C++ STL :: vector

ফেব্রুয়ারি 16, 2011zobayer2009১টি মন্তব্য

ভেক্টরঃ

অ্যারে আমাদের সবারই বিশেষভাবে পরিচিত, এবং সবার খুবই প্রিয় একটা ডাটা-স্ট্রাকচার হল অ্যারে। কোন কোডে অ্যারে ব্যবহার করতে পারলেই কেমন যেন শান্তি শান্তি লাগে... অ্যারের সবচেয়ে আকর্ষনীয় ফিচার মনে হয় তার ইন্ডেক্সিং (মানে র‍্যান্ডম অ্যাকসেস)। কিন্তু যতই দিন যায়, সব কেমন যেন কঠিন হয়ে যায়... তাইতো আজকাল অ্যারে নিয়েও মাঝে মাঝে ঝামেলায় পড়তে হয়, বিশেষতঃ যখন কিনা অ্যারের আকার আকৃতি রানটাইমে পরিবর্তন করার দরকার হয়। কারন, অ্যারে একবার ডিক্লেয়ার করলে রানটাইমে C++ এ তা আর ছোট-বড় করা যায় না। আর এখানেই ভেক্টরের আবির্ভাব।

ভেক্টরকে বলা যেতে পারে একটা ডায়নামিক অ্যারে, দরকার মত যেটার সাইজ বাড়ানো কমানো যায়। আবার অ্যারের মত মাল্টি ডাইমেনশন এবং ইন্ডেক্সিং সুবিধাগুলো ব্যবহার করা যায়।

কিভাবে ব্যবহার করবো?

ভেক্টর C++ STL এর একটা কন্টেইনার ক্লাস। অর্থাৎ এটাও একটা টেমপ্লেট ক্লাস, যার কারনে এটাকে অন্য টেমপ্লেটের সাথে সহজেই ব্যবহার করা যায়। তবে সবার প্রথমে দরকার স্ট্যান্ডার্ড হেডার vector ইনক্লুড করাঃ

```
1  #include <vector>
2  using namespace std;
```

এর পরের কাজ হল ভেক্টর ডিক্লারেশন, সেটাও আবার কয়েকভাবে করা যায়, তবে সাধারন স্টাইল হলঃ vector < type > Var; এখানে type হতে পারে যে কোন টাইপ, অন্য কোন ইউজার ডিফাইনড টাইপ, টেমপ্লেট ক্লাস টাইপ, বেসিক টাইপগুলো। ভেক্টর ডিক্লেয়ার করার সাধারন সিন্ট্যাক্সগুলো নিচে দেখানো হলঃ

```
1  // constructing vectors
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main () {
7      // constructors used in the same order as described above:
8      vector<int> first;                                // empty vector of
   ints
9      vector<int> second (4,100);                       // four ints with
   value 100
10     vector<int> third (second.begin(),second.end());  // iterating
   through second
11     vector<int> fourth (third);                       // a copy of third
12
13     // the iterator constructor can also be used to construct from arrays:
14     int myints[] = {16,2,77,29};
15
```

```
    vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );


    cout << "The contents of fifth are:";
    for (unsigned i=0; i < fifth.size(); i++) cout << " " << fifth[i];
cout << endl;


    return 0;
}
```

ভেক্টর ডিক্লেয়ার করার সময় [] আর () অপারেটরের মধ্যে পার্থক্যটা খেয়াল করা উচিতঃ

```
#include <iostream>
#include <vector>
using namespace std;


int main () {
    // use of [] and ()
    vector<int> v1[100]; // creates an array of vectors, i.e. 100 vectors
    vector<int> v2(100); // creates 1 vector of size 100


    // creating a 2D array 100x2 where each element is a vector,
    // so in total, it is a 3D structure, as vector itself is 1D
    vector<int> v3[100][2];
    return 0;
}
```

উপরের কোডে যেমন দেখা গেল... চাইলেই আমরা ভেক্টরের অ্যারে তৈরি করতে পারি (এটা ভেক্টরের অন্যতম একটা গুরুত্বপূর্ন ব্যবহার)। কিন্তু এছাড়াও ভেক্টরে দিয়ে মাল্টিডাইমেনশনাল স্ট্রাকচার তৈরি করা যায়, যেখানে প্রতিটা ডাইমেনশনই ডায়নামিকঃ

```
#include <iostream>
#include <vector>
using namespace std;


int main () {
```

```
    // multidimentional vector
    vector< vector< int > > V2D;
    // the above is basically a vector where each element is a vector,
    // we can also increase the level of nesting if needed.


    // demonstrate a triangular structure..
    vector< int > temp;
    for(int i = 0; i < 10; i++) {
        temp.clear();
        for(int j = 0; j <= i; j++) {
            temp.push_back(i+1);
        }
        V2D.push_back(temp);
    }
    return 0;
}
```

হুম, তো আমরা খালি একটা ভেক্টরের মধ্যে আরেকটা ভেক্টর পুশ করলাম, ফলাফল 2D ভেক্টর। আগেই বলা হয়েছে, ভেক্টর একটা টেমপ্লেট ক্লাস, তাই, যে কোন টাইপ নিয়ে এটা বানানো যায়। যেমন চাইলেই আমরা কিউএর একটা ভেক্টর বানাতে পারিঃ vector< queue > Vq;

ভেক্টর অ্যাকসেসঃ

ভেক্টর দুইভাবে অ্যাকসেস করা যায়, ইটারেটরের সাহায্যে, ইন্ডেক্সিং-এর সাহায্যে। ইটারেটরের সাহায্যে করলে কিঞ্চিত ফাস্ট হয়, ভেক্টরের ইটারেটর একটা র‍্যান্ডম অ্যাকসেস ইটারেটর।

```
#include <iostream>
#include <vector>
using namespace std;


int main () {
    vector< vector< int > > V2D;


    // creating a 2D triangular structure

```

```
9       for(int i = 0; i < 10; i++) {
10          vector< int > temp;
11          for(int j = 0; j <= i; j++) {
12              temp.push_back(i);
13          }
14          V2D.push_back(temp);
15      }
16
17      // using iterator
18      cout << "using iterator:\n";
19      vector< vector< int > > :: iterator outer;
20      vector< int > :: iterator inner;
21      for(outer = V2D.begin(); outer != V2D.end(); outer++) {
            for(inner = outer->begin(); inner != outer->end();
inner++) {
22              cout << *inner << ' ';
23          }
24          cout << '\n';
25      }
26
27      // using index
28      cout << "\nusing indexes:\n";
29      for(unsigned i = 0; i < V2D.size(); i++) {
30          for(unsigned j = 0; j < V2D[i].size(); j++) {
31              cout << V2D[i][j] << ' ';
32          }
33          cout << '\n';
34      }
        return 0;
}
35
36
37
```

