# 1 STL Useful Tips

## 1.1 Common libraries

```
/*** Functions ***/
#include<algorithm>
#include<functional> // for hash
#include<climits> // all useful constants
#include<cmath>
#include<cstdio>
#include<cstdlib> // random
#include<ctime>
#include<iostream>
#include<sstream>
#include<iomanip> // right justifying std::right and std::setw(width)
/*** Data Structure ***/
#include<deque> // double ended queue
#include<list>
#include<queue> // including priority_queue
#include<stack>
#include<string>
#include<vector>
```

## 1.2 I/O

```
// iostream and cstdio are both using I/O streams
// However, they have different behavior,
// pay attention on them if you're using them together.

// cin does not concern with '\n' at end of each line
// however scanf or getline does concern with '\n' at end of each line
// '\n' will be ignored when you use cin to read char.

// when you use getline(cin, str) to read a whole line of input
// please add an extra getline before inputing if previous inputs are numbers
cin >> n;
getline(cin, str) // wasted getline
getline(cin, str) // real input string
```

## 1.3 Useful constant

```
INT_MIN
INT_MAX
LONG_MIN
LONG_MAX
LLONG_MIN
LLONG_MAX
(~0u) // infinity (for long and long long)
      // use (~0u)>>2 for int.
```

## 1.4 Space waster

```
// consider to redefine data types to void data range problem
#define int long long // make everyone long long
#define double long double // make everyone long double

// function definitions

#undef int // main must return int
int main(void)
#define int long long // redefine int

// rest of program
```

## 1.5 Tricks in cmath

```
// when the number is too large. use powl instead of pow.
// will provide you more accuracy.
powl(a, b)
(int)round(p, (1.0/n)) // nth root of p
```

## 1.6 Initialize array with predefined value

```
// for 1d array, use STL fill_n or fill to initialize array
fill(a, a+size_of_a, value)
fill_n(a, size_of_a, value)
// for 2d array, if want to fill in 0 or -1
memset(a, 0, sizeof(a));
  // otherwise, use a loop of fill or fill_n through every a[i]
  fill(a[i], a[i]+size_of_ai, value) // from 0 to number of row.
```

## 1.7 Modifying sequence operations

```
void copy(first, last, result);
void swap(a,b);
void swap(first1, last1, first2); // swap range
void replace(first, last, old_value, new_value); // replace in range
void replace_if(first, last, pred, new_value); // replace in conditions
  // pred can be represented in function
  // e.x. bool IsOdd (int i) { return ((i%2)==1); }
void reverse(first, last); // reverse a range of elements
void reverse_copy(first, last, result); // copy a reverse of range of elements
void random_shuffle(first, last); // using built-in random generator to shuffle array
```

## 1.8 Merge

```
// merge sorted ranges
void merge(first1, last1, first2, last2, result, comp);
// union of two sorted ranges
void set_union(first1, last1, first2, last2, result, comp);
// intersection of two sorted ranges
void set_interaction(first1, last1, first2, last2, result, comp);
```

```
// difference of two sorted ranges
void set_difference((first1, last1, first2, last2, result, comp);
```

## 1.9 String

```
// Searching
unsigned int find(const string &s2, unsigned int pos1 = 0);
unsigned int rfind(const string &s2, unsigned int pos1 = end);
unsigned int find_first_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_of(const string &s2, unsigned int pos1 = end);
unsigned int find_first_not_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_not_of(const string &s2, unsigned int pos1 = end);
// Insert, Erase, Replace
string& insert(unsigned int pos1, const string &s2);
string& insert(unsigned int pos1, unsigned int repetitions, char c);
string& erase(unsigned int pos = 0, unsigned int len = npos);
string& replace(unsigned int pos1, unsigned int len1, const string &s2);
string& replace(unsigned int pos1, unsigned int len1, unsigned int repetitions, char c);
// String streams
stringstream s1;
int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

## 1.10 Heap

```
template <class RandomAccessIterator>
  void push_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void push_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);

template <class RandomAccessIterator>
  void pop_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void pop_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);

template <class RandomAccessIterator>
  void make_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void make_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp );

template <class RandomAccessIterator>
  void sort_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void sort_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);
template <class RandomAccessIterator>
  RandomAccessIterator is_heap_until (RandomAccessIterator first,
                    RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  RandomAccessIterator is_heap_until (RandomAccessIterator first,
                    RandomAccessIterator last
                    Compare comp);
```

## 1.11 Sort

```
void sort(iterator first, iterator last);
void sort(iterator first, iterator last, LessThanFunction comp);
void stable_sort(iterator first, iterator last);
void stable_sort(iterator first, iterator last, LessThanFunction comp);
void partial_sort(iterator first, iterator middle, iterator last);
void partial_sort(iterator first, iterator middle, iterator last, LessThanFunction comp);
bool is_sorted(iterator first, iterator last);
bool is_sorted(iterator first, iterator last, LessThanOrEqualFunction comp);
// example for sort, if have array x, start_index, end_index;
sort(x+start_index, x+end_index);

/** sort a map **/
// You cannot directly sort a map<key type, mapped data type>
// if you only want to sort in key type
// you can use insert method to copy map into another map
// b.insert(make_pair(it->first, it->second) /* it is a map iterator */
// this will result a map which sorts key type in increasing order
// if you want to sort key type in decreasing order, then declare your map as
// something like:
// map<char, int, greater<char> >

// if you want to sort based on key, you need to copy the data to a vector
// where elements of vector are pair.
// you can define a PAIR type by using:
typedef pair<char, int> PAIR;

// suppose this is the map
map<char, int> a;

// sort vector in decreasing order
bool cmp_by_value(const PAIR& lhs, const PAIR& rhs) {
  return lhs.second > rhs.second;
}

// sort key in increasing order
bool cmp_by_char(const PAIR& lhs, const PAIR& rhs) {
  return lhs.first < rhs.first;
}

// copy map data to vector
vector<PAIR> b(a.begin(), a.end());

// sort data
sort(b.begin(), b.end(), cmp_by_value);

// you can still call your data by b[i].first and b[i].second.
// THE ABOVE CODES ARE EXAMPLE FOR SORTING A MAP.
// PLEASE USE IT FOR YOUR OWN DEMANDS.
```

## 1.12 Permutations

```
bool next_permutation(iterator first, iterator last);
bool next_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
bool prev_permutation(iterator first, iterator last);
bool prev_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
```

### 1.13 Searching

```
// will return address of iterator, call result as *iterator;
iterator find(iterator first, iterator last, const T &value);
iterator find_if(iterator first, iterator last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator last, const T &value);
bool binary_search(iterator first, iterator last, const T &value, LessThanOrEqualFunction comp);
```

### 1.14 Random algorithm

```
srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```