Michael Swift

Sections

- Ask questions about lecture material
- Catch-up on the schedule
- Quizzes
- Project discussion
- C programming

Project 0

- Sort a file
  - You will write a program that reads variable-length records with a key and a value
  - **Files can be of any length, with records up to 1024 bytes**
  - Program input: read a file specified on the command line: `vsort –i input.txt –o output.txt`
- Competition
  - The fastest sort program in the class wins a prize (a water bottle)

Facilities

- Department Linux machines (penguins):
  - 1350: mumble-##
  - 1351: king##
  - 1370: adelie##
- Unix Orientation classes
  - This week

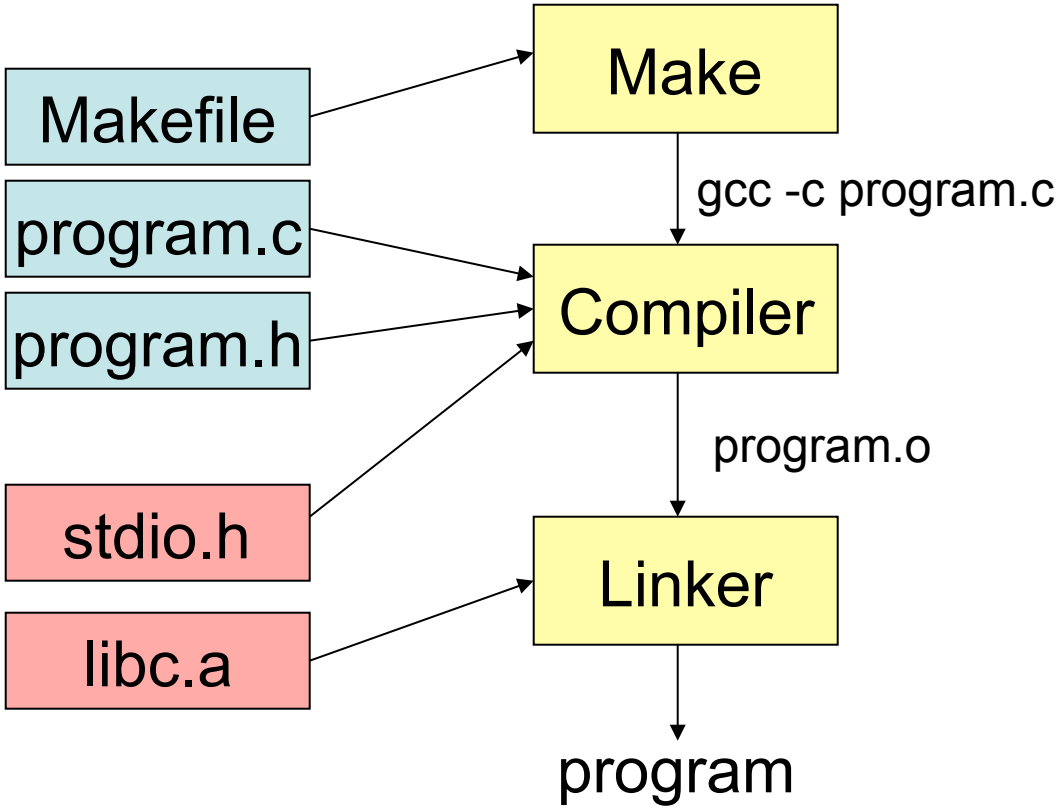
Why C

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

Issues with C

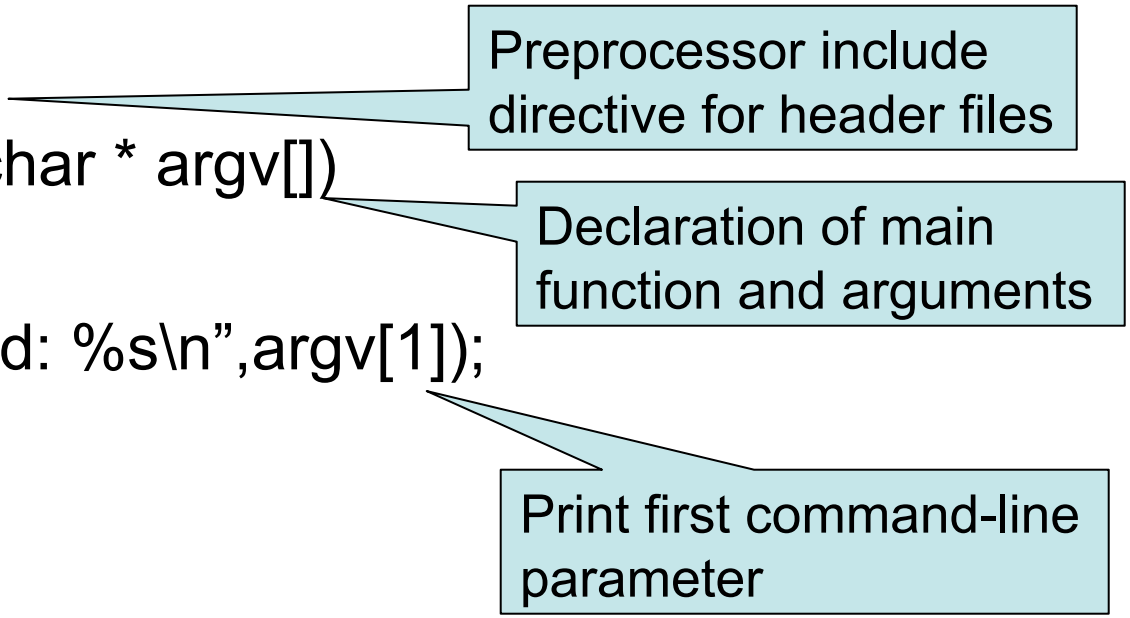
- Little hand-holding for programmer
  - Manual memory management
  - Small standard library
  - No native support for threads and concurrency
  - Weak type checking

Using C and Unix



C language

```
#include <stdio.h>
int main(int argc, char * argv[])
{
  printf("Hello, world: %s\n",argv[1]);
  return(0);
}
```



Issues with C

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

Example

```
#include <stdio.h>

int main(int argc, char * argv[])
{
  int i;
  for (i = 0; i < argc; i++) {
    printf("argv[%d] = %s\n",i, argv[i]);
  }
  return(0);
}
```

Strings

- Strings in C are arrays of bytes:
  - char str[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

String Example

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
  char s[100];
  strcpy(s,"hello");
  strcat(s,", world");
  printf("S = %s\n",s);
}
```

Memory

- You have to mange memory yourself.
- Fixed-size variables can be allocated on a stack
  - The contents of these variables go way when the function returns.

    ```
    char str[100] = “hello, world\n”;
    ```

