-<https://www.geeksforgeeks.org/c-magicians-stl-algorithms/>

https://www.geeksforgeeks.org/getline-[string](https://www.geeksforgeeks.org/getline-string-c/)-c/

<https://www.codementor.io/@satwikkansal/stl-cheatsheet-for-competitive-programming-arrlk3rqn>

see modifier in <https://www.geeksforgeeks.org/strings-library-in-cpp-stl/>

and <https://www.geeksforgeeks.org/vector-in-cpp-stl/>

**1. vector<int> vect2=vect1;**

* **it = std::find (vec.begin(), vec.end(), ser);**

**if (it != vec.end())**

**auto i=find(arr,arr+n,x);**

**if(i!=arr+n)**

**return i-arr;**

**else return -1;**

**4.// Declaring the sequence to be searched into**

vector<int> v1 = { 1, 2, 3, 4, 5, 6, 7 };

// Declaring the subsequence to be searched for

vector<int> v2 = { 3, 4, 5 };

// Declaring an iterator for storing the returning pointer

vector<int>::iterator i1;

// Using std::search and storing the result in

// iterator i1

i1 = std::search(v1.begin(), v1.end(), v2.begin(), v2.end());

// checking if iterator i1 contains end pointer of v1 or not

if (i1 != v1.end()) {

cout << "vector2 is present at index " << (i1 - v1.begin());

} else {

cout << "vector2 is not present in vector1";

}

return 0;

}

Output:

**vector2 is present at index 2**

**5. int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };**

int n = sizeof(arr) / sizeof(arr[0]);

/\*Here we take two parameters, the beginning of the

array and the length n upto which we want the array to

be sorted\*/

**sort(arr, arr + n);**

**vector<int> vect(arr,arr+n);**

6.

**-set-sorted+unique values,-element access is logn(access mean insertion and deletion)**

**-map-<key,value>**

**-difference b/w**

pair<int,int> p[5]

Here, p is defined as an array of pairs which stores integer pairs (like points in a 2D plane).

The **pairs are not sorted**, they are stored in the order you put it in.

**Pair has complexities just like vector.**

**It allows duplicates**.

**Search is O(N)** . We can sort it accordingly to improve the search.

**Element access is O(1)** .

**-map< int,int > m (m[key]=value)**

map <key,value>

Here, our defined **map ‘m’ contains key-value pairs with unique keys**. It works like a Hash Table (not exactly) where each key is uniquely associated with a value.

Because of the design, **it doesn’t allow ‘key’ duplicates**.

Elements are sorted on the basis of key values.

Search is O(logn)

Element access is also O(logn)

The reason for logarithmic complexity is because maps are usually implemented as red-black trees.

-**Priority queue takes logn time for insert,**

**-Priority queue of pairs(can be treated as ordered pair ) is better option in some cases instead of map/set becoz**

**Pq(max,min) can be created based on first value**

**It doesn’t need to be unique(or multiset aur multimap)**

**priority\_queue<int> gq //increasing**

*priority\_queue <int, vector<int>, greater<int>> g = gq;  //decreasing*

| map | unordered\_map

---------------------------------------------------------

Ordering | increasing order | no ordering

| (by default) |

Implementation | Self balancing BST | Hash Table

| like [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/) |

search time | log(n) | O(1) -> Average

| | O(n) -> Worst Case

Insertion time | log(n) + Rebalance | Same as search

Deletion time | log(n) + Rebalance | Same as search

**7.add greater<int> for decreasing order**

**8. Heap sort** is an in-place **algorithm**. **Time Complexity**: **Time complexity** of heapify is O(Logn). **Time complexity** of createAndBuildHeap()/make\_heap() is O(n) and overall **time complexity** of **Heap Sort** is O(nLogn)

**Array of n values will have heap of int size n/2**

**9/EMPLACE VS INSERT VS PUSH\_BACK**

primitive data types, it does not matter which one we use. But for objects, use of emplace() is preferred for efficiency reasons.

