Serial Interface in C

Application to Magswipe

UCSD: Physics 121; 2012

Lecture Content

- In this lecture, we have a broken-up a C-code that reads magswipe data into its composite chunks
- We will go through these chunks, describing their functions and bits of new stuff
- Your job will be to put the chunks together and get it all to work
- We start with a template of where things go…

Winter 2012 2

UCSD: Physics 121; 2012

Magswipe Handling Program Outline

```
#include <various things, including conio.h>

int main(int argc, char* argv[])
{
  // type definitions

  // open serial port as COMXX (COM1 if built-in)

  // establish whether 5 or 7 bits through argv[1], and pick mask

  while (!kbhit())
  {
    ReadFile(hSerial, sInBuff, 1, &dwBytesRead, NULL);
    if (dwBytesRead > 0)
    {
      // apply masks
      // parity check
      // LRC calculation
      // string formatting
    }
  }
  // print results
  CloseHandle(hSerial);
  return 0;
}
```

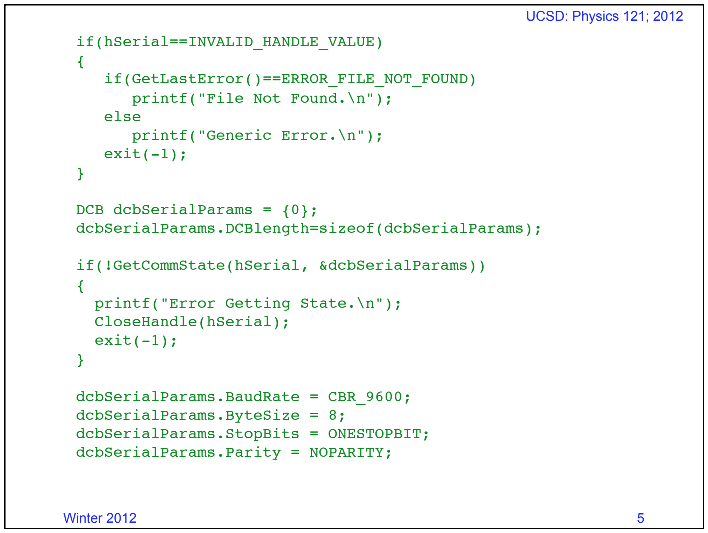
Winter 2012 3

UCSD: Physics 121; 2012

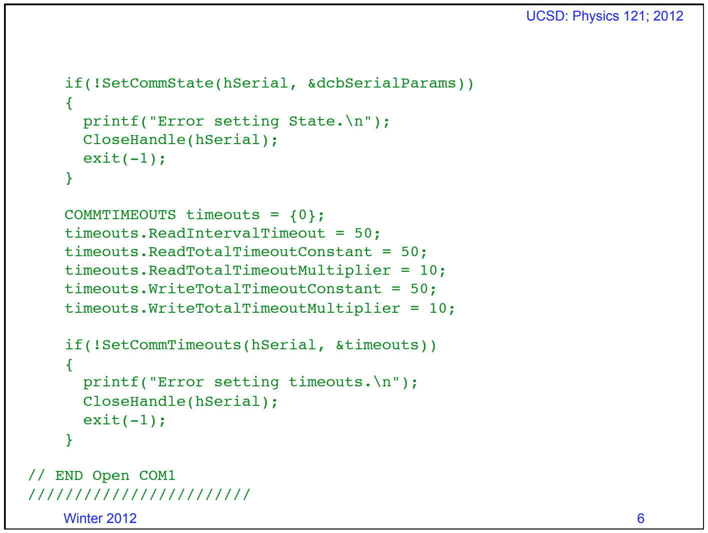
Serial Port Access in Windows (in 3 pieces)

```
#include <fcntl.h>
#include <errno.h>
#include <windows.h>

////////////////////////
// Open COMXX (COM1 if built-in)

    HANDLE hSerial;

    hSerial = CreateFile("COMXX",
            GENERIC_READ | GENERIC_WRITE,
            0,
            0,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            0);
```

