CSA2090: Systems Programming

Introduction to C

Lecture 7: Input and Output

Dr. Christopher Staff

Department of Computer Science & AI

University of Malta

Aims and Objectives

- Input and Output
- Source File Organisation

© 2004- Chris Staff

File I/O under UNIX

- Whenever you do file i/o, C interacts with the operating system to satisfy your request
- Files in C are *streams* of bits or bytes
- You can think of interactions being *buffered* through special areas in memory

© 2004- Chris Staff

File Pointers and Descriptors

- A *file pointer* is a special data type for files
  - supports opening and closing streams
  - reading and writing streams (when legal)
- *File descriptors* are integers that point into a table of information about opened files
- see file.c

Input and Output Redirection

- UNIX allows input and output to be redirected using < and >
- E.g., cat .cshrc > myconfig
- E.g., cat < myconfig
- *Filters* can be written in C that read from stdin and write to stdout
- See line_nums.c

Interactive Output

- Writing to files is usually buffered
- The file is only written to when the buffer is flushed
  - When a \n is encountered
  - When the buffer is filled
- If your program crashes *before* a buffer is flushed, then...

Interactive Output

- To force output, use
  - `stderr` instead of `stdout`
  - `fflush(stdout)`
  - `setbuf(stdout, NULL)`
    - `setbuf(stdout, !NULL)` to resume buffering

Interactive Input

- scanf and gets are unsafe, because you cannot check the size of input before you receive it
- sscanf and fgets are safer, but now fgets will leave newline characters in the input string

Source File organisation

- Preprocessor facilities
- Multiple source files
- Make

© 2004- Chris Staff

C Preprocessor

- Any directive starting with #
- Source file inclusion
  - #include
- Macro replacement
  - #define
    - Symbolic constants: #define TRUE 1
    - Macro: #define MAX(x,y) ((x) > (y) ? (x) : (y))

C Preprocessor

- Conditional inclusion
  - – #ifdef, #else, #endif
  - – #if defined(DEBUG)... #endif
  - – #if 0... #endif (useful for “switching off” code, instead of commenting it out)

Multiple Source Files

- Keeps programs modular
- Allows multiple programmers to work simultaneously on same program
- Allows library files to be written
- Can separate out parts of code that are platform dependent (e.g., i/o routines)

Multiple Source Files

- Functions can be kept in separate source files
- However, function prototypes, global variables, symbolic constants, etc., needed by multiple source files will be defined in a header file
- The header file will be #included into each source file using it
- See multi.h, multi1.c, multi2.c

© 2004- Chris Staff

Make

- With all these source files knocking all over the place, it’s easy to lose track of file dependencies
- And we also don’t need to recompile everything, every time, unless individual components have changed
- Enter Make

Make

- A makefile describes how executable files are obtained

```
pgm: a.o b.o #target:dependencies
   cc -Aa a.o b.o -o pgm #tab!
a.o: incl.h a.c
   cc -Aa -c a.c
b.o: incl.h b.c
   cc -A -c b.c
```

Make

- Typing make in a directory containing one makefile will execute it from the beginning
- You can run from any part by typing make target, e.g., make a.o
- See Love 15.3 for another example...

© 2004- Chris Staff

cstaff@cs.um.edu.mt

Conclusion

- That's it!