

计算机网络编程

第3章 Ethernet帧的封装与解析

信息工程学院 方徽星
fanghuixing@hotmail.com

大纲

- 设计目的
- 相关知识
- 例题分析

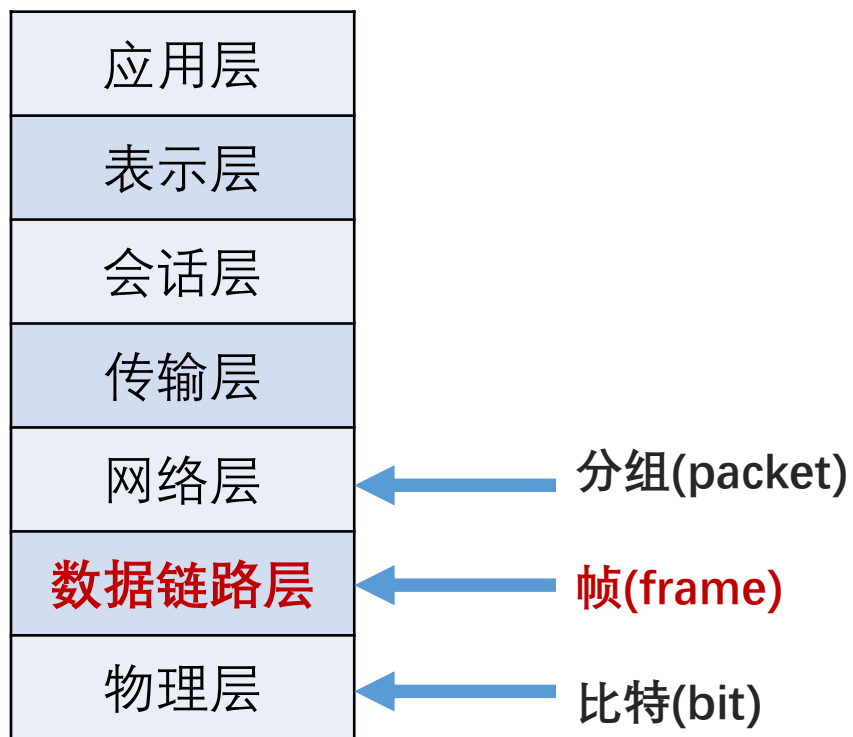
1. 设计目的

- 帧是在数据链路层进行数据传输的基本单元
- 目的：
 - 根据数据链路层的基本原理，通过封装标准格式的Ethernet帧，
 - 了解Ethernet帧结构中各字段的含义与用途，
 - 从而深入理解网络协议的工作原理

2. 相关知识

- 数据链路层的概念

OSI参考模型



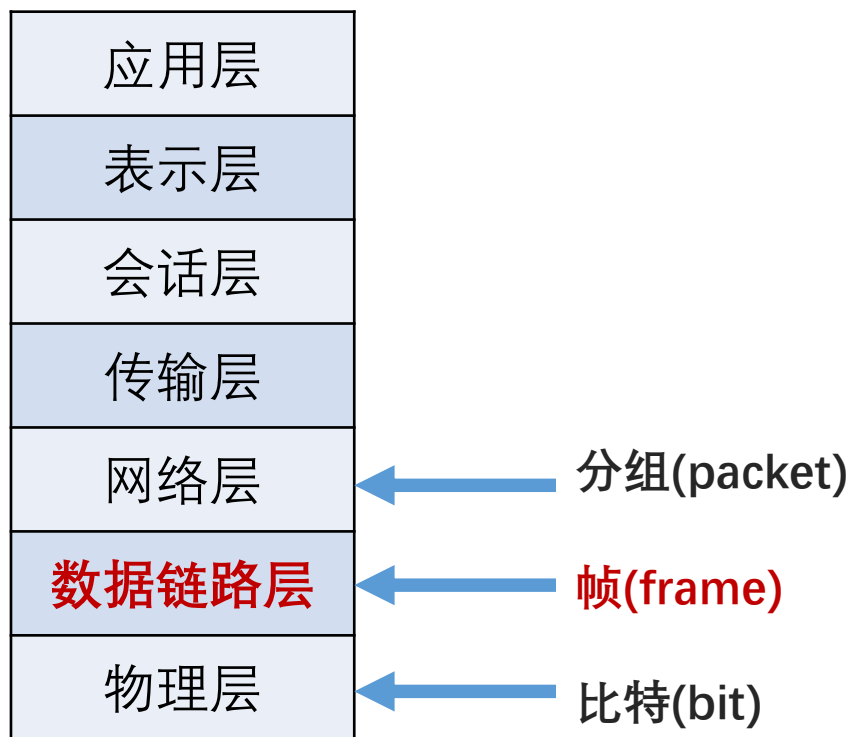
数据链路层(Data Link Layer)的主要功能：

- 在**物理层**提供的服务**基础上**在主机之间**建立数据链路**,
- 传输**以帧为单位**的数据包,
- 采取**差错控制**与**流量控制**的方法, 将有差错的物理连接**变成无差错的数据链路**
- 为上层提供服务, **屏蔽**各种物理网络以及传输介质的**差异性**

