

A decorative graphic on the left side of the slide, consisting of a network of thin, light-blue lines and small circles, resembling a circuit board or a neural network structure.

2110104: COMPUTER PROGRAMMING

VECTOR

DEPT. OF COMPUTER ENGINEERING
CHULALONGKORN UNIVERSITY

ใช้อาร์เรย์เก็บกลุ่มข้อมูลที่เป็นประเภทเดียวกัน

- ข้อเสีย
 - ต้องจองจำนวนช่องไว้ให้พอ
 - การเพิ่ม การแทรก การลบ ข้อมูลในอาร์เรย์ ต้องจัดการเอง
 - ถ้าจำนวนช่องที่จองไว้ไม่พอ ก็ต้องจัดการขยายขนาดเอง
 - ส่งอาร์เรย์ให้ฟังก์ชันใด ก็ต้องส่งขนาดของอาร์เรย์ไปด้วย
- ใช้ vector ง่ายกว่า

```
int n;  
cin >> n;  
char grades[n];  
for (int i=0; i<n; i++)  
    cin >> grades[i];
```

```
int count_A(char grades[], int n) {  
    int c = 0;  
    for (int i=0; i<n; i++)  
        if (grades[i]=='A') c++;  
    return c;  
}
```

vector : รายการของข้อมูล

- รายการของข้อมูล ตั้งแต่ 0 ตัวขึ้นไป
- ข้อมูลเป็นประเภทเดียวกันหมด
- ข้อมูลแต่ละตัวมีเลข index กำกับ 0, 1, 2, ...
- เพิ่ม / ลบข้อมูลได้ระหว่างใช้งาน
- คล้าย ๆ อาร์เรย์ แต่คล่องตัวกว่า

```
#include <vector>
```

```
vector<int> v = {0,1,1,2,3,5,8,0};  
v[7] = v[5] + v[6];  
v.push_back( v[6]+v[7] );
```

การสร้าง vector

ต้องกำหนดว่า ข้อมูลที่จะเก็บเป็นประเภทใด

```
vector<int>    v1;        // เก็บ int
vector<double> v2;        // เก็บ double
vector<string> v3;        // เก็บ string
vector<vector<int>> v4;    // เก็บ vector ที่เก็บ int
```

ทุกกรณีข้างบนนี้ ได้ vector ว่าง ๆ

`v1.size()`, `v2.size()`, ... มีค่าเป็น 0

การให้ค่าเริ่มต้นกับ vector

```
vector<int>    v1 = {3, 5, 4, 1}; // initializer list
vector<double> v2 = {3.1, 5.2, 4, 3};
vector<string> v3 = {"Yes", "No", "Ok"};
vector<vector<int>> v4 = {{}, {9}, {2,3}};
```

```
vector<int>    v1 (3, 9); // {9, 9, 9}
vector<double> v2 (3, 0); // {0, 0, 0}
vector<char>    v3 (3, 'Y'); // {'Y', 'Y', 'Y'}
vector<char>    v4 = vector<char>(3, 'Y');
```

```
vector<char>    v5 ( v3 ); // copy ข้อมูลจาก v3
```

เปรียบเทียบ vector ได้ (คล้าย tuple)

```
vector<int> v0 = {1,2,3};  
vector<int> v1 = {3,2,1};  
vector<int> v2 = {3,2,1,0};  
vector<int> v3 = {3,2,2};  
  
cout << (v0 != v1) << endl;    // 1  
cout << (v1 != v2) << endl;    // 1  
cout << (v1 < v2) << endl;     // 1  
cout << (v2 < v3) << endl;     // 1
```

การใช้ข้อมูลในช่องต่าง ๆ ของ vector

```
vector<int> v = {9, 3, 4, 1, 3, 1};  
float s = 0;  
for (int i=0, n=v.size(); i<n; ++i) {  
    s += v[i];  
}  
cout << "Average = " << (s/v.size()) << endl;
```

`v.size()` ได้จำนวนข้อมูลที่เก็บใน vector v

`v[i]` คือช่องที่อินเด็กซ์ i ของ vector v

push_back : การเพิ่มข้อมูลต่อท้าย

0 1 1 2 3 5 8 13 21 34 ...

```
vector<int> fibo = {0, 1};  
int N = 1000;  
for (int i=2; i<N; ++i) {  
    fibo.push_back( fibo[i-1] + fibo[i-2] );  
}
```

v.push_back(e)


```
int n;
cin >> n;
pair<string, string> grades[n];
string sid, grade;
for (int i=0; i<n; ++i) {
    cin >> sid >> grade;
    grades[i] = make_pair(sid, grade);
}
```

ใช้ array

ต้องเตรียมอาร์เรย์ที่มี
จำนวนช่องให้พอก่อน

```
int n;
cin >> n;
vector<pair<string, string>> grades;
string sid, grade;
while (n--) {
    cin >> sid >> grade;
    grades.push_back(make_pair(sid, grade));
}
```