সাধারনত কেউ ভেক্টরে ইটারেটর ব্যবহার করে না, কারন ইন্ডেক্সের ব্যবহার অনেক সহজ, এবং অধিকাংশ C++ কোডারের পয়েন্টারের প্রতি আজন্মের ভয়। কিন্তু ইটারেটরের ব্যবহার আরো বেশি সিগ্নিফিক্যান্ট, এবং অর্থবহ। অবশ্য কিভাবে ব্যবহার করতে হবে সেটা কোডারের ব্যক্তিগত ইচ্ছা। যার যেটায় সুবিধা সেটাই ব্যবহার করা উচিত। উপরের কোডে ভ্যালুগুলো চাইলে অ্যারের মত করে মডিফাইও করা যাবে, যেমন, V2D[i][j] = 100; লিখে দিলেই ওই পজিশনটার মান ১০০ হয়ে যাবে।

পুশব্যাক, পপব্যাক, সাইজ, এম্পটি

আগের কোডটায় আমরা দেখলাম পুশব্যাক দিয়ে ভেক্টরে ভ্যালু ইন্সার্ট করা হচ্ছে, তাই আর নতুন করে সেটা দেখনোর কিছু নাই, push_back() ফাংশনের সাহায্যে ভেক্টরের শেষে একি ধরনের একটা আইটেম অ্যাড করা যায়, আর pop_back() দিয়ে ভেক্টরের শেষ থেকে একটা আইটেম হাওয়া করে দেওয়া যায়। অন্যান্য STL মেম্বারের মত ভেক্টরের size() আর empty() ফাংশন দুইটাও আইডেন্টিক্যাল। নিচের কোডে এগুলোর ব্যবহার দেখানো হলঃ

```
#include <iostream>
#include <vector>
using namespace std;

int main () {
    vector< int > simple;

    for(int i = 1; i < 10; i++) {
        simple.push_back(i*100);
        cout << "inserting: " << simple.back() << endl;
    }
    cout << "--------------------\n";

    while(!simple.empty()) {
        cout << "size: " << simple.size();
        cout << " last element: " << simple.back() << endl;
        simple.pop_back();
    }
    cout << "vector empty\n";
    return 0;
}
```

রিসাইজ, ইরেজ, ক্লিয়ার এবং রিসাইজ নিয়ে কিছু কাহিনীঃ

ভেক্টরের সাইজ মডিফায় করা যায় এরকম কিছু মেম্বার হল resize(), erase(), clear(), এদের মধ্যে clear() এর ব্যবহার অন্য যে কোন কন্টেইনার ক্লাসের মতই, সব এলিমেনত ডিলিট করে দেয়, ফলাফম সাইজ ০ হয়ে যায়। আর resize()

ফাংশন দিয়ে ভেক্টরের সাইজ পরিবর্তন করা যায়। erase() এর কাজ কোন এলিমেন্ট বা একটা রেঞ্জ ডিলিট করা যায়। erase() এর দুইটা ফরম্যাট আছে, erase(iterator pos), erase(iterator first, iterator last), অর্থাৎ কোন ভ্যালুর সাপেক্ষে ডিলিট করা যায় না, তার ইটারেটর পজিশন দিতে হবে, আর দ্বিতীয় ধরনে ভেক্টরের [first, last) এই রেঞ্জের সব আইটেম ডিলিট করে দিবে। বলাই বাহুল্য, আজগুবি ইটারেটর দিলে রানটাইম এররর খেয়ে বসে থাকতে হবে...

```
#include <iostream>
#include <vector>
using namespace std;

int main () {
    unsigned int i;
    vector<unsigned int> myvector;
    // set some values (from 1 to 10)
    for (i=1; i<=10; i++) myvector.push_back(i);
    // erase the 6th element
    myvector.erase (myvector.begin()+5);
    // erase the first 3 elements:
    myvector.erase (myvector.begin(),myvector.begin()+3);
    cout << "myvector contains:";
    for (i=0; i<myvector.size(); i++)
        cout << " " << myvector[i];
    cout << endl;

    // clear all
    myvector.clear();
    cout << "size: " << myvector.size() << endl;

    // resize and then check size
    myvector.resize(10);
    cout << "size: " << myvector.size() << endl;
    return 0;
}
```

কিছু বিষয়ে এখানে সতর্ক হতে হবে, যেমনঃ resize() ফাংশনটা ভেক্টরের আগের অবস্থাত কোন তোয়াক্কা না করেই তার সাইজ চেঞ্জ করবে, ফলে কিছু এলিমেন্ট বাদও পড়তে পারে, আবার কিছু জায়গা খালিও থাকতে পারে। কিন্তু ভেক্টরের সাইজ চেক করলে নতুন সাইজ এ পাওয়া যাবে, যদিও, সেটা হয়তো একটা খালি ভেক্টর ছিল। তাই, resize() এর পরে push_back() ব্যবহার করলে কোডার যাই আশা করুক না কেন, সে রিসাজকৃত স্পেসের পরেই পুশ করবে। যেমন, মনে করি, ভেক্টরে ছিল ১৮ টা আইটেম, আমি সেটাকে রিসাইজ করলাম ৫০ এ। তখন push_back() করলে সে ১৯ তম পজিশনে করবে না, করবে ৫১ তম পজিশনে (মানে ইন্ডেক্স ৫০)। একই ভাবে যদি ১২ রে রিসাইজ করতাম, তাহলেও সে ১৯ এ না করে ১৩ তে পুশ করতো। সাধারনত clear() এর অল্টারনেট হিসাবে resize() ব্যবহার করা যায়, তবে এর সাথে push_back() ব্যবহার না করাই ভাল। নিচের কোডঃ

```
#include <iostream>
#include <vector>
using namespace std;


int main () {
    int n, a;
    vector< int > v;
    while(cin >> n) {
        v.resize(n);
        for(a = 0; a < n; a++) {
            cin >> v[a];
        }
        for(a = 0; a < n; a++) {
            cout << v[a] << endl;
        }
    }
    return 0;
}


```

erase() এর ব্যাপারেও একটা কথা আছে, তা হল, এর কমপ্লেক্সিটি লিনিয়ার, তাই রান্ডম ইনসার্ট আর ডিলিটের জন্য ভেক্টর সুবিধাজনক না। x.clear() করা আর x.erase(x.begin(), x.end()) একই কথা। তাই কোডে খুব বেশি clear না মারাই ভাল।

ভেক্টরের ইটারেটর পাব কিভাবে?