2. 相关知识

- 数据链路层的概念

OSI参考模型

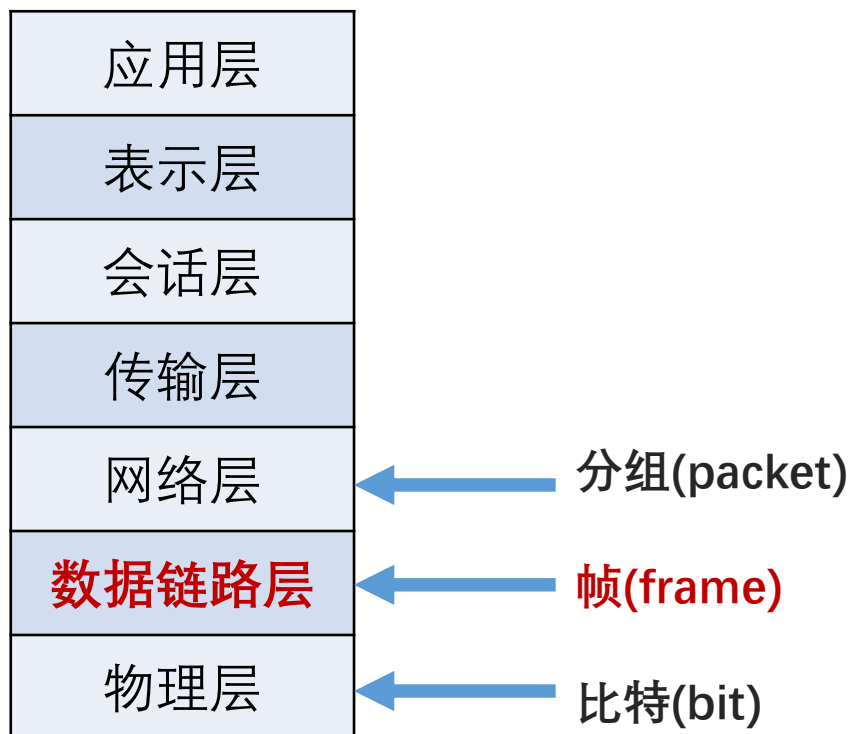


- 1980年，IEEE 802委员会成立，制定了针对不同类型局域网的IEEE 802标准
- IEEE 802.3标准是以太网(Ethernet)的协议标准，覆盖数据链路层和网络层两部分
- 数据链路层标准主要描述介质访问控制问题
- 网络层标准主要描述使用的传输介质，如：双绞线、光纤、同轴电缆等

2. 相关知识

- 数据链路层的概念

OSI参考模型



<http://grouper.ieee.org/groups/802/3/>

[IEEE 802\(R\): Overview and Architecture](#)

[IEEE 802.1: Bridging and Management](#)

[IEEE 802.3: Ethernet](#)

[IEEE 802.11: Wireless LANs](#)

[IEEE 802.15: Wireless PANs](#)

[IEEE 802.16: Broadband Wireless MANs](#)

[IEEE 802.19: TV White Space Coexistence Methods](#)

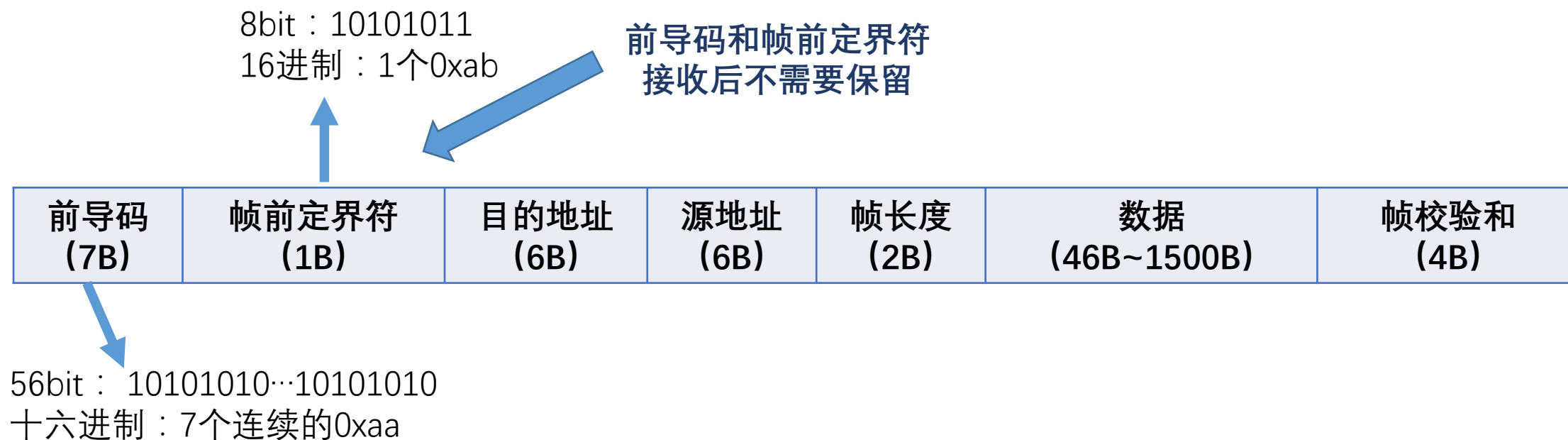
[IEEE 802.21: Media Independent Handover Services](#)

