# MORE C

Samira Khan

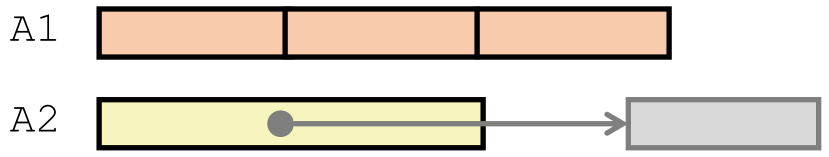
UNIVERSITY of VIRGINIA

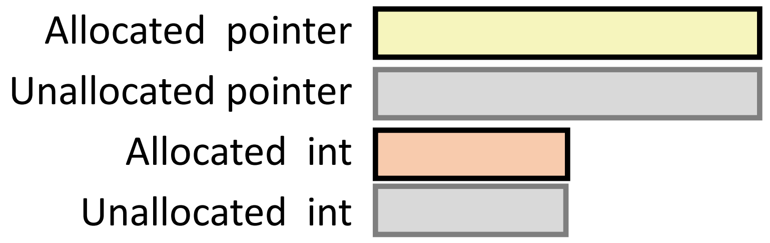
## Agenda

- Pointer vs array
- Using man page
- Structure and dynamic allocation
- Undefined behavior
- Into to instruction set architecture (ISA)

# Understanding Pointers & Arrays

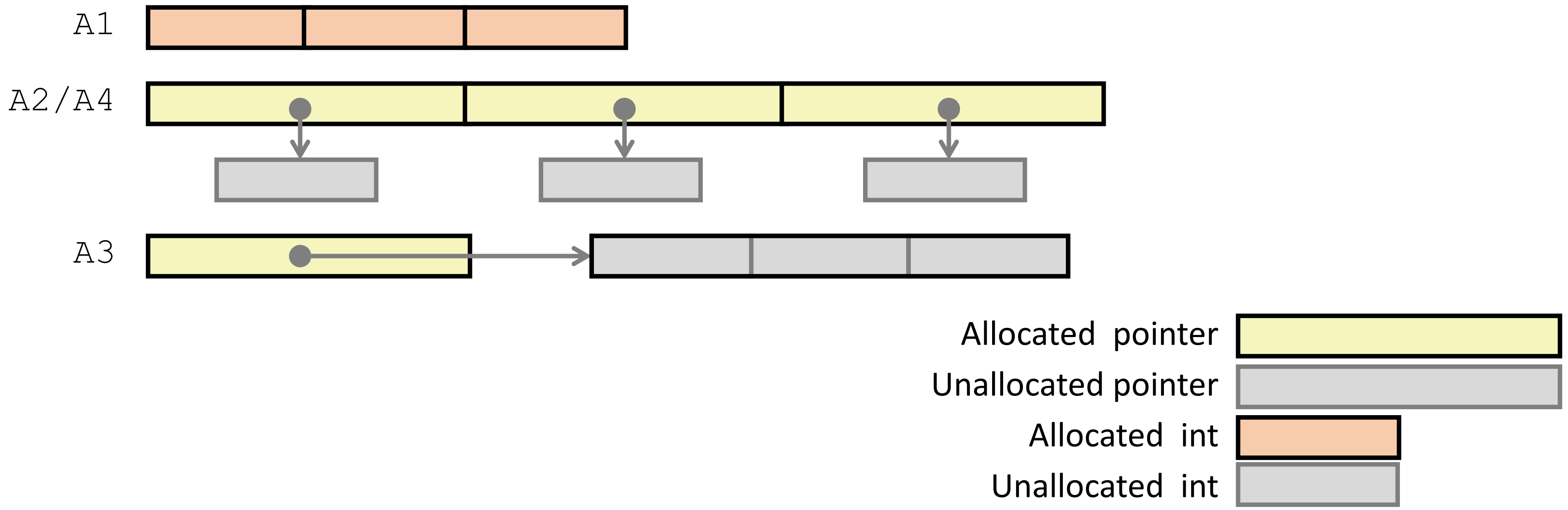
| Variable Decl | size |
|---|---|
| `int A1[3]` | 12 |
| `int *A2` | 8 |





# Understanding Pointers & Arrays

| Variable Decl | Size |
|---|---|
| `int A1[3]` | 12 |
| `int *A2[3]` | 24 |
| `int (*A3)[3]` | 8 |
| `int (*A4[3])` | 24 |



