

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, extending vertically from the top to the bottom.

2110104: COMPUTER PROGRAMMING

SET

DEPT. OF COMPUTER ENGINEERING
CHULALONGKORN UNIVERSITY

ถ้าต้องการเก็บข้อมูลเพื่อค้น ๆ เพิ่ม ๆ ลบ ๆ

- ใช้ vector
 - เข้าใช้ข้อมูลด้วย index -> เร็วมาก
 - เพิ่มที่ท้าย ลบตัวท้าย -> เร็วมาก
 - ค้นข้อมูล -> ช้า (ขึ้นกับว่าหาเจอที่ไหน)
 - ต้องการเรียงข้อมูล -> sort
- ใช้ set
 - เพิ่ม ลบ ค้น ข้อมูลใด ๆ -> เร็ว
 - ใช้ iterator เข้าใช้ข้อมูลได้แบบเรียงลำดับ
 - ข้อมูลไม่มี index กำกับ, ไม่เก็บค่าซ้ำ

set : เซตของข้อมูล

- กลุ่มข้อมูลประเภทเดียวกัน ค่าไม่ซ้ำกัน
- เพิ่ม ลบ ค้นหา ข้อมูลใด ๆ -> เร็ว
- ข้อมูลไม่มