Commonly used methods in `std::string`	
`string` `( );` **`string`** `( const char * s );` **`string`** `( size_t n, char c );`	Subset of commonly used constructors for `string`: 1. No-argument; 2: C string→C++; 3. String with `n` copies of `c`.
`size_t size();`	Returns length of the string.
`string& insert(size_t pos1, const string& str);`	Insert string `str` at `pos1`.
`string& insert ( size_t pos1, const string& str, size_t pos2, size_t n );`	Insert `n` characters, starting at index `pos2`, of `str` at `pos1`.
`string& erase(size_t pos = 0, size_t n = npos);`	Erases a sequence of `n` characters starting at position `pos`.
`string substr(size_t pos=0, size_t n=npos) const;`	Returns a substring of `n` characters starting at index `pos`.
`size_t find(const string& str, size_t pos=0) const;`	Searches for first occurrence of `str` starting at index `pos`. Returns `string::npos` if str is not found.
`size_t rfind (const string& str, size_t pos=npos) const;`	Searches for last occurrence of `str` starting at index `pos`. Returns `string::npos` if `str` is not found.
`size_type find_first_of( CharT ch, size_type pos = 0 ) const;`	Finds the first character equal to one of the characters in the given character sequence.
`size_type find_first_not_of( CharT ch, size_type pos = 0 ) const;`	Finds the first character not equal to one of the characters in the given character sequence.
`istream& getline(istream& is, string& s, CharT delim = '\n')`	Read a line delimited by `delim` from `is` into `s` and returns `is`.
`int stoi(s)` `double stod(s)`	Convert `s` to `int` / `double`.
`std::to_string(int/double x)`	Convert x to string

Commonly used methods in `std::vector<T>`	
`vector(iterator first, iterator last);`	Constructs vector consisting of elements between [`first`, `last`)
`vector(size_type n, const T& value= T());`	Constructs a vector with `n` copies of `value`.
`erase(iterator first, iterator last);` `erase(iterator element);`	Removes elements in vector between [`first`, `last`) or element at `x`; e.g. `vec.erase(vec.begin() + x)`
`insert(iterator position,const T& x );`	Inserts value `x` at `position`. If `x` is `int` write as: `vec.insert(vec.begin() + x)`
`push_back( const T& x );`	Add `x` to end of the vector.
`size_t size();`	Returns number of elements in the vector.
`begin(); rbegin();`	Returns iterator to beginning or reverse beginning.
`end(); rend();`	Returns iterator to end or reverse-end.

Commonly used methods in `std::unordered_map <Key, Value>`	
`unordered_map()`	Creates an empty map. **Entries have: `first`, `second` for key,value**
`find(Key k)`	Iterator to element with key `k` if found. `end()` otherwise.
`Value& at(Key k) const;`	Returns value for key `k`. Throws exception if `k` not found
`operator[](Key k)`	Add/access entry in map for key `k`.
`size_t size();`	Returns number of elements in the map.
`begin(); rbegin();`	Returns iterator to beginning or reverse beginning.
`end(); rend();`	Returns iterator to end or reverse-end.

I/O streams and related API	
`std::ifstream inFile(const std::string& p)`	Create input stream to read text file at path `p`
`std::ofstream outFile(const std::string& p)`	Create an output stream to write to text file at path `p`
`std::istringstream is(const std::string& p)`	Create a string stream to read data from a string `p`.
`std::ostringstream os()`	Create a string to write data. Use `.str()` method to get string
`std::ifstream::eof()`	Returns true if at end-of-file (eof)
`std::istream_iterator<T>(std::istream& is);`	Create an iterator to read objects of type `T` from input stream `is` using `operator>>()`
`std::ostream_iterator<T> (std::ostream& os, const std::string& delim);`	Create an iterator to write objects of type `T` to output stream `os`

	delimited by `delim` `operator<<()`
`std::back_inserter(Container& c)`	Output iterator to append values to `c` via `push_back` method

