+CLANG FORMAT COMMAND:
clang-format -i -style=file (file_name here)

RUN WITH:
$ clang -Wall -Wextra -Werror -Wpedantic -o binary file.c
./binary

- **UNIX commands (makefile)**
  - **$^ - evaluates all sources (all prereq)**
  - **$< - last target file (first prereq)**
  - **$@ - automatically substitutes main**
  - **$? - list of dependencies more recent than the target**
  - **$(wildcard *.c) - gets all .c files**
- **UNIX commands**
  - **diff - compares files line by line**
  - **tee - copies stdin to stdout, copying the read contents to 0 or more files**
  - **sort - sort or merge lines of files**
  - **uniq - report or filter out duplicate lines in a file**
  - **wc - report word, line, character, and/or byte count**
  - **mv - move/rename file**
  - **cp - copy file**
  - **rm - remove file**

ESCAPE SEQUENCES:

\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
A	Vertical Tab

\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

- Format Specifiers

SPECIFIER	**USED FOR**
%c	a single character
%s	a string
%hi	short (signed)
%hu	short (unsigned)

%Lf	long double
%n	prints nothing
%d	a decimal integer (assumes base 10)
%i	a decimal integer (detects the base automatically)
%o	an octal (base 8) integer
%x	a hexadecimal (base 16) integer
%p	an address (or pointer)
%f	a floating point number for floats
%u	int unsigned decimal

%e	a floating point number in scientific notation
%E	a floating point number in scientific notation
%%	the % symbol

- Commands
  - printf(“ … “); - prints output in parenthesis
  - scanf(“ … “); - reads input
  - getchar(); - gets next input character
  - putchar(x); - prints contents of variable as a character
  - ndigit[x]; - **array**: name[total elements];

- Loops
  - main()
    {
    ...
    }
  - for (...)
  - while (...) {
    …
    }
  - do
    …
    while (...);
  - ***loops need bodies***

- Conditionals: (evaluated in order)
  - if (condition)
    statement
  - else if (condition)
    statement
  - else (condition)
    statement

- Functions:
    - return-type function-name(argument declarations)
      {
          declarations
          statements
      }
        - **functions always need a return statement (need to return something to main())**

,

- ***statement*** - expression followed by a semicolon
- ***semicolon*** - statement terminator
- ***comma*** - pair of expressions separated are evaluated from left to right
    - commas separating function arguments, variables in the declaration are **NOT** comma operators; don't guarantee left to right evaluation
- ***character string/string constant (double quotations)*** - " … "
- ***character constant (single quotations)*** - ' … '
    - represents integer value equal to the numerical value of the character in the machine's character set (ASCII value)
    - escape sequences are also legal to use
- ***comment*** - /* … */
- ***definition*** - place where the variable is created or is assigned storage
- ***declaration*** - announces properties of variables, consists of type name and list of variables **(NO STORAGE ALLOCATED)**
    - type variable(s);
- ***assignment statements*** - sets variables to initial
- ***return statement*** - return *expression*;
- **EOF** - end of file
- ***external variables*** - variables that can be accessed by name by any function
    - extern type name;
    - **must be defined once outside of any function and declared in each function that wants access**
    - can be omitted if the definition occurs in the source file before its use in a function (place extern definitions at the beginning of the source file)
- ***enumeration constant*** - list of constant integers that are named
    - enum name { … }
- ***block (compound statement)*** - { … }, group of declarations and statements
- ***break statement*** - provides early exit from loops and switch statements
- ***continue statement*** - causes next iteration of enclosing loop to being
- ***goto statement*** - jumps to label
- ***label*** - same form as a variable name; followed by a colon; the scope is the entire function

- Relational operators (1 = true, 0 = false)
    - **>** - greater than
    - **>=** - greater than or equal to
    - **<** - less than
    - **<=** - less than or equal to
        - above all same preference
    - **!=** - not equal to
    - **=** - equal to
        - ***Precedence of != > =, order of operations will to != first***
    -
    - **==** - is equal to

- Logical operators (evaluated from left to right) (1 = true, 0 = false)
    - **&&** - AND
        - ***PRECEDENCE HIGHER THAN OR***
    - **| |** - OR
    - ! - NOT