## Array vs. Pointer

```
int array[100];
int *pointer;
```

```
pointer = array;
```
✓

- same as pointer = &(array[0]);

```
array = pointer;
```
✗

## Array vs. Pointer

```
int array[100];
int *pointer = array;
```

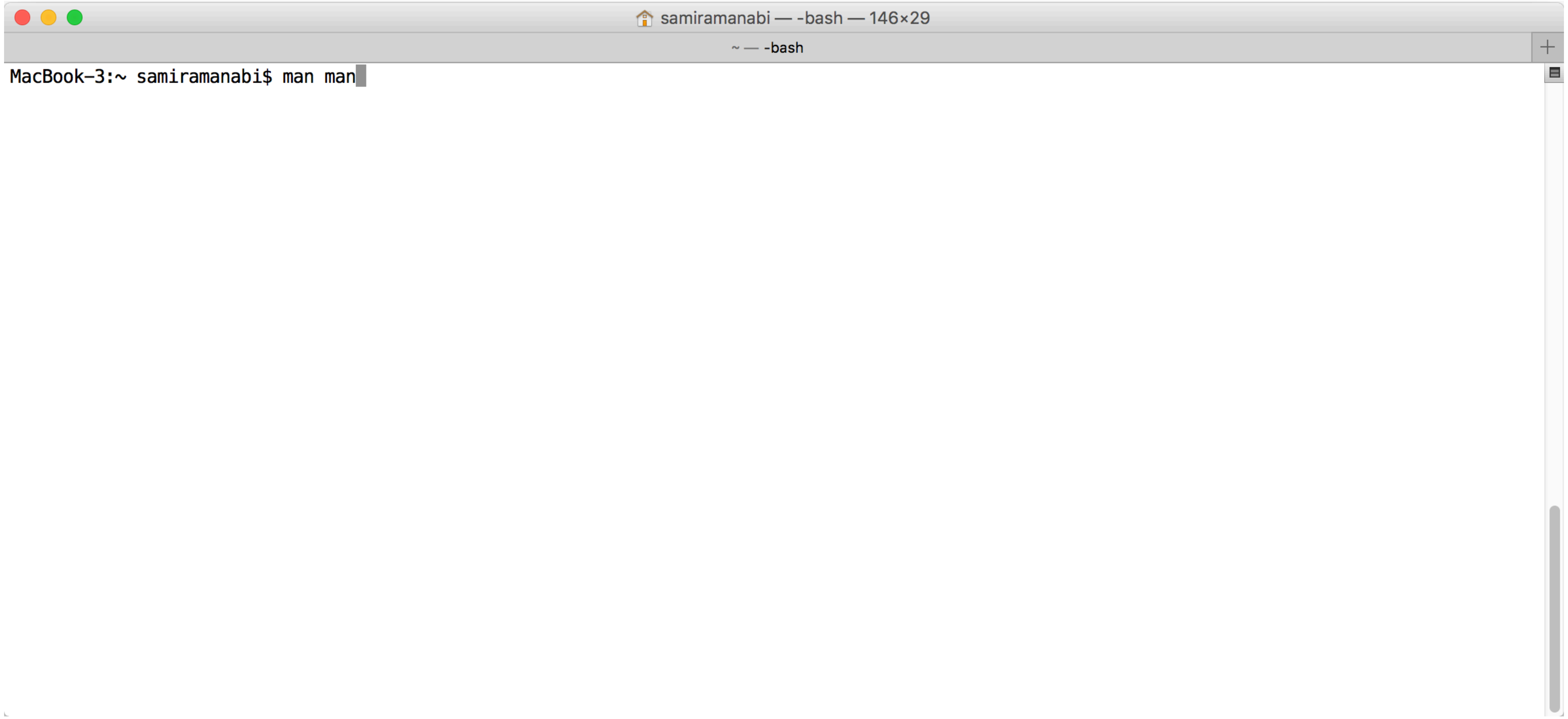
- sizeof(array) == 400 (size of all elements)
- sizeof(pointer) == 8 (size of address)

- sizeof(&array[0]) == ???
- (&array[0] same as &(array[0]))

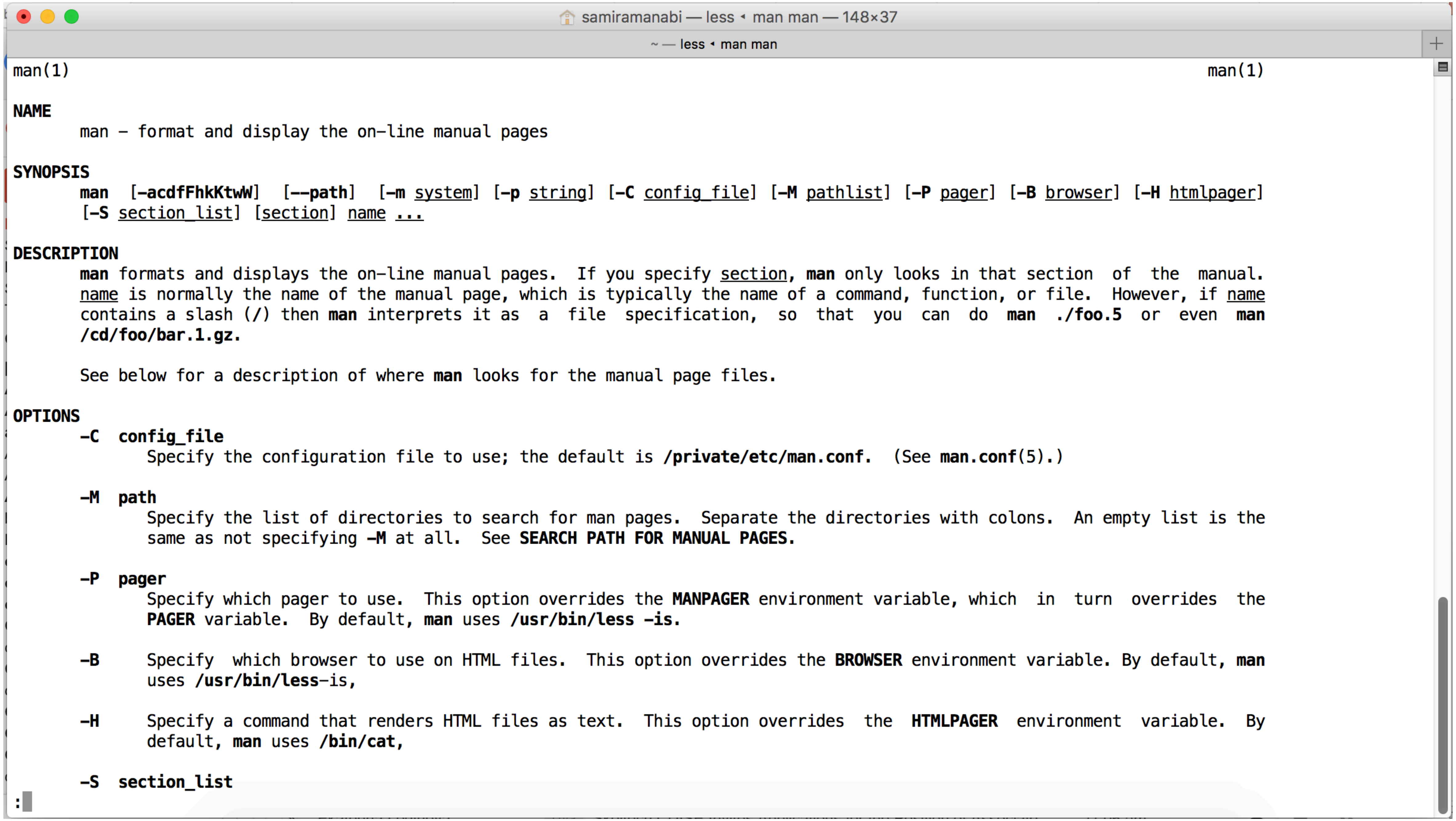
## Agenda

- Pointer vs array
- Using man page
- Structure and dynamic allocation
- Undefined behavior
- Into to instruction set architecture (ISA)