উপরের কোডগুলোতে প্রায়ই দেখা গেছে begin() আর end() নামের দুটি ফাংশন। এরা যথা ক্রমে ভেক্টরের শুরু আর শেষের ইটারেটর রিটার্ন করে। মনে রাখতে হবে, STL এর যে কোন মেম্বারই একটা কমন রুল ফলো করে, তা হল, এখানে রেঞ্জ ডিফাইন করা হয় [first, last) দিয়ে, তার মানে last টা ইনক্লুডেড না। একই ভাবে end() ইটারেটর হল লাস্ট আইটেমের পরের ইটারেটর টা। এদের উলটা ফাংশন ২টা হলঃ rbegin(), rend(), যথাক্রমে রিভার্স বিগিন আর রিভার্স এন্ড ইটারেটর রিটার্ন করে।

অনেক হল...

ভেক্টর আসলে অনেক বিশাল একটা ব্যাপার, এটা কিভাবে C++ এ ইমপ্লিমেন্ট করা হয়, সেটাও আরেক মজার ব্যাপার। এক পোস্টে সব বলা সম্ভব না, আর দরকারও নাই, গুগল আছে কি করতে... তবে এই ফাংশন গুলার বাইরে কোনটাই লাগে না। এগুলার এক্সপার্ট মানেই ভেক্টরের এক্সপার্ট, তাও শেষে ভেক্টরের একটা বহুল ব্যবহার দেখাই, সেটা হল, অ্যাডজাসেনসি লিস্ট দিয়ে গ্রাফ রিপ্রেজেন্টেশনঃ

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  const int MAX = 100;
6  typedef pair< int, int > Edge; // to, weight
7
8  int main () {
9      int n, e, i, j, u, v, w;
10     vector< Edge > G[MAX]; // adjacency list for MAX vertices
11     while(cin >> n >> e) {
12         // n nodes in range [0,n), e edges
13         for(i = 0; i < n; i++) G[i].clear(); // forget previous info
14         for(i = 0; i < e; i++) {
15             // directed edge from u to v with cost w
16             cin >> u >> v >> w;
17             G[u].push_back(Edge(v, w));
18         }
19         // now show the graph
20         for(i = 0; i < n; i++) {
21             cout << "Degree of " << i << ": " << G[i].size() <<
endl;
22             cout << "Adjacents:\n";
23             for(j = 0; j < (int)G[i].size(); j++) {
24                 cout << ' ' << G[i][j].first << "(" <<
G[i][j].second << ")\n";
25             }
26             cout << endl;
27         }
28     }
```

```
27      return 0;
28  }
29
30
```

রেফারেন্সঃ

ভেক্টরের কারনে কাজ অনেক সহজ হয়ে যায়, তবে ভেক্টর কিছুটা স্লো আর বেশি মেমরি নেয়। যারা অপ্টিমাইজেশন পছন্দ করেন তাদের জন্য এটা খুব একটা মজার কিছু না।

ডিটেইলসঃ http://www.cplusplus.com/reference/stl/vector/

বিভাগ:ডাটা স্ট্রাকচার, প্রোগ্রামিংট্যাগসমূহ:c++, data structure, programming, stl

C++ STL :: pair

অক্টোবর 1, 2010zobayer200911 comments

কিছু কথাঃ

যারাই কিনা সি/সি++ নিয়ে বেশকিছুদিন নাড়াচাড়া করছেন, তারা প্রায় সবাই সি এর একটা দারুন জিনিস স্ট্রাকচার-এর সাথে পরিচিত, আর, আরেকটু অ্যাডভান্সডরা তো মনে হয় অলরেডি সি++ এর ক্লাস নামের জিনিসটা মোয়া বানিয়ে ফেলেছে।

সি একটা ফাটাফাটি ল্যাঙ্গুয়েজ আর সি++ এর কথা তো বলাই বাহুল্য। যারা অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর সুবাতাস পেয়েছেন তারা এটা আরো ভাল করে জানেন। আমরা জানি, সি++ এ কিছু প্রিমিটিভ ডাটা টাইপ ডিফাইন করা আছে, যাদের উপরে আমরা খুব সহজেই বিভিন্ন অপারেশন চালাতে পারি, কিন্তু মাঝে মাঝে এই রকমের ডাটা টাইপের ব্যবহার আমাদের কে ঝামেলায় ফেলতে পারে। যেমন, মনে করা যাক আমাকে একটা 2D গ্রিডের কিছু পয়েন্ট স্টোর করতে হবে। শুধু int টাইপের অ্যারে দিয়ে এটা মেইনটেইন করাটা বেশ মুশকিলের ব্যাপার। কিন্তু যদি এরকম একটা ডাটা টাইপ থাকতো ‘point’ নামে, যা কিনা (x, y) আকারে কো-অর্ডিনেট রাখতে পারতো!!! সি এ সেই ব্যাবস্থা করেই দেয়া আছে, যেন প্রোগ্রামাররা চাইলেই ইচ্ছা মতো ডাটা টাইপ বানিয়ে নিতে পারেন, আর সি++ এটাকে আরেক ডিগ্রি এগিয়ে নিয়ে গেছে। এখানে প্রোগ্রামার চাইলে তার বানানো ডাটা টাইপের আচার আচরণও বলে দিতে পারেন।

কিন্তু, প্রশ্ন হল, এটা কন্টেস্ট প্রোগ্রামিং এর জন্য কতটা সুইটেবল?

পেয়ার কি?

কন্টেস্ট প্রোগ্রামিং-এ সাধারনতঃ খুব কমপ্লেক্স ডাটা টাইপ বানাতে হয় না। সেখানে বেশি প্রয়োজন খুব দ্রুত আর নির্ভুল কোডিং। যেমন, প্রায়ই দেখা যায়, একটা গ্রাফের এজ গুলো স্টোর করতে হবে, বা জিয়োমেট্রির প্রবলেমের জন্য কো-অর্ডিনেট নিয়ে কাজ করতে হবে, কখনো বা দেখা যায় কিছু স্ট্রিং-এর জন্য কিছু নাম্বার দিতে হবে, অথবা একগাদা ডাটা বিভিন্ন ক্রাইটেরিয়ার ভিত্তিতে সর্ট বা সার্চ করতে হবে। সাধরনতঃ এসব ক্ষেত্রে প্রোগ্রামার যদি নিজে থেকে ডাটাটাইপ বানাতে যায়, কোন সন্দেহ নাই তাতে তার মূল্যবান কিছু সময় নষ্ট হবে। যেমন, নিচের প্রবলেমটা দেখিঃ

আমাকে একগাদা 2D পয়েন্ট থাকবে, আমাকে সেগুলা সর্ট করতে হবে। এখন, আমি চাইলেই একটা স্ট্রাকচার বানাতে পারিঃ

