部门 _____ 科室 _____ 姓名 _____ 工号 _____

考试时长:3小时,总分:100分

试题背景说明：

(1) 基本数据类型定义：

```
typedef  char *             LPSTR;
typedef  unsigned   char ** LPLPSTR;
typedef  signed     char    CHAR;
typedef  unsigned   char    BOOL8;
typedef  unsigned   char    BYTE;
typedef  unsigned   short   WORD;
typedef  unsigned   long    DWORD;
typedef  unsigned   short   WORD;
typedef  unsigned   long    DWORD;
typedef  unsigned   short   WORD16;
typedef  unsigned   long    WORD32;
```

(2) 数据接口调用函数原型说明：

```
void dbCall( WORD wEvent, LPSTR pReq, LPSTR ptAck );
```

其中:wEvent表示事件号,pReq表示入参结构体指针,ptAck表示出参结构体指针。入参结构体的定义一般如下：

```
typedef struct
{
    /* 消息类型同步调用或异步调用,调用者必须填写该参数: */
    BYTE       bMsgType;
    ......
}D_XXX_REQ, * LPD_XXXX_REQ;
typedef struct
{
    /* 接口调用结果—返回成功或者失败码,    */
    /* 接口必须返回该参数的,供调用者使用。 */
    WORD      wRetCode;
    ......
}D_XXX_ACK,* LPD_XXX_ACK;
```

1 改错题(每道题目至少1处错误或者隐患) 80分

1.1 请找出下面代码中的隐患或者错误,说明故障原因并改正。 7分

```
typedef struct
{
    WORD     retCODE;
    BYTE     baPad[2];
    DWORD    dwSipTestCfg[DB_SIPTEST_ITEM_NUM];
}  DM_GETSIPTESTCFG_ACK,  _FAR* LP_DM_GETSIPTESTCFG_ACK;
```

```
typedef struct tagPPLAT_SIP_TESTCFG_CFG_T
{
    WORD dwTestCFG[DB_SIPTEST_ITEM_NUM];
} PPLAT_SIP_TESTCFG_CFG_T,*LP_PPLAT_SIP_TESTCFG_CFG_T;

WORD  Test(PPLAT_SIP_TESTCFG_CFG_T ** pptCFG)
{
    DM_GETSIPTESTCFG_REQ    tReq;
    DM_GETSIPTESTCFG_ACK    tAck;
    PPLAT_SIP_TESTCFG_CFG_T tcfg;

    memset(&tCFG, 0, sizeof(PPLAT_SIP_TESTCFG_CFG_T));
    memset(&tReq, 0, sizeof(DM_GETSIPTESTCFG_ACK));
    tReq.msgType = MSG_CALL;

    dbCall (DM_GETSIPTESTCFG, (LPSTR) & tReq, (LPSTR) & tAck);
    if (RC_OK != tAck.retCODE)
    {
        return PPLAT_SIP_ERROR_DBA_ACCESS;
    }
    memcpy(tCFG.dwTestCFG,tAck.dwSipTestCfg,DB_SIPTEST_ITEM_NUM);
    *ptCFG = & tCFG;

    return PPLAT_SIP_OK;
}
```