- Binary operators:
    - **&** - AND operator
    - **|** - OR operator
    - **^** - XOR operator
    - **~** - ones complement operator (unary, flips bits)
    - **<<** - left shift
    - **>>** - right shift

- Arithmetic operators:
    - **+** - addition
    - **-** - subtraction
    - ***** - multiplication
    - **/** - division
    - **%** - modulus operator
        - **cannot be applied to float or double types**
    - **++** - increment by one
        - ++x increments before the value is used
        - x++ increments after the value is used
    - **--** - decrement by one

- Bitwise operators:
    - **&** - bitwise AND
    - **|** - bitwise inclusive OR
    - **^** - bitwise exclusive OR
    - **<<** - left shift
    -

    - **>>** - right shift
        - unsigned - fills vacated bits with zero
        - signed - fill with sign bits OR 0 bits (varies by machine)
    - **-** - one's complement (unary)

- Assignment operators: (*op*=, $expr_1$ *op* = $expr_2$ -> $expr_1$ = ($expr_1$) *op* ($expr_2$))
    - +=
    - -=
    - *=
    - /=
    - %=
    - <<=
    - >>=
    - &=
    - ^=
    - |=

TABLE 2-1. PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

OPERATORS	ASSOCIATIVITY
`() [] -> .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

Unary +, -, and * have higher precedence than the binary forms.

- Symbolic Constants
    - #define name replacement text
        - **NO SEMICOLON**
    - Include information about standard library - #include <library>
        - **NO SEMICOLON**

- **Variables and Arithmetic Expressions**

- Types, Operators, and Expressions: (Ch. 2)

- **Use lower case for variables**
- **Use upper case for symbolic constants**
- ***keywords*** - reserved, don't use as variable names
- ***constant expression*** - expression that involves only constants

- Data Types and Sizes
- Basic Types
    - ***char*** - single byte, can hold one character in the local character set
        - printable characters are always positive
    - ***int*** - an integer, typically reflects the natural size of integers on the host machine, either 16 bits or 32 bits
    - ***float*** - single-precision floating-point
    - ***double*** - double-precision floating-point
- Other types
    - ***long double*** - specifies extended-precision floating point
- Qualifiers
    - ***short*** - 16 bits
    - ***long*** - 32 bits
    - ***signed*** - values between -128 - 127
    - ***unsigned*** - always positive or zero, values between 0 - 255, obey laws of modulo 2^n
    - ***const*** - specifies that the value of the variable will not be changed
- Headers
    - **<math.h>**
    - **<stdio.h>**
        - **standard library**
    - **<limits.h>**
    - **<float.h>**
    - **<string.h>**
    - **<ctype.h>**
        - family of functions that provide tests and conversions that are independent of character set
- Implicit Arithmetic Conversion Rules (INFORMAL):
    - If either operand is **long double**, convert the other to **long double.**
    - Otherwise, if either operand is **double**, convert the other to **double**.
    - Otherwise, if either operand is **float**, convert the other to **float**.
    - Otherwise, convert **char** and **short** to **int**.
    - Then, if either operand is **long**, convert the other to **long**.

- Explicit type conversion:
    - can be forced in any expression with unary operator ***cast***
    - (type) expression

- Conditional expressions:

- $expr_1$? $expr_2$: $expr_3$
  - expr1 evaluated first
    - if true, expr 2 is evaluated and is the value of the expression
    - else, expr 3 is evaluated and is the value of the expression
  - **if expr 2 and expr 3 are different types, conversion rules apply**

- Switch Statement:
  - multiway decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly
  - switch (*expression*) {
    case *const-expr: statements*
    case *const-expr: statements*
    default: *statements*
    }
    - default takes place if no other case matches; optional and no action happens if there isnt one
  - break statement - causes an immediate exit from the switch statement

- minimal function - dummy () {} /* does nothing, returns nothing */
- void argument - takes no arguments
- many numerical functions return double type
- the name of the array is just a pointer to the first element of the array
- typecasting - (*type*) *expression*

Lecture Notes:

