## CSE 232 — Computer Network Assignment 2

**Name:** Bhavya Narnoli        **Due Date:** : September 12, 2023
**Student Number:** 2021316        **Assignment:** Number 2

**Problem 1** **Explaining the private members of the `bytestream.hh` class**

(a) `size_t capacity`:

This member variable stores the maximum capacity of the byte stream. It defines the maximum number of bytes that can be held in the stream at any given time which isn't read yet.

(b) `std::deque<char> buffer`:

The 'buffer' is an internal data structure used to store the bytes in the stream. It is implemented as a double-ended queue (deque) and is used to hold data that has been written into the stream but has not been read yet.

(c) `bool inputended`:

The 'inputended' is a boolean flag that indicates whether the input to the byte stream has ended. When this flag is set to 'true', it signifies that no more data can be written into the stream.

(d) `size_t readcounter = 0`:

'readcounter' is an unsigned integer that keeps track of the number of bytes that have been read from the byte stream. It is initialized to zero and is incremented each time data is read from the stream.

(e) `size_t writecounter = 0`:

'writecounter' is an unsigned integer that keeps track of the number of bytes that have been written into the byte stream. It is initialized to zero and is incremented each time data is successfully written into the stream.

(f) `bool _error`:

The '_error' member is a boolean flag that serves as a private indicator of whether the byte stream has encountered an error. It is used internally to track error conditions within the stream.

```cpp
class ByteStream {
  private:
    // Your code here -- add private members as necessary.
    size_t capacity;
    std::deque<char> buffer;  // Using a deque for the buffer (which are not read yet).
    bool inputended ;
    size_t readcounter=0 ;
    size_t writecounter =0;
    bool _error;  //!< Flag indicating that the stream suffered an error.
```

Figure 1: .hh private members

# 1 Explanation of C++ Code

In the provided C++ code, we have a class named `ByteStream` that is designed to manage a byte stream. Let's break down the code and explain it step by step:

```cpp
class ByteStream {
private:
    size_t capacity;        // Maximum capacity of the byte stream
    bool inputended;        // Flag to indicate the end of input
    size_t writecounter;    // Counter for bytes written
    size_t readcounter;     // Counter for bytes read
    std::deque<char> buffer; // Deque to store the byte stream
    data

public:
    // Constructor to initialize the ByteStream object
    ByteStream(const size_t capa);

    // Method to write data into the byte stream
    size_t write(const std::string &data);

    // Method to peek a specified number of bytes from the output
    side of the buffer
    std::string peek_output(const size_t len) const;

    // Method to remove a specified number of bytes from the
    output side of the buffer
    void pop_output(const size_t len);

    // Method to read and return the next "len" bytes from the
    stream
    std::string read(const size_t len);

    // Method to mark the end of input
    void end_input();
```

```
28    // Method to check if input has ended
29    bool input_ended() const;
30
31    // Method to get the current size of the buffer
32    size_t buffer_size() const;
33
34    // Method to check if the buffer is empty
35    bool buffer_empty() const;
36
37    // Method to check if the end of the stream has been reached
38    bool eof() const;
39
40    // Method to get the total number of bytes written
41    size_t bytes_written() const;
42
43    // Method to get the total number of bytes read
44    size_t bytes_read() const;
45
46    // Method to calculate the remaining capacity in the buffer
47    size_t remaining_capacity() const;
48 };
```

