获取本机设备列表,如果获取失败输出 Error in pcap_findalldevs，并退出程序

```
    pcap_if_t* alldevs;   //所有网络适配器
        pcap_if_t* d;   //选中的网络适配器
        int inum;   //选择网络适配器
        int i = 0;   //for 循环变量

        if (pcap_findalldevs(&alldevs, errbuf) == -1)
        {
            fprintf(stderr, "Error in pcap_findalldevs: %s\n", errbuf);
            fprintf(fp, "Error in pcap_findalldevs: %s\n", errbuf);
            exit(1);
        }
```

打印列表，同时写入日志文件

```
        for (d = alldevs; d; d = d->next)
        {
            printf("%d. %s", ++i, d->name);
            fprintf(fp, "%d. %s", i, d->name);
            if (d->description) {
                printf(" (%s)\n", d->description);
                fprintf(fp, " (%s)\n", d->description);
            }
            else {
                printf(" (No description available)\n");
                fprintf(fp, " (No description available)\n");
            }
            //查询并保存网络设备的 ip 地址
            char* str = (char*)"0.0.0.0";
            for (pcap_addr_t* a = d->addresses; a; a = a->next) {
                if (a->addr->sa_family == AF_INET) {
                    if (a->addr) {
                        str                    =                    iptos(((struct
sockaddr_in*)a->addr)->sin_addr.s_addr, i);
                        break;
                    }
                }
            }
            printf("   IP Address: %s\n", str);
            fprintf(fp, "   IP Address: %s\n", str);
        }

        if (i == 0)
        {
            printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
```

```
            return -1;
        }
```

输入所要选择的设备序号

```
        printf("\nEnter the interface number (1-%d):", i);
        fprintf(fp, "\nEnter the interface number (1-%d):", i);
        scanf_s("%d", &inum);
        fprintf(fp, "%d\n", inum);
```

如果输入的序号错误，释放设备列表，结束程序

```
        if (inum < 1 || inum > i)
        {
            printf("\nInterface number out of range.\n");
            fprintf(fp, "\nInterface number out of range.\n");
            /* 释放设备列表 */
            pcap_freealldevs(alldevs);
            return -1;
        }
```

由于 alldevs 是一个链表，故通过 for 循环，跳转到选中的适配器

```
        for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);
```

打开设备，若打开失败，释放设备列表，结束程序

```
        if ((adhandle = pcap_open(d->name,          // 设备名
            65536,              // 65535 保证能捕获到不同数据链路层上的每个数据包
的全部内容
            PCAP_OPENFLAG_PROMISCUOUS,    // 混杂模式
            1000,               // 读取超时时间
            NULL,               // 远程机器验证
            errbuf              // 错误缓冲池
        )) == NULL)
        {
            fprintf(stderr, "\nUnable to open the adapter. %s is not supported
by WinPcap\n", d->name);
            fprintf(fp, "\nUnable to open the adapter. %s is not supported by
WinPcap\n", d->name);
            /* 释放设备列表 */
            pcap_freealldevs(alldevs);
            return -1;
        }
```

通过 getMacAddr 函数获取所选的设备 mac 地址,若成功获取则 res=0,反之 res=1

```
        int res = getMacAddr(inum);
```

若 getMacAddr 函数未能自动获取到本机 mac 地址，手动输入

```
        if (res != 0) {
            printf("Cannot get MAC address automatically, please input MAC
address: ");
            fprintf(fp, "Cannot get MAC address automatically, please input MAC
address: ");
            u_int temp;
            for (i = 0; i < 6; i++) {
                scanf_s("%d", &temp);
                net_mac_addr[i] = temp;
                fprintf(fp, "%d ", temp);
            }
            fprintf(fp, "\n");
        }
```

输入目的 ip 地址

```
        printf("Input the IP Address of destination: ");
        fprintf(fp, "Input the IP Address of destination: ");
        u_int temp;
        for (i = 0; i < 4; i++) {
            scanf_s("%d", &temp);
            dst_ip[i] = temp;
            fprintf(fp, "%d ", temp);
        }
        fprintf(fp, "\n");
```