1.2 请找出下面代码中的隐患或者错误,说明故障原因并改正。 6分

```
#define MaxDataLen (WORD)1536
#define INITminLen (WORD)(sizeof(InitChunk))
Associat * Association;

typedef struct initchunk
{
    ChunkHead   ChunkInfo;
    DWORD       dwInitiateTag; /* Initiate Tag */
    DWORD       dwRwnd; /* Advertised Receiver Window Credit */
    WORD        wOutStreams;    /* Number of Outbound Streams */
    WORD        wInStreams;     /* Number of Inbound Streams  */
    DWORD       dwInitialTSN;   /* Initial TSN */
}InitChunk;

typedef struct sctpchunks
{
    BYTE        bNetLayerID;     /* 0: IP; 1: UDP */
    DWORD       dwDestIPAddr;
    DWORD       dwSourIPAddr;
    WORD        wSourcePortNum; /* Source Port Number */
    WORD        wDestPortNum;    /* Destination Port Number */
    DWORD       dwVerifTag;      /* Verification Tag */
```

```
    BYTE        bChunkNum;
    BYTE        ChunkInfo[ MaxDataLen];
} SctpChunks;

void UnexpectInit( WORD    wIndex, SctpChunks *pChunk )
{
    WORD         ParaType, wLen1, wLen2, i ;
    BYTE         *ptr   = NULL ;
    WORD         *pWord = NULL;
    SctpDlp      SctpToDlp;
    StateCookie  Cookie;

    InitChunk    *pInitChunk    = NULL;
    IPv4AddrPara *pIPv4AddrPara = NULL;

    if (NULL == pChunk)
    {
        return;
    }

    pInitChunk = (InitChunk*)&(pChunk->ChunkInfo[0]);
    wLen1 = pInitChunk->ChunkInfo.wChunkLen;

    if (wLen1 < INITminLen)
    {
        return;
    }

    Cookie.wParaLen = sizeof(StateCookie);

    wLen2 = 0;
    wLen1 = wLen1 - (WORD)sizeof(InitChunk);
    if ( wLen1 >= 4 )
    {
        ptr = (BYTE *)&(pChunk->ChunkInfo[sizeof(InitChunk)]);
        do
        {
            pWord = (WORD *)ptr;
            ParaType = *pWord;
            if (ParaType ==IPv4TYPE)
            {
                pIPv4AddrPara = (IPv4AddrPara *)ptr;

                if ((D_dwSCTPSwitches&PARAMT297_TAG_RESTARTASSO)
                    != 0)
                {    /* 开关打开 */
                    for (i = 0; i < MaxIPAddrList; i++)
                    {
                        if (pIPv4AddrPara->dwIPv4Addr ==
```

```
                            Association[ wIndex].TCB.PeerAddr[ i].
                                dwPeerIP)
                        {
                            Cookie.PerAddrList[ Cookie.
                                bPeerAddrCount] =
                                 pIPv4AddrPara->dwIPv4Addr;
                            Cookie.bPeerAddrCount++;
                            break;
                        }
                    }
                    ptr    = ptr + IPv4PARALEN;
                    wLen1  = wLen1 - IPv4PARALEN;
                }
                else
                {
                    if( pIPv4AddrPara->dwIPv4Addr != pChunk->
                        dwSourIPAddr
                        && Cookie.bPeerAddrCount < MaxIPAddrList )
                    {
                        Cookie.PerAddrList[ Cookie.bPeerAddrCount]
                            = pIPv4AddrPara->dwIPv4Addr;
                        Cookie.bPeerAddrCount++;
                    }
                    ptr    = ptr + IPv4PARALEN;
                    wLen1  = wLen1 - IPv4PARALEN;
                }
            }
            else
            {
                ptr    = ptr + COOKIEPRELEN;
                wLen1  = wLen1 - COOKIEPRELEN;
            }
        }while(wLen1>=4);
    }

    return;
}
```

1.3 请找出下面代码中的隐患或者错误，说明故障原因并改正。 7分

```
VOID PH248_CA_ExecAbortReq( PH248_CONTEXT * ptRoomIndex,
                            VEINU_ABORT_IND * ptIn )
{
    PID                 tSrcPid;
    PH248_CELL    *     ptCell            = NULL;

    SELF(&tSrcPid);
    if (PH248_CA_JudgeContextValid(ptRoomIndex) != TRUE)
    {
        return;
```

```
    }
    /* NULL context direct send rpl to
     * L4 and release the context of CA */

    if ( ptRoomIndex ->dwContextId == NULL_CONTEXT &&
         (D_dwH248Switches & PARAMT_DATA2_DONOTDOWNSUB))
    {     /*全局业务开关单板空上下文下 SUB 开关对中继用户不起作用,LXL */
        /*>0 不下 sub*/
        PH248_CA_LocateTMRoom(&ptIn->tMsgHead.tTid, &ptCell);
        ptCell->bCmdType = COMMAND_MODIFY;
        ptCell->bTransactionType = TRANSACTION_REQUEST;
        /* 初始化命令描述符 */
        PH248_CA_InitCmdDes(COMMAND_MODIFY);
        /*Fill Empty signal*/
        PH248_CA_FillEmptySig();
        PH248_CA_FillOneDigitMapReq(&(ptCell->tTid),
                                    0,
                                    0,
                                    0);
        /* 命令描述符填写结束,设置当前命令的描述符以及命令信息 */
        PH248_CA_SetTtmHeadCmdInfo( COMMAND_MODIFY,
                                    (DWORD)ptCell,
                                    &ptCell->tTidString);
        ptRoomIndex->bfToMg = BUFFER_USED;
    }
    else
    {     /* process the Event */
        PH248_CA_RevAbortReq( ptRoomIndex );
    }
    if (ptRoomIndex->bfToMg == BUFFER_USED)
    {     /* Judge the ToMg Flag*/
        /* send msg to Ttm */
        P_H248_SendSingleReq();
    }
    /* Remove the Context directly */
    PH248_CA_CleanContext(ptRoomIndex, FALSE);

    return;
}
```

