Sira Songpolrojjanakul

Standard Template Library (STL) Container III

list

เป็นตัวเก็บเชิงเส้นแบบหนึ่ง

เติมหัว เติมท้าย ได้อย่าง `vector/queue`

ท่องไปภายในได้ อย่าง `vector`

และสามารถแทรกข้อมูลตรงกลาง `list` ได้

การประกาศ

- #include<list>
- ประกาศเวกเตอร์ list<**type**> **ชื่อ**;
- ฟังก์ชันที่สำคัญพื้นฐาน
- v.push_back(x) // ใส่ x ต่อด้านท้าย
- v.push_front(x) // ใส่ x ต่อด้านหน้า
- v.pop_back() // เอาตัวท้ายสุดออก
- v.pop_front() // เอาตัวหน้าสุดออก
- v.clear() // ลบค่าทุกตัวใน v ออก
- v.size() // return ขนาดของ v
- v.empty() // return true ถ้า v ว่าง

insert(it,v)

ใส่ข้อมูลที่ `it` ใน `O(1)`

```
int main ()
{
  list<int> mylist;
  list<int>::iterator it;

  // set some initial values:
  for (int i=1; i<=5; ++i) mylist.push_back(i); // 1 2 3 4 5

  it = mylist.begin();
  ++it;       // it points now to number 2           ^

  mylist.insert (it,10);                        // 1 10 2 3 4 5

  // "it" still points to number 2                     ^
  mylist.insert (it,2,20);                      // 1 10 20 20 2 3 4 5

  --it;       // it points now to the second 20            ^

  std::vector<int> myvector (2,30);
  mylist.insert (it,myvector.begin(),myvector.end());
                                                // 1 10 20 30 30 20 2 3 4 5
                                                //               ^
  std::cout << "mylist contains:";
  for (it=mylist.begin(); it!=mylist.end(); ++it)
```

mylist contains: 1 10 20 30 30 20 2 3 4 5

```
  return 0;
}
```

insert(it,v)

insert 1 ตัวใช้O(1)
insert N ตัวใช้O(N)

erase(it)

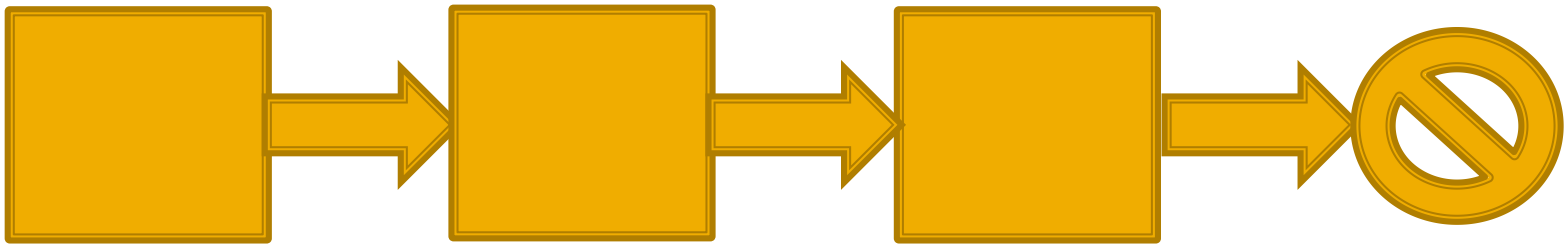
ลบข้อมูล ณ `it`

`return iterator` ตัวถัดไป

```
int main ()
{
  std::list<int> mylist;
  std::list<int>::iterator it1,it2;

  // set some values:
  for (int i=1; i<10; ++i) mylist.push_back(i*10);

                               // 10 20 30 40 50 60 70 80 90
  it1 = it2 = mylist.begin(); // ^^
  advance (it2,6);            // ^                 ^
  ++it1;                      //    ^              ^

  it1 = mylist.erase (it1);   // 10 30 40 50 60 70 80 90
                              //    ^           ^

  it2 = mylist.erase (it2);   // 10 30 40 50 60 80 90
                              //    ^           ^

  ++it1;                      //       ^        ^
  --it2;                      // 10 30 60 80 90
                              //       ^

  mylist.erase (it1,it2);     // 10 30 60 80 90
                              //        ^

  std::cout << "mylist contains:";
  for (it1=mylist.begin(); it1!=mylist.end(); ++it1)
    std::cout << ' ' << *it1;
  std::cout << '\n';
```