```
1   struct point { int x, y; } P[128];
```

অর্থাৎ, P[] হল একটা পয়েন্ট টাইপের অ্যারে, এখন এটাকে সর্ট করতে হবে। কাজ তো সোজাই, অ্যালগোরিদমের হেডারটা ইনক্লুড করে sort() মেরে দিলেই হয়... কিন্তু আসলে এটা এত সিম্পল না, কারন, sort() একটা টেমপ্লেট ফাংশন, মানে যে কোন ডাটাটাইপের জন্য কাজ করবে, কিন্তু তার আগে তাকে ডাটাটাইপের ভাবগতি জানাতে হবে। আমি ‘struct point’ দিয়ে কি বুঝাতে চাই, এটা তার বুঝার কোন কারনই নাই যদি না আমি বলে দেই। তার মানে খালি sort() কে ডাকলেই চলবে

না, তার সাথে অপারেটর ওভারলোড বা কম্পেয়ার ফাংশন লিখে বুঝিয়ে দিতে হবে যে আমি আসলে কি চাই। আর এখানেই std::pair এর প্লাস পয়েন্ট।

std::pair জিনিশটা তেমন কিছুই না, জাস্ট দুইটা ভ্যালু কে একসাথে করে রাখে, যেখানে ভ্যালু দুইটা যে কোন টাইপের হতে পারে, ডিফারেন্ট টাইপেরও হতে পারে। পেয়ার ক্লাসটা ডিফাইন করা থাকে std <utility> নামের হেডারে।

```
#include <utility>
using namespace std;
```

pair ক্লাসের ডেফিনিশনঃ

```
template <class T1, class T2> struct pair {
    T1 first;
    T2 second;
    pair(): first(T1()), second(T2()) {}
    pair(const T1 &x, const T2 &y): first(x), second(y) {}
    template <class U, class V> pair(const pair<U, V> &p): first(p.first),
second(p.second) {}
};
```

যাদের টেমপ্লেট সম্পর্কে আইডিয়া নাই, তাদের জন্য অল্প কথায়, টেমপ্লেট একটা সিস্টেম যেখানে কোন অ্যাকচুয়াল টাইপের জন্য ডিজাইন থাকে না, বরং যে কোন টাইপের জন্য তার একটা ইন্সট্যান্স তৈরি করা যায়। টেমপ্লেট শিখতে হলে এই পেজটি দেখা যেতে পারে।
pair এর সুবিধা হল, এটা টেমপ্লেট টাইপ, তাই STL algorithm এর ফাংশন গুলার জন্য সাধারনতঃ pair অবজেক্ট গুলার আলাদা করে কোন পরিচয় দেওয়ার দরকার পড়ে না। যেমন আগের প্রবলেমে জাস্ট sort() কল দিলেই চলে, সে অটমেটিক প্রথমে পেয়ারের প্রথম মেম্বার, তার পর ২য় টা, তার পর ৩য় টা, এভাবে বাকি গুলা কম্পেয়ার করে দেখবে। প্রোগ্রামারকে এর জন্য কিছুই বলতে হবে না।

কিভাবে ব্যবহার করে?

pair নিয়ে কাজ করতে চাইলে <utility> হেডার ইনক্লুড করা উচিৎ, অবশ্য যে কোন STL হেডার ইনক্লুড করলেই pair ব্যবহারের সুবিধা পাওয়া যায়। আর pair টাইপের অবজেক্ট সহজে ক্রিয়েট করার জন্য <utility> হেডারের make_pair() ফাংশন ব্যবহার করা যায়। আর pair-এর ১ম আর ২য় এলিমেন্টকে যথাক্রমে .first আর .second মেম্বার দিয়ে অ্যাক্সেস করা যায়। নিচে একটা উদাহরন দেখানো হলঃ

```
#include <iostream>
#include <string>
#include <utility>
using namespace std;


int main() {
    // simple constructions
    pair< int, int > px, py;
```

```
    pair< int, int > p1(23, 43);
    pair< int, int > p2 = pair< int, int >(234, 534);
    px = p1;
    py.first = p2.first * px.second, py.second = p2.second * px.first;
    cout << "py: (" << py.first << ", " << py.second << ")\n";


    // bit more complex
    pair< pair< int, int >, pair< int, int > > p3;
    p3 = pair< pair<int, int>, pair< int, int > > (px, py);
    cout << "p3: ((";
    cout << p3.first.first << ", " << p3.first.second << "), (";
    cout << p3.second.first << ", " << p3.second.second << "))\n";


    // using make_pair()
    pair< double, pair< string, int > > p4;
    p4 = make_pair(3.14159, make_pair("pi", 5) );
    cout << "this is " << p4.second.first << ", value: " << p4.first;
    cout << " precision: " << p4.second.second << " digits\n";
    return 0;
}
```

pair নিয়ে সবচেয়ে বেশি যে কথাটা শোনা যায় তা হল, মানুষজন নাকি .first.second.second.first…. এরকম চেইন লিখতে লিখতে টায়ার্ড হয়ে যায়। কিন্তু একটু কাজ করেই এইটাকে সহজ করে ফেলা যায়, যেমন, নিচের কোডে pair কে #define করে নেওয়া হয়েছেঃ

```
#include <iostream>
using namespace std;


#define pii pair< int, int >
#define ppi pair< pii, int >
#define ff first
#define ss second
```

