দ্বিতীয় অংশ - স্ট্যান্ডার্ড টেম্প্লেট লাইব্রেরী

**স্ট্যাক**

    stack< int > st;  
    st.push( 100 ); // inserting 100  
    st.push( 101 ); // inserting 101  
    st.push( 102 ); // inserting 102  
      
    while( !st.empty() ) {  
        cout << st.top() << endl; // printing the top  
        st.pop(); // removing that one  
    }

**কিউ**

ধরো তুমি বাসের টিকেট কিনে লাইনে দাঁড়িয়ে আছো। এখন বাসে ওঠাটা হচ্ছে আমার কাজ(প্রসেসিং)। কাকে আগে বাসে উঠতে দিবে? যে সবার আগে এসেছে, তাকে। এটাকে বলে কিউ - যে সবার আগে এসেছে তাকে আগে প্রসেস করা।   
    queue< int > q;  
    q.push( 100 ); // inserting 100  
    q.push( 101 ); // inserting 101  
    q.push( 102 ); // inserting 102  
      
    while( !q.empty() ) {  
        cout << q.front() << endl; // printing the front  
        q.pop(); // removing that one  
    }

**প্রায়োরিটি কিউ**

এটাও এক ধরণের কিউ শুধু পার্থক্য হচ্ছে যার দাম যত বেশি তাকে তত আগে প্রসেস করা হচ্ছে।

    priority\_queue< int > q;  
    q.push( 10230 ); // inserting 10230  
    q.push( 1021 ); // inserting 1021  
    q.push( 102322 ); // inserting 102322  
      
    while( !q.empty() ) {  
        cout << q.top() << endl; // printing the top  
        q.pop(); // removing that one  
    }

**ইটারেটর**

ইটারেটার হলো  অনেকটা সি এর পয়েন্টারের মত একটা জিনিস। ইটারেটর আসলে পরে কাজে লাগবে, কারণ অনেক জায়গায়ই STL এর ফাংশনগুলো একটা অ্যাডরেস পাঠায়, যে আমি যেই ডাটাটাকে খুঁজছি, সেটা ঠিক কোথায় আছে।

ইটারেটর ডিক্লেয়ার করে এইভাবে  
    vector< int > :: iterator i;  
    vector< double > :: iterator j;

আর ফর লুপ দিয়ে একটা ভেক্টরের প্রথম থেকে শেষ পর্যন্ত সব এলিমেন্টের গলা কাটতে চাই তাহলে সেটা লিখতে হবে এভাবে।  
    vector< int > v; v.pb( 1 ); v.pb( 2 ); v.pb( 3 );  
    vector< int > :: iterator i;  
    for( i = v.begin(); i < v.end(); i++ ) {  
        printf("%d\n", \*i);  
        // ei khane gola kato!  
    }

**সর্ট**

ধরো আমার কাছে কিছু নাম্বার আছে, আমি সেগুলোকে ছোট থেকে বড়তে সাজাবো, বা উল্টো কাজটা করবো, বড় থেকে ছোটতে সাজাবো। এই কাজটাকে বলে সর্ট করা। যদি তুমি সর্ট করার নিয়ে পড়াশুনা করে ফাটাই ফেলতে চাও তাইলে  [এইখানে](http://en.wikipedia.org/wiki/Sorting_algorithm) একটু ঢু মারো।

STL এ সর্ট করা খুব সহজ। ধরো আমার একটা ভেক্টর v আছে, সেটা আমি সর্ট করবো। তাহলো আমার শুধু লিখতে হবে -

sort( v.begin(), v.end() );

তাহলে সে ছোট থেকে বড় তে ভেক্টরটাকে সর্ট করে ফেলবে।

বেশ তো, এখন আমার ডাটাটা হচ্ছে এরকম - (চাচা চৌধুরী আবার বংশ নিয়ে মাথা ঘামায় না)

struct data {  
    char name[100];  
    int height, weight;  
    long long income;  
};

