[Year]

C++ Cheat Sheet

QUICK GUIDE TO C++

RAGHAVENDRA BAZARI

Table of Contents

C++

Input & Output

Action	Code	Prerequisite
Output	cout<<"something is cooking"<<endl;	#include <iostream> using namespace std;
Input	string str; cin>>str;	#include <iostream> using namespace std;

Datatypes

Text

Character	Declaration	`char c = 'A';`	
	Size	`sizeof(c) = 1`	
	Convert to ASCII integer	`int ic = (int)c;`	
	Check if alphanumerical (a-z or A-Z or 0-9)	`isalnum(c)`	
	Convert lowercase	`tolower('A')`	
	Check if digit	`Isdigit()`	
	Convert to Base10 integer	`int ic = c - '0';`	
String	Declaration	`string s;`	
	Length	`s.size()`	

	Convert to Int	`/*`
		`stoi() can take upto three parameters, the second parameter is for starting index and third parameter is for base of input number.`
		`int stoi (const string&  str, size_t* index = 0, int base = 10);`
		`*/`
		`string s = "42345235";`
		`int i = stoi(s); // stol`	

	Append	`string firstName = "Harry ";`	

		string lastName = "Bhai";
		firstName.append(lastName);
		
		Or
		firstName + lastName
		
		// firstName:Harry Bhai	

	Tokenising	string line = "How are you?";
		
		// Vector of string to save tokens
		vector <string> tokens;
		
		// stringstream class check1
		stringstream check1(line);
		
		string intermediate;
		
		// Tokenizing w.r.t. space ' '
		while(getline(check1, intermediate, ' '))
		  tokens.push_back(intermediate);
			#include<vector>
			#include <sstream>

	String matching algorithm	https://en.wikipedia.org/wiki/Knuth–Morris–Pratt_algorithm Always use DP when two strings are given	

	Finding index in string S which matches word W.	https://leetcode.com/problems/wildcard-matching/submissions/ https://leetcode.com/problems/edit-distance/ **Parenthesis problem:-** 1.https://leetcode.com/problems/generate-parentheses 2.https://leetcode.com/problems/score-of-parentheses 3.https://leetcode.com/problems/valid-parentheses 4.https://leetcode.com/problems/valid-parentheses Easy 5.https://leetcode.com/problems/remove-outermost-parentheses Easy 6.https://leetcode.com/problems/different-ways-to-add-parentheses/ Medium 7.https://leetcode.com/problems/remove-invalid-parentheses Hard 8.https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses Medium 9.https://leetcode.com/problems/maximum-nesting-depth-of-the-parentheses Easy 10.https://leetcode.com/problems/longest-valid-parentheses/ Hard **Counting of substring based on some condition:-** 1.https://leetcode.com/problems/number-of-wonderful-substrings Medium 2.https://leetcode.com/problems/sum-of-beauty-of-all-substrings/ Medium 3.https://leetcode.com/problems/maximum-number-of-occurrences-of-a-substring Medium 4.https://leetcode.com/problems/number-of-wonderful-substrings Medium **Check types of string:-**	

		1.https://leetcode.com/problems/isomorphic-strings Easy 2.https://leetcode.com/problems/valid-anagram Easy 3. https://leetcode.com/problems/additive-number Medium 4.https://leetcode.com/problems/buddy-strings Easy 5.https://leetcode.com/problems/longest-happy-prefix Hard 6.https://leetcode.com/problems/increasing-decreasing-string Easy 7.https://leetcode.com/problems/check-if-a-string-can-break-another-string Medium 8.https://leetcode.com/problems/determine-if-two-strings-are-close Medium 9.https://leetcode.com/problems/check-if-two-string-arrays-are-equivalent Easy 10.https://leetcode.com/problems/check-if-word-equals-summation-of-two-words Easy 11.https://leetcode.com/problems/check-if-one-string-swap-can-make-strings-equal Easy **Palindromic string:-** 1.https://leetcode.com/problems/palindrome-partitioning Medium 2.https://leetcode.com/problems/palindrome-partitioning-ii Hard 3.https://leetcode.com/problems/valid-palindrome Easy 4.https://leetcode.com/problems/shortest-palindrome Hard 5.https://leetcode.com/problems/palindrome-pairs Hard 6.https://leetcode.com/problems/longest-palindrome Easy 7.https://leetcode.com/problems/longest-palindromic-subsequence Medium 8.https://leetcode.com/problems/find-the-closest-palindrome Hard 9.https://leetcode.com/problems/palindromic-substrings Medium 10.https://leetcode.com/problems/valid-palindrome-ii Easy 11.https://leetcode.com/problems/longest-chunked-palindrome-decomposition Hard 12.https://leetcode.com/problems/break-a-palindrome Medium 13. https://leetcode.com/problems/can-make-palindrome-from-substring Medium 14.https://leetcode.com/problems/palindrome-partitioning-iii Hard 15.https://leetcode.com/problems/minimum-insertion-steps-to-make-a-string-palindrome Hard 16.https://leetcode.com/problems/remove-palindromic-subsequences Easy	

