**size of data types ( finding it, using types as sizes, casting types)**

[**https://www.geeksforgeeks.org/program-find-data-type-user-input/**](https://www.geeksforgeeks.org/program-find-data-type-user-input/)

<https://www.geeksforgeeks.org/c-cpp-program-to-find-the-size-of-int-float-double-and-char/>

<https://www.geeksforgeeks.org/difference-strlen-sizeof-string-c-reviewed/>

<https://www.geeksforgeeks.org/data-types-in-c/>

<https://www.geeksforgeeks.org/type-conversion-c/>

<https://www.geeksforgeeks.org/structures-c/>

**C 'strings'**

<https://www.geeksforgeeks.org/strings-in-c-2/>

**type creation/aliasing: enums, structs, unions, typedef (size difference between structs and unions)**

Structures do not have shared location for its members so the size of structure is equal or greater than the sum of size of all the data members. A union does not have separate locations for each of its member so its size or equal to the size of largest member among all data members.

<https://www.geeksforgeeks.org/difference-structure-union-c/>

<https://jraleman.medium.com/c-programming-language-types-structures-and-enumeration-dc7849081eb5>

<https://www.geeksforgeeks.org/enumeration-enum-c/#:~:text=Enumeration%20(or%20enum)%20is%20a,easy%20to%20read%20and%20maintain.&text=The%20keyword%20'enum'%20is%20used,in%20C%20and%20C%2B%2B>.

<http://people.scs.carleton.ca/~mjhinek/W13/COMP2401/notes/Notes_Structs_Union_Enum.pdf>

<https://www.tutorialspoint.com/difference-between-structure-and-union-in-c-program#:~:text=As%20mentioned%20in%20definition%20Structure,member%20among%20all%20data%20members>.

<https://www.studytonight.com/c/typedef.php>

**Pointers**

**size** Pointers are blocks of memory (8 bytes on 64-bit machines) that reference memory addresses of any data type in C.

**void** <https://www.geeksforgeeks.org/void-pointer-c-cpp/>

**to structured types** https://www.studytonight.com/c/pointers-to-structure-in-c.php or f we have a pointer to structure, members are accessed using arrow ( -> )

**& and \* operators**:

“\*” Operator is used as pointer to a variable.

The & is a unary operator in C which returns the memory address of the passed operand.

**incrementing/addition** <https://www.geeksforgeeks.org/pointer-arithmetics-in-c-with-examples/>

**to functions** <https://www.geeksforgeeks.org/function-pointer-in-c/>

**local vs global scope (i.e. stack vs heap pointers)**

<https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/>

**malloc/free operation & use**

<https://www.studytonight.com/c/dynamic-memory-allocation-in-c.php>

https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

**normal operation**

**arguments**

[**https://www.geeksforgeeks.org/difference-between-malloc-and-calloc-with-examples/**](https://www.geeksforgeeks.org/difference-between-malloc-and-calloc-with-examples/)

**behavior on error**

[**https://stackoverflow.com/questions/35026910/malloc-error-checking-methods/35027099**](https://stackoverflow.com/questions/35026910/malloc-error-checking-methods/35027099)

**correct/incorrect use**

**- GCC ---**

**Headers: creation and user ( guard definition )**

Guard is a particular construct used to avoid the problem of *double inclusion* when dealing with the [include directive](https://en.wikipedia.org/wiki/Include_directive).

#ifndef ADD\_H

#define ADD\_H

// your declarations here

#endif

**Contents: definitions/macros/prototypes**

A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. There are two kinds of macros. They differ mostly in what they look like when they are used. Object-like macros resemble data objects when used, function-like macros resemble function calls.

[**https://www.linuxtopia.org/online\_books/an\_introduction\_to\_gcc/gccintro\_34.html**](https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_34.html)

[**https://www.geeksforgeeks.org/write-header-file-c/**](https://www.geeksforgeeks.org/write-header-file-c/)

**how to include and where to include**

**- File IO ---**

**file descriptor - type and function**

**Functions:**

**open**

**paramters and use**

**when it errors (and error discovery)**

**read/write**

**parameters and use**

**interpreting/reacting to retrun value**

**blocking/non-blocking IO ..**

**meaning**

**purpose**

**working with (IO loops)**

[**https://www.studytonight.com/c/file-input-output.php**](https://www.studytonight.com/c/file-input-output.php)

[**https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/**](https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/)

**opendir/readdir**

**parameters and use**

**characater and use of return value**

[**https://www.thegeekstuff.com/2012/06/c-directory/**](https://www.thegeekstuff.com/2012/06/c-directory/)

**i-nodes: parts / what they do / how they do it (in general)**

The inode (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory. Each inode stores the attributes and disk block locations of the object's data.

[**https://linoxide.com/linux-command/linux-inode/**](https://linoxide.com/linux-command/linux-inode/)

**Processes ---**

**Structure: parts and their function**

**Lifecycle**

**fork(): use / operation / actions:** [**https://www.geeksforgeeks.org/fork-system-call/**](https://www.geeksforgeeks.org/fork-system-call/)

**creates new process by duplicating the existing process**

**exec(): use / operation / actions:** [**https://www.geeksforgeeks.org/exec-family-of-functions-in-c/**](https://www.geeksforgeeks.org/exec-family-of-functions-in-c/)

**wait(): use / operation / actions:** [**https://www.geeksforgeeks.org/wait-system-call-c/**](https://www.geeksforgeeks.org/wait-system-call-c/)

**Error management and degenerate states**

**peculiarities of fork() and exec(): reference urls above**

**zombies and orphans:** [**https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/**](https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/)

**signals: operation / use / single handler construction:** [**https://www.geeksforgeeks.org/signals-c-language/**](https://www.geeksforgeeks.org/signals-c-language/)