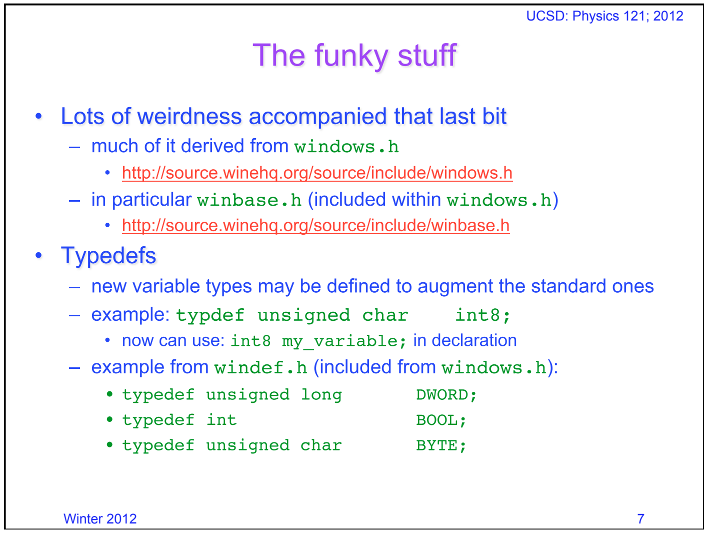
Winter 2012 Windows serial code worked out by Dan Creveling 4



```
if(hSerial==INVALID_HANDLE_VALUE)
{
   if(GetLastError()==ERROR_FILE_NOT_FOUND)
      printf("File Not Found.\n");
   else
      printf("Generic Error.\n");
   exit(-1);
}

DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength=sizeof(dcbSerialParams);

if(!GetCommState(hSerial, &dcbSerialParams))
{
  printf("Error Getting State.\n");
  CloseHandle(hSerial);
  exit(-1);
}

dcbSerialParams.BaudRate = CBR_9600;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;
```

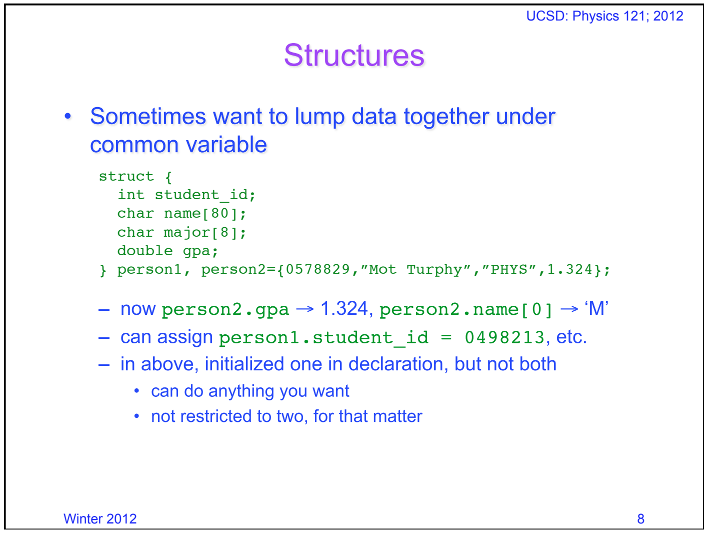


```
    if(!SetCommState(hSerial, &dcbSerialParams))
    {
      printf("Error setting State.\n");
      CloseHandle(hSerial);
      exit(-1);
    }

    COMMTIMEOUTS timeouts = {0};
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;

    if(!SetCommTimeouts(hSerial, &timeouts))
    {
      printf("Error setting timeouts.\n");
      CloseHandle(hSerial);
      exit(-1);
    }

// END Open COM1
////////////////////////
```



The funky stuff

- Lots of weirdness accompanied that last bit
  - much of it derived from `windows.h`
    - http://source.winehq.org/source/include/windows.h
  - in particular `winbase.h` (included within `windows.h`)
    - http://source.winehq.org/source/include/winbase.h
- Typedefs
  - new variable types may be defined to augment the standard ones
  - example: `typdef unsigned char    int8;`
    - now can use: `int8 my_variable;` in declaration
  - example from `windef.h` (included from `windows.h`):
    - `typedef unsigned long       DWORD;`
    - `typedef int                 BOOL;`
    - `typedef unsigned char       BYTE;`

