CS-202 Computer Systems
Summary

Isaac Metthez

C – Structure & Preprocessor

Program skeleton:

```
type fnc(type1, ...);        // declaration (prototype)
int global;                  // external variable

int main(int argc, char *argv[]) {
    int local = 0;           // local variable
    return 0;
}

type fnc(type1 arg1, ...) {  // definition
    return value;
}
```

Preprocessor directives:

- `#include <lib>` system header `#include "file"` user header
- `#define NAME text` – text replacement (no semicolon!)
- `#define max(A,B) ((A)>(B)?(A):(B))` – macro (always parenthesize args)
- `#undef NAME` `#` stringify arg `##` concatenate tokens
- `#if expr` / `#ifdef NAME` / `#ifndef NAME` / `#else` / `#elif` / `#endif`
- `defined(NAME)` – test if macro defined (usable inside `#if`)
- `\` at end of line – line continuation

`exit(status)` – terminate program (0 = success)

Types & Storage Classes

`char`	1 byte, character or small int
`short`	16-bit integer
`int`	integer (typically 32-bit)
`long`	32-bit integer
`float`	single-precision floating point
`double`	double-precision floating point
`void`	no value / generic

Qualifiers: `signed` / `unsigned` `const` (read-only) `volatile`

Storage classes:

- `extern` – declared elsewhere (external linkage)
- `static` – local scope but persists, or file-private if global
- `register` – hint to put in CPU register (no `&` allowed)
- `auto` – default for local vars (rarely written)

Composite: `struct` `union` `enum`
`typedef int myint;` – create alias `sizeof(type)` / `sizeof obj` – size in bytes

Initialization:

```
int a = 5;
int arr[] = {1, 2, 3};       // size inferred
char s[] = "hello";          // null-terminated
int m[2][3] = {{1,2,3},{4,5,6}};
```

Constants

- Integer suffixes: `L`/`l` long, `U`/`u` unsigned, `UL` unsigned long
- Float suffixes: `F`/`f` float, no suffix = double
- Bases: `0755` octal, `0xFF` hex, `1.5e-3` exponential
- Char: `'a'` `'\n'` newline `'\t'` tab `'\r'` CR `'\0'` null `'\\'` backslash `'\"` quote
- Octal/hex char: `'\012'` `'\x0A'`
- String: `"abc"` – array of `char` ending with `'\0'` (null terminator)

Pointers, Arrays & Structs

Pointer basics:

```
int x = 5;
int *p = &x;     // p holds address of x
*p = 10;         // dereference: writes to x
p++;             // advances by sizeof(int)
```

- `void *` – generic pointer (must cast before dereferencing)
- `NULL` – null pointer (cast of 0)
- `const char *p` – ptr to const data `char * const p` – const ptr

Arrays:

```
int a[5];            // 5 ints, index 0..4
int m[3][4];         // 3 rows, 4 cols: m[i][j]
// array name decays to pointer to first element
// a[i] == *(a + i)
```

Struct:

```
struct Point { int x; int y; };  // template
struct Point p = {1, 2};         // init
p.x = 3;                         // member access
struct Point *pp = &p;
pp->y = 4;    // pointer member (== (*pp).y)
```

`union` – all members share the same memory (size = largest member)
`int flags : 3;` – bit field (3 bits wide)
`typedef struct { int x, y; } Point;` – anonymous struct with alias

Pointer to Function

```
// declaration
int  cmp(const void *a, const void *b);

// pointer-to-function variable
int (*fcmp)(const void *, const void *);
fcmp = cmp;        // assign (no & needed)
fcmp(a, b);        // call via pointer
(*fcmp)(a, b);     // equivalent explicit form

// typedef makes it readable
typedef int (*CmpFn)(const void *, const void *);
CmpFn fn = cmp;

// common use: callbacks for qsort / bsearch
int int_cmp(const void *a, const void *b) {
    return *(int*)a - *(int*)b;
}
qsort(arr, n, sizeof(int), int_cmp);

// function returning a function pointer
int (*get_op(char op))(int, int);
```

Rule: **read declarations inside-out** – `int (*pf)(int)` = "`pf` is a pointer to a function taking `int` returning `int`".

