Coding
Coding for comprehensibility

#include <stdio.h>
char *T="IeJKLMaYQCE[ ]jbZRskc[SldU^V\X\|/_<[:90:"$434-.\/2>]s",
K[3][1000],*F,x,A,*g,N,Y,*Q,W,0,x,X(){r [r[r[3]]=M[1-(x&1)][*r=W,1],255,[*r+=Y,*g+=((x[13]>>1)>>1)>1]-1]*r=r[x>>3],(1+*r[3])>>1);}E(){A|X(x=0,g=r[ ]=r[=*]*W,1]=*Q+2,1]=x+1+Y,*g++=((((x& 7) -1)>>1)-1)?*r:r[x>>3],(1+*r[3])>>1);}l(){E(--q&&l());}B(){*J&&B((D=*J,Q[2]<D&&D<k[1]&&(*g+=q), !(D-W&&D-9&&D-10&&D-13)&&(!*r&&(*g++=D-95)))&&D>=97&&*r=r[3]+123&&(*r[9]+D-95)||!(D-k[1]))&&(*r[4]=D-32)||D>k[3]&&&K[-1]&&(*r=1]];t(){(j(A=(*K)[D*W+x]),++,q<(*q+1)&t&&t();}R(){(A=(t(q=0),,n),j,++,q<(*q+1)&t&&t();}O(){(j(0,=q)&&(O(g=-q)));C(){(j=gets(K2),*r=K[2],q=K[2],r[1];Y=N<<(*r=!A))};main(){C((l((K=A[M]=(F=(k=(M[!A]=(Q=T+(q=(Y=(W=32)-(N=4))))(+7)+7),Y=N<<(*r=!A))));)}
Can code be self documenting?

- Incorrect comments are worse than missing comments
- Comments should not repeat what is clear from the code
- Code should be written to minimize the need for comments
- Code that is too complicated to explain should be rewritten
// This whole thing is too complicated for me to understand or explain, but
// here is where the actual work takes place, I think.
private void ListenerWorker(RTPListener.RTPStream rtpStream){
    try {
        rtpListener.Subscribe(rtpStream.SSRC);
        Listen(rtpStream);
    }
    catch (System.Exception se){
        LogEvent(se.ToString(), EventLogEntryType.Error);
    }
}
Commenting

- Comment data declarations, including units and ranges
- Comment meanings of control structures
- Avoid commenting structures that are difficult to maintain
- Write comments before, or while coding, not afterwords!
memcpy

/* memcpy -- copy a block of size bytes from pvFrom to pvTo */

void *memcpy(void *pvTo, *void pvFrom, size_t size){

    return pvTo;
}

void *memcpy(void *pvTo, void *pvFrom, size_t size)
{
    byte *pbTo = (byte *)pvTo;
    byte *pbFrom = (byte *)pvFrom;

    while (size-- > 0)
    {
        *pbTo++ = *pbFrom++;
    }
    return (pvTo);
}
memcpy

void *memcpy(void *pvTo, void *pvFrom, size_t size)
{
    byte *pbTo = (byte *)pvTo;
    byte *pbFrom = (byte *)pvFrom;
    if (pvTo == NULL || pvFrom == NULL)
    {
        fprintf(stderr, "Bad args in memcpy\n");
        abort();
    }
    while (size-- > 0)
    {
        *pbTo++ = *pbFrom++;
    }
    return (pvTo);
}
No errors here...

- ..but it’s bigger and slower
- So, exploit the preprocessor
memcpy II

```c
void *memcpy(void *pvTo, void *pvFrom, size_t size)
{
    byte *pbTo = (byte *)pvTo;
    byte *pbFrom = (byte *)pvFrom;
    #ifdef DEBUG
    if (pvTo == NULL || pvFrom == NULL)
    {
        fprintf(stderr, "Bad args in memcpy\n");
        abort();
    }
    #endif
    while (size-- > 0)
    {
        *pbTo++ = *pbFrom++;
    }
    return pvTo;
}
```
memcpy III

```c
void *memcpy(void *pvTo, void *pvFrom, size_t size)
{
    byte *pbTo = (byte *)pvTo;
    byte *pbFrom = (byte *)pvFrom;

    assert(pvTo != NULL && pvFrom != NULL);

    while (size-- > 0)
    {
        *pbTo++ = *pbFrom++;
    }
    return pvTo;
}
```

- Assertions can be turned on and off
  - You probably shouldn't consider rewriting the assert macro
memcpy IV

void *memcpy(void *pvTo, void *pvFrom, size_t size)
{
    byte *pbTo = (byte *)pvTo;
    byte *pbFrom = (byte *)pvFrom;

    assert(pvTo != NULL && pvFrom != NULL);
    assert(pbTo >= pbFrom+size || pbFrom >= pbTo+size);

    while (size-- > 0)
    {
        *pbTo++ = *pbFrom++;
    }

    return pvTo;
}
assertions

- Don’t use assertions to check unusual conditions
  - You need explicit error code for this
- Only use them to ensure that illegal conditions are avoided
Memory

- The memcpy examples are from Writing Solid Code: Microsoft's Techniques for Developing Bug-Free C Programs
- Although the book is general, lots of the guidelines focus on memory issues
  - Marking freed memory
  - Not accessing freed memory
  - Dealing with details of realloc
- These are real issues, but appear less frequently in other languages
Writing solid code

- Shred your garbage

```c
void FreeMemory(void *pv){
    Assert(pv != NULL);
    memset(pv, 0xA3, sizeofBlock(pv);
    free(pv);
}
```

- Force early failure, increase determinism
- Why 0xA3?
Should debug code be left in shipped version

- **Pro:**
  - Debug code useful for maintenance
  - Removing debug code change behavior
    - Bugs in release but not debug versions

- **Con:**
  - Efficiency issues
  - Different behavior for debug vs. release
    - Early fail vs. recover