File Access (7.5)

CSE 2031
Fall 2010

Declaring and Opening Files

```
FILE *fp; /* file pointer */
FILE *fopen(char *name, char *mode);
```

Example:

```
FILE *ifp, *ofp;
char iname[50], oname[50];
ifp = fopen( iname, "r" );
if ( ifp == NULL ) { ... }
ofp = fopen( oname, "w" );
```

Modes

```
fp = fopen( name, "r" );
```

- Returns NULL if file does not exist, or has no read permission.

```
fp = fopen( name, “w" );
```

- If file does not exist, one will be created for writing.
- If file already exists, the content will be erased when the file is opened. So be careful!
- Returns NULL if file has no write permission.

3

Modes (cont.)

```
fp = fopen( name, “a" );   /* append */
```

- If file does not exist, one will be created for writing.
- If file already exists, the content will be preserved.
- Returns NULL if file has no write permission.

- May combine multiple modes.

```
fp = fopen( name, "rw" );
```

File may be read first, but the old content will be erased as soon as something is written to the file.

```
fp = fopen( name, "ra" );
fp = fopen( name, “aw" );  /* same as “a” */
```

4

Reading and Writing Files

```
int getc( FILE *fp )
int putc( int c, FILE *fp )
int fscanf( FILE *fp, char *format, ... )
int fprintf( FILE *fp, char *format, ... )

int c;
while ( (c = getc( ifp )) != EOF )
   putc( c, ofp );

char ch;
while ( fscanf(  ifp, "%c", &ch ) != EOF )
   fprintf(  ofp, "%c", ch );
```

5

Closing Files

```
int fclose( FILE *fp )

fclose( ifp );
fclose( ofp );
```

- Most operating systems have some limit on the number of files that a program may have open simultaneously ⇒ free the file pointers when they are no longer needed.
- **`fclose`** is called automatically for each open file when a program terminates normally.
- For output files: **`fclose`** flushes the buffer in which **`putc`** is collecting output.

6

Reminder: I/O Redirection

- In many cases, I/O redirection is simpler than using file pointers.

```
a.out < input_file > outout_file

a.out < input_file >> outout_file
```

7

Review: `printf()` and `scanf()`

8

printf( ) (7.2)

```
int printf(char *format, arg1, arg2, ...);
```

- converts, formats, and prints its arguments on the standard output under control of the **format**.
- returns the number of characters printed (usually we are not interested in the returned value).

9

printf( ) Examples

```
printf(":%s:",        "hello, world");
printf(":%10s:",      "hello, world");
printf(":%.10s:",     "hello, world");
printf(":%-10s:",     "hello, world");
printf(":%.15s:",     "hello, world");
printf(":%-15s:",     "hello, world");
printf(":%15.10s:",   "hello, world");
printf(":%-15.10s:",  "hello, world");
```

```
:%s:          :hello, world:
:%10s:        :hello, world:
:%.10s:       :hello, wor:
:%-10s:       :hello, world:
:%.15s:       :hello, world:
:%-15s:       :hello, world   :
:%15.10s:     :     hello, wor:
:%-15.10s:    :hello, wor     :
```

10

printf Conversions

Character	Argument type; Printed As
d, i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x, X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
e, E	double; [-]*m.ddddddE+/-xx* or [-]*m.ddddddE+/-xx*, where the number of *d*'s is given by the precision (default 6).
g, G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

11

Output Formatting with printf( )

- A **minus sign**, which specifies **left adjustment** of the converted argument.
- A number that specifies the **minimum field width**. The converted argument will be printed in **a field at least this wide**. If necessary it will be padded on the left (or right, if left adjustment is called for) to make up the field width.
- A **period**, which **separates the field width from the precision**.
- A number, the **precision**, that specifies the **maximum number of characters to be printed from a string**, or the **number of digits after the decimal point** of a floating-point value, or the **minimum number of digits for an integer**.

12

scanf Conversions (7.4)

Character	Input Data; Argument type
d	decimal integer; `int *`
i	integer; `int *`. The integer may be in octal (leading `0`) or hexadecimal (leading `0x` or `0X`).
o	octal integer (with or without leading zero); `int *`
u	unsigned decimal integer; `unsigned int *`
x	hexadecimal integer (with or without leading `0x` or `0X`); `int *`
c	characters; `char *`. The next input characters (default 1) are placed at the indicated spot. The normal skip-over white space is suppressed; to read the next non-white space character, use `%1s`
s	character string (not quoted); `char *`, pointing to an array of characters long enough for the string and a terminating `'\0'` that will be added.
e,f,g	floating-point number with optional sign, optional decimal point and optional exponent; `float *`
%	literal %; no assignment is made.

13

scanf( ) Examples

```
int num;
scanf("%d", &num);

char c; float f;
scanf("%c %f", &c,
  &f);

long number;
scanf("%ld", &number)
```

```
double dnum;
scanf("%lf", &dnum)

char strg[100];
scanf("%s", strg);
```

Notes:

- no **&** in front of variable (why?)
- **'\0'** is added automatically to end of string.

14

scanf( )

```
int scanf(char *format, arg1, arg2, ...);
```

- reads characters from the standard input, interprets them according to the specification in **`format`**, and stores the results through the remaining arguments.
- stops when it exhausts its format string, or when some input fails to match the control specification.
- returns the number of successfully matched and assigned input items (e.g., to decide how many items were found).
- returns 0 if the next input character does not match the first specification in the format string (i.e., an error).
- On the end of file, EOF is returned.
- Note: **`arg1, arg2, ...`** must be pointers!

15

Final Exam

- December 15, 9am – 12pm.
- Location: check Registrar's office web site.
- Written exam only.

16