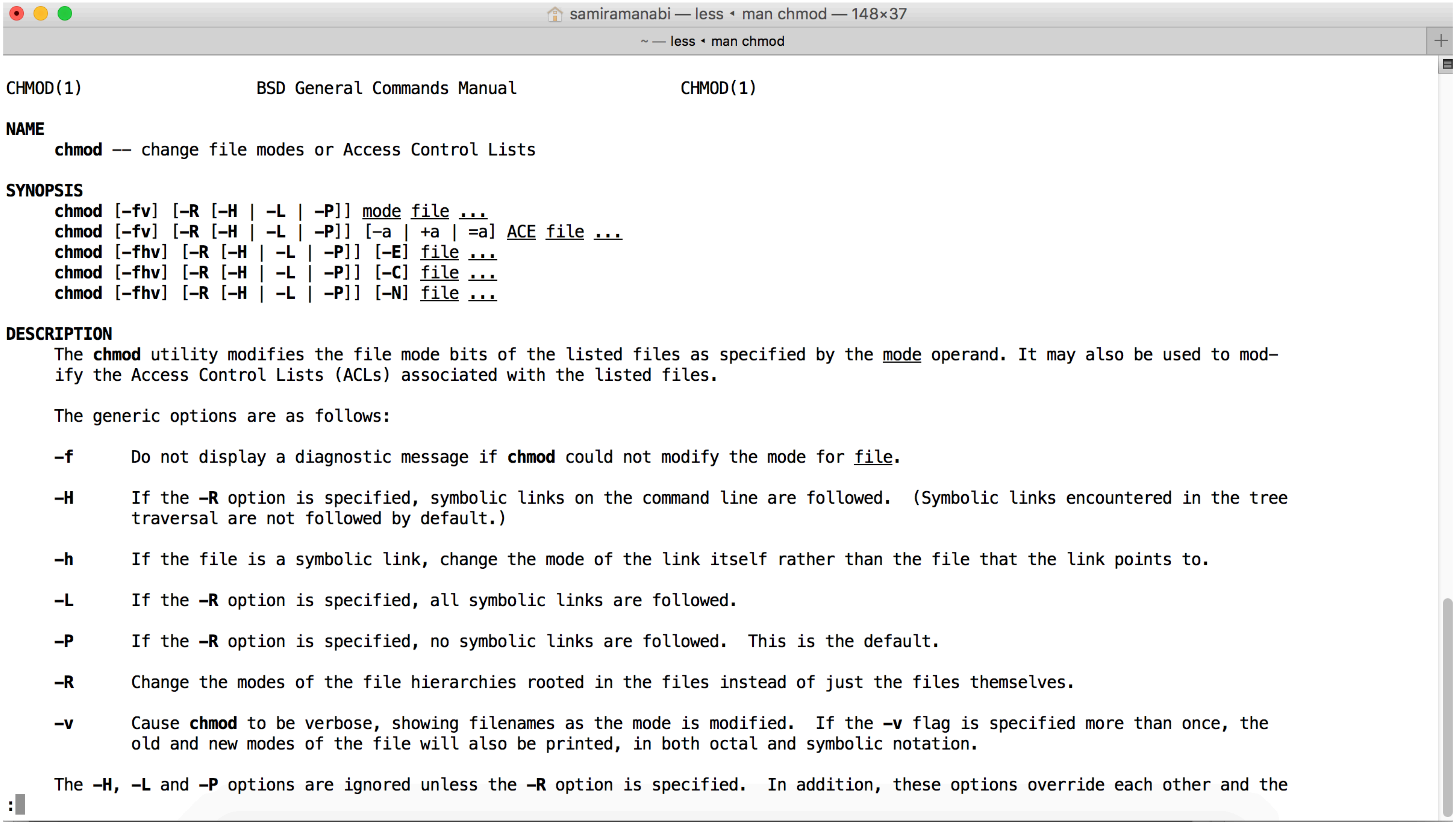
# interlude: command line tips



```
MacBook-3:~ samiramanabi$ man man
```



```
man(1)                                                                                          man(1)

NAME
       man - format and display the on-line manual pages

SYNOPSIS
       man  [-acdfFhkKtwW]  [--path]  [-m system] [-p string] [-C config_file] [-M pathlist] [-P pager] [-B browser] [-H htmlpager]
       [-S section_list] [section] name ...

DESCRIPTION
       man formats and displays the on-line manual pages.  If you specify section, man only looks in that section  of  the  manual.
       name is normally the name of the manual page, which is typically the name of a command, function, or file.  However, if name
       contains a slash (/) then man interprets it as  a  file  specification,  so  that  you  can  do  man  ./foo.5  or  even  man
       /cd/foo/bar.1.gz.

       See below for a description of where man looks for the manual page files.

OPTIONS
       -C  config_file
              Specify the configuration file to use; the default is /private/etc/man.conf.  (See man.conf(5).)

       -M  path
              Specify the list of directories to search for man pages.  Separate the directories with colons.  An empty list is the
              same as not specifying -M at all.  See SEARCH PATH FOR MANUAL PAGES.

       -P  pager
              Specify which pager to use.  This option overrides the MANPAGER environment variable, which  in  turn  overrides  the
              PAGER variable.  By default, man uses /usr/bin/less -is.

       -B     Specify  which browser to use on HTML files.  This option overrides the BROWSER environment variable. By default, man
              uses /usr/bin/less-is,

       -H     Specify a command that renders HTML files as text.  This option overrides  the  HTMLPAGER  environment  variable.  By
              default, man uses /bin/cat,

       -S  section_list
:
```