1.4 请找出下面代码中的隐患或者错误,说明故障原因并改正。 12分

```
#define HASCHANNELID 0x00800000
BYTE P_Q931_Call_ChoiceBchL2(LPSTR  pMsgData,
                             BYTE   bEvent,
                             P_Q931_BRAINFO *ptBraInfo,
                             BYTE   *pbExclusive,
                             BYTE   *pbSelection)
{
```

```
WORD    wB1Status   = ptBraInfo->wB1Status;
WORD    wB2Status   = ptBraInfo->wB2Status;
BYTE    bSelection, bExclusive;
BYTE    bCauseValue = ISDN_OK;
DWORD   dwFlag;
BYTE    bBCRate     = bcRate0;
P_Q931_ChannelID * pChannelId = NULL;

if (ptBraInfo == NULL || pbExclusive == NULL
    || pbSelection == NULL)
{
    return ~ISDN_OK;
}

switch ( bEvent )
{
    case 1:
        pChannelId =
            &(((P_Q931_ConnectMsg *)pMsgData)->channel);
        dwFlag = ((P_Q931_ConnectMsg *)pMsgData)->flag;
        break;
    case 2:
        pChannelId =
            &(((P_Q931_SetupMsg *)pMsgData)->channel);
        dwFlag = ((P_Q931_SetupMsg *)pMsgData)->flag;
        bBCRate = ((P_Q931_SetupMsg *)pMsgData)->bearer.rate;
    case 3:
        pChannelId =
            &(((P_Q931_RetrieveMsg *)pMsgData)->channel);
        dwFlag = ((P_Q931_RetrieveMsg *)pMsgData)->flag;
        break;
}

if(!(dwFlag && HASCHANNELID))
{ /* 消息中没有“通路识别”参数认为对方是“未选通路,可接受任何通道” */
    bSelection = chanSelNo;
    bExclusive = chanExclusivePri;/*0x00*/
}
else
{
    bSelection = pChannelId->selection;
    bExclusive = pChannelId->exclusive;
}

if(bExclusive == chanExclusiveOnly &&
   (bSelection == chanSelB1 || bSelection == chanSelB2))
{    /* 情形 a: 指明通路不接收其他选择的通路 */
    if((bSelection == chanSelB1 &&
        wB1Status != P_Q931_BCH_STATE_IDLE/*空闲*/)||
```

```
            (bSelection == chanSelB2 &&
             wB2Status != P_Q931_BCH_STATE_IDLE/*空闲*/))
        {
            bCauseValue = ISDN_RqCircuitUnavail;
        }
    }
    else if(bExclusive == chanExclusivePri &&
            (bSelection == chanSelB1 || bSelection == chanSelB2))
    {     /* 情形 b: 指明通路可接收其他选择的通路 */
        if(wB1Status != P_Q931_BCH_STATE_IDLE &&
           wB2Status != P_Q931_BCH_STATE_IDLE)
        {     /* 两条通路都不可用 */
            bCauseValue = ISDN_NoCircuitAvailable;
        }
        else
        {     /* 某一条可用*/
            bSelection = (wB1Status == P_Q931_BCH_STATE_IDLE)?
                chanSelB1:chanSelB2;
        }
    }
    else
    {     /* 情形 c: 接收任何通路此时  bExclusive 不起作用*/
        if(wB1Status != P_Q931_BCH_STATE_IDLE &&
           wB2Status != P_Q931_BCH_STATE_IDLE)
        {     /* 两条通路都不可用 */
            bCauseValue = ISDN_NoCircuitAvailable;
        }
        else
        {
            bSelection = (wB1Status == P_Q931_BCH_STATE_IDLE)?
                chanSelB1:chanSelB2;
        }
    }
    if(bCauseValue != ISDN_OK)
    {     /*出现选路错误*/
        return bCauseValue;
    }
    else
    {     /*选择了通路*/
        *pbSelection = bSelection;
        *pbExclusive = bExclusive;
        return ISDN_OK;
    }
}
```