		16.https://leetcode.com/problems/construct-k-palindrome-strings Medium 17.https://leetcode.com/problems/split-two-strings-to-make-palindrome Medium **Sorting on String:-** 1.https://leetcode.com/problems/sort-characters-by-frequency Medium 2.https://leetcode.com/problems/custom-sort-string **Longest and shortest kind of String Problem :-** https://leetcode.com/problems/longest-duplicate-substring Hard 2.https://leetcode.com/problems/longest-string-chain Medium 3.https://leetcode.com/problems/longest-common-subsequence Medium 4.https://leetcode.com/problems/longest-happy-string Medium 5.https://leetcode.com/problems/maximum-length-of-a-concatenated-string-with-unique-characters Medium 6.https://leetcode.com/problems/find-longest-awesome-substring Hard 7.https://leetcode.com/problems/largest-substring-between-two-equal-characters Easy 8.https://leetcode.com/problems/largest-odd-number-in-string Easy	

Numerics

Integer	Declaration	`int i;`	
	Size	`sizeof(int) <= sizeof(long)`	
	Minimum value and Maximum value	`INT_MIN //-2147483648` `INT_MAX //+2147483647`	`#include <bits/stdc++.h>`
	Convert to char	`char c = static_cast<char>(48+is);`	
	Convert to string	`string str = to_string(123123);`	
	Convert to double	`double d = a;`	

	Addition, Multiplication, Subtraction & Division	`int a = 5;`
		`int b = 4;`
		
		`/*`
		`Note: division of an integer by an integer will round to nearest possible`
		`integer, if want a double, either cast numerator or denominator in double first`
		`*/`
		
		`a+b, a-b, a*b, a/b`	

	Minimum and Maximum	`min(a, b), max(a, b)` `min({a, b, c}), max({a, b, c})`	
	Modulo	`#define MOD 10000007`	

		num % MOD // module all the operations if numbers are large	
	Absolute	abs()	

	Square Root & Power & log	sqrt(a); //returns double
		pow(a, b); //returns double
		log(a);// natural log, ln(a)
		log10(a); //log base 10
			#include <cmath>

	Random	rand(); //random call generates a number between 0 to RAND_MAX;
		(typically as high as INT_MAX)
		lb + rand() % (ub-lb+1)/ RAND_MAX; generates number [lb, ub]	

Long (use for large numbers)	Declaration	long l; // same as 'long int l'
		long long ll; // same as 'long long int ll'	

	Size	sizeof(long) <= sizeof(long long)	
Float/Double	Declaration	float f; double d;	
	Ceiling & Floor	ceil(d); floor(d);	
Boolean	Declaration	bool b;	

Pointers

Declaration	datatype object_datatype; // Pointer are not nullptr by default assign them nullptr explicitly
	
	// pointer_datatype points to memory location of object_datatype
	datatype* pointer_datatype = &object_datatype;
	
	// can also be initialised using 'new'
	datatype* pointer_datatype = new datatype(…);
		Note: Always pass by reference in a recursive function of c++ otherwise it will make a copy of it and run time will be huge (use const if you don't want it to be modified).

Access	// access to member from pointer pointer_datatype -> member_name;	
Dereference	*pointer_datatype	

Structure & Class

Structure	Declaration	/**
		By default all the members in struct are public
		*/
		struct CustomStruct{
		int i;
		char c;
		
		int increment(){
		return i+1;
		}
		};
		// declaration
		CustomStruct s = {34, 'c'};
		CustomStruct s = CustomStruct(34, 'c'); //not possible
		
		// call function on structure
		s.increment();	

Class	Declaration	class Base{
		// can be accessed by derived class + object
		public:	

```
/**
    members
 */
int i;
string s;

/**
    functions
 */

// constructor
Base(int _i){
    i = _i;
}

// Pure virtual function, acting as abstract function
virtual void pure_virtual_func() = 0;

// Virtual function which allows run time binding
virtual void virtual_func(){ cout<<"base virtual_func"<<endl; };

// non-virtual function which is bonded at compile time
void func(){ cout<<"base func:"<<i<<endl; };
```

		// can be accessed by derived classes but not by objects
		protected:
		