samiramanabi — less ◂ man chmod — 148×37

~ — less ◂ man chmod

```
CHMOD(1)                  BSD General Commands Manual                  CHMOD(1)

NAME
     chmod -- change file modes or Access Control Lists

SYNOPSIS
     chmod [-fv] [-R [-H | -L | -P]] mode file ...
     chmod [-fv] [-R [-H | -L | -P]] [-a | +a | =a] ACE file ...
     chmod [-fhv] [-R [-H | -L | -P]] [-E] file ...
     chmod [-fhv] [-R [-H | -L | -P]] [-C] file ...
     chmod [-fhv] [-R [-H | -L | -P]] [-N] file ...

DESCRIPTION
     The chmod utility modifies the file mode bits of the listed files as specified by the mode operand. It may also be used to mod-
     ify the Access Control Lists (ACLs) associated with the listed files.

     The generic options are as follows:

     -f      Do not display a diagnostic message if chmod could not modify the mode for file.

     -H      If the -R option is specified, symbolic links on the command line are followed.  (Symbolic links encountered in the tree
             traversal are not followed by default.)

     -h      If the file is a symbolic link, change the mode of the link itself rather than the file that the link points to.

     -L      If the -R option is specified, all symbolic links are followed.

     -P      If the -R option is specified, no symbolic links are followed.  This is the default.

     -R      Change the modes of the file hierarchies rooted in the files instead of just the files themselves.

     -v      Cause chmod to be verbose, showing filenames as the mode is modified.  If the -v flag is specified more than once, the
             old and new modes of the file will also be printed, in both octal and symbolic notation.

     The -H, -L and -P options are ignored unless the -R option is specified.  In addition, these options override each other and the
:
```

# chmod

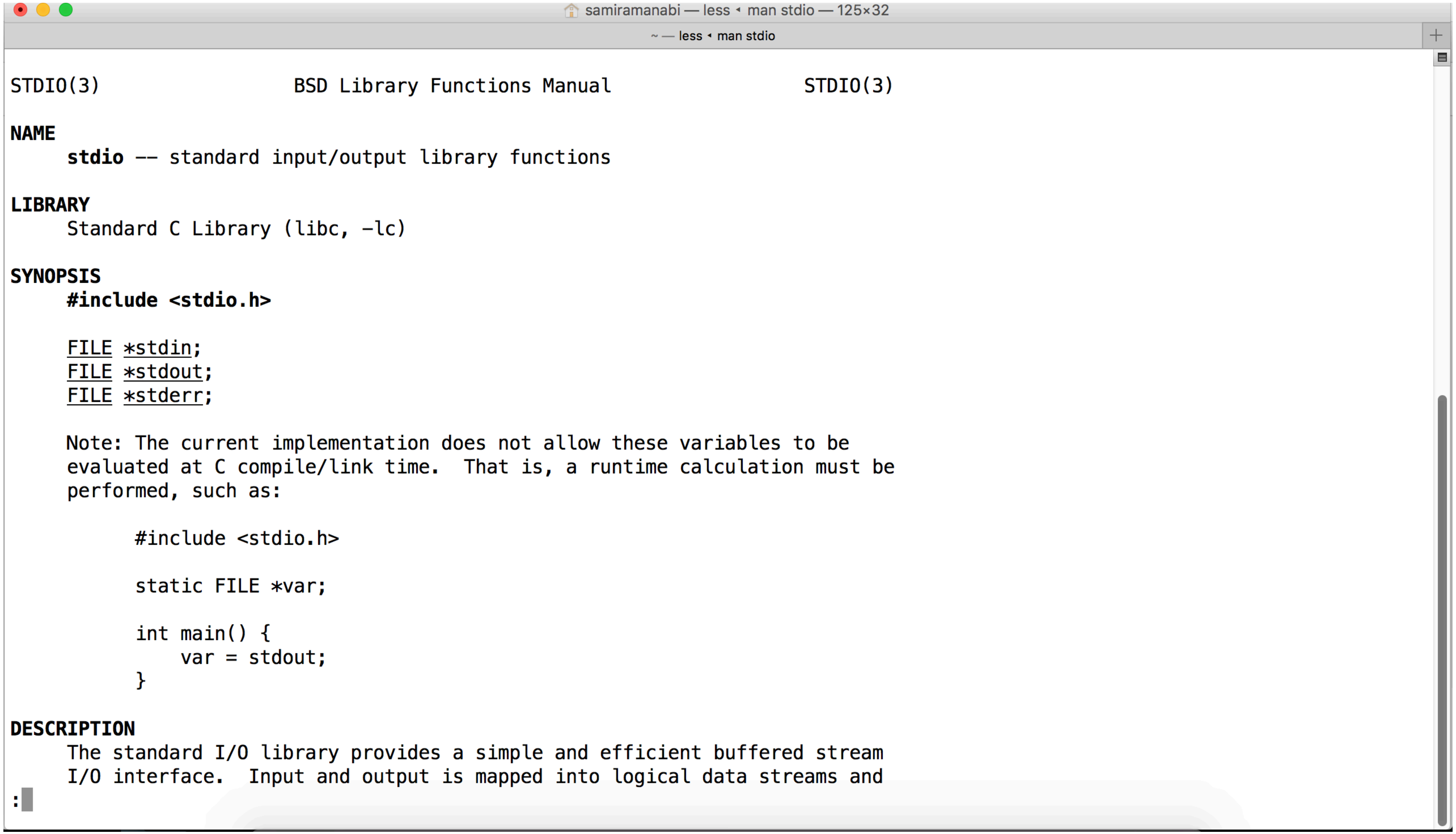
- chmod --recursive og-r /home/USER
- og → others and group (student)
  - user (yourself) / group / others
- - → remove
  - - remove / + add
- r → read
  - read / write / execute

## tar

- Standard Linux/Unix file archive utility
- Table of contents: tar tf filename.tar
- eXtract: tar xvf filename.tar
- Create: tar cvf filename.tar directory
- (v: verbose; f: file — default is tape)

