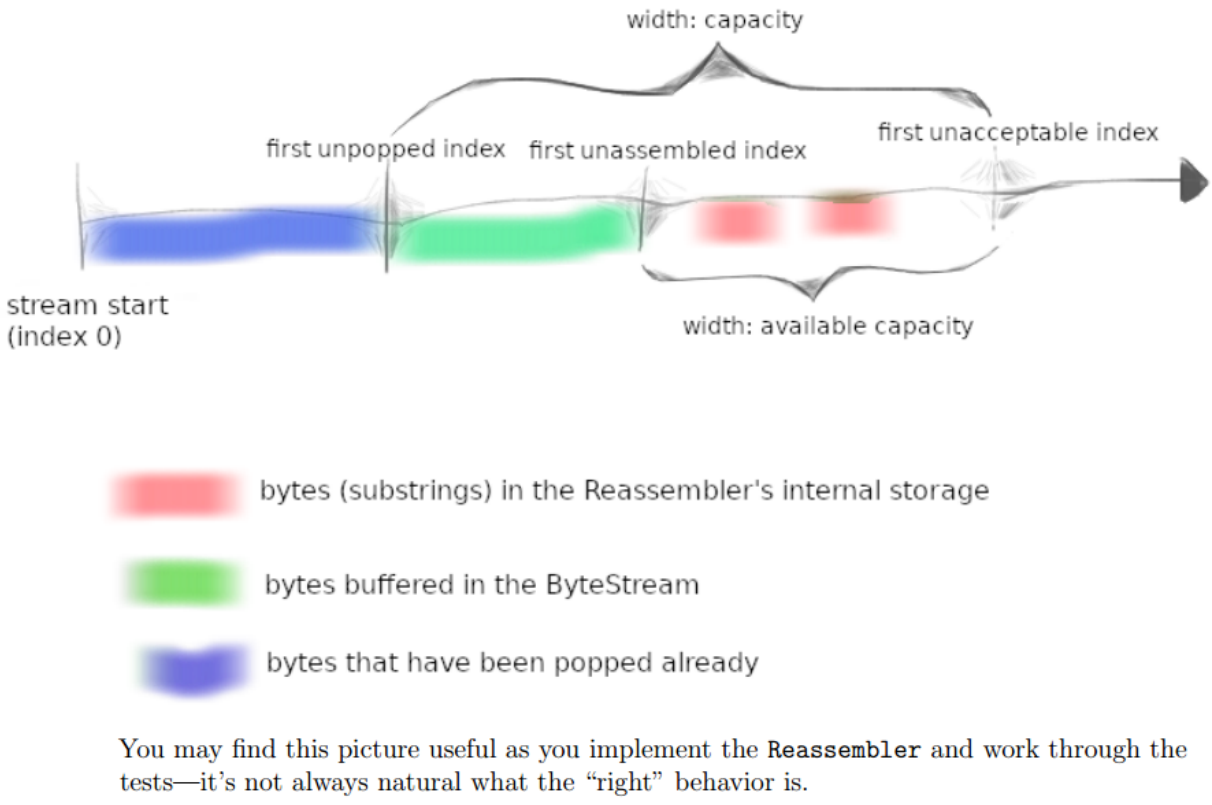
check 1

1. Geting started

2. Puting substrings in sequence

> TCP发送数据时，会将数据分段，每段所包含数据不会超过1460字节，这样可以保证每段数据可以放进数据报中。在网络传输过程中，这些数据报可能会变得无序、或者被丢弃、或着多次发送相同数据报。接收方必须将这些数据报按序重新组装，恢复到与发送时一样的字节流。



You may find this picture useful as you implement the `Reassembler` and work through the tests—it's not always natural what the "right" behavior is.

first unpoppen index:即将pop出的字节索引（ByteSteream）

first unassembled index:即将组装的字节索引

first unacceptablde index:即将接受并存储的字节索引

reassembleer.hh

```
#pragma once
#include "byte_stream.hh"
#include <string>
class Reassembler
{
protected:
```

```
  uint64_t first_unassembled_index;    //首个位组装字节索引
  uint64_t first_unacceptable_index;
  uint64_t end_index;                  //最后的数据段索引

  std::deque<char> assembledBuf;       //存储提前到来的字符串
  std::deque<bool> flagBuf;            //字符是否有效
public:

  void insert( uint64_t first_index, std::string data, bool is_last_substring,
Writer& output );

  // How many bytes are stored in the Reassembler itself?
  uint64_t bytes_pending() const;
 //init Reassembler()
 Reassembler():first_unassembled_index(0)
                              ,first_unacceptable_index(0)
                              ,end_index(-1)
                              ,assembledBuf()
                              ,flagBuf()
                              {}
};
```

