Project 2 - Client Server  
Isaac Menis

10/18/23

# 

[**Data Flow Diagram 2**](#_4kjm6n1g4zew)

[**Routines 3**](#_oy86dmjvwfrd)

[charToBits 3](#_p4x66mqnz67c)

[addParity 3](#_k5jkv6qip8z3)

[toFrame 3](#_dm89kmebv1ea)

[fromFrame 3](#_drxi96pcynxw)

[validateRemoveParity 3](#_ecdchigof4ip)

[bitsToChar 3](#_u47l3p77qai2)

[toUpperCase 3](#_fdqnrbrpp255)

[readInput 3](#_43r102f7tjby)

[outputToFile 4](#_ayqfyuuzhz0j)

[errorSimulation 4](#_kdtexpfihm8c) **5**

[**Testing and Use 5**](#_q17qisriytdu)

[**Testing 5**](#_6ufzuwmfkzm)

# Data Flow Diagram

Figure 1: Data Flow Diagram

# Routines

## charToBits

Takes a stream of characters from standard in, converts them to binary format, and outputs the result to standard out

## addParity

Takes a stream of 7 characters at a time from standard in, calculates the parity bit, then outputs the 7 bits + the parity bit to standard out

## toFrame

Takes a steam of 64 characters at a time from standard in, outputs two control symbols, outputs the length if there were less than 64 characters left otherwise outputs 64, then outputs the inputted characters to standard out, along with filler 0s to fill the frame

## fromFrame

Takes 3 characters from standard in, compares the first two to the ascii control symbol, 22, then checks if the length exists and reads in the next 64 characters. Finally outputs the numbers of characters equal to the length of the frame to standard out

## validateRemoveParity

Takes a stream of 8 characters at a time from standard in, checks that the odd parity is correct, and outputs the other 7 characters to standard out

## bitsToChar

Takes a stream of 7 bits at a time from standard in, converts it to the decimal form, and outputs the char representation to standard out

## toUpperCase

Takes a stream of characters, converting each lowercase letter to uppercase if not already, and outputting the result to standard out

## readInput

Takes a filename from standard in, opens it and reads every character out to standard out

## outputToFile

Takes a filename as an argument, and a stream of characters from standard in, and creates a file with that filename, outputting everything from standard in to the new file and standard out

## errorSimulation

Takes in a frame from standard input, flips ~5% of bits, and sends it to the next service

## tagCreator

Contains different pairs of functions to get size of future message and to create message based off of parameters

## tagParser

Helper methods for client server language. Handles consuming unneeded tags, extra bits, and returning current tag to be processed.

## errorSimulation

Takes in a frame from standard input, flips ~5% of bits, and sends it to the next service

# Testing and Use

In order to use this program run the make command, then run the main file with the desired test file name without the extension. For example an input file of yellow.inpf would be run as

1. gcc server.c -o server
2. gcc client.c -o client
3. It will ask for a username then you can type in a message, then it asks for a recipient username and encoding method.
4. gcc encodingDemonstration.c -o encodingDemonstration -lm “filename” “encoding method”
5. This will demonstrate error correction