7 Lecture: Starting in with unix/Unbuffered vs. Buffered IO

Outline:

- Announcements
- From last time: Compound Data Wrapup
  - So Far
  - Type definitions
  - Unions and enumerated types
- Pointer review: It's all the same
  - Pointers to Functions: Ridiculous
  - Summing up
- C odds and ends
  - Short-circuit evaluation
  - The Ternary Operator
  - The Comma Operator
  - Variable modifiers
  - Promotion rules
- Change of plan for today
- Notes on Asgn2
- Separate Compilation and the C Preprocessor
  - The C-Preprocessor
- Separate compilation
- Final Observation
- Tools of last Week: grep and truss
- Unix Overview
- Identity Issues: logging in
  - Looking at system files
- From Last, Last, Last Time: Unix Overview
- Files and Directories
  - Directories
  - Directory Manipulation
- System Calls
- IO Services
  - Unbuffered IO: unix
  - Buffered IO: stdio (C)
- Programs and Processes
  - Programs
  - Processes
  - Flavors of ps
  - Job Control
- Interprocess Communication: Signals

7.1 Announcements

- Coming attractions:

Event	Subject	Due Date	Notes

Use your own discretion with respect to timing/due dates.

- Comment on class leading the assignments
- C quiz next Wednesday.
  - Old exams on the web. (think 202-like material)
  - Bring questions for review Monday
- For Next Time: Know, love, and become one with Stevens Ch. 1.
- Nul-terminate your damned strings
- C quiz samples out
- Dealing with binary data. `cat -v` or `od -tx1` or others
- `time(1)` `/usr/bin/time` tells you how long a program runs: user, system, and wallclock.
- Only one late day allowed for asgn2.
- Only turn in one...
- asgn1

```
% tallac% grep Passed */COMMENT | grep out | sed 's/.*://' | sort | uniq -c | sort -rn -k 3
     72 Passed 13 out of 13 tests.
      4 Passed 12 out of 13 tests.
      1 Passed 8 out of 13 tests.
      1 Passed 1 out of 13 tests.
tallac%
```

7.2 From last time: Compound Data Wrapup

7.2.1 So Far

- Arrays
- Structs

7.2.2 Type definitions

7.2.3 Unions and enumerated types

There are two more types we haven't discussed: unions and enums

An enumerated (**enum**) type allows you to use symbolic names for values:

```
typedef enum {mon,tue,wed,thu,fri,sat,sun} day;
day tomorrow = fri;
```

or

```
typedef enum {false, true} boolean;
boolean b = false;
```

A **union** is a like a structure except all of the fields occupy the same space.

```
union kludge{
  int num;
  char bytes[sizeof(int)];
};
int i;
union kludge k;
k.num = 0x0A0B0C0D;
for(i=0; i<sizeof(k.num); i++)
   printf("Byte[%d]: 0x%02x\n", i, k.byte[i]);
putchar('\n');
```

7.3 Pointer review: It's all the same

Pointers are memory addresses.

- Pointers to simple data
- Pointers to compound data
- Pointers to functions? See below

7.3.1 Pointers to Functions: Ridiculous

Don't even worry about knowing about these at this point. It's bad enough to know they exist:

Declaration: int (*fname)(int);

Asignment: (assiming foo() exists) fname = foo;

Use: x = (*fname)(2)

7.3.2 Summing up

Things you know:

- Dynamic data structures are no more complicated than allocating space and remembering where you put things.
- A pointer is a variable that contains a memory address.
- Pointers are uninitialized; they do not necessarily point at anything in particular.
- Operators:

&	address of
*	dereference

- NULL is a standard invalid pointer.

- Think *types.* Pointer vs. pointee.

  Given an integer pointer, p stored at address 0x12345678 and an integer i, stored at 0xABCD-ABCD, containing the value 3:

```
int i=3;
int *p=&i
```

Expression:	&p	p	*p
Type:	int **	int *	int
Value:	0x1234	0xABCD	3

	Expression					
Variable	**&x**	**x**	***x**	****x**	*****x**	******x**
int x	int *	int	—	—	—	—
int *x	int **	int *	int	—	—	—
int **x	int ***	int **	int *	int	—	—
int ***x	int ****	int ***	int **	int *	int	—

- `sizeof()` is a macro that evaluates to the size of a given data structure.

- The name of an array is equivalent to the address of that array. Even though it is casually called a pointer, it is not a variable.

- The same is true of the name of a function.

- Free not that which thou didst not malloc().

7.4 C odds and ends

7.4.1 Short-circuit evaluation

C conditionals are evaluated left to right and evaluation stops as soon as the overall sense of the expression is known.

