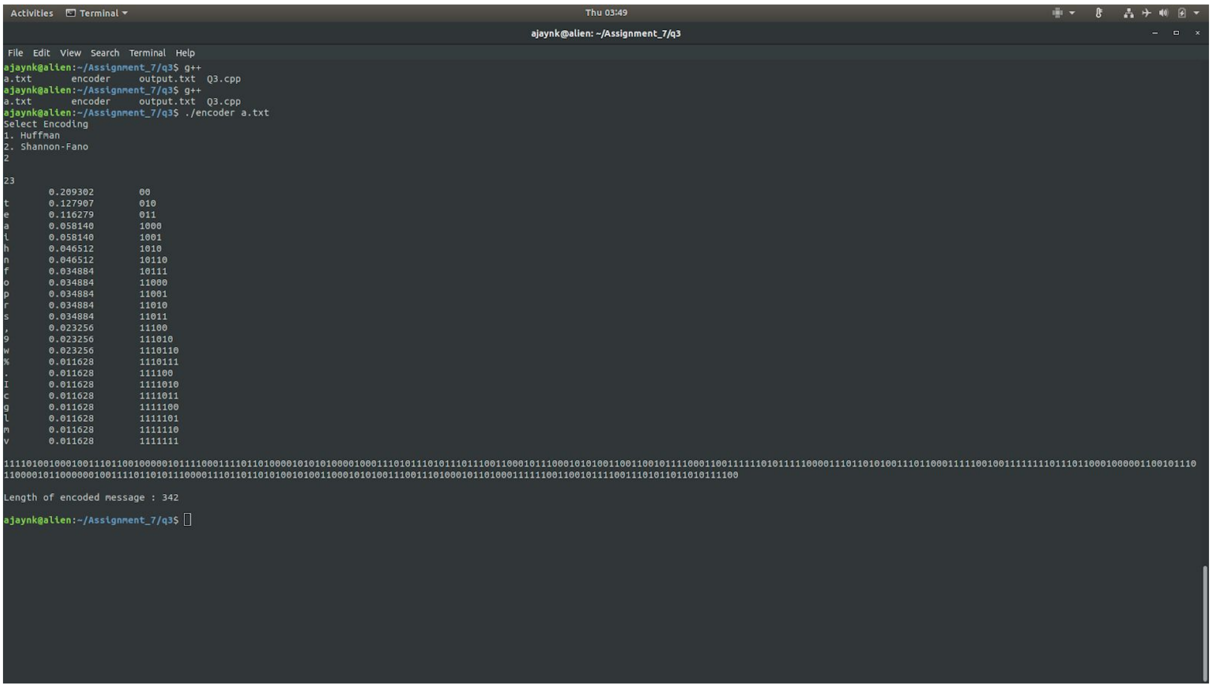


Activities
Terminal
Thu 03:48
ajaynk@alien: ~/Assignment_7/q3
File Edit View Search Terminal Help

```
0:0

Original string was :
0111101110100000110000011101100011001111110011001000101100101000010011000101011010101011110001111011101000001100000110110001100010010111111011011101111110100101011111010110100110111100010111011111101100011110111
1000101001110111001101000011011101100101000110110

Encoded string is :
0111101110100000110000011101100011001111110011001000101100101000010011000101011010101011110001111011101000001100000110110001100010010111111011011101111110100101011111010110100110111100010111011111101100011110111
1000101001110111001101000011011101100101000110110
Encoded message length:260

Decoded string is:
0111101110100000110000011101100011001111110011001000101100101000010011000101011010101011110001111011101000001100000110110001100010010111111011011101111110100101011111010110100110111100010111011111101100011110111
100010100111011100110100001101110110010100011011ajaynk@alien:~/Assignment_7/q3$ ./encoder a.txt
Select Encoding
1. Huffman
2. Shannon-Fano
1
Huffman Codes are :

l:1111111
9:111110
h:11110
c:1110111
.:1110110
e:011
,:111010
v:010010
g:010011
 :00
i:0101
w:01000
f:10011
a:1000
r:11011
m:100100
t:101
%:1111110
o:11000
s:11001
p:11010
I:100101
n:11100

Original string was :
It is a fact that 99% of the people, when given a pen to test, write their name first.

Encoded string is :
1001011010001011100100100000100111000111011110100101111110100010100111110111111011111100011000100110010111110011001101001111000110101111111011111010000100011110011111000001001101010100100111110000100000110100111111
000010111000000101011110011011110100001000110110101101010110010111110011010111011001110010001001000110010011010111011110011011110110
Encoded message length:340

Decoded string is:
It is a fact that 99% of the people, when given a pen to test, write their name first.ajaynk@alien:~/Assignment_7/q3$ ▯
```