**10.**

every key is mapped with default value zero when the map is declared

**11.lower bound and upper bound**

**Takes o(logn) on sorted vector/array using binary search**

**Gives element <=x (lower bound) (output 6 for bellow)//** 5 5 5 6 6 6 7 7

**If 6 was not present it will point to just greater element to 6**

**If not found pointes to last**

**And > x (upperbound) (output 7 for below )//** 5 5 5 6 6 6 7 7

**here it will always point to just greater**

**If not found pointes to last**

**sort(v.begin(), v.end()); // 5 5 5 6 6 6 7 7**

**vector<int>::iterator lower, upper;**

**lower = lower\_bound(v.begin(), v.end(), 6);**

**upper = upper\_bound(v.begin(), v.end(), 6);**

**cout << "lower\_bound for 6 at position "**

**<< (lower - v.begin() + 1) << '\n';**

**cout << "upper\_bound for 6 at position "**

**<< (upper - v.begin() + 1) << '\n';**

**All the function implemented in c++ are in the range**

**[first,last) so arr+n and v.end() is never included**

**lower\_bound for 6 at position 4**

**upper\_bound for 6 at position 7**

**count of 6 is cout<<upper-lower**

**or**

**string str = "geeksforgeeks";**

**cout << "Number of times 'e' appears : "**

**<< count(str.begin(), str.end(), 'e');**

**Number of times 'e' appears : 4**

**11. //\*g1.end() has garbage value but actually last element is \*g1.end()-1 but \*g1.begin() has a value of first element ie is a[0];**

**vector<int> v={5 ,5, 5, 6 ,6 ,6 ,7 ,1};**

**after sort(v.begin(), v.end()-1);**

**output is // 5 5 5 6 6 6 7 1**

**in all functions in c++ last value is never included [v.begin(),v.end())**

**13.**

**Heap implementation in STL(refer abdul bari)(not used to much instead use priority q)**

**Priority q is implemented using the below functions**

**Priority\_queue(v.begin(),v.end()) takes o(n) time**

**Where as set and map takes O(nlogn)**

**// using pop\_heap() to delete maximum element (send to last place in vector)**

**pop\_heap(v1.begin(), v1.end());**

**//deleted from the vector from last**

**v1.pop\_back();**

**v1.push\_back(35);**

**// using push\_heap() to reorder elements (heapify)**

**push\_heap(v1.begin(), v1.end());**

* **Logn<rootn<nlogn**
* **max({A,B,C,D});**

**max can be used as well**

* **Giving output as float,int etc**

**Cout<<(int) tgamma(5)/11;**

* **Using string stringstream**

**We can take input/output from a string similar to file in a variable**

|  |  |  |
| --- | --- | --- |
| **Container Class** | **Iterator Type** | **Container Category** |
| vector | random access | sequential |
| deque | random access |
| list | bidirectional |
| set | bidirectional | associative |
| multiset | bidirectional |
| map | bidirectional |
| multimap | bidirectional |
| stack | none | adaptor |
| queue | none |
| priority queue | none |

**Forward\_list use forward iterators ie they cannot decremented**

**Bidirectional cannot use =, +/-,== ,can use ++,--**

**9.Heap memory is dynamically allocated and pointers can pe used to access it**

**at 14.02 in the below video**

**it has no link with heap data structure**

**<https://www.youtube.com/watch?v=_8-ht2AKyH4>**

**10.pointer arithmetic**

**\*(A+i)=A[i];**

**Pointer to 2D matrix**

**\*p[3]=**

**<https://www.youtube.com/watch?v=sHcnvZA2u88>**

**11.Iteration over map in which vector is mapped to integar /anyother**

**map<int, vector<int> > diagonalPrint;**

**diagonalPrintUtil(root, 0, diagonalPrint);**

**cout << "Diagonal Traversal of binary tree: \n";**

**for (auto it :diagonalPrint)**

**{**

**vector<int> v=it.second;**

**for(auto it:v)**

**cout<<it<<" ";**

**cout<<endl;**

**}**

**}**

**Values are mapped like**    diagonalPrint[d].push\_back(root->data);

This will take care of both insert and add values

(this syntax is not suitable for map)

Diagonalprint[d] means push value in dth row

* **After Break doesn’t increments i once in loops meaning there is diffence with normal**
* **substring**

