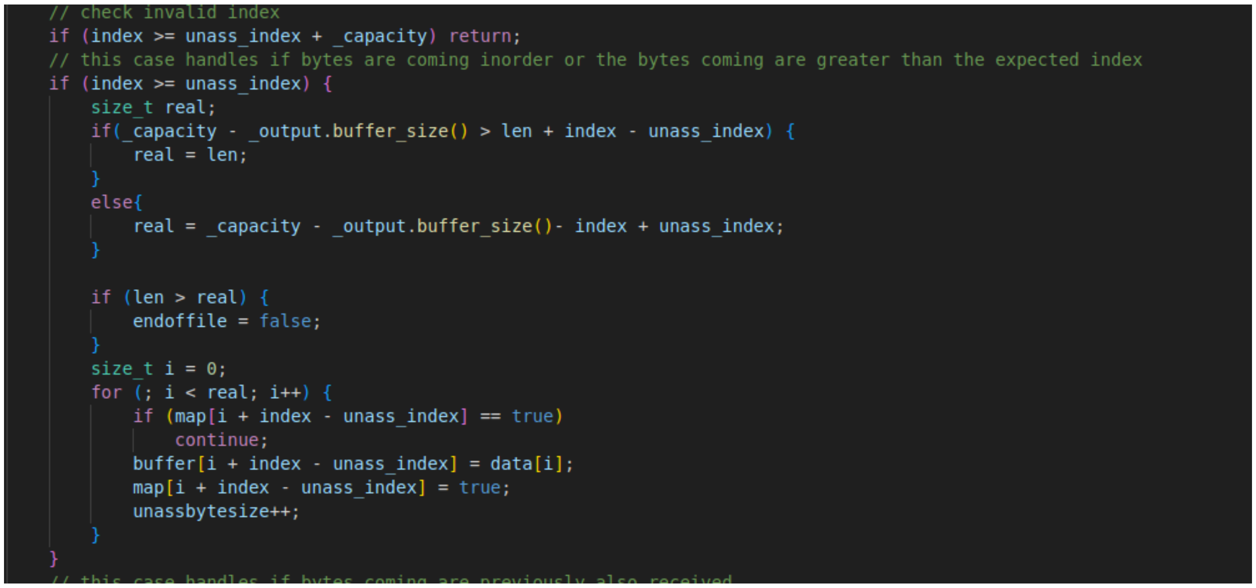
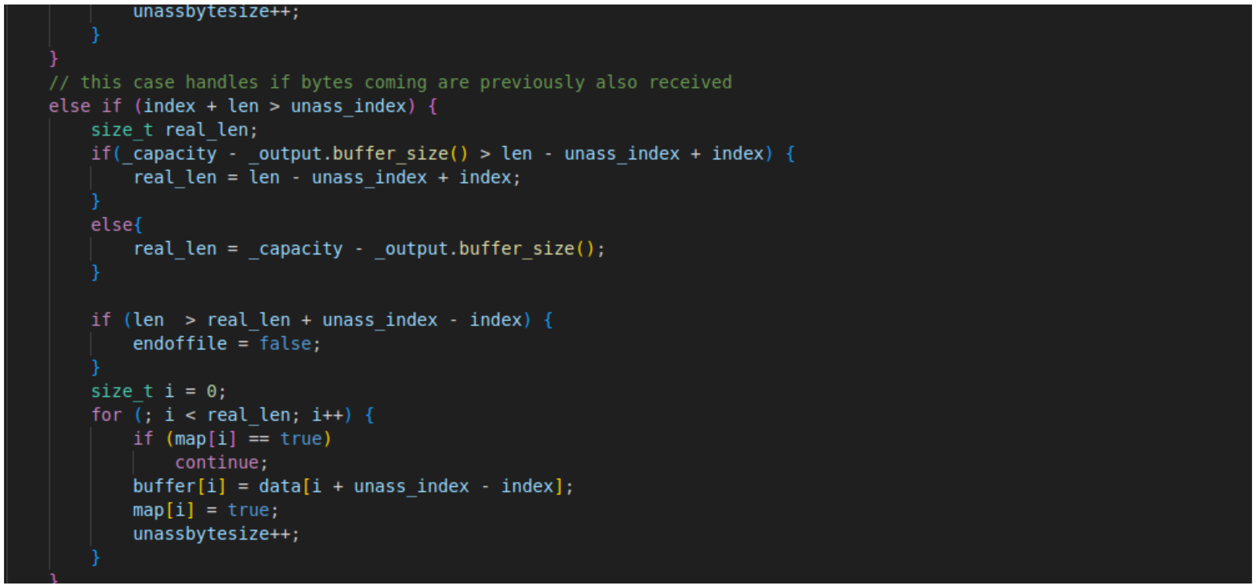
CN Assignment 2

Stream Reassembler

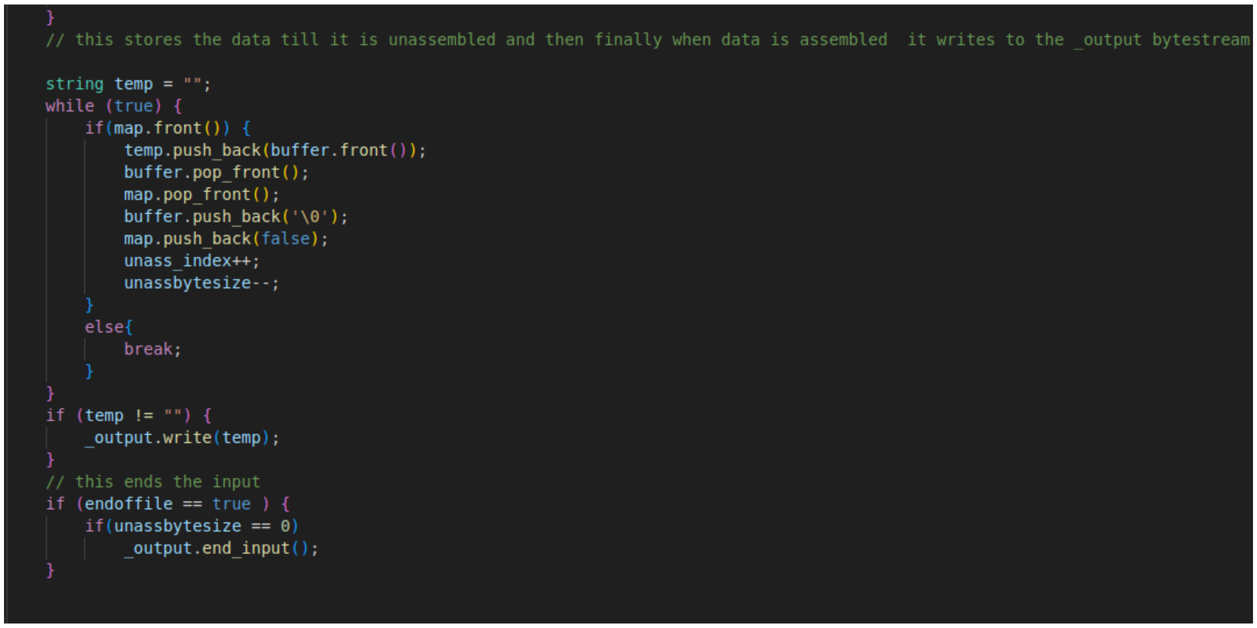
Push_Substring function



```
    // check invalid index
    if (index >= unass_index + _capacity) return;
    // this case handles if bytes are coming inorder or the bytes coming are greater than the expected index
    if (index >= unass_index) {
        size_t real;
        if(_capacity - _output.buffer_size() > len + index - unass_index) {
            real = len;
        }
        else{
            real = _capacity - _output.buffer_size()- index + unass_index;
        }

        if (len > real) {
            endoffile = false;
        }
        size_t i = 0;
        for (; i < real; i++) {
            if (map[i + index - unass_index] == true)
                continue;
            buffer[i + index - unass_index] = data[i];