```
7
8  int main() {
9      ppi p1;
10     pii p2;
11     cin >> p2.ff >> p2.ss;
12     p1 = ppi( p2, p2.ss * p2.ff );
13     cout << "entry: " << p1.ff.ff << ", " << p1.ff.ss << endl;
14     cout << "product: " << p1.ss << endl;
15     return 0;
16 }
17
```

অন্যান্য STL কন্টেইনারের ডাটা টাইপ হিসাবেও pair ব্যবহার করা যায়। যেমন, বি.এফ.এস. অ্যালগরিদমে প্রায়ই কিউতে একজোড়া নাম্বার রাখতে হতে পারে, অথবা ডায়াকস্ট্রার অ্যালগরিদমে পুরা একটা এজকে প্রাইওরিটি কিউতে রাখার ব্যাবস্থা করতে হয়। এটাও খুব সহজেই করা যায়ঃ

```
1  #include <queue>
2  using namespace std;
3
4  #define pii pair< int, int >
5  #define edge pair< int, pii >
6  // edge.first is weight, edge.second is a pair indicating endpoints
7  queue< pii > Q;
8  priority_queue< edge, vector< edge >, greater< edge > > PQ;
```

সতর্কতাঃ

অন্যান্য সব STL অবজেক্টের মত, এখানেও একটা ব্যাপার খেয়াল করতে হবে তা হলঃ উপরের উদাহরন গুলাতে দেখা যায় কন্সট্রাকশন শেষ করার সময় মাঝে মাঝে পর পর দুইটা > ব্যবহার করতে হয়, (যেমন প্রাইওরিটি কিউ এর উদাহরনে শেষে .....greater< edge > >, এখানে শেষ ২টা > এর মাঝখানে একটা স্পেস ব্যবহার করা হয়েছে, এটা কিন্তু সৌন্দর্য বর্ধনের জন্য না। এটা না দিলে অনেক সময় কম্পাইলারের মাথা গরম হয়ে যায়। কারন সি++ এ >> আরো কয়েকটা অপারেটরের কাজ করে। আর যেসব যায়গায় দেখা যায় pair ইউজ করলে আরো ঝামেলা হয়, সেখানে নরমাল স্ট্রাকচার ব্যবহার করাটাই বেশি ভাল। এটা জেনে রাখা ভাল, আর জানলেই যে বসাতে হবে এমনও কোন কথা নাই।

ব্যবহারঃ

সাধারনতঃ STL এর টেমপ্লেট ক্লাসগুলোর ওভারলোডিং সুবিধা নেওয়ার জন্য এবং ছোটো খাটো স্ট্রাকচারের শর্টহ্যান্ড হিসাবে pair ব্যবহার করা হয়ে থাকে। এছাড়া std::map এ হাশিং এর সময় pair ব্যবহার করা হয়। প্রোগ্রামিং প্রবলেমগুলাতে গ্রাফ আর জিওমেট্রিক রিপ্রেজেন্টেশনে pair অনেক ব্যবহার করা হয়।

আশা করি যারা আগ্রহি হবে তারা এটাকে নিয়ে আরো কিছুক্ষন ঘাটাঘাটি করবে, কারন ঘাটাঘাটি খোচাখুচি করাই প্রোগ্রামিং শেখার সবচেয়ে ভাল টেকনিক।

বিস্তারিতঃ http://www.cplusplus.com/reference/std/utility/pair/
বিভাগ:ডাটা স্ট্রাকচার, প্রোগ্রামিংট্যাগসমূহ:c++, data structure, pair, programming, stl

C++ STL :: priority_queue

অগাষ্ট 17, 2010zobayer20093 comments

প্রায়োরিটি কিউঃ

সব ডাটা-স্ট্রাকচারই যে ধোয়া তুলসি পাতা, এইটা বলা যায় না, বিশেষতঃ যখন "জোর যার মুল্লুক তার" টাইপের এই ডাটা-স্ট্রাকচারটা প্রায়ই ব্যবহার করতে হয়, C++ STL এর priority_queue, এটা একটা বাইনারি হিপ ডাটা-স্ট্রাকচার (ডিফল্টঃ ম্যাক্স হিপ)। সহজ বাংলায়, হিপ হল এমন একটা ট্রি ডাটা-স্ট্রাকচার যেখানে সবচেয়ে বড় (ম্যাক্স হিপ) বা সবচেয়ে ছোট (মিন হিপ) এলিমেন্টটা সবসময় রুটে থাকবে। তাই, যদি এমন একটা ঘটনা ঘটে যে, একদল লোক লাইন দিছে কাঙ্গালি ভোজে, এমন টাইমে মাস্তান টাইপের এক ফকির আসছে, আর লোকজন ভয় পেয়ে তাকে লাইনের সামনে দাঁড়া করায় দিবে, তখন প্রায়রিটি কিউ ছাড়া উপায় নাই, অর্থাৎ খালি আগে আসলেই হবে না, যথেষ্ট পরিমানে ভাব নিয়ে আসতে হবে। এইটা খুব সহজেই করা যায়, বার বার চেক করে যে নেক্সট কার প্রায়রিটি বেশি, কিন্তু এইভাবে করাটা আন-এফিসিয়েন্ট, তার চেয়ে হিপের মত একটা ট্রি ডাটা-স্ট্রাকচার ব্যবহার করে অনেক কম সময়ে এই কাজ করা যায়, আর সেই কাজটাকে আর সহজ করে দেওয়ার জন্যই আছে priority_queue, queue এর মত এটাও একটা অ্যাডাপ্টার ক্লাস, তার মানে হল, যে সব STL কন্টেইনার front(), push_back(), pop_back() এই অ্যাকসেস দেয়, তাদেরকে priority_queue এর ইন্টারনাল কন্টেইনার হিসাবে ব্যবহার করা যাবে। আর এটা ব্যবহার করতে চাইলে std <queue> হেডারটা প্রোগ্রামে ইনক্লুড করতে হবেঃ

```
1  #include <queue>
2  using namespace std;
```

কনস্ট্রাকশনঃ

priority_queue বেশ কয়েকভাবে বানানো যায়, বাই ডিফল্ট এটা ম্যাক্স হিপ ইম্পলিমেন্ট করে আর এলিমেন্ট কম্পেয়ার (ছোট-বড় বুঝার জন্য) করার জন্য <queue> এর less<type_t> ক্লাস ব্যাবহার করে। চাইলে এটাকে অন্য কোন কন্টেইনার যেমন vector কে ইন্টারনাল কন্টেইনার হিসাবে ডিফাইন করে দেওয়া যায়, আর বিল্ট ইন বা ওভারলোডেড টেমপ্লেট টাইপ ছাড়াও এটা বানানো যায়, সেক্ষেত্রে ডাটার নিজের ডিফল্ট কম্পেয়ারিজন ক্লাস ব্যবহার করতে হবেঃ < কে ওভারলোড করে, অথবা এক্সপ্লিসিট কম্পেয়ারিজন ক্লাসে () কে ওভারলোড করে। আর, মিন হিপ বানানোও সহজ, <queue> এ ডিফাইন করা greater<type_t> ক্লাস ব্যবহার করে খুব সহজেই করা যায়। যেমনঃ