চাচা চৌধুরী যেটা নিয়ে মাথা ঘামায় সেটা হলো পোলার কত টাকা কামাই। যদি দুইটা পোলার সমান কামাই হয়, তাইলে যেই পোলার হাইট ভালো, সেই পোলা লিস্টে আগে থাকবে। আর যদি দুই পোলার হাইট সমান হয় তাইলে যেই পোলার ওজন কম, সেই পোলা আগে থাকবে। আর যদি দুই পোলার ওজন সমান হয়, তাইলে যেই পোলার নাম ছোট সেই পোলা আগে থাকবে।

এই কাজটা দুই ভাবে করা যায়। সবচে সহজটা হলো একটা কম্পেয়ার ফাংশন লিখে।

bool compare( data a, data b ) {  
    if( a.income == b.income ) {  
        if( a.height == b.height ) {  
            if( a.weight == b.weight )

              return strlen( a.name ) < strlen( b.name );  
            else return a.weight < b.weight;  
        }else return a.height > b.height;  
    }else return a.income > b.income;  
}

এই ফাংশনটা গ্লোবালি ডিক্লেয়ার করে যেখানে তুমি সর্ট করতে চাও সেখানে লিখতে হবে।

sort( v.begin(), v.end(), compare );

কম্পেয়ার ফাংশনটা রিটার্ন করবে a কি b এর আগে বসবে কি না। আর কিছু না।

সর্ট করার অন্য পথটা হচ্ছে অপারেটর ওভারলোড করে। ধরো, আমরা যখন বলি ২ < ৩ আমরা বুঝে নেই যে ২ হচ্ছে ৩ এর ছোট - মানের দিক দিয়ে। এখন একটা স্ট্রাকচার কখন অন্য আরেকটা স্ট্রাকচারের চেয়ে ছোট হবে? এই জিনিসটা তোমার প্রোগ্রামে ডিফাইন করে দিতে হবে। এখানে খেয়াল করো, ছোট হবার মানে বোঝাচ্ছে সে লিস্টে আগে থাকবে।

আমি যদি একই কাজটা অপারেটর ওভারলোড দিয়ে করতে চাই, সেটা এরকম হবে।

struct data {  
    char name[100];  
    int height, weight;  
    long long income;  
      
    bool operator < ( const data& b ) const {  
        if( income == b.income ) {  
            if( height == b.height ) {  
                if( weight == b.weight )

                   return strlen( name ) < strlen( b.name );  
                else return weight < b.weight;  
            }else return height > b.height;  
        }else return income > b.income;  
    }  
  
};

এখানে কিন্তু আমি এই ডাটাটাকেই অন্য আরেকটা ডাটা b এর সাথে তুলনা করছি, সেজন্য আমার আগেরটার মতো a কে লাগছে না।

আর আমার সর্ট এর কমান্ড লিখতে হচ্ছে এইভাবে।

     sort( v.begin(), v.end() );

তোমার যদি ভেক্টর ব্যবহার করতে আপত্তি থাকে, ধরো ভেক্টর দেখলেই হাঁচি আসা শুরু করে, নাক চুলকায় কিংবা এধরণের কিছু, তুমি সাধারণ অ্যারেই ব্যবহার করতে পারো।

ধরো সেক্ষেত্রে অ্যারেটা হবে এরকম -

data array[100];

sort( array, array + n );

যেখানে n হচ্ছে অ্যারেতে কতগুলো ডাটাকে তুমি সর্ট করতে চাও।

তুমি যদি 3 নাম্বার (0 based)থেকে 10 নাম্বার পর্যন্ত সর্ট করতে চাও লিখো

 sort( array+3, array+11 );

**সেট**

কোন কিছুর সেট বলতে আসলে বুঝায় শুধু জিনিসগুলোর নাম একবার করে থাকাকে।

যেমন A = { রহিম, করিম, গরু, বিড়াল, করিম, বালিশ, রহিম, করিম } একটা সেট না, কিন্তু

 A = { রহিম, করিম, গরু, বিড়াল, বালিশ } একটা সেট।

STL এর সেট করে কি, সেট এ সব ডাটা গুলো একবার করে রাখে, আর ডাটাগুলোকে সর্ট ও করে রাখে। এটা হলো সেট এর কাজ কারবার -

    set< int > s;  
    s.insert( 10 ); s.insert( 5 ); s.insert( 9 );  
      
    set< int > :: iterator it;  
    for(it = s.begin(); it != s.end(); it++) {  
        cout << \*it << endl;  
    }