            map[i + index - unass_index] = true;
            unassbytesize++;
        }
    }
    // this case handles if bytes coming are previously also received
```

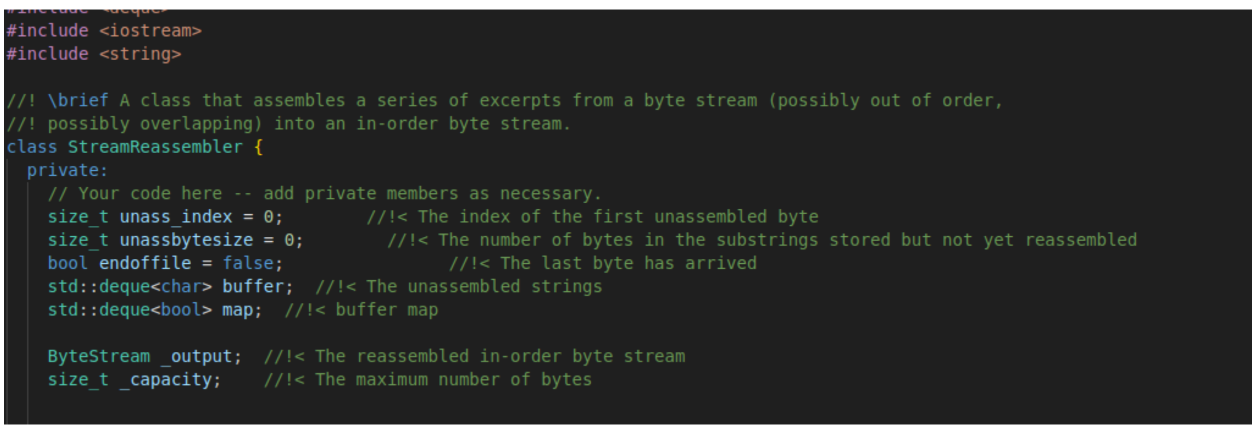


```
            unassbytesize++;
        }
    }
    // this case handles if bytes coming are previously also received
    else if (index + len > unass_index) {
        size_t real_len;
        if(_capacity - _output.buffer_size() > len - unass_index + index) {
            real_len = len - unass_index + index;
        }
        else{
            real_len = _capacity - _output.buffer_size();
        }

        if (len  > real_len + unass_index - index) {
            endoffile = false;
        }
        size_t i = 0;
        for (; i < real_len; i++) {
            if (map[i] == true)
                continue;
            buffer[i] = data[i + unass_index - index];
            map[i] = true;
            unassbytesize++;
