

File IO  
C Preprocessor  
The Debugger  
Midterm Review

CSE142  
Doug Treder

# File IO Review

## -Writing to a File

---

```
FILE *fwrite ; int status;
```

```
fwrite = fopen ("name.ext", "w") ; /*  
write */
```

---

```
fprintf (fwrite, "%...", var, ... ) ;
```

---

```
fclose (fwrite);
```

---

# File IO Review

## Reading from a file

---

```
FILE *fthread; int status;
```

```
fthread = fopen ("name.ext", "r"); /*  
read */
```

```
status = fscanf (fthread, "%...",  
&var, ... );
```

```
/* fscanf returns EOF on end of  
file */
```

```
fclose (fthread);
```

# Rules of scanf and fscanf

- See pages 357-358 in C: A Reference Manual

```
scanf(" %c%d", &c &i);
```

control string

An arrow points from the text 'control string' to the first argument of the scanf function call, which is the format string " %c%d".

output parameters

An arrow points from the text 'output parameters' to the second argument of the scanf function call, which is the address of variable 'i' (&i).

# Rules of scanf and fscanf

- Control String
  - if there is a whitespace in the control string, all whitespace in the input is discarded
  - A conversion spec is a % followed by one or two letters
  - Any other characters in control string must match the input exactly for anything to match
  - "cat %d"
    - matches "cat 4"
    - doesn't match "cat a"
    - doesn't match "ca 4"

# Rules of scanf and fscanf

- More control string examples:
  - `scanf("%c", &onechar);`
    - "a b" – onechar gets 'a'
    - " a b" – onechar gets ' '
  - `scanf(" %c", &onechar);`
    - "a b" – onechar gets 'a'
    - " a b" – onechar gets 'a'
  - `scanf("% c", &onechar); // error!`
    - "a b" – onechar gets nothing
    - " a b" – onechar gets nothing
  - Whitespace is spaces, tabs, and newlines!!!

# Rules of scanf and fscanf

- Type and number of output parameters must match the control string!
- Put '&' in front of all variables in output parameters.

```
int i, j  
scanf("%d %d", &i, &j);
```

# Reading a single character

- Sometimes you just want to check the next character
- For example, you found an error and want to skip all characters to the end of the line!

# Reading a single character

```
char c;  
FILE* fh = fopen("file", "r");  
  
c = fgetc(fh);  
  
// skip chars until we get to end  
// of file, or end of line  
while (c != EOF && c != '\n') {  
    c = fgetc(fh);  
}
```

# fgetc has undo!

```
char c;
FILE* fh = fopen("file", "r");

c = fgetc(fh);

// looking for 'b'
while (c != EOF && c != 'b') {
    c = fgetc(fh);
}

// since we read the 'b', put it back
ungetc(c, fh);
```

# The C Preprocessor

- Three steps to creating an executable:
  - Preprocessing
  - Compiling
  - Linking
- The preprocessor manipulates the text of your source code **BEFORE** the compiler runs!

# The C Preprocessor

- Heavy use of preprocessing is deprecated in modern C and C++ programming
  - Use `const` instead of `#define`
  - Use functions instead of `#define` macros
- But you will find lots of code using it, so I'll teach it.

# #include

- Copies in the requested file
- On most compilers there are two paths for #include to find files:
  - The system include paths are for all the header files that came with your system or compiler
    - #include <stdio.h>
  - The user include paths are for header files you've created yourself
    - #include "project1.h"

# #define

- Can create “constants”
- Can also create macros
  - #define HYPOT(a,b) sqrt(a \*a + b \* b)
  - What’s the problem with this?

# #define

- `#define HYPOT(a,b) sqrt(a *a + b * b)`
- Try this code:

```
while (i < 100 && j < 100) {  
    printf("Hypotenuse of %d and %d is %lf",  
        i, j, HYPOT(i++, j++));  
}
```

# #ifdef

- Conditional compilation
- You can test whether a macro or constant has been #define'd
- Requires an #endif when finished

```
#define LINUX
#ifdef LINUX
    //do some linux type stuff
#else
    // do windows type stuff
#endif
```

# #ifdef

- Where does this get used most? Header files!

```
#ifndef PROJECT1_HEADER_FILE
#define PROJECT1_HEADER_FILE

// entire header file

#endif
```

# The Debugger

- Debuggers let you
  - step through code
  - watch variables
  - catch errors before the program crashes
  - show a trace of function calls (aka 'stack trace')
- Every compiler has a different debugger

# Debugging

- Start debugging either by "stepping into" the first line
- Or by setting a breakpoint and running the program
- Breakpoints are lines of code where the debugger stops the program

# Midterm Review