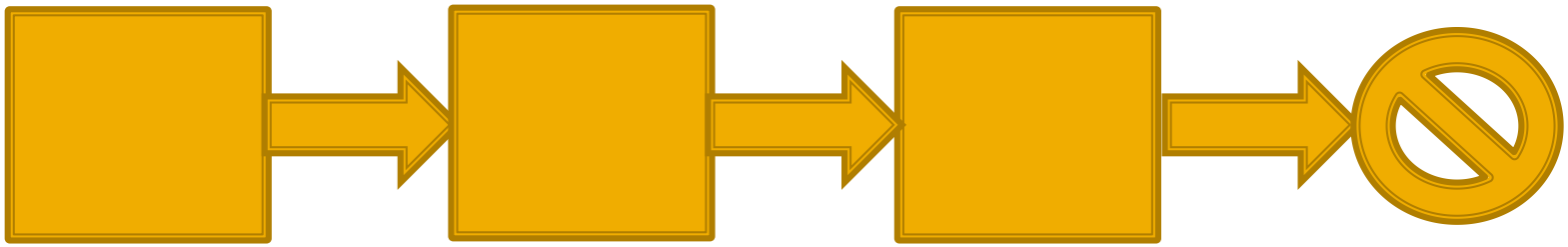
mylist contains: 10 30 60 80 90

รูปแบบการทำงานของ list



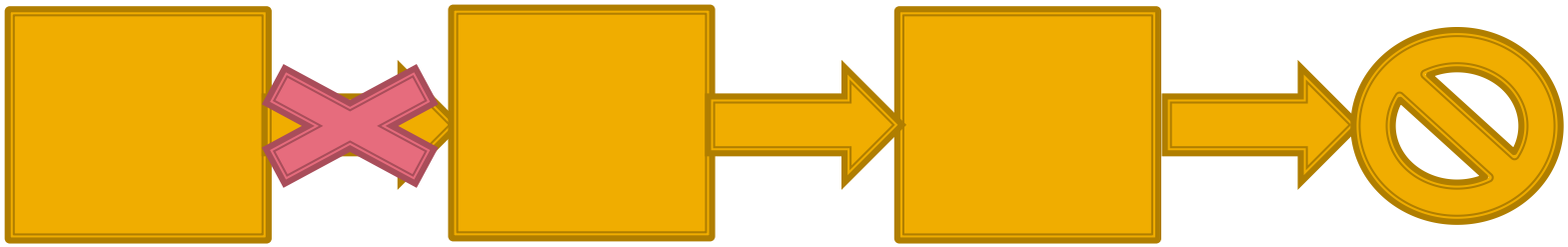
`list` จะเก็บข้อมูล `2` อย่าง คือ ข้อมูล และ ตัวถัดไปของมันคืออะไร ตัวท้ายสุดของลิสต์ มันจะชี้ไป `null` ซึ่งเป็นการบอกว่า ไม่มีตัวถัดไปแล้ว ตัวซ้ายสุด (ตัวแรกใน `list`) เรียกว่า `root`