```
1  // constructing priority queues
2  #include <iostream>
3  #include <queue>
4  using namespace std;
5
6  class mycomparison {
7      bool reverse;
8  public:
   mycomparison(const bool& revparam=false) { reverse =
   revparam; }
```

```
    bool operator() (const int& lhs, const int&rhs) const {
        if (reverse) return (lhs>rhs);
        else return (lhs<rhs);
    }
};


int main () {
    int myints[]= {10,60,50,20};

    // default construction
    priority_queue<int> first;
    priority_queue<int> second (myints,myints+3);

    // using greater<> to create min heap
    priority_queue< int, vector<int>, greater<int> > third (myints,myints+3);

    // using "mycomparison" comparison class
    priority_queue< int, vector<int>, mycomparison > fourth;

    typedef priority_queue<int,vector<int>,mycomparison> mypq_type;
    mypq_type fifth (mycomparison());
    mypq_type sixth (mycomparison(true));

    return 0;
}
```

কমপ্লেক্সিটিঃ ইনিশিয়াল এলিমেন্টের সাপেক্ষে লিনিয়ার, ইনিশিয়ার এলিমেন্ট অ্যাসাইন না করলে কন্সট্যান্ট।

পুশ, পপ, টপঃ

priority_queue এর এলিমেন্ট কে অ্যাকসেস করার জন্য ৩ টা ফাংশন ডিফাইন করা আছে, push(), pop() আর top(). কিউতে কোন এলিমেন্ট অ্যাড করতে চাইলে push() ফাংশনটা ব্যবহার করতে হয়, top() হিপের বর্তমান রুটকে রিটার্ন করে, আর pop() সেটা ট্রি থেকে ডিলিট করে। অর্থাৎ top() আর pop() একত্রে ম্যাক্স হিপের জন্য extract_max() আর

মিন হিপের জন্য extract_min() [see: Introduction To Algorithms — CLRS — MIT Press] এর কাজ করে। নিচে উদাহরণ দেওয়া হলঃ

```
// priority_queue::push/pop/top
#include <iostream>
#include <queue>
#include <ctime>
#include <cstdlib>
using namespace std;

int main () {
    priority_queue<int> mypq;
    srand(time(NULL));

    cout << "Pushing some random values...\n";
    for(int n, i = 0; i < 10; i++) {
        n = rand();
        cout << " " << n;
        mypq.push(n);
    }
    cout << endl;

    cout << "Popping out elements...\n";
    while (!mypq.empty()) {
        cout << " " << mypq.top();
        mypq.pop();
    }
    cout << endl;
    return 0;
}
```

কমপ্লেক্সিটিঃ push() লগারিদমিক, top() কন্সট্যান্ট, কিন্তু pop() লগারিদমিক।

কিউ এর সাইজ চেক করাঃ

priority_queue এর বর্তমান এলিমেন্ট কতগুলা আছে বা, কিউ খালি কিনা এটা টেস্ট করার জন্য ২ টা ফাংশন ডিফাইন করা আছে, size() আর empty(). কিউ থেকে top() আর pop() ব্যবহার করার আগে অবশ্যই কিউ খালি কিনা চেক করা উচিৎ, তা না হলে রান টাইম এরর জেনারেট হতে পারে। আর, কিউ খালি কিনা এটা empty() মেথড দিয়েই চেক করা উচিত, soze()==0 দিয়ে নয়। STL এর সব কন্টেইনার ক্লাসেই এদের ব্যবহার একি রকমঃ

```
1  // priority_queue ::size/empty
2  #include <iostream>
3  #include <queue>
4  using namespace std;
5
6  int main () {
7      priority_queue<int> myints;
8      cout << "0. size: " << (int) myints.size() << endl;
9
10     for (int i=0; i<10; i++) myints.push(i);
11     cout << "1. size: " << (int) myints.size() << endl;
12
13     myints.pop();
14     cout << "2. size: " << (int) myints.size() << endl;
15
16     while(!myints.empty()) myints.pop();
17     cout << "3. size: " << (int) myints.size() << endl;
18
19     return 0;
20 }
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ব্যবহারঃ

বাইনারি হিপ তৈরিতে, ডায়াকস্ট্রা আর প্রিম অ্যালগরিদমে ব্যবহার করা হয়, টেমপ্লেট ক্লাস হওয়ায় যে কোন ডাটা টাইপের জন্য খুব দ্রুত কোড ইমপ্লিমেন্ট করা যায়।
বিস্তারিতঃ http://www.cplusplus.com/reference/stl/priority_queue/
বিভাগ:ডাটা স্ট্রাকচার, প্রোগ্রামিংট্যাগসমূহ:c++, data structure, priority queue, programming, stl

C++ STL :: queue

জুলাই 30, 2010zobayer2009১টি মন্তব্য

কিউঃ

আগে আসলে আগে পাবেন, এই রকমের একটা ডাটা স্ট্রাকচার হল কিউ, যাকে আমরা বলি ফার্স্ট ইন ফার্স্ট আউট (FIFO)। এটা C++ STL এর সবচেয়ে বেশি ব্যবহৃত কন্টেইনার ক্লাস। queue এর সামনের দিক থেকে ডাটা এক্সট্র্যাক্ট করা হয়, আর ইনসার্ট করা হয় তার বিপরীত দিক থেকে। তাই, যে সব STL কন্টেইনার push_back() আর pop_front() সাপোর্ট করে সেগুলো দিয়ে কিউ ইমপ্লিমেন্ট করা যায়, যেমন list আর deque. অবশ্য queue এর ডিফল্ট কন্টেইনার হল deque, যদি কিছু বলে না দেয়া হয়। stack এর মত queue ও একটা অ্যাডাপ্টার ক্লাস।
queue ব্যবহার করতে চাইলে std <queue> হেডারটা প্রোগ্রামে ইনক্লুড করতে হবেঃ

```
1  #include <queue>
2  using namespace std;
```