- Arrays and Strings:
  - array - homogeneous collection of elements
    - usually one dimension (vector)
    - can have two or more dimensions (2 = matrix, > 2 = tensor), treated as arrays of arrays
    - ordered (a[0] -> a[1] -> a[2] -> …)
    - can be initialized with initialization list (doesnt need explicit count)
      - vice versa
    - *variable*[] -> variable gives address of variable[0]
  - address of - *&variable*
  - sizeof(variable/type) - number of bytes used
    - gives size of array
    - gives size of pointer (*)
  - arrays and pointer are equivalent
- Pointers:
  - pointer - variable that holds memory address
    - points to location of object in memory
      - point to address at which they are assigned

                - can assign pointer to address of variable using address of operator
                - multiple pointers can point to same address
                - pointers have addresses too
            - not all pointers have address (NULL pointer)
            - NULL pointer = 0
                - ((void *)0)
                - 0
                - 0L
                - depends on compiler used
            - dereferencing operator - *
                - can be used to manipulate several variables via call by reference
    - pass by reference
        - allows returning multiple values
        - can pass large amounts of data quickly
    - pointer arithmetic
        - ++ - increments to next address
        - -- - decrements to previous address
        - + - can only add numeric value to pointer
        - - - if pointer is subtracted from another pointer, distance between addresses
        - pointers can be used with relational operators
        - pointer offset
        - cannot sum, divide, or multiply pointers
    - array pointers
        - harder to understand
        - arr[i] = *(arr + i),
        - can be written with pointers
        - global array in data area
        - dynamically declaring an array allocates it on the heap
    - strings as arrays
        - strings handled as arrays
        - string = pointer to array of chars
        - strings can be used as arrays
    - pointers to pointers
        - can be used to pass arrays of arrays (ex. list of strings (char **argv))
    - function pointer
        - pointers to executable code in memory
        - dereferencing function pointer yields function
        - function pointer needs parentheses
- <u>Sorting:</u>
    - sorting - act of putting things in defined order
    - dictionaries - lexicographical order (alphabetical)
    - numbers can be sorted in natural order or reverse order

    - total and partial ordering
    - sorting allows assumptions for ordering
    - fundamental operation
    - algorithm complexity - more difference than faster computer
- Stacks and Queues:
    - arrays - random access (any order)
    - linked lists - sequential access (particular order)
    - stacks
        - last in first out order
        - has set capacity
            - can be increased with realloc
        - implemented as arrays, but can be implemented differently as long as semantics are preserved (dynamic stack)
    - queues
        - abstract data type (ADT)
        - first in first out order
    - priority queue
        - like a queue, but elements have priority associated with them
            - higher priority dequeued before lower priority
            - if same priority, position of the element is used
- Dynamic Memory Allocation
    - dynamic memory - memory allocated at run time
        - allocated from the heap
        - dynamic memory allocation
            - memory is allocated on the fly during run time
            - calculated and allocates memory during run time
    - compile time allocation
        - memory for variables is allocated by compiler
        - requires exact size and type of storage
    - stack space is limited
    - functions to allocate memory - malloc, calloc, realloc
        - dynamically allocate memory and return pointer to it
        - MUST BE FREED USING FREE()
            - not doing so causes memory leak
            - depletes system resources
    - heap
        - large region of unmanaged, anonymous memory
        - limited by computer's limitations
        - slower to read/write from due to pointers
            - can be used to access variables in heap
        - memory fragmentation possible as blocks are allocated/deallocated
    - malloc
        - returns pointer to bytes of memory allocated on heap
        - memory may contain junk data

        - doesnt check for size overflow in arithmetic operations
    - calloc
        - returns pointer to *number of objects * size* bytes of memory
            - each byte initialized to 0
        - slower than malloc
        - contents of allocated memory are known (0)
    - realloc
        - reallocated pointer to new point at *size* bytes of memory
            - deallocates old object and returns new pointer of uninitialized memory
        - if size is greater than size of original memory, original block is retained but extra memory is uninitialized
        - if size is less than size of original memory, beginning part of original block is retained
    - free
        - deallocates memory space pointer to by pointer
        - can cause segmentation faults/core dumps if program tries to access restricted memory
        - set freed pointer to NULL to mitigate after-use vulnerability
    - valgrind
        - collection of dynamic analysis tools
        - memcheck can detect
            - use of uninitialized memory
            - read/write memory after free
            - read/write off end of allocated blocks
            - read/write on inappropriate areas of stack
            - memory leaks
        - --leak-check=full
        - --show-leak-kinds=all
    - static analyzer
        - analyze source code before running
        - code compared to set/sets of coding rules
        - surface level only (can't check for proper function)
    - dynamic analyzer
        - track down errors during execution
        - can check for proper function
        - only analyze during execution
    - infer
        - static analysis tool for debugging
        - checks for
            - null pointer exceptions
            - resource leaks
            - race conditions
            - missing lock guards

