C function prototypes and structs

```
int accept(int sock, struct sockaddr *addr, int *addrlen)
int bind(int sock, struct sockaddr *addr, int addrlen)
int close(int fd)
int closedir(DIR *dir)
int connect(int sock, struct sockaddr *addr, int addrlen)
int dup2(int oldfd, int newfd)
int execl(const char *path, const char *arg0, ... /*, (char *)0 */)
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */)
int execvp(const char *file, char *argv[])
int fclose(FILE *stream)
int FD ISSET(int fd, fd set *fds)
void FD SET(int fd, fd set *fds)
void FD CLR(int fd, fd set *fds)
void FD ZERO(fd set *fds)
char *fgets(char *s, int n, FILE *stream)
int fileno(FILE *stream)
pid t fork(void)
FILE *fopen(const char *file, const char *mode)
int fprintf(FILE * restrict stream, const char * restrict format,
...)
size t fread(void *ptr, size t size, size t nmemb, FILE *stream)
int fseek(FILE *stream, long offset, int whence)
/* whence has the value SEEK_SET, SEEK_CUR, or SEEK_END*/
size t fwrite(const void *ptr, size t size, size t nmemb, FILE
*stream);
pid t getpid(void)
pid t getppid(void)
int getc(FILE *stream)
int getchar(void)
unsigned long int htonl(unsigned long int hostlong) /* 4 bytes */
unsigned short int htons(unsigned short int hostshort) /* 2 bytes */
char *index(const char *s, int c)
int isalpha(int c)
int kill(int pid, int signo)
int listen(int sock, int n)
void *malloc(size t size)
int open(const char *path, int oflag)
/* oflag is O_WRONLY | O_CREAT for write and O_RDONLY for read */
DIR *opendir(const char *name)
int pipe(int filedes[2])
int putchar(int char)
int putc(int char, FILE *stream)
```

```
ssize t read(int d, void *buf, size t nbytes);
struct dirent *readdir(DIR *dir)
int select(int maxfdp1, fd set *readfds, fd set *writefds, fd set
*exceptfds, struct timeval *timeout)
int sigaction(int signum, const struct sigaction *act, struct
sigaction *oldact)  /* actions include SIG_DFL and SIG_IGN */
int sigaddset(sigset t *set, int signum)
int sigemptyset(sigset t *set)
int sigprocmask(int how, const sigset t *set, sigset t *oldset)
/* how has the value SIG BLOCK, SIG UNBLOCK, or SIG SETMASK */
unsigned int sleep(unsigned int seconds)
int socket(int family, int type, int protocol)
/* family=PF INET, type=SOCK STREAM, protocol=0 */
int sprintf(char *s, const char *format, ...)
int stat(const char *file name, struct stat *buf)
char *strchr(const char *s, int c)
size t strlen(const char *s)
char *strncat(char *dest, const char *src, size t n)
int strncmp(const char *s1, const char *s2, size t n)
char *strncpy(char *dest, const char *src, size t n)
long strtol(const char *restrict str, char **restrict endptr, int
base)
int toupper(int c)
int wait(int *status)
int waitpid(int pid, int *stat, int options) /* options = 0 or
WNOHANG*/
ssize t write(int d, const void *buf, size t nbytes);

WIFEXITED(status)      WEXITSTATUS(status)
WIFSIGNALED(status)    WTERMSIG(status)
WIFSTOPPED(status)     WSTOPSIG(status)
```

Useful structs

```
struct sigaction {
   void (*sa handler)(int);
   sigset t sa mask;
   int sa flags;
   }
struct hostent {
   char *h name; // name of host
   char **h aliases; // alias list
   int h addrtype; // host address type
   int h length; // length of address
   char *h addr; // address
   }
struct sockaddr in {
   sa family t sin family;
   unsigned short int sin port;
   struct in addr sin addr;
   unsigned char pad[8]; /*Unused*/
}
struct stat {
   dev t st dev; /* ID of device containing file */
   ino t st ino; /* inode number */
   mode t st mode; /* protection */
   nlink t st nlink; /* number of hard links */
   uid t st uid; /* user ID of owner */
   gid t st gid; /* group ID of owner */
   dev t st rdev; /* device ID (if special file) */
   off t st size; /* total size, in bytes */
   blksize t st blksize; /* blocksize for file system I/O */ blkcnt
t st blocks; /* number of 512B blocks allocated */ time t st atime;
/* time of last access */
   time t st mtime; /* time of last modification */
   time t st ctime; /* time of last status change */
   };
```

Shell comparison operators

Shell	Description
-d filename	Exists as a directory
-f filename	Exists as a regular file.
-r filename	Exists as a readable file
-w filename	Exists as a writable file.
-x filename	Exists as an executable file.
-z string	True if empty string
str1 = str2	True if str1 equals str2
str1 != str2	True if str1 not equal to str2
int1 -eq int2	True if int1 equals int2
-ne, -gt, -lt, -le	For numbers
!=, >, >=, <, <=	For strings
-a, -o	And, or.

Useful Makefile variables:

$@	target
$^	list of prerequisites
$<	first prerequisite
$?	return code of last program executed

Useful shell commands:

cat, cd, chmod, cp, cut, echo, expr, ls, mkdir, read, sort, uniq, set
ps aux - prints the list of currently running processes
grep (returns 0 if match is found, 1 if no match was found, and 2 if there was an error)
grep -v displays lines that do not match
wc (-clw options return the number of characters, lines, and words respectively)
diff (returns 0 if the files are the same, and 1 if the files differ)

$0	Script name
$#	Number of positional parameters
$*	List of all positional parameters
$?	Exit value of previously executed command

Operator Preferences

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right