কন্সট্রাকশনঃ

অন্যান্য কন্টেইনার ক্লাসের মত queue এর সাধারণ কন্সট্রাক্টর queue< type_t > myqueue; এই ধরনের। এছাড়া এক্সপ্লিসিট কন্টেইনার ডিক্লেয়ার করেও queue কন্সট্রাক্ট করা যায়, যেমনঃ

```
1  // constructing queues
2  #include <deque>
3  #include <list>
4  #include <queue>
5  using namespace std;
6
7  int main ()
8  {
9      deque< int > mydeck(3,100); // deque with 3 elements
10     list< int > mylist(2,200); // list with 2 elements
11
12     // implicit declaration
13     queue< int > first; // empty queue
14     queue< int > second(mydeck); // from a mydeck
15     queue< int > third(second); // from another queue
16
17     // explicit declaration
18     queue< int, list< int > > fourth; // empty queue
19     queue< int, list< int > > fifth(mylist); // from mylist
20     queue< int, deque< int > > sixth; // empty queue
21     queue< int, deque< int > > seventh(mydeck); // from mydeck
```

```
20
21      // initialization with operator=
22      queue< int > eighth = first; // initialized with first
23
24      return 0;
25  }
26
27
```

কমপ্লেক্সিটিঃ কন্সট্রাক্টরের সাপেক্ষে কন্সট্যান্ট, অর্থাৎ কন্টেইনারের উপরে নির্ভর করে।

পুশ আর পপঃ

queue এ ডাটা ইন্সার্ট আর এক্সট্র্যাক্ট করার জন্য ২ টা মেম্বার ফাংশন আছে, push() আর pop()। push() এর কাজ হল queue এর শেষে এলিমেন্ট ইন্সার্ট করা আর pop() এর সাহায্যে queue এর সামনে থেকে কোন এলিমেন্ট বের করে দেয়া। যেমনঃ

```
1   // queue::push/pop
2   #include <iostream>
3   #include <queue>
4   using namespace std;
5
6   int main()
7   {
8       queue< int > Q;
9
10      // push 5 integers
11      for(int i=1; i<=5; i++) Q.push(i);
12
13      // pop 5 integers
14      // will be popped in the same order they were pushed
15      for( ; !Q.empty() ; )
16      {
17          cout << Q.front() << endl;
            Q.pop();
        }
```

```
18
19      return 0;
20  }
21
22
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ফ্রন্ট আর ব্যাক

queue তে এলিমেন্ট এক্সেসের জন্য ২ টা ফাংশন হল front() আর back(); front() দিয়ে queue এর ফার্স্ট এলিমেন্ট এক্সেস করা যায়, আর back() দিয়ে লাস্ট ইন্সার্ট করা এলিমেন্ট কে পাওয়া যায়। front() আর back() এর এলিমেন্ট কে স্ট্যান্ডার্ড অপারেটর গুলর সাহায্যে মডিফাই করা যায়। যেমনঃ

```
1   // queue::front/back
2   #include <iostream>
3   #include <queue>
4   using namespace std;
5
6   int main ()
7   {
8       queue<int> myqueue;
9
10      myqueue.push(77);
11      myqueue.push(16);
12      cout << "myqueue.front() = " << myqueue.front() << endl;
13      cout << "myqueue.back() = " << myqueue.back() << endl;
14
15      // modify front element
16      myqueue.front() -= myqueue.back();
17      cout << "myqueue.front() is now " << myqueue.front() << endl;
18
19      // modify back element
20      myqueue.back() += myqueue.front();
      cout << "myqueue.back() is now " << myqueue.back() << endl;
```

```
    return 0;
}
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

কিউ কি খালি?

queue এ এই মুহূর্তে কত গুলা এলিমেন্ট আছে সেটা জানা যায় size() ফাংশনের মাধ্যমে, আর empty() ফাংশন টা boolean, queue খালি থাকলে true দেয়, না হলে false; queue খালি কি না, সেটা চেক করা হয় empty() দিয়ে, কখনই size() এর ভ্যালু ০ কিনা এটা দেখে queue খালি কিনা, সেই টেস্ট করা উচিৎ না, আর queue তে pop() করার আগে আবশ্যই দেখে নিতে হবে queue এ কোন এলিমেন্ট আছে কিনা, তা না হলে run-time error হতে পারে। নিচের কোডে size() আর empty() এর প্রয়োগ দেখানো হলঃ

```
// queue::size/empty
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
    queue<int> Q;

    // just push some elements
    for(int i = 0; i < 5; i++) Q.push(i*i);

    cout << "Queue has " << Q.size() << " elements" << endl;

    Q.pop(); // not a good way, first check for Q.empty()

    if(!Q.empty()) Q.pop(); // this is the porper way

    while(!Q.empty()) Q.pop(); // pop all element

    if(Q.size()==0) cout << "Q is empty" << endl; // not a good way
```

```
21      if(Q.empty()) cout << "Q is empty" << endl; // this is the
proper way
22
23      return 0;
24  }
25
```

কমপ্লেক্সিটিঃ কন্সট্যান্ট।

ব্যবহারঃ

FIFO ডাটা স্ট্রাকচার হিসাবে, BFS ট্রাভার্সাল, বিভিন্ন গ্রাফ এলগরিদমে queue ইম্পলিমেন্ট করা হয়।
বিস্তারিতঃ http://www.cplusplus.com/reference/stl/queue/
বিভাগ:ডাটা স্ট্রাকচার, প্রোগ্রামিংট্যাগসমূহ:c++, data structure, programming, queue, stl

C++ STL :: stack

জুলাই 29, 2010zobayer200912 comments

স্ট্যাকঃ

STL কন্টেইনারদের মধ্যে সম্ভবত সবচেয়ে সিম্পল ডাটা স্ট্রাকচার হল stack, এটা একটা লাস্ট ইন ফার্স্ট আউট (LIFO) ডাটা স্ট্রাকচার, মানে হল যে সবার শেষে আসবে, সে সবার আগে ভাগবে... সোজা কথায় এই কন্টেইনারের শুধুমাত্র একটা দিকেই ডাটা ইন্সার্ট বা এক্সট্র্যাক্ট করা হয়। আর STL এ stack তার ডিফল্ট ইন্টার্নাল ডাটা স্ট্রাকচার হিসাবে ব্যবহার করে STL এরই deque কন্টেইনার, তবে চাইলে vector বা list ও ব্যবহার করা যেতে পারে। যে সব কন্টেইনার push_back() আর pop_back() মেথড ২ টা সাপোর্ট করে সেগুলোকেই stack এর কন্টেইনার ক্লাস হিসাবে ব্যবহার করা যায়। stack আসলে একটা অ্যাডাপ্টার ক্লাস, অর্থাৎ, এটা তৈরি করা হয় এর ইন্টারনাল কন্টেইনারের স্পেসিফিক কিছু ফাংশনকে এলিমেন্ট একসেসের অনুমতি দিয়ে।
stack ব্যবহার করতে চাইলে সর্বপ্রথম কাজটা হল std <stack> হেডারটা প্রোগ্রামে ইনক্লুড করাঃ

```
1   #include <stack>
2   using namespace std;
```