Shannon

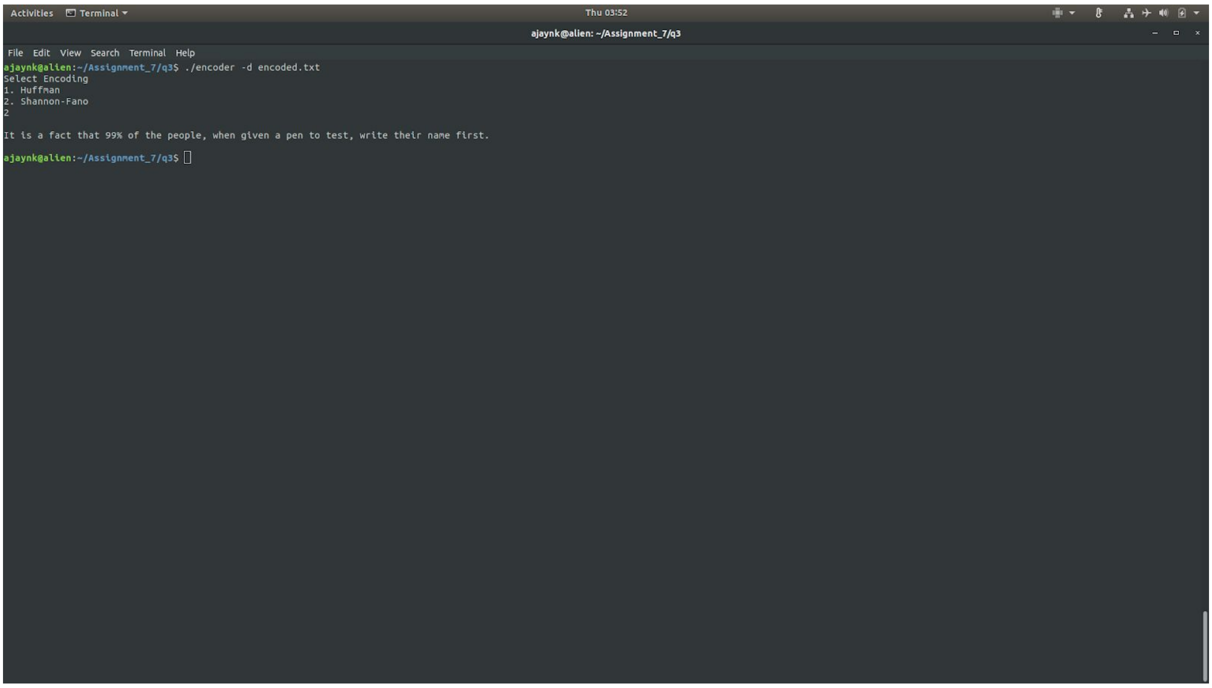
Encoding


Activities
Terminal
Thu 03:49
ajaynk@alien: ~/Assignment_7/q3
File Edit View Search Terminal Help

```
ajaynk@alien:~/Assignment_7/q3$ g++
a.txt       encoder     output.txt  Q3.cpp
ajaynk@alien:~/Assignment_7/q3$ g++
a.txt       encoder     output.txt  Q3.cpp
ajaynk@alien:~/Assignment_7/q3$ ./encoder a.txt
Select Encoding
1. Huffman
2. Shannon-Fano
2


23
        0.209302        00
t       0.127907        010
e       0.116279        011
a       0.058140        1000
i       0.058140        1001
h       0.046512        1010
n       0.046512        10110
f       0.034884        10111
o       0.034884        11000
p       0.034884        11001
r       0.034884        11010
s       0.034884        11011
,       0.023256        11100
9       0.023256        111010
w       0.023256        1110110
%       0.011628        1110111
.       0.011628        111100
I       0.011628        1111010
c       0.011628        1111011
g       0.011628        1111100
l       0.011628        1111101
m       0.011628        1111110
v       0.011628        1111111

1111010010001001110110010000010111100011110110100001010101000010001110101110101110111001100010111000101010011001100101111000110011111101011111000011101101010011101100011111001001111111101110110001000001100101110
11000010110000000100111101101011100001110110110101001010011000101010011100111010001011010001111110011001011110011101011011010111100

Length of encoded message : 342

ajaynk@alien:~/Assignment_7/q3$ 
```

Shannon

Decoding


Activities
Terminal
Thu 03:52
ajaynk@alien: ~/Assignment_7/q3
File Edit View Search Terminal Help

```
ajaynk@alien:~/Assignment_7/q3$ ./encoder -d encoded.txt
Select Encoding
1. Huffman
2. Shannon-Fano
2

It is a fact that 99% of the people, when given a pen to test, write their name first.

ajaynk@alien:~/Assignment_7/q3$ 
```