Assignment 5 Designing ADTs

1. 日期运算

题目描述

使用c++的类设计并实现一个ADT来表达公历日期，具体表现为类 `Date`，需要支持以下功能：

- 默认构造函数：将日期设置为1900年1月1日。
- 构造函数：以年月日为输入来初始化日期，例如：

  ```
  Date moonLanding(1969,7,20);
  ```

- getter方法 `getDay`，`getMonth` 和 `getYear`： `moonLanding.getDay()` = 20, `moonLanding.getMonth()` = 7, `moonLanding.getYear()` = 1969。
- `ToString` 方法: 返回形如 `dd-mmm-yyyy` 的字符串，其中 `dd` 为1-2位数字的日期， `mmm` 为三个字母的英语月份缩写， `yyyy` 为固定4位数字的年份。调用 `moonLanding.toString()` 应该返回字符串"20-Jul-1969"
- `add` 方法：以 `int n` 为参数，将当前日期向后移动n天，例如运行

  ```
  moonLanding.add(5);
  ```

  后 `moonLanding.toString()` 应该返回字符串"25-Jul-1969"
- `sub` 方法：以 `int n` 为参数，将当前日期向前移动n天，例如运行

  ```
  moonLanding.sub(5);
  ```

  后 `moonLanding.toString()` 应该返回字符串"15-Jul-1969"
- `diff` 方法：以另一个 `Date d` 为参数，返回两个日期之间相差的天数,不考虑两个日期的前后关系，即有 `d1.diff(d2) = d2.diff(d1)`。

输入输出格式

本题中你的任务是实现一个完整的ADT，而非一个具体的程序。因此没有固定输入输出格式。我们在main.cpp中给出了一部分测试程序。我们建议你自行编写main函数来测试并修正你的实现，之后使用我们的测试程序结合judger做最终测试。

最终考核给分的形式与此类似，你只需要提供 `Date.cpp` 和 `Date.h` 文件，我们将采用同一的 `main.cpp` 进行测试。

数据范围和异常处理

要求设计的类能够表达从1900年1月1日到2900年12月31日之间的所有日期。当 `add` 或 `sub` 计算的结果超出范围或初始化的输入不合法时统一输出"Error"并换行，并将日期置为默认值1900年1月1日。此规定只是为了便于检查例外情况的发生，不会出现刻意利用由此得到的默认值进行进一步计算的用例。

提示

在定义ADT的过程中，建议分批完成成员函数的实现与测试。具体来说可以先在`Date.cpp`中实现一部分成员函数，并在`Date.h`中注释其余暂未实现的成员函数声明以便在在`main.cpp`中测试。确认实现正确后再实现其余成员函数。

2. 比特串数据传输发送者

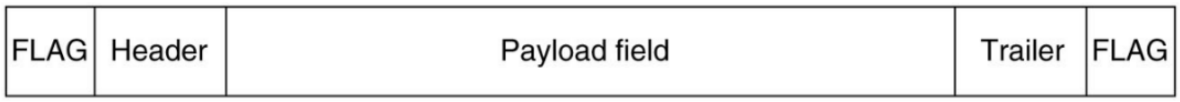
题目描述

原始数据的处理

在计算机网络的数据传输过程中，数据发送方需要将比特串传输给另一个端点 。然而，发送方并不是直接传输原始数据，而是需要将原始数据分割并封装成一个个“ **帧(frame)** ”，以帧为单位进行传输。

其中，每一帧中数据的大小不是无限的，**最大传输单元(MTU)** 规定了一帧中数据部分的最大长度。当需要传输的比特流长度大于这个数值，就需要将数据划分为多个传输单元进行传输。在本题中，MTU默认大小为128位，用户也可以修改该值。例如，若修改MTU为8位，对于原始数据`001111111100`，需要将它分割为两个传输单元：`00111111`和`1100`。

由于发送的数据是连续的比特流，对于每一个传输单元，需要在它的起始处和结束处插入 **界定符(Flag)** ，将它封装为帧。在点对点的传输协议中，这个界定符就是比特串`01111110`。需要注意的是，连续两帧之间只需要一个界定符，如果出现连续两个界定符，说明这是一个需要被丢弃的空帧。除了我们真正要传输的数据(Payload)，在帧的头部(Header)和尾部(Trailer)，还会有一些控制信息。为了将帧的结构进行简化，本题中我们 **忽略** 帧头部和尾部的控制信息。



在上述的例子中，两个传输单元会被封装为`01111110 00111111 01111110 1100 01111110`。

从该例子可以发现，使用界定符进行封装有一个明显的问题：我们传输的数据中，也可能会出现`01111110`这个比特串，如果直接在传输单元的起始和结束处加上界定符，就可能让接收方错误地对数据帧进行拆分。为了防止数据中出现Flag序列，可以采用 **比特填充(bit stuffing)** 的方法来对原始数据进行预处理。当发送方遇到数据比特流中出现5个连续“1”的时候，它就自动在比特流中插入一个“0”，这样就不可能包含Flag中连续的6个“1”了。当然，接收方得到这些数据后，还要将这些冗余的“0”删去。

原始数据`001111111100`经过比特填充处理后变为`0011111011100`，分割为两个传输单元`00111110`和`11100`，插入Flag之后封装为`01111110 00111110 01111110 11100 01111110`。

发送方Sender类的设计

我们的任务是设计C++类`Sender`，来实现对原始数据的比特填充、分割和封装功能。

简而言之，我们的任务是：

- 设定`Sender`类的成员字段，需要包括
    - 原始数据
    - 最大传输单元MTU
- 设定并实现`Sender`类的成员方法，需要包括
    - 初始化构造方式。若初始化时没有规定最大传输单元MTU，它将默认为128位。若没有传入任何原始数据，则原始数据流为空。

- `string getOriginData()`方法，用来获取原始数据。
- `string bitStuffing (string data)`方法，对数据部分进行比特填充。
- `string framing()`方法，将经过比特填充处理后的数据划分为传输单元，并添加界定符封装为帧。

输入输出格式

同上题，需要实现Sender类，而非一个具体的程序，在main.cpp中给出了一部分测试程序，用来结合judger做最终测试。

提交时只需要提供`Sender.cpp`和`Sender.h`文件。

数据范围和异常处理

- MTU的值在unsigned int范围内。
- 原始数据是一个比特串，其长度在unsigned int范围内。
- 原始数据流为空时，待传输数据是一个无效的空帧，即两个连续的界定符。

提交格式

你提交的文件结构应该类似如下形式：

```
<your student number>.zip
|- 1_date
|  |- Date.cpp
|  |- Date.h
|
|- 2_transfer
|  |- Sender.cpp
|  |- Sender.h
|
|- cs1604.txt (incldue it if you use StanfordCppLib)
```