Lab 1 Writeup

My name: 王聪颖

My Student number : 211220180

This lab took me about 8 hours to do. I did attend the lab session.

1. Program Structure and Design:

ByteStream成员变量定义如下:

```
class ByteStream {
  private:
    // Your code here -- add private members as necessary.
    deque<char> _buffer{};
    size_t _capacity;
    // size_t _size{0}; not defined, use the _buffer.size() instead to
simplify code and avoid bug
    size_t _total_written{0};
    size_t _total_read{0};
    bool _flag_input_ended{false};
    // Hint: This doesn't need to be a sophisticated data structure at
    // all, but if any of your tests are taking longer than a second,
    // that's a sign that you probably want to keep exploring
    // different approaches.

    bool _error{false};  //!< Flag indicating that the stream suffered an
error.
}
```

底层容器本打算使用queue（符合input->output字节流抽象定义），但发现peek需要访问队头若干字节，queue无法提供随机访问功能，故底层容器采用deque； _size定义被注释，ByteStream不维护自己的_size变量，采取deque.size()接口，以简化代码和省去维护_size的工作，防止_size维护不当出现不一致性等bug； 成员变量命名规范与框架代码保持一致，以_开头；

ByteStream接口实现需要注意的地方：

```
size_t write_len = min(remaining_capacity(), data.length());
size_t peek_len = min(buffer_size(), len);
size_t pop_len = min(buffer_size(), len);
```

三个len均需要min操作以防止越界访问操作；

```
std::string ByteStream::read(const size_t len) {
    // check len legality in peek_* and pop_* omit redundant check
    string &&read_str = peek_output(len);
    pop_output(len);
    return read_str;
}
```

read的len检查交给调用接口peek和pop，本实现不作重复检查（也可在read中检查，接口实现不检查，但会出现外部接口存在越界风险）； 由此满足了任意外部接口均有越界检查保护（read接口保护通过peek和pop保护实现）； 同时采用右值引用接收peek返回字符串，以减少拷贝操作，提高运行效率；

StreamReassembler成员变量定义如下：

```
class StreamReassembler {
  private:
    // Your code here -- add private members as necessary.
    Index _expect_index{0};  // index that next to enter _output
    size_t _size_unassembled_bytes{0};
    bool _flag_eof{false};  // true when last byte enter _unassembled_bytes
    Index _eof_index{0};    // index of eof
    vector<char> _unassembled_bytes;
    vector<bool> _byte_stored;
    ByteStream _output;  //!< The reassembled in-order byte stream
    size_t _capacity;    //!< The maximum number of bytes
    // push unassembled bytes from data, first byte at index , range[begin,end)
    size_t push_unassembled_bytes(const std::string &data, Index index, Index begin, Index end);
    // push possible assembled bytes into output
    size_t push_output();
}
```

注释详细解释了每个成员变量的作用；额外定义了两个private接口，作用如注释所注；

```
    vector<char> _unassembled_bytes;
    vector<bool> _byte_stored;
```