[IEEE 802.22: Wireless Regional Area Networks](#)

<https://ieeexplore.ieee.org/browse/standards/get-program/page/series?id=68>

2. 相关知识

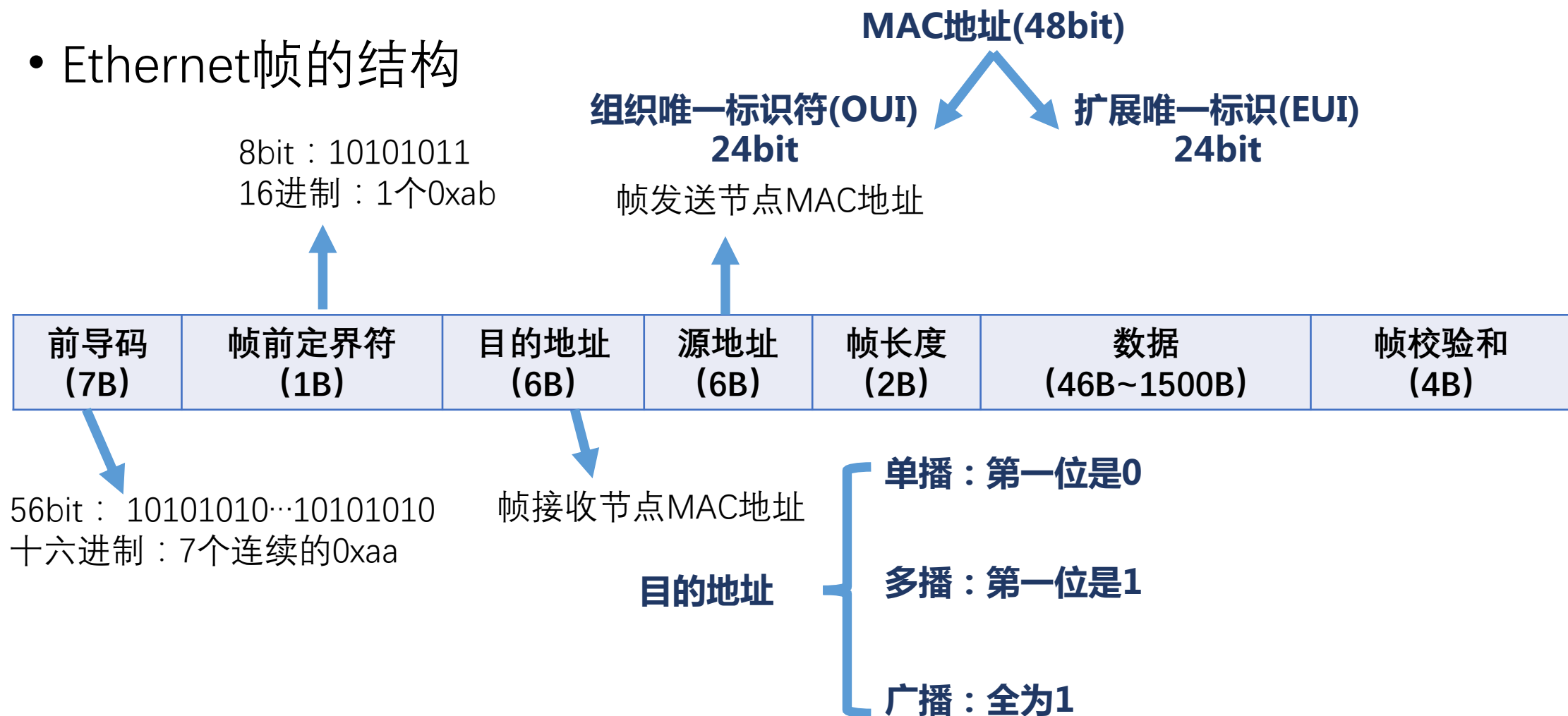
- Ethernet帧的结构



发送节点在发送数据的前后各添加特殊的字符构成帧，这些特殊的字符是帧头与帧尾

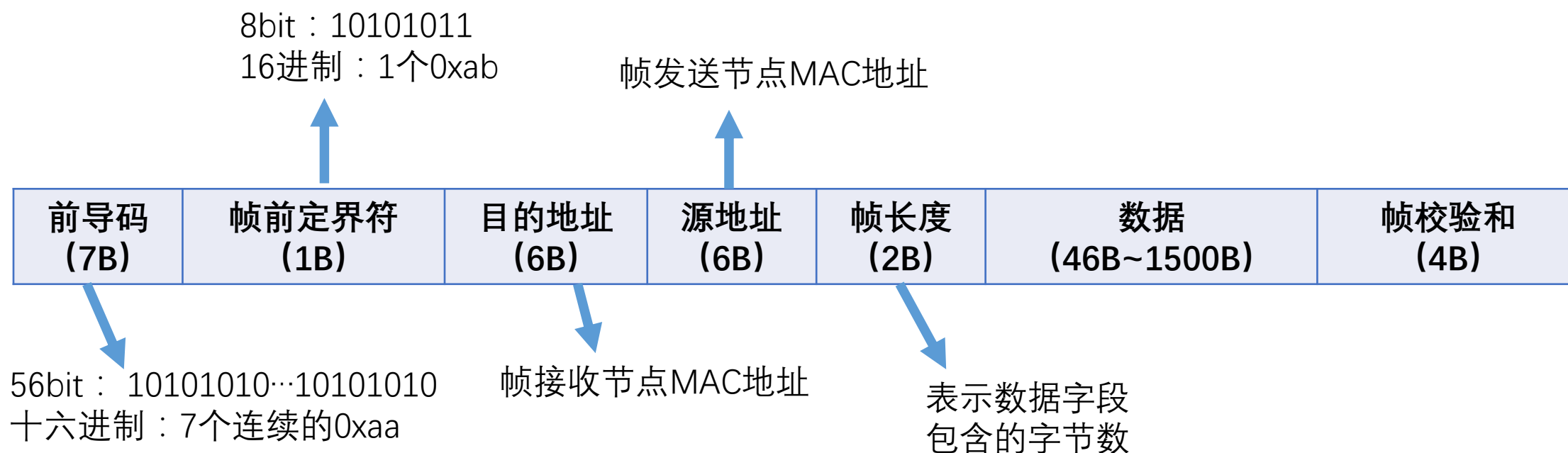
2. 相关知识

- Ethernet帧的结构



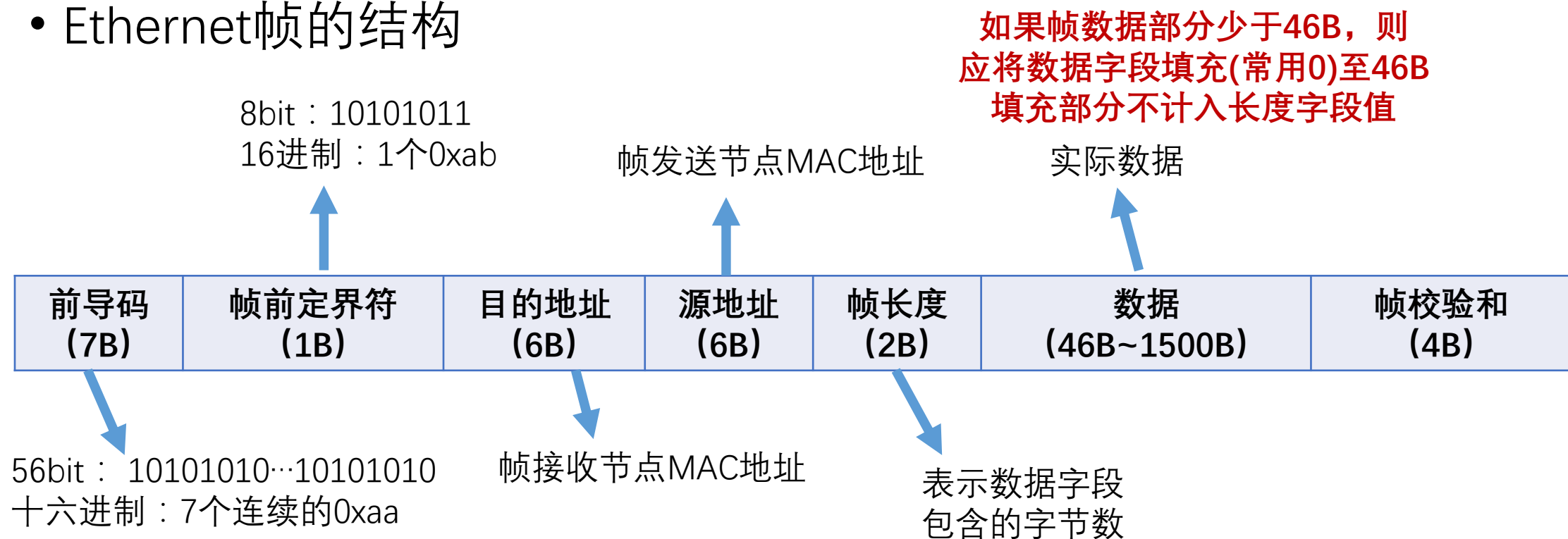
2. 相关知识

- Ethernet帧的结构



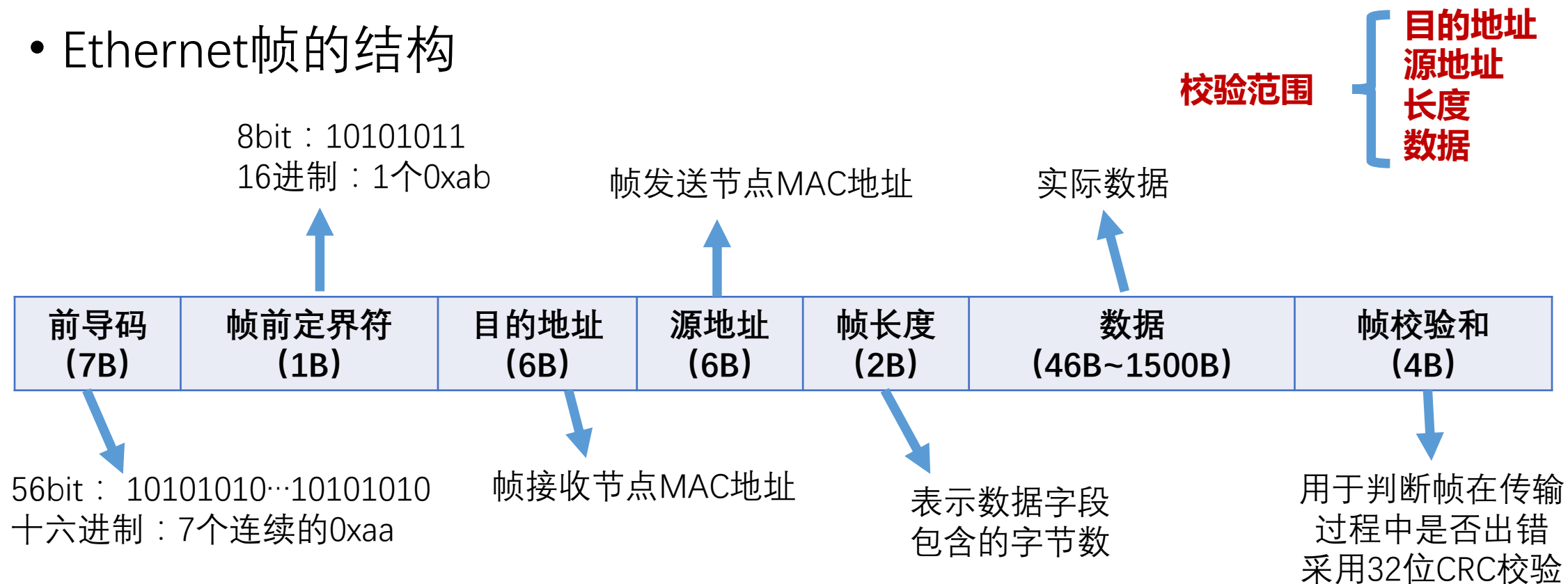
2. 相关知识

- Ethernet帧的结构



2. 相关知识

- Ethernet帧的结构



3. 例题分析

- 例题

