2 A Naive Packet

需要提交的文件：*packet.h*。

背景

> IP 是 **互联网协议**（internet protocol）的简称，是 TCP/IP 协议栈中的网络层协议。IP 协议在发展的过程中，衍生出 IPv4 和 IPv6 两个不同版本。其中，历史版本 IPv4 目前仍广泛使用；后继版本 IPv6 世界各地正在积极部署。
>
> IP 协议的通信单元是 **IP 包**（packet），同样分为 IPv4 和 IPv6 两个版本。它应该分为头部（header）和数据两大部分；其中头部应该包含源地址、目的地址以及数据类型等字段。

题目描述

> IPv4 地址由 32 位二进制数组成。为了方便记忆，将 IPv4 地址 32 位进制数进行分段，每段 8 位，共 4 段，然后将每段 8 位二进制数转换为十进制数，中间用 “.” 分割。这种表达方式称之为 **点分十进制**。
>
> IPv6 地址由 128 位二进制数组成，采用十六进制表示法，分 8 组表示，每组 16 位二进制数换算为 4 位十六进制数。各组之间用 “:” 号隔开，例如：1080:0000:0000:0000:0008:0800:200c:417a。
>
> 定义 0000 为零组。在 IPv6 地址段中有时会出现连续 n 组零组（$n \geq 1$），为了简化书写，这些 0 可以用 “::” 代替，但一个地址中只能出现一次 “::”。例如，上面的地址可以被简化为 1080::0008:0800:200c:417a。

现在，助教定义了 IP 地址的最简形式以及标准形式。

- 最简形式：
  - 对于 IPv4 地址，其处于最简形式当且仅当每段十进制数不含前导零。如：`127.0.0.1`。
  - 对于 IPv6 地址，其处于最简形式当且仅当每段十六进制数都由 4 位十六进制数表示，且不存在零组。如：`abcd:00a1::`。
- 标准形式：
  - 对于 IPv4 地址，其处于标准形式当且仅当每段十进制数都由 3 位十进制数表示。如：`127.000.000.001`。
  - 对于 IPv6 地址，其处于标准形式当且仅当每段十六进制数都由 4 位十六进制数表示，且不含 `::`。如：`abcd:0000:0000:0000:0000:0000:0000:0000`。

注意，十六进制数中大于 9 的数均应使用小写字母表示。

你的任务是实现一个简易的 Packet，示意图如图 1。

你需要完成：

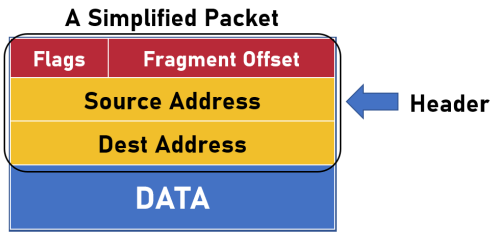

图 1: A Simplified Packet

构造函数与析构函数

你需要完成 `Packet`, `IPv4Packet`, `IPv6Packet` 类的构造函数与析构函数。构造函数参数共三个，类型均为 `const char*`，按顺序传入：

- **最简形式**的源地址 `src`，
- **最简形式**的目标地址 `dest`，
- 数据内容 `data`。

保证此三个参数对应的字符串在评测程序中不会被修改。

IP 包的分片函数

你需要完成 `IPv4Packet`，`IPv6Packet` 类的 `Segment(int MTU)` 函数。

> MTU (Maximum Transmission Unit) 是指网络支持的最大传输单元，以 **字节**为单位。MTU 的大小决定了发送端一次能够发送报文的最大字节数。
> IP 包的分片 (fragmentation) 是指将大的 IP 数据包分割成更小的单位。如果 IP 包的数据长度超过了 MTU，就要进行分片，使分片后的每个片段大小不超过 MTU。
> IPv4 允许路由器对超大的 IP 数据包进行分片，但 IPv6 不允许路由器执行分片操作。

对于 `IPv6Packet`，你 **只需要** 输出 `IPv6Packet is already segmented.\n`。
对于 `IPv4Packet`，你需要将 `char* data` 分为若干片，每片按如下格式输出：

```
Fragment <fragment>: Size <size>; Header Size <header_size>; Data Size <data_size>;
Flag <flags>; Fragment Offset <offset>; Data <segmented_data>.\n
```

- `<fragment>`：该分段的序号（从 0 开始）。
- `<size>`：该分段的大小（等于 `head_size` 与 `data_size` 的和）。
- `<header_size>`：头部的大小，默认为 20。
- `<data_size>`：`<segmented_data>` 的长度。
- `<flags>`：表示该分段是否为最后一段是否为中间段，中间段用 1 表示，最后一段用 0 表示。
- `<offset>`：表示该段在原数据中的偏移量，以 **8 字节**为单位。简单来说，设原报文为 `char* data`，那么该分段起始地址为 $data + offset \times 8$。因此，**除最后一段分段外**，所有分段的 `data_size` 应为 8 的倍数。为最大化利用一次能够传输报文的长度，除最后一段报文外，你应该取 `data_size` 为 **最大**的可能值。
- `<segmented_data>`：该分段所包含的数据，为字符串。

本题中 header size 均为 20。保证传入的 MTU 合法。
下面举一个具体的例子解释分段过程。对于以下的代码：

```
Packet* p = new IPv4Packet("192.168.0.1", "192.168.0.1", "0000000001111111111");
p->Segment(30);
```