Commonly used STL algorithms

`for_each (InputIterator first, InputIterator last, Function f);`	Applies function *f* to each of the elements in the range [`first`,`last`).
`copy ( InputIterator first, InputIterator last, OutputIterator result);`	Copies the elements in the range [`first`,`last`) into a range beginning at `result`. See: `std::back_inserter`.
`copy_if( InputIterator first, size_t n, OutputIterator result, UnaryPredicate pred);`	Similar to `copy` (above) but `copy_if` only copies elements for which `pred` returns `true`.
`unique_copy ( InputIterator first, InputIterator last, OutputIterator result);`	Copies the values of the elements in the range [`first`,`last`) to the range positions beginning at `result`, except for the duplicate consecutive elements, which are not copied
`InputIterator find ( InputIterator first, InputIterator last, const T& value );`	Returns an iterator to the first element in the range [`first`,`last`) that compares equal to `value`, or `last` if not found.
`ForwardIterator min_element ( ForwardIterator first, ForwardIterator last);`	Returns iterator to smallest value in range [`first`, `last`)
`ForwardIterator max_element (ForwardIterator first,ForwardIterator last);`	Returns iterator to largest value in range [`first`, `last`)
`void replace(ForwardIterator first, ForwardIterator last, T& oldVal, T& newVal)`	Replaces all elements with `oldVal` in the range [first, last) with `newVal`
`void sort(RandmomIt first, RandomIt last);` `void sort(RandmomIt first, RandomIt last, Compare comp);`	Sorts values in the range [`first`, `last`). Optionally, takes a binary comparator to compare 2 elements.

Commonly used OpenMP pragmas

`#pragma omp parallel if` (scalar-expression) **`num_threads`**(integer-expression) **`private`**(*list*) **`shared`**(*list*) **`default`**`(shared|none)` **`firstprivate`**(*list*) **`reduction`**`({+,-,*,&,|,^,&&,||}:` *list*) **`copyin`**(*list*) [where *list* is comma separated list of one or more identifiers]
#**`pragma omp for private`**(*list*) **`shared`**(*list*) **`default`**`(shared|none)` **`firstprivate`**(*list*) **`reduction`**`({+,-,*,&,|,^,&&,||}:` *list*) **`lastprivate`**(*list*) **`orderd nowait schedule`**(*sched*, [chunk-size]) [where *list* is comma separated list of one or more identifiers and *sched* can be: `static`, `dynamic`, `guided`, or `runtime`]
`#pragma omp sections private`(*list*) **`shared`**(*list*) **`default`**`(shared|none)` **`firstprivate`**(*list*) **`reduction`**`({+,-,*,&,|,^,&&,||}:` *list*) **`nowait`** [where *list* is comma separated list of one or more identifiers]
`#pragma omp section`
`#pragma omp critical`(identifier)
`#pragma omp atomic`

Commonly used Slurm commands

Command	***Description***	***Example usage***

	Slurm options:
	• `--nodes` : Number of compute nodes
	• `--tasks-per-node`: Num. of cores per node
	• `--mem` : Total memory for job
	• `--time` : Max runtime in `hh:mm::ss` format
		`$ srun -A PMIU0184 \  # Fixed account`
		`--nodes 2 \     # Two nodes`
		`--tasks-per-node 6 \ # 6 cores`
		`--mem 4gb \     # total 4 GB RAM`
		`--time 2:30:00 # 2.5 hours`

`sinteractive`	Start interactive job using above parameters	`$ sinteractive`
`srun`	Runs a given job in foreground.	`$ srun -A PMIU0184 --nodes 3 ./prog`
`sbatch`	Submit a batch job to run in background	`$ sbatch job.sh`
`scancel`	Cancel a submitted or running job	`$ scancel 489720`
`squeue`	List all queued jobs	`$ squeue -u $USER`

BOOST MPI Functions (`namespace mpi = boost::mpi`)

Function Signature	*Description*
`mpi::environment env(int& argc, char& *argv[])`	Starts MPI runtime. Used in `main`.
`mpi::communicator world`	Starts a communicator to send/recv messages to all processes.
`int world.size()`	Returns the number of processes in the communicator `world`.
`int world.rank()`	Returns the rank of the calling process in communicator `world`.
`void world.send(int drank, int dtag, const T& src)`	Sends value `val` tagged with `tag` to process `dest` in a blocking manner. `T` can be any primitive type, `std::string`, or `std::vector`.
`mpi::status world.recv(int srank, int stag, T& dest)`	Receive value `val` tagged with `tag` from process `srank` in a blocking manner. `T` can be any primitive type, `std::string`, or `std::vector`.
`mpi::status world.sendrecv(int drank, int dtag, cosnt T& src, int srank, int stag, T& dest)`	Returns the number of elements in a message using `status` and `datatype` in `count`.
`mpi::status world.sendrecv(int drank, int dtag, cosnt T& src, int srank, int stag, T& dest)`	Performs blocking send and receive calls simultaneously by sending `src` to `drank` process with `dtag` while receiving value from `srank` process with tag `stag` into `dest`.
`mpi::status world.probe(int src, int tag)`	Performs a blocking probe and returns `status` information about first pending message with `tag` from `src` process.
`std::optional<mpi::status> world.iprobe(int src, int tag)`	Performs a non-blocking probe and optionally returns status information about pending message with `tag` from `src` process.
`template<typename T> request world.isend(int dest, int tag, const T& val) const;`	Sends `val` tagged with `tag` to specified `dest` process in communicator `comm` in a non-blocking manner. The function returns a `request` to determine status of operation.
`template<typename T> request world.irecv(int dest, int tag, const T& val) const;`	Receives `val` tagged with `tag` to specified `dest` process in communicator `comm` in a non-blocking manner. The function returns a `request` to determine status of operation.
`optional<status> request.test();`	Returns the status object, if `request` is complete. Otherwise, returns an empty optional<>
`status request.wait();`	Blocks until the non-blocking operation identified by `request` completes and then updates `status`.
`void request.cancel();`	Cancel a pending communication, assuming it has not already completed.
`optional<status> world.iprobe(int source = any_source, int tag = any_tag) const;`	Performs a non-blocking probe for the first pending message with `tag` from `source` process and if a message exists, it sets flag to `true` and returns `status` information about the message.