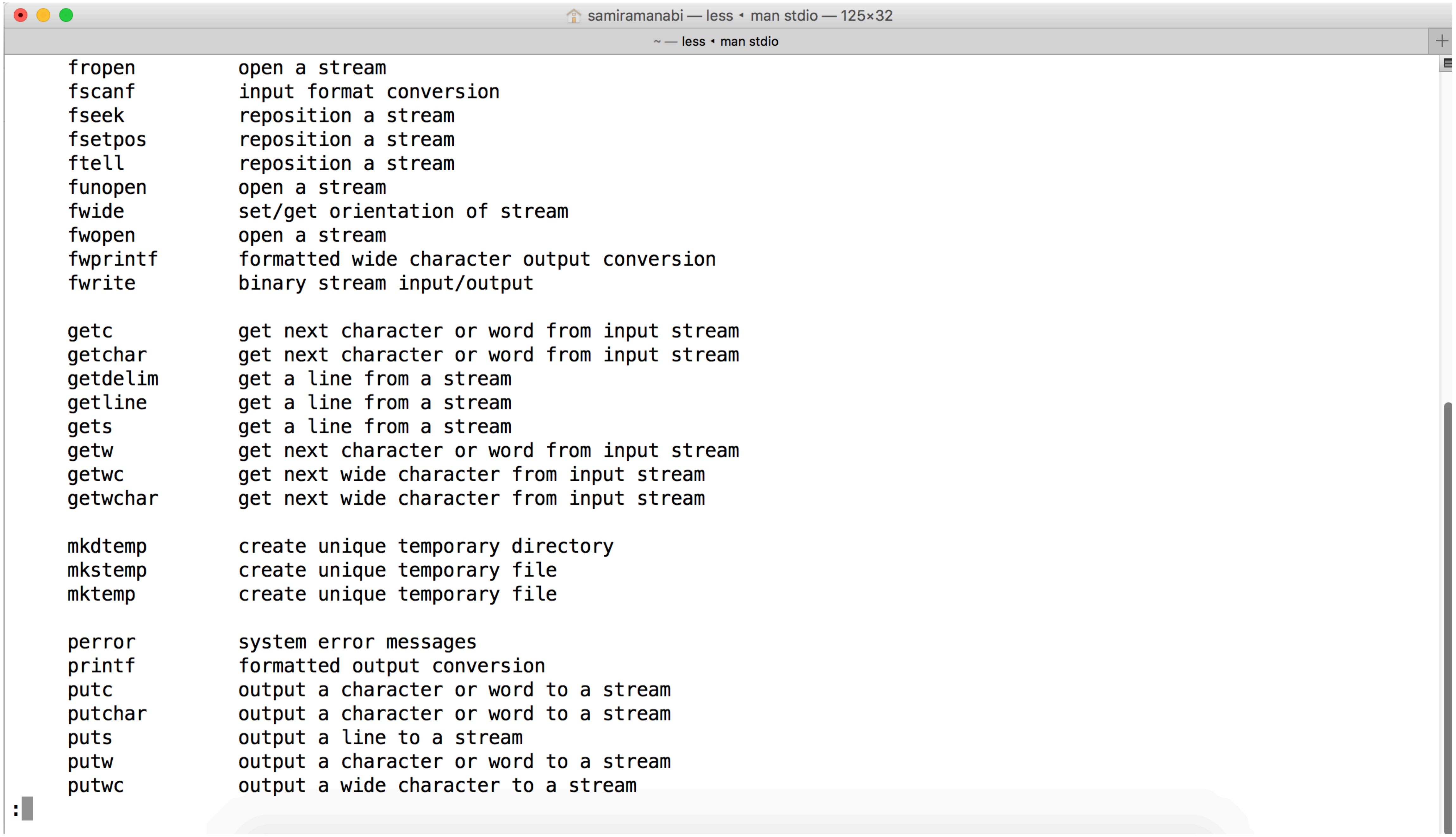
## stdio

- C does not have <iostream>
- instead <stdio.h>


samiramanabi — less ◂ man stdio — 125×32

~ — less ◂ man stdio

```
STDIO(3)                 BSD Library Functions Manual                 STDIO(3)

NAME
     stdio -- standard input/output library functions

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     FILE *stdin;
     FILE *stdout;
     FILE *stderr;

     Note: The current implementation does not allow these variables to be
     evaluated at C compile/link time.  That is, a runtime calculation must be
     performed, such as:

           #include <stdio.h>

           static FILE *var;

           int main() {
               var = stdout;
           }

DESCRIPTION
     The standard I/O library provides a simple and efficient buffered stream
     I/O interface.  Input and output is mapped into logical data streams and
:
```



```
     fropen          open a stream
     fscanf          input format conversion
     fseek           reposition a stream
     fsetpos         reposition a stream
     ftell           reposition a stream
     funopen         open a stream
     fwide           set/get orientation of stream
     fwopen          open a stream
     fwprintf        formatted wide character output conversion
     fwrite          binary stream input/output

     getc            get next character or word from input stream
     getchar         get next character or word from input stream
     getdelim        get a line from a stream
     getline         get a line from a stream
     gets            get a line from a stream
     getw            get next character or word from input stream
     getwc           get next wide character from input stream
     getwchar        get next wide character from input stream

     mkdtemp         create unique temporary directory
     mkstemp         create unique temporary file
     mktemp          create unique temporary file

     perror          system error messages
     printf          formatted output conversion
     putc            output a character or word to a stream
     putchar         output a character or word to a stream
     puts            output a line to a stream
     putw            output a character or word to a stream
     putwc           output a wide character to a stream
:
```

## printf

```
int custNo = 1000;
const char *name = "Jane Smith"
printf("Customer #%d: %s\n”, custNo, name);
// "Customer #1000: Jane Smith"
// same as:
//cout << "Customer #" << custNo
// << ": " << name << endl;
```

Format string must match types of argument

# printf formats quick reference

| Specifier | Argument Type | Example |
|---|---|---|
| %s | char * | Hello, World! |
| %p | any pointer | 0x4005d4 |
| %d | int/short/char | 42 |
| %u | unsigned int/short/char | 42 |
| %x | unsigned int/short/char | 2a |
| %ld | long | 42 |
| %f | double/float | 42.000000<br>0.000000 |
| %e | double/float | 4.200000e+01<br>4.200000e-19 |
| %g | double/float | 42, 4.2e-19 |
| %% | no argument | % |

man 3 printf

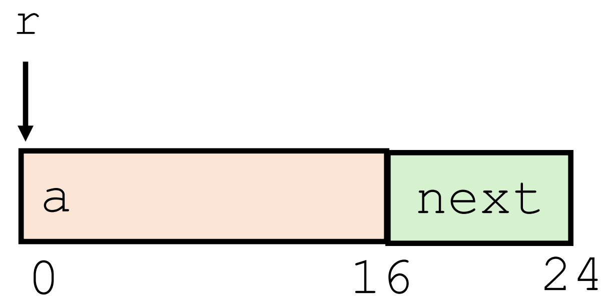
## Agenda

- Pointer vs array
- Using man page
- Structure and dynamic allocation
- Undefined behavior
- Into to instruction set architecture (ISA)