**// Copy three characters of s1 (starting**

**// from position 1)**

**string r = s1.substr(1, 3);**

**substr(0,i) 0->i-1 and substr(0,0) has no output**

**// Take any string**

**string s = "dog:cat";**

**// Find position of ':' using find()**

**int pos = s.find(":");**

**// Copy substring after pos**

**string sub = s.substr(pos + 1);**

**// prints the result**

**cout << "String is: " << sub;**

**return 0;**

**}**

**Output:**

**String is: cat // prints the result**

**For dp vector initailisation**

* **int n = 3;**

**int m = 4;**

**/\***

**We create a 2D vector containing "n"**

**elements each having the value "vector<int> (m, 0)".**

**"vector<int> (m, 0)" means a vector having "m"**

**elements each of value "0".**

**Here these elements are vectors.**

**\*/**

**vector<vector<int>> vec (create globally)**

**vec.resize( n//rows , vector<int> (m//columns, -1));**

**mem[100][100];**

**memset(mem, -1, sizeof(mem[0][0])\*100\*100);**

**//vec.shrink\_to\_fit(); is also useful**

**for(int i = 0; i < n; i++)**

**{**

**for(int j = 0; j < m; j++)**

**{**

**cout << vec[i][j] << " ";**

**}**

**cout<< endl;**

**}**

**output**

0 0 0 0

0 0 0 0

0 0 0 0

**Or**

**// table dynamically**

**int\*\* dp;**

**dp = new int\*[n+1];**

**// loop to create the table dynamically**

**for (int i = 0; i < n+1; i++)**

**dp[i] = new int[W + 1];**

**f(int \*\*dp)**

**Dynamically allocated memory effectively has no scope. That is, it stays allocated until it is explicitly deallocated or until the program ends (and the operating system cleans it up, assuming your operating system does that).**

**The static variables are stored in the data segment of the memory.**

**Static Memory Allocation is done before program execution. Dynamic Memory Allocation is done during program execution. In static memory allocation, once the memory is allocated, the memory size can not change. ... In this memory allocation scheme, execution is faster than dynamic memory allocation.**

**The stack is faster because the access pattern makes it trivial to allocate and deallocate memory from it (a pointer/integer is simply incremented or decremented), while the heap has much more complex bookkeeping involved in an allocation or free.**

* **deque is used when elements are added in front and last frequently**

**the complexity is O(1) for first and last element and O(2) for second last/first**

* **ans.push\_back({0,0,a});**

**directly push values/variables in 2d vector**

* **case in which worst and avg is ok is a better solution then in which avg is good but worst is bad.**
* **There's no compact syntax in C language that would initialize all elements of an array to a single value, regardless of whether the array is multi-dimensional or single-dimensional. There's syntax that would set all elements to zero specifically (and it is = { 0 }), but for any other value it is impossible.**

**In online compliers**

* **max size of locally defined int array ~ 10^6.(10^5 to be safe)**
* **max size of globally defined int array ~ 10^8 as global variables (10^7 to be safe) are stored in heap**

**ideally it is 10^9 or to be 2^32**

* **1e9 means 10 to the power 9**
* **Find min**

**Int mn=INT\_MAX**

**For(int i=0;…..)**

**Mn=min(mn,x)**

* **Use memorize when Expected time complexity is not more then n\*m**
* **Find string in a string using kmp**