        }
    }
```



```
    }
    // this stores the data till it is unassembled and then finally when data is assembled  it writes to the _output bytestream

    string temp = "";
    while (true) {
        if(map.front()) {
            temp.push_back(buffer.front());
            buffer.pop_front();
            map.pop_front();
            buffer.push_back('\0');
            map.push_back(false);
            unass_index++;
            unassbytesize--;
        }
        else{
            break;
        }
    }
    if (temp != "") {
        _output.write(temp);
    }
    // this ends the input
    if (endoffile == true ) {
        if(unassbytesize == 0)
            _output.end_input();
    }
```

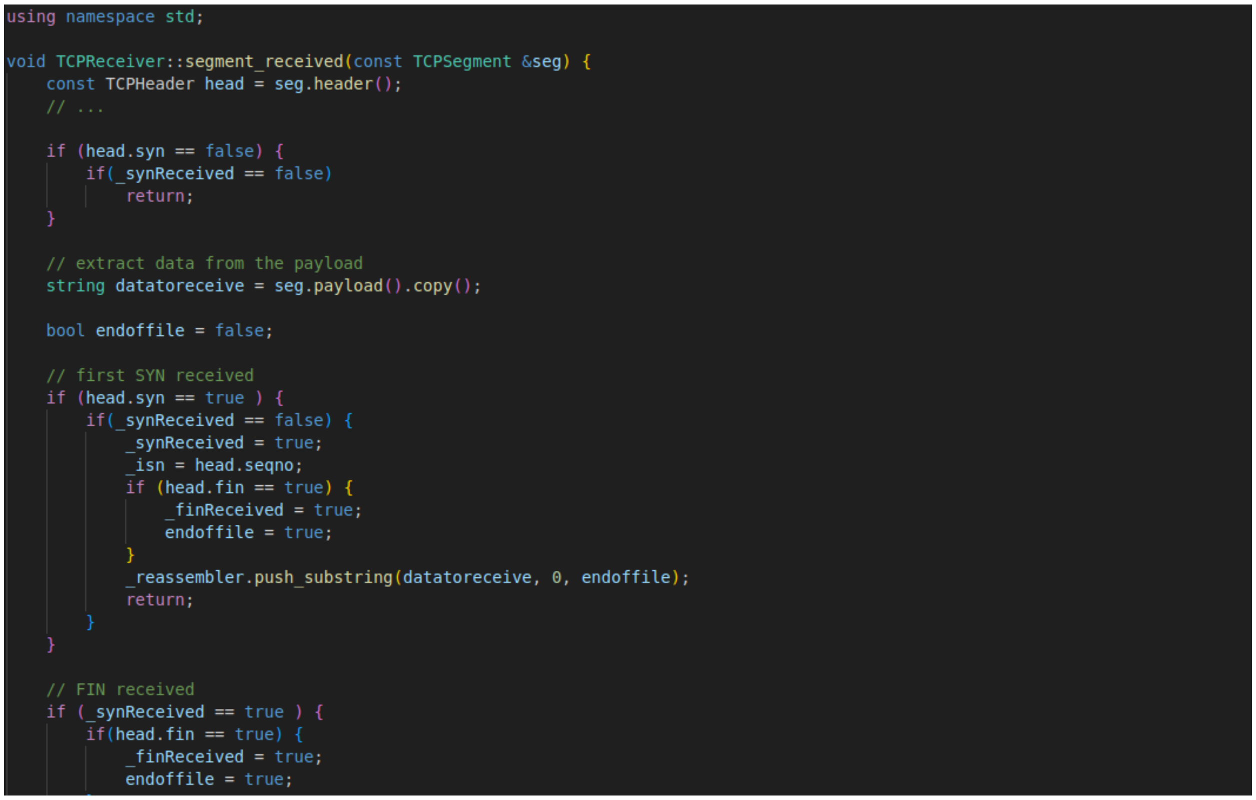
Stream Receiver.hh



```
#include <iostream>
#include <string>

//! \brief A class that assembles a series of excerpts from a byte stream (possibly out of order,