Operators (by precedence, high to low)

`() [] . ->`	call, subscript, member
`++ --`	post-increment/decrement
`++ -- + - ! ~ * & (type) sizeof`	unary (right-to-left)
`* / %`	multiply, divide, modulo
`+ -`	add, subtract
`« »`	bitwise shift
`< <= > >=`	relational
`== !=`	equality
`&`	bitwise AND
`^`	bitwise XOR
`|`	bitwise OR
`&&`	logical AND
`||`	logical OR
`?:`	conditional (right-to-left)
`= += -= *= /= %= &= ^= |= «= »=`	assignment (right-to-left)
`,`	comma (sequence)

All binary operators group **left-to-right** except unary, `?:`, and assignment.

Flow of Control

`if (e) s` `else if (e) s` `else s`
`while (e) s` `do s while (e);` (always once) `for (init; cond; step) s`

`switch (e) { case C: ...  break; default:  ...  }` – falls through without `break`

Jump: `break` exit loop/switch `continue` next iteration `return e;` `goto lbl;`

I/O `<stdio.h>`

Std: `stdin stdout stderr EOF`
`getchar() putchar(c)` `gets(s,max) puts(s)`
`printf("fmt",...) scanf("fmt",&x,...)`
`sprintf(s,"fmt",...) sscanf(s,"fmt",&x,...)`

File: `FILE *fp = fopen("f","r/w/a");`
`getc(fp) putc(c,fp) fgets(s,max,fp) fputs(s,fp)`
`fprintf(fp,"fmt",...) fscanf(fp,"fmt",...)`
`fclose(fp) feof(fp) ferror(fp)`

Format codes: `d`/`i` int `u` uint `c` char `s` str `f` double `e`/`E` exp `o` oct `x`/`X` hex `p` ptr
Flags: `-` left `+` sign `0` zero-pad `w.p` width.prec mod: `h` short `l` long

Strings `<string.h>`

`strlen(s)` `strcpy(d,s) strncpy(d,s,n)`
`strcat(d,s) strncat(d,s,n)` `strcmp(s,t) strncmp(s,t,n)`
`strchr(s,c) strrchr(s,c)` `memcpy(d,s,n) memmove(d,s,n)`
`memcmp(s,t,n) memchr(s,c,n) memset(s,c,n)`

Stdlib `<stdlib.h>`

Alloc: `malloc(n) calloc(n,sz) realloc(p,n) free(p)`
Conv: `atoi(s) atof(s) atol(s) strtol(s,e,b) strtod(s,e)`
Misc: `abs(n) rand() srand(n) exit(s) system(s)`
Sort/Search: `qsort(a,n,sz,cmp) bsearch(k,a,n,sz,cmp)`

Ctype `<ctype.h>`

`isalnum isalpha isdigit islower isupper isspace ispunct iscntrl`
`tolower(c) toupper(c)`

Varargs `<stdarg.h>`

`va_list ap; va_start(ap,last); va_arg(ap,type); va_end(ap);`

Standard Headers

`<assert.h> <errno.h> <limits.h> <float.h> <math.h>`
`<setjmp.h> <signal.h> <stddef.h> <locale.h> <time.h>`

Math `<math.h>` (args/ret `double`): `sin cos tan asin acos atan atan2`
`sinh cosh tanh  exp log log10  pow sqrt  ceil floor fabs fmod`

Time `<time.h>`: `clock() time() difftime(t2,t1) mktime(tp)`
`asctime(tp) ctime(tp) gmtime(tp) localtime(tp) strftime(s,n,fmt,tp)`
`tm`: `tm_sec tm_min tm_hour tm_mday tm_mon tm_year tm_wday tm_yday`