- Recursion:
    - recursion - natural, not inherently inefficient
    - USE WHERE IT MAKES SENSE

- Make
    - make program
        - utility on most unix systems that automatically builds executable programs and libraries from source code
        - has several derivatives
        - cmake - produces Makefile for unix systems
        - has command line flags/options to select from
        - NEED MAKEFILE
    - makefile
        - plaintext file that contains instructions for make
            - has syntax, like a script
        - usually resides in same directory as source code
        - composed of rules
            - associated with a target, dependencies, and set of commands
            - *target*: *dependencies*
            - <TAB> *commands*
        - target
            - name of rule
            - *make target name*
            - usually name of file that is generated via execution
            - phony target
                - target that doesnt produce a file with the same name
                - cleaning directory
                - debugging
                - running all appropriate targets
            - flags/options
                - -C *directory*, --directory=<dir_name>
                    - changes directory before looking/running makefiles
                - -d
                    - print debug info and processing info
                - -f <file_name>, -- file=<file_name>, --makefile=<file_name>
                    - specifies file to be read as makefile
                - -I <dir_name>, --include-dir=<dir_name>
                    - specifies directory to search in for makefiles
                - --warn-undefined-variables
                    - warns about referencing undefined variables
        - variables
            - = - lazy assignment

        - recursively expanded variables
        - contents of assignment stored as is
        - make waits to expand variable references until usage
    - := - immediate assignment
        - simply expanded variables
        - behave like C variables
        - assignment is evaluated and result assigned to variable
        - if assignment is variable reference, reference is expanded before assignment
    - ?= - conditional assignment
        - behaves like =, except assignment only occurs if variable hasnt been assigned yet
    - += - conceatenation
        - behaves like = if variable hasnt been defined
        - adds extra text to defined variable separated by space
        - useful for adding debug flags
    - to use values, surround in parentheses/curly brackets and prepend dollar sign
    - used to factor makefiles to make maintenance easier
    - CC - C compiler
    - CFLAGS - list of compiler flags
    - OBJ - list of object files (.o)
    - SRC - list of source files (.c)
- dependency
    - target or file name
    - if rule has dependency that has been modified or target doest exist, make tries to fill dependency by executing rule with dependency name
    - if dependency hasnt been modified or target already exists, make doesnt execute rule to make dependency
- topological ordering
    - ensures linear ordering or dependencies such that dependencies will be compiled before their targets
- command
    - action to be executed
    - can use shell scripting commands generally (bash)
    - rule can have more than one command, each on own line
    - must have tab character in front of it
- shell function
    - communicates outside of make
    - performs command expansion
        - takes shell command and evals to output
    - newlines are converted to spaces
- wildcard function

                - can be used in rules as * operator, and is expanded by the shell
                - if used for variable assignment, wildcard expansion doesnt occur unless wildcard function is specified
            - patsubst function
                - formatted as *$(patsubst pattern, replacement, text)*
                - finds whitespace separated words in text that match pattern and replaces with replacement
            - % operator
                - pattern match placeholder
                - pattern matching
                    - scalability (apply one rule to many files)
                    - doesnt match all filenames
                    - executes if theres a dependency that needs to be created
            - recursion in make
                - useful for separate makefiles in subsystems
                - using make as command ($(MAKE))
            - including makefiles
                - include makefile
                - various programs that need common set of variable definitions
- <u>Data Compression:</u>
    - two lossless compression algorithms
        - huffman
        - LZ78
        - cant compress messages of uniform randomness
    - entropy
        - measure of uncertainty of occurrence of an event
        - $H = -\sum_{i=1}^{n} p_i \log_2(p_i)$
        - p_i - the probability of event i