//! possibly overlapping) into an in-order byte stream.
class StreamReassembler {
  private:
    // Your code here -- add private members as necessary.
    size_t unass_index = 0;        //!< The index of the first unassembled byte
    size_t unassbytesize = 0;        //!< The number of bytes in the substrings stored but not yet reassembled
    bool endoffile = false;                //!< The last byte has arrived
    std::deque<char> buffer;  //!< The unassembled strings
    std::deque<bool> map;  //!< buffer map

    ByteStream _output;  //!< The reassembled in-order byte stream
    size_t _capacity;    //!< The maximum number of bytes
```

Changes done in the private members of StreamReceiver.hh

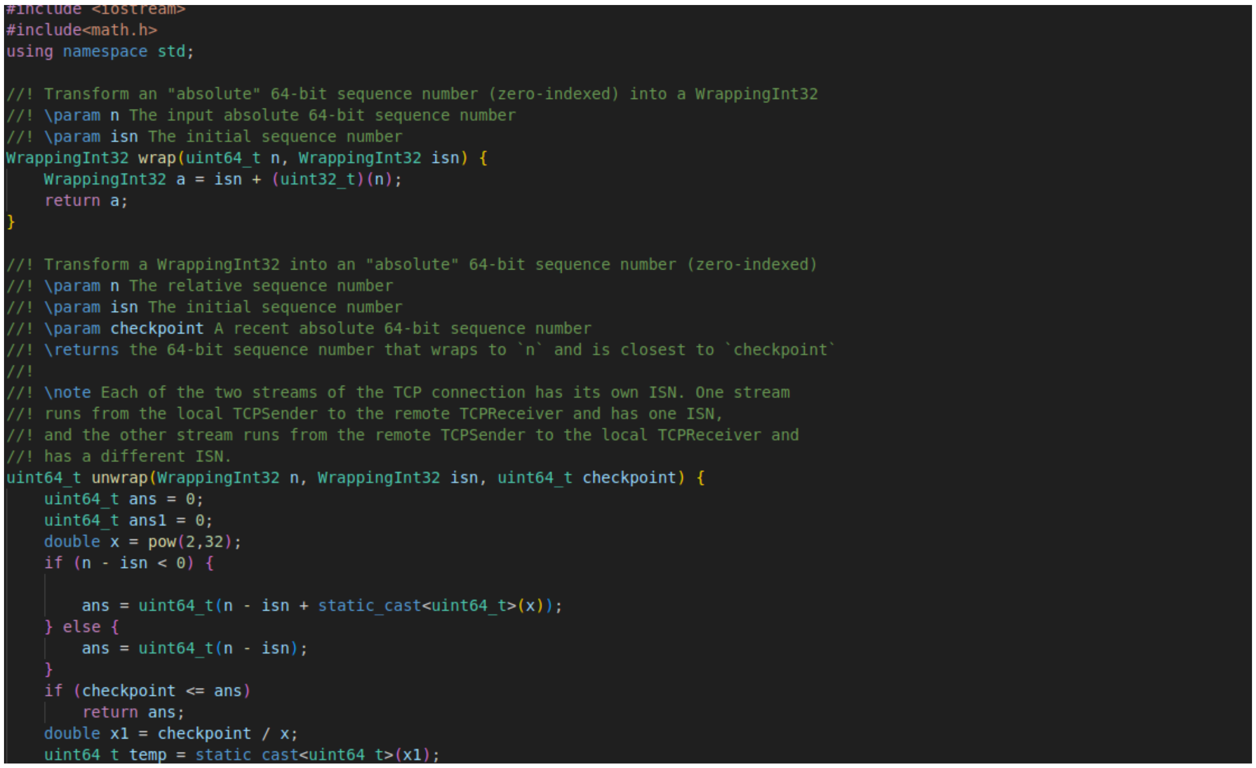
TCP Receiver.cc



```cpp
using namespace std;

void TCPReceiver::segment_received(const TCPSegment &seg) {
    const TCPHeader head = seg.header();
    // ...