MTU 为 30，减去 header 占用的 20，得出 data size 不超过 10，无法容纳长度为 20 的 data，故需要分段。
除最后一段分段外 data size 应为 8 的倍数，故在此例中，受 MTU 限制，data size 的最大可能值为 8。
由上述可知，data 共被分为三段，长度分别为 8，8，4。
对于前两段，为中间段，故 flag 为 1；对于最后一段，非中间段，故 Flag 为 0。
Offset 分别为每一段起始地址除以 8 的结果，即：

- `0 / 8 = 0`;
- `8 / 8 = 1`;
- `16 / 8 = 2`。

Data 为每一段分段所包含的内容。

- 对于第一段，起始地址为 0，长度为 8，故包含内容为 `00000000`。
- 对于第二段，起始地址为 8，长度为 8，故包含内容为 `00111111`。
- 对于第三段，起始地址为 16，长度为 4，故包含内容为 `1111`。

因此，输出结果应该是：

```
Fragment 0: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 0; Data 00000000.
Fragment 1: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 1; Data 00111111.
Fragment 2: Size 24; Header Size 20; Data Size 4; Flag 0; Fragment Offset 2; Data 1111.
```

打印函数

你需要完成 `IPv4Packet`，`IPv6Packet` 类的 `Print()` 函数。

- 首先，你应该调用 `Packet` 类的 `Print()` 函数；
- 其次，你应该分别输出（**标准形式**的 `src`，`dest`）以及 `data`，中间以 `'\t'` 作为分隔符；
- 最后，你应该输出换行符 `'\n'`。

提示 你可能需要以下工具帮助你完成本题。你可以点击以下超链接来访问这些网站。
注：如果选用 C 风格处理方式，注意在进行 `new` 操作时需要多 `new` 出一位以保存 `'\0'`，否则在调用下列 C-style 函数时可能会出现 Invalid read/write 的问题，从而导致 memory leak。

- C-style
  - strlen, strnlen_s - cppreference.com
  - strcat, strcat_s - cppreference.com
  - strcpy, strcpy_s - cppreference.com
  - strtok, strtok_s - cppreference.com

  - memset, memset_explicit, memset_s - cppreference.com
  - putchar - cppreference.com
- std::string
  - std::basic_string - cppreference.com
- std::stringstream
  - std::basic_stringstream - cppreference.com

你需要使用以下模板完成此题。你不应该修改模板已经存在的任何内容（标注内容除外）；你不应该包含其他头文件；你不应该新增 `Packet` 类的成员变量。保证所有输入均合法。

```
#ifndef PACKET_H_
#define PACKET_H_

#include <iostream>
#include <sstream>
#include <cstring>
#include <vector>

class Packet {
 private:
  // You should let ... be whatever you want.
  using T = ...;
  T src;
  T dest;
  char* data;

 public:
  // TODO: constructor and destructor
  const T& GetSrc() const { return src; }
  const T& GetDest() const { return dest; }
  const char* GetData() const { return data; }
  virtual void Segment(int MTU) = 0;
  virtual void Print() const {
    std::cout << "Print start..." << std::endl;
  }
};

class IPv4Packet : public Packet {
 private:
  // Do whatever you want
 public:
```

```
  // TODO
};

class IPv6Packet : public Packet {
 private:
  // Do whatever you want
 public:
  // TODO
};

#endif // PACKET_H_
```

样例测试程序 1

```
#include "packet.h"
int main() {
  Packet* p = nullptr;
  p = new IPv4Packet("192.168.0.1", "192.168.0.1", "00000001111111");
  p->Print();
  delete p;
  p = new IPv6Packet("1080::0008:0800:200c:417a", "1080::0008:0800:200c:417a",
    "00000001111111");
  p->Print();
  delete p;
  return 0;
}
```

样例测试程序 2

```
#include "packet.h"
int main() {
  Packet* p = nullptr;
  p = new IPv4Packet("192.168.0.1", "192.168.0.1", "000000000011111111111");
  p->Segment(30);
  delete p;
  p = new IPv6Packet("1080::0008:0800:200c:417a", "1080::0008:0800:200c:417a",
    "000000000011111111111");
  p->Segment(30);
  delete p;
  return 0;
}
```

样例输出 1

```
Print start...
192.168.000.001        192.168.000.001        00000001111111
Print start...
1080:0000:0000:0000:0008:0800:200c:417a        1080:0000:0000:0000:0008:0800:200c:417a
00000001111111
```

样例输出 2

```
Fragment 0: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 0; Data 00000000.
Fragment 1: Size 28; Header Size 20; Data Size 8; Flag 1; Fragment Offset 1; Data 00111111.
Fragment 2: Size 24; Header Size 20; Data Size 4; Flag 0; Fragment Offset 2; Data 1111.
IPv6Packet is already segmented.
```

测试点说明

测试点	测试函数	内存泄漏检查	分数
1	Test for `IPv4Packet: Print`	无	8
2	Test for `IPv6Packet: Print`	无	12
3	Harder test for `IPv6Packet: Print`	无	12
4	Test for: `Segment`	无	24
5	Test for: `Segment`	无	8
6	All	无	8
7	All	无	8
8–14	同 1–7	有	20