- <u>Linked Lists:</u>
    - linked data structure
        - singly linked - each node has data field and pointer to next node in list
            - circular singly linked list - last node points back to tail
        - doubly linked - each node has data field and pointer to next and previous nodes in list
            - allows transversal in two directions
            - less memory efficient than singly linked list
            - sentinel node implementation
                - designated dummy nodes to mark end of list
                - in head and tail
            - circular doubly linked list - head points back to tail, vice versa
        - last node points to terminator (NULL pointer)
        - linked structures

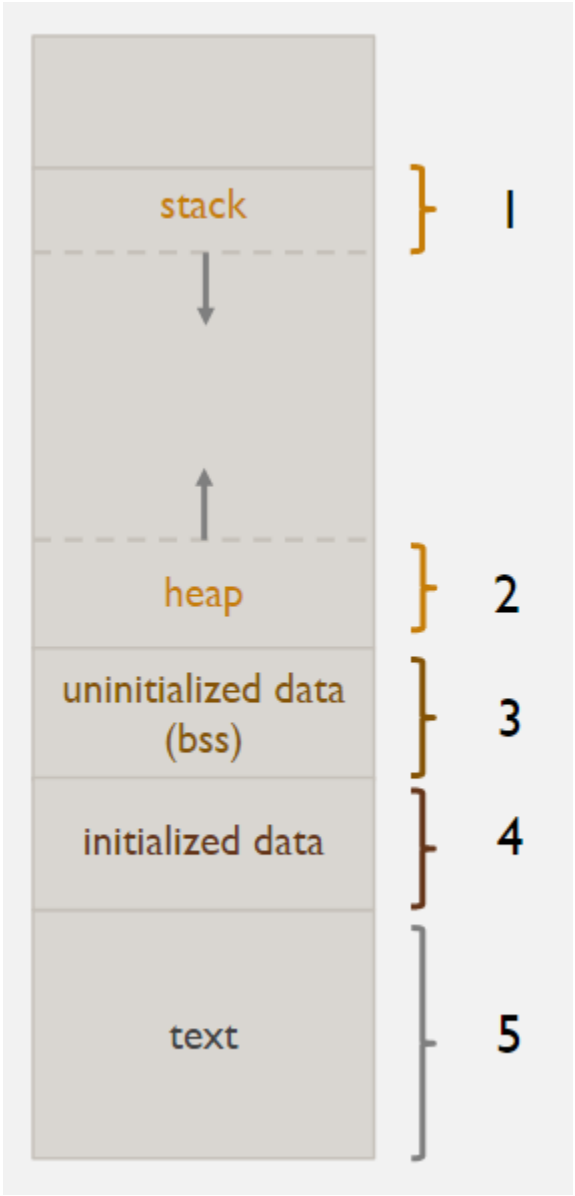
                - linked lists
                - trees
                - tries
                - graphs
                - sparse matrices
            - advantages
                - no fixed memory allocation
                - update address of pointer only to next node pointer
                - easy implementation of linear data structures (stacks, queues)
            - disadvantages
                - memory usage
                    - storing pointer requires memory
                    - arrays friendlier to processor cache
                    - slightly less efficient than arrays
                - traversal
                    - cannot randomly access elements (must traverse all elements to the one with desired access)
                    - reverse transversal difficult
                        - doubly linked list easy but extra memory usage

- Trees:
    - tree - type of directed acyclic graph, typically of nodes
        - exactly one path between two nodes
        - can be NULL
        - can be node pointing to two trees
    - node - smallest entity in tree
        - typically has a value
        - binary tree
            - node has up to two children
        - k-ary tree
            - node has up to k children
    - root - starting point of tree, if NULL tree is empty
    - parent - node that points to child nodes
    - child - node connected to parent, can be root of subtree
    - subtree - tree rooted at some node, must contain descendants of said node
        - proper subtree cant have same root as entire tree
    - leaf - node with no children, children are NULL
    - traversal
        - preorder (root - > left -> right)
        - inorder (left -> root -> right)
        - postorder (left -> right -> root)
        - level order (prints levels up -> down, left -> right)
    - binary search tree
        - ordered tree
            - nodes dont need order, but more useful if ordered