Limits `<limits.h>`: `CHAR_BIT INT_MAX INT_MIN LONG_MAX UINT_MAX ...`
Float `<float.h>`: `FLT_EPSILON FLT_MAX FLT_MIN DBL_EPSILON DBL_MAX ...`

Syscall Sequences – Files & Sockets

File I/O: `open` → `read`/`write`/`lseek` → `close`

UDP client: `socket` → `bind(port=0)` → `sendto`/`recvfrom` → `close`
port=0: kernel assigns ephemeral port `getsockname` to retrieve it

UDP server: `socket` → `bind` → `recvfrom`/`sendto` → `close`

TCP client: `socket` → `connect` → `send`/`recv` → `close`

TCP server: `socket` → `bind` → `listen` → `accept` → `send`/`recv` → `close`

Multiplexing: `select(nfds, &rd, &wr, &ex, timeout)` – blocks until ≥ 1 fd ready

Syscall Headers

`<unistd.h>`	`read write close lseek getpid fork exec`
`<fcntl.h>`	`open` + flags `O_RDONLY O_CREAT` ...
`<sys/stat.h>`	`stat fstat` + permission macros `S_IRUSR` ...
`<sys/socket.h>`	`socket bind connect listen accept send recv`
`<netinet/in.h>`	`sockaddr_in INADDR_ANY htons htonl`
`<arpa/inet.h>`	`inet_pton inet_ntop`
`<netdb.h>`	`getaddrinfo freeaddrinfo getnameinfo`
`<sys/select.h>`	`select FD_SET FD_CLR FD_ISSET FD_ZERO`
`<errno.h>`	`errno EACCES ENOENT EAGAIN EINTR` ...

Syscall Reference – Files & Sockets

`open(const char *path, int flags [, mode_t mode])` → `int fd` / `-1`
flags: `O_RDONLY O_WRONLY O_RDWR` | `O_CREAT O_TRUNC O_APPEND O_NONBLOCK`
mode (`mode_t`, only with `O_CREAT`): octal perms e.g. `0644`

`read(int fd, void *buf, size_t n)` → `ssize_t bytes` / `0 EOF` / `-1`
`write(int fd, const void *buf, size_t n)` → `ssize_t bytes written` / `-1`
`lseek(int fd, off_t offset, int whence)` → `off_t new pos` / `-1`
whence: `SEEK_SET` absolute `SEEK_CUR` relative `SEEK_END` from end
`close(int fd)` → `0` / `-1`

`socket(int domain, int type, int proto)` → `int fd` / `-1`
domain: `AF_INET` IPv4 `AF_INET6` IPv6 `AF_UNIX` local
type: `SOCK_STREAM` (TCP) `SOCK_DGRAM` (UDP) proto: `0` auto

`bind(int fd, const struct sockaddr *addr, socklen_t len)` → `0` / `-1`
`struct sockaddr_in a; a.sin_family=AF_INET; a.sin_port=htons(p); a.`

`connect(int fd, const struct sockaddr *addr, socklen_t len)` → `0` / `-1`
TCP: initiates 3-way handshake UDP: sets default peer (no handshake)

`listen(int fd, int backlog)` → `0` / `-1`
backlog: max pending connections in queue (e.g. `5`)
`accept(int fd, struct sockaddr *addr, socklen_t *len)` → `int new fd` / `-1`
blocks until client connects; fills `addr` with client address

`send(int fd, const void *buf, size_t n, int flags)` → `ssize_t` / `-1`
`recv(int fd, void *buf, size_t n, int flags)` → `ssize_t` / `0 closed` / `-1`
flags: `0` (normal) `MSG_WAITALL MSG_DONTWAIT MSG_PEEK`

`sendto(int fd, const void *buf, size_t n, int flags, const struct sockaddr *dst, socklen_t len)` → `ssize_t` / `-1` (UDP)
`recvfrom(int fd, void *buf, size_t n, int flags, struct sockaddr *src, socklen_t *len)` → `ssize_t` / `-1` (UDP)

