Standard Template Library and other C++ stuffs

Farhan Ahmad, Nafis Ul Haque Shifat, Debojoti Das Soumya and Jarif Rahman

January 7, 2022

Contents

1 CP Setup

Jarif Rahman

Competitive Programming এ আমাদের অনেক তাড়াতাড়ি কোড করতে হয়। এ কারণে বার বার হেডার ইনক্লুড করা অনেক ঝামেলা হতে পারে। খুশির খবর হলো `bits/stdc++.h` নামক হেডার ফাইল প্রায় সব দরকারি হেডার ফাইল (যেমন: `iostream, vector, string` ইত্যাদি) ইনক্লুড করে নেয়। যে যে ফাইল ইনক্লুড করে তার লিস্ট এখানে পাওয়া যাবে।

C++ এ ইনপুট অউটপুট এর জন্য আমরা `cin` আর `cout` ব্যবহার করে থাকি। C তে আমরা ইনপুট আর অউটপুট এর জন্য `printf` আর `scanf` ব্যবহার করি। `printf` আর `scanf`, `cin` আর `cout` এর তুলনায় অনেক ফাস্ট। এই কারণে পারলে C++ এও `printf` আর `scanf` ব্যবহার করা উচিৎ। কিন্তু যদি কেউ `cin` আর `cout` ব্যবহার করে অভ্যস্ত হয়ে যেয়ে থাকো (including me :)) তাহলে `cin` আর `cout` কেও `fast I/O` নামক ট্রিক ব্যবহার করে ফাস্ট করা যায়। তার জন্য `main` ফাংকশান এর শুরুতে এই দুই লাইন কোড লিখতে হবে:

```
ios::sync_with_stdio(0);
cin.tie(0);
```

আর C++ এর লাইন শেষ করার জন্য `endl` না ব্যবহার করে `"\n"` ব্যবহার করে উচিত। `"\n"`, `endl` এর চেয়ে ফাস্ট কাজ করে।

সব শেষে CP Setup এরকম দেখতে হবে:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    //code here
}
```

2 Macro

Jarif Rahman

C++ এ কোডকে ছোটো করার জন্য একটা সুন্দর ফিচার হলো `#define`। `#define` ব্যবহার করে আমরা একটা macro ডিফাইন করতে পারি। নিচে কিছু উদাহরণ দেখানো হলো।

```
#define vi vector<int>
#define pi pair<int, int>
#define f first
#define sc second
```

এখন

```
pi p = {1, 2};
cout << p.f << " " << p.sc < "\n";
vi v(10, 10);
```

```
for(int i = 0; i < 10; i++) cout << v[i] << " ";
cout << "\n";
```

লেখা আর

```
pair<int, int> p = {1, 2};
cout << p.first << " " << p.second < "\n";
vector<int> v(10, 10);
for(int i = 0; i < 10; i++) cout << v[i] << " ";
cout << "\n";
```

লেখা একই বেপার।

`#define` এ প্যারামিটার ও ব্যবহার করা যায়। যেমন:

```
#define s(a, b) a+b
#define f(i, n) for(int i = 0; i < n; i++)

f(i, 10); // is as same as for(int i = 0; i < 10; i++)
s(10, 15); //10+15
//I know the last one does not shorten the code :P
//It was just for example
```

3 Lambda Expressions

Jarif Rahman

C++ এ ল্যাম্বডা এক্সপ্রেশন নামের ফাংকশানের মতো এক ধরনের এক্সপ্রেশন তৈরি করা যায়। সাধারণত C++ এ সব ধরনের ফাংকশান গ্লোবালি ডিক্লেয়ার করতে হয়। কিন্তু ল্যাম্বডা এক্সপ্রেশন এর সুবিধা হলো এটাকে লোকালিও ডিক্লেয়ার করা যায়। এটার সিনট্যাক্স কিছুটা এরকম:

```
function<ReturnType(Parameters)> functionName = [capture](parameters){/*body*/}
```

`function<ReturnType(Parameters)>` এর বদলে `auto` ব্যবহার করলেও হবে। ল্যাম্বডা এক্সপ্রেশন এ লোকালি ডিক্লেয়ার করা ভেরিয়েবলও ব্যবহার করা যাবে। `capture` এর জায়গায় `&` দিলে, লোকাল ভেরিয়েবল এর মান পরিবর্তন করা যাবে এবং মুল লোকাল ভেরিয়েবল এর মানও পরিবর্তিত হয়ে যাবে। `capture` এর জায়গায় যদি `=` ব্যবহার করা হয় তাহলে লোকাল ভেরিয়েবল ব্যবহার করা যাবে কিন্তু মান পরিবর্তন করা যাবে না। `capture` ফাঁকা রাখলে লোকালি ডিক্লেয়ার করা ভেরিয়েবল ব্যবহার করা যাবে না। প্রত্যেকটা ভেরিয়েবলকে আলাদা ভাবেও `captue` করা যায়। এখানে দেখান উদাহরণ ছাড়াও আরো অনেক ভাবে ল্যাম্বডা ফাংকশান ব্যবহার করা যায়। কিন্তু এখানে competitive programming এ যতটুকু সচারচর কাজে লাগে শুধু ততোটুকু দেখানো হলো। ল্যাম্বডা ফাংকশানকে নিয়ে বিস্তারিত আলোচনা এখানে পাওয়া যাবে।

```
//All of these can be done inside another function
//They don't have to be global

function<int(int, int)> sum = [](int a, int b){
    return a+b;
}
cout << sum(4, 7) << "\n"; //11

int x = 10;
```

```
auto sth = [=](int y){
    return x+y;
};
cout << sth(1) << "\n"; //11

int X = 1;
auto will_change = [&](){
    X++;
};
cout << X << "\n"; //2

auto wont_change = [=](){
    X--;
}; //compilation error as X is read-only this time