为存储接收到的无序字节容器和标识每个字节是否有效的bool向量； 需要注意的三个接口实现（难理解的部分已用注释标注）：

```
void StreamReassembler::push_substring(const string &data, const Index index, const bool eof) {
    Index index_bound = _expect_index + _capacity - _output.buffer_size();
    // omit bytes that already assembled
    Index begin = max(index, _expect_index);
    // discard whole data
    if (begin >= index_bound)
        return;
    // omit bytes that out of bound
    Index end = min(index_bound, index + data.length());
    // discard whole data
    if (begin > end)
        return;
    // judge whether last byte enter _unassembled_bytes
    if (eof && end == index + data.length()) {
        _flag_eof = true;
        _eof_index = end;
    }
    // push [begin,end) into _unassembled_bytes
    _size_unassembled_bytes += push_unassembled_bytes(data, index, begin,
end);
    // assemble possible newly contiguous substring
    _size_unassembled_bytes -= push_output();

    if (_flag_eof && empty())
        _output.end_input();
}
size_t StreamReassembler::push_unassembled_bytes(const std::string &data,
Index index, Index begin, Index end) {
    size_t count = 0;
    for (Index cur_index = begin; cur_index < end; ++cur_index) {
        if (!_byte_stored[cur_index % _capacity]) {
            _unassembled_bytes[cur_index % _capacity] = data[cur_index -
index];
            _byte_stored[cur_index % _capacity] = true;
            ++count;
        }
    }
    return count;
}
size_t StreamReassembler::push_output() {
    size_t count = 0;
    if (_byte_stored[_expect_index % _capacity]) {
        size_t i = _expect_index % _capacity;
        string write_str = "";
        while (_byte_stored[i % _capacity]) {
            write_str.push_back(_unassembled_bytes[i % _capacity]);
            _byte_stored[i % _capacity] = false;
            ++count;
            ++i;
        }
        _output.write(write_str);
    }
    _expect_index += count;
    return count;
}
```