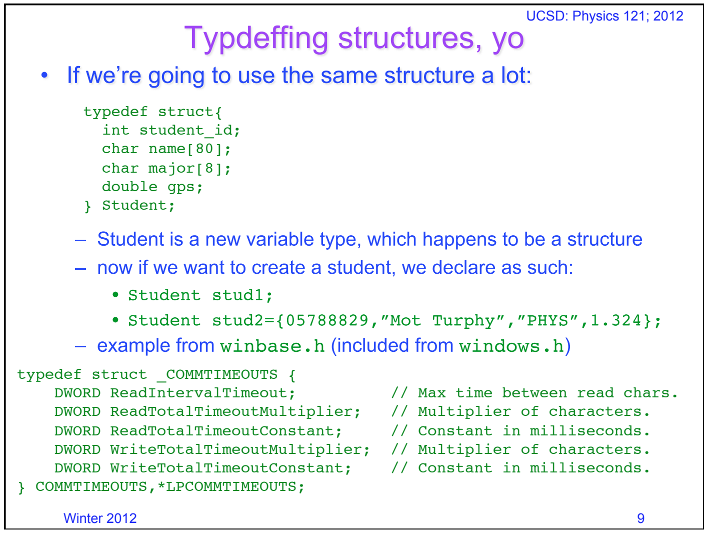


Structures

- Sometimes want to lump data together under common variable

```
struct {
  int student_id;
  char name[80];
  char major[8];
  double gpa;
} person1, person2={0578829,"Mot Turphy","PHYS",1.324};
```

  - now `person2.gpa` → 1.324, `person2.name[0]` → ‘M’
  - can assign `person1.student_id = 0498213`, etc.
  - in above, initialized one in declaration, but not both
    - can do anything you want
    - not restricted to two, for that matter



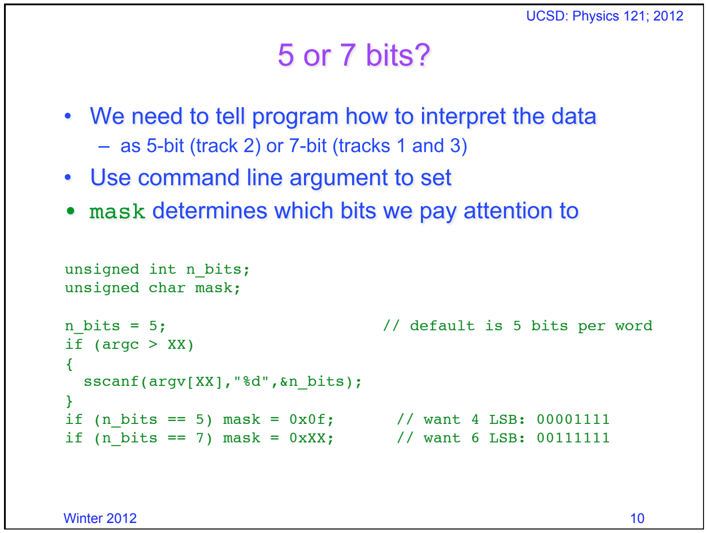
Typdeffing structures, yo

- If we're going to use the same structure a lot:

```
typedef struct{
  int student_id;
  char name[80];
  char major[8];
  double gps;
} Student;
```

  - Student is a new variable type, which happens to be a structure
  - now if we want to create a student, we declare as such:
    - `Student stud1;`
    - `Student stud2={05788829,"Mot Turphy","PHYS",1.324};`
  - example from `winbase.h` (included from `windows.h`)