reassembleer.cc

```
#include "reassembler.hh"

using namespace std;
void Reassembler::insert( uint64_t first_index, string data, bool
is_last_substring, Writer& output )
{
  // Your code here.
  //初始化assembledBuf 和 flagBuf
  assembledBuf.resize(output.available_capacity(),0);
  flagBuf.resize(output.available_capacity(),0);

  //是否是最后的子串
  if(is_last_substring){
    //计算最后字节索引
    end_index=first_index+data.size();
  }
  //计算未组装首索引和未接受首索引
  first_unassembled_index=output.bytes_pushed();

 first_unacceptable_index=first_unassembled_index+output.available_capacity();

  //处理data
  uint64_t str_begin;
  uint64_t str_end;
```

```cpp
  uint64_t str_len=first_index+data.size();
  if(!data.empty()){
    if(str_len<first_unassembled_index||first_index>=first_unacceptable_index)
{
      //数据已经有过或着数据索引超出接受范围
      data="";
    }else{
      str_begin=first_index;
      str_end=str_len-1;    //索引从0开始
       //去头，头部分数据已经进入ByteStream
      if(first_index<first_unassembled_index)
str_begin=first_unassembled_index;
      //去尾，尾部分数据超出可接受范围
      if(str_len>first_unacceptable_index) str_end=first_unacceptable_index-1;

      for(auto i=str_begin;i<=str_end;i++){
        assembledBuf[i-first_unassembled_index]=data[i-first_index];
        flagBuf[i-first_unassembled_index]=true;
      }
    }
  }
  //pop
  string str="";
  while(flagBuf.front()){
    str+=assembledBuf.front();
    assembledBuf.pop_front();       //del a space
    assembledBuf.push_back(0);      //add a space
    flagBuf.pop_front();            //del a space
    flagBuf.push_back(false);       //add a space
  }
  output.push(str);                 //push into stram
  //已经push的字节数等于最后一个字节索引，说明push完毕
  if(output.bytes_pushed()==end_index){
    output.close();
  }

  (void)first_index;
  (void)data;
  (void)is_last_substring;
  (void)output;
}
uint64_t Reassembler::bytes_pending() const
{
  // Your code here.
  uint64_t count=0;
  for(auto i=flagBuf.begin();i!=flagBuf.end();i++){
    if(*i==true)
      count++;
```

```
  }
  return count;
}

```

the result

```
Test project /home/sgt/cs/minnow/build
      Start  1: compile with bug-checkers
 1/17 Test  #1: compile with bug-checkers ........   Passed    3.20 sec
      Start  3: byte_stream_basics
 2/17 Test  #3: byte_stream_basics ...............   Passed    0.01 sec
      Start  4: byte_stream_capacity
 3/17 Test  #4: byte_stream_capacity .............   Passed    0.01 sec
      Start  5: byte_stream_one_write
 4/17 Test  #5: byte_stream_one_write ............   Passed    0.02 sec
      Start  6: byte_stream_two_writes
 5/17 Test  #6: byte_stream_two_writes ...........   Passed    0.02 sec
      Start  7: byte_stream_many_writes
 6/17 Test  #7: byte_stream_many_writes ..........   Passed    0.07 sec
      Start  8: byte_stream_stress_test
 7/17 Test  #8: byte_stream_stress_test ..........   Passed    0.49 sec
      Start  9: reassembler_single
 8/17 Test  #9: reassembler_single ...............   Passed    0.02 sec
      Start 10: reassembler_cap
 9/17 Test #10: reassembler_cap ..................   Passed    0.01 sec
      Start 11: reassembler_seq
10/17 Test #11: reassembler_seq ..................   Passed    0.03 sec
      Start 12: reassembler_dup
11/17 Test #12: reassembler_dup ..................   Passed    0.05 sec
      Start 13: reassembler_holes
12/17 Test #13: reassembler_holes ................   Passed    0.02 sec
      Start 14: reassembler_overlapping
13/17 Test #14: reassembler_overlapping ..........   Passed    0.02 sec
      Start 15: reassembler_win
14/17 Test #15: reassembler_win ..................   Passed    5.29 sec
      Start 28: compile with optimization
15/17 Test #28: compile with optimization ........   Passed    1.43 sec
      Start 29: byte_stream_speed_test
             ByteStream throughput: 0.46 Gbit/s
16/17 Test #29: byte_stream_speed_test ...........   Passed    0.36 sec
      Start 30: reassembler_speed_test
             Reassembler throughput: 0.36 Gbit/s
17/17 Test #30: reassembler_speed_test ...........   Passed    0.64 sec

100% tests passed, 0 tests failed out of 17

Total Test time (real) =  11.70 sec
```

```
42 Built target check1
```