**Exceptional computation**

**signals, exit and \_exit: use and differences:** [**https://www.geeksforgeeks.org/exit-vs-\_exit-c-cpp/#:~:text=In%20C%2C%20exit()%20terminates,after%20the%20exit()%20function.&text=The%20\_Exit()%20function%20in,execute%20functions%20registered%20with%20atexit**](https://www.geeksforgeeks.org/exit-vs-_exit-c-cpp/#:~:text=In%20C%2C%20exit()%20terminates,after%20the%20exit()%20function.&text=The%20_Exit()%20function%20in,execute%20functions%20registered%20with%20atexit)**.**

**- Threads ---**

**use, construction, differences from Processes:**

**create(), join(), wait()**

[**https://www.geeksforgeeks.org/thread-functions-in-c-c/**](https://www.geeksforgeeks.org/thread-functions-in-c-c/)

[**https://www.geeksforgeeks.org/difference-between-process-and-thread/**](https://www.geeksforgeeks.org/difference-between-process-and-thread/)

**- Synchronization ---**

**purpose and necessity:** Thread **synchronization** is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as a critical section

**mechanisms and primitives**

**identity, use and operation of mutexes**

[**https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/**](https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/)

**identity and general application of barriers and cond. Vars**

[**https://en.wikipedia.org/wiki/Barrier\_(computer\_science)**](https://en.wikipedia.org/wiki/Barrier_(computer_science))

[**https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/**](https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/)

**semaphores**

**differences and similarities from/to mutexes:** [**https://www.geeksforgeeks.org/mutex-vs-semaphore/**](https://www.geeksforgeeks.org/mutex-vs-semaphore/)

**as asynch. threadsafe mechanisms:**

[**https://stackoverflow.com/questions/44739176/semaphores-for-single-threaded-asynchronous-async-await-style-programming**](https://stackoverflow.com/questions/44739176/semaphores-for-single-threaded-asynchronous-async-await-style-programming)

**monitors: purpose and application:**

[**https://www.geeksforgeeks.org/monitors-in-process-synchronization/**](https://www.geeksforgeeks.org/monitors-in-process-synchronization/)

In concurrent programming (also known as parallel programming), a **monitor** is a **synchronization** construct that allows threads to have both mutual exclusion and the ability to wait (block) for a certain condition to become false.

**deadlock: cause / four conditions to cause / two rules for avoidance**

[**https://wiki.sei.cmu.edu/confluence/display/c/POS51-C.+Avoid+deadlock+with+POSIX+threads+by+locking+in+predefined+order#:~:text=Deadlock%20requires%20all%20four%20conditions,predefined%20order%20using%20POSIX%20threads**](https://wiki.sei.cmu.edu/confluence/display/c/POS51-C.+Avoid+deadlock+with+POSIX+threads+by+locking+in+predefined+order#:~:text=Deadlock%20requires%20all%20four%20conditions,predefined%20order%20using%20POSIX%20threads)**.**

[**https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/**](https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/)

[**https://www.codeguru.com/cpp/misc/misc/threadsprocesses/article.php/c15545/Deadlock-the-Problem-and-a-Solution.htm#:~:text=The%20common%20advice%20for%20avoiding,then%20you'll%20never%20deadlock**](https://www.codeguru.com/cpp/misc/misc/threadsprocesses/article.php/c15545/Deadlock-the-Problem-and-a-Solution.htm#:~:text=The%20common%20advice%20for%20avoiding,then%20you'll%20never%20deadlock)**.**

**- File permissions ---**

**permission types and range of application**

[**https://c-for-dummies.com/blog/?p=4101**](https://c-for-dummies.com/blog/?p=4101)

**chmod and chown, use and operation**

[**https://www.thegeekdiary.com/basic-chmod-command-examples-in-linux/**](https://www.thegeekdiary.com/basic-chmod-command-examples-in-linux/)

**mnemonic and octet representations:**

In computer assembler (or assembly) language, a **mnemonic** is an abbreviation for an operation. It's entered in the operation code field of each assembler program instruction. For example, on an Intel microprocessor, inc ("increase by one") is a **mnemonic**.

<https://www.geeksforgeeks.org/octal-numbers-c/>

**- File Ops ---**

**read/dump: cat, less, more, head, tail**

**compare/examine: diff, wc, od**

**similarities and differences / use and operation / properties**

**standard/default file descriptors, name and purpose, use of**

**- Libraries and Makefiles ---**

**library definition, costruction and use in compilation (gcc -c ..)**

**Makefile structure and constructions, target/requirement dependency resolving, conventions in use (all, clean, first directive)**

**- Sockets --- see Class 12-3-2020**

[**https://www.geeksforgeeks.org/socket-programming-cc/**](https://www.geeksforgeeks.org/socket-programming-cc/)

**construction and use of**

**function call sequence to build each type**

**common errors (bind on port in use, block on read)**

**blocking nature of accept (and common use of threads in networking)**

**- Networking --- SEE CS214.txt Class 12-10-2020**

[**https://www.gnu.org/software/libc/manual/html\_node/Ports.html#:~:text=A%20socket%20address%20in%20the,such%20as%20finger%20and%20telnet%20**](https://www.gnu.org/software/libc/manual/html_node/Ports.html#:~:text=A%20socket%20address%20in%20the,such%20as%20finger%20and%20telnet%20)**.**

**definition and operation of an vs the internet, host, machine, router and switch**

**port number and use of**

**IP addresses**

**use of**

**dotted octet representation**

**class-based and classless routing (CIDR)**

**ISO/OSI stack (obj-oriented design applied to networking)**

**identity of layers**

**basic operation/responsibilities of layers**