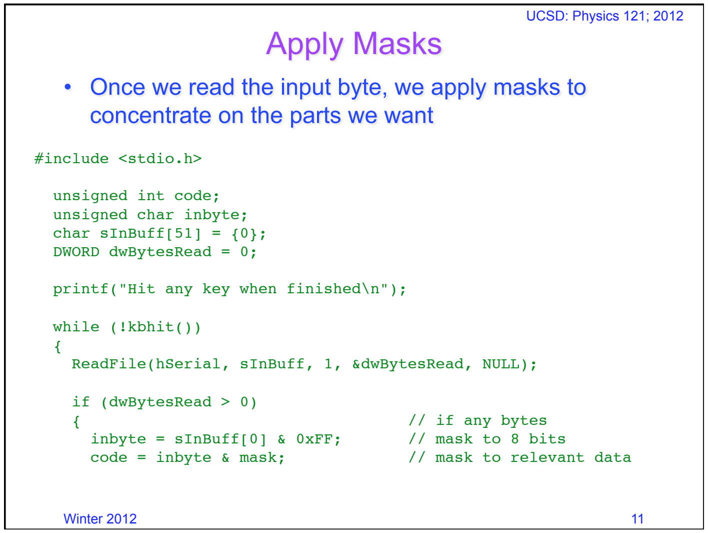
```
typedef struct _COMMTIMEOUTS {
    DWORD ReadIntervalTimeout;          // Max time between read chars.
    DWORD ReadTotalTimeoutMultiplier;   // Multiplier of characters.
    DWORD ReadTotalTimeoutConstant;     // Constant in milliseconds.
    DWORD WriteTotalTimeoutMultiplier;  // Multiplier of characters.
    DWORD WriteTotalTimeoutConstant;    // Constant in milliseconds.
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```



5 or 7 bits?

- We need to tell program how to interpret the data
  - as 5-bit (track 2) or 7-bit (tracks 1 and 3)
- Use command line argument to set
- `mask` determines which bits we pay attention to

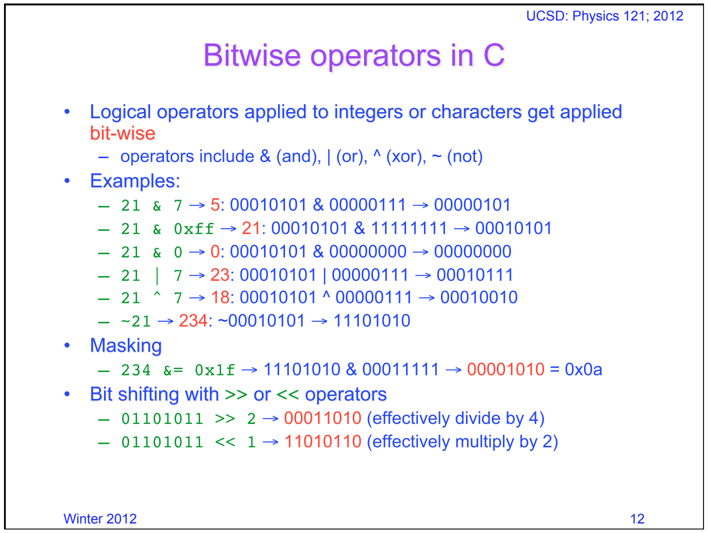
```
unsigned int n_bits;
unsigned char mask;

n_bits = 5;                      // default is 5 bits per word
if (argc > XX)
{
  sscanf(argv[XX],"%d",&n_bits);
}
if (n_bits == 5) mask = 0x0f;       // want 4 LSB: 00001111
if (n_bits == 7) mask = 0xXX;       // want 6 LSB: 00111111
```



Apply Masks

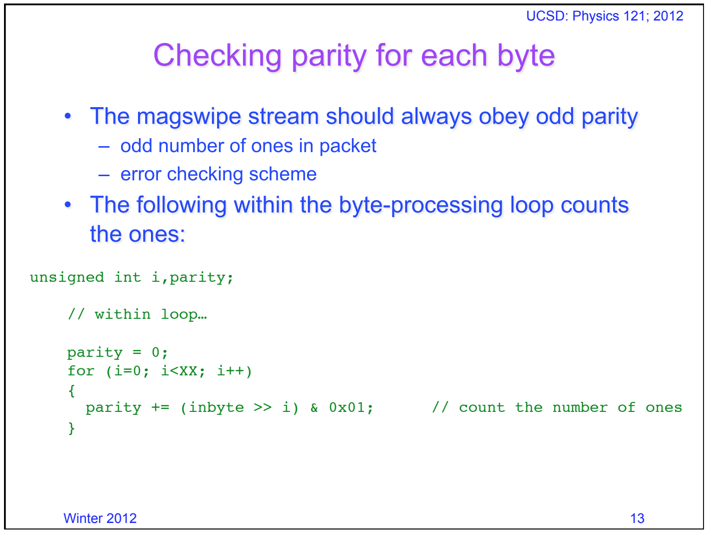
- Once we read the input byte, we apply masks to concentrate on the parts we want