Thus, the following is safe:

```
if ( p && *p ) ...
```

7.4.2 The Ternary Operator

Useful in some situations, but never at the cost of clarity. Good usage:

```
printf("Found %d word%s.\n", num, (num==1)?"":"s");
```

7.4.3 The Comma Operator

The operands of *comma* are evaluated left to right, and the value of the overall expression is the value of the rightmost operand. (i.e., the value of x, y is y)

For use, see Figure 35.

```
void Pof2(int n) {
    /* print the first n powers of two */
    unsigned long i, num;
    for(i=0, num=1; i < n; i++, num*=2)
        printf("2^%-2d = %u\n", i, num);
}
```

Figure 35: Pof2(): Printing the first n powers of 2.

7.4.4 Variable modifiers

unsigned
const
extern
static
volatile
register
restrict

7.4.5 Promotion rules

C data types will automatically be promoted to "wider" types. Note that while magnitude is preserved, precision may not be. (e.g., `int` to `float`)

Promotion is only done as needed. Consider

```
f = 100.0 * 3/4;
g = 3/4 * 100.0;
printf("f = %f\ng = %f\n", f, g);

f = 75.000000
g = 0.000000
```

7.5 Change of plan for today

- C Preprocessor
- Predefined macros and conditional compilation
- Macros with paramters
- Uniq
- Unix to Friday

7.6 Notes on Asgn2

- All code here must be your own.
- Design! This program decomposes fairly well.
- Reading long strings and `realloc(3)`
- My asgn2 is 569 lines total. (this is not a large program)
- `time(1)`
- `-O` turns on the optimizer
- Go ahead and look up a hash function, just don't appropriate someone's code. (cite it!)
- "Notes on Partnerships" — why

See example in Figure 36.

```
lagniappe% wc /usr/share/dict/words
 99171  99171 938848 /usr/share/dict/words
lagniappe%
falcon% time fw.solaris /usr/dict/words /usr/share/man/*/*
The top 10 words (out of 69900) are:
   452919 fr
   275401 the
   201833 sp
   156752 in
    88201 to
    87961 fb
    81270 pp
    78950 n
    76514 is
    75718 of
6.61u 0.53s 0:12.01 59.4%
falcon%
```

Figure 36: fw in action

`/usr/man` on unixX is approx. 110MB
`/usr/share/dict/words` is 4.8MB and contains 480k words.

7.7 Separate Compilation and the C Preprocessor

7.7.1 The C-Preprocessor

- `#include "file.h"`
- `#include <file.h>`
- `#ifdef ...#endif`
- `#ifndef ...#endif`
- null-macros: `#define BLAH`
- simple-macros: `#define BLAH OTHERTHING`

  Some of these are predifined and very useful. For example

  gcc `__FUNCTION__`, `__FILE__`, `__LINE__`, `__GNUC__`,

  c++ `__cplusplus`

  architecture-dependent Things like `__sun`, `sparc`, `unix`, etc.
  See Figure 37.

- real-macros: `#define sqr(c) ((c)*(c))`

  be very careful with parenthesization and side-effects

```
falcon% touch foo.c
falcon% gcc -dM -E foo.c
#define __GCC_NEW_VARARGS__ 1
#define __sparc 1
#define __svr4__ 1
#define __GNUC_MINOR__ 95
#define __sun 1
#define sparc 1
#define __sun__ 1
#define __unix 1
#define __unix__ 1
#define __SVR4 1
#define sun 1
#define __GNUC__ 2
#define __sparc__ 1
#define unix 1
falcon%
```

(a) SunOS on a Sparc

```
pnico% touch foo.c
pnico% gcc -E -dM foo.c
#define __USER_LABEL_PREFIX__
#define __SIZE_TYPE__ unsigned int
#define __PTRDIFF_TYPE__ int
#define __HAVE_BUILTIN_SETJMP__ 1
#define __i386 1
#define __GNUC_PATCHLEVEL__ 0
#define __ELF__ 1
#define __WCHAR_TYPE__ long int
#define __GNUC_MINOR__ 96
#define __WINT_TYPE__ unsigned int
#define __tune_i386__ 1
#define __unix 1
#define unix 1
#define __REGISTER_PREFIX__
#define __linux 1
#define __GNUC__ 2
#define i386 1
#define __linux__ 1
#define __VERSION__ "2.96 20000731 (Red Hat Linux 7.0)"
#define __i386__ 1
#define linux 1
#define __unix__ 1
pnico%
```

(b) Linux on an x86

Figure 37: Pre-defined macros on different platforms

7.8 Separate compilation

(extern data)

- Compilation
- linkage
- libraries
- Names

  extern name is known, but not defined here.

  static name is defined here and only here. (file scope)

  static allocated in the data segment, not the stack. (Link it now.)

7.9 Final Observation

This has been a whirlwind presentation of a complex language. I can't teach you C. Practice is required.