//recursive
function<long long(long long)> factorial = [&](long long x){
    if(x == 0) return 1LL;
    else return x*factorial(x-1);
};
//function type cannot be auto this time
//capture can has to &
//I tried with = and it resulted in seg fault
//I am not sure whether other captures will work or not
cout << factorial(5) << "\n"; //5! = 120
```

4 Pair and Tuple

Jarif Rahman

বি.দ্র.: যদি `<bits/stdc++.h>` ইনক্লুড না করা হয় তাহলে tuple ব্যবহার করতে `<tuple>` ইনক্লুড করা লাগবে।

4.1 Pair

Pair দুইটা এলেমেন্টকে স্টোর করে রাখতে পারে। একটা অ্যারেও এলেমেন্ট স্টোর করে রাখতে পারে। কিন্তু pair এর বৈশিষ্ট্য হলো এর দুইটা এলেমেন্ট এর টাইপ একই হয়া লাগবে না। Pair কে ডিক্লেয়ার করতে হয় এভাবে: `pair<type, type>`[1]। এর প্রথম এলেমেন্টকে `first` আর দ্বিতীয় এলেমেন্টকে `second` দ্বারা অ্যাক্সেস করা যায়। (আমি জানি এই কথাগুলা নতুনদের মাথার উপর দিয়ে যেতে পারে :P , চিন্তা করো না, কয়েকটা উদাহরণ দেখলেই ঠিক হয়ে যাবে)।

```
pair<int, int> p; //pair of two integers
p.first = 0;
p.second = 1;
cout << p.first << " " << p.second << "\n"; //0 1
p.first++;
p.second = -1;
cout << p.first << " " << p.second << "\n"; //1 -1

pair<int, string> p2; //pair of an integer and a string
p2 = {1, "hello"};
cout << p2.first << " " << p2.second << "\n"; //1 hello
```

[1] <> গুলা হলো C++ template । এগুলা সম্পর্কে না জানলে আপাতত তেমন একটা সমস্যা হবে না। কিন্তু যেনে রাখা ভালো। click

```
//pair of pairs is also possible, example:
pair<int, pair<int, int>> p3 = {1, {2, 3}};
cout << p3.first << " " << p3.second.first << " " << p3.second.second << "\n";
//1 2 3
```

Pair এর value সরাসরি curly braces ({,}) দিয়ে পরিবর্তন করা যায় এবং pair এর জায়গায় curly braces ব্যবহার করে যায়। তবে কখন কখন curly braces ব্যবহার করলে সমস্যা হতে পারে। যেমন ধর `p` একটা pair । `if(p < {1, 2})` compilation error দিবে। এরকম সময়ের জন্য C++ এ `make_pair` নামের একটা ফাংকশান আছে। এটা দুইটা এলেমেন্টকে প্যারামিটার হিসেবে নেয় আর তাদের Pair রিটার্ন করে। আগের কোডটার জায়গায় আমরা `if(p < make_pair(1, 2))` ব্যবহার করতে পারব।

যখন দুইটা pair কে compare করা হয় (`==`, `!=`, `<`, `>`, `<=`, `>=` ইত্যাদি দ্বারা, আগের উদাহরণেও এটা দেখান হয়েছে) তখন তাদের কে lexographically compare করা হয়। ধর কোন pair টা বড় নির্নয় করতে চাচ্ছ। যেটার প্রথম এলেমেন্ট বড় সেটা বড়। যদি দুইটারই প্রথম এলেমেন্ট সমান হয় তাহলে যেটার দ্বিতীয় এলেমেন্ট বড় সেটা বড়। অবশ্যই দুইটা Pair কে সমান হতে হলে তাদের দুইটা এলেমেন্টই সমান হতে হবে।

```
make_pair(1, 1) == make_pair(1, 1); //true
make_pair(1, 2) < make_pair(1, 3); //true
make_pair(1, 2) < make_pair(1, 1); //false
```

4.2 Tuple

Tuple কে extended Pair বলা যায়। Pair এ মাত্র দুইটা এলেমেন্ট থাকতে পারে। কিন্তু tuple এ ইচ্ছা মত এলেমেন্ট রাখা যায়।

```
tuple<int, int> a = {1, 2};
tuple<int, string, int, bool, char, double> b = {1, "hello", 10, true, 'a', 10.99};
tuple<int, char> c = make_tuple(1, 'c');
tuple<int, tuple<int, pair<int, tuple<int, int>>>> d = {1, {2, {3, {4, 5}}}};
```

Tuple এ যেহেতু ইচ্ছা মত এলেমেন্ট থাকতে পারে তাই অবশ্যই তার এলেমেন্টদের `first`, `second` দিয়ে এক্সেস করা যাবে না। তাদের এক্সেস করার জন্য `get` ফাংকশান ব্যবহার করা লাগবে। এর সিনট্যাক্স হলো: `get<index>(tuple)`। আগের উদাহরণ কে extend করলে পাই:

```
cout << get<0>(a) << " " << get<1>(a) << "\n"; //1 2
get<0>(a)--;
cout << get<0>(a) << " " << get<1>(a) << "\n"; //0 2
```

দুইটা tuple কেও যখন compare করা হয় তখন lexographically compare করা হয়। যার প্রথম এলেমেন্ট বড় সে বড়। দুইটা সমান হলে যার দ্বিতীয় এলেমেন্ট বড় সে বড়। এভাবে চলতে থাকবে। সব এলেমেন্ট সমান হয়ে গেলে tuple দুইটা সমান।

4.3 Tie function and C++17 stuctured binding

Pair বা tuple থেকে বারবার `first`, `last` বা `get` ব্যবহার করে এলেমেন্ট এক্সেস করা ঝামেলা হতে পারে। pair বা tuple এর এলেমেন্ট গুলাকে অন্য ভেরিয়েবলে সেভ করে ব্যবহার করা বেশি সুবিধাজনক হতে পারে। সেটাকে একটা

একটা করেও করা যেতে পারে কিন্তু C++ এ এটার জন্য একটা শর্টকাট আছে। সেটা হলো `tie` ফাংকশান। নিচে `tie` ফাংশানের কিছু উদাহরণ দেখানো হলো।