```
#include <stdio.h>

  unsigned int code;
  unsigned char inbyte;
  char sInBuff[51] = {0};
  DWORD dwBytesRead = 0;

  printf("Hit any key when finished\n");

  while (!kbhit())
  {
    ReadFile(hSerial, sInBuff, 1, &dwBytesRead, NULL);

    if (dwBytesRead > 0)
    {                                  // if any bytes
      inbyte = sInBuff[0] & 0xFF;      // mask to 8 bits
      code = inbyte & mask;            // mask to relevant data
```



Bitwise operators in C

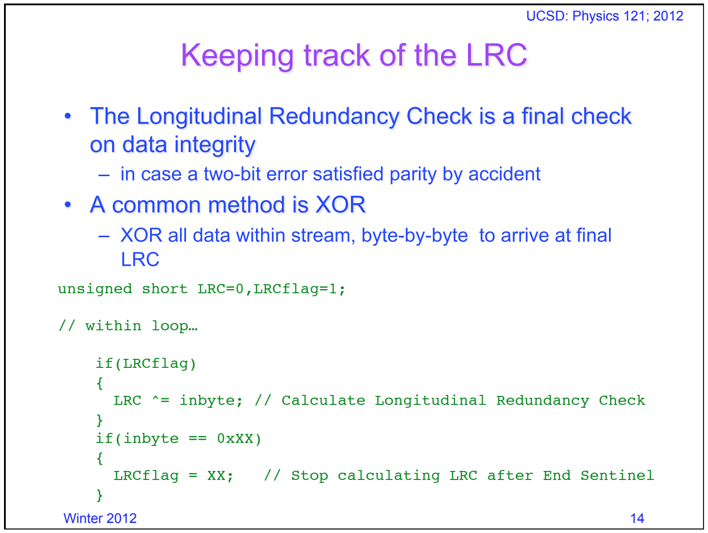
- Logical operators applied to integers or characters get applied bit-wise
  - operators include & (and), | (or), ^ (xor), ~ (not)
- Examples:
  - `21 & 7` → 5: 00010101 & 00000111 → 00000101
  - `21 & 0xff` → 21: 00010101 & 11111111 → 00010101
  - `21 & 0` → 0: 00010101 & 00000000 → 00000000
  - `21 | 7` → 23: 00010101 | 00000111 → 00010111
  - `21 ^ 7` → 18: 00010101 ^ 00000111 → 00010010
  - `~21` → 234: ~00010101 → 11101010
- Masking
  - `234 &= 0x1f` → 11101010 & 00011111 → 00001010 = 0x0a
- Bit shifting with >> or << operators
  - `01101011 >> 2` → 00011010 (effectively divide by 4)
  - `01101011 << 1` → 11010110 (effectively multiply by 2)



Checking parity for each byte

- The magswipe stream should always obey odd parity
  - odd number of ones in packet
  - error checking scheme
- The following within the byte-processing loop counts the ones:

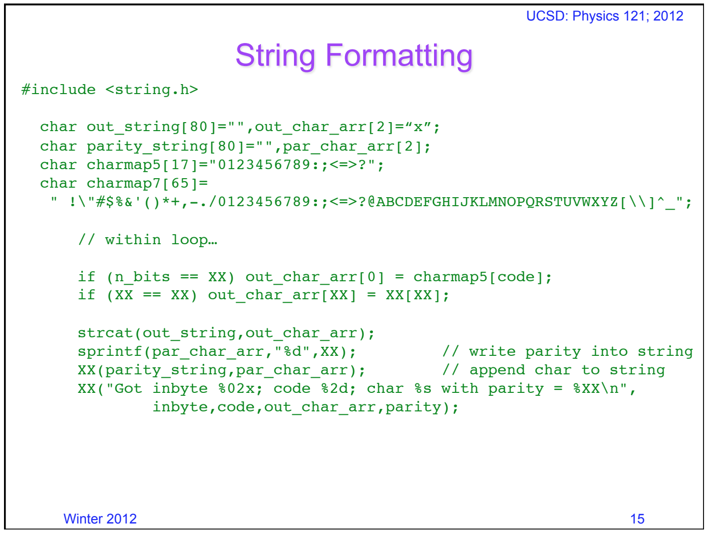
```
unsigned int i,parity;