**size\_t found = str.find(str1);**

**if (found != string::npos)**

**cout << "First occurrence is " << found << endl;**

* **String stream and getline**

**string S, T;**

**getline(cin, S); //getline(s,sizeofcharcters);**

**stringstream X(S);**

**while (getline(X, T, ' ')) {**

**cout << T << endl;**

**}**

* **String compare != or == operator**

**// returns a value < 0 (s1 is smaller then s2)**

**if((s1.compare(s2)) < 0)**

**cout << s1 << " is smaller than " << s2 << endl;**

**// returns 0(s1, is being comapared to itself)**

**if((s1.compare(s1)) == 0)**

**cout << s1 << " is equal to " << s1 << endl;**

**else**

**cout << "Strings didn't match ";**

**// Replaces 7 characters from 0th index by s2**

**s1.replace(0, 7, s2);**

**if 7<sizeof s1 then nothing happens**

**cout << s1 << endl;**

* **String⬄int/long/float**

**// The object has the value 12345 and stream**

**// it to the integer x**

**int x = 0;**

**geek >> x;**

**string s = "12345";**

**// object from the class stringstream**

**stringstream geek(s);**

**string str1 = to\_string(12.10);**

**// Converting integer to string**

**string str2 = to\_string(9999);**

* **We can use stringstring and ctype to extract/parse string**
* string s={"1134:00 AM 11:42 PM"};
* stringstream geek(s);
* vector<*char* >v;
* *char* ch;
* while(geek>>ch)
* {
* //    cout<<ch;
* if(isdigit(ch)!=0)
* v.push\_back(ch);
* }
* **1134001142**
* **Remove from string ( by a character multiple occurrences)**

**s.erase(remove(s.begin(),s.end(), 'A'), s.end());**

* **Erase string**

**remove 1 character from 0 value (including 0)s.erase(0,1);**

**0 to pos-1   str.erase(str.begin() + 4);**

**str.erase(str.begin() + 0, str.end() - 6);**

**Delete 4th pos**

**str.erase(1); delete after1**

**we use index when remove x value from till a place**

**else we use begin and end**

* **getlin(cin/stringstream,targetstring,dilimator(‘\n space’))**
* **use debug statements wisely**

**in dp**

**for(int i=0;i<n+1;i++)**

**{for(int j=0;j<m+1;j++)**

**cout<<dp[i][j]<<" ";**

**cout<<endl;}**

* **use output statements at every process point**
* **size\_t** or any unsigned type might be seen used as loop variable as loop variables are typically greater than or equal to 0.  
  **Note:** When we use a **size\_t** object, we have to make sure that in all the contexts it is used, including arithmetic, we want only non-negative values.
* **npos is a constant static member value with the *greatest possible* value for an element of type *size\_t*.**
* **To put it simply, think of npos as *no-position*. As a return value, it is usually used to indicate that no matches were found in the string. Thus, if it returns true, matches were found at no positions (i.e., no matches).**
* **Convert string to int**

**Int x=stoi(str);**

* **Convert a string to number (string[i] - '0')**
* **We can use memset for only 0 and -1**
* **Let 0 is mapped to vector then**

**M[0]->[1,2,3,4,5]**

**If we do m[0].push\_back(6)**

**M[0]->[1,2,3,4,5,6]**

* **Heap of pairs is gives max value based on first**
* **Pq will be sorted when we pop all elements**
* **Instead of make\_pair we can use pq.push{4,5}**
* **Vector can be equated**
* **Arr[s.top()] Is allowed**
* **Use vector carefully in case of big number of values or initialise it before**
* **A row can be directly accesed in 2d array/vector as A[0];**
* **Use index with count for eg**

**In stl erase(2,8)-> 2nd (includng) index se 8 place**

**And iterator for normal**

* string s1={"adi"};
* s1+="tya";
* cout<<s1;

**output =Aditya**

|  |  |  |
| --- | --- | --- |
| [islower()](https://www.geeksforgeeks.org/isupper-islower-application-c/) | This function identifies the lowercase alphabets. | Returns 0 if the passed argument is not a lowercase alphabet Returns nonzero value if the passed argument is a lowercase alphabet |

* **Never use f(a--) in fuction**
* **Memory for 10^6 array is 4 MB**

**In general we are given 256MB of space**

**256/4=64 arrays**

**Scan ‘9’, it’s a number, we push it to the stack. Stack now becomes |9’|**

**|5| .  
7) Scan ‘-‘, it’s an operator, pop two operands from stack, apply the – operator on operands, we get 5 – 9 which**

* **We donnot need {} after : in switch**
* **Stoi() /stod etc: The stoi() function takes a string as an argument and returns its value.:**