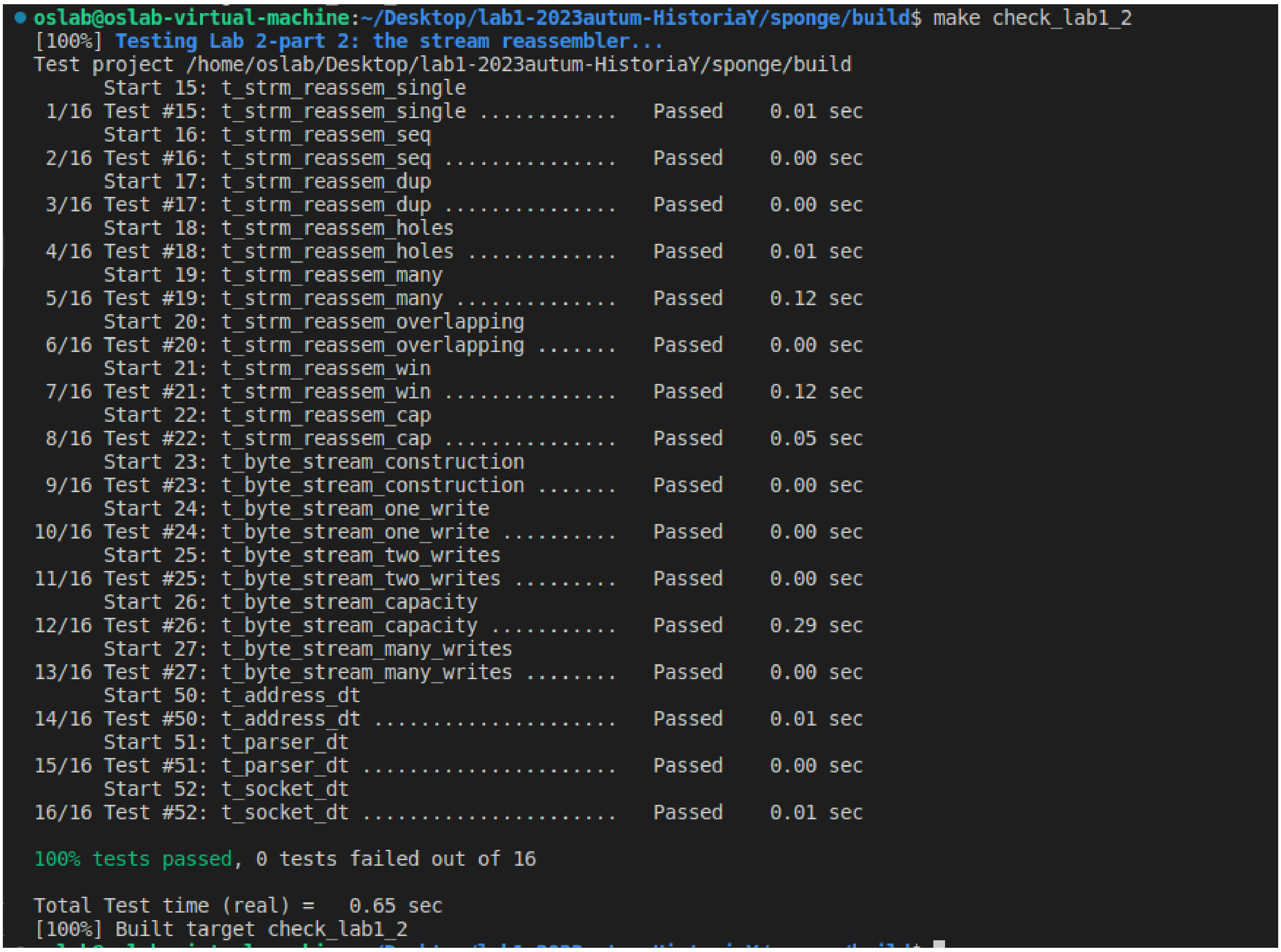
其余部分难度不大，在此不多赘述；

2. Implementation Challenges:

ByteStream实现难度不大，需要注意的地方是，为每个外部接口提供越界访问保护（你永远不知道用户会怎么调用你写的接口）； StreamReassembler难度在于区分assembled和unassembled的两部分字符串，unassembled->assembled的转化过程和eof的判定条件；

3. Remaining Bugs:

本次实验测试样例全部通过，无遗留bug；



```
● oslab@oslab-virtual-machine:~/Desktop/lab1-2023autum-HistoriaY/sponge/build$ make check_lab1_2
  [100%] Testing Lab 2-part 2: the stream reassembler...
  Test project /home/oslab/Desktop/lab1-2023autum-HistoriaY/sponge/build
        Start 15: t_strm_reassem_single
   1/16 Test #15: t_strm_reassem_single ...........   Passed    0.01 sec
        Start 16: t_strm_reassem_seq
   2/16 Test #16: t_strm_reassem_seq ..............   Passed    0.00 sec
        Start 17: t_strm_reassem_dup
   3/16 Test #17: t_strm_reassem_dup ..............   Passed    0.00 sec
        Start 18: t_strm_reassem_holes
   4/16 Test #18: t_strm_reassem_holes ............   Passed    0.01 sec
        Start 19: t_strm_reassem_many
   5/16 Test #19: t_strm_reassem_many .............   Passed    0.12 sec
        Start 20: t_strm_reassem_overlapping
   6/16 Test #20: t_strm_reassem_overlapping .......   Passed    0.00 sec
        Start 21: t_strm_reassem_win
   7/16 Test #21: t_strm_reassem_win ..............   Passed    0.12 sec
        Start 22: t_strm_reassem_cap
   8/16 Test #22: t_strm_reassem_cap ..............   Passed    0.05 sec
        Start 23: t_byte_stream_construction
   9/16 Test #23: t_byte_stream_construction .......   Passed    0.00 sec
        Start 24: t_byte_stream_one_write
  10/16 Test #24: t_byte_stream_one_write ..........   Passed    0.00 sec
        Start 25: t_byte_stream_two_writes
  11/16 Test #25: t_byte_stream_two_writes .........   Passed    0.00 sec
        Start 26: t_byte_stream_capacity
  12/16 Test #26: t_byte_stream_capacity ...........   Passed    0.29 sec
        Start 27: t_byte_stream_many_writes
  13/16 Test #27: t_byte_stream_many_writes ........   Passed    0.00 sec
        Start 50: t_address_dt
  14/16 Test #50: t_address_dt .....................   Passed    0.01 sec
        Start 51: t_parser_dt
  15/16 Test #51: t_parser_dt ......................   Passed    0.00 sec
        Start 52: t_socket_dt
  16/16 Test #52: t_socket_dt ......................   Passed    0.01 sec

  100% tests passed, 0 tests failed out of 16

  Total Test time (real) =   0.65 sec
  [100%] Built target check_lab1_2
```