যদি তুমি স্ট্রাকচার টাইপের ডাটা রাখতে চাও সেট এ, শুধু < অপারেটরটা ওভারলোড করে ওকে বলে নিও, যে তুমি ছোট বলতে কি বুঝাচ্ছো।  বাকি কাজ ওই করবে।

সেট সাধারণত এধরণের প্রবলেমগুলোতে কাজে লাগে। আমাকে অনেকগুলো সংখ্যা দিয়ে বলল, এখানে ইউনিক কয়টা সংখ্যা আছে। সেক্ষেত্রে আমি খালি একটার পর একটা সংখ্যা ইনপুট নিতে থাকবো তো নিতেই থাকবো, আর সেটে ঢুকাবো তো ঢুকাতেই থাকবো, তারপর খালি সেটের সাইজ প্রিন্ট করে দিবো। কেল্লা ফতেহ!

**ম্যাপ**

ম্যাপও সেটের মতো একটা জিনিস। কিন্তু ম্যাপ সেটের মত কোন জিনিস একটা রেখে ওই ধরণের বাকি সবাইকে বাইরে ফেলে দেয় না।

তবে এভাবে ভাবার চেয়ে ম্যাপকে আরেকটু সহজভাবে ভাবা যায়। একটা অ্যারের কথা চিন্তা করো, আমরা করি কি অ্যারের একটা ইনডেক্সে ডাটা জমাই না? কেমন হতো, যদি ইনডেক্সটা শুধু সংখ্যা না হয়ে যেকোন কিছু হতে পারতো? ধরো, ১ নম্বর ইনডেক্সে না রেখে, "বাংলাদেশ" নামের ইনডেক্সে ডাটা যদি রাখতে পারতাম? তখন ব্যাপারটা দাঁড়াতো আমাদের একটা ম্যাজিক অ্যারে আছে যেটাই আমরা যেকোন ধরণের ডাটা জমিয়ে রাখতে পারি আমাদের ইচ্ছা মতো যে কোন ধরণের ইনডেক্স দিয়ে।

সহজভাবে ম্যাপকে তুমি এভাবে চিন্তা করতে পারো, ম্যাপ হচ্ছে একটা অ্যারে, যেটার ইনডেক্স যেকোন কিছুই হতে পারে, আর সেটাতে যেটা ইচ্ছে সেটাই রাখা যেতে পারে!  
    map< string, int  > m;  
    string goru;  
      
    while( cin >> goru ) {  
        if( goru == "moro" ) break;  
        m[ goru ] ++;  
        cout << goru <<" ase " << m[ goru ] << " ta :D " << endl;          
    }  
      
    এই প্রোগ্রামটা করবে কি, গরুর নাম ইনপুট নিতে থাকবে, আর প্রতিবার বলবে যে ওই জাতের কয়টা গরু আছে।  ম্যাপকে অ্যারের মত ধরেই ইনক্রিমেন্ট করা যায়।

অবশ্য তুমি যদি তোমার বানানো কোন স্ট্রাকচার/ক্লাস রাখতে চাও ইনডেক্স হিসেবে, তোমাকে সেটার জন্য < অপারেটরটা ওভারলোড করে দিতে হবে।

**স্ট্রিংস্ট্রিম**

 ধরো কোন শয়তান খুব শখ করে প্রবলেম সেট তৈরী করলো, আমাকে বলল, "তোমাকে একলাইনে যত্তগুলা ইচ্ছা তত্তগুলা করে সংখ্যা দিমু, তুমি আমারে সর্ট কইরা দিবা! মুহাহাহাহা!" তখন কষে একটা চড় মারতে ইচ্ছে করলেও কিছু করার নেই। তোমাকে তাই করতে হবে।

আমরা লাইনের ইনপুট নেই হচ্ছে গেটস দিয়ে।

তো ব্যাপারটা এরকম হবে।

    char line[1000];  
    while( gets( line ) ) {  
        stringstream ss( line ); // initialize kortesi  
        int num; vector< int > v;  
        while( ss >> num ) v.push\_back( num ); // :P  
        sort( v.begin(), v.end() );  
        // print routine  
    }

ss এর পরের হোয়াইল লুপ অংশটা তুমি cin এর মতো করেই ভাবতে পারো! ;) আমি সেভাবেই চিন্তা করি।