```
int a, b;
pair<int, int> p = {1, 2};
tie(a, b) = p;
cout << a << " " << b << "\n"; //1 2

tuple<int, int, int> t = {1, 2, 3};
int x, y;
tie(x, ignore, y) = t; //'ignore' ignores that element
cout << x << " " << y << "\n"; //1 3
```

যদি C++17 ব্যবহার করা হয় তাহলে আরো সহজে এই কাজ করা যাবে structured binding ব্যবহার করে।

```
pair<int, int> p = {1, 2};
auto [a, b] = p;
cout << a << " " << b << "\n"; //1 2

tuple<int, int, int> t = {1, 2, 3};
auto [x, y, z] = t;
cout << x << " " << y << " " << z << "\n"; //1 2 3
```

5 Array

Jarif Rahman

বি.দ্র.: C++ array প্রায় useless । এটার সব কাজই C-style অ্যারে (যেমন: `int a[100]`) করতে পারে। কিন্তু এটা যেহেতু stl এর অংশ তাই এটাকে শিখানো হচ্ছে।

বি.দ্র.: যদি `<bits/stdc++.h>` ইনক্লুড না করা হয় তাহলে array ব্যবহার করতে `<array>` ইনক্লুড করা লাগবে।

Array ডিক্লেয়ার করতে হয় এভাবে: `array<type, size>`। এখানে `size` অবশ্যই ধ্রুবক হতে হবে। অর্থাৎ একটা সংখ্যা হতে হবে অথবা `const` সহ ডিক্লেয়ার করা ভেরিয়েবল হতে হবে। C++ array হলো fixed-size অ্যারে। একবার ডিক্লেয়ার করার পর এর সাইজ পরিবর্তন করা যায় না।

```
array<int, 10> a; //array of integers of size 10
//note values are not initialsized and can be random like arrays in C
for(int i = 0; i < 10; i++){
    //do something with a[i]
}
a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
for(int i = 0; i < 10; i++){
    cout << a[i] << " ";
}
cout << "\n";
//1 2 3 4 5 6 7 8 9 10
a[0]++;
cout << a[0] << "\n"; //2

array<string, 10> b; //array of strings
array<array<int, 10>, 10> c; //2D array
array<array<array<int, 10>, 10>, 10> d; //3D array
//their can be more dimensions :)
```

আমরা সাধারণত অ্যারে এবং অন্যান্য STL কন্টেইনার এর এলেমেন্ট ব্যবহার করার সময় ইনডেক্স (যেমন: `a[5]`) ব্যবহার করি। কিন্তু বেশিরভাব STL ফাংকশান ইটারেটর ব্যবহার করে ইমপ্লেমেন্ট করা। ইটারেটর কিছুটা পয়েন্টারের মতো কাজ করে। এটা অ্যারের (এবং অন্যান্য কন্টেইনারের) কোনো এক এলেমেন্ট এর দিকে পয়েন্ট করে। অ্যারের শুরুর এলেমেন্টের ইটারেটর পাওয়া যায় `arrayName.begin()` এর মাধ্যমে। শেষ এলেমেন্টের ঠিক পরের ইটারেটর পাওয়া যায় `arrayName.end()` এর মাধ্যমে।

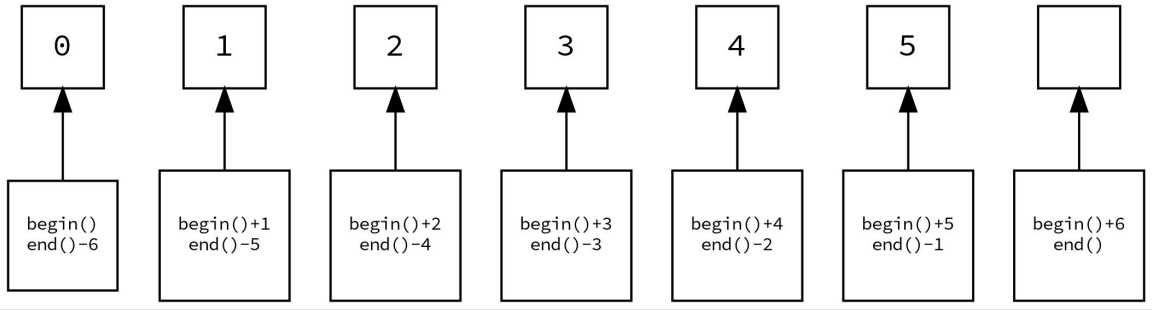

Figure 1: ইটারেটর

অ্যারের ইটারেটর হলো Random Access Iterator । অর্থাৎ আমরা ইটারেটর এর সাথে সহজেই যোগ বিয়োগ করতে পারবো।

```
array<int, 5> a = {1, 10, 5, 3, 2};
array<int, 5>::iterator it = a.begin();
//note array<int, 5>::iterator is too long
//we usually use auto here
cout << *it << "\n"; //1
cout << *(it+3) << "\n"; //3
cout << *a.end() << "\n"; //something random
cout << *(a.end()-1) << "\n"; //2
cout << a.end()-a.begin() << "\n"; //5
```

C++ অ্যারেকে ইনডেক্স করা যায়। তাই এখানে ইটারেটরকে useless মনে হতে পারে। কিন্তু C++ STL এর সব ডাটা স্ট্রাকচারকে ইনডেক্স করা যায় না (যেমন: set)। তাদের ক্ষেত্রে ইটারেটরে দিয়েই কাজ চালাতে হয়। সবাইকে ইনডেক্স করা যায় না কিন্তু সবার ইটারেটর আছে। তাই C++ এর বেশিরভাগ STL ফাংকশান ইটারেটর দিয়ে ইমপ্লেমেন্ট করা। যেমন:

```
array<int, 5> a = {1, 2, 3, 4, 5};
// "reverse(a, b)" function reverses a container from iterator a to iterator b-1
reverse(a.begin(), a.end()); //a = {5, 4, 3, 2, 1}
reverse(a.begin(), a.begin()+3); //a = {3, 4, 5, 2, 1}
sort(a.begin(), a.end()); //a = {1, 2, 3, 4, 5}
cout << count(a.begin(), a.end(), 1) << "\n"; //1
```

6 Vector

Nafis Ul Haque Shifat

Vector কে "Resizable Array" বলা যায়। সাধারণ Array এর সাইজ শুরুতেই বলে দিতে হয়, এর সাইজ বাড়ানো ও যায় না, কিংবা কমানো ও যায় না। Vector এর মজার দিক হচ্ছে যখন ইচ্ছা এর সাইজ বাড়ানো বা কমানো যায়, এতে যেকোনো সময় নতুন element যোগ করা যায়, আবার রিমুভ ও করা যায়।

Initializing:

```
vector<int> v1; // v1 = {}
vector<int> v2(5); // v2 = {0, 0, 0, 0, 0};
vector<string> v3 = {"hi", "Bangladesh", "hello"};
```

এখানে <> এর মধ্যে vector টি কোন ধরণের variable (যেমন `int`, `char`, `string`) জমা রাখবে তা লিখতে হয়. Array তে কোনো এলিমেন্ট যেভাবে access করতে হয়, এখানেও সেভাবেই!

```
vector<int> v = {2, 6, 3, 5};
cout << v[2] << endl; // prints 3
v[1] = 100; // v = { 2, 100, 3, 5}
cout << v[1] << endl; // prints 100
```

push_back(): Vector এর শেষে কোনো উপাদান যোগ করতে হয় push_back() ফাংশন দিয়ে। এটি constant সময় নেয়, অর্থাৎ এর complexity $O(1)$.

```
vector<string> v; // v = {};
v.push_back("Hi"); // v = {"HI"}
v.push_back("Hello"); // v = {"Hi", "Hello"}
cout << v[1] << endl; // prints "Hello"
```

pop_back(): Vector এর শেষ উপাদান pop_back() দিয়ে রিমুভ করা যায়। এটির ও complexity $O(1)$

```
vector<int> v = {1, 2, 5}
v.pop_back(); // v = {1, 2}
v.pop_back(); // v = {1}
```

size(): বুঝাই যাচ্ছে, এটি দিয়ে vector এর সাইজ বের করা যায়। এটির ও complexity O(1)

```
vector<int> v = {4, 5, 2, 19};
cout << v.size() << endl; // prints 4
v.pop_back(); // v = {4, 5, 2};
cout << v.size() << endl; // prints 3
```

insert(): push_back() ফাংশনটি দ্বারা শুধু শেষে কোনো নতুন উপাদান যুক্ত করা যায়, তবে যেকোনো index এ নতুন উপাদানটি বসাতে insert() ব্যবহার করা যায়। এটি ২ টি parameter নেয়, ১টি হচ্ছে position (মূলত ১টি iterator , যা ঐ position কে নির্দেশ করে), আরেকটি হচ্ছে নতুন উপাদান টির মান।

```
vector<int> v = {2, 3, 4, 5};
v.insert(v.begin(), 1); // v  = {1, 2, 3, 4, 5};
v.insert(v.begin() + 2, 10); // v = {1, 2, 10, 3, 4, 5};
```

এখানে `v.begin() + 2` দিয়ে ২য় index কে নির্দেশ করছে, কাজেই `v[2] = 10` হয়ে যাবে। আগে 2nd index এ ছিল 3 , এখন 3 চলে যাবে 3rd index এ, একই ভাবে আগে যার index ছিল 3, এখন তার index হবে 4।

বুঝতেই পারছ এই অপারেশনে vector এর অনেকগুলো উপাদানের অবস্থান পরিবর্তন হয়, তাই এটির complexity $O(1)$ নয়। যদি n টি উপাদানের পসিশন চেঞ্জ হয়, তবে এর complexity $O(n)$.

erase(): এটি অনেকটা insert() এর মতই, শুধু পার্থক্য হচ্ছে এটি উপাদান যোগ না করে রিমুভ করে! এর complexity ও $O(n)$

```
vector<int> v = {1, 2, 3, 5};
v.erase(v.begin()); // v = {2, 3. 5}
v.erase(v.begin() + 2); // v = {2, 3}
```

clear(): এটি vector এর সবগুলি উপাদান রিমুভ করে দেয়, কাজেই vector টি ফাঁকা হয়ে যায়!