ใช้ vector

เวกเตอร์ขยาย
ตามปริมาณข้อมูล

Input

```
9
9999 B+
1111 F
2222 A
3333 B
4444 B
5555 D
6666 C
7777 D
8888 A
```

ดังนั้น ไม่ต้องรู้จำนวนข้อมูลก่อนล่วงหน้าก็ได้

```
vector<pair<string,string>> grades;  
string sid, grade;  
while (cin >> sid) {  
    cin >> grade;  
    grades.push_back(make_pair(sid, grade));  
}
```

Ctrl+Z
or
command+Z



Input

~~9~~

9999 B+

1111 F

2222 A

3333 B

4444 B

5555 D

6666 C

7777 D

8888 A

ตำแหน่งของทีเก็บในเวกเตอร์ : ใช้ index

- ใช้ index ได้เหมือนอาร์เรย์
- เลข index เป็นจำนวนเต็ม 0, 1, 2, ..., v.size()-1
- ใช้ index เพื่อเข้าใช้ข้อมูล v[0], v[1], v[2], ...
- ถ้า index อยู่นอกช่วง จะเกิดอะไร ? ไม่รู้ !!!

```
vector<int> v = {0,1,1,2,3,5,8,11};  
for (int i=0, n=v.size(); i<n; ++i)  
    cout << v[i] << endl;
```

ตำแหน่งของที่เก็บในเวกเตอร์ : ใช้ iterator


- ขอตำแหน่งเริ่มต้น (ซ้ายสุด) : `itr = v.begin()`
- เลื่อนถัดไป 1 ตำแหน่งด้วย `++itr` หรือ `itr++`
- ถอยหลัง 1 ตำแหน่งด้วย `--itr` หรือ `itr--`
- ไปข้างหน้าหรือถอยหลังหลายตำแหน่งด้วย `itr + k` หรือ `itr - k`
- เข้าใช้ข้อมูลด้วย `*itr`
- ถ้าเท่ากับ `v.end()` แสดงว่า ข้อมูลหมดแล้ว

```
vector<int> v = {0,1,1,2,3,5,8,11};  
vector<int>::iterator itr = v.begin();  
while (itr != v.end()) {  
    cout << *itr << endl;  
    ++itr; itr + 1;  
}
```

v.end()

- ไม่ใช่ตำแหน่งของข้อมูลตัวสุดท้าย
- เป็นตำแหน่งที่เลขตัวสุดท้าย

```
vector<int> v = {0,1,1,2,3,5,8,11};  
vector<int>::iterator itr = v.end();  
--itr;  
cout << *itr << endl; // 11
```



ใช้ auto ลดความซับซ้อน

```
vector<int>::iterator itr = v.begin();  
while (itr != v.end()) {  
    cout << *itr << endl;  
    ++itr;  
}
```

```
auto itr = v.begin();  
while (itr != v.end()) {  
    cout << *itr << endl;  
    ++itr;  
}
```

ใช้ auto แล้ว
compiler
จะคิดให้เองว่า
ควรเป็นประเภทอะไร

ใช้ auto ลดความซับซ้อน

```
vector<int> v1 = {2,3,5,7,11,13};  
auto v2 = vector<int>(); // vector<int>  
auto v3(v1);             // vector<int>  
auto e = v3.end();        // vector<int>::iterator  
vector<pair<char,int>> v6 = {{'A',4}, {'B',3}};  
auto p0 = v6[0];          // pair<char,int>  
auto ch = p0.first;       // char  
auto n = p0.second;       // int
```

```
auto v9 = {1, 3, 4};      // ???  
auto p = make_pair("ABC", 3);  
int n = p.first.size();   // ???
```

vector : iterator arithmetic

```
vector<int> v = {2,4,6,0,1,9,7};  
auto itr = v.begin(); // ชี้ตัวแรก *itr ได้ 2  
itr++; // เลื่อนไปชี้ตัวถัดไป *itr ได้ 4  
itr += 4; // เลื่อนไปชี้ 4 ตัวถัดไป *itr ได้ 9  
itr -= 2; // ถอยกลับ 2 ตัว *itr ได้ 0  
int k = itr - v.begin(); // k มีค่าเป็น 3  
int n = v.end() - v.begin(); // เท่ากับ v.size()  
int i = itr + v.begin(); // ผิด
```


ใช้ for loop ในการเปลี่ยนตำแหน่งของ iterator

```
auto itr = v.begin();  
while (itr != v.end()) {  
    ...  
    ++itr;  
}
```

```
for (auto itr = v.begin(); itr != v.end(); ++itr) {  
    ...  
}
```

ใช้ range-based for loop

```
for (auto itr = v.begin(); itr != v.end(); ++itr) {  
    ...  
}
```

```
for (auto itr = v.begin(), end=v.end(); itr != end; ++itr) {  
    int e = *itr;  
    ...  
}
```

```
for (auto e : v) {  
    ...  
}
```

ในแต่ละรอบ จะ copy
ข้อมูลของ v มาเก็บใส่ e
เริ่มจากตัวแรกไปตัวสุดท้ายของ v

ใช้ range-based for loop กับข้อมูลหลายแบบ

ใช้กับ vector

```
vector<int> v = {0,1,1,2,3,5};  
for (int e : v)  
    cout << e << endl;
```