# Structure

- Structure represented as block of memory
  - **Big enough to hold all of the fields**
- Fields ordered according to declaration

```
struct rec {
    int a[4];
    struct rec *next;
};
```



## struct

```
struct rational {
int numerator;
int denominator;
};
// ...
struct rational two_and_a_half;
two_and_a_half.numerator = 5;
two_and_a_half.denominator = 2;
struct rational *pointer = &two_and_a_half;
printf("%d/%d\n",
pointer->numerator,
pointer->denominator);
```

## typedef struct

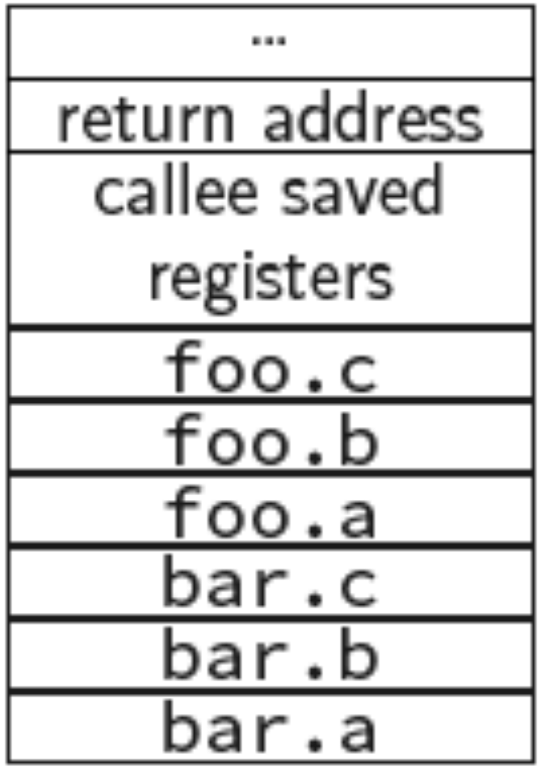
```
struct other_name_for_rational {
int numerator;
int denominator;
};
typedef struct other_name_for_rational rational;
// ...
rational two_and_a_half;
two_and_a_half.numerator = 5;
two_and_a_half.denominator = 2;
rational *pointer = &two_and_a_half;
printf("%d/%d\n",
pointer->numerator,
pointer->denominator);
```

## typedef struct

```
struct other_name_for_rational {
int numerator;
int denominator;
};
typedef struct other_name_for_rational rational;
// same as:
typedef struct other_name_for_rational {
int numerator;
int denominator;
} rational;
// almost the same as:
typedef struct {
int numerator;
int denominator;
} rational;
```

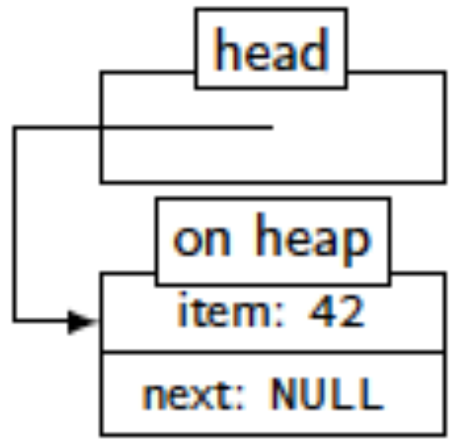
# structs aren’t references

```
typedef struct {
long a; long b; long c;
} triple;
...
triple foo;
foo.a = foo.b = foo.c = 3;
triple bar = foo;
bar.a = 4;
// foo is 3, 3, 3
// bar is 4, 3, 3
```



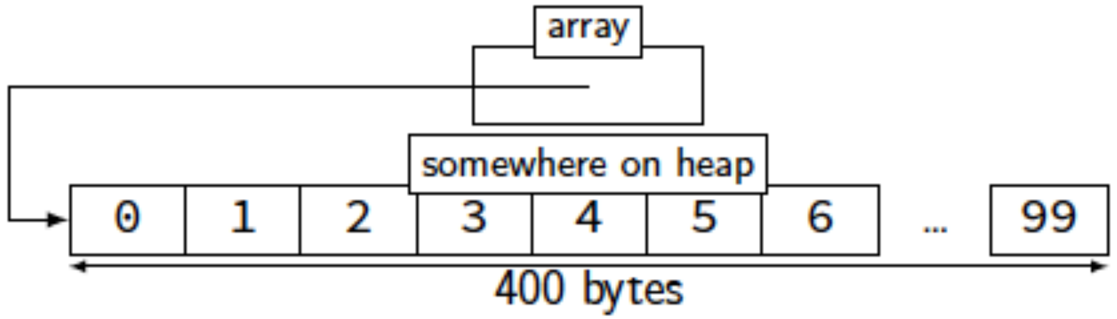
## Dynamic allocation

```
typedef struct list_t {
int item;
struct list_t *next;
} list;
// ...
list* head = malloc(sizeof(list));
/* C++: new list; */
head->item = 42;
head->next = NULL;
// ...
free(head);
/* C++: delete list */
```



## Dynamic arrays

```
int *array = malloc(sizeof(int)*100);
// C++: new int[100]
for (i = 0; i < 100; ++i) {
array[i] = i;
}
// ...
free(array); // C++: delete[] array
```



# unsigned and signed types

| Type | min | max |
|---|---|---|
| signed int = signed = int | $-2^{31}$ | $2^{31}$ - 1 |
| unsigned int = unsigned | 0 | $2^{32}$ - 1 |
| signed long = long | $-2^{63}$ | $2^{63}$ - 1 |
| unsigned long | 0 | $2^{64}$ - 1 |

# Agenda

- Pointer vs array
- Using man page
- Structure and dynamic allocation
- Undefined behavior
- Into to instruction set architecture (ISA)

## unsigned/signed comparison trap

```
int x = -1;
unsigned int y = 0;
printf("%d\n", x < y);
```

- result is 0
- short solution: don’t compare signed to unsigned:
  - (long) x < (long) y
- Compiler converts both to same type first
  - int if all possible values fit
  - otherwise: first operand (x, y) type from this list:
  - unsigned long, long, unsigned int, int