1.5 请找出下面代码中的隐患或者错误,说明故障原因并改正。 11分

```
typedef struct tagTableStru{
    DB_HANDLE    hTable;
    char         lpTableName[TABLE_NAME_LEN];
    BYTE         btFieldsNum;
    BOOL         bSave;
}TABLE_STRU, *LP_TABLE_STRU;

#define MAX_TABLE_NUM       100
TABLE_STRU    TableStru[MAX_TABLE_NUM];
#define  MAKE_TBL_FILENAME(TableLoc, Dir, TblFileName, Postfix) \
    memset(TblFileName, 0, sizeof(TblFileName));                \
    strcat(TblFileName,TableStru[TableLoc].lpTableName);        \
    strcat(TblFileName,".");                                    \
    strcat(TblFileName,Postfix)

extern STATUS  _Tables_Remove(LPSTR Dir, LPSTR Postfix)
{
    BYTE         FileName[80];
    DB_HANDLE     TableLoc = 0;
    FILE     *     dbFile;

    while (TableStru[TableLoc].hTable != INVALID_DB_HANDLE)
    {
        if (TableStru[TableLoc].bSave)
        {
            memset(FileName, 0, sizeof(FileName));
            MAKE_TBL_FILENAME(TableLoc, Dir, FileName, Postfix);
            if ((dbFile = fopen(FileName,"rb")) != NULL)
            {
                fclose(dbFile);
                if (OK != remove(FileName))
                {
                    DbgMsg(INFORM_ERR,("fail to remove %s !\n",
                                       FileName));
                    return ERROR;
                }
            }
            else
            {
                DbgMsg(INFORM_ERR,("fail to open %s!\n",
                                   FileName));
            }
            TableLoc++;
        }
    }
    return OK;
}
```

1.6 请找出下面代码中的隐患或者错误,说明故障原因并改正。 10分

```
#define TEMP_BUF_LEN 64
WORD Syn_SetToDB(VOID *para_in )
{
    BYTE    location[TEMP_BUF_LEN]   = {0};
    BYTE    location_2[TEMP_BUF_LEN] = {0};
    DWORD * pParaIn = NULL;
    DWORD   handle;
    DWORD   ret     = 0;
    *pParaIn = *(DWORD *)para_in;
    handle = Data_GetKey(ROOT , ROOT , strlen(ROOT));
    if(!Valid_Loc(handle))
    {
        ret = Data_Set( ROOT , ROOT, strlen(ROOT),
                        _DATA_OPRTYPE_NOWRITE, _DATA_TYPE_ENTRY,
                        NULL, 0);
        if(ret != DATA_SET_SUC)
        {
            return ERR_SET_DB_FAIL;
        }
    }
    snprintf(location,TEMP_BUF_LEN,"%s%s%s",
             ROOT,NODE_SPLIT, SYN);
    handle = Data_GetKey(ROOT , location , strlen(location));
    if(!Valid_Loc(handle))
    {
        ret = Data_Set( ROOT , location, strlen(location),
                        _DATA_OPRTYPE_NOWRITE, _DATA_TYPE_BRANCH,
                           NULL, 0);
        if(ret != DATA_SET_SUC)
        {
            return ERR_SET_DB_FAIL;
        }
    }
    snprintf(location_2,TEMP_BUF_LEN,"%s%s%s",
             location,NODE_SPLIT,SYN_ENABLE);
    handle = Data_GetKey(ROOT , location_2 , strlen(location_2));
    {
        ret = Data_Set( ROOT , location_2,strlen(location_2),
                        _DATA_OPRTYPE_NOWRITE ,_DATA_TYPE_DWORD,
                        (LPSTR)pParaIn,sizeof(DWORD));
        if(ret != DATA_SET_SUC)
        {
            return ERR_SET_DB_FAIL;
        }
    }
    return SUCC_AND_NOPARA;
}
```