ใช้กับ array

```
int a[] = {0,1,2,3,4,5};  
for (int e : a)  
    cout << e << endl;
```

ใช้กับ initializer list

```
for (int e : {0,1,2,3,4,5})  
    cout << e << endl;
```

ใช้กับ string

```
string s = "CEDT";  
for (char c : s)  
    cout << c << endl;
```

ระวัง : การเปลี่ยนข้อมูลใน loop

```
vector<int> v = {0,1,1,2,3,5};  
for (int e : v)  
    if (e%2==0) e = 9;
```

```
for (auto itr = v.begin(), end=v.end(); itr != end; ++itr) {  
    int e = *itr;  
    if (e%2==0) e = 9;  
}
```

copy ข้อมูล
มาเก็บที่ e

เปลี่ยนที่ e แต่ข้อมูล
ในกลุ่มไม่เปลี่ยน

v ไม่เปลี่ยน

ระวัง : การเปลี่ยนข้อมูลใน loop

```
vector<int> v = {0,1,1,2,3,5};  
for (int e : v)                v ไม่เปลี่ยน  
    if (e%2==0) e = 9;
```

```
int a[] = {0,1,2,3,4,5};  
for (int e : a)                a ไม่เปลี่ยน  
    if (e%2==0) e /= 2;
```

```
string s = "CEDT";  
for (char c : s)                s ไม่เปลี่ยน  
    if (c == 'E') c = '-';
```

ถ้าต้องการเปลี่ยนข้อมูลของกลุ่มใน loop: ใช้ reference

```
vector<int> v = {0,1,1,2,3,5};  
for (int & e : v)  
    if (e%2==0) e = 9;
```

v เปลี่ยน

```
int a[] = {0,1,2,3,4,5};  
for (int & e : a)  
    if (e%2==0) e /= 2;
```

a เปลี่ยน

```
string s = "CEDT";  
for (char & c : s)  
    if (c == 'E') c = '-';
```

s เปลี่ยน

range-based for loop ใช้ได้กับ

ที่เก็บข้อมูล (container) ใน

ที่ใช้ iterator ได้

คือมี begin, end และ iterator ทำ ++ ได้

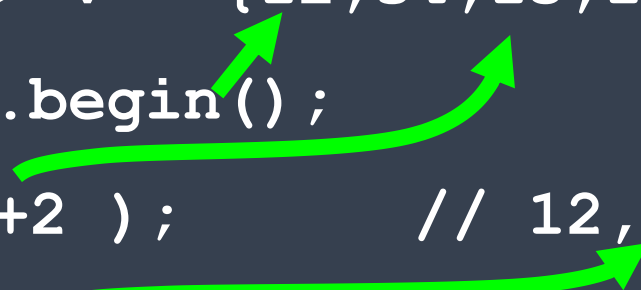
(ใช้ได้กับ set, map ได้ ที่จะอธิบายในบทต่อ ๆ ไป)

กลับมาที่ vector

vector ให้บริการ

- `insert(itr,e)` เพื่อเพิ่มข้อมูล
- `erase(itr)` เพื่อลบข้อมูล
- ใช้ iterator ระบุดำแหน่งที่จะเพิ่มหรือลบข้อมูล

```
vector<int> v = {12,34,15,18};  
auto b = v.begin();  
v.erase( b+2 );           // 12,34,18  
v.insert( b+1, 99 );       // 12,99,34,18
```



itr2 = v.erase(itr1)

- `v.erase(itr1)` คืน iterator ที่ชี้ไปยังข้อมูลหลังตัวที่ถูกลบ

```
vector<int> v = {1,2,3,4,5,6,7};  
auto itr = v.begin();  
while (itr != v.end()) {  
    itr = v.erase(itr);  
    if (itr == v.end()) break;  
    ++itr;  
}
```

ลบตัวเว้นตัว

2 4 6

ตัวอย่าง: erase_all

```
void erase_all(vector<int> &v, int e) {  
    auto itr = v.begin();  
    while (itr != v.end()) {  
        if (*itr == e)  
            itr = v.erase(itr);  
        else  
            ++itr;  
    }  
}  
  
int main() {  
    vector<int> u = {1,2,2,3,1,1,3};  
    erase_all(u, 1);  
    for (auto e : u)  
        cout << e << ' '; // 2 2 3 3  
}
```

ต้องรับ reference
เพราะต้องการ
เปลี่ยนข้อมูลใน v

itr2 = v.insert(itr1, e)