Listing 1: ByteStream Class Definition

In the provided C++ code, this is the implementation of the code

```
1  #include "byte_stream.hh"
2
3  #include <algorithm>
4
5  // You will need to add private members to the class declaration
       in `byte_stream.hh`
6
7  /* Replace all the dummy definitions inside the methods in this
       file. */
8
9  #include <iostream>
10
11 using namespace std;
12
13 ByteStream::ByteStream(const size_t capa) : capacity(capa),
       inputended(false), writecounter(0), readcounter(0)
14 {}
15
16 size_t ByteStream::write(const string &data) {
17     if (inputended ) {
18         // Stream has ended or encountered an error, cannot write.
19         return 0;
20     }
21
22     size_t space_available = remaining_capacity();
23     size_t bytes_to_write_in_available_buffer = std::min( data.
       length() , space_available );
24
```

```cpp
25      // Calculate iterators for the range of data to insert into
        the deque.
26      auto dataBegin = data.begin();
27      auto dataEnd = dataBegin + bytes_to_write_in_available_buffer;
28
29      // Insert the data into the deque.
30      buffer.insert(buffer.end(), dataBegin, dataEnd);
31
32      // Update the write counter.
33      writecounter += bytes_to_write_in_available_buffer;
34
35      return bytes_to_write_in_available_buffer;
36  }
37
38  //! \param[in] len bytes will be copied from the output side of
        the buffer
39  string ByteStream::peek_output(const size_t len) const {
40
41    size_t BYTES_AVAILABLE = buffer.size();
42    size_t bytes_to_peek = std::min(len, BYTES_AVAILABLE);
43
44    std::string result( buffer.begin(), buffer.begin() +
      bytes_to_peek );
45    return result;
46    }
47
48  //! \param[in] len bytes will be removed from the output side of
        the buffer
49  void ByteStream::pop_output(const size_t len) {
50
51      if (len > buffer.size()) {
52          set_error();
53          return;
54      }
55
56      size_t Bytes_to_remove_from_buffer = std::min(len, buffer.size
        ());
57      readcounter += Bytes_to_remove_from_buffer; // not sure on
        this
58      buffer.erase(buffer.begin(), buffer.begin() +
        Bytes_to_remove_from_buffer);
59
60  }
61
62  //! Read (i.e., copy and then pop) the next "len" bytes of the
        stream
63  //! \param[in] len bytes will be popped and returned
64  //! \returns a string
65
66  std::string ByteStream::read(const size_t len) {
67
68   if (len > buffer.size()) {
69          set_error();
70          return ""; // e.sg., an empty string
```

```
71      }
72
73      std::string result(buffer.begin(), buffer.begin() + len);
74
75      // Erase the read data from the buffer.
76      buffer.erase(buffer.begin(), buffer.begin() + len);
77
78      readcounter += len;
79      return result;
80
81 }
82
83 void ByteStream::end_input() { inputended = true;}
84
85 bool ByteStream::input_ended() const { return inputended;;}
86
87 size_t ByteStream::buffer_size() const {return buffer.size(); }
88
89 bool ByteStream::buffer_empty() const {return buffer.empty() ; }
90
91 bool ByteStream::eof() const { return  inputended && buffer.empty
       (); }
92
93 size_t ByteStream::bytes_written() const { return writecounter; }
94
95 size_t ByteStream::bytes_read() const { return readcounter;  }
96
97 size_t ByteStream::remaining_capacity() const { return capacity -
       buffer.size(); }
```
Listing 2: ByteStream Class Definition

The `ByteStream` class has private member variables to store information about the stream, including its capacity, whether input has ended, counters for bytes written and read, and a deque called `buffer` to hold the byte stream data.

The class provides several methods to interact with the stream:

a) `ByteStream(const size_t capa)`: Constructor to initialize the `ByteStream` object with the given capacity.

b) `size_t write(const std::string &data)`: Writes data from the provided string `data` into the buffer. It checks if input has ended and returns the number of bytes written in the remaining capacity of buffer (can't exceed that)

c) `std::string peek_output(const size_t len) const`: Retrieves a specified number of bytes from the output side of the buffer without removing them.

6

d) `void pop_output(const size_t len)`: Removes a specified number of bytes from the output side of the buffer, unless if size is more than that of buffer, it sets the error flag.

e) `std::string read(const size_t len)`: Reads and returns the next "len" bytes from the stream. It also removes the read data from the buffer unless if size is more than that of buffer, it sets the error flag.

f) `void end_input()`: Marks the end of input by setting the `inputended` flag.

g) `bool input_ended() const`: Checks if input has ended.

h) `size_t buffer_size() const`: Gets the current size of the buffer.

i) `bool buffer_empty() const`: Checks if the buffer is empty.

j) `bool eof() const`: Checks if the end of the stream has been reached (both inputended and buffer is empty).

k) `size_t bytes_written() const`: Gets the total number of bytes written.

l) `size_t bytes_read() const`: Gets the total number of bytes read.

m) `size_t remaining_capacity() const`: Calculates the remaining capacity in the buffer.

This `ByteStream` class is designed to provide basic functionality for managing and manipulating byte streams

Figure 2: Test cases pass