- 根据给出的IEEE 802.3格式的Ethernet帧结构，编写程序解析并显示帧的各个字段，将数据字段组合写入输出文件
- Ethernet帧数据从输入文件中获得，默认的输入文件为二进制数据文件
- 只读取帧校验字段，不就行CRC校验
- 数据字段最大长度100B

3. 例题分析

- 具体要求

1. 程序为命令程序，可执行文件名
FrameParse.exe

命令行格式

FrameParse input_file output_file

↑
输入文件

↑
输出文件

2. 将全部字段内容显示在控制台上，
具体格式为：

帧1开始解析

前导码：xx xx xx xx xx xx xx

帧前定界符：xx

目的地址：xx-xx-xx-xx-xx-xx

源地址：xx-xx-xx-xx-xx-xx

长度字段：xx xx

数据字段：...

帧校验字段：xx xx xx xx

帧2开始解析

...

数据字段按字符串格式输出
其他字段按16进制格式输出

3. 例题分析

- 具体要求

- 3. 良好的编程规范与注释

- 4. 撰写说明文档，包括

- ① 程序的开发思路

- ② 工作流程

- ③ 关键问题

- ④ 解决思路

- ⑤ 进一步改进

3. 例题分析

- 关键问题：文件的读写操作

C++ 标准库 fstream 定义了三个数据类型

数据类型	描述
ofstream	表示输出文件流，用于创建文件并向文件写入信息
ifstream	表示输入文件流，用于从文件读取信息
fstream	通常表示文件流，且同时具有 ofstream 和 ifstream 两种功能，它可以创建文件，向文件写入信息，从文件读取信息

3. 例题分析


- 关键问题：文件的读写操作

打开文件

- 在从文件读取信息或者向文件写入信息之前，必须先打开文件。
- ofstream 和 fstream 对象都可以用来打开文件进行写操作，
- 如果只需要打开文件进行读操作，则使用 ifstream 对象。

open() 函数是 fstream、ifstream 和 ofstream 对象的一个成员：

void open(const char *filename, ios::openmode mode);



