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`:

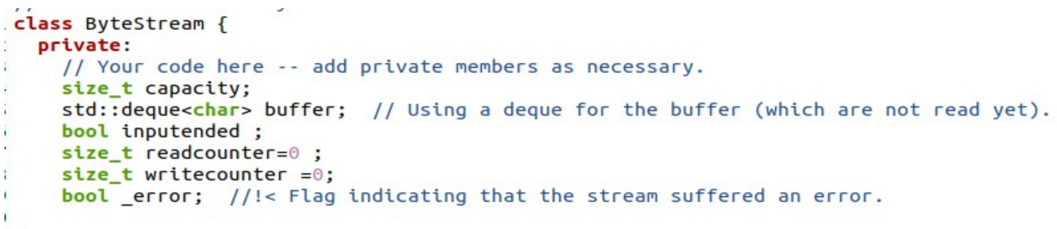
'readcounter' is an unsigned integer that keeps track of the number of bytes that have been read from the byte stream. It is initialized to zero and is incremented each time data is read from the stream.

(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.



```
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

Explaining C++ Code with LaTeX

September 12, 2023

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();

    // Method to check if input has ended
    bool input_ended() const;

    // Method to get the current size of the buffer
    size_t buffer_size() const;

    // Method to check if the buffer is empty
    bool buffer_empty() const;

    // Method to check if the end of the stream has been reached
    bool eof() const;

    // Method to get the total number of bytes written
    size_t bytes_written() const;

    // Method to get the total number of bytes read
    size_t bytes_read() const;

    // Method to calculate the remaining capacity in the buffer
    size_t remaining_capacity() const;
};
```

Listing 1: ByteStream Class Definition

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

```cpp
#include "byte_stream.hh"

#include <algorithm>

// You will need to add private members to the class declaration
    in `byte_stream.hh`

/* Replace all the dummy definitions inside the methods in this
    file. */

#include <iostream>

using namespace std;

ByteStream::ByteStream(const size_t capa) : capacity(capa),
    inputended(false), writecounter(0), readcounter(0)
{}

size_t ByteStream::write(const string &data) {
    if (inputended ) {
        // Stream has ended or encountered an error, cannot write.
        return 0;
    }

    size_t space_available = remaining_capacity();
    size_t bytes_to_write_in_available_buffer = std::min( data.
    length() , space_available );

    // Calculate iterators for the range of data to insert into the deque.
    auto dataBegin = data.begin();
    auto dataEnd = dataBegin + bytes_to_write_in_available_buffer;

    // Insert the data into the deque.
    buffer.insert(buffer.end(), dataBegin, dataEnd);

    // Update the write counter.
    writecounter += bytes_to_write_in_available_buffer;

    return bytes_to_write_in_available_buffer;
}

//! \param[in] len bytes will be copied from the output side of the buffer
string ByteStream::peek_output(const size_t len) const {

  size_t BYTES_AVAILABLE = buffer.size();
  size_t bytes_to_peek = std::min(len, BYTES_AVAILABLE);

  std::string result( buffer.begin(), buffer.begin() + bytes_to_peek );
  return result;
  }

//! \param[in] len bytes will be removed from the output side of the buffer
void ByteStream::pop_output(const size_t len) {

    if (len > buffer.size()) {
        set_error();
        return;
    }

    size_t Bytes_to_remove_from_buffer = std::min(len, buffer.size());
    readcounter += Bytes_to_remove_from_buffer; // not sure on this
    buffer.erase(buffer.begin(), buffer.begin() + Bytes_to_remove_from_buffer);

}

//! Read (i.e., copy and then pop) the next "len" bytes of the stream
//! \param[in] len bytes will be popped and returned
//! \returns a string

std::string ByteStream::read(const size_t len) {

 if (len > buffer.size()) {
        set_error();
        return ""; // e.sg., an empty string
    }

    std::string result(buffer.begin(), buffer.begin() + len);

    // Erase the read data from the buffer.
    buffer.erase(buffer.begin(), buffer.begin() + len);

    readcounter += len;
    return result;

}

void ByteStream::end_input() { inputended = true;}

bool ByteStream::input_ended() const { return inputended;;}

size_t ByteStream::buffer_size() const {return buffer.size(); }

bool ByteStream::buffer_empty() const {return buffer.empty() ; }

bool ByteStream::eof() const { return  inputended && buffer.empty
    (); }

size_t ByteStream::bytes_written() const { return writecounter; }

size_t ByteStream::bytes_read() const { return readcounter;  }

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.

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.

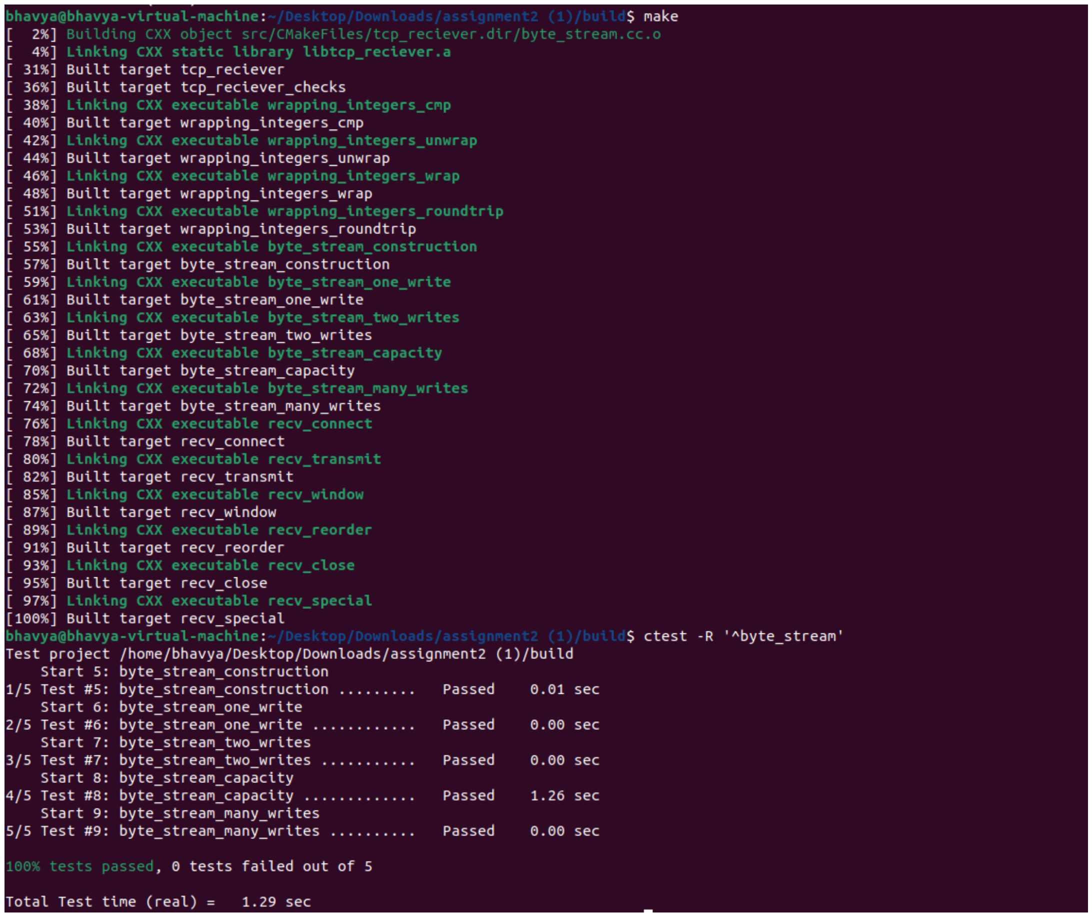
l) `size_t bytes_read() const`: Gets the total number of bytes read.

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

References

`https://rqdmap.top/posts/cs144-lab0/`



