CS 537
Section 1
Programming in Unix and C

Michael Swift

© 2004-2009 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Project 0

- Word count histogram
  - You will write a program that reads a text file and reports the total count of words of each length
  - Program input: read a file specified on the command line: `word-count filename.txt`
  - The output should be as follows:

```
length 3: 2 words
length 44: 44 words
```

- Debugging
  - Start your program under the debugger.
  - Single step through input of a single line.
  - Print out the values of at least two different variables.

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Facilities

- Department Linux machines (penguins):
  - 1350: mumble-##
  - 1351: king##
  - 1370: adelie##, humboldt##, macaroni##
- Unix Orientation classes
  - Today, Wednesday at 4 pm in CS 1325
- CS1000
  - http://www.cs.wisc.edu/csl/cs1000/

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Why C

- All modern operating systems are written in C
- Why?
  - Control
  - Predictable code
  - Expressive
  - Optimizable
  - Powerful pre-processor

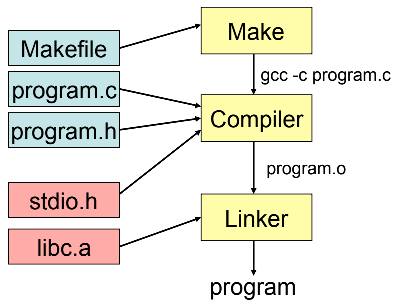
© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Issues with C

- Little hand-holding for programmer
  - Manual memory management
  - Small standard library
  - No native support for threads and concurrency
  - Weak type checking

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Using C and Unix



© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

C language

```
#include <stdio.h>
int main(int argc, char * argv[])
{
  printf("Hello, world\n");
  return(0);
}
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Issues with C

- Memory allocation
  - malloc(), free()
- Pointer arithmetic and arrays
- Preprocessor

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Example

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Memory

- You have to mange memory yourself.
- Stack allocated memory: becomes invalid when you return from function. This will not work:

```
char * f() {
  char str[100];
  strpy(str,"hello, world\n");
  return(str);
}
```

- Memory from malloc only becomes invalid when you free it:

```
char * f() {
  char *str;
  str = malloc(100);
  strpy(str,"hello, world\n");
  return(str);
}
```

- is o.k., but someone has to call free(str);

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Strings

- Strings in C are arrays of bytes:
  - char str[100];
- Or pointers to memory
  - char * str;
  - str = malloc(100);
- They are null terminated – so you need to make space for it
  - str[0] = '\0';
  - strlen(str) = 0;
- There are a bunch of functions for working with them:
  - strlen, strcpy, strcat

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

File I/O

- f* functions for accessing files:
  - struct FILE *: represents an open file
  - f = fopen("foo","r") – open file foo for reading
  - fclose(f) - says you are done with f
  - bytes = fread(buffer,size,,count,f) = reads size x count bytes from f into buffer
  - fwrite(buffer, size, count ,f) = writes size x count bytes to f from buffer
  - fgets(str, size, f) = reads up to size-1 bytes from a single line of f into str, including newline

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

More advanced topics

- Compiler errors and warnings
  - gcc -Wall foo.c
- Optimization for faster and smaller code
  - gcc -O foo.c
  - gcc -O2 foo.c
- Separate compilation
  - gcc -c foo.c
  - gcc -c bar.c
  - gcc -o foobar foo.o bar.o

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Documentation

- Unix/Linux man pages
  - example: "man close"

```
CLOSE(3)              BSD Library Functions Manual                FCLOSE(3)

NAME
     fclose -- close a stream

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     int
     fclose(FILE *stream);

DESCRIPTION
     The fclose() function dissociates the named stream from its underlying
     …

RETURN VALUES
     Upon successful completion 0 is returned.  Otherwise, EOF is returned and
     …
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Man pages

- Documentation is divided into sections
  1. Programs, commands
  2. System calls
  3. Subroutine libraries
  4. Hardware
  5. Config files
  6. Games
  7. Miscellaneous
  8. System administration
- man returns the result from the lowest-numbered section
- apropos searches for commands with a word

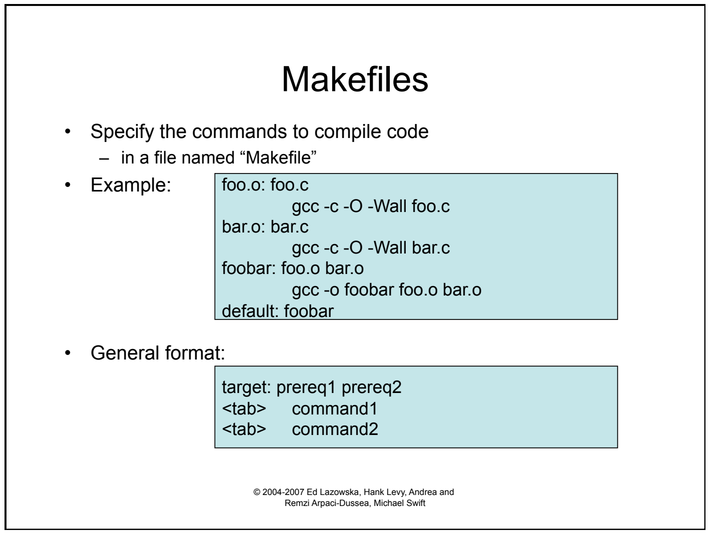
© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Debugging

- Compile with debugging using "-g"
  - gcc -g -o foo.o foo.c
- Run your program with gdb

```
gdb foobar
GNU gdb 6.3
<copyright omitted>
(gdb) break main
breakpoint 1 at 0x80483b0: in file foo.c, line 5
(gdb) run
Starting program: /afs/cs.wisc.edu/…/foobar
Breakpoint 1, main (argc=1, argv=0xbfe27804) at foo.c:5
5      if (argc > 1) {
(gdb) print argc
$1 = 1
(gdb)
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift



Makefiles

- Specify the commands to compile code
  - in a file named “Makefile”
- Example:

```
foo.o: foo.c
	gcc -c -O -Wall foo.c
bar.o: bar.c
	gcc -c -O -Wall bar.c
foobar: foo.o bar.o
	gcc -o foobar foo.o bar.o
default: foobar
```

- General format:

```
target: prereq1 prereq2
<tab>     command1
<tab>     command2
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dussea, Michael Swift