要打开的文件的
名称和位置



文件被打开的模式

3. 例题分析

- 关键问题：文件的读写操作

模式标志	描述
<code>ios::app</code>	追加模式。所有写入都追加到文件末尾
<code>ios::ate</code>	文件打开后定位到文件末尾
<code>ios::binary</code>	以二进制的方式对打开的文件进行读写
<code>ios::in</code>	打开文件用于读取
<code>ios::out</code>	打开文件用于写入
<code>ios::trunc</code>	如果该文件已经存在，其内容将在打开文件之前被截断，即把文件长度设为 0

可以把两种或两种以上的模式结合使用：

```
ofstream outfile;
```

```
outfile.open("file.dat", ios::out | ios::trunc );
```

打开文件的方式在类ios(是所有流式I/O类的基类)中定义

3. 例题分析

- 关键问题:文件的读写操作

文件位置指针：一个整数值，指定了从文件的起始位置到指针所在位置的字节数

istream 和 **ostream** 都提供了用于重新定位文件位置指针的成员函数：

- 关于 **istream** 的 **seekg** ("seek get")

- 关于 **ostream** 的 **seekp** ("seek put")

seekg 和 **seekp** 的参数通常是一个长整型。第二个参数可以用于指定查找方向：

- **ios::beg** (默认的，从流的开头开始定位)

- **ios::cur** (从流的当前位置开始定位)

- **ios::end** (从流的末尾开始定位)

3. 例题分析

- 关键问题:文件的读写操作

文件位置指针：一个整数值，指定了从文件的起始位置到指针所在位置的字节数

// 定位到 fileObject 的第 n 个字节（假设是 ios::beg）

fileObject.seekg(n);

// 把文件的读指针从 fileObject 当前位置向后移 n 个字节

fileObject.seekg(n, ios::cur);

// 把文件的读指针从 fileObject 末尾往回移 n 个字节

fileObject.seekg(n, ios::end);

// 定位到 fileObject 的末尾

fileObject.seekg(0, ios::end);

3. 例题分析

- 关键问题：文件的读写操作

文件操作函数	描述
<code>file.read()</code>	从指针位置读取指定字节的数据
<code>file.write()</code>	向指针位置写入指定字节的数据
<code>file.get()</code>	从指针位置读取1B的数据
<code>file.put()</code>	向指针位置写入1B的数据
<code>file.tellg()</code>	获得指针位置的偏移量

3. 例题分析

- 关键问题：文件的读写操作

```
// read a file into memory
#include <iostream>    // std::cout
#include <fstream>     // std::ifstream
using std::ios;
using std::ifstream;
using std::cout;
int main() {
    ifstream is("test.txt", ios::binary);
    if (is) {
        printf("file opened!\n");
        // get length of file:
        is.seekg(0, is.end);
        int length = is.tellg();
        is.seekg(0, is.beg);
```

```
        // allocate memory:
        char * buffer = new char[length];

        // read data as a block:
        is.read(buffer, length);
        is.close();
        // print content:
        cout.write(buffer, length);
        delete[] buffer;
    }
    return 0;
}
```

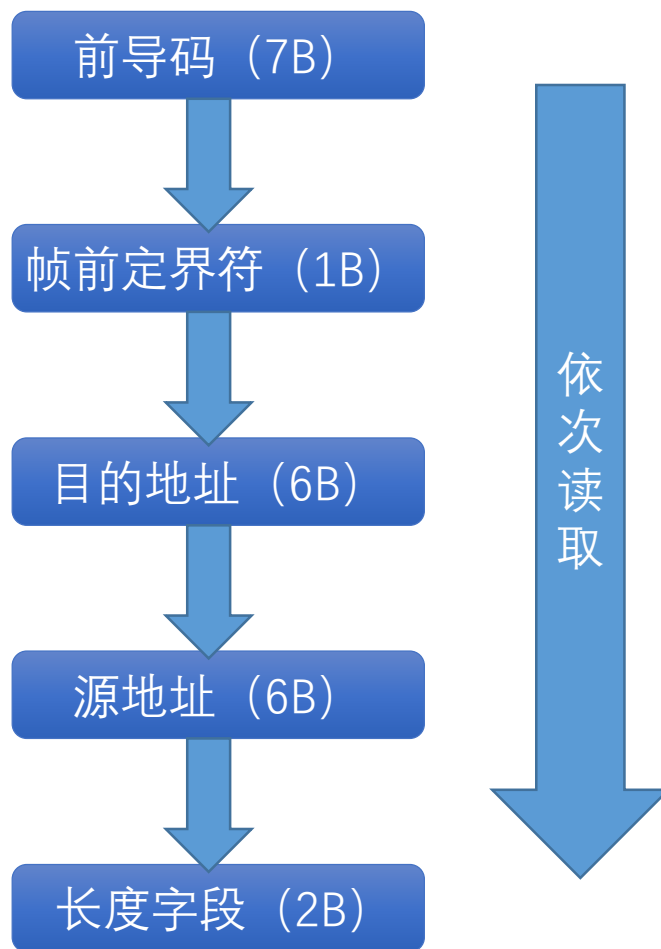
3. 例题分析

- 关键问题：解析帧头部各个字段

确定帧开始位置的伪代码

```
//将指针移到文件开始
file.seekg(0, ios::beg);
while(文件未结束)
{
    //查找前导码位置
    for(i=0; i<7; i++)
        if(file.get() != 0xaa)
            ...

    //查找帧前定界符位置
    if(file.get() != 0xab)
        ...
}
```