รูปแบบการทำงานของ list – insert

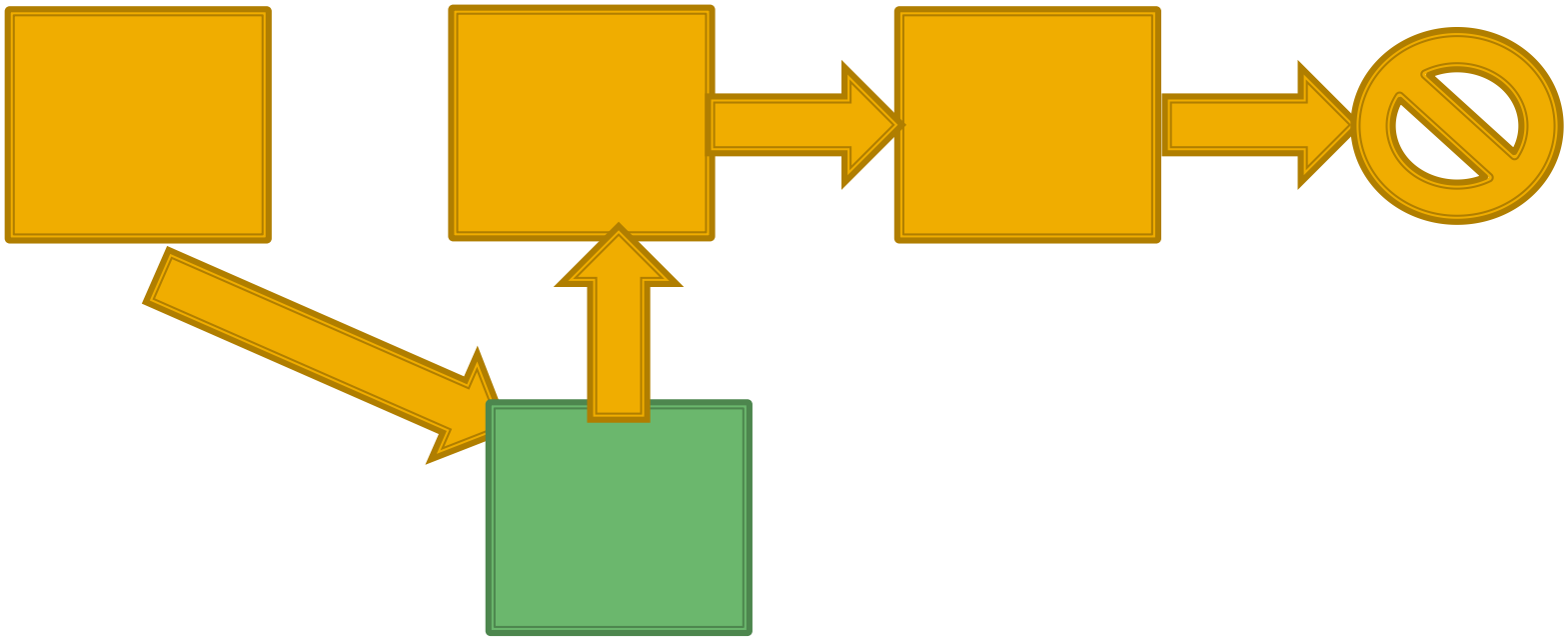


จะแทรกตัวที่ 2

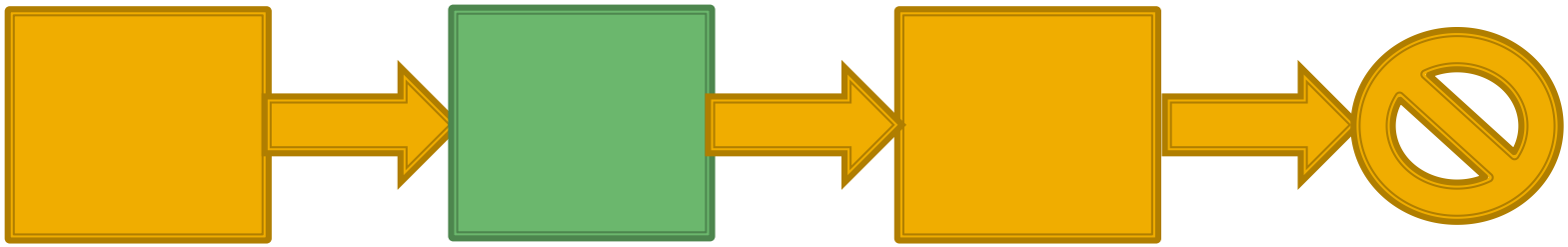
รูปแบบการทำงานของ list – insert



รูปแบบการทำงานของ list – insert

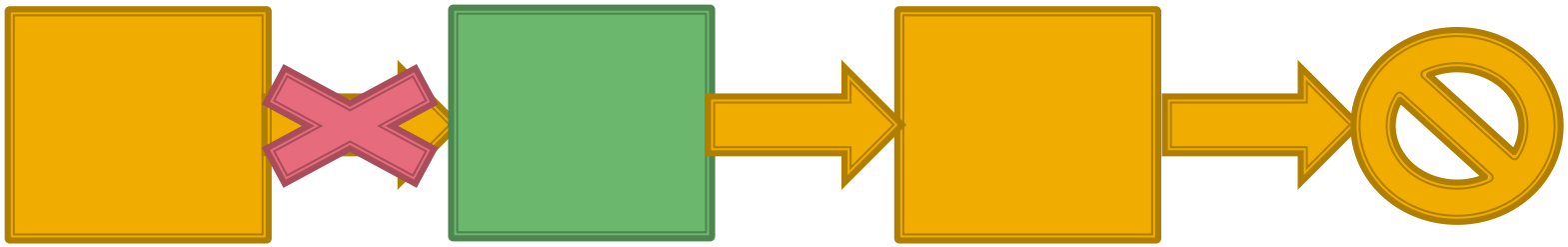