		// neither derived class nor object can access privete declarions
		private:
		
		};
		
		class Derived:public Base{
		public:
		
		  void pure_virtual_func(){ cout<<"derived pure_virtual_func"<<endl; };
		  void virtual_func(){ cout<<"derived virtual_func"<<endl; };
		  void func(){ cout<<"derived func:"<<i<<endl; };
		
		  Derived(int _i) :Base(_i){}
		
		};	

	Usage	Derived d(45);
		d.pure_virtual_func(); // derived pure_virtual_func
		d.virtual_func(); // derived virtual_func
		d.func(); // derived func:45	

		Base* b = new Derived(23);
		b -> pure_virtual_func(); // derived pure_virtual_func
		b -> virtual_func(); // derived virtual_func
		b -> func(); // base func:23	

Vector

Vetor	Declaration	vector<int> vec;
		// 10 elements initialised with 0
		vector<int> vec2(10, 0);
		// a 2-dim vector (10x3) each with value 9
		vector<vector<int>> vec3(10, vector<int>(3, 9));
			#include<vector>

	Insert	vec.push_back(value);	

	Access	vec.front(); // front element
		vec[index] or v.at(index); // index'th element
		vec.back(); // back element
		
		// Iterate
		for(vector<int>::iterator it = vec.begin(); it != vec.end(); it++) {
		cout << *it << endl;
		}
		// using indices
		for(int i = 0; i < vec.size(); i++) {
		cout<< vec[i] <<endl;
		}	

	Modify	vec.at(index) = new_value; // Or
		vec[index] = new_value;	

	Remove	vec.pop_back();	

Size	vec.size()	

Sorting	vector<int> vec = { 4, 10, 7, 3};
	
	// sorts in increasing order
	sort(vec.begin(), vec.end());
	
	// sorts in decreasing order
	sort(vec.begin(), vec.end(), greater<int>());
	
	// sorts in custom order
	bool CustomCompare(Object o1, Object o2){
	  return o1.key < o2.key;
	}
	sort(vec.begin(), vec.end(),CustomCompare);
	
	class CustomClass{
	  public:
	  int key;
	};
	
	/*	

```
 Sorting Vector of Custom Class
 */
// Overriding '<' operator
bool operator<(const CustomClass& c1, const CustomClass& c2){ return
c1.key < c2.key; }

// sorts in increasing order
sort(vec.begin(), vec.end());



/**
 Soritng pointer / structure
 */
struct PairSorter{
    bool operator()(const pair<int, string>* a, const pair<int, string>* b){
        return a -> first < b -> first;
    }
};


vector<pair<int, string>*> nums = {
        new pair<int, string>(45, "Dfg"),
        new pair<int, string>(7, "453"),
        new pair<int, string>(6, "sdf"),
        new pair<int, string>(9, "Dfg"),
```

	};
	
	sort(nums.begin(), nums.end(), PairSorter());	

Filling	fill(v.begin(), v.end(), 0);	

Min?max/reverse	int res = *min_element(a.begin(), a.end());
	int res = *max_element(a.begin(), a.end());
	reverse(a.begin(), a.end());	

Binary Search	/**
	Finding pivot of pivoted sorted array
	*/
	int binarySearch(vector<int> nums){
	int l = 0;
	int r = nums.size()-1;
	while(l<=r){
	int m = l + (r-l)/2;
	if(m==nums.size()-1 || nums[m] > nums[m+1]) return m;
	// '<=' is imp as m can be equal to l if r = l+1 in this case we need to search on right half
	else if(nums[l] <= nums[m]) l = m+1;	

		else r = m-1;
		}
		return -1;
		}
		
		In case we only have two conditions:
		
		int start = 0, end = m-1;
		while(**start < end**){
		int mid = start + (end - start)/2;
		int sum = 0;
		for(int i = 0; i< n; i++)
		sum += binaryMatrix.get(i, mid);
		
		if(sum == 0){
		start = mid+1;
		}else{
		end = mid;
		}
		}
		
		Answer is start	

	Binary Search on range	vector<int> nums = {1, 3, 3, 3, 5, 5, 10, 15};
		auto iter = upper_bound(nums.begin(), nums.end(), 4);
		
		cout<<*iter<<endl; //5
		cout<<iter-nums.begin()<<endl; //4
		
		return index of first element greater than passed value	

Queue

Queue (Singe Ended Queue, FIFO)	Declaration	queue<Int> q;	#include<queue>
	Insert	q.push(23);	
	Access	q.front(); q.back();	
	Modify	NA	
	Remove	q.pop();	
	Size	q.size(); q.empty() // shorthand for q.size() == 0	