# C evolution and standards

- 1978: Kernighan and Ritchie publish The C Programming Language— “K&R C”
  - very different from modern C
- 1989: ANSI standardizes C — C89/C90/-ansi
  - compiler option: -ansi, -std=c90
  - looks mostly like modern C
- 1999: ISO (and ANSI) update C standard — C99
  - compiler option: -std=c99
  - adds: declare variables in middle of block
  - adds: // comments
- 2011: Second ISO update — C11

## Undefined behavior example (1)

```
#include <stdio.h>
#include <limits.h>
int test(int number) {
return (number + 1) > number;
}
int main(void) {
printf("%d\n", test(INT_MAX));
}
```

- without optimizations: 0
- with optimizations: 1

# Undefined behavior example (2)

```
int test(int number) {
return (number + 1) > number;
}
```

- Optimized:

```
test:
movl $1, %eax # eax   1
ret
```

- Less optimized:

```
test:
leal 1(%rdi), %eax # eax   rdi + 1
cmpl %eax, %edi
setl %al           # al   eax < edi
movzbl %al, %eax   # eax   al (pad with zeros)
ret
```

## Undefined behavior

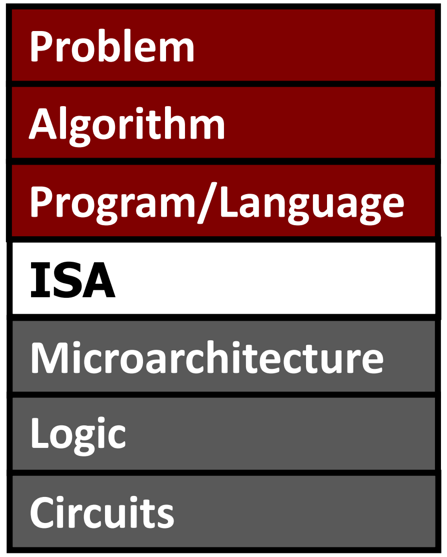
- compilers can do whatever they want
  - what you expect
  - crash your program
  - …
- common types:
  - signed integer overflow/underflow
  - out-of-bounds pointers
  - integer divide-by-zero
  - writing read-only data
  - out-of-bounds shift (later)

## Agenda

- Pointer vs array
- Using man page
- Structure and dynamic allocation
- Undefined behavior
- Into to instruction set architecture (ISA)

# LEVELS OF TRANSFORMATION

- ISA
  - Agreed upon interface between software and hardware
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write system/user programs
- Microarchitecture
  - Specific implementation of an ISA
  - Not visible to the software
- Microprocessor
  - **ISA, uarch**, circuits
  - "Architecture" = ISA + microarchitecture



# ISA VS. MICROARCHITECTURE

- What is part of ISA vs. Uarch?
  - Gas pedal: interface for “acceleration”
  - Internals of the engine: implements “acceleration”
  - Add instruction vs. Adder implementation

- Implementation (uarch) can be various as long as it satisfies the specification (ISA)
  - Bit serial, ripple carry, carry lookahead adders
  - x86 ISA has many implementations: 286, 386, 486, Pentium, Pentium Pro, …

- Uarch usually changes faster than ISA
  - Few ISAs (x86, SPARC, MIPS, Alpha) but many uarchs
  - *Why?*

## ISA

- Instructions
  - Opcodes, Addressing Modes, Data Types
  - Instruction Types and Formats
  - Registers, Condition Codes
- Memory
  - Address space, Addressability, Alignment
  - Virtual memory management
- Call, Interrupt/Exception Handling
- Access Control, Priority/Privilege
- I/O
- Task Management
- Power and Thermal Management
- Multi-threading support, Multiproc

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 1:
Basic Architecture

## ISAs being manufactured today

- x86 — dominant in desktops, servers
- ARM — dominant in mobile devices
- POWER — Wii U, IBM supercomputers and some servers
- MIPS — common in consumer wifi access points
- SPARC — some Oracle servers, Fujitsu supercomputers
- z/Architecture — IBM mainframes
- Z80 — TI calculators
- SHARC — some digital signal processors
- Itanium — some HP servers (being retired)
- RISC V — some embedded
- …

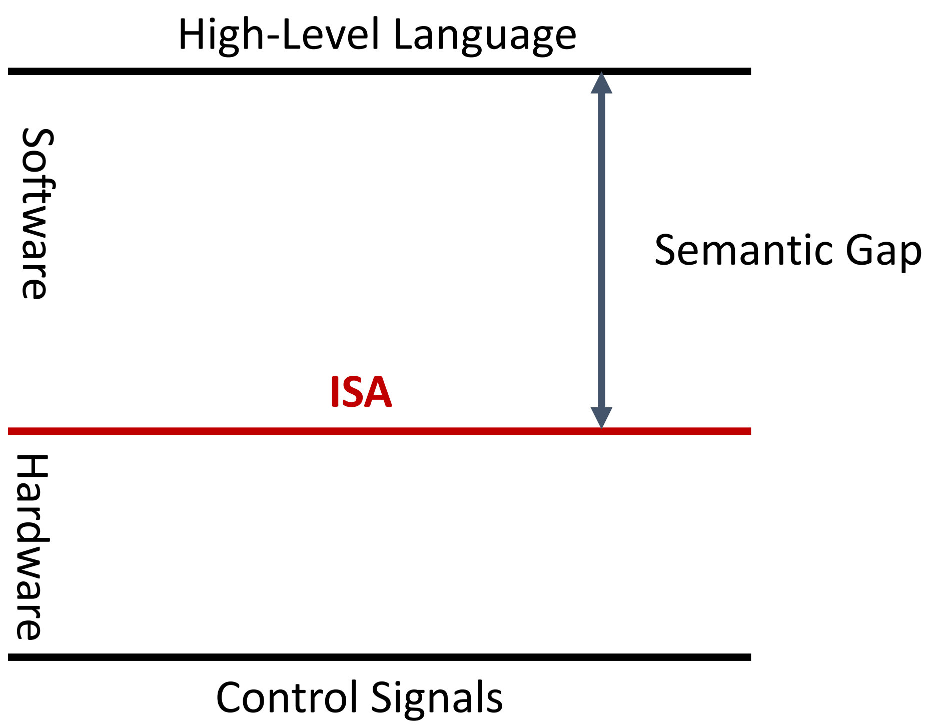
# Microarchitecture

- Implementation of the ISA under specific design constraints and goals
- Anything done in hardware without exposure to software
  - Pipelining
  - In-order versus out-of-order instruction execution
  - Memory access scheduling policy
  - Speculative execution
  - Superscalar processing (multiple instruction issue?)
  - Clock gating
  - Caching? Levels, size, associativity, replacement policy
  - Prefetching?
  - Voltage/frequency scaling?
  - Error correction?