3. 例题分析

- 关键问题：解析数据字段

IEEE 802.3标准规定数据字段

- 最小长度为46B,
- 最大长度为1500B
- 如果数据小于46B, 通过填充 “0” 来补足46B, 这些 “0” 的个数不计入长度字段
- 处理数据字段时, 需要将额外填充的 “0” 从数据字段中去掉
- 如果得到长度值等于1500, 则需要判断其后是否仍有一个帧

输出数据字段的伪代码

```
//数据从缓冲区写入文件
char * data = new char[length];

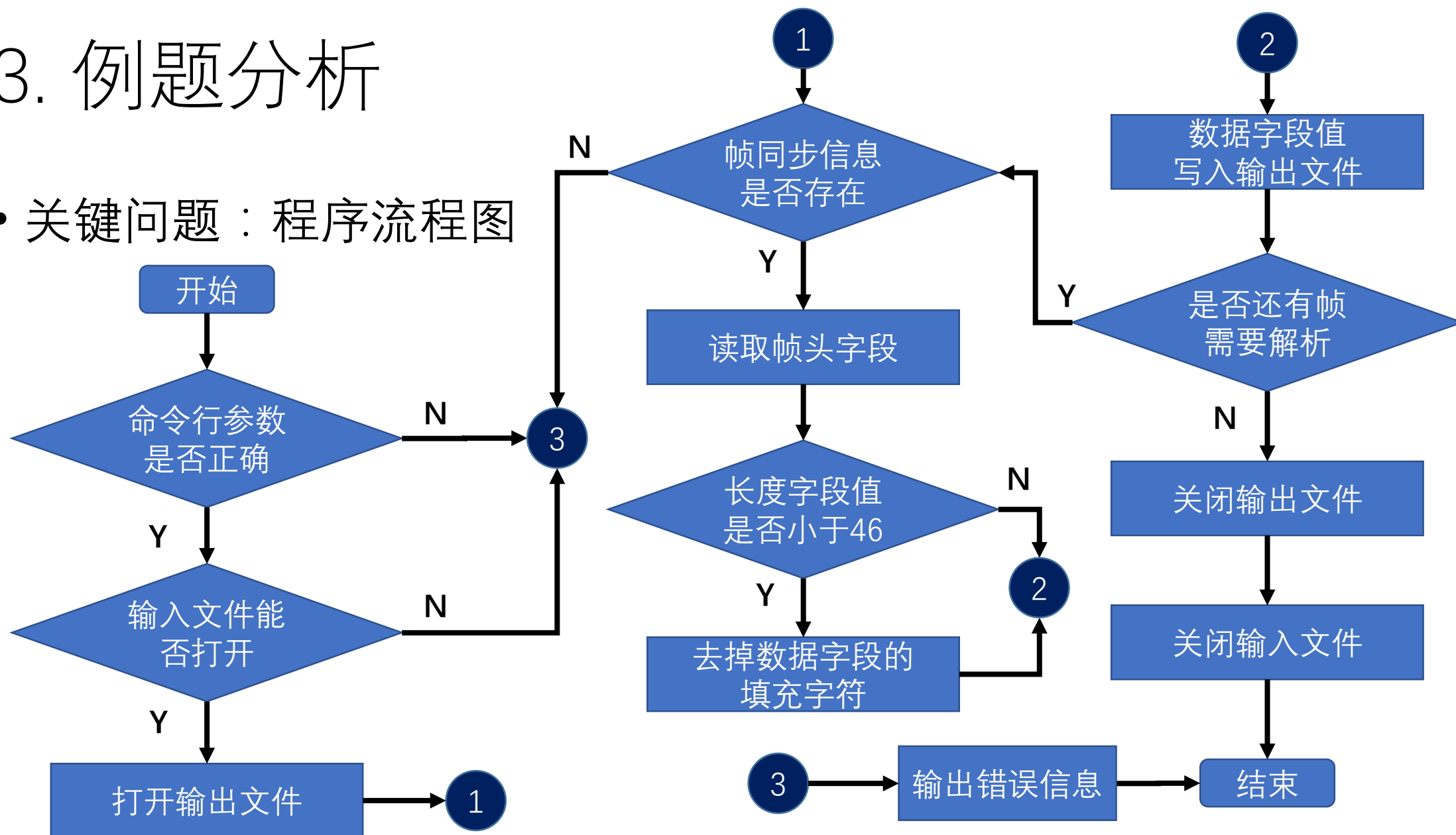
file.read(data, length);

outfile.write(data, length);

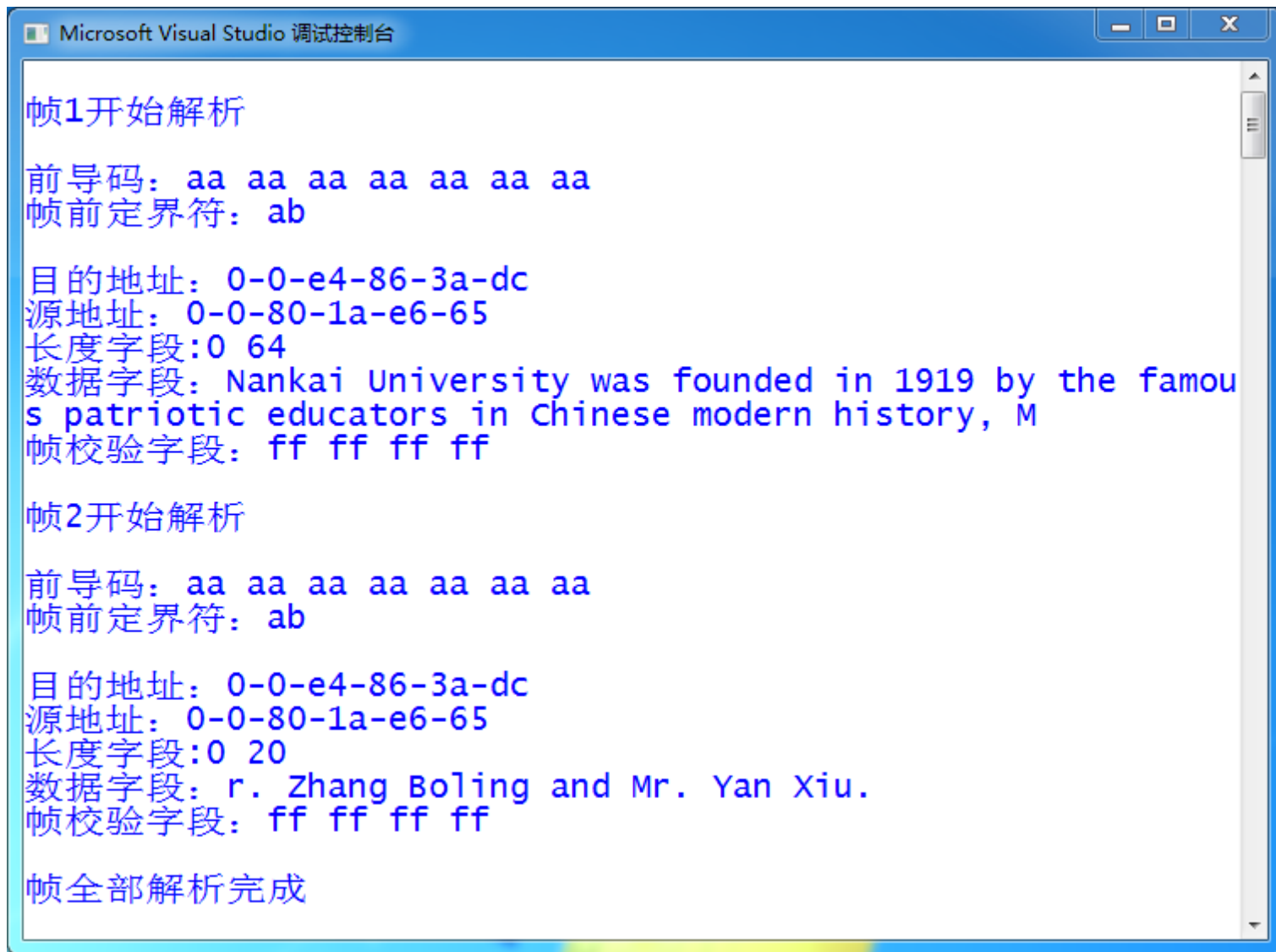
//数据字段长度小于46B, 去掉填充数据
if(length<46)
    for (i=0; i<46-length;i++)
        file.get();
```

3. 例题分析

- 关键问题：程序流程图



3. 例题分析



```
Microsoft Visual Studio 调试控制台

帧1开始解析

前导码: aa aa aa aa aa aa aa
帧前定界符: ab

目的地址: 0-0-e4-86-3a-dc
源地址: 0-0-80-1a-e6-65
长度字段: 0 64
数据字段: Nankai University was founded in 1919 by the famous patriotic educators in Chinese modern history, M
帧校验字段: ff ff ff ff

帧2开始解析

前导码: aa aa aa aa aa aa aa
帧前定界符: ab

目的地址: 0-0-e4-86-3a-dc
源地址: 0-0-80-1a-e6-65
长度字段: 0 20
数据字段: r. Zhang Boling and Mr. Yan Xiu.
帧校验字段: ff ff ff ff

帧全部解析完成
```

练习题（实验）

- 课本-第34页

本章小结

- 熟悉数据链路层帧结构
- 帧的解析过程