- `v.insert(itr1,e)` คืน iterator ที่ชี้ไปยังข้อมูลใหม่ที่ถูกเพิ่ม

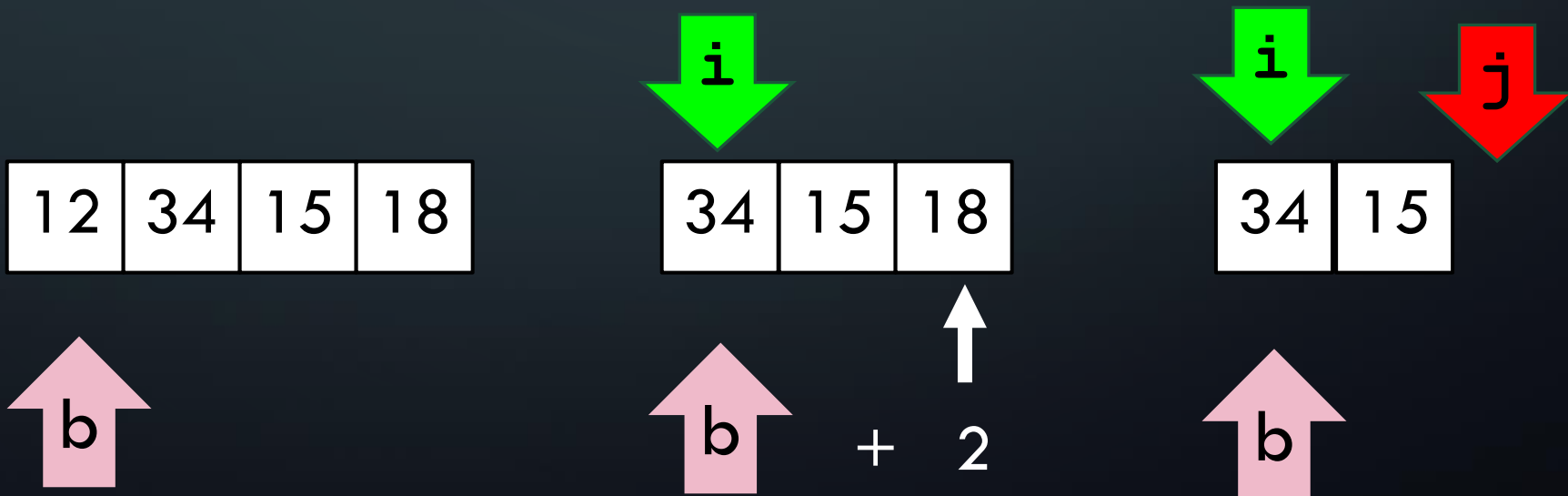
```
vector<int> v;  
auto itr = v.begin();  
for (auto e : {4,3,1,2}) {  
    itr = v.insert(itr, e);  
}
```

2 1 3 4

ข้อมูลใน vector เก็บติดกันในหน่วยความจำ

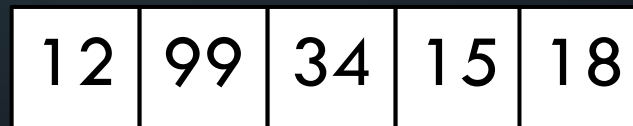
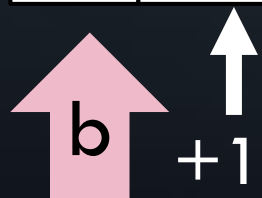
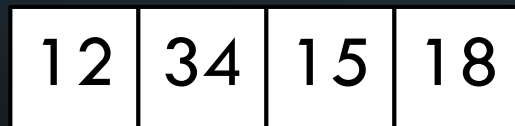
```
vector<int> v = {12,34,15,18};  
auto b = v.begin();  
auto i = v.erase( b );  
auto j = v.erase( b+2 );
```

ลบแล้ว
ตำแหน่งแรก
ไม่เปลี่ยน



ระวัง: begin อาจเปลี่ยนหลัง insert/push_back

```
vector<int> v = {12,34,15,18};  
auto b = v.begin();  
auto i = v.insert( b+1, 99 );  
auto j = v.insert( b+2, 88 ); // ผิด
```



ทำไม ?
จะเรียนในวิชา
โครงสร้างข้อมูล

หลัง insert/push_back

iterator ที่เคยเก็บไว้ อาจใช้ไม่ได้ จึงต้องขอใหม่

```
vector<int> v = {12,34,15,18};  
auto b = v.begin();  
auto i = v.insert( b+1, 99 );  
auto j = v.insert( v.begin()+2, 88 );
```

12	99	88	34	15	18
----	----	----	----	----	----

ตัวอย่าง: insert

เพิ่มข้อมูลใหม่เข้าไปใน vector ที่ข้อมูลเรียงจากน้อยไปมาก
แล้วยังคงเรียงจากน้อยไปมาก หลังเพิ่ม

```
void insert(vector<int> & v, int e) {  
    ...  
}  
  
int main() {  
    vector<int> v = {1,3,4,6,9,10};  
    insert(v, 8);    // 1,3,4,6,8,9,10  
    insert(v, 29);   // 1,3,4,6,8,9,10,29  
    insert(v, -5);   // -5,1,3,4,6,8,9,10,29  
}
```

ตัวอย่าง: insort (แบบง่าย)