কন্সট্রাক্টরঃ

অন্যান্য STL কন্টেইনার ক্লাসের মত stack এরও সাধারণ কন্সট্রাক্টর stack< type_t > myStack; এই রকমের, তবে আরো অনেকভাবে ডিক্লেয়ার করা যায়। যেমনঃ

```
1   // constructing stacks
2   #include <list>
3   #include <vector>
4   #include <deque>
5   #include <stack>
6   using namespace std;
7
    int main ()
```

```
{
    // using default container deque

    stack< int > first; // empty stack
    deque< int > mydeque(3, 100); // deque with 3 elements
    stack< int > second(mydeque); // from mydeque
    stack< int > third(second); // from another stack second

    // explicit container declarations

    stack< int, deque< int > > fourth; // empty stack using deque
    deque< int > newdeque(10, 100); // deque with 10 elements
    stack< int, deque< int > > fifth(newdeque); // from newdeque

    stack< int, vector< int > > sixth;  // empty stack using vector
    vector< int > myvector(2, 200); // vector with 2 elements
    stack< int, vector< int > > seventh(myvector); // from myvector

    stack< int, list< int > > eighth; // empty srack using list
    list< int > mylist(4, 100); // list with 4 elements
    stack< int, list< int > > ninth(mylist); // from mylist

    // can refer to some other stack
    stack< int > tenth = first; // declaration time initialization

    return 0;
}
```

কমপ্লেক্সিটিঃ কন্টেইনার কন্সট্রাকশনের সাপেক্ষে কন্সট্যান্ট, অতএব কি ধরণের কন্টেইনার ব্যবহার করা হচ্ছে তার উপরে নির্ভর করে।

এলিমেন্ট একসেসঃ

stack ক্লাসের এলিমেন্ট গুলাকে একসেস করার জন্য ৩ টা মেম্বার ফাংশন আছেঃ

1. top()
2. push()
3. pop()

push() ফাংশনটার কাজ stack এর শেষে কোন এলিমেন্ট ইন্সার্ট করা, আর pop() দিয়ে লাস্ট এলিমেন্ট টা বের করে দেয়া। top() এর সাহায্যে কারেন্টলি stack এ সবার উপরের এলিমেন্ট কে পাওয়া যায়। top() এলিমেন্টকে স্ট্যান্ডার্ড অপারেটরদের সাহায্যে মডিফাই করা যায়।

আর, stack এর এলিমেন্ট কাউন্ট করার জন্য ২ টা ফাংশন আছেঃ

1. size()
2. empty()

size() ব্যবহার করে জানতে পারি এই মুহূর্তে stack এ কতগুলো এলিমেন্ট আছে, আর empty() একটা boolean ফাংশন, stack খালি থাকলে এটা true দেয়, না হলে false, stack এ pop() মেথডটা ব্যবহার করতে চাইলে আগে অবশ্যই চেক করে নিতে হবে stack এ কিছু আছে কিনা, তা না হলে run-time error হতে পারে।

নিচে এই ফাংশন গুলার কাজ দেখানো হলঃ

```
1  // stack::push/pop/top/size/empty
2  #include <iostream>
3  #include <stack>
4  using namespace std;
5
6  int main ()
7  {
8      stack< int > mystack;
9      // lets push and pop some values
10     for (int i=0; i<5; i++) mystack.push(i);
11
12     cout << "Popping out elements...";
13     while (!mystack.empty())
14     {
15         cout << " " << mystack.top();
16         mystack.pop();
17     }
18     cout << endl;
19
20     // changing the value of top
```

```
19      mystack.push(100);
20      mystack.top() += 1000;
21      cout << "Now top is: " << mystack.top() << endl;
22      mystack.pop();
23
24      // not a good way to check if stack is empty
25      if(mystack.size() == 0) cout << "Stack is empty" << endl;
26      // better we do this
27      if(mystack.empty()) cout << "Stack is empty" << endl;
28
29      return 0;
30  }
31
32
```

কমপ্লেক্সিটিঃ প্রতিটা ফাংশনের কম্পলেক্সিটি কন্সট্যান্ট।

ব্যবহারঃ

সাধারণত এক্সপ্রেশন / গ্রামার প্রসেসিং, রিকারসিভ এলগরিদমের নন রিকারসিভ প্রয়োগ, DFS ট্রাভার্সাল, LIFO অপারেশনে stack ব্যবহার করা হয়। STL এর stack একটা টেমপ্লেট ক্লাস, তাই যে কোন ডাটা টাইপের জন্য খুব দ্রুত stack ইম্পলিমেন্ট করা যায় STL ব্যবহার করে।
বিস্তারিতঃ http://www.cplusplus.com/reference/stl/stack/
বিভাগ:ডাটা স্ট্রাকচার, প্রোগ্রামিংট্যাগসমূহ:c++, data structure, programming, stack, stl

Hello World!

নভেম্বর 21, 2009zobayer2009১টি মন্তব্য

এইটা হল এই ব্লগের প্রথম পোস্ট

স্বাগতম ওয়ার্ডপ্রেস ব্লগে.
পি.সি. তে বাংলা ভাল দেখার জন্য যা করতে পার –>

1. সোলায়মান লিপি ফন্টটা ইন্সটল করতে পার এখান থেকে।
2. অভ্র কিবোর্ড ইন্সটল করতে পার এখান থেকে।
3. এর পর ব্রাউজারের অপশন মেনু থেকে বাংলার জন্য ফন্ট হিসাবে সোলায়মানলিপি সেট করে দাও। যেমন ফায়ারফক্সে এটা পাওয়া যাবেঃ Tools -> Options -> Content Tab এ গিয়ে Fonts & Colors সেকশন থেকে Advanced -> Fonts For ফিল্ড থেকে Bengali সিলেক্ট করে নিচে সোলায়মান লিপি সেট করে দিতে হবে।
   কি লেখা যায় এখানে চিন্তা করতেছি...