打印输出适配器的 mac 地址，目的 ip 地址，本机 ip 地址，并写入日志文件

```
        printf("\nThe MAC Address of Adapter %d: ", inum);
        fprintf(fp, "\nThe MAC Address of Adapter %d: ", inum);
        printAddr(net_mac_addr, MACADDR);

        printf("The IP Address of Adapter %d: ", inum);
        fprintf(fp, "The IP Address of Adapter %d: ", inum);
        printAddr(net_ip_addr[inum], IPADDR);
        printf("The IP Address of destination: ");
        fprintf(fp, "The IP Address of destination: ");
        printAddr(dst_ip, IPADDR);
```

通过 sendARP 函数发送 ARP 数据包，若发送成功则 res=0，反之 res=1

```
        res = sendARP(net_ip_addr[inum], dst_ip);

        if (res == 0) {
            printf("\nSend packet successfully\n\n");
```

```
            fprintf(fp, "\nSend packet successfully\n\n");
        }
        else {
            printf("Failed to send packet due to: %d\n", GetLastError());
            fprintf(fp, "Failed to send packet due to: %d\n", GetLastError());
        }
```

过滤器设置

```
        netmask                                                                    =
((sockaddr_in*)((d->addresses)->netmask))->sin_addr.S_un.S_addr;
        pcap_compile(adhandle, &fcode, filter, 1, netmask); //编译过滤器
        pcap_setfilter(adhandle, &fcode);   //设置过滤器
```

释放设备列表并开始抓包

```
        pcap_freealldevs(alldevs);

       i = 0;

        printf("Catching packets...\n\n");
        fprintf(fp, "Catching packets...\n\n");
```

抓包并解析，由于 ARP 包封装在 MAC 帧，MAC 帧首部占 14 字节，故 arpheader 偏移 14 字节，若所抓取的数据包源 ip 地址不等于本机 ip 地址，则继续抓包，直到抓到源 ip 地址等于本机 ip 地址的数据包，继续解析，若是 request，则写入日志文件并继续抓包，若是之前 request 的 reply 则停止抓包

```
        int begin = -1;
        //获取数据包并解析
        while (res = pcap_next_ex(adhandle, &header, &pkt_data) >= 0) {
            //超时
            if (res == 0) {
                continue;
            }

            //解析 ARP 包，ARP 包封装在 MAC 帧，MAC 帧首部占 14 字节
            ArpHeader* arpheader = (ArpHeader*)(pkt_data + 14);
            if (begin != 0) {
                begin     =     memcmp(net_ip_addr[inum],     arpheader->sip,
sizeof(arpheader->sip));
                if (begin != 0) {
                    continue;
                }
            }
            //获取时间戳
            local_tv_sec = header->ts.tv_sec;
```

```
            ltime = localtime(&local_tv_sec);
            strftime(timestr, sizeof(timestr), "%H:%M:%S", ltime);
            printf("(%s) ", timestr);
            fprintf(fp, "(%s) ", timestr);

            printf("message %d:\n", ++i);
            fprintf(fp, "message %d:\n", i);
            //设置标志，当收到之前发送的 request 的 reply 时结束捕获
            bool ok = false;
            if (arpheader->op == 256) {
                printf("request message.\n");
                fprintf(fp, "request message.\n");
            }
            else {
                printf("reply message.\n");
                fprintf(fp, "reply message.\n");
                //如果当前报文时 reply 报文，则通过比较 ip 来判断是否时之前发送的
request 对应的 reply
                if          (memcmp(arpheader->dip,          net_ip_addr[inum],
sizeof(arpheader->dip)) == 0) {
                    memcpy(dst_mac, arpheader->smac, 6);
                    ok = true;
                }
            }
```

将 ARP 数据包中的信息打印输出并写入日志文件

```
            //获取以太网帧长度
            printf("ARP packet length: %d\n", header->len);
            fprintf(fp, "ARP packet length: %d\n", header->len);

            //打印源 mac
            printf("source mac: ");
            fprintf(fp, "source mac: ");
            printAddr(arpheader->smac, MACADDR);
            //打印源 ip
            printf("source ip: ");
            fprintf(fp, "source ip: ");
            printAddr(arpheader->sip, IPADDR);
            //打印目的 mac
            printf("destination mac: ");
            fprintf(fp, "destination mac: ");
            printAddr(arpheader->dmac, MACADDR);
            //打印目的 ip
            printf("destination ip: ");
```