1.7 请找出下面代码中的隐患或者错误,说明故障原因并改正。 5分

```
#define MAX_BLADE_NUM_PER_POOL   20

typedef
{
    BOOL8   blEnabled;
    BYTE    bConfigWeight;
    WORD    wIp;
    ...
}T_BladeNode;

typedef struct
{
    BYTE          bAliveBladeNum;
    WORD          wIndex;
    T_BladeNode   atBladeArray[ MAX_BLADE_NUM_PER_POOL];
}T_BladePool;

/* 根据选定算法选取一个可用的Blade */
int Mc_SlbChoose(T_VServer *pVServer, T_SlbInfo *pSlbInfo)
{
    T_BladePool *p = &(pVServer->tPool);
    WORD32      j  = 0;
    BYTE        cw = 0; /* 当前权值 */

    if( (0 == p->wNum) || (0 == p->bAliveBladeNum) )
    {
        return MCS_FAIL;
    }
    switch(pVServer->bSlbType)
    {
        /* 轮询,这种 SLB 方法需要记录上次的结果,包括 wIndex */
        case SLBTYPE_ROUNDROBIN:
        {
            j = p->wIndex;
            do
            {
                j = (j+1) % MAX_BLADE_NUM_PER_POOL;
                if( p->atBladeArray[j].blEnabled )
                {
                    p->wIndex             = j;
                    pSlbInfo->bBladeIndex = j;
                    pSlbInfo->wBIp = p->atBladeArray[j].wIp;
                    return MCS_OK;
                }
            }while(j != p->wIndex);
            return MCS_FAIL;
        }
```

```
        /* 根据权重进行轮询,这种 SLB 方法需要使用记录上次的结果,
         * 包括 CurWeight, wIndex 。
         * 这种方式是先设置 CurWeigh 为最大值,然后从头到尾寻找
         * Weight 大于等于此值的 Blade ,
         * 降低 CurWeight 为 CurWeight-gcd(S),然后再从头到尾寻找
         * Weight 大于等于此值的 Blade */
        case SLBTYPE_WEIGHTROUNDROBIN:
        {
            j  = p->wIndex;      /* 获取上次使用 Blade Index */
            cw = p->bCurWeight; /* 获取当前使用的 Weight */
            while(1)
            {
                j = (j+1) % MAX_BLADE_NUM_PER_POOL;
                if(0 == j)
                { /* 表示 Pool 中第一个Blade,
                   * 初始时,或者循环了一圈再次开始 */
                    if(cw <= p->bWeightGCD)
                    {
                        cw = p->bWeightMax;
                        if(0 == cw)
                        {
                            return MCS_FAIL;
                        }
                    }
                    else
                    {
                        cw -= p->bWeightGCD;
                    }
                }
                if(  p->atBladeArray[j].blEnabled &&
                     p->atBladeArray[j].bConfigWeight >= cw)
                {    /* 选定此 Blade */
                    pSlbInfo->bBladeIndex = j;
                    pSlbInfo->wBIp = p->atBladeArray[j].wIp;
                    /* 记录下当前使用的 Blade index */
                    p->wIndex      = j;
                    /* 记录下当前使用的 CurWeight */
                    p->bCurWeight  = cw;
                    return MCS_OK;
                }
            }
        }
        default:
            return MCS_FAIL;
    }
}
```

