

C++ Cheat-Sheet - SOI Camp 2019

C++ Template

Use this template to avoid integer overflow with int's and time limit exceeded because of slow input/output.

```
#include <bits/stdc++.h> // includes everything
using namespace std; // avoid typing std::

#define int int64_t // make int to long long

signed main() { // main must return int, and
                // signed is an alias for int
    // Disable synchronization between cin/cout and scanf/printf
    ios_base::sync_with_stdio(false);
    // Disable automatic flush of cout when reading from cin
    cin.tie(0);

    // your code goes here
    ...
}
```

Read from file

For IDEs only: Redirect sample files to stdin:

```
signed main() {
    freopen("sample01.in", "r", stdin);
}
```

Pass by value/reference

Bad: Pass by values. Makes a copy. For vectors: $\mathcal{O}(n)$:

```
int top(vector<int> v) { return v.back(); }
```

Ugly: Pass by mutable reference ($\mathcal{O}(1)$):

```
int top(vector<int>& v) { return v.back(); }
```

Good: Pass by const reference ($\mathcal{O}(1)$):

```
int top(vector<int> const& v) { return v.back(); }
```

Containers

vector ("better array"):

```
vector<int> v(n); // v={0,0,...,0}, v.size()==n
v.clear(); // v={}
v.push_back(5); // v={5}
int x = v[0]; // x=5
for (size_t i=0; i<v.size(); ++i) // custom loop
    cout << v[i] << '\n';
for (vector<int>::iterator it=v.begin(); // iterators
     it!=v.end(); ++it)
    cout << *it << '\n';
for (auto it=v.begin(); it!=v.end(); ++it) // auto
    cout << *it << '\n';
for (auto& elem : v) // range based for
    cout << elem << '\n';
```

map (key-value pairs, access by key, always sorted):

```
map<string, int> m;
m["key"] = 5; // m={"key": 5}
// If the key doesn't exist, it is inserted with default value
int x = m["new"]; // x=0, m={"key": 5, "new": 0}
for (auto& elem : v) // range based for
    cout << elem.first << ' ' << elem.second << '\n';
for (auto& [key, value] : v) // range based for with structured binding
    cout << key << ' ' << value << '\n';
```

Container Overview

vector<T>: $v[i]$, $v.push_back(x)$, $v.pop_front()$.
map<K,V>: $m[k]=v$, $v.erase(x)$, $m.contains(x)$.
set<T>: $s.insert(x)$, $s.erase(x)$, $s.contains(x)$.
multimap<K,V>/multiset<T>: like map/set but allows duplicated keys.

deque<T>: like vector, but with $push_front$
queue<T>/stack<T>: use $deque<T>$ or $vector<T>$ instead.
priority_queue<T>: Has operations ($top()$, $pop()$, $push(x)$), the largest element is at the beginning. To reverse the order:
// Smallest element at $pq.top()$
priority_queue<int, vector<int>, greater<int>> > pq;

Strings

```
string s; // just like vector<char>
cin >> s; // can input and output
s = "cystoflagellate"
string t = s.substr(2, 5); // start at 2, length 5: t=="stofl"
```

Algorithms

```
vector<int> v{5,4,3,2,1};
sort(v.begin(), v.end()); // sort everything
sort(v.begin()+1, v.begin()+4); // sort at indices 1,2,3
// custom compare function: is lhs<rhs?
bool comp(int lhs, int rhs) { return lhs<rhs; }
sort(v.begin(), v.end(), comp); // sort using comp
// define custom less than operator
bool operator<(const mystruct& lhs, const mystruct& rhs) {...}
```

Other useful functions:

```
vector<int> v{1,2,2,5,7,7,8};
// v must be sorted
bool b = binary_search(v.begin(), v.end(), 5); // b==true
auto it = lower_bound(v.begin(), v.end(), 5); // points to 5
v.erase(unique(v.begin(), v.end()), v.end()); // v={1,2,5,7,8}
// v can be arbitrary
auto it = find(v.begin(), v.end(), 5); // it points to first 5
reverse(v.begin(), v.end()); // reverse v
```

Terminal

```
# Compiling C++ (program prog.cc)
$ g++ -Wall -Wextra -std=c++17 -g3 -ggdb3 \
  -fsanitize=address,undefined \
  -D_GLIBCXX_DEBUG prog.cc -o prog
# -Wall -Wextra:
# Enable warnings and extra warnings
# -std=c++17:
# Newest version of C++. On old compilers try one of
# -std=c++14 or -std=c++11 or -std=c++0x.
# -g3 -ggdb3: Write debug informations for gdb
# -D_GLIBCXX_DEBUG:
# Bounds-checking for containers and iterators
# -fsanitize=address,undefined: undefined behaviour
```

```
# testing
$ ./prog <sample01.in # input sample01.in
# input all *.in
$ for f in *.in; do echo "-- $f --"; ./prog <$f; done
```

```
# using gdb
$ gdb prog
(gdb) run <sample01.in
(gdb) bt # show backtrace
(gdb) q # quit
```

Operator overloading

```
struct Point{
    int x, y, id;
};
bool operator<(Point const& a, Point const& b){
    return a.x < b.x;
}
bool operator==(Point const& a, Point const& b){
    return a.x == b.x;
}
Point operator+(Point const& a, Point const& b){
    return Point{a.x+b.x, a.y+b.y, -1};
}
```

Pair, Tuple, Array

Pairs: Two fields, has comparison operators by default

```
pair<int, char> p{42, 'x'};
cout << p.first << " " << p.second; // 42 x
pair<int, char> q = make_pair(42, 'x');
p.second = 'n';
auto [a, b] = p; // unpack
```

Tuple: More fields than a tuple

```
tuple<int, int, bool, int, char> p;
auto [a, b, c, d, e, f] = p; // unpack
bool x = get<2>(p); // get at index
```

Arrays: use if all types are the same

```
array<int, 2> point{4, 2};
auto [x, y] = p; // unpack like in a tuple
cout << p[0] << " " << p[1] << '\n'; // index like an array
```

printf-Debugging

```
// During debugging
#define DEB(x) cerr << x << '\n'
// When submitting
#define DEB(x) // cerr << x << '\n'

for (int i=0; i<10; ++i) {
    DEB("i=" << i << " sum=" << sum);
    sum += i;
}
```

Better: compile with flag "-DDEBUG".

```
#ifndef DEBUG
# define DEB(x) cerr << x << '\n'
#else
# define DEB(x)
#endif
```

Grader errors

TLE (Time limit exceeded): Estimate if your algorithm is fast enough. Set $\log(x) \approx 10$ (for all x) and plug in limits. 10^7 always passes, 10^8 if you optimize, 10^9 usually not.

RE (Runtime error): Out of bounds? Assertion error? Very deep recursion? Too much memory? Pop empty stack?