**(part of string is also supported)**

**Simple ascii values can be used to convert it to integar**

* **for safety always initialize your array/container with a value**

**don’t rely on default value**

**three types of custom comparator**

**using structure/operator**

* **struct Person {**

**int age;**

**float height;**

**// this will used to initialize the variables**

**// of the structure**

**Person(int age, float height)**

**: age(age), height(height)**

**{**

**}**

**};**

**// this is an strucure which implements the**

**// operator overloading**

**Imp point sign is opposite that which we want**

**struct CompareHeight {**

**bool operator()(Person const& p1, Person const& p2)**

**{**

**// return "true" if "p1" is ordered**

**// before "p2", for example:**

**return p1.height < p2.height;**

**}**

**};**

**int main()**

**{**

**priority\_queue<Person, vector<Person>, CompareHeight> Q;**

**2.Comparator functions Using structure**

**struct Activitiy**

**{**

**int start, finish;**

**};**

**// A utility function that is used for sorting**

**// activities according to finish time**

**bool activityCompare(Activitiy s1, Activitiy s2)**

**{**

**return (s1.finish < s2.finish);**

**}**

**3. Using comparator (vector of pairs)**

**// Driver function to sort the vector elements**

**// by second element of pairs in ascending order**

**The rule is: const applies to the thing left of it. If there is nothing on the left then it applies to the thing right of it.**

**bool sortbysec(pair<int,int> const &a, pair<int,int> const &b)**

**{**

**return (a.second < b.second);**

**}**

**Always return bool values**

* **use type cast in fractional values like fractional**

**[check](https://practice.geeksforgeeks.org/problems/fractional-knapsack-1587115620/1)**

**static bool cmp(struct Item a,struct Item b)**

**{**

**double r1=(double)a.value/a.weight;**

**double r2=(double)b.value/b.weight;**

**return r1>r2;**

**}**

**//always initailize double/float variable with 0.0**

**double ans=0.0;**

* **set<char>s (str, str+n-1); for char array**
* **Whenever need to take input use (especially long long )**

**ios\_base::sync\_with\_stdio(0), cin.tie(0), cout.tie(0);**

* **Execution speed of printf() and scanf() is more...  
  By writing ios\_base::sync\_with\_stdio(0);  
  cin and cout works as fast as printf and scanf,**

**This breaks the contact of c from c++, now c++ will only execute c++ functions making the c++ program run faster by 5-10%.**

* **You cannot rename/modify the hashmap key once added. Only way is to delete/remove the key and insert with new key and value pair. Reason : In hashmap internal implementation the Hashmap key modifier marked as final**
* **Use map.find() instead of t[] or count**
* **V.size() in 2d vector is number of rows**

**V[].size() is number of colums**

* **Remember push\_back is always done in a empty vector**

**If you initialize a vector with some size then 0 values will be made with that size**

**Values can be Overwrite in that case**

**Try to avoid v.clear()**

* **Vector can be directly equated**

**For kth smallest element**

* **If we use MaxHeap, we have O(k) space and O(nLogk) time. If we use MinHeap, we have O(n) space and O(n + klogn) time. If k is small such that k~logk, MinHeap gets reduced to ~O(n + logn) MaxHeap gets reduced to ~O(n) If k is of the order n,<=n: Min gives O(n + nlogn) Max gives O(nlogn) So always prefer max for kth smallest and min for kth largest. Side note: if it's explicitly given all elements are distinct, then and only then go for quickselect method.**
* **next\_permutation(a.begin(),a.end());**

**returns true if next permutation is greater then previous**

**and changes a to next lexicolgically greater**

**for eg for a =cba or vector 3 2 1 return will be zero**

**do{**

**cout<<s<<" ";**

**}while (next\_permutation(s.begin(),s.end()));**

**cout<<endl;**

* **Important point be careful with the order of n and m in string /dp question donot mix them**

**dp[m][n] ->** **dp.resize(m+1,vector<int>(n+1,-1));->**

* **func(s,t,m,n);**

**range based for loop (used when we donot need to use index)**

**If you don't want to change the items as well as want to *avoid* making copies, then auto const & is the correct choice:**