1.8 请找出下面代码中的隐患或者错误,说明故障原因并改正。 6分

```
/* 收集接口上 igmp 配置信息,送后台 ddm 显示 */
VOID igmp_ddm(T_DdmProtocolDAQuyAck *ptAck, BYTE oprationflag)
{
    WORD16                  i            = 0;
    WORD16                  j            = 0;
    static DWORD            dwFrom       = 0;
    static DWORD            dwPortNo     = 0;
    PID                     tSender;
    DWORD                   dwMsglen     = 0;
    net_if                  *n           = NULL;
    struct mgroup_address   *pGroup      = NULL;
    BYTE                    bGroupFinish = 0;

    if (NULL == ptAck)
    {
        return;
    }

    dwMsglen = sizeof(T_DdmProtocolDAQuyAck);
    OSS_ASSERT(BRS_DDMDATA_MAX_LEN >= dwMsglen);

    /* 发送者找不到,返回 */
    if (OSS_SUCCESS != OSS_Sender(&tSender))
    {
        return;
    }

    /* oprationflag 的第 1 bit 为 1 ,表示这是本轮查询的第一个请求,
     * 应从数据区的第一条记录开始组包,否则应从记录的当前查询位置
     * 开始继续组包。igmp_ddm_data_change 用于记录一轮查询期间
     * 前台数据是否已经发生变化,因此,在收到一轮查询的第一个请求
     * 时将其复位为0 。 */
    if(oprationflag & 0x01)
    {
        dwFrom   = 0;
        dwPortNo = 0;
    }

    /* 找到开始的 net */
    i = 0;
    for(n=If_list; n; n=n->link)
    {
        i++;
        if(i > dwPortNo)
        {
            break;
        }
```

```
    }

    /* 收集 IGMP 的配置信息 */
    i = 0;
    while(n)
    {
        if( !Interface_is_ethernet_port(n) || !Interface_is_banded
            (n) )
        {
            dwPortNo++;
            n = n->link;
            continue;
        }

        if( i > OAM_DDM_PROTOCOLDA_MAXNUM )
        {
            /* 一包满了,发出去再开始组另一包 */
            ptAck->ucResult   = OAM_DDM_SUCCESS;
            ptAck->dwItemFrom = dwFrom;
            ptAck->wItemNum   = i;
            OSS_SendAsynMsg(EV_DDM_PROTOCOLDA_QUERY_ACK,
                            (BYTE*)(ptAck),
                            dwMsglen,
                            COMM_RELIABLE,&tSender);
            dwFrom += i;
        }

        strcpy(ptAck->tProtocolItem[i].tIgmp.ucIfName, n->s_name);
        ptAck->tProtocolItem[i].tIgmp.ucModuleType = 1;
        ptAck->tProtocolItem[i].
            tIgmp.ucIsRouter = n->enable_igmp_router;

        i++;
        dwPortNo++;
        n = n->link;
    }

    /* 最后一包数据 */
    ptAck->ucResult   = OAM_DDM_SUCCESS;
    ptAck->wEndMark   = BRS_DDM_PACKET_END;
    ptAck->dwItemFrom = dwFrom;
    ptAck->wItemNum   = i;

    OSS_SendAsynMsg(EV_DDM_PROTOCOLDA_QUERY_REQ,(BYTE*)(ptAck),
                    dwMsglen,COMM_RELIABLE,&tSender);
}
```

1.9 请找出下面代码中的隐患或者错误,说明故障原因并改正。 10分

```
#define MAX_DB_BUF 40960

typedef struct
{
    INT32 LocLen;
