All useful programs perform some form of Input/Output (I/O)

We have been using unformatted I/O to read from keyboard and write to screen

Other I/O options are available

- Formatted I/O to control format of data
- Batch I/O to read and write to files instead of keyboard/screen
Input/Output

• In C, I/O is performed by calling library functions
  
  ➢ The `printf` function
    – Performs output to standard output device (monitor)
    – Requires format string which provides text/values to print out and specifications on how to print them out
    – Manages the process of converting the bit pattern into proper sequence of ASCII characters based on formal spec provided by programmer
  
  ➢ The `scanf` function
    – Performs input from standard input device (keyboard)
    – Requires format string and list of addresses of variables into which the values retrieved from the input device will be stored
    – Converts the input according to conversion characters in format string and assigns converted values to the contents of the variables listed
Standard Input/Output

• Using `printf` or `scanf`, data can be formatted in many different ways
  ➢ Scientific notation
  ➢ Right justified
  ➢ Left justified
  ➢ Filled with a fill character
  ➢ Specify number of positions
  ➢ Etc.

• Refer to Chapter 7 in Uckan to determine how to accomplish these formats
Unformatted vs Formatted Output

• With unformatted output, values are printed in minimum predefined amount of space, beginning at current cursor position

  ```c
  printf ("%f", 0.511);    \rightarrow 0.511000
  printf ("%f", 365.3);   \rightarrow 365.3000
  printf ("%d", 42);      \rightarrow 42
  ```

• To specify field width, insert decimal constant before conversion character

  ```c
  printf ("%8d", 42);         \rightarrow 42
  ```
 ➢ Default pad character is a space
  ➢ To make default pad character a 0, set first digit of field width to 0

  ```c
  printf ("%08d", 42);       \rightarrow 00000042
  ```
Format Specifiers for Formatted Output

- **Format specifiers** (a.k.a. Flag Characters) are optional format modifiers that specify alignment, +/- and scientific notation

  - **Left Justification**
    
    ```
    printf ("%-8d", 42);  \Rightarrow 42^^^^^^
    ```

  - **Prefix number with + or -** (default is no + for positive numbers)
    
    ```
    printf ("%+8d", 42);  \Rightarrow ^^^^^^+42
    ```

  - **(space) Prefix number with space or -** (default is no space for positive numbers)
    
    ```
    printf ("% 8d", 42);  \Rightarrow ^^^^^^^42
    ```

  - **Precision** - specify number of places to print after decimal point
    
    ```
    printf ("%10.1f", 0.511);  \Rightarrow ^^^^^^^^0.5
    ```

  - **Scientific Notation**
    
    ```
    printf ("%10.3e", 0.511);  \Rightarrow ^5.110e-01
    ```
Non-interactive I/O - Batch Programs

- Not all programs interact with a user
  - IRS tax return audit program
- Instead of using keyboard input and monitor output, data can be read from a data file and written to a data file
- Programs that access files and perform non-interactive I/O are called batch programs
Streams

- C performs I/O through streams that are associated with files or devices
- A stream is an abstraction for a flow of data from a source to a sink
- A stream consists of an ordered series of bytes
- Reading/writing to a file or device involves reading from/writing to a stream
- Three standard streams are automatically opened for every C program
  - stdin - Standard Input Stream, usually the keyboard
  - stdout - Standard Output Stream, usually the monitor
  - stderr - Standard Error Stream, usually the monitor
File Input/Output - Declaring Files

- Some programs require ability to read/write ASCII data files
- To perform I/O operations, associate a stream with a file or device
  - Declare a pointer to a structure type called FILE (all caps!). FILE is defined within header file stdio.h

  Format:
  ```c
  FILE *<internal_name>; /* identify a stream name*/
  FILE *test_data; /*test_data is internal file name*/
  ```

  This creates a file pointer -- pointer to the FILE structure, which provides operating system with bookkeeping information

  FILE structure contains fields to hold information such as file's name, access mode, and pointer to next character in the stream (file position indicator)

  File pointer maps a particular stream to a particular file or device

  Use file pointer to read from, write to, or close the stream
File Input/Output - Opening Files

➢ Before you can read from or write to a file, you must open it with the \texttt{fopen()} function

➢ \texttt{fopen()} maps the file pointer to a physical file/device. For files, maps external file name to internal file name

➢ Specify how the file will be accessed; e.g., read, write, append

\begin{verbatim}
Format: 
<internal_name> = fopen("<external_name>", "<access_mode>");
\end{verbatim}

Example:
\begin{verbatim}
test_data = fopen("simresults.dat", "r"); 
    /*open simresults.dat as read only*/
\end{verbatim}

➢ This establishes \texttt{test_data} as an output stream between program and external file \texttt{simresults.dat}
File Input/Output - Read/Write

- Once file is opened, use file pointer to perform read/write operations
  - One object at a time - fscanf(), fprintf()
  - One character at a time - fgetc(), fputc()
  - One line at a time - fgets(), fputs()
    - fgets() reads characters until it reaches new line, end-of-file, or maximum number of characters specified
    - fputs() writes to the file and then inserts a new-line character
  - One block at a time - fread(), fwrite()
File Input/Output - Closing Files

