Practical Programming Methodology
(CMPUT-201)

Michael Buro

Lecture 12

- C I/O continued

FILE Functions (3)

- `size_t fwrite(void *ptr, size_t size, size_t n, FILE *fp);`
  writes `size*n` bytes to file `*fp` starting at address `ptr`;
  `T a[N];... fwrite(a, sizeof(*a), N, fp);`
- `size_t fread(void *ptr, size_t size, size_t n, FILE *fp);`
  reads `size*n` bytes from file `*fp` and stores them at `ptr`
  `T a[N];... fread(a, sizeof(*a), N, fp);`
- `fwrite`/`fread` return number of successfully written/read items, use `feof` and `ferror` to distinguish end-of-file and read errors
- `void fflush(FILE *fp);`
  forces a write of all buffered data to device/file

Formatted Output

```
typedef const char *ccptr;

int printf(ccptr format, ...);
  = fprintf(stdout, format, ...);

int fprintf(FILE *fp, ccptr format, ...);
```

- formatted data output
- variable # of parameters to be printed, must match format string. Modern compilers check that.
- e.g. `printf("%d %d %f\n", i, j, real);` prints two integers and a double value in readable form to stdout

Format String

- %c : character
- %s : C-string
- %d : integer number
- %f : double precision floating point number
- %e : -"- , scientific notation
- ... many more: man fprintf
- %% = %
- general:
  % [flags] [width] [prec] [len-mod] conv-spec

```
#include <cstdio>

char c = 'x';
int i = 12345;
double f = 3.1415926535;
char s[] = "foo";

printf("%% c=%c i=%d f=%f s=%s", c, i, f, s);
// "% c=x i=12345 f=3.141593 s=foo"

printf("|%d TEST|", i);          // |12345 TEST|
printf("|%8dTEST|", i);          // |   12345TEST|
printf("|%08dTEST|", i);         // |00012345TEST|
printf("|%-8dTEST|", i);         // |12345   TEST|

printf("|%f TEST|", f);          // |3.141593 TEST|
printf("|%.1f TEST|", f);        // |3.1 TEST|
printf("|%7.2f TEST|", f);       // |   3.14  TEST|
printf("|%+13.8f TEST|", f);     // |  +3.14159265 TEST|

printf("|%.3e|", f);             // |3.142e+00|
```

Formatted Input

```
int scanf(ccptr format, ...);
  = fscanf(stdin, format, ...);

int fscanf(FILE *fp, ccptr format, ...);
```

- formatted data input
- variable number of pointers to variables to be read, must match format string
- returns number of successfully read values
- `fscanf(fp, "%d %d %f", &i, &j, &real);`
  reads two integers and a double value and returns 3 if OK
- DANGEROUS! Hopefully the compiler reports type errors

Input Examples

```
#include <cstdio>

int a, b, c;

if (scanf("%d %d %d", &a, &b, &c) != 3) {
  // less than 3 values read from stdin => error
}

int c = fgetc(stdin); // read one byte
if (c == EOF) {       // end of file reached or error
  if (feof(stdin))    // end of file
  else                // error

char buffer[N];
// read until EOF, '\n', or max #bytes-1 is reached
int r = fgets(buffer, N, stdin);
if (r == 0) // nothing read or error
```