`select(int nfds, fd_set *rd, fd_set *wr, fd_set *ex, struct timeval *tv)`
nfds = max fd + 1 tv=NULL blocks forever returns # ready fds
`FD_ZERO(&s) FD_SET(fd,&s) FD_CLR(fd,&s) FD_ISSET(fd,&s)`

Helpers: `htons/htonl` host→network `ntohs/ntohl` network→host (byte order)
`inet_pton(AF_INET, "1.2.3.4", &addr.sin_addr)` – string → binary
`inet_ntop(AF_INET, &addr.sin_addr, buf, sizeof buf)` – binary → string
`getaddrinfo(host, port, &hints, &res)` → 0 / err `freeaddrinfo(res)`

Recv / Send Loop Pattern

```
ssize_t n;
while ((n = recv(fd, buf, sizeof buf, 0)) > 0) {
    // process n bytes in buf
}
// n == 0: connection closed by peer  n < 0: error
```

```
size_t sent = 0;
while (sent < len) {
    ssize_t n = send(fd, buf + sent, len - sent, 0);
    if (n <= 0) break;   // error or closed
    sent += n;
}
```

Hostname → IP (`getaddrinfo`):

```
struct addrinfo hints = {0}, *res;
hints.ai_family   = AF_INET;       // AF_UNSPEC for IPv4+IPv6
hints.ai_socktype = SOCK_STREAM;  // SOCK_DGRAM for UDP
getaddrinfo("example.com", "80", &hints, &res);


// res->ai_addr ready to pass to connect/bind
// ((struct sockaddr_in *)res->ai_addr)->sin_addr
freeaddrinfo(res);
```

Fork per Connection (TCP server)

```
while (1) {
    int client = accept(srv, NULL, NULL);
    if (fork() == 0) {
        close(srv);     // child: does not need listener
        // handle client...
        close(client);
        exit(0);
    }
    close(client);      // parent: does not keep client fd
}
```

`fork()` returns 0 in child, child PID in parent, -1 on error.
Add `signal(SIGCHLD, SIG_IGN)` or `waitpid` to avoid zombie processes.

GCC

`gcc file.c -o out` – compile & link
`gcc -c file.c` – compile only → `file.o` `gcc *.o -o out` – link objects
Flags: `-Wall -Wextra` warnings `-g` debug info `-O0/O1/O2/O3` optimization
`-std=c99` standard `-I dir` include path `-L dir` lib search path
`-E` preprocessor only `-S` assembly only `-v` verbose
Linking: `gcc main.c util.c -o out` `-lfoo` links `libfoo.so/libfoo.a`
Convention: `-l` + name without `lib` prefix and extension e.g. `libm.so` → `-lm`
Common: `-lm` math `-lpthread` threads `-lssl` OpenSSL
Find: `ldconfig -p | grep name` or `pkg-config -libs name`

GDB

`gdb ./out` – start `gdb ./out core` – post-mortem with core dump

`run [args]`	start program
`break file:line` / `b fn`	set breakpoint
`next` / `n`	next line (step over)
`step` / `s`	step into function
`continue` / `c`	resume until next breakpoint
`print expr` / `p`	print value
`backtrace` / `bt`	call stack
`info locals`	all local variables
`watch expr`	break when expr changes
`quit` / `q`	exit
`layout src`	TUI: source view
`layout asm`	TUI: assembly view
`layout split`	TUI: source + asm

Make

```
CC     = gcc
CFLAGS = -Wall -g

out: main.o util.o          # target: dependencies
 $(CC) $(CFLAGS) -o out main.o util.o

%.o: %.c                    # pattern rule


 $(CC) $(CFLAGS) -c $<   # $< = first dep, $@ = target

clean:
 rm -f *.o out
```

`make` builds first target `make clean` runs clean rule
`.PHONY: clean` – declare non-file targets to avoid conflicts