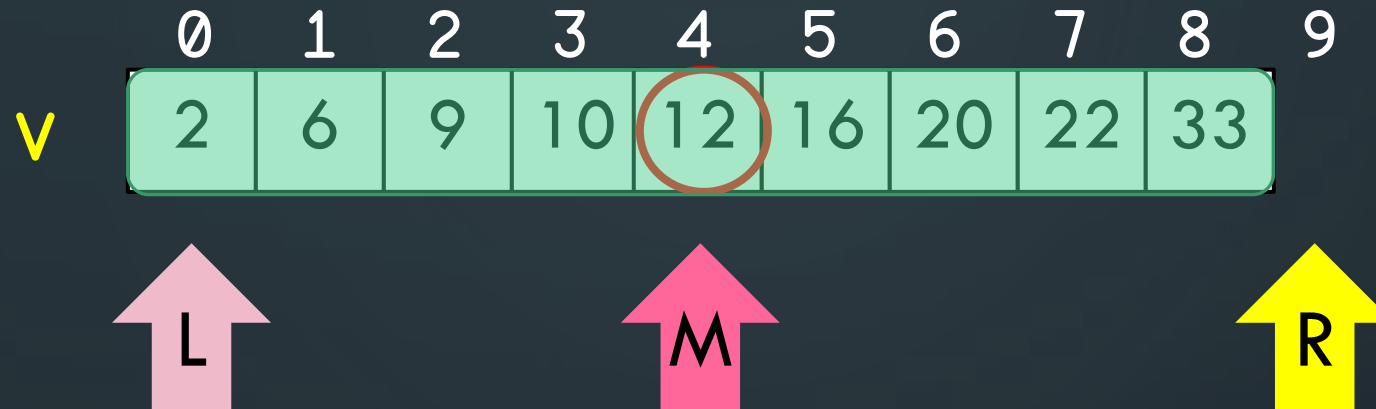
เพิ่มข้อมูลใหม่เข้าไปใน vector ที่ข้อมูลเรียงจากน้อยไปมาก
แล้วยังคงเรียงจากน้อยไปมาก หลังเพิ่ม

```
void insort(vector<int> & v, int e) {  
    auto itr = v.begin();  
    for(auto end=v.end(); itr != end; ++itr) {  
        if (*itr >= e) break;  
    }  
    v.insert(itr, e);  
}
```

ถ้าข้อมูลมาก
แบบนี้ไม่ค่อยเร็ว

1 2 5 7 9 11 e = 8

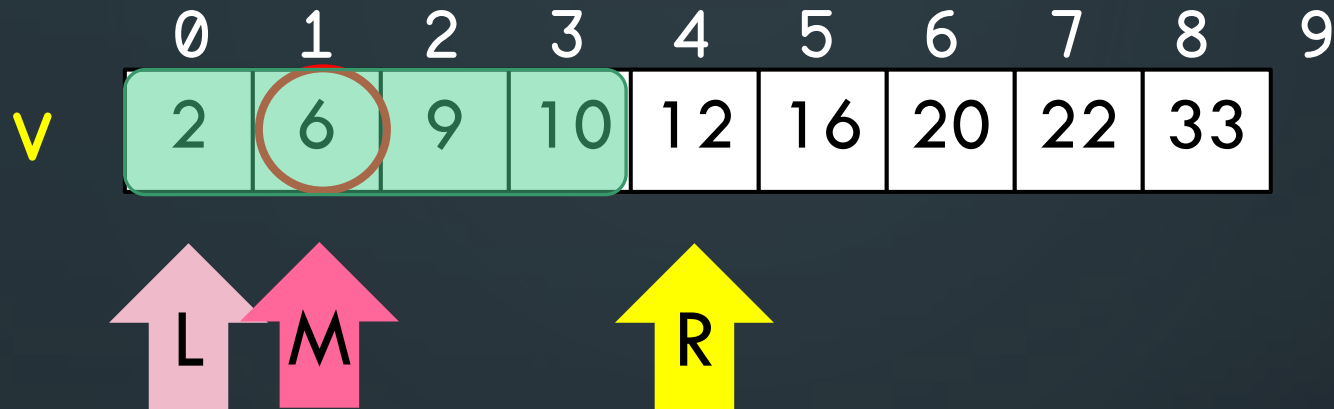
ตัวอย่าง: insert (แบบเร็ว)



เพิ่ม 7

`while (L < R)`

ตัวอย่าง: insert (แบบเร็ว)



เพิ่ม 7

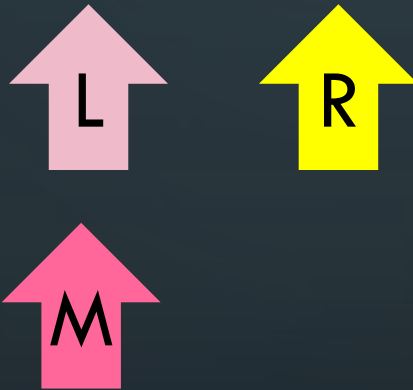
`while (L < R)`

ตัวอย่าง: insert (แบบเร็ว)

v

0	1	2	3	4	5	6	7	8	9
2	6	9	10	12	16	20	22	33	

เพิ่ม 7



while (L < R)

