Use wireshark or tcpdump to view the packet.

```
$> cp ~promig3/pub/sample.pcap .
$> tcpdump -xx -r sample.pcap

0x0000:  0006 25da af73 0008 744f 3623 0800 4500
0x0010:  0030 01cb 4000 8006 0000 c0a8 0166 8077
0x0020:  f50c 101f 0050 f532 64b1 0000 0000 7002
0x0030:  faf0 e648 0000 0204 05b4 0101 0402
```

The Ethernet header is 14 bytes long, so IP header starts at byte 14.

```
$> cp ~promig3/pub/sample.pcap .
$> tcpdump -xx -r sample.pcap

0x0000:  0006 25da af73 0008 744f 3623 0800 4500
0x0010:  0030 01cb 4000 8006 0000 c0a8 0166 8077
0x0020:  f50c 101f 0050 f532 64b1 0000 0000 7002
0x0030:  faf0 e648 0000 0204 05b4 0101 0402
```

Using the GNU Debugger to see inside the packet

```
$> gdb project4
(gdb) break pk_processor
(gdb) run -f sample.pcap
(gdb) x/30xb packet
0x63c720: 0x00 0x30 0xc1 0x61 0xeb 0xed 0x00 0x08
0x63c728: 0x74 0x4f 0x36 0x23 0x08 0x00 0x45 0x00
0x63c730: 0x00 0x4e 0x01 0xc2 0x00 0x00 0x80 0x11
0x63c738: 0x00 0x00 0xc0 0xa8 0x01 0x66


(gdb)
```

Using the GNU Debugger to see inside the packet

```
$> gdb project4
(gdb) break pk_processor
(gdb) run -f sample.pcap
(gdb) x/20xb packet+14
0x63c728: 0x45 0x00 0x00 0x4e 0x01 0xc2 0x00
0x63c738: 0x80 0x11 0xc0 0xa8 0x01 0x66


(gdb)
```

From inside the program we can access each part of the packet by treating it as an array.

Add the following to the pk_processor() function:

```
std::cout << packet[14] << std::endl;
```

```
E == 69 ASCII == 0x45
```

Use Pointer Arithmetic to point at each header.

```
char *ipHeader = (char *)(packet+14);

std::cout << ipHeader[0] << std::endl;
```

What if we want something longer than a byte.

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															

```
std::cout << ipHeader[2] << std::endl;
```

```
std::cout << std::hex << ipHeader[2] << ipHeader[3] << std::endl;
```

```
std::cout << std::hex << (u_short)ipHeader[2] << (u_short)ipHeader[3] << std::endl;
```

Then to merge the two values:

```
std::cout << (ipHeader[2] << 8) + ipHeader[3] << std::endl;
```

That’s a pain (and ugly, and error prone)…

Then to merge the two values:

```
struct ipHdr {
        __u8    ver_dhr;
        __u8    tos;
        __u16   tl;
}
```

```
stuct ipHdr *ipHeader = (struct ipHdr *)(packet + 14);
std::cout << (int)ipHeader->tl << std::endl;  // Wrong because of byte order.
Std::cout << ntohs(int)ipHeader->tl );
```

Reminder – watch out for byte-ordering issues. Use ntohs() or ntohl()

We can use bitfields to get at items smaller than a byte.

```
struct ipHdr {
        __u8    ip_hl:4, ip_v:4;
        __u8    tos;
        __u16   tl;
}
```

```
stuct ipHdr * ipHeader = (struct ipHdr *)(packet + 14);

std::cout << (int)ipHeader->ip_v << std::endl;
std::cout << (int)ipHeader->ip_hl << std::endl;
std::cout << ntohs(ipHeader->tl ) << std::endl;
```

Header files you might want:

```
#include <iomanip>
#include <linux/types.h>

#include <netinet/ip.h>   // Others here as well for other headers.
```