**for (auto const &x : vec)**

**Whoever suggests you to use auto & is wrong. Ignore them.**

**Here is recap:**

* **Choose auto x when you want to work with copies.**
* **Choose auto &x when you want to work with original items and may modify them.**
* **Choose auto const &x when you want to work with original items and will not modify them.**
* **For returning array use static or dynamic array (both has life time of whole program)**

**We can return a array given to us in function as it exist in the function which calls it**

**Count of number of given string in 2D character array**

**Backtrack complexity**

* + **To compute the time complexity, we can consider the searching path a DFS 4-way tree traversal with the depth equal to the length of searched string k - 1 (root's depth is 0), so searching each tree takes O(4^(k-1)) = O(2^2(k-1)) = O(2^2k). Since each node can be a root, there will be m\*n (dimension of the map) trees to search. Therefore, the general time complexity is O(m\*n\*2^2k). Since 2^2k grows much faster than m\*n, simply O(2^2k).**
    - **I<=n-1 == i<n**

**It means if we add = we add - 1**

* + - * **Memiozation always gives last values**
* **Donot use unordered\_map for simple counter (if you are using more then one use arrays ) unordered\_map can give tle**

**When using alpabets use arr[26]**

**For eg int a[256]={0}; is same as map accept it doesn’t contain functions**

* + **for recursion no of loops are equal to variables**

**sort is important to match the order/sequence of answer**

**this is used combinedly with unique**

**v.erase(unique(v.begin(),v.end()),v.end());**

**removes consecutive same elements**

**It returns an iterator to the element that follows**

**the last element not removed**

**(resize can be used then)**

* + - **It does not delete all the duplicate elements, but it removes duplicacy by just replacing those elements by the next element present in the sequence which is not duplicate to the current element being replaced. All the elements which are replaced are left in an unspecified state.(remove function also works similar)**
    - **bool IsOdd(int i)**

**{**

**return ((i % 2) == 1); }**

**can be used with remove\_if (vec2.begin(), vec2.end() , IsOdd);**

* + - * **bool Pred(char a, char b)**

**{**

**// Checking if both the arguments are same and equal**

**// to 'G' then only they are considered same**

**// and duplicates are removed**

**if (a == b && a == 'G') {**

**return 1;**

**} else {**

**return 0;**

**}**

**}**

**unique(s.begin(), s.end(), Pred);**

**in general sorting is done based on first value it is equal then it done bases of second**

* + - **vector<vector<*int*>> vec{ { 10, 121, 3 },**

**{ 10, 53, 62 },**

**{ 10, 53, 62 },**

**};**

* **base condition sometime changes to return 0 instead dp[n]=0**
* **merge in stl**

**int arr3[n+m];  
merge(arr1,arr1+n,arr2,arr2+m,arr3);**

* + **Brute Force approach finds all the possible solutions and selects desired solution per given the constraints.**
  + **- Dynamic Programming also uses Brute Force approach to find the OPTIMUM solution, either maximum or minimum.**
  + **- Backtracking also uses Brute Force approach but to find ALL the solutions.**
  + **- Solutions to the Backtracking problems can be represented as State-Space Tree.**
  + **- The constrained applied to find the solution is called Bounding function.**
  + **- Backtracking follows Depth-First Search method.**
  + **- Branch and Bound is also a Brute Force approach, which uses Breadth-First Search method.**
  + **for values we use {} for eg = vector<int> a={1,2,3};**

**for initialize/arrugument we use () vector<int>(N,-1)**

* + - **unordered\_set<string> ans;**

**vector<string>{ans.begin(), ans.end()};**

**conversion of set to string**

**vector<string>s={"are","you","there","you"};**

**unordered\_set<string>a(s.begin(),s.end());**

**s=vector<string>{a.begin(),a.end()};**

**output->there you are**

* **// for float or double**

**float f ;**

**//both are correct**

**// c++ calculate ans in int by default so this is necessary**

**f=(float)1/8;**

**f=1.0/8.0;**

**void f(int& a)**

**{**

**cout<<a;**

**}**

**int main()**

* **{**

**//f(5); un comment will give error**

**int b=5;**

**f(b);}**

**constant value cannot be passed tor refernce variables**

* **Using memset difficult with dynamic array**

**To be more simple, Memoization uses the top-down approach to solve the problem i.e. it begin with core(main) problem then breaks it into sub-problems and solve these sub-problems similarly. In this approach same sub-problem can occur multiple times and consume more CPU cycle, hence increase the time complexity**