    if (head.syn == false) {
        if(_synReceived == false)
            return;
    }

    // extract data from the payload
    string datatoreceive = seg.payload().copy();

    bool endoffile = false;

    // first SYN received
    if (head.syn == true ) {
        if(_synReceived == false) {
            _synReceived = true;
            _isn = head.seqno;
            if (head.fin == true) {
                _finReceived = true;
                endoffile = true;
            }
            _reassembler.push_substring(datatoreceive, 0, endoffile);
            return;
        }
    }

    // FIN received
    if (_synReceived == true ) {
        if(head.fin == true) {
            _finReceived = true;
            endoffile = true;
```

The TCP Receiver takes TCP Segments from the sender returns them unassembled base index and then sends the unassembled data to the reassembler to reassemble it.

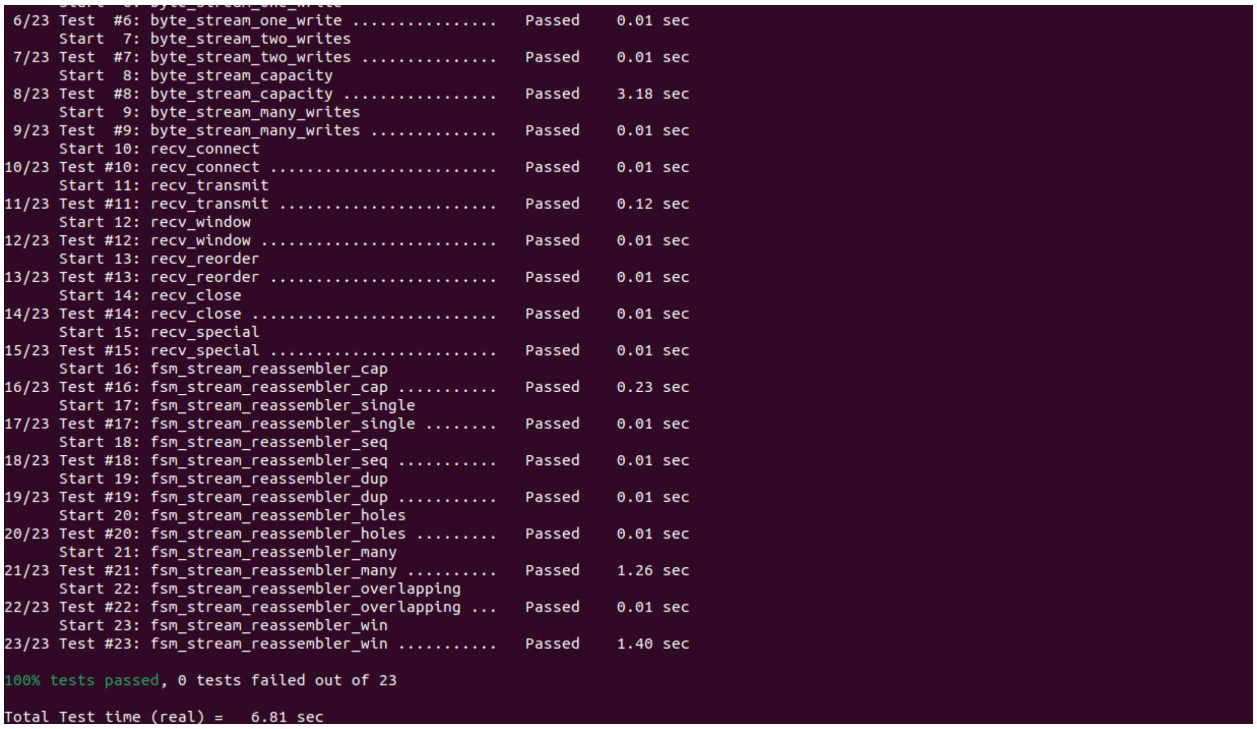
Wrappinginteger.cc



```
#include <iostream>
#include<math.h>
using namespace std;

//! Transform an "absolute" 64-bit sequence number (zero-indexed) into a WrappingInt32
//! \param n The input absolute 64-bit sequence number
//! \param isn The initial sequence number
WrappingInt32 wrap(uint64_t n, WrappingInt32 isn) {
    WrappingInt32 a = isn + (uint32_t)(n);
    return a;
}

//! Transform a WrappingInt32 into an "absolute" 64-bit sequence number (zero-indexed)
//! \param n The relative sequence number
//! \param isn The initial sequence number
//! \param checkpoint A recent absolute 64-bit sequence number
//! \returns the 64-bit sequence number that wraps to `n` and is closest to `checkpoint`
//!