**পেয়ার**

STL এর একটা স্ট্রাকচার বানানো আছে, যার অবস্থা মোটামুটি এইরকম।

struct pair {  
    int first, second;  
};

তবে জিনিসটা এমন না, তুমি যেকোন টাইপে কাজ করতে পারো। যেমন এটা যদি আমি STL এর পেয়ার দিয়ে লিখি, জিনিসটা হবে এরকম   
pair< int, int > p;

এই চেহারাটা কি মনে পড়ে? একে কি আগে দেখেছো? হুমম, ম্যাপের স্ট্রাকচারে আরেকবার চোখ বুলাও। ;)

আমরা ইচ্ছে মতো পেয়ার ডিফাইন করতে পারি, যেভাবে ইচ্ছে। ম্যাপের ডাটা টাইপের মতনই!

pair< int, int > p;  
pair< int, double > x;  
pair< double, string > moru;  
pair< goru, goru > fau;

যা ইচ্ছে!

**নেক্সট পারমুটেশন, প্রিভ পারমুটেশন**

 ধরো হঠাৎ একদিন ঘুম থেকে উঠে দেখলা যে তোমার এগারোটা বাচ্চা এবং কালকে ঈদ আর আজকে তোমার ওদের জন্য ঈদের জামা কিনতে হবে। সমস্যা হচ্ছে, তোমার বউ এরই মধ্যে এগারোটা জামা কিনে ফেলেছে আর আরো সমস্যা হচ্ছে সেটা সে লটারি করে দিয়ে দিয়েছে এবং সেজন্য যাদের যাদের জামা পছন্দ হয়নি তারা কান্নাকাটি করছে। তো তোমার খুব মন খারাপ, তুমি চাও ঈদের দিনের সুখ যাতে সবচে' বেশি হয়। আর তুমি এটাও জানো কোন জামা পড়লে কোন বাচ্চা কতটুকু সুখি হবে। এখন আমাদের সবার সুখের যোগফল ম্যাক্সিমাইজ করতে হবে।

এধরণের প্রবলেমকে বলা হয় কম্প্লিট সার্চ। আমাদের সবগুলো অপশন ট্রাই করতে হবে। ধরো তিনটা বাচ্চার জন্য অল পসিবল ট্রাই করা হচ্ছে এরকম - (জামার নাম্বার দিয়ে)

আবু গাবু ডাবু

  ১     ২    ৩

  ১     ৩    ২

  ২     ১    ৩

  ২     ৩    ১

  ৩     ১    ২

  ৩     ২    ১

এখন এভাবে যদি আমি এগারোটা বাচ্চার জন্য ঈদের জামা পড়িয়ে দেখতে চাই আমার খবরই আছে - 11! ভাবে ট্রাই করতে হবে। তো সেই জন্যই আছে STL এর নেক্সট পারমুটেশন

    vector< int > v;  
    for(int i=0; i<11; i++) v.push\_back( i );  
      
    do {  
        // protitat jama prottekke porai dekho shukh maximize hochche kina  
    }while( next\_permutation( v.begin(), v.end() ) );

আমরা ৩ এর জন্য যেভাবে সবগুলো পারমুটেশন জেনারেট করেছি, সেটাই এই নেক্সট পারমুটেশন করবে। খেয়াল কোর যে, নেক্সট পারমুটেশন কিন্তু ঠিক অ্যালফাবেটিকালি পরের পারমুটেশনটাকে নেয়। তুমি যদি সব পারমুটেশন চাও, প্রথমে অবশ্যই অ্যারেটাকে সর্টেড রেখো।

**রিভার্স**

রিভার্স হচ্ছে একটা কিছুকে ধরে উল্টাই দেয়া।

ধরো আমার একটা ভেক্টর আছে।

vector< int > nacho;

reverse( nacho.begin(), nacho.end() );

পরের স্টেটমেন্টটা লিখলে, সে নাচোকে উল্টাই দিবে।

তো এই ছিলো ব্যাসিক সি++ আর STL।