```
vector<int> v = {1, 2, 3, 5};
v.clear(); // v = {}
cout << v.size() << endl; // prints 0
```

এবার দেখা যাক vector কখন কাজে লাগে!
তোমার কাছে n টি বক্স আছে, সাথে আছে k টি বল। i তম বলটির গায়ে লিখা সংখ্যা হচ্ছে x[i] , এবং বলটি y[i] তম বক্সে রাখা হবে। এবার তোমাকে প্রতিটি বক্সে থাকা বলের গায়ে লিখা সংখ্যাগুলো প্রিন্ট করতে হবে।

প্রথমেই দেখ, তুমি শুরুতে বলে দিতে পারবে না কোন বক্সে কয়টি করে বল যাবে। তবে যেহেতু মোট বল আছে k টি, তাই প্রতি বক্সে সর্বোচ্চ k টি বল ই যেতে পারবে। তাই আমরা k সাইজ এর n টি array নিতে পারি, সেখানে i -তম array তে থাকবে কোন কোন বল সেই বক্সে গিয়েছে। তবে লক্ষ্য কর এতে প্রচুর মেমরি অপচয় হচ্ছে। যদি $k = 10^5$ হয়, তবে আমরা n টি 10^5 সাইজের array নিয়েছি। কোনো বক্সে ১টি বল গেলেও যতখানি মেমরি লাগবে, 10^5 টি বল গেলেও ওই বক্স বাবদ ততখানি মেমরি ই লাগছে! যারা complexity বুঝে তারা জানে এতে $O(nk)$ মেমরি লাগছে!
এই অবস্থা থেকে বাঁচার জন্য vector রয়েছে, আমরা n টি vector নিব, শুরুতে প্রতিটি ফাঁকা থাকবে। পরে যখন ই কোনো বল i -তম বক্সে যাবে, তখন i -তম vector এ ওই বলটির গায়ে লিখা সংখ্যাটি পুশ করে দিব। এতে কোনো বক্সে যতটি বল যাচ্ছে, সেই বক্স বাবদ ততখানি মেমরি লাগছে। যেহেতু মোট বল আছে k টি, কাজেই মেমরি কমপ্লেক্সিটি দাঁড়াচ্ছে $O(k)$! Code অনেকটা এমন হবে-

```
vector<int> Box[n + 1]; // declaring n + 1 vectors, basically an array of vectors
for(int i = 1; i <= k; i++) {
     Box[y[i]].push_back(x[i]);
}
for(int i = 1; i <= n; i++ {
     for(int j = 0; j < Box[i].size(); j++) {
          cout<< Box[i][j] << " ";
     }
     cout<<endl;
}
```

7 Deque

Debojoti Das Soumya

Deque এর মধ্যে `vector` এর সব function গুলাই আছে। শুধু পার্থক্যটা হলো এখানে দুইটা function extra আছে সেগুলো হলো প্রথমে insert ও remove করা যায় (তাই এটা পড়ার আগে `vector` সম্পর্কে জেনে আস)।

নিচে দেখানো সব function এর complexity $O(1)$ ।

Construction

এইটি deque কে এভাবে declare করতে হয়।

```
// deque<data type> name;
deque<int> dq;
```

push_front()

`push_front()` ব্যবহার করে deque এর শুরুতে নতুন element insert করা যায়।

```
deque<int> dq;
dq.push_front(1); // dq = {1}
dq.push_front(2); // dq = {2, 1}
dq.push_back(3); // dq = {2, 1, 3}
cout << dq[0] << " " << dq[1] << " " << dq[2] << "\n"; // 2 1 3
```

pop_front()

`pop_front()` ব্যবহার করে deque এর প্রথম element remove করা যায়।

```
deque<int> dq;
dq.push_front(1); // dq = {1}
dq.push_front(2); // dq = {2, 1}
dq.pop_front(); // dq = {1}
dq.push_back(3); // dq = {1, 3}
dq.pop_back(); // dq = {1}
```

8 Stack

Nafis Ul Haque Shifat

Stack একটি খুব ই সাদামাটা data structure । তুমি যদি অনেকগুলো প্লেট একটির উপর আরেকটি সাজাতে থাক, তবে নতুন একটি প্লেট বসাতে হলে নিশ্চয়ই সব গুলো প্লেটের উপর নতুন প্লেট টি বসাবে। আবার একটি প্লেট সরাতে হলেও নিশ্চয়ই সবার উপরের প্লেটটি আগে সরাবে। Stack এর কাজটাও এমন ই! stack এ নতুন এলিমেন্ট যোগ করলে (কিংবা stack এর ভাষায় push করলে) এটি সবার উপরে নতুন এলিমেন্ট বসায়, রিমুভ করতে হলে (কিংবা stack এর ভাষায় pop করলে) উপরের এলিমেন্ট টি ই রিমুভ হয়।

বুঝতেই পারছ যে উপাদানটি শেষে যোগ হচ্ছে সেটি সবার আগে রিমুভ হচ্ছে, এই ধরনের data-structure কে LIFO (Last In First Out) data-structure বলা হয়।

stack ব্যবহার করতে <stack> হেডার ফাইলটি include করতে হয় (অথবা `<bits/stdc++.h>`)।

```
#include<stack>
.....
stack<int>  a;
stack<string> b;
stack<char> c;
```

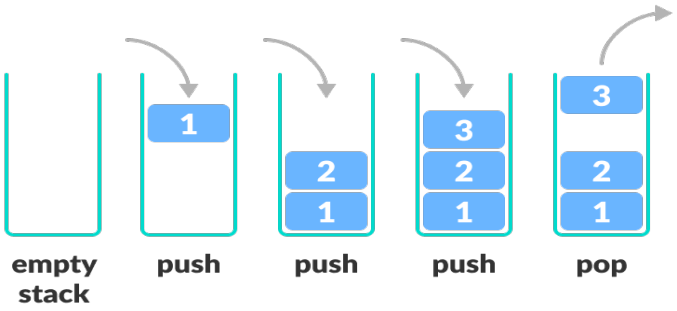

Figure 2: stack

এখানে <> এর মধ্যে stack টি কোন ধরণের variable (যেমন `int`, `char`, `string`) জমা রাখবে তা লিখতে হয়।

push(): stack এ কোনো এলিমেন্ট যোগ করতে হয় push() ফাংশন দিয়ে। এর complexity $O(1)$।

```
stack<int> st; // st = {}
st.push(2); // st = {2}
st.push(1); // st = {2, 1}
```

pop(): stack উপরের এলিমেন্ট রিমুভ করতে হয় pop() দিয়ে। এর complexity $O(1)$ ।