Priority Queue (FIFO, Priority based)	Declaration	// max-heap (greatest element at top)
		priority_queue<int> p;
		
		//min-heap
		priority_queue<int, vector<int>, greater<int>> p;
		
		class CustomClass{
		public:
		int key;
		};
		
		// Overriding '<' operator
			#include<queue>

		bool operator<(const CustomClass& c1, const CustomClass& c2){
		return c1.key < c2.key;
		}
		//max-heap for CustomClass
		priority_queue<CustomClass> p2;
		
		// Use below for struct or pointers
		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;
		}
		};	

	Insert	p.push(value);	
	Access	p.top();	
	Modify	NA	
	Remove	p.pop();	
	Size	p.size(); p.empty() // shorthand for p.size() == 0	
	Example Problems	1. Find k max out of n:	

		O(klogn) = create max heap of n items and extract k items O(nlogk) = create min heap of k items, any time a greater than top come, remove top and insert new.	
Deque (Doubly Ended Queue, FIFO)	Declaration	deque<int> d;	#include<deque>
	Insert	d.push_front(value); d.push_back(value);	

	Access	d.front();
		d.at(index);
		d.back();	

	Modify	NA	
	Remove	d.pop_front(); d.pop_back();	
	Size	d.size() d.empty() // shorthand for d.size() ==0	

Monoqueue (Contains increasing or decreasing portion of the array)
 	
Useful for finding window kind of question
	Declaration	template <class T> class Monoqueue{
		  private:
		  // first -> number of elements deleted until prev smaller, second -> element
		  deque<pair<int, T>> d;
		
		  public:
		  Monoqueue(){}
		
		  void push(T num){
		  int count = 0;
		  while(!empty() && d.back().second >= num){
		  count += d.back().first + 1;
		  d.pop_back();
		  }
		  d.push_back({count, num});
		  }
		
		  void pop(){
		  if(d.front().first > 0)
		  d.front().first--;
		  else
		  d.pop_front();	

		}
		
		T front(){
		return d.front().second;
		}
		
		bool empty(){ return d.empty(); }
		
		};
		
		Monoqueue<long> mq;	

	Insert	mq.push(23);	
	Access	mq.front()	
	Remove	mq.pop()	
	Size	mq.empty()	

	Example Problems:	https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k/submissions/
		https://leetcode.com/problems/minimum-size-subarray-sum/submissions/
		https://leetcode.com/problems/sliding-window-maximum/submissions/
		
		1. Monoque is helpful if we want to get local or global minim position standing at end idx (which varies from 0 to n).
		2. If we want to find max length substring with specific properties, then
		a. If space is asked to reduce consider sliding the window (using start) until that property violets.
		
		If your function is always increasing and the we need to find max substring/max consecutive/longest substring which holds certain property:
		1. Start with Sliding Window:
		start = 0;
		ans = 0;
		for(end =0 ; end < arr.size(); end++){
		// add here the code to track that
		ModifyProperty(arr, end);
		
		// now check if that property has violated?, if yes than increase the start pointer until that property holds again
		If(proprtyViolets){
		Modify the property if we remove start from sliding window
		Start++;
		}
		ans = max(ans, ens-start+1);
		}
		
		2. Approach 2: consider hashing that property for all I till end and then lookup from hash map if that property matches with that at endIdx.
		https://leetcode.com/problems/subarray-sum-equals-k/
		int subarraySum(vector<int>& nums, int k) {	

```
        vector<int> sum = { 0 };
        for(int i =0; i < nums.size(); i++)
            sum.push_back(sum.back() + nums[i]);


        unordered_map<int, int> hash;
        int count = 0;
        for(int i = 0; i < sum.size(); i++){
            int curr = sum[i];
            if(hash.find(curr-k) != hash.end())
                count += hash[curr-k];
            hash[curr]++;
        }
        return count;
    }
```

3. If its not always increasing: use monoqueue.

Stack

Stack (LIFO)	Declaration	stack<int> s;	#include<stack>
	Insert	s.push(20);	
	Access	s.top()	
	Modify	NA	
	Remove	s.pop()	
	Size	s.size() s.empty() // shorthand for s.size() ==0	

Sets

Unordered Set	Declaration	unordered_set<int> us;	#include<unordered_set>
	Insert	us.insert(value);	

	Access	// find unordered_set<int>::iterator =
		us.find(value);
		
		//check if element exists
		bool exists = us.find(value) != us.end();
		
		for(unordered_set<int>::iterator = us.begin(); it != us.end(); it++)
		cout<<*it<<endl;	

	Modify	NA	
	Remove	us.erase(value);	
	Size	us.size() us.empty() // shorthand for us.size() ==0	

Set (Sorted - implemented using BST.)
 	