รูปแบบการทำงานของ list – erase

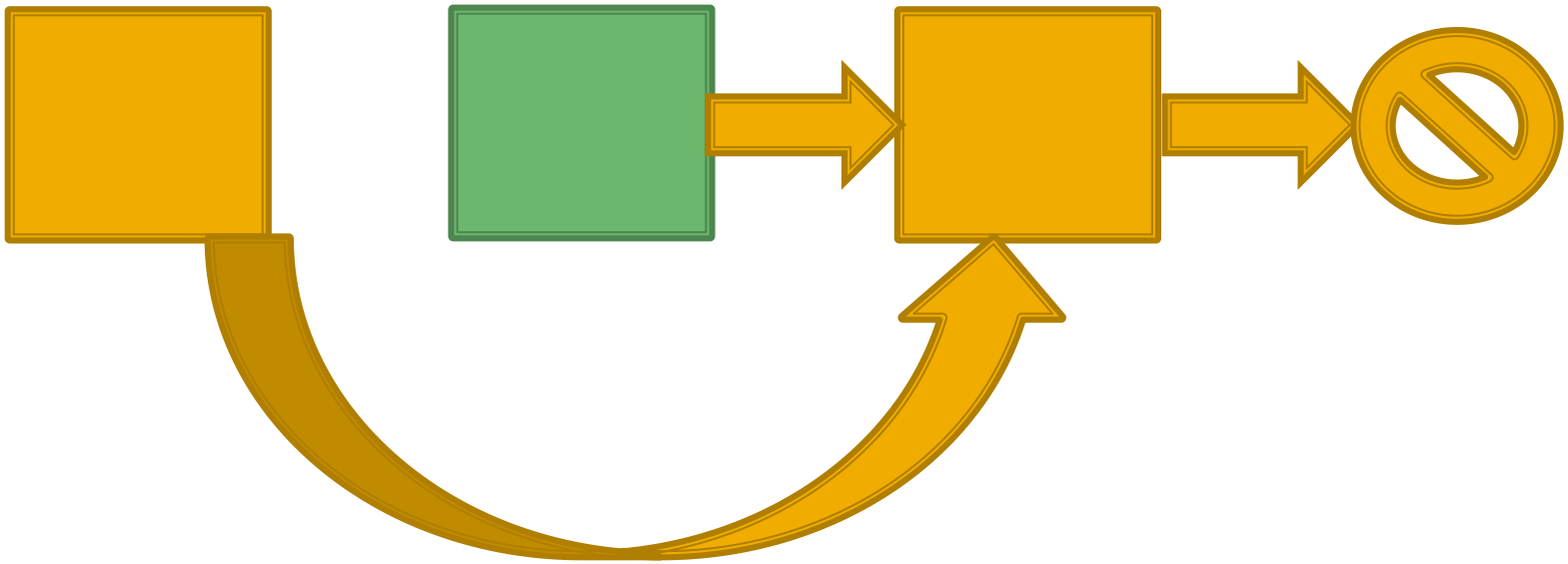


จะลบตัวที่ 2

รูปแบบการทำงานของ list – erase



รูปแบบการทำงานของ list – erase



สังเกตวิธีการทำ

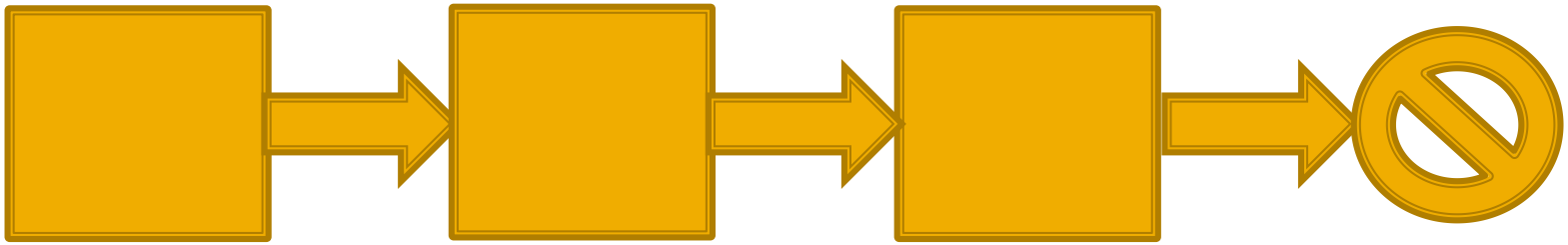
ทั้งการแทรกและลบ ล้วนใช้ O(1) ในการทำงาน