MPI Collective Functions (`namespace mpi = boost::mpi`)

Function Signature	*Description*
`void world.barrier();`	Performs barrier synchronization between all the processes in `world`.
`void mpi::broadcast(const communicator& comm, T & value, int root);`	Broadcasts `value` from `root` process to all processes in `comm` (e.g., `world`)
`void reduce(const communicator& comm, const T& in, T& out, Op op, int target);`	Combines `in` value of all processes using operation `op` and places result in `out` in the `target` process in `comm`. See list of `Op` below.
`void scan(const communicator& comm, const T& in, T & out, Op op);`	Performs prefix scan of `in` values by applying operation `op` and places result in `out`. See list of `Op` below.
`void all_reduce(const communicator& comm, const T& in, T& out, Op op);`	Combines `in` value of all processes using operation `op` and places result in `out` at all of the processes in `comm`. See list of `Op` below.
`void gather(const communicator& comm, const T& in, std::vector<T> & out, int root);`	Collects in value(s) from all processes in `comm` into `out` at the `root` process.
`void scatter(const communicator& comm, const std::vector<T>& in, T& out, int root);`	Distributes `sendcount` elements of `senddatatype` from `source` process to each process in `comm`.
`void all_to_all(const communicator& comm, const std::vector<T>& in, int n, std::vector<T>& out);`	Each process sends `n` elements from `in` vector to every process in `comm`, including itself. The values are gathered in `out`.
List of valid `Op`: `mpi::minimum`, `mpi::maximum`, `std::plus`, `std::minus`, `std::multiplies`, `std::divides`, `std::modulus`, `std::logical_and`, `std::logical_or`, `std::bit_and`, `std::bit_or`.	

Commonly used Linux shell commands

Command	***Description***	***Example usage***
`exit`	Log out of the Linux box	`$ exit`

`cd`	Change directory	`$ cd /usr/X11R6/bin`
		`$ cd ..`

`pwd`	Show present working directory	`$ pwd`

`ls`	List files.	`$ ls -l`
		`$ ls -l *.s`

`mkdir`	Make new directory	`$ mkdir csa-470`
`rmdir`	Remove empty directory	`$ rmdir csa-570`

`cp`	Copy file or files. You can copy entire directories recursively as well.	`$ cp a.s bak.s`
		`$ cp -r a?b*.s subDir`

`scp`	Copy files from local machine to/from remote machine.	`$ scp a.txt user@host.edu:remoteDir`
		`$ scp user@host.edu:remoteDir/a.txt localDir`

`mv`	Move file or files. You can move directories as well.	`$ mv ../a.s .`

`rm`	Remove files and directories.	`$ rm a.s`
		`$ rm -rf directory`

`cat`	Print contents of file on console	`$ cat hello.java`
`ps`	Process list	`$ ps -fe`

`grep`	Print lines that match a given regular expression	`$ grep "static" hello.java`
		`$ ps -fe | grep "ra?d*"`

`kill`	Stop a specific process	`$ kill -9 1234`
`g++`	Runs the GNU C++ compiler program(s)	`$ g++ -std=c++17 -g -Wall one.cpp -o one`
`diff`	Print difference between 2 files, if any.	`$ diff a.txt b.txt`
`chmod`	Change file permissions	`$ chmod u+rw-x,og+r-wx test.txt`

Useful Formulas

$$\mu = \sum_{1}^{n} t_i/n \quad \textbf{and} \quad \sigma = \sqrt{\left(\sum_{1}^{n}(t_i-\mu)^2\right)/n}$$

$$95\%\ CI = (2.776\, s)/\sqrt{n}$$

S = Ts / Tp E = S / p To = pTp – Ts