**Put simply, a bottom-up algorithm "starts from the beginning," while a recursive algorithm often "starts from the end and works backwards."**

* **Char to int = num-‘0’**

**Int to char = num+’0’**

* **Is\_sorted (syntax like sort ) checks if a container is sorted aur not**
* **Check for duplicates in vector**

**if(unique(values.begin(),values.end())!=values.end())**

**then duplicate element is present**

* **struct Interval**

**{**

**int low, high;**

**};**

**// Structure to represent a node in Interval Search Tree**

**struct ITNode**

**{**

**Interval \*i; // 'i' could also be a normal variable**

**int max;**

**ITNode \*left, \*right;**

**};**

**Excess element like i.low**

* **for(int i=’a’;i<’z’;i++)**

**int a=5;**

**char ch='h';**

**string an;**

**an.push\_back('a'+a-1);**

**cout<<an;**

**->output is e**

* **stod and stof are same as stoi**
* **use continue instead of i++ in loop**
* **vector<vector<int> > ans(a,vector<int>(a,0));**

**is not same as vector<vector<int> > ans(a); this will only create column**

* **void relational\_operation(string s1, string s2)**

**{ s1=94 s2=91**

**string s3 = s1 + s2;**

**if(s1 != s2)**

**cout << s1 << " is not equal to " << s2 << endl;**

**if(s1 > s2)**

**cout << s1 << " is greater than " << s2 << endl;**

**else if(s1 < s2)**

**cout << s1 << " is smaller than " << s2 << endl;**

**if(s3 == s1 + s2)**

**cout << s3 << " is equal to " << s1 + s2 << endl;**

**}**

**String can compared to empty string as well like i==”” also vector can be compared**

* 94 is not equal to 91

94 is greater than 91

9491 is equal to 9491

And in set we can give direct value for eg to delete 4 we do set.erase(4);

* **from x to y there are x-y+1 numbers including x and y(only one end (any of 2) is included)**

**from x to y not including X and y there are x-y-1 numbers**

**stoi error when number is later for eg aojf98 and in out of range**

**[coversion diff number system to decimal](https://www.geeksforgeeks.org/stdstol-and-stdstoll-functions-in-c/)**

**[vice versa](https://www.geeksforgeeks.org/stdoct-stddec-and-stdhex-in-cpp/)**

**using streamstream ,strings,hex,oct,dec**

**like**

**int n=10;**

**string a;**

**stringstream s;**

**s<<hex<<n;**

**char x;**

**while(s>>x)**

**a.push\_back(x);**

**floating point manipulation can be done using fixed<<setprecision()<<n**

**and stringstream**

* **whereever it is asked to mod the you need to mod every its calculation is done**
* **for map and set we can use**

**iterator as well value as key to erase value**

**multiset is useful data structure to erase all value by key as it will delete multiple occurences**

**if key is of size n (string or vector) then time complexity is n\*logm**

**where m is size of map**

1. **unordered keys cannot use pair,vector, or any other complex container**
2. **string has a defined hash fuction in unordered\_map**

**log(n!)=log(n^n)=nlogn as**

* **lps of string “a” (1 size) is 0**
* **if(count(A.begin(), A.end(), '.')>1)**
* **-(negative range of int+1) =positive range of int**

