Lab 1 Writeup
==========

My name: 郑凯琳 TAY KAI LIN

My Student number : 205220025

This lab took me about 10 hours to do. I did attend the lab session.

---

1. Program Structure and Design:

<u>An in-memory reliable byte stream 可靠字节流</u>

思路：
- 实现一个字节流类 `ByteStream`，支持写入 & 读出，有固定的 `capacity`。
- `ByteStream` 与 队列的特性相似，先进先出。
    - 写入时顺序排列在字节流后面
    - 读取时候从头部开始读取并且 `pop` 出来
- `vector` 作为字节流容器，效率比 queue 高。
    - 初始化时直接 `reserve` 分配好空间，防止 `push_back` 时 copy 数据造成额外消耗。
- 写入 & 读取：添加 `_begin` 和 `_end` 两个变量来维护区间。
    - 写入时正常 `push_back` 字符
    - 读取时移动 `_end`
    - 使用 `_begin` 和 `_end` 的时候需对 `_capacity` 取模

核心代码：

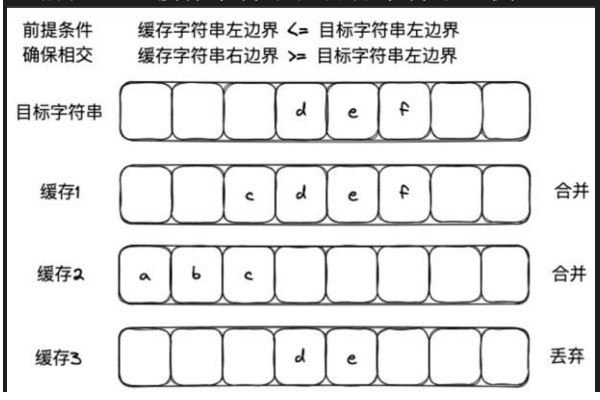
```
    size_t ByteStream::write(const string &data) {
        int write_size = min(data.size(), remaining_capacity());
        for(int i = _begin, j = 0; i < _begin + write_size; i++, j++)
        {
            int real_index = i % _capacity;
            _data[real_index] = data[j];
        }
        _begin += write_size;
        write_cnt += write_size;
        return write_size;
    }

    std::string ByteStream::read(const size_t len) {
        string read_res;
        int read_size = min(buffer_size(), len);
        for(int i = _end; i < _end + read_size; i++)
```

```
        {
            int real_index = i % _capacity;
            read_res += _data[real_index];
        }
        read_cnt += read_size;
        _end += read_size;
        return read_res;
    }

<u>Putting substrings in sequence 流重组器</u>

思路：
- 实现一个 `StreamReassembler` 类 `ByteStream`
    - 保证传入 `ByteStream` （lab1_1）的字节流可靠、有序、不重复
    - 为确定每次 push 进来的字节流的顺序，每个字节流都有一个 `index`
- 合并 & 去重

（1）字节流可能为乱序，不能直接 push 到 `ByteStream` 的 string，先缓存
- 检查是否为预期的字节流：维护一个当前预期接受的 `assemble_idx`，传入的 string 的 `index > assemble_idx` 是预期的字节流
    - 预期字节流：直接 push 到 `ByteStream` 的 string，更新 `assemble_idx`，合并缓存中满足条件的 string
    - 不是预期字节流：缓存，检查能否和缓存的字符串合并

核心代码：

    void StreamReassembler::push_substring(const string &data, const size_t index, const bool eof) {
        if(eof)
            eof_idx = data.size() + index;

        //Non-expected segement: Cache
        if(index > assemble_idx)
        {
            merge_segments(index, data);
            return;
        }

        //Expected segment: Write to ByteStream
        int _begin = assemble_idx - index;
        int write_cnt = data.size() - _begin;
        //no enough space
        if(write_cnt < 0)
            return;
```

```
        assemble_idx += _output.write(data.substr(_begin, write_cnt));

        //Find next segment
        vector<size_t> pop_list;
        for(auto segment : _segments)
        {
            //processed or empty string
            if(segment.first + segment.second.size() <= assemble_idx ||
segment.second.size() == 0)
            {
                pop_list.push_back(segment.first);
                continue;
            }

            //not yet
            if(assemble_idx < segment.first)
                continue;

            _begin = assemble_idx - segment.first;
            write_cnt = segment.second.size() - _begin;
            assemble_idx += _output.write(segment.second.substr(_begin,
write_cnt));
            pop_list.push_back(segment.first);
        }

        //Remove useless segment
        for(auto segment_id : pop_list)
            _segments.erase(segment_id);

        if(empty() && assemble_idx == eof_idx)
            _output.end_input();
    }
```

（2）不是预期字节流：缓存，合并缓存中的字符串

- 情况 1：缓存字符串在目标字符串左侧



- 情况 2：缓存字符串在目标字符串右侧

核心代码：

```
    void StreamReassembler::merge_segments(size_t index, const string& data)
    {
        size_t data_lft = index;
        size_t data_rgt = index + data.size();
        string data_cpy = data;
        vector<size_t> remove_list;
        bool cached = true;

        for(auto segment : _segments)
        {
            size_t seg_lft = segment.first;
            size_t seg_rgt = segment.first + segment.second.size();

            if(data_lft <= seg_lft && data_rgt >= seg_lft)
            {
                if(data_rgt >= seg_rgt)
                {
                    remove_list.push_back(segment.first);
                    continue;
                }
                else if(data_rgt < seg_rgt)
                {
                    data_cpy = data_cpy.substr(0, seg_lft - data_lft) +
segment.second;
                    data_rgt = data_lft + data_cpy.size();
                    remove_list.push_back(segment.first);
                }
            }

            if(data_lft > seg_lft && data_lft <= seg_rgt)
            {
```

```
                if(data_rgt <= seg_rgt)
                    cached = false;
                else if(data_rgt > seg_rgt)
                {
                    data_cpy = segment.second.substr(0, data_lft - seg_lft) +
data_cpy;
                    data_lft = seg_lft;
                    data_rgt = data_lft + data_cpy.size();
                    remove_list.push_back(segment.first);
                }
            }
        }

        //Remove overlapping data
        for(auto remove_idx : remove_list)
            _segments.erase(remove_idx);

        if(cached)
            _segments[data_lft] = data_cpy;
    }
```

2. Implementation Challenges:

问题：调试程序

解决方法：
- 方法 1： 在相应的函数和 test 文件中使用 cout，test 文件若没有输出，表示有问题；在相应的函数 cout 相关变量，观察并追踪其变化
- 方法 2： gdb 调试

3. Remaining Bugs:

...

...

More details and requirements of sections above can be found in `lab1_tutorials.pdf/6.submit`