Data structures and enums
enum name { a, b, c };
union name { … };
struct bitfield { unsigned int x : 1; unsigned int y : 8 };

Malloc, realloc, calloc
stdlib.h
void *calloc(size_t nmemb, size_t size);
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);

Gdb
- set args / show args
- display - show value at each step

Misc library functions
stdio.h
- FILE *fopen(const char *path, const char *mode);
- FILE *fdopen(int fildes, const char *mode);
- int fclose(FILE *stream);
- int feof(FILE *stream);
- int fileno(FILE *stream);

stdlib.h
- void qsort(void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *));
- void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
- int atexit(void (*function)(void));

string.h
- int strcmp(const char *s1, const char *s2);
- int strncmp(const char *s1, const char *s2, size_t n);
- char *strcat(char *dst, const char *src);
- char *strncat(char *dst, const char *src, size_t n);
- char *strtok(char *s1, const char *s2);

strings.h
- int strcasecmp(const char *s1, const char *s2);
- int strncasecmp(const char *s1, const char *s2, size_t n);

System functions
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
Modes: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

int dup(int oldfd);
int dup2(int oldfd, int newfd);

Pipes
#include <unistd.h>
int pipe(int filedes[2]);
filedes[0] for reading,
filedes[1] for writing

Parsing with yacc
```
%right  '='
%right   '?'
%left   '+' '-'
%left   '*' '/' '%'
%left   UNARYMINUS
%right  '^'
expr '?' expr ':' expr
{ code(ternary); }
void
ternary(void)
{
        Datum d1, d2,
        d3;
        d3 = pop();
        d2 = pop();
        d1 = pop();
        if (d1.val)
        {
                push(d2);
        }
        else
        {
                push(d3);
        }
}
```

Process control
#include <unistd.h>
int execl(const char *path, const char *arg, ...); /* end with (char *)NULL */
< 0 if program not found, >= 0 failure

#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
0 if child, pid if parent, -1 if failed

#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);

Make
```
CC=gcc
CFLAGS=-g -Wall -W
LIBS=

testing: testing.o replace.o
        $(CC) $(CFLAGS) $^ -o $@
        $(LIBS)

.PHONY: clean tar

clean:
        rm -f *.o

tar: replace.tar

replace.tar: testing *.c *.h
        tar -cvf $@ $^

depend:
        makedepend -Y *.c

replace.o: replace.h
testing.o: replace.h
```

Threads
Priority inversion
Lock-free programming (compare & swap)
Semaphores:
wait(&sem)
signal(&sem)

Function pointers
return_type
(*f)(params…)

Signals
Sending signals:
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);

Catching signals:
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);

SIGINT (interrupt), SIGFPE (arithmetic), SIGKILL, SIGBUS, SIGSEGV, SIGPIPE, SIGARLM

Threads

```
int main() {
        pthread_t tid;
        pthread_create(&tid, NULL [attrib], thread, NULL
        [args, void *]);
        pthread_join(tid, NULL [ret val, void **]);
        exit(0);
}
void *thread(void *vargp) {
}
#include <pthread.h>
pthread_t pthread_self(void);
int pthread_cancel(pthread_t thread);
void pthread_exit(void *retval);
pthread_mutex_t fastmutex =
PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

pthread_mutex_lock( &condition_mutex );
while( count >= COUNT_HALT1 && count <=
COUNT_HALT2 )
{
  pthread_cond_wait( &condition_cond, &condition_mutex );
}
pthread_mutex_unlock( &condition_mutex );

pthread_mutex_lock( &condition_mutex );
if( count < COUNT_HALT1 || count > COUNT_HALT2 )
{
   pthread_cond_signal( &condition_cond );
}
pthread_mutex_unlock( &condition_mutex );
```

Intellectual property

- Trademarks
  - Symbol used in selling goods/services for identification
  - Being the first to use it
  - Issues: pop-up adverts, domain name registration, meta-tags
- Patents
  - A limited monopoly awarded to inventor for sharing knowledge
  - Must be filed within specified time
  - Issues: patenting of algorithm (GIF), Red Hat open source, not properly checked - parasites
- Copyright
  - Artistic works to give producer exclusive rights over them
  - Requires it to be non-ephemeral
  - Requires originality
  - Comparison with public domain/open source
  - Breach: may have been independently contrived
  - Issues: P2P - contributory (P2P developer knew of such infringement or should have known, and contributed), vicarious (developer had supervisory role, financial benefit), both (direct infringement occurred by party other than developer); viewing copyrighted articles
- Trade secrets
  - Private information, where patents/copyrights insufficient, keep from public
  - Reverse engineering, SMB protocol
- Internet law is being developed, however inappropriately, on existing legal frameworks

Errors

- "Systems view" of errors, wider context
- Socio-technical environment: software, hardware, physical environment, procedures, personnel, legal, data
- Reporting of incidents, documentation, overriding safety checks

Shell

- #!/bin/sh
- $variable
- Control flow:

```
if condition
then
        echo Condition is zero
else
        echo Condition non-zero
fi

for name in $*
do
        ...
done

while condition
do
        ...
done

case $variable in
1) x;;
2) y;;
*) else;;
esac
```

Shell commands:

- who
- cut
- tr [A-Z] [a-z] (takes -c -s -d)
- comm (-123, suppress uniq left/right/both)
- stat -c%A

CGI shell script (ends in .cgi)

```
#!/bin/sh
echo "Content-type: text/plain"
echo
echo "Content of page..."
```

Fns covered in sls

Getopt:

```
int getopt(int argc, char * const argv[],
        const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;

opterr = 0;
while ((c = getopt(argc, argv, "cL:")) != EOF) {
switch (c) {
case 'c': /* An option */
case 'L': /* An option that needs a value */
logfile = optarg;
case '?': /* An illegal option */
case ':': /* Missing required value */
/* keep reading from optind */

#include <unistd.h>
#include <sys/stat.h>
int stat(const char *pathname, struct stat *buffer);
s.st_uid, S_ISDIR(s.st_mode)

Directories:
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
int closedir(DIR *dir);
while (de = readdir(dir))
printf("%s\n", de->d_name);
#include <unistd.h>
int chdir(const char *path);

User db:
#include <pwd.h>
struct passwd *getpwnam(const char *username);
member pw_uid
```