```
stack<int> st; // st = {}
st.push(2); // st = {2}
st.push(1); // st = {2, 1}
st.pop(); // st = {2}
st.pop(); // st = {}
```

top(): এটি stack এর উপরের এলিমেন্ট টি return করে। এর complexity $O(1)$ ।

```
stack<int> st; // st = {}
st.push(2); // st = {2}
st.push(1); // st = {2, 1}
cout << st.top() << endl; // prints 1
st.push(69); // st = {2, 1, 69}
cout << st.top() << endl; // prints 69
```

size(): এটি stack এর সাইজ return করে। এর complexity $O(1)$ ।

```
stack<int> st; // st = {}
st.push(2); // st = {2}
st.push(1); // st = {2, 1}
cout << st.size() << endl; // prints 2
st.pop(); // st = {2}
cout << st.size() << endl; // prints 2
```

stack এর সব কাজ ই STL Vector দিয়েই করে ফেলা যায়। আবার vector এ index এও access করা যায় (যেমন `v[2]` কি আছে বের করতে পারব), তবে তা stack এ করা যাবে না। stack আর vector এর ফাংশন গুলোর complexity একই হলেও stack এর ক্ষেত্রে constant factor কম, তাই তুলনামূলক ভাবে stack কিছুটা fast ।

9 Queue

Debojoti Das Soumya

Queue কে FIFO (first in first out) data structure বলা হয়। Queue এর speciality হল এখানে constant time এ প্রথম element remove করা যায় ও queue এর শেষে নতুন element add করা যায়।

নিচে দেখানো সব function এর complexity $O(1)$।

Construction

এইটি queue কে এভাবে declare করতে হয়।

```
// queue<data type> name;
queue<int> q;
```

push()

`push()` ব্যবহার করে নতুন element queue এর শেষে insert করা যায়।

```
queue<int> q;
q.push(1); // q = {1}
q.push(5); // q = {1, 5}
```

front()

`front()` function টি queue এর প্রথম element return করে।

```
queue<string> q;
q.push("hello"); // q = {"hello"}
cout << q.front() << "\n"; // prints "hello"
q.push("world"); // q = {"hello", "world"}
cout << q.front() << "\n"; // prints "hello"
```

back()

`back()` function টি queue এর শেষ element return করে।

```
queue<string> q;
q.push("hello"); // q = {"hello"}
cout << q.back() << "\n"; // prints "hello"
q.push("world"); // q = {"hello", "world"}
cout << q.back() << "\n"; // prints "world"
```

size()

`size()` function টি queue এ কয়টি element আছে তা return করে।

```
queue<int> q;
cout << q.size() << "\n"; // prints 0
q.push(100); // q = {100}
cout << q.size() << "\n"; // prints 1
q.push(5); // q = {100, 5}
cout << q.size() << "\n"; // prints 2
```

empty()

`size()` এই function queue empty কিনা তার উত্তরে একটি boolean return করে। `true` মানে empty এবং `false` মানে not empty.

```
queue<int> q;
cout << q.empty() << "\n"; // prints 1
q.push(5);
cout << q.empty() << "\n"; // prints 0
```

pop()

`pop()` এই function queue এর প্রথম element remove করে।

```
queue<int> q;
q.push(5); // q = {5}
q.pop(); // q = {}
q.push(50); // q = {50}
q.push(60); // q = {50, 60}
q.push(100); // q = {50, 60, 100}
cout << q.size() << "\n"; // prints 3
cout << q.front() << "\n"; // prints 50
q.pop(); // q = {60, 100}
cout << q.size() << "\n"; // prints 2
cout << q.front() << "\n"; // prints 60
```

10 Set

Farhan Ahmad

Set শব্দটির সাথে তোমরা হইত সবাই পরিচিত গণিত বইয়ের মাধ্যমে। কিছু বস্তুর সমাবেশ বা সংগ্রহ কে set বলা হয়। C++ এ এই সংজ্ঞা অনুসারে একটি in-built STL data-structure রয়েছে। এই in-build data-structure (set) এর কিছু বৈশিষ্ট্য দেওয়া হল:

- Set এর সব কিছু sorted ভাবে থাকে।
- Set এ সব উপাদান unique হয়ে থাকে।

- Set এর মধ্যে কোন একটি element কে একবার insert করলে, ওই element আর modify করা সম্ভব নয়। তবে element টিকে remove করে, আবার modified element কে insert করা যাবে।
- প্রতিটি উপাদান unindexed হয়ে থাকে।

Syntax:

```
set < datatype > setname;
```

datatype দিয়ে value type নির্দেশ করে। উদাহরণস্বরূপ: int, float, char, etc.
কিছু উদাহরণ:

```
set < int > s; // empty set
set < int > s = {1 , 5 , 2, 3}; // Set with values
set < char > s = {'a' , 'c'};
```

কিছু Set এর function সাথে পরিচয় করা যাক।

- **insert():** function টি কোন element insert এ ব্যবহার হয়। *
- **erase():** function টি কোন element erase এ ব্যবহার হয়। *
- **begin():** function টি প্রথম element এর iterator return করে।
- **end():** function টি শেষের পরের কাল্পনিক element এর iterator return করে।
- **size():** function টি set এর size return করে।
- **count():** function টি set এর মধ্যে কোন element আছে নাকি, return করে। *
- **empty():** function টি set টি ফাকা কি না, return করে।

যেগুলো তে * দেওয়া আছে, ওই সব funtion এর time complexity $O(\log(N))$ এবং বাকিদের $O(1)$ |
কিছু উদাহরণ দেওয়া হল:

```
//STL C++ Set examples:

set < int > s;// {}

s.insert(2);// {2}
s.insert(5);// {2, 5}
s.insert(3);// {2, 3, 5}
s.insert(2);// {2, 3, 5}

cout << s.count(2) << endl; // outputs 1

auto itr = s.begin(); // iterator of the first element
s.erase(*itr); // erases 2 for the set. {3, 5}

cout << s.count(2) << endl; // outputs 0
```

```
cout << s.size() << endl; // outputs 2

s.insert(100);// {3, 5, 100};

for(auto i = s.begin(); i != s.end(); i++){
    cout << *i << ' ';
} // prints 3 5 100

cout << endl;

cout << s.size() << endl; // outputs 3

auto it = s.end();
it--;
cout << *it << endl;// outputs 100

s.erase(*it);// {3, 5};
```

11 Priority Queue

Debojoti Das Soumya

Priority queue হলো এমন ডাটা স্ট্রাকচার যেখানে element গুলো একটা নির্দিষ্ট order এ sort করা থাকে। এই ডাটা স্ট্রাকচারে নতুন element insert ও এই order অনুযায়ী sort করলে যে প্রথম element পাওয়া যাবে সেটা access করা যায় ও remove করা যায়।

Construction

Priority queue কে এভাবে declare করা হয়। কোনো custom comparator include না করলে সাধারণত **বড় থেকে ছোট** ক্রমে সাজানো থাকে।

```
// priority_queue<data type> name;
priority_queue<int> pq;
```

push()

এই function ব্যবহার করে priority queue তে নতুন element insert করা যাবে $O(\log n)$ time এ যেখানে n হলো priority queue এর size ।

```
priority_queue<int> pq;
pq.push(1); // pq = {1}
pq.push(2); // pq = {2, 1}
```

top()

এই function priority queue এর প্রথম element return করবে $O(1)$ time এ।

```
priority_queue<int> pq;
pq.push(1); // pq = {1}
cout << pq.top() << endl; // prints 1
```

```
pq.push(4); // pq = {4, 1}
cout << pq.top() << endl; // prints 4
```

pop()

এই function priority queue এর প্রথম element remove করবে $O(\log n)$ time এ যেখানে n হলো priority queue এর size ।

```
priority_queue<string> pq;
pq.push("a"); // pq = {"a"}
pq.push("ab"); // pq = {"ab", "a"}
pq.push("b"); // pq = {"b", "ab", "a"}
cout << pq.top() << endl; // prints "b"
pq.pop(); // pq = {"ab", "a"}
cout << pq.top() << endl; // prints "ab"
pq.pop(); // pq = {"a"}
cout << pq.top() << endl; // prints "a"
```

size()

এই function priority queue এর size return করবে $O(1)$ time এ।

```
priority_queue<int> pq;
cout << pq.size() << endl; // prints 0
pq.push(4); // pq = {4}
cout << pq.size() << endl; // prints 1
```

12 Mutliset

Farhan Ahmad

Multiset ও অনেকটা set এর মতো, এখানে শুধু একই element একের অধিক বার থাকতে পারে। Multiset এর কিছু বৈশিষ্ট্য দেওয়া হল:

- Multiet এর সব কিছু sorted ভাবে থাকে।
- Multiset এ সব উপাদান unique নাও হতে পারে।
- Multiset এর মধ্যে কোন একটি element কে একবার insert করলে, ওই element আর modify করা সম্ভব নয়। তবে element টিকে remove করে, আবার modified element কে insert করা যাবে।
- প্রতিটি উপাদান unindexed হয়ে থাকে।

Syntax:

```
multiset < datatype > setname;
```

datatype দিয়ে value type নির্দেশ করে। উদাহরণস্বরূপ: int, float, char, etc.
কিছু উদাহরণ:

```
multiset < int > m; // empty set
multiset < int > m = {1 , 5 , 2, 3 , 1, 1, 2}; // Set with values
multiset < char > m = {'a' , 'c' , 'a'};
```

কিছু Multiset এর function এর সাথে পরিচয় করা যাক।

- **insert():** function টি কোন element insert এ ব্যবহার হয়। *
- **erase():** function টি কোন element erase এ ব্যবহার হয়। *
- **begin():** function টি প্রথম element এর iterator return করে।
- **end():** function টি শেষের পরের কাল্পনিক element এর iterator return করে।
- **size():** function টি set এর size return করে।
- **count():** function টি set এর মধ্যে কোন element এর count return করে। *
- **empty():** function টি set টি ফাকা কি না, return করে।

যেগুলো তে * দেওয়া আছে, ওই সব funtion এর time complexity $O(log(N))$ এবং বাকিদের $O(1)$ |
কিছু উদাহরণ দেওয়া হল:

```
//STL C++ Multiset examples:

multiset < int > m;// {}

m.insert(2);// {2}
m.insert(5);// {2, 5}
m.insert(3);// {2, 3, 5}
m.insert(2);// {2, 2, 3, 5}

cout << m.count(2) << endl; // outputs 2

auto itr = m.begin(); // iterator of the first element
m.erase(*itr); // erases 2 for the set. {3, 5}

m.insert(3); // {3, 3, 5}
m.insert(3); // {3, 3, 3 , 5}

//if you want to delete only one occurence of 3
m.erase(m.find(3)); // {3, 3, 5}

cout << m.count(2) << endl; // outputs 0
cout << m.size() << endl; // outputs 3

m.insert(100);// {3, 3, 5, 100};

for(auto i = m.begin(); i != m.end(); i++){
    cout << *i << ' ';
} // prints 3 3 5 100
```