ตัวอย่าง: insert (แบบเร็ว)

v

0	1	2	3	4	5	6	7	8	9
2	6	9	10	12	16	20	22	33	

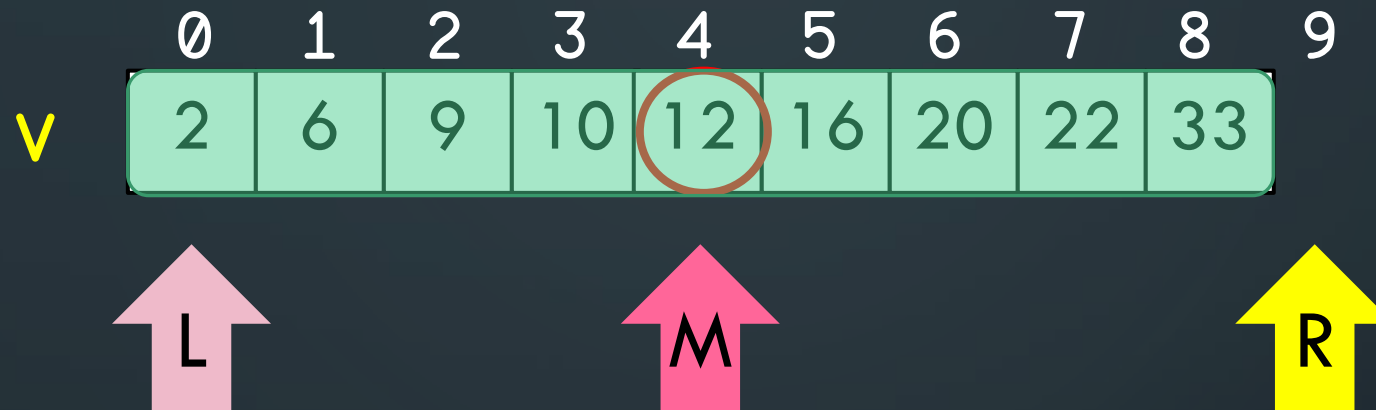
เพิ่ม 7



while [L < R] false

`v.insert(v.begin()+L, 7)`

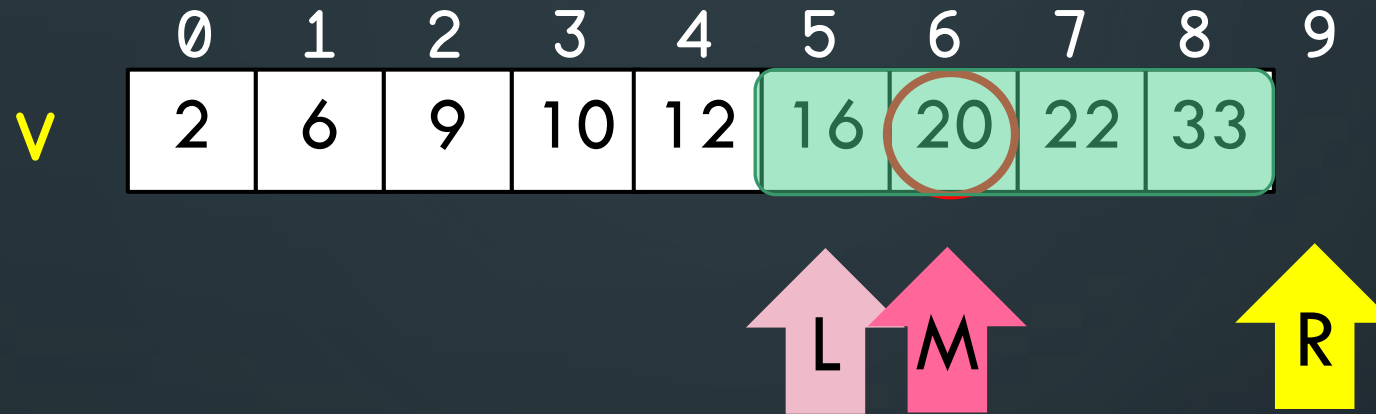
ตัวอย่าง: insert (แบบเร็ว)



