链滴

tcpv4 checksum 机试挑战。。

作者：zaoqigou
原文链接：https://ld246.com/article/1534226205300
来源网站：链滴

题目：

tcpv4 checksum 笔试题目

*** 目标：**

* 写一个函数 来计算 包含tcp包的ipv4包checksum，并且返回包含正确的checksum的ipv4包

*** 要求：**

* 只需要计算ipv4和tcp组合在一起的checksum，ipv4的Fragment不用管。不需要处理其他类其他组合的包的checksum，比如：ipv6不用管,udp不用管,icmp不用管。

* 传入的 tcp ipv4 包里面的checksum可能是正确的也可能是错误的。调用者只需要返回一个其内容相同，但是checksum正确的 tcp ipv4包

* 函数输入是一个 带长度的字节数组，函数输出是一个 带长度的字节数组。

* 可选功能，对输入包的正确性检查。

* 如果笔试者选择不实现该功能，那么调用者会保证输入的二进制满足 ipv4与tcp组合在一起时的效的包的二进制数组，保证不会使用ipv4的fragments功能。

* 如果笔试者选择实现该功能，那么调用者可能会输入任意二进制内容。此时笔试者应该把所有现错误的情况都打印到标准输出。程序运行过程中不应该出现内存溢出，下标越界等无法预料的情况。

* 使用可以满足上面要求的编程语言都可以。

* 需要写该函数的自动单元测试，以便证明你的函数的正确性。自动单元测试必须包含后面的 "输输出例子1" 一共1个例子。自动测试的输入与输出应该都在你的代码里面，并且自动判断。

* 该函数应该由你自己独立完成，该函数的任何部分都不允许复制其他人的代码。

* 该函数内部有checksum计算的每一个计算步骤，checksum计算本身不允许直接调用库实现。

* 函数实现可以修改传入的字节数组，也可以不修改传入的字节数组。

* 该函数名称为 RecomputeChecksum。

* 代码应当具有一定的可读性，人工应当能很轻松的找到所有实际执行代码的位置。建议避免使继承。

*** 笔试结果检查：**

* 检查者 运行所有的自动单元测试，检查是否可以编译。编译后运行结果，检查结果是否完全符预期，这一步有问题笔试失败。

* 检查者 随机修改某个自动单元测试的某个输入参数，检查编译运行后，结果结果是否运行失败这一步有问题笔试失败。

* 检查者 加入一个满足需求并且笔试者不知道的 测试例子，检查结果是否符合预期，如果这一部有问题笔试失败。

* 如果 代码不能编译运行 本次笔试失败。

* 如果 自动测试运行不符合预期 本次笔试失败。

* 如果 自动测试不包含后面的 输入输出例子1 本次笔试失败。

* 如果 自动测试的输入与输出 没有包含在代码里面 本次笔试失败。

* 如果不满足 独立完成要求 本次笔试失败。

* 如果不满足 包含对每个输入字节的实现细节 要求，本次笔试失败。

* 如果 找不到名为 RecomputeChecksum 的函数，本次笔试失败。

* 如果 RecomputeChecksum 函数的输入输出参数 数量或类型 不满足要求，本次笔试失败。

* 如果代码可读性非常差，阅读代码时很难搞清楚实际执行的代码的位置，则本次笔试失败，此由检查者决定是否通过。

* 人工阅读 自动单元测试代码以及实现的代码，检查是否 满足目标和要求。

* 自动测试 里面覆盖的细节全面，笔试分数有大量加分。

* 检查输入的包是否正确，任意二进制输入都能正确错误处理，而且程序不会出现内存溢出，下越界等无法预料的情况，笔试分数有大量加分。

* 代码复杂度低，代码易于理解，笔试分数有少量加分。

* 使用 golang 语言，笔试分数有少量加分。

* 代码文件使用 utf8无bom 编码，笔试分数有少量加分。

*** 参考：**

* ipv4 checksum https://tools.ietf.org/html/rfc1071

* tcp https://tools.ietf.org/html/rfc793

* tcp checksum https://en.wikipedia.org/wiki/Transmission_Control_Protocol#CP_checksum_for_IPv4