- Files that are opened in a program should be closed after processing is complete
  - `fclose()` cuts the connection between the program and the file

  Format:
  ```c
  fclose(<internal_name>);
  ```

  Example:
  ```c
  fclose(test_data); /*close file that test_data was pointing to: i.e., simresults.dat*/
  ```

- File pointer no longer exists. Program cannot access file until it has been re-opened
File Input/Output - Error Handling

- Each I/O function returns a special value if an error occurs
  - If successful, `fopen()` function returns a file pointer to the physical file that can be used to access the file later in the program. If not successful, returns null.
  - It is a good practice to check that `fopen()` was successful

```c
FILE *test_data;
test_dat = fopen("simresults.dat", "r");
if (test_data == NULL)
    printf("Error opening simresults.dat\n");
```

- Caution: Error value varies from one function to another. Some functions return zero for error, others return non-zero value
File Input/Output - Error Handling (cont.)

- A stream's error status can be checked via pre-defined error functions
  
  - `feof()` checks whether an end-of-file was encountered during previous read operation
    - Attempting to read data past the end-of-file marker will cause an end-of-file condition
    - Can be used in while loop to read data from a file and check for end of data
  
  - `ferror()` returns integer error code if error occurred while reading/writing to a stream
Communicating with Peripherals

• In addition to communicating with files, programs can also communicate with peripherals -- external devices such as motors and sensors

• When communicating with a peripheral, processor usually issues commands to the device and waits for device to complete the assigned task

• Two basic techniques are used to communicate with peripherals
  ➢ Polling - processor sends command and keep asking device task has been completed
  ➢ Interrupts - processor sends command and continues working until device sends a "completed" signal
Communicating with Peripherals - Polling

- Example: Processor commands solar array to rotate 90°. After command is sent, processor wants to know when rotation is finished.

- Polling: Processor repeatedly checks to see if task has been completed. Peripheral response "Yes" or "No"
  - Similar to "Are we there yet?"
  - Processor spends large amount of otherwise useful time asking the question and getting a negative response.
  - To implement in software, create a loop that reads the status register of the device.

- Pro: Guaranteed response time - device responds with "Yes" or "No" within predetermined deadline. Typically used for data acquisition.

- Con: Inefficient - processor spends large amount of time asking question and getting negative response.
Communicating with Peripherals - Interrupts

- Interrupts are asynchronous electrical signals from a peripheral to the processor
  - Processor issues commands to peripheral as before, but then waits for an interrupt from the peripheral to signal completion of the work
  - While processor is waiting for interrupt, it is free to continue working on other things.
  - When interrupt signal is asserted,
    - Processor temporarily sets aside current work
    - Executes a small piece of code called Interrupt Service Routine (ISR)
    - Returns to work that was interrupted

- Pro: More efficient use of processor - uses waiting time to perform other work

- Con: Response time harder to predict - interrupt could happen at any time. Also, overhead involved in servicing interrupt.
Polling vs Interrupts

**Polling**

- **Processor**
  - Execute task
  - done?
  - done?
  - done?
  - done?
  - done?
  - transmit result

- **Peripheral**
  - executing task
  - No
  - No
  - No
  - No
  - Yes
  - here is result

**Interrupt**

- **Processor**
  - Execute task
  - run other tasks
  - stop tasks, run ISR
  - done

- **Peripheral**
  - executing task
  - send interrupt
  - transmit result
  - here is result
Device Drivers

• Each external peripheral device has an associated software module called a device driver

• A device driver is a way to hide the hardware from the program

• A device driver is a collection of software routines that
  - Controls the operation of the peripheral
  - Isolates the application software from the details of that particular hardware device

• Modularity is key
  - Structure of overall software is easier to understand
  - Because only one module interacts directly with peripheral, state of hardware can be more accurately tracked
  - Software changes that result from hardware changes are localized to device driver module
Watching Over the Health of the Processor - Watchdog Timers

- A **watchdog timer** is a special piece of hardware designed to monitor an embedded system and detect if the software hangs

- How it works

  - Watchdog timer counts down from a large number to zero. This process takes a predetermined amount of time to complete (typically 0.5 - 2 secs for spacecraft processor watchdogs)
  
  - Embedded processor routinely resets the counter to the originally large number.
    
    - E.g., on TOMS-EP spacecraft, watchdog counter timed-out in 0.5 seconds. 80186 processor reset the watchdog every 0.25 seconds.

  - If the counter reaches zero, the watchdog timer resets the embedded processor, which restarts the software

  - Common way for spacecraft to recover from unexpected software hangs
Review

• Many forms of I/O
  ➢ To/from standard I/O (keyboard, monitor) - unformatted and formatted
  ➢ To/from files
  ➢ To/from peripherals - polling vs interrupt techniques, device drivers
  ➢ Monitoring the processor

• Wed and Fri lectures by Prof. Crawley on Serial/Parallel I/O, and A/D and D/A conversion

• Answers to Mud submissions have been posted
  ➢ How to access Handyboard LEDs
  ➢ How to clear out main function on Handyboard