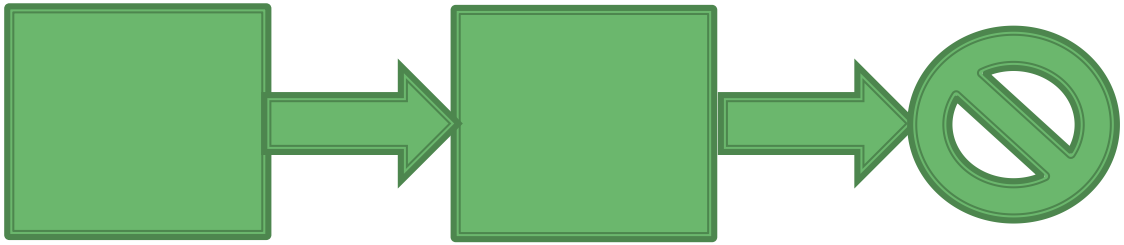
ปกติที่เขาสอน จะแบ่งกรณี แทรกและลบ เป็น 3 แบบ คือ ทำที่ root/ ทำภายใน list/ทำส่วนท้าย

แต่เราสามารถเพิ่ม node root และ node null เพิ่มเข้าไป เพื่อให้ใช้ตัวดำเนินการทำใน list อย่างเดียวพอ

หยุดคิด

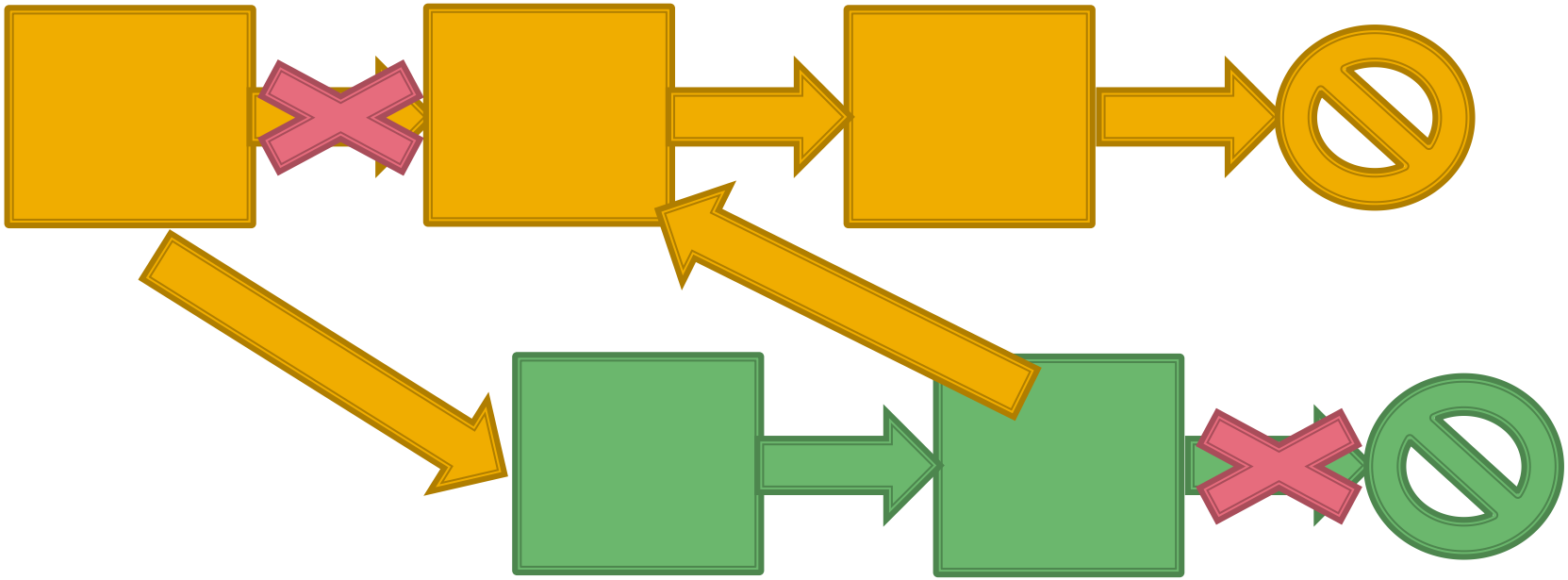
ต้องการแทรก `List` ลงไปใน `List` ใช้เวาเท่าไหร่





เฉลย

O(1)



STL list is dump

แต่ `insert` ใน `STL list` ใช้ `O(N)`

โจทย์หลายๆ ข้อ ใน สสวท. จำเป็นต้องเขียน `list` เอง เนื่องจาก `STL list` ใช้งานได้ไม่ดีพอ