```
cout << endl;

cout << m.size() << endl; // outputs 4

auto it = m.end();
it--;
cout << *it << endl;// outputs 100

m.erase(*it);// {3, 3, 5};
```

13 Map

Jarif Rahman

সাধারণত আমরা array, vector বা deque যাই ব্যবহার করি না কেন, এদের ইনডেক্স একটা পূর্ণসংখ্যা হয় এবং এর মান 0 এবং কন্টেইনার এর সাইজ মাঝে হয়। কিন্তু map এ ইনডেক্স এমন যেকোনো ডাটা স্ট্রাকচার হতে পারে যাদেরকে "<" দ্বারা compare করা যায় (যেমন: `int`, `long long`, `float`, `pair`, `tuple`, `string` ইত্যাদি)। এবং এদের মান যেকোনো হতে পারে। Map এ ইনডেক্স গুলাকে key বলা হয়। Map, set এর মতো স্ট্রাকচার ব্যবহার করে। এর প্রত্যেকটা key এর মান বের করতে, মান পরিবর্তন করতে এবং নতুন key insert করতে $O(\log n)$ সময় লাগে যেখানে n হলো map টার বর্তমান সাইজ। map সবসময় সেট এর মতো sorted থাকে। map কে ডিক্লেয়ার করতে হয় এভাবে: `map<keyType, valueType>`।

```
map<int, int> mp1;
map<int, pair<int, int>> mp2;
//if value for a key is not assigned, default value of valueType will be assigned
//for integers it is 0, for pair<int, int> it is {0, 0}
cout << mp1[0] << "\n"; //0
cout << mp2[0].first << " " << mp2[0].second << "\n"; //0 0

map<int, int> mp3 = {{1, 2}, {3, 4}, {5, 6}};
cout << mp3[1] << " " << mp3[3] << " " << mp3[5] << "\n"; //2 4 6

map<pair<int, int>, int> mp4;
mp4[{4, 5}] = -1;
mp4[{1, 3}] = -2;
cout << mp4[{1, 3}] << " " << mp4[{4, 5}] << "\n"; //-1 -2

//each element of map is a {key, value} pair
//and they are sorted

for(pair<pair<int, int>, int> p: mp4){
    cout << p.first.first << " " << p.first.second << " " << p.second << "\n";
}
/*
1 3 -2
4 5 -1
*/

map<int, int> mp5;

cout << mp1.size() << " " << mp2.size() << " " << mp3.size() << " " << mp4.size()
<< mp5.size() << "\n"; // 1 1 3 2 0
```

যেহেতু map এ কোনো key এর মান না থাকলে তার জন্য default মান সেট করে দেয়া হয় তাই শুধু `[]` ব্যবহার করে কোনো key map টাতে আছে কি না তা নির্নয় করা ঝামেলা। এই যায়গায় `map.find()` ফাংকশানটা ব্যবহার করা যেতে

পারে। যেই ফাংকশানটা কোনো key এর ইটারেটর রিটার্ন করে। যদি key টা না থাকে তাহলে `map.end()` রিটার্ন করে।

```
map<int, int> mp;
if(mp.find(0) == mp.end()) cout << "not found\n";
else cout << "found\n";
//not found
mp[0] = 1;
if(mp.find(0) == mp.end()) cout << "not found\n";
else cout << "found\n";
//found
```

14 Indexed Set (PBDS)

Jarif Rahman

C++ এর set এর ইটারেটর Random Access Iterator না। আমরা এর ইটারেটর এর সাথে চাইলেই যেকোনো সংখ্যা যোগ করতে বা বিয়োগ করতে পারি না। আবার দুইটা ইটারেটর এর দূরত্বও নির্নয় করতে পারি না। যদি `s` একটা set হয়, `s.begin()+2`, `s.end()-2`, `any_set_iterator-s.begin()` এগুলা compilation error দিবে। আমরা শুধু `++` অথবা `--` করে এক এক করে বারাতে বা কমাতে পারি। এই কাজ গুলা করার জন্য C++ এ `prev`, `next` এবং `distance` নামের ফাংকশান আছে। কিন্তু এরা worst case এ $O(n)$ সময় নেয়। Set এর ইটারেটর Random Access না হওয়ায় এদের ইনডেক্সও করা যায় না। এই সমস্যা সমাধানের এর জন্য Policy-based data structures ব্যবহার করা যায়। G++ compiler C++ STL ডাটা স্ট্রাকচার বাদেও আরো কিছু ডাটা স্ট্রাকচার suupport করে। এদের Policy-based data structures বলা হয়। Policy-based data structures ব্যবহার করতে হলে `<ext/pb_ds/assoc_container.hpp>` ইনক্লুড করা লাগবে আর `__gnu_pbds` namespace ব্যবহার করা লাগবে। অর্থাৎ:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
```

এর পরে indexed set ডিক্লেয়ার করা যাবে এভাবে:

```
template<class T>
using indexed_set = tree<T,null_type,less<T>,rb_tree_tag,
    tree_order_statistics_node_update>;
```

indexed set এ set এর সব ফাংকশান ছারাও আরো কিছু ফাংকশান ব্যবহার করা যাবে। যেমন:
`s.find_by_order(i)`, `s` এর `i`তম এলেমেন্টের ইটারেটোর রিটার্ন করবে। আবার `s.order_of_key(x)`, `x` এর ইনডেক্স রিটার্ন করবে।

```
indexed_set<int> s;
s.insert(2);
s.insert(3);
s.insert(5);
s.insert(-1);
s.insert(4);

for(int x: s) cout << x << " "; cout << "\n"; //-1 2 3 4 5

cout << *s.find_by_order(0) << "\n"; //-1
```

```
cout << *s.find_by_order(2) << "\n"; //3
cout << *s.find_by_order(3) << "\n"; //4

cout << s.order_of_key(-1) << "\n"; //0
cout << s.order_of_key(2) << "\n"; //1
cout << s.order_of_key(5) << "\n"; //4
```

উল্লেখ্য যে এই indexed set এর constant factor set এর চেয়ে অনেক বেশি। তাই বিনা কারণে indexed set ব্যবহার করা উচিত না।