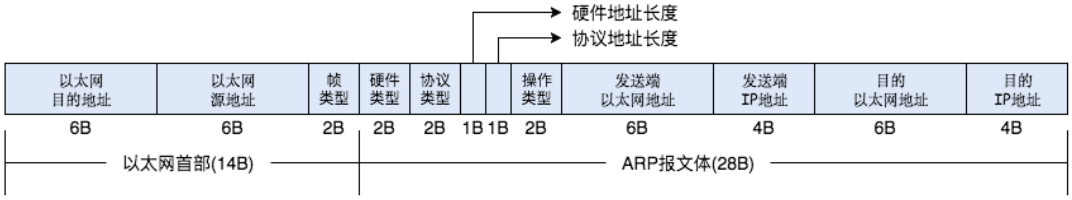
```
            fprintf(fp, "destination ip: ");
            printAddr(arpheader->dip, IPADDR);

            printf("\n\n");
            fprintf(fp, "\n\n");
            if (ok) {
                printf("Get the MAC address of destination: ");
                fprintf(fp, "Get the MAC address of destination: ");
                printAddr(dst_mac, MACADDR);
                printf("\nEnd of catching...\n\n");
                fprintf(fp, "\nEnd of catching...\n\n");
                break;
            }
        }
```

关闭文件流，结束程序

```
        fclose(fp);
         getchar();
         return 0;
```

定义 ARP 协议格式




```
    #define ETH_ARP          0x0806   //以太网帧类型表示后面数据的类型，对于 ARP 请求或应答来说，该字段的值为 x0806
    #define HARDWARE    1   //硬件类型字段值为表示以太网地址
    #define ETH_IP           0x0800   //协议类型字段表示要映射的协议地址类型值为 x0800 表示 IP 地址

    //14 字节以太网首部
    struct EthHeader
    {
        u_char DestMAC[6];
        u_char SourMAC[6];
        u_short EthType;
    };


    //28 字节 ARP 帧结构
```

```
struct ArpHeader
{
    unsigned short hdType;
    unsigned short proType;
    unsigned char hdSize;
    unsigned char proSize;
    unsigned short op;
    u_char smac[6];
    u_char sip[4];
    u_char dmac[6];
    u_char dip[4];
};

//定义整个 arp 报文包，总长度 42 字节
struct ArpPacket {
    EthHeader ed;
    ArpHeader ah;
};
```

封装 ARP 数据包并广播发送

```
int sendARP(u_char * src_ip, u_char * dst_ip)
{
    unsigned char sendbuf[42]; //arp 包结构大小，42 个字节
    EthHeader eh;
    ArpHeader ah;
    memcpy(eh.DestMAC, dst_mac, 6);     //以太网首部目的 MAC 地址,全为广播地
址
    memcpy(eh.SourMAC, net_mac_addr, 6);//以太网首部源 MAC 地址
    memcpy(ah.smac, net_mac_addr, 6);   //ARP 字段源 MAC 地址
    memcpy(ah.dmac, dst_mac, 6);        //ARP 字段目的 MAC 地址
    memcpy(ah.sip, src_ip, 4);          //ARP 字段源 IP 地址
    memcpy(ah.dip, dst_ip, 4);          //ARP 字段目的 IP 地址
    eh.EthType = htons(ETH_ARP);        //htons：将主机的无符号短整形数转换
成网络字节顺序
    ah.hdType = htons(HARDWARE);
    ah.proType = htons(ETH_IP);         //上层协议设置为 IP 协议
    ah.hdSize = 6;
    ah.proSize = 4;
    ah.op = htons(REQUEST);
    memset(sendbuf, 0, sizeof(sendbuf));   //ARP 清零
    memcpy(sendbuf, &eh, sizeof(eh));
    memcpy(sendbuf + sizeof(eh), &ah, sizeof(ah));
    return pcap_sendpacket(adhandle, sendbuf, 42);  // 发送 ARP 数据包并返回
```

```
发送状态
    }
```