**-(INT\_MAX+1)=INT\_MIN**

**2147483647 -2147483648**

* **Insert in vector and string**

**auto it = vec.insert(vec.begin(), 3);**

**The function returns an iterator which points to the newly inserted element.**

**str1.insert(0, str2);**

* **Remainder of 7.5/2.1 = 1.2**

**double x = 7.5, y = 2.1;**

**fmod of integer value is 0**

* **string decimalToBinary(int n)**

**{**

**//finding the binary form of the number and**

**//coneverting it to string. 64 means 64 bit binary**

**string s = bitset<64> (n).to\_string();**

**//Finding the first occurance of "1"**

**//to strip off the leading zeroes.**

**auto loc1 = s.find('1');**

**if(loc1 != string::npos)**

**return s.substr(loc1);**

**return "0";**

**}**

* **bit manuplation using string**

**string a={"1001"};**

**bitset<8>b(a);**

**cout<<b;**

**00001001**

* ***long* *long* a=LLONG\_MAX+1;**

**cout<<a;**

**-9223372036854775808 ->last -ve value same in +ve**

**a\*10 ->0;**

**lly if a= LLONG\_MIN then a\*10 -> 0**

**if we use unisigned *long* *long* a=LLONG\_MAX+1 then 0**

**long double can hold a lot values bit it wil cause precision errors**

**so use unsigned long long for max possible +ve value ie 10^19**

* **string ans(n + m, '0');**
* **v[v.size()-1].push\_back()-> add answer in last row**

**v.push\_back()->make a new row and add answer in that**

* **double pow(double x, double y);**

**cout<<(int)pow(2,32);**

* **print number of decimal places**

**long long a=1e10;**

**long long b=1e10;**

**cout<<to\_string(a+b).size(); ->11**

* **if one of the variable is long then ans will be long long as well**
* **but for safety keep both long long**

**long long a=1e10;**

**int b=1e9;**

**cout<<to\_string(a+b).size();->11**

* **we cannot print value of iterator ~~cout<<it~~**
* **advance(ptr, 3); can be used for non sequencial containers**
* **#define fac(a) tgamma(a+1)**
* **The number 0xAAAAAAAA is a 32 bit number with all even bits set as 1 and all odd bits as 0.**

**Set all even bits of x to 0 by doing bitwise and of x with 0xAAAAAAAA.**

**Set all even bits to 0** n & 0xaaaaaaaa

**And, for some more info,**

**The number 0x55555555 is a 32 bit number with all odd bits set as 1 and all even bits as 0.**

* **Always clear global variables in main function**
* **Use long long int when using mod**
* **Next(iterator)/prev(iterator) can be used in bidirectional iterator**
* **If u donnot know return for a case then add a condion before calling refer interleaving of string**
* **for(int &x:a){cin>>x;} ->input in vector of size pre defined**
* **iterators are usefull when we want the exact value in map/set**
* **Unordered set/map doesn’t insert values in order**
* **//we donnot . operator inside class (it is understood we are talking about root)**
* **int sz=adj->size(); gives the size of first element of adj**
* **Max function can also calculate max character**
* **Always write 0ll in long long**
* **To calculete double in divide use (double)a[i]/(double)a[i+1]**
* **Using tuple is useful in queue**

**const vector<tuple<int,int,int,int>> directions={{1, 0,8,8}, {-1, 0,8,8}, {0, 1,6,7}, {0, -1,3,4}};**

**int ans=0;**

**cout<<get<2>(directions[0]);//8**

**for (const auto& [dx, dy,d,f] : directions)**

**ans +=dx+dy+d+f;**

**//1 2 3 4 5**

* **RIght rotate (From mid) ->4 5 1 2 3**

**rotate(shifted.begin(), shifted.begin() + N / 2, shifted.end());**

* **LEft rotate <https://www.geeksforgeeks.org/rotate-in-cpp-stl/>**
* **A way to traverse in a container (for eg vector) without index is popping last element of the container on each iteratation**
* **int mid = (n - 1) / 2;**

**This handles both odd and even values of n**

* **Way to handle consicutive element**

**for (int i = 0; i < n; i++) {**

**if (s[i] == '0')**

**nofConsecutiveElement++;**

**else**

**nofConsecutiveElement = 0;**

**Condition when nofConsecutiveElement is equal to simething**

* **In custom cumparator pehle is down in pq**

**Pq array {25,5},{30,5.5},{23,5.6},{20,6},{33,6.1}**

33 6.1

20 6

23 5.6

30 5.5 sorted based on second (p1.height < p2.height)

25 5

* **smallest will be left side(for sorting) and down side(pq) when**

**We write p1.first<p2.first**

**Type cast**

* float ans = (float)a / (float)b;
* **we use map isntead of array when the val of given array cannot be contained in array as a index**