- Variable-size variables are allocated using **malloc**, like new() in Java.
  - Memory from malloc only becomes invalid when you free it:

    ```
    char *str;
    str = malloc(n);
    strpy(str,”hello, world\n”);
    free(str);
    ```

Memory Example 1

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char   * argv[]) {
  int * array = NULL;
  int i;
  array = (int *) malloc(100 * sizeof(int));
  if (array == NULL) {
    printf("Malloc failed\n");
    return(0);
  }
  for (i = 0; i< 100; i++)
    array[i] = 100-i;
  for (i = 0; i< 100; i++)
    printf("array[%d] = %d\n",i,array[i]);
  free(array);
  return(0);
}
```

Memory Example 2

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  char * s;
  s = malloc(100);
  if (s == NULL) {
    printf("Malloc failed\n");
    return(0);
  }
  strcpy(s,"hello");
  strcat(s,", world");
  printf("S = %s\n",s);
  free(s);
}
```

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

File Example

```
#include <stdio.h>
int main(int argc, char * argv[]) {
    FILE * f;
    char s[101];
    f = fopen("test.txt","r");
    if (f == NULL) {
      printf("Error opening file\n");
      return(0);
    }
    while (fgets(s, 100, f) != NULL)
            printf("%s",s);
   fclose(f);
   return(0);
}
```

Data Structures in C

- Global arrays
  - int global_data[500];
- Dynamic arrays are pointers
  - int * dyn_array;
  - dyn_array = malloc(sizeof(int) * 100);
- Structures
  - typedef struct __rec_t {
  - int key;
  - int size;
  - int record[NUMRECS];
  - } rec_t;
  - rec_t r;
  - r.key = 10; r.size = 10; r.record[0] = 0; …

More advanced topics

- Compiler errors and warnings
  - gcc –o foo -Wall foo.c
- Multiple files
  - gcc –o foo foo.c bar.c baz.c

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
     …
```

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

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)
```