End of Line

• There are two characters that determine end-of-line
  – Carriage return (CR, \r, ^M)
  – Line feed (LF, \n)
• Early operating systems defined their own conventions using one or both of CR and LF.
  – Unix: LF,
  – DOS/Windows: CR LF
  – Mac classic: CR
Network Line Ending

• Transferring data between machines with different operating systems, means deciding on a common line ending.

• CR LF is the standard

• (Of course it is possible with regular expression matching to mostly ignore this issue, but still better to conform.)
Byte order

- **Big-endian**

  \[
  91,329 = \begin{array}{cccc}
  A & A+1 & A+2 & A+3 \\
  00 & 01 & 64 & C1 \\
  \end{array}
  \]

- **Little-endian**

  \[
  91,329 = \begin{array}{cccc}
  A+3 & A+2 & A+1 & A \\
  00 & 01 & 64 & C1 \\
  \end{array}
  \]

- Intel is little-endian, and Sparc is big-endian
Network byte order

- To communicate between machines with unknown or different “endian-ness” we convert numbers to network byte order (big-endian) before we send them.
- There are functions provided to do this:
  - `unsigned long htonl(unsigned long)`
  - `unsigned short htons(unsigned short)`
  - `unsigned long ntohl(unsigned long)`
  - `unsigned short ntohs(unsigned short)`
Arrays of bit strings

- FD_SETSIZE is bigger than 32.

```c
struct bits {
    unsigned int field[N];
};

typedef struct bits Bitstring;
Bitstring a, b;
setzero(&a);
b = a;
a.field[0] = ~0;
```
Setting and unsetting

int set(unsigned int bit, Bitstring *b) {
    int index = bit / 32;
    b->field[index] |= 1 << (bit % 32);
    return 1;
}

int unset(unsigned int bit, Bitstring *b) {
    int index = bit / 32;
    b->field[index] &= ~(1 << (bit % 32));
}
Testing and emptying

```c
int ifset(unsigned int bit, Bitstring *b) {
    int index = bit / 32;
    return ( (1 << (bit % 32)) & b->field[index]);
}

int setzero(Bitstring *b){
    if(memset(b,0,sizeof(Bitstring)) == NULL)
        return 0;
    else
        return 1;
}
```
char *intToBinary(unsigned int number) {
    char *binaryString = malloc(32+1);
    int i;
    binaryString[32] = '\0';
    for (i = 31; i >= 0; i--) {
        binaryString[i] = ((number & 1) + '0');
        number = number >> 1;
    }
    return binaryString;
}