*** 直接返回输入的函数定义：**

* go语言版本：

```
func RecomputeChecksum(in []b
te)(out []byte){
  return in
}
```

* c语言版本：

```
typedef struct Slice{
  int len;
```

```
    unsigned char* data;
} Slice;
Slice RecomputeChecksum(Slice in){
    Slice output = {};
    output.len = in.len;
    output.data = in.data;
    return output;
}
```

* c++语言 应该使用和 c语言完全一致的输入输出接口。（在输入输出接口上仍然使用struct，不用类）

* js语言版本：(传入传出参数类型都是 Uint8Array 类型)

```
function RecomputeChecksum(inPara){
    var outBuffer = new Uint8Array(inPara.length);
    for (var i=0;i<inPara.length;i++){
        outBuffer[i] = inPara[i];
    }
    return outBuffer
}
```

*** 输入与输出例子：**

* 输入与输出例子1（后面使用的是golang语法）：

* 输入：

```
[]byte{0x45,0x00,0x00,0x8c,0x28,0xd1,0x00,0x00,0xff,0x06,0x00,0x00,0x73,0xef,0xd2,0x1
,
        0xac,0x15,0x00,0x01,0x00,0x50,0xe7,0xa3,0x93,0x2d,0xac,0xdb,0x9d,0x0e,0x0f,0x41,
        0x50,0x10,0xff,0xff,0x00,0x00,0x00,0x00,0x34,0x70,0x78,0x3b,0x70,0x61,0x64,0x64,
        0x69,0x6e,0x67,0x2d,0x6c,0x65,0x66,0x74,0x3a,0x31,0x30,0x70,0x78,0x3b,0x70,0x61,
        0x64,0x64,0x69,0x6e,0x67,0x2d,0x72,0x69,0x67,0x68,0x74,0x3a,0x31,0x30,0x70,0x78,
        0x3b,0x63,0x75,0x72,0x73,0x6f,0x72,0x3a,0x64,0x65,0x66,0x61,0x75,0x6c,0x74,0x3b,
        0x6f,0x76,0x65,0x72,0x66,0x6c,0x6f,0x77,0x3a,0x68,0x69,0x64,0x64,0x65,0x6e,0x3b,
```

0x77,0x68,0x69,0x74,0x65,0x2d,0x73,0x70,0x61,0x63,0x65,0x3a,0x6e,0x6f,0x77,0x72,
0x61,0x70,0x7d,0x2e,0x63,0x2d,0x64,0x72,0x6f,0x70,0x64,0x6f,}

* 输出:

[]byte{0x45,0x00,0x00,0x8c,0x28,0xd1,0x00,0x00,0xff,0x06,0xa0,0x79,0x73,0xef,0xd2,0x1
,
0xac,0x15,0x00,0x01,0x00,0x50,0xe7,0xa3,0x93,0x2d,0xac,0xdb,0x9d,0x0e,0x0f,0x41,
0x50,0x10,0xff,0xff,0xff,0xe6,0x00,0x00,0x34,0x70,0x78,0x3b,0x70,0x61,0x64,0x64,
0x69,0x6e,0x67,0x2d,0x6c,0x65,0x66,0x74,0x3a,0x31,0x30,0x70,0x78,0x3b,0x70,0x61,
0x64,0x64,0x69,0x6e,0x67,0x2d,0x72,0x69,0x67,0x68,0x74,0x3a,0x31,0x30,0x70,0x78,
0x3b,0x63,0x75,0x72,0x73,0x6f,0x72,0x3a,0x64,0x65,0x66,0x61,0x75,0x6c,0x74,0x3b,
0x6f,0x76,0x65,0x72,0x66,0x6c,0x6f,0x77,0x3a,0x68,0x69,0x64,0x64,0x65,0x6e,0x3b,
0x77,0x68,0x69,0x74,0x65,0x2d,0x73,0x70,0x61,0x63,0x65,0x3a,0x6e,0x6f,0x77,0x72,
0x61,0x70,0x7d,0x2e,0x63,0x2d,0x64,0x72,0x6f,0x70,0x64,0x6f,}