เพิ่ม 99

`while (L < R)`

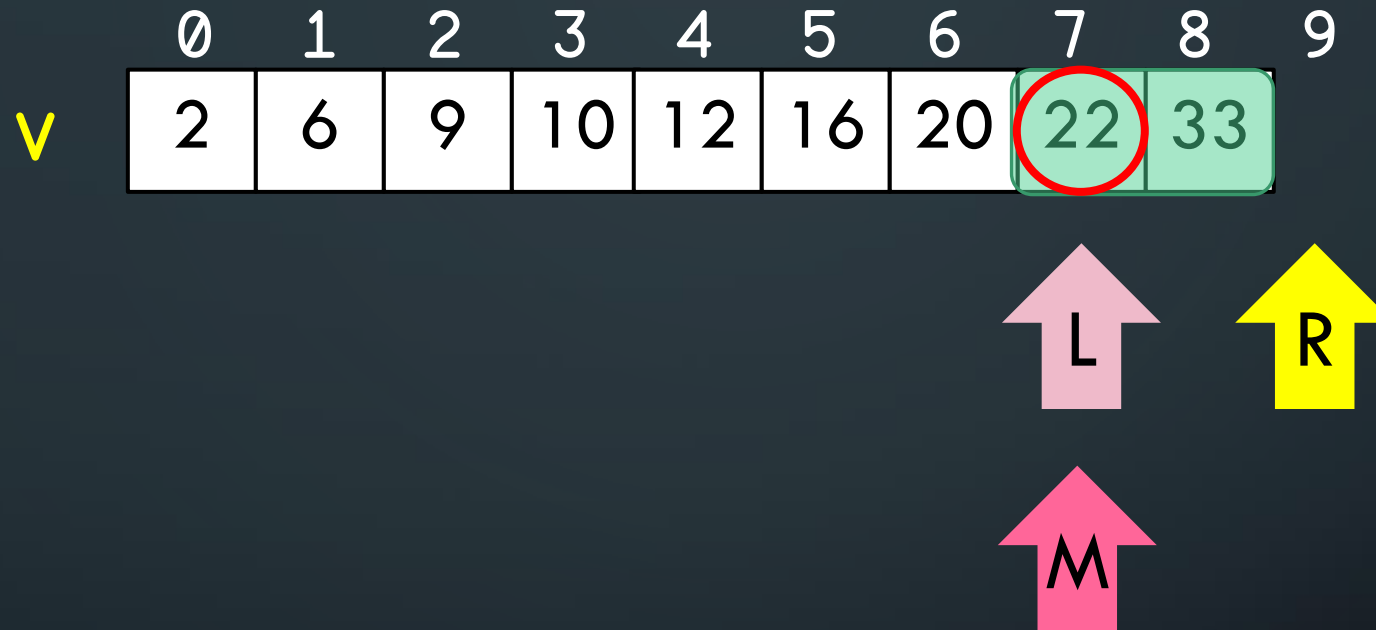
ตัวอย่าง: insert (แบบเร็ว)



เพิ่ม 99

`while (L < R)`

ตัวอย่าง: insert (แบบเร็ว)



เพิ่ม 99

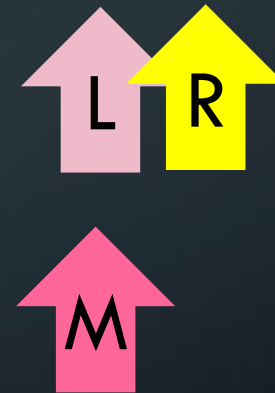
while (L < R)

ตัวอย่าง: insert (แบบเร็ว)

v

0	1	2	3	4	5	6	7	8	9
2	6	9	10	12	16	20	22	33	

เพิ่ม 99



while (L < R)

ตัวอย่าง: insert (แบบเร็ว)

v

0	1	2	3	4	5	6	7	8	9
2	6	9	10	12	16	20	22	33	

เพิ่ม 99



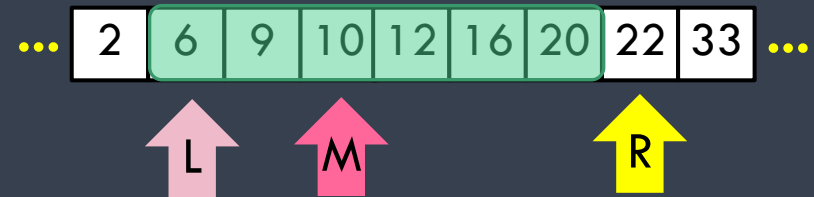
while [L < R]

false

v.insert(v.begin()+L, 99)

ตัวอย่าง: insert (แบบเร็ว)

```
void insert(vector<int> & v, int e) {  
    int mid, left = 0, right = v.size();  
    while (left < right) {  
        mid = (left + right - 1) / 2;    // ***  
        if (e < v[mid])  
            right = mid;  
        else if (e > v[mid])  
            left = mid+1;  
        else  
            left = right = mid;  
    }  
    v.insert(v.begin()+left, e);  
}
```



แต่ละรอบ ข้อมูลจาก
L ถึง R ลดลง
ทีละครึ่ง
ถ้าข้อมูลสั้นตัว
หมุนไม่เกิน 20 รอบ

กลับมาที่ vector

- มีบริการอื่น ๆ
 - `v.pop_back()` ลบตัวท้ายสุดทิ้ง
 - `v.clear()` ลบข้อมูลทั้งหมด เหลือ 0 ตัว
 - `v.empty()` ถ้าไม่มีข้อมูลคืนจริง ไม่ขึ้นคืนเท็จ
 - และอีกหลายบริการ (ที่ขอไม่ครอบคลุมในวิชานี้)

<https://cplusplus.com/reference/vector/vector/>

หาข้อมูลใน vector : คำนวณ index

```
int find(vector<int> & v, int e) {  
    int i = 0;  
    for (int n=v.size(); i<n; ++i)  
        if (v[i] == e) break;  
    return i;  
}
```

