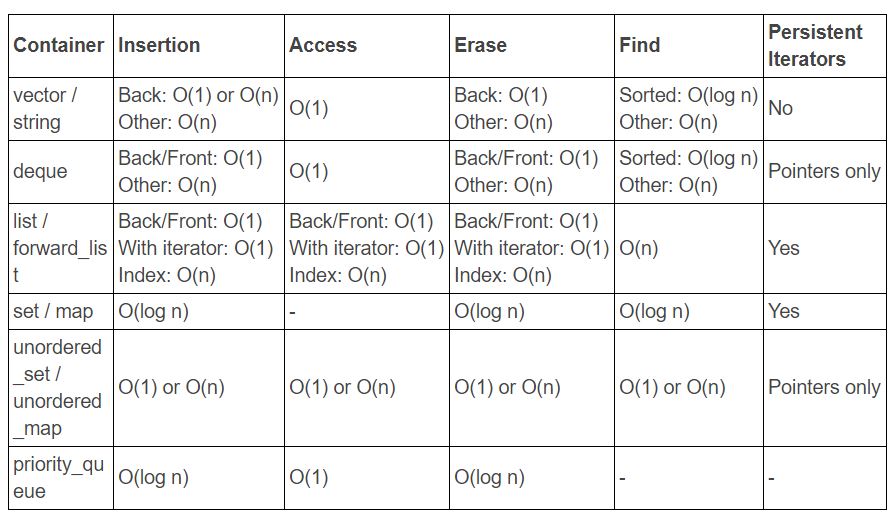
**STL efficiency**

****

**map**

map is always sorted, and it is implemented by binary tree(key, value).

It does not have duplicated key.

keys can not be modified, they are read-only. values can be modified

itr should be assigned before the manipulation.

* When to use:

Usually, main purpose of using map stl container is for efficient search operations and sorted order retrieval.

* Declaration:

map<char,int> mymap;

* Insertion:

mymap.insert(pair<char,int>('a',100));

mymap.insert(make\_pair('z',200));

mymap.insert(itr,make\_pair('b',400));

//reveal if insertion is sucessful, and it will fail because of duplicated key.

pair<map<char,int>::iterator,bool> ret;

ret=mymap.insert(make\_pair('a',1000));

* Show content:

for(itr=mymap.begin();itr!=mymap.end();itr++){

cout << itr->first <<" => " << itr->second <<endl;

} // same as cout << (\*itr).first <<" => " << (\*itr).second <<endl;

* Find:

itr=mymap.find('z');

* Erase:

itr=mymap.find('c');

if(itr!=mymap.end()){

mymap.erase(itr);

}//Be Careful: if iterator does not find content, this will return the end iterator

int iferase = mymap.erase('b');

**Unordered map**

Unordered map’s order is messy, and it’s implemented by hash table(key, value).

It does not have duplicated key.

keys can not be modified, they are read-only. values can be modified

uitr should be assigned before the manipulation.

* When to use:

For efficient search operations, use std::map when you need elements to automatically be sorted. Use std::unordered\_map other times.

* Declaration:

unordered\_map<char,int> u\_map;

* Insertion:

u\_map.insert(pair<char,int>('a',100));

u\_map.insert(make\_pair('z',200));

u\_map.insert(uitr,make\_pair('b',400));

//reveal if insertion is sucessful, and it will fail because of duplicated key.

pair<unordered\_map<char,int>::iterator,bool> ret2;

ret2=u\_map.insert(make\_pair('a',1000));

* Show content:

for(uitr=u\_map.begin();uitr!=u\_map.end();uitr++){

cout << uitr->first <<" => " << uitr->second <<endl; // same as cout << (\*uitr).first <<" => " << (\*uitr).second <<endl;

}

* Find:

uitr=u\_map.find('a');

* Erase:

uitr=u\_map.find('c');

if(uitr!=u\_map.end()){//Be Careful: if iterator does not find content, this will return the end iterator

u\_map.erase(uitr);