        - keys less than node value go under left subtree
        - greater than node value go under right subtree
        - duplicates generally ignored
- Files:
    - long term info storage
        - large amounts of data - changed 6 orders of magnitude
        - information stored must survive termination of process using it
        - memory doesnt survive powering computer off
        - files accessed with names, memory by addresses
    - file names
        - base names and extensions
            - extension separate entity in some OSes
            - unix uses extension by convention
    - file access
        - sequential access
            - read bytes from beginning
            - cannot jump around
            - magnetic tape
        - random access
            - read bytes in any order
            - essential for databases
            - read
                - move file marker then read or vice versa
    - operations
        - create
        - delete
        - open
        - close
        - read
        - write
        - append
        - seek
        - get attributes
        - set attributes
        - rename
    - directories
        - naming better than numbers
        - folders
        - makes it easier to find files
        - operations
            - create
            - delete
            - opendir
            - closedir

                - readdir
                - rename
                - link
                - unlink
- Cryptography:
    - ciphertext - information available
    - plaintext - known only to people with keys
    - keys - shared secret
    - algorithms -publicly known
    - one time pad
        - unbreakable code
        - use of truly random key as long as the message
    - algorithms
        - data encryption standard
            - 56 bit keys
            - same key to encrypt/decrypt
        - AES
            - 128 bit keys
            - adding one bit makes key twice as hard to guess
    - simon
        - family of lightweight block ciphers released by NSA
        - balanced feistel cipher with n bit word, block of 2n
        - no successful attacks known
- Debugging:
    - bug - error or flaw in program that produces unexpected or incorrect output
        - syntax errors
        - logical errors
        - semantic errors
    - debugging - process of identifying and fixing errors
    - assert
        - used to verify preconditions and postconditions (before and after execution)
        - checks can be turned off during compile time (-DNDEBUG flag)
        - argument is boolean expression (false prints error)
    - scan build
        - static analyzer
        - overrides CC environment variable to build with a fake compiler
            - fake compiler uses clang or gcc
            - static analyzer then executed to analyze
    - valgrind
        - invalid read - process tried to read outside available addresses
        - invalid write - process tried to write outside available addresses
    - lldb
        - higher performance debugger

        - can set breakpoints in program
        - step through program line by line
        - requires compilation with debugger flags
- Language Translators:
    - language translator - program that maps input language to output language
        - converting high level source to machine level language
        - provide diagnostics if programmer violates specifications of high level language
    - compilation process
        - pre processor
            - source code passed directly to
            - macroprocessor replaces with macro instructions
            - output stored in intermediate file to go to compiler (.i)
        - compiler
            - converts intermediate file to assembly code
                - lexical phase
                - break down source code into lexemes, generating sequence of tokens as units in grammar of language
                - tokens construct parse tree
                - syntax phase
                - interpretation phase
                - optimization
                - storage assignment
                - code generation
                - assembly phase
        - assembler
            - converts assembly language to machine language
            - one pass conversion
                - traverse through assembly once
                - symbols used before being defined must include an errata at end of object code (lets linker know to overwrite undefined symbol with respective definition)
            - two pass conversion
                - create mapping of symbols and values in first traversal
                - replace symbols with definitions in second traversal
        - linker
            - links .o file along with object files and libraries
            - ensures dependencies are resolved
            - merges everything into one executable file
    - loader
        - lays in OS
        - ensure program and libraries are placed in RAM to prepare for execution
        - functions

        - allocation - allocate memory for program
        - linking - resolve symbolic references between programs
        - relocation - fix all dependent locations and point them to new allocated space
        - loading - place machine code and data directly into processor
    - memory
        - 

        - stack - local variables, return addresses, arguments, return values
        - heap - dynamic memory allocation
        - uninitialized data - (bss), declared global and static variables initialized to 0
        - initialized data - holds global, static, constant, and extern variables that are already initialized
        - text - executable instructions