    // within loop…

    parity = 0;
    for (i=0; i<XX; i++)
    {
      parity += (inbyte >> i) & 0x01;      // count the number of ones
    }
```



Keeping track of the LRC

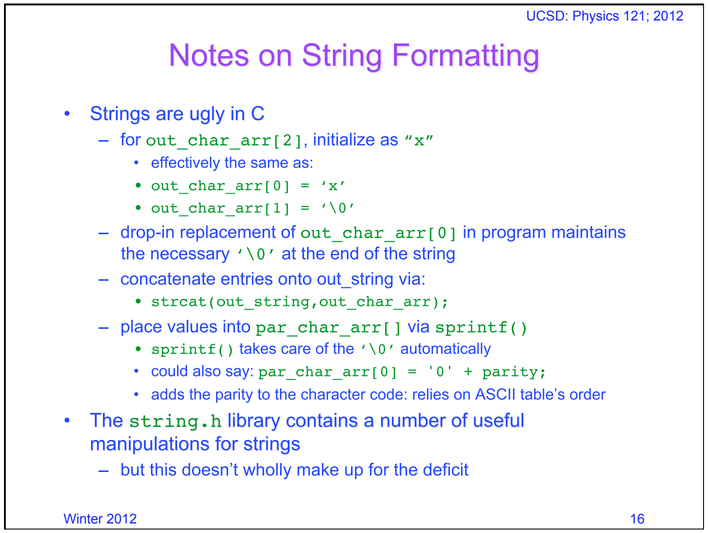
- The Longitudinal Redundancy Check is a final check on data integrity
  - in case a two-bit error satisfied parity by accident
- A common method is XOR
  - XOR all data within stream, byte-by-byte to arrive at final LRC

```
unsigned short LRC=0,LRCflag=1;

// within loop…

    if(LRCflag)
    {
      LRC ^= inbyte; // Calculate Longitudinal Redundancy Check
    }
    if(inbyte == 0xXX)
    {
      LRCflag = XX;   // Stop calculating LRC after End Sentinel
    }
```



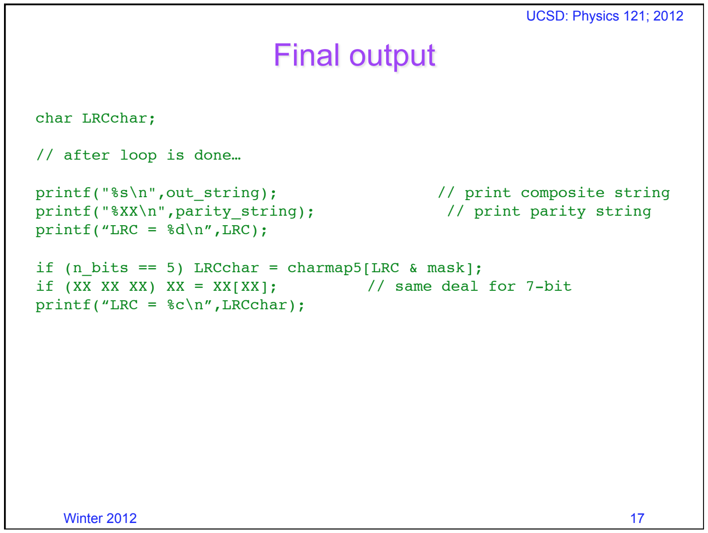
String Formatting

```
#include <string.h>

  char out_string[80]="",out_char_arr[2]=“x”;
  char parity_string[80]="",par_char_arr[2];
  char charmap5[17]="0123456789:;<=>?";
  char charmap7[65]=
   " !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_";