    CHAR  Location[ MAX_DB_BUF + 1];
}CMD_OSPF_LOCATION;

typedef struct
{
    BYTE bacl_no_0;
    BYTE bexp_acl_no_1;
    BYTE bacl_name_2;
    BYTE bReserved;
}CMD_OSPF_DISTRIBUTE_LIST_PARA;

WORD32 cmd_ospf_distribute_list_para_check
    (INTER_CMD_STRU *pCmdStru, /* 入参 : 命令参数 */
     MSG_COMM_OAM *pRtnMsg)    /* 入出参 : 缓存区信息 */
{
    INT32 ospf_id = 0;
    CMD_OSPF_LOCATION tLoc;
    CMD_OSPF_DISTRIBUTE_LIST_EX   *pCmdPara = NULL;
    CMD_OSPF_DISTRIBUTE_LIST_PARA Distribute_List_DB;
    CMD_OSPF_DISTRIBUTE_LIST_PARA Distribute_List_INPUT;

    XOS_ASSERT(pCmdStru && pRtnMsg);
    if ((pCmdStru == NULL) && (pRtnMsg == NULL))
    {
        return ROSNG_PARAM_ERROR;
    }

    MEMSET(&tLoc, 0, sizeof(tLoc));

    /*得到参数值*/
    pCmdPara = (CMD_OSPF_DISTRIBUTE_LIST_EX *)(pCmdStru->CmdPara);
    Distribute_List_INPUT.bacl_no_0     = pCmdPara->bacl_no_0;
    Distribute_List_INPUT.bexp_acl_no_1 = pCmdPara->bexp_acl_no_1;
    Distribute_List_INPUT.bacl_name_2   = pCmdPara->bacl_name_2;

    /* check ospf_id */
    ospf_id == XOS_NtoH32(*(INT32 *) pCmdStru->sBackupBuf);
    if (ospf_id <1 || ospf_id> 65535)
    {
        pRtnMsg->ReturnCode = ERR_INPUT_PARA_WRONG;
        ROSNG_TRACE_ERROR("Invalid ospf instance id.\n");
        return ROSNG_CFG_ERR_PARA_CHECK_ERR;
```

```
    }

    /* extract location */
    OSPF_COMPOSE_LOCATION(&tLoc.Location, OSPF_DB_ROOT_NODE,
                          ospf_id, &tLoc.LocLen);

    /* check DB */
    if (Valid_Loc(Data_GetKey(0, tLoc)) != TRUE)
    {
        ROSNG_TRACE_ERROR("FILE[%s]:LINE[%d]There is no such ospf
            information in db [%s:%d]\n",
                          __FILE__, __LINE__, tLoc.Location);
        pRtnMsg->ReturnCode = ERR_INPUT_PARA_WRONG;
        return ROSNG_CFG_ERR_PARA_CHECK_ERR;
    }

    if (Data_Get(0, tLoc, (CHAR *) &Distribute_List_DB,
                 sizeof(Distribute_List_DB)) != DATA_GET_SUC)
    {
        ROSNG_TRACE_ERROR("FILE[%s]:LINE[%d]Read DB failed [%s].\n
            ", __FILE__, __LINE__,tLoc.Location);
        pRtnMsg->ReturnCode = ERR_GET_DB_FAIL;
        return ROSNG_CFG_ERR_PARA_CHECK_ERR;
    }

    if( MEMCMP(&Distribute_List_DB, &Distribute_List_INPUT,
               sizeof(CMD_OSPF_DISTRIBUTE_LIST_PARA)))
    {
        /* 不存在相同配置 */
        if (pCmdStru->bIfNo)
        {
            /*No 命令直接返回*/
            RETURN_WHILE_CHECK_RECONFIGURATION();
        }
    }
    else
    {
        /* 存在相同配置 */
        if (!pCmdStru->bIfNo)
        {
            /* 直接返回 */
            RETURN_WHILE_CHECK_RECONFIGURATION();
        }
    }

    return ROSNG_SUCCESS;
}
```