การเขียน `List` เอง จะกล่าวอีกทีในภายหลัง

`list` จะใช้กับโจทย์ประเภทมีแทรกตรงกลางได้

อย่างเช่นข้อ `cline/editor`

ท่องไปใน list

list ไม่มี index เหมือน vector
ดังนั้นการ loop ใน list ต้องใช้ iterator เท่านั้น

บวก/ลบ iterator

เพราะ list ไม่มี index เหมือน vector
ดังนั้น iterator ทำได้แค่ ++ กับ – (เพิ่ม ลด ได้ทีละ 1)
ไม่เหมือนกับ vector ที่ +/- ด้วยค่าอื่นได้

set

ภายใน `set` ข้อมูลจะเรียงจากน้อยไปมากอยู่เสมอ และ ไม่มีการเก็บตัวซ้ำกัน

การประกาศ

- #include<set>
- ประกาศเวกเตอร์ set<**type**> **ชื่อ**;
- ฟังก์ชันที่สำคัญพื้นฐาน
- s.insert(x) // ใส่ x ลงไป
- s.erase(x) // ใส่ x ลงไป
- s.find(x) // return it ที่เจอ
- s.clear() // ลบค่าทุกตัวใน s ออก
- s.size() // return ขนาดของ s
- s.empty() // return true ถ้า s ว่าง

remark

Operation ทุกตัวบนเซต O(log N)
ยกเว้น clear O(N) และ size,empty O(1)

การใช้ set

set มักใช้กับ สิ่งที่**ต้องการหาค่า**ที่มีการ update เพราะถ้าไม่มีการ update อะไร ก็เก็บลง vector->sort แล้วก็ใช้ upper_bound/lower_bound เอา ใน vector ก็มี find() แต่ใช้ O(N) นั่นคือใช้ lower_bound ในการหาค่าใน vector ที่ sort แล้วจะดีกว่า

s.insert(x)

ใส่ x ลงใน s

```
return pair<iterator,bool>
```

iterator ระบุ ตำแหน่งของค่าใหม่ที่จะใส่

bool เป็น true ถ้าเป็นเลขที่ยังไม่มี / false ถ้ามีแล้ว

กรณีมีเลขนั้นแล้ว มันจะไม่ใส่ลงในเซตเพิ่มอีก

s.insert(it,x)

ถ้าตำแหน่งที่ใส่ `x` อยู่ถัดจาก `it` มันจะใช้ประมาณ `O(1)`
ถ้าไม่ ก็จะเป็น `O(log N)`

`return` เหมือนกันกับหน้าที่แล้ว

ปกติใช้ `s.insert(x)` ก็พอแล้ว

ท่องไปใน set

ต้องใช้ `iterator` ในการท่องไปใน `set`

ถ้าพิมพ์ค่าที่ท่องออกมา จะได้ว่าเลขเรียงจากมากไปน้อย

```
int main ()
{
  std::set<int> myset;
  std::set<int>::iterator it;
  std::pair<std::set<int>::iterator,bool> ret;

  // set some initial values:
  for (int i=1; i<=5; ++i) myset.insert(i*10);    // set: 10 20 30 40 50

  ret = myset.insert(20);               // no new element inserted

  if (ret.second==false) it=ret.first;  // "it" now points to element 20

  myset.insert (it,25);                 // max efficiency inserting
  myset.insert (it,24);                 // max efficiency inserting
  myset.insert (it,26);                 // no max efficiency inserting

  int myints[]= {5,10,15};              // 10 already in set, not inserted
  myset.insert (myints,myints+3);

  std::cout << "my
  for (it=myset.b
    std::cout <<
  std::cout << '\

  return 0;
}
```

```
myset contains: 5 10 15 20 24 25
26 30 40 50
```

s.erase(x)

ลบ `x` ใน `s`

`return` จำนวนที่ลบได้

ใช้เวลา `O(log N)`

s.erase(it)

ลบที่ตำแหน่ง `it`

ใช้เวลา `O(1)`

```
int main ()
{
  std::set<int> myset;
  std::set<int>::iterator it;

  // insert some values:
  for (int i=1; i<10; i++) myset.insert(i*10);  // 10 20 30 40 50 60 70 80 90

  it = myset.begin();
  ++it;                                         // "it" points now to 20

  myset.erase (it);

  myset.erase (40);

  it = myset.find (60);
  myset.erase (

  std::cout << "myset contains:";
  for (it=myset.begin(); it!=myset.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

myset contains: 10 30 50

s.find(x)

return ตำแหน่งของ x ใน s
ถ้าไม่มี return s.end()

ใช้เวลา O(log N)

```
// set::find
#include <iostream>
#include <set>

int main ()
{
  std::set<int> myset;
  std::set<int>::iterator it;