      // within loop…

      if (n_bits == XX) out_char_arr[0] = charmap5[code];
      if (XX == XX) out_char_arr[XX] = XX[XX];

      strcat(out_string,out_char_arr);
      sprintf(par_char_arr,"%d",XX);         // write parity into string
      XX(parity_string,par_char_arr);        // append char to string
      XX("Got inbyte %02x; code %2d; char %s with parity = %XX\n",
              inbyte,code,out_char_arr,parity);
```



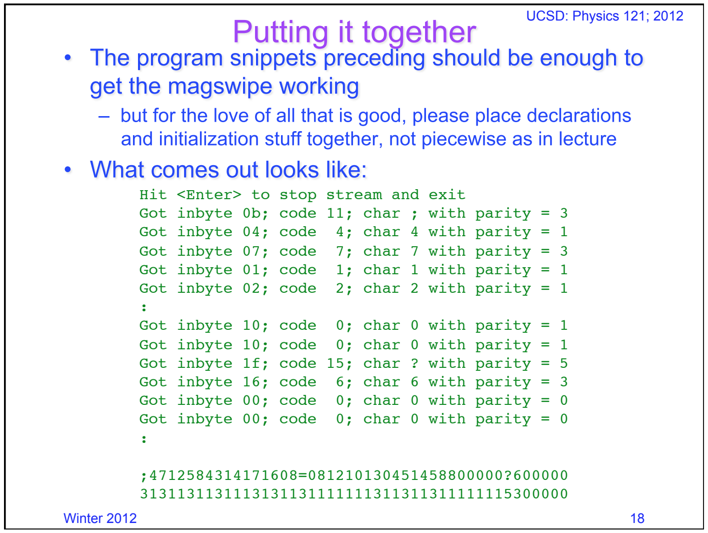
Notes on String Formatting

- Strings are ugly in C
  - for `out_char_arr[2]`, initialize as `“x”`
    - effectively the same as:
    - `out_char_arr[0] = ‘x’`
    - `out_char_arr[1] = ‘\0’`
  - drop-in replacement of `out_char_arr[0]` in program maintains the necessary `‘\0’` at the end of the string
  - concatenate entries onto out_string via:
    - `strcat(out_string,out_char_arr);`
  - place values into `par_char_arr[]` via `sprintf()`
    - `sprintf()` takes care of the `‘\0’` automatically
    - could also say: `par_char_arr[0] = '0' + parity;`
    - adds the parity to the character code: relies on ASCII table’s order
- The `string.h` library contains a number of useful manipulations for strings
  - but this doesn’t wholly make up for the deficit



Final output

```
char LRCchar;

// after loop is done…

printf("%s\n",out_string);                 // print composite string
printf("%XX\n",parity_string);              // print parity string
printf(“LRC = %d\n”,LRC);

if (n_bits == 5) LRCchar = charmap5[LRC & mask];
if (XX XX XX) XX = XX[XX];         // same deal for 7-bit
printf(“LRC = %c\n”,LRCchar);
```



Putting it together

- The program snippets preceding should be enough to get the magswipe working
  - but for the love of all that is good, please place declarations and initialization stuff together, not piecewise as in lecture
- What comes out looks like:

```
Hit <Enter> to stop stream and exit
Got inbyte 0b; code 11; char ; with parity = 3
Got inbyte 04; code  4; char 4 with parity = 1
Got inbyte 07; code  7; char 7 with parity = 3
Got inbyte 01; code  1; char 1 with parity = 1
Got inbyte 02; code  2; char 2 with parity = 1
:
Got inbyte 10; code  0; char 0 with parity = 1
Got inbyte 10; code  0; char 0 with parity = 1
Got inbyte 1f; code 15; char ? with parity = 5
Got inbyte 16; code  6; char 6 with parity = 3
Got inbyte 00; code  0; char 0 with parity = 0
Got inbyte 00; code  0; char 0 with parity = 0
:

;4712584314171608=0812101304514588000000?600000
3131131131113131131111111131131131111111115300000
```