1.10 请找出下面代码中的隐患或者错误，说明故障原因并改正。 6分

```
/*****************************************************************
* 函数名称：H248_DelAllCallFromIadQueue
* 功能描述：将某终端的所有呼叫数据节点从终端的注册数据区中删除
*****************************************************************/
void H248_DelAllCallFromIadQueue( DWORD dwNodeId )
{
    H248_CTX_DATA *ptCall     = NULL;
    H248_CTX_DATA *ptTemp     = NULL;
    DWORD          dwTermStep = 0;
    DWORD          dwStmStep  = 0;
    DWORD          dwCallNum  = 0;

    gvH248_Statis.tCallToIad.dwDelAllCallFromIadQueueNum++;

    if( dwNodeId == 0 ||  dwNodeId >= gdwMGNodeNum )
    {
        gvH248_Statis.tCallToIad.
            dwDelAllCallFromIadQueueFailNum++;
        return;
    }

    /* 找到要删除的呼叫数据链表的头部 */
    ptCall = gvMGNode[dwNodeId].ptCallToIadQueue;

    /* 发现要释放的链表为空无需做什么， */
    if ( NULL == ptCall )
    {
        gvMGNode[dwNodeId].dwCallToIadNum = 0;
        return;
    }

    /* 从链表头部一直释放到尾部 */
    while ( NULL != ptCall && dwCallNum < gdwH248MaxCallCap )
    {
        dwCallNum++;

        /* 释放 rtp 资源 */
        gvH248_Statis.tCallToIad.dwDelAllCallRelRtpNum++;
        for( dwTermStep = 0;
             dwTermStep < ptCall->bTidNum
                 && dwTermStep < MAX_COMMANDS_NUM;
             dwTermStep++ )
        {
            if(ptCall->tTermData[dwTermStep].bTidType == H248_RTP)
            {
                for( dwStmStep = 0;
                     dwStmStep <
```

```
                        ptCall->tTermData[ dwTermStep].bStreamNum
                        && dwStmStep < H248_MAX_STREAM_NUM;
                    dwTermStep ++ )
                {
                    if ( !H248_RMRetRtp(
                    &ptCall->tTermData[ dwTermStep].
                             tStreamData[ dwStmStep].tRtpInfo.tRtp,
                    ptCall->tTermData[ dwTermStep].
                            tStreamData[ dwStmStep].tRtpInfo.wMlid,
                    ptCall->tTermData[ dwTermStep].
                           tStreamData[ dwStmStep].tRtpInfo.wMrid))
                    {
                        gvH248_Statis.tCallToIad.
                            dwDelAllCallRelRtpFailNum++;
                    }
                }
            }
        }

        /* 释放每个 DB 呼叫数据节点 */
        H248_ReleaseCallData( ptCall, FALSE, TRUE );

        /* 寻找下一个要释放的呼叫数据节点 */
        ptCall = ptCall->ptIadNext;
    }

    /* 已释放至链尾,此处正常退出 */
    gvMGNode[dwNodeId].ptCallToIadQueue = NULL;
    gvMGNode[dwNodeId].dwCallToIadNum   = 0;

    return;
}

/*****************************************************
- Function Name : H248_ReleaseCallData
- DESCRIPTION   : H248 封装的释放呼叫数据区的函数
*****************************************************/
void H248_ReleaseCallData( H248_CTX_DATA *ptCallData,
                           BOOL blInCallFlow,
                           BOOL blSyncCall )
{
    BOOL                blCaller = TRUE;
    WORD                wCdrIdx  = 0;
    /* H248_CALL_UNIT  *ptCallInfo = NULL;
       NOt Used, CRDCR00378057 */

    H248_ASSERT( NULL != ptCallData , H248_FILE_TRAN );
    if( NULL == ptCallData )
    {
```

```
        return;
    }

    H248_DelCallFromIadQueue( ptCallData );

    /* 释放3. DB 的呼叫数据区 */
    blCaller = ptCallData->blCaller; /*add for CRDCR00386426*/
    wCdrIdx  = ptCallData->wCdrIdx;
    ptCallData->wCdrIdx = H248_CDR_INVALIDIDX;
    H248_DB_RelCallDataBySeq( ptCallData->dwsequence,
                              ptCallData->dwMgId );

    /* 出4. CDR 话单 */
    if( blInCallFlow )
    {
        if( blCaller ) /* modify for CQ CRDCR00386426 */
        {
            H248_CDR_CALLER_HUNG( wCdrIdx );
        }
        else
        {
            H248_CDR_CALLEE_HUNG( wCdrIdx );
        }
    }
    else
    {
        H248_CDR_ABNORMAL_HUNG( wCdrIdx );
    }
    return;
}

/************************************************************
* 函数名称:H248_DelCallFromIadQueue
* 功能描述:将终端的某个呼叫数据节点从终端的呼叫数据队列中删除
************************************************************/
void H248_DelCallFromIadQueue( H248_CTX_DATA *ptCall )
{
    gvH248_Statis.tCallToIad.dwDelCallFromIadQueueNum++;

    if ( NULL == ptCall )
    {
        return;
    }

    if( ptCall->dwMgId == 0 || ptCall->dwMgId >= gdwMGNodeNum )
    {
        return;
    }
```

```
    /* 该呼叫数据节点是链表头部节点 */
    if (( ptCall == gvMGNode[ ptCall->dwMgId].ptCallToIadQueue )
        && ( ptCall->ptIadPrev == NULL ))
    {
        /*链表只有一个节点 */
        if ( ptCall->ptIadNext == NULL )
        {
            gvMGNode[ ptCall->dwMgId].ptCallToIadQueue = NULL;
        }
        /* 链表不是只有一个节点 */
        else
        {
            gvMGNode[ ptCall->dwMgId].ptCallToIadQueue =
                ptCall->ptIadNext;
        }

    }
    /* 该呼叫数据节点是链表尾部节点 */
    else if ( ptCall->ptIadNext == NULL )
    {
        /* 链表不是只有一个节点 */
        if ( ptCall->ptIadPrev != NULL )
        {
            ptCall->ptIadPrev->ptIadNext = NULL;
        }
    }
    /* 该呼叫数据节点在链表中部 */
    else
    {
        if ( ptCall->ptIadPrev != NULL
             && ptCall->ptIadNext != NULL )
        {
            ptCall->ptIadPrev->ptIadNext = ptCall->ptIadNext;
            ptCall->ptIadNext->ptIadPrev = ptCall->ptIadPrev;
        }
    }

    ptCall->ptIadPrev = NULL;
    ptCall->ptIadNext = NULL;

    gvMGNode[ ptCall->dwMgId].dwCallToIadNum--;

    return;
}
```

2 编程题 20分

2.1 编程求一字符串中最小字符的位置，并将该字符及后面子串中的小写字母转换成大写字母，输出转换后的字符串。如假设字符串为:qwertmn，则转换后的字符串为:qwERTMN。 8分

2.2 一个五位数字ABCDE*4=EDCBA，这五个数字不重复，请编程求出来。

12分