  // set some initial values:
  for (int i=1; i<=5; i++) myset.insert(i*10);    // set: 10 20 30 40 50

  it=myset.find(20);
  myset.erase (it);
  myset.erase (myset.find(40));

  std::cout << "myset contains:";
  for (it=myset
    std::cout <
  std::cout << '\n';

  return 0;
}
```

myset contains: 10 30 50

s.lower_bound(x)

return ตำแหน่งที่มีค่า **>=x** ตัวแรก
ถ้าไม่มี `return s.end()`

ใช้เวลา `O(log N)`

s.upper_bound(x)

return ตำแหน่งที่มีค่า >x ตัวแรก
ถ้าไม่มี return s.end()

ใช้เวลา O(log N)

! lower_bound/upper_bound บน set
ใช้ method ของมัน อย่าใช้ function ใน
algorithm

```
// set::lower_bound/upper_bound
#include <iostream>
#include <set>

int main ()
{
  std::set<int> myset;
  std::set<int>::iterator itlow,itup;

  for (int i=1; i<10; i++) myset.insert(i*10); // 10 20 30 40 50 60 70 80 90

  itlow=myset.lower_bound (30);                //       ^
  itup=myset.upper_bound (60);                 //                   ^

  myset.erase(itlow,itup);                     // 10 20 70 80 90

  std::cout << "myset contains:";
  for (std::set
    std::cout <
  std::cout << '\n';

  return 0;
}
```

myset contains: 10 20 70 80 90

!!!

อยากเก็บเลขซ้ำใน `set` ทำอย่างไร

ใช้ `multiset`

ทุกอย่างเหมือนกับ `set` แต่เก็บเลขซ้ำได้

`!` ระวังเรื่อง `erase` ถ้าสั่ง `s.erase(x)` `x`ทุกตัวใน `s` จะถูกลบหมด

หากต้องการลบตัวเดียวใช้ `s.erase(s.find(x))`

`#include<set>` เหมือนเดิม

map

ใช้เป็นอาเรย์ ที่มี index เป็นอะไรก็ได้

ปกติใช้เป็น hash

มีรูปแบบการเก็บแบบ set ซึ่งก็คือ red black tree = balance binary search tree

การประกาศ

- #include<map>
- ประกาศเวกเตอร์ map<**type_index**,**type_value**> **ชื่อ**;
- ฟังก์ชันที่สำคัญพื้นฐาน
- mp.find(x) // return it ที่เจอ index
- ตัวแรก (key) ของ mp เข้าถึงได้โดย mp[x].first ตัวหลัง (data) ใช้ mp[x].second

```
// assignment operator with maps
#include <iostream>
#include <map>

int main ()
{
  std::map<char,int> first;
  std::map<char,int> second;

  first['x']=8;
  first['y']=16;
  first['z']=32;

  second=first;                // second now contains 3 ints
  first=std::map<char,int>();  // and first is now empty

  std::cout << "Size of first: " << first.size() << '\n';
  std::cout << "Size of second: " << second.size() << '\n';
  return 0;
}
```

```
Size of first: 0
Size of second: 3
```

```
// accessing mapped values
#include <iostream>
#include <map>
#include <string>

int main ()
{
  std::map<char,std::string> mymap;

  mymap['a']="an element";
  mymap['b']="another element";
  mymap[
  std::co
  std::co
  std::co
  std::co
  std::co
  return 0;
}
```