7.10 Tools of last Week: grep and truss

grep locate strings in a file. from ed: g/re/p

Example:

```
% grep -il "STDIN_FILENO" /usr/include/*.h
/usr/include/unistd.h
%  grep -il
```

strace (linux), truss (solaris) watch system calls go by: These programs show you all system calls made along with their arguments and

7.11 Unix Overview

When discussing an operating system, everything depends on everything else. Today we want to talk about a process's view of its world in terms of various services an OS provides.

A note on the examples: They use code contained in Appendix B.

`http://www.kohala.com/start/apue.html`

A process's view of the world:

- Identity (uid, gid)
- Filesystem (storage, organization, ownership, access)
- IO (What good is a filesystem if you can't use it)
- Programs and Processes
- Interprocess Communication: Signals

7.12 Identity Issues: logging in

Before doing anything else on a unix machine, you need to work out who you are. (User and group identities)

Identity consists of uid and gid

Identity is controlled by several system files:

- `/etc/passwd` — this maps login name to uid (and stores passwd)
- `/etc/group` — primary and supplemental groups
- shadow passwds: `/etc/shadow`
- yellowpages (or other directory schemes, LDAP, etc.): shared info across machines

A password line:

```
csc357:*:1659:350:csc357:/home/cscstd/abcd/csc357:/bin/tcsh
```

This is "login:passwd:uid:gid:comment:home dir:shell"

How the password algorithm works: Your typed password and the salt (the first two characters) are combined via a cryptograhic hash to produce 13 printable characters in "[a–zA–Z0–9./]".

```
#include <unistd.h>

char *crypt(const char *key, const char *salt);
```

Traditionally a variation on DES, now likely to be something else..

The salt perturbs the dictionary 4096 different ways.

Your uid is your identity. The password file is the only place your login name appears in the system.

This exposure makes them vulnerable.

7.12.1 Looking at system files

real vs. shadow or yp

shadow/yp on hornet and drseuss

7.13 Files and Directories

The entire system is connected through the filesystem. (discussion of limits.h)

```
NAME_MAX:
Solaris: 512
linux: 256
```

7.13.1 Directories

A directory is a file containing a list of:

- name
- attributes (not actually in the directory)
- where (inode)

discribed in "dirent.h":

```
struct dirent
{
    long d_ino;                 /* inode number */
    off_t d_off;                /* offset to this dirent */
    unsigned short d_reclen;    /* length of this d_name */
    char d_name [NAME_MAX+1];   /* file name (null-terminated) */
}
```

Directory entries are unordered.
Directories are protected.

7.13.2 Directory Manipulation

Manipulation functions `opendir(3), readdir(3), closedir(3), rewinddir(3)`

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
int closedir(DIR *dir);
struct dirent *readdir(DIR *dirp);
void rewinddir(DIR *dir);
```

As part of its environment, every process has a:

Home Directory defined in /etc/passwd

Current Working Directory starts at home and follows chdir()

7.14 System Calls

Before we talk about IO services...

Look like C functions, but are direct requests for OS services.

A system call is an entry into the kernel.

Linux: (RH7.0) 222
Solaris: 253
Minix: 53

7.15 IO Services

What good is a filesystem if you can't use it?

7.15.1 Unbuffered IO: unix

IO is done in terms of file descriptors: small non-negative integers.

By convention, shells open `STDIN_FILENO`, `STDOUT_FILENO`, and `STDERR_FILENO`.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

`open(2), read(2), write(2), close(2)`

```
int open(const char *pathname, int flags,  mode_t mode);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int close(int fd);
```

`O_RDONLY, O_WRONLY or O_RDWR`

Return the number of bytes read, or -1 on error. 0 indicates there's nothing more to read (EOF).

7.15.2 Buffered IO: stdio (C)

Hides implementation details.

Works on *streams* (FILE*).

`fopen(3), fclose(3), getchar(3), putchar(3)`

String oriented: `fgets(3), fputs(3), printf(3), scanf(3)`

Defines: `stdin`, `stdout`, `stderr`.

7.16 Programs and Processes

7.16.1 Programs

A program is an executable file. (Aside on permissions(chmod)?)
Once a program is executed, it becomes a process.
`fork()` and `exec()`

7.16.2 Processes

processes share the identity of their creator:
getpid(), getpgid()
Creating one:
fork(), exec(), wait() (waitpid())
A process doesn't really terminate until it's waited for. (Init's job in this.)

7.16.3 Flavors of ps

ps -aux vs. ps -edf

Berkeley-style ps -aux

- -a all processes with a tty
- x processes without a controlling tty
- u display user-oriented format
- g everything, even "uninteresting" processes

SYSV-style ps -eadf

- -e List information about every process now running.
- -f Generate a full listing.

Also useful to know: "-u logname" list processes belonging to that login name.

7.16.4 Job Control

Starting, stopping and whatnotting jobs

7.17 Interprocess Communication: Signals

kill -l
SIGINT
SIGSTOP
SIGCONT
SIGTERM (15)
SIGKILL (9)
SIGCHLD
Possible actions:

- default
- ignore
- catch