The Standard I/O Library

```
#include <stdio.h>
```

All I/O involves a stream of data, there are three standard streams that are defined in stdio.h

```
stdin  - the standard input, for reading
stdout - the standard output, for writing
stderr - for error messages
```

You can declare a pointer to your own stream in the following way:

```
FILE* fp;
```

Use the fopen() procedure to attach a file for reading, writing or appending data.

```
FILE* fopen(char* FileName, char* AccessMode);
```

The most popular access modes are:

```
"r" - a read-only file
"w" - write, starting at the front
"a" - write, appending to the end
```

- Thus we can open local file "tally.data" for reading with the command
  ```
  fp = fopen ("tally.data", "r");
  ```
- A value of NULL is returned if the file can't be opened

Use fclose() to flush buffered data to the file

```
fclose(fp);
```

Formatted Input and Output

Example

```
int  i, n;
float  f;
double  d;
char  str[30];

n = scanf("%d %f %lf %s", &i, &f, &d, &str[0]);
```

- If this call is successful the value of n will be 4
- This format expects to see one integer, two floating point numbers and a string. The first floating point number is stored in a float and the second one is stored in a double--we used %lf as the format for that.
- If we didn't use the %lf format for d, the value stored in d would be incorrect
  All the parameters have an & so we are passing a pointer instead of the value. We could replace &str[0] by str, since str is already a pointer--and this is often done.

In addition to scanf() and printf() we have the prototypes:

```
int fscanf (FILE* fp, const char* format, *arg1, *arg2, ... );
int fprintf (FILE* fp, const char* format, expr1, expr2, .. );

FILE* fp = fopen ("tally.data", "r");

n = fscanf (fp, "%d %f %lf %s", &i, &f, &d, str);
```

Beware use of %s to input an array. Stops at first blank.

Use fgets() or NOT gets() for string of characters input.

File Opening Example

```
FILE* fp;  FILE* fptr;
scanf ("%d", &item);
fscanf (stdin, "%d", &item);

    fp = fopen ("tally.data", "r");
    fscanf (fp, "%d", &item);

printf ("%d", item);
fprintf (stdout, "%d", item);    /* as above   */

    fptr = fopen ("tally.out", "a");
    fprintf (fptr, "%d", item);  /* append to tally.out *
```

Character at a Time I/O

- There are three functions for doing single character I/O:
  ```
  int getc (FILE* fp);
  void putc (char c, FILE* fp);
  void ungetc (char c, FILE* fp);
  ```
- The getc() procedure reads a single character from the stream pointed to by the parameter. The value returned is the character read, or the special value EOF if the end-of-file is encountered.
- The putc() procedure writes the character c, onto the stream specified by the parameter fp
- ungetc() puts the last character read by getc() back on the input stream. Necessary to undo the last char. read.

- The special functions getchar() and putchar() are perhaps best forgotten from now on.

Memory I/O

- One can also do input output to an array or other region in memory. The relevant prototypes are:
  ```
  int sscanf (char* buffer, char* format, *arg1, *arg2, ..)
  char* sprintf (char* buffer, char* format, arg1, arg2, ..
  ```
- sscanf() and sprintf() allow you to read and write to a memory buffer, just as if it were a file.
- That is, to use a pointer to an array or memory, just as if it were a pointer to a file.

Line at a time I/O

- There are two functions for moving a whole line of data from/to a file:
  ```
  char* fgets (char* buffer, int BLength, FILE* fp);
  void fputs (char* buffer, FILE* fp);
  ```
- These functions require care in their usage.
  fgets() copies all the characters until a '\n' into the array pointed to by buffer, replacing the '\n' by '\0'. To protect against overflow, only BLength-1 characters are moved. Fgets() return NULL when an EOF is seen.
  fputs() copies the contents of a null-terminated string to the output file, replacing the terminating '\0' with '\n' as one would hope.

Example

The fgets() and fputs() functions could be used as follows:

```
#define MAX_LENGTH 256
char buffer[MAX_LENGTH];
FILE* fp = fopen("tally.data", "r");
FILE* fptr = fopen ("tally.out", "a");

while (fgets(buffer, MAX_LENGTH, fp) != NULL)
{
        /*  read until EOF    */
        /*  process a line of the file    */
        /*  append tally.data to tally.out    */
    fputs (buffer, fptr);
}
fclose (fp);
fclose (fptr);
```

- We could also dyamically acquire memory for buffer[] as follows:

  ```
  char*  buffer;
  buffer = (char*) malloc (MAX_LENGTH * sizeof(char));
  ```
- Instead of close() we would use:
  ```
  free(buffer);       /* to give back the space  */
  ```
- The companion special functions gets() and puts() that read/write stdin and stdout are best forgotten.

Prototypes of Other Procedures

- `void rewind (FILE* fp);`
- `int fseek (FILE* fp, int offset, int kind);`
- `int ftell(FILE* fp);`

You are not likely to need them in C201, but you now know enough to read about them when you do.