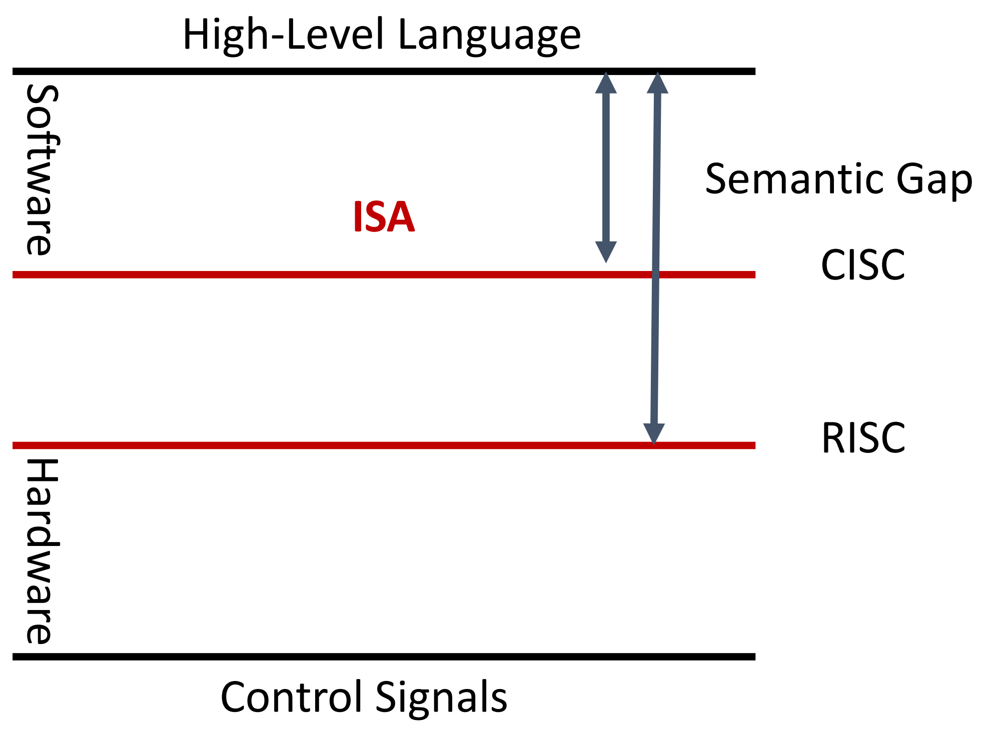
# ISA-LEVEL TRADEOFFS: SEMANTIC GAP

- Where to place the ISA? Semantic gap
  - Closer to high-level language (HLL) or closer to hardware control signals? → Complex vs. simple instructions
  - RISC vs. CISC vs. HLL machines
    - FFT, QUICKSORT, POLY, FP instructions?
    - VAX INDEX instruction (array access with bounds checking)
      - e.g., A[i][j][k] one instruction with bound check

# SEMANTIC GAP



# SEMANTIC GAP



# ISA-LEVEL TRADEOFFS: SEMANTIC GAP

- Where to place the ISA? Semantic gap
  - Closer to high-level language (HLL) or closer to hardware control signals? → Complex vs. simple instructions
  - RISC vs. CISC vs. HLL machines
    - FFT, QUICKSORT, POLY, FP instructions?
    - VAX INDEX instruction (array access with bounds checking)
  - Tradeoffs:
    - Simple compiler, complex hardware vs. complex compiler, simple hardware
      - Caveat: Translation (indirection) can change the tradeoff!
    - Burden of backward compatibility
    - Performance?
      - Optimization opportunity: Example of VAX INDEX instruction: who (compiler vs. hardware) puts more effort into optimization?
      - Instruction size, code size

## SMALL SEMANTIC GAP EXAMPLES IN VAX

- FIND FIRST
  - Find the first set bit in a bit field
  - Helps OS resource allocation operations
- SAVE CONTEXT, LOAD CONTEXT
  - Special context switching instructions
- INSQUEUE, REMQUEUE
  - Operations on doubly linked list
- INDEX
  - Array access with bounds checking
- STRING Operations
  - Compare strings, find substrings, …
- Cyclic Redundancy Check Instruction
- EDITPC
  - Implements editing functions to display fixed format output

- Digital Equipment Corp., "VAX11 780 Architecture Handbook," 1977-78.

# CISC vs. RISC

REPMOVS

*x86: REP MOVS DEST SRC*

```
X:
MOV
ADD
COMP
MOV
ADD
JMP X
```

**Which one is easy to optimize?**

# SMALL VERSUS LARGE SEMANTIC GAP

- CISC vs. RISC
  - Complex instruction set computer → complex instructions
    - Initially motivated by “not good enough” code generation
  - Reduced instruction set computer → simple instructions
    - John Cocke, mid 1970s, IBM 801
      - Goal: enable better compiler control and optimization

- RISC motivated by
  - Memory stalls (no work done in a complex instruction when there is a memory stall?)
    - When is this correct?
  - Simplifying the hardware → lower cost, higher frequency
  - Enabling the compiler to optimize the code better
    - Find fine-grained parallelism to reduce stalls

# SMALL VERSUS LARGE SEMANTIC GAP

- John Cocke's RISC (large semantic gap) concept:
  - Compiler generates control signals: open microcode

- Advantages of Small Semantic Gap (Complex instructions)

  + Denser encoding → smaller code size → saves off-chip bandwidth, better cache hit rate (better packing of instructions)

  + Simpler compiler

- Disadvantages

  - Larger chunks of work → compiler has less opportunity to optimize

  - More complex hardware → translation to control signals and optimization needs to be done by hardware

- Read Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.