}

int ifuerase = u\_map.erase('b');

* Hash table APIs:

//load factor(no. of elements/ no. of buckets)

cout << u\_map.load\_factor()<<endl;

//bucket count

cout << u\_map.bucket\_count()<<endl;

//bucket position of a

cout << u\_map.bucket('a')<<endl;

**Binary file**

* Write a number: n

fstream file;

file.open("binaryfile.bin", ios::out|ios::binary);

file.write((char\*)&n,sizeof(n));

file.close();

* Read a number: r

fstream finC;

finC.open("binaryfile.bin",ios::in|ios::binary);

finC.read((char\*)&r,sizeof(r));

finC.close();

**Split string**

vector<string> split(string str, char delimiter) {

vector<string> outcome;

stringstream ss(str); // Turn the string into a stream.

string substr; //split part

while(getline(ss, substr, delimiter)) {

outcome.push\_back(substr);

}

return outcome;

}

**Regression**

Use the function in CMatrix.h

Regress y on 1, x1,x2,x3,…

Y=[y] , X=[1 x1 x2 x3 …]

Beta\_hat=((X'X)^-1)X'Y

**Shared\_ptr**

Manage memory resource to prevent memory leak(forget to delete) or dangling pointer(forget deleting, undefined behavior).

An object should be assigned to a smart pointer as soon as it is created. Raw pointer should not be used

//Bad example

Dog\* d =new Dog("Tank");//bad idea

shared\_ptr<Dog> p(d); //p.use\_count()==1

cout << p.use\_count() <<endl;

shared\_ptr<Dog> p2(d); //p2.use\_count()==1

cout << p.use\_count() <<endl;

* Declaration:

#include<memory>

shared\_ptr<Dog> p(new Dog("Gun"));

shared\_ptr<Dog> p(d);

shared\_ptr<Dog> p=make\_shared<Dog>("Tank");//faster and saver

* Count number:

p.use\_count()

* Delete memory:

shared\_ptr<Dog> p1=make\_shared<Dog>("Gun");

shared\_ptr<Dog> p2=make\_shared<Dog>("Tank");

//Gun is deleted after any of the following three opeerations

p1=p2;

p1=nullptr;

p1.reset();

* Customized deleter:

shared\_ptr<Dog> p2=shared\_ptr<Dog>(new Dog("Tank"),[](Dog\* p){

cout<< "Custom deleting: ";

delete p;

});

shared\_ptr<Dog> p4(new Dog[3],[](Dog\*p){

delete[] p;

});// All 3 dogs will be deleted when p4 goes out of scope

**Vector**

vector<int> vec2(vec);// copy constructor

vec.clear();//clear

vec.swap(vec2);//swap

itr=find(vec.begin(),vec.end(),5);//find

vec.insert(itr,10); //insert (before the position iterator point to)

itr=min\_element(v.begin(),v.end());//minimum

sort(v.begin(),itr);//sort

* Insert Iterator:

vector<int> vec3={3,4};

vector<int> vec4={1,2,10,13};

vector<int>::iterator it=find(vec4.begin(),vec4.end(),10);

insert\_iterator<vector<int>> i\_itr=inserter(vec4,it);

copy(vec3.begin(),vec3.end(), //source

i\_itr); //destination

cout <<"insert vec3 into vec4:" <<endl;

for(it=vec4.begin();it!=vec4.end();it++){

cout << \*it <<" ";

}

* Insert from end:

itr=v.begin()+2;

vector<int> vec5={9};

vec5.insert(vec5.end(),itr,v.end());

* Reverse:

reverse\_iterator<vector<int>::iterator> ritr;

cout <<"reverse vec4:"<<endl;

for(ritr=vec4.rbegin();ritr!=vec4.rend();ritr++){

cout << \*ritr <<" ";

}

**Visual Studio**

comment many lines: ctrl+k, ctrl+c

uncomment many lines: ctrl+k, ctrl+u

find and replace all you find: Ctrl+H(current file) typing Alt+A