Same as Priority Queue but
	Declaration	// increasing order
		set<int> s;
		
		// decreasing order
		set<int, greater<int>> s;
			#include<set>

with *No duplicates + O(1) removal*)	Insert	s.insert(value);	

	Access	//O(log(n)) unordered_set<int>::iterator =
		s.find(value);
		
		//check if element exists
		bool exists = s.find(value) != s.end();
		
		// first element in sorted set
		*s.begin()
		
		for(set<int>::iterator = s.begin(); it != s.end(); it++)
		cout<<*it<<endl;	

	Modify	NA	
	Remove	s.erase(value); // O(log(n))	
	Size	s.size() s.empty() // shorthand for s.size() ==0	

Multi Set (Sorted + allows duplicates - implemented using BST.)
 	
Same as Priority Queue but
	Declaration	// increasing order
		multiset<int> ms;
		
		// decreasing order
		multiset<int, greater<int>> s;
			#include<set>

	Insert	ms.insert(value);	

with *O(long) removal*)	Access	//O(log(n)) find unordered_set<int>::iterator
		ms.find(value);
		
		//check if element exists
		bool exists = ms.find(value) != ms.end();
		
		// first element in sorted set
		*ms.begin()
		
		for(set<int>::iterator = ms.begin(); it != ms.end(); it++)
		cout<<*it<<endl;	

	Modify	NA	
	Remove	ms.erase(value); // find O(log(n))	
	Size	ms.size() ms.empty() // shorthand for ms.size() ==0	

Map

Unordered map	Declaration	// Single dimension hash
		unordered_map<int, string> um;
		
		// initialized with some value
		unordered_map<int, string> um ={
		{ 2, "abc"},
		{ 3, "pqr"},
		};
		
		// using fill syntax
		unordered_map<int, string> um ={
		pair<int, string>(2, "abc"),
		pair<int, string>(3, "pqr"),
		};
		
		// multi dimension hash
		unordered_map<int, unordered_map<int, string>> um;
			#include<unordered_map>

	Insert	um.insert({2, “abc”});	
	Access	// find unordered_map<int, string>::iterator it = um.find(2);	

<table>
<tr><td></td><td></td><td>// if element exists
bool exist = it != um.end();

um.at(key) or um[key] // to access value

// Iterate over all amp
for(unordered_map<int, string>::iterator it = um.begin(); it != um.end();
it++)
cout<<it -> first<<"->"<<it-> second<<endl;</td><td></td></tr>
<tr><td></td><td>Modify</td><td>um[index] = new_value;</td><td></td></tr>
<tr><td></td><td>Remove</td><td>um.erase(index);</td><td></td></tr>
<tr><td></td><td>Size</td><td>um.size()</td><td></td></tr>
<tr><td>Deterministic Finite Automata</td><td></td><td>Picture the DFA as a directed graph, where each node is a state, and each edge is a transition labeled with a character group (digit, exponent, sign, or dot). There are two key steps to designing it.

1. Identify all valid combinations that the aforementioned boolean variables can be in. Each combination is a state. Draw a circle for each state, and label what it means.
2. For each state, consider what a character from each group would mean in the context of that state. Each group will either cause a transition into another state, or it will signify that the string is invalid. For each valid transition, draw a directed arrow between the two states and write the group next to the arrow.

States: {0, 1, 2, …, n-1};</td><td></td></tr>
</table>

		Dfa: for each state, map of valid actions to next state [{ action -> newSate }]	
	Example Problems	https://leetcode.com/problems/valid-number/solution/ https://leetcode.com/problems/number-of-ways-to-paint-n-3-grid/	
Map	Declaration	map<int, int> m;	#include<unordered_map>

	Insert	// Single dimension hash
		map<int, string> m;
		
		// initialized with some value
		map<int, string> m = {
		{ 2, "abc"},
		{ 3, "pqr"},
		};
		
		// using full syntax
		map<int, string> m ={
		pair<int, string>(2, "abc"),	

		pair<int, string>(3, "pqr"),
		};
		
		// multi dimension hash
		map<int, map<int, string>> m;	

	Access	// find: log(n)
		map<int, string>::iterator it = m.find(2);
		
		// if element exists
		bool exist = it != m.end();
		
		// Iterate over all amp
		for(map<int, string>::iterator it = m.begin(); it != m.end(); it++)
		cout<<it -> first<<"->"<<it-> second<<endl;	

	Modify	um[index] = new_value;	
	Remove	m.erase(index);// log(n)	
	Size	m.size()	