```
bhavya@bhavya-virtual-machine:~/Desktop/Downloads/assignment2 (1)/build$ make
[  2%] Building CXX object src/CMakeFiles/tcp_reciever.dir/byte_stream.cc.o
[  4%] Linking CXX static library libtcp_reciever.a
[ 31%] Built target tcp_reciever
[ 36%] Built target tcp_reciever_checks
[ 38%] Linking CXX executable wrapping_integers_cmp
[ 40%] Built target wrapping_integers_cmp
[ 42%] Linking CXX executable wrapping_integers_unwrap
[ 44%] Built target wrapping_integers_unwrap
[ 46%] Linking CXX executable wrapping_integers_wrap
[ 48%] Built target wrapping_integers_wrap
[ 51%] Linking CXX executable wrapping_integers_roundtrip
[ 53%] Built target wrapping_integers_roundtrip
[ 55%] Linking CXX executable byte_stream_construction
[ 57%] Built target byte_stream_construction
[ 59%] Linking CXX executable byte_stream_one_write
[ 61%] Built target byte_stream_one_write
[ 63%] Linking CXX executable byte_stream_two_writes
[ 65%] Built target byte_stream_two_writes
[ 68%] Linking CXX executable byte_stream_capacity
[ 70%] Built target byte_stream_capacity
[ 72%] Linking CXX executable byte_stream_many_writes
[ 74%] Built target byte_stream_many_writes
[ 76%] Linking CXX executable recv_connect
[ 78%] Built target recv_connect
[ 80%] Linking CXX executable recv_transmit
[ 82%] Built target recv_transmit
[ 85%] Linking CXX executable recv_window
[ 87%] Built target recv_window
[ 89%] Linking CXX executable recv_reorder
[ 91%] Built target recv_reorder
[ 93%] Linking CXX executable recv_close
[ 95%] Built target recv_close
[ 97%] Linking CXX executable recv_special
[100%] Built target recv_special
bhavya@bhavya-virtual-machine:~/Desktop/Downloads/assignment2 (1)/build$ ctest -R '^byte_stream'
Test project /home/bhavya/Desktop/Downloads/assignment2 (1)/build
    Start 5: byte_stream_construction
1/5 Test #5: byte_stream_construction .........   Passed    0.01 sec
    Start 6: byte_stream_one_write
2/5 Test #6: byte_stream_one_write ............   Passed    0.00 sec
    Start 7: byte_stream_two_writes
3/5 Test #7: byte_stream_two_writes ...........   Passed    0.00 sec
    Start 8: byte_stream_capacity
4/5 Test #8: byte_stream_capacity .............   Passed    1.26 sec
    Start 9: byte_stream_many_writes
5/5 Test #9: byte_stream_many_writes ..........   Passed    0.00 sec

100% tests passed, 0 tests failed out of 5

Total Test time (real) =   1.29 sec
```

Figure 2: Test cases pass