//! \note Each of the two streams of the TCP connection has its own ISN. One stream
//! runs from the local TCPSender to the remote TCPReceiver and has one ISN,
//! and the other stream runs from the remote TCPSender to the local TCPReceiver and
//! has a different ISN.
uint64_t unwrap(WrappingInt32 n, WrappingInt32 isn, uint64_t checkpoint) {
    uint64_t ans = 0;
    uint64_t ans1 = 0;
    double x = pow(2,32);
    if (n - isn < 0) {

        ans = uint64_t(n - isn + static_cast<uint64_t>(x));
    } else {
        ans = uint64_t(n - isn);
    }
    if (checkpoint <= ans)
        return ans;
    double x1 = checkpoint / x;
    uint64_t temp = static_cast<uint64_t>(x1);
```

For security reasons and also to avoid confusion by older segments TCP tries to see that the sequence numbers can’t be guessed and they are assigned randomly so we have to write the functions to wrap and unwrap the integers.

Testcases run on terminal



```
 6/23 Test  #6: byte_stream_one_write ................   Passed    0.01 sec
      Start  7: byte_stream_two_writes
 7/23 Test  #7: byte_stream_two_writes ...............   Passed    0.01 sec
      Start  8: byte_stream_capacity
 8/23 Test  #8: byte_stream_capacity .................   Passed    3.18 sec
      Start  9: byte_stream_many_writes
 9/23 Test  #9: byte_stream_many_writes ..............   Passed    0.01 sec
      Start 10: recv_connect
10/23 Test #10: recv_connect .........................   Passed    0.01 sec
      Start 11: recv_transmit
11/23 Test #11: recv_transmit ........................   Passed    0.12 sec
      Start 12: recv_window
12/23 Test #12: recv_window ..........................   Passed    0.01 sec
      Start 13: recv_reorder
13/23 Test #13: recv_reorder .........................   Passed    0.01 sec
      Start 14: recv_close
14/23 Test #14: recv_close ...........................   Passed    0.01 sec
      Start 15: recv_special
15/23 Test #15: recv_special .........................   Passed    0.01 sec
      Start 16: fsm_stream_reassembler_cap
16/23 Test #16: fsm_stream_reassembler_cap ...........   Passed    0.23 sec
      Start 17: fsm_stream_reassembler_single
17/23 Test #17: fsm_stream_reassembler_single ........   Passed    0.01 sec
      Start 18: fsm_stream_reassembler_seq
18/23 Test #18: fsm_stream_reassembler_seq ...........   Passed    0.01 sec
      Start 19: fsm_stream_reassembler_dup
19/23 Test #19: fsm_stream_reassembler_dup ...........   Passed    0.01 sec
      Start 20: fsm_stream_reassembler_holes
20/23 Test #20: fsm_stream_reassembler_holes .........   Passed    0.01 sec
      Start 21: fsm_stream_reassembler_many
21/23 Test #21: fsm_stream_reassembler_many ..........   Passed    1.26 sec
      Start 22: fsm_stream_reassembler_overlapping
22/23 Test #22: fsm_stream_reassembler_overlapping ...   Passed    0.01 sec
      Start 23: fsm_stream_reassembler_win
23/23 Test #23: fsm_stream_reassembler_win ...........   Passed    1.40 sec

100% tests passed, 0 tests failed out of 23

Total Test time (real) =   6.81 sec
```