หาไม่พบ
คืน v.size()

```
int find(vector<int> & v, int e) {  
    auto itr = v.begin();  
    for (auto end=v.end(); itr != end; ++itr)  
        if (*itr == e) break;  
    return itr - v.begin();  
}
```

หาข้อมูลในช่วงของ vector : คำนวณ index

```
int find(vector<int> & v, int i, int j, int e) {  
    for (; i<j; ++i)  
        if (v[i] == e) break;  
    return i;  
}
```

หาไม่พบ
คืน j

```
int find(vector<int> & v, int i, int j, int e) {  
    auto itr = v.begin()+i;  
    for (auto end=v.begin()+j; itr != end; ++itr)  
        if (*itr == e) break;  
    return itr - v.begin();  
}
```

หาข้อมูลในช่วงของ vector : คำนวณ iterator

```
auto find(vector<int>::iterator beg,
          vector<int>::iterator end, int e) {
    for (; beg != end; ++beg)
        if (*beg == e) break;
    return beg;
}

int main() {
    vector<int> v = {1,1,2,3,4,1,1,1};
    auto b = v.begin();
    auto itr = find(b+2, b+5, 1);
    cout << (itr == b+5); // 1 -> not found
}
```

หาไม่พบ
คืน end

#include <algorithm> : บริการค้นข้อมูล

find(first_itr, last_itr, val)

```
vector<int> v = {3,4,1,5,6,1,1,1,3,1,9,1};  
auto itr = find(v.begin(), v.end(), 1);  
if (itr != v.end())  
    cout << "Found at index " << (itr - v.begin());  
else  
    cout << "Not found";
```

Found at index 2

#include <algorithm> : บริการค้นข้อมูล

`find(first_itr, last_itr, val)`

```
vector<int> v = {3,4,1,5,6,1,1,1,3,1,9,1};  
auto b = v.begin(), e = v.end();  
auto itr = b-1;  
bool found = false;  
while ((itr = find(++itr, e, 1)) != e) {  
    cout << "Found at index " << (itr - b) << endl;  
    found = true;  
}  
if (!found) cout << "Not found";
```

แสดงทุก
index ที่พบ

`#include <algorithm>` : บริการเรียงข้อมูล

`sort(first_itr, last_itr)`

```
vector<int> v = {3,4,1,5,6,7,3,2,3,1,9,0};  
sort(v.begin(), v.end());  
for (auto e : v)  
    cout << e << ' ';
```

0 1 1 2 3 3 3 4 5 6 7 9

ตัวอย่าง: นับจำนวนอักษรตัวพิมพ์ใหญ่

Input	Output
CCCBDDDD BBAAB BADABCC	B : 7 C : 5 A : 4 D : 4

```
chars  C B D A
counts 5 7 4 4
```

```
d      [-5,C), [-7,B), [-4,D), [-4,A)
sort   [-7,B), [-5,C), [-4,A), [-4,D)
```

ตัวอย่าง: นับจำนวนอักขระตัวพิมพ์ใหญ่

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {

    ...

}
```

```
int main() {  
    vector<char> chars;  
    vector<int> counts;  
    string line;  
    getline(cin, line);  
    for (char c : line) {  
        if ('A' <= c && c <= 'Z') {  
            auto itr = find(chars.begin(), chars.end(), c);  
            if (itr == chars.end()) {  
                chars.push_back(c);  
                counts.push_back(1);  
            } else {  
                ++counts[itr - chars.begin()];  
            }  
        }  
    }  
}
```

line CCCBBDDD BBAAB BADABCC
chars C, B, D, A
counts 5, 7, 4, 4

```

int main() {
    vector<char> chars;
    vector<int> counts;
    ...

    vector<pair<int, char>> d;
    for (int i=chars.size()-1; i>=0; --i) {
        d.push_back( make_pair(-counts[i], chars[i]) );
    }
    sort(d.begin(), d.end());
    for (auto p : d)
        cout << p.second << ':' << -p.first << endl;
}

```

chars C, B, D, A
 counts 5, 7, 4, 4
 d [-4,A], [-4,D], [-7,B], [-5,C]
 sort [-7,B], [-5,C], [-4,A], [-4,D]

Output

```

B:7
C:5
A:4
D:4

```

vector : สรุป

- เหมาะกับการเก็บข้อมูลที่ลำดับของข้อมูลมีความสำคัญ
- เพิ่ม/ลบข้อมูลได้ ย้ายที่เก็บได้เอง (ดีกว่าอาร์เรย์)
- `v.push_back(e)` และ `v.pop_back()` ทำงานได้เร็ว
- `v.insert(itr,e)` และ `v.erase(itr)` ใช้เวลาขึ้นกับตำแหน่ง (ต้นเวกเตอร์ช้า ท้ายเวกเตอร์เร็ว)
- iterator วิ่งตามลำดับข้อมูล (ซ้ายไปขวา / ขวามาซ้าย)
- `find(itr1, itr2, e)` ใช้เวลาขึ้นกับตำแหน่งที่พบ
- `sort(itr1, itr2)` เร็ว (เร็วกว่าเรียงเอง)