```
mymap['a'] is an element
mymap['b'] is another element
mymap['c'] is another element
mymap['d'] is
mymap now contains 4 elements.
```

remark

ปกติinsert ลง map โดยการใส่ค่าลงไปตรงๆ
หากเรียกช่องที่ไม่เคยใส่ค่าไว้ มันจะได้ค่า 0 หรือ null
อย่างไรก็ดี แม้จะเรียกช่องแบบอาเรย์ แต่เวลาในการใช้จริงๆ ของ map คือ O(log N) ไม่ใช่ O(1)
map มี insert/erase ด้วย (เหมือน set) แต่โดยปกติไม่ได้ใช้

remark

กรณีต้องการตรวจสอบว่า มี `index x` ใน `map` หรือเปล่า

ไม่ควรใช้ `if(mp[x]==0)`

เนื่องจากทำให้ใน `map` มีข้อมูลเพิ่มขึ้น ทำให้ช้าขึ้น

ควรใช้ `mp.find(x)==mp.end()` จะดีกว่า

ท่องไปใน map

ใช้ iterator ในการท่องไปใน map
เมื่อพิมพ์ค่าออกมา ค่าจะเรียงตามตัวแรก (key)

remark

key 1 ค่า มี data ได้ค่าเดียว

กรณีต้องการมากกว่าหนึ่งค่า ใช้ multimap

หากใช้ multimap ก็ต้องไล่ค่าในช่องเอง ไม่สามารถเข้าถึงค่าตรงๆ

โดย mp[x] ได้

ปกติ ไม่ได้ใช้ multimap

unordered_map [C++11]

เหมือน `map`

แต่ สามารถเข้าถึงค่าได้**เร็วกว่า** คือ ประมาณ `O(1)`

แต่ ไม่สามารถท่องเข้าไปใน `unordered_map` ได้

ทำได้แค่ จิ้มค่าเท่านั้น

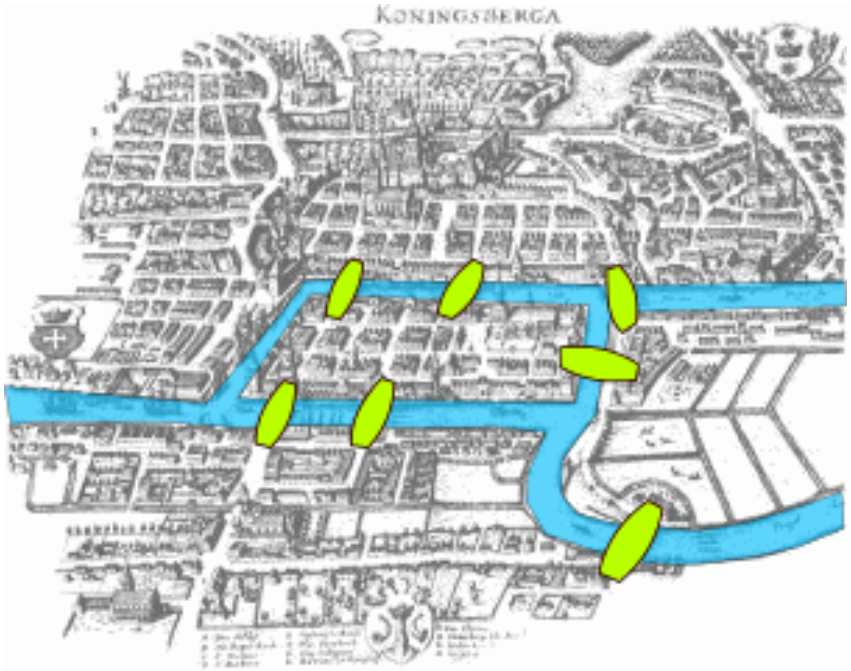
```
#include<unorderd_map>
```

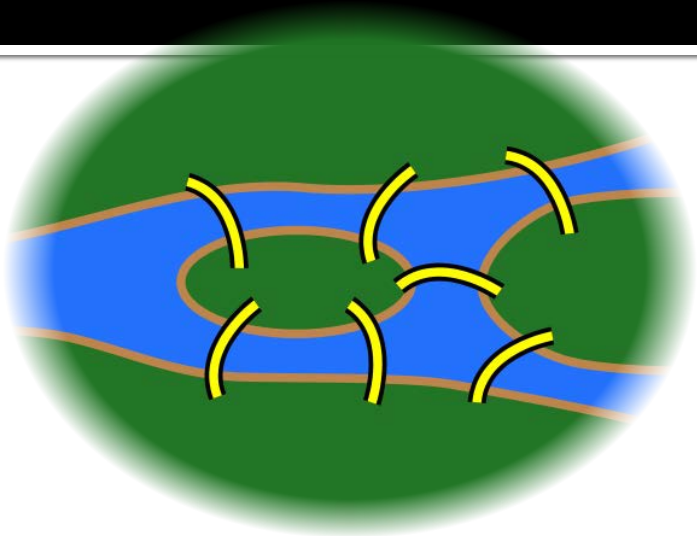
ใช้ได้เฉพาะ `C++11`

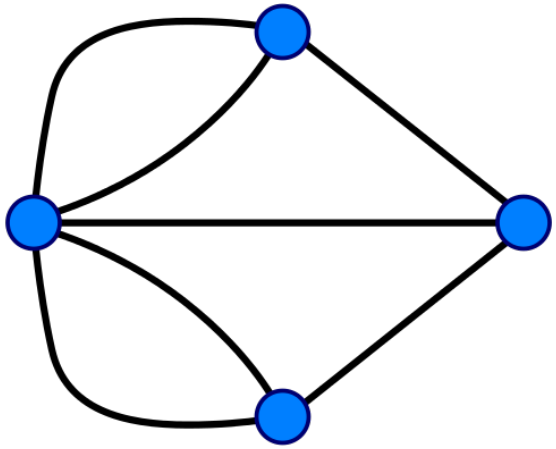
End of STL

To be continued…

ตัวอย่างตอนต่อไป


KONINGSBERGA





จบเนื้อหา

สวัสดี