- Processes:
    - process - code, data, stack, program state
        - has own address space
        - cpu registers, program counter (location), stack pointer
        - only one process can be running in a CPU core at a given time
            - cpus with multiple cores can run multiple processes
    - address space - region of computers memory where program executes
        - instructions have own address
        - bytes of data have own address
        - protected from other programs accessing it
    - loader - can relocate instructions by address
    - base register - first byte of programs memory
    - memory hierarchy
        - registers
        - cache - small amount of fast, expensive memory
            - L1 - CPU chip
            - L2 - may be on or off CPU chip
            - L3 - off chip, SRAM
        - main memory
            - medium speed, medium price (DRAM)
        - disk
            - gigabytes of slow, cheap, non-volatile storage
        - managed by memory manager
    - memory management
        - components - OS, single process
        - goal - lay out in memory
        - no swapping or paging
    - fixed partitions - divide memory into fixed spaces assign process to free space
    - mechanisms - separate input queues for each partition
    - multiprogramming
        - memory need relocation and protection
        - OS cannot be certain about program location in memory
        - process memory must be kept separate
            - protects process from other processes changing its memory
            - protects process from modifying its memory in undesirable ways
    - cpu registers
        - base and limit
            - access limited to system mode
            - base contains start of process's memory partition
            - limit contains length of memory partition
        - address generation
            - physical address - location in actual memory, base + logical address
            - logical address - location from process POV

            - if larger than limit -> error
    - process creation
        - system initialization
        - execution of process creation call
    - process end
        - conditions voluntary/involuntary
            - voluntary
                - normal/error exit
            - involuntary
                - fatal error
                - killed by another process
    - process states
        - created
        - ready
        - running
        - blocked
        - exit
    - threads
        - allow single application to do many things at once
        - faster to create and destroy
        - overlap computation and i/o
- <u>Scripting:</u>
    - shell
        - acts as command interpreter
        - allows users to give commands to OS
            - interactively - prompt/command line
            - batched - script (file with sequence of commands)
    - commands
        - each line treated as single command
        - each line split into tokens/words by whitespace
        - first word on line is command ot execute, following words are arguments
        - aliases
            - interactive shells
            - replaced with what they are aliasing for before execution
        - builtins
            - bash provided commands
        - functions
            - sequence of commands that perform a task
            - bash functions can accept arguments
        - executables
            - specified and executed by file path
            - or by names if directory is in the $PATH environment variable
    - $PATH
        - colon delimited list of directory names

            - show where executables are located
        - /bin - for commands required by system for repairs and booting
        - /usr/bin - primary directory for executables on the system
        - current directory should not be in $PATH for security reasons
            - reason why executing own programas requires relative path
- script - sequence of commands in file (usually executable)
    - first line of BASH script should be interpreter directive
        - hashbang/shebang
        - indicates which interpreter to interpret script with
            - #!/bin/bash - interpret script with /bin/bash
- dotfiles
    - files prefixed by a dot in a home directory
    - .bashrc
        - script that is executed whenever bash is started interactively
            - commands that should be run every time a new interactive shell is started
            - customized shell prompts, aliases, etc.
    - .bash_profile
        - script that is executed only at the start of a new login shell
            - commands that should only be run once
            - modifying/exporting $PATH
- parameters/variables
    - used to store data
    - variable - kind of parameter
        - has a name
            - can consist only of letters, digits, and underscores and can start with a letter/underscore
        - = - assignment operator
            - cannot use spaces around it
        - can be strings, integers, or arrays
    - expansion
        - can expand parameters with $ - expansion operator
        - variable expansion substitutes the variable with its value
    - arithmetic operators
        - performed in ((...))
        - results expanded with $((...))
        - var++, var--
        - ++var,--var
        - +, -, *, /, %
        - <, <=, =, !=, >=, >
        - &, ^, |, <<, >>
        - +=, -=, *=, /=, %=
    - special parameters
        - not variables

- functions
  - sequences of commands
  - can have multiple in one script
  - allows local variables
- arrays
  - list of strings
  - arr=(...)
  - bash allows arrays created using explicit indices
    - gaps between indices - sparse array
- loops
  - while
  - until
  - for w in word
    - loop for each word of words, set w as each word
  - for ((...;...;...))
- file descriptors
  - stdin - 0
  - stdout - 1
  - stderr - 2
- pipes
  - connects stdout of process to stdin of another
  - | - pipe operator
  - can be chained together
  - programs can be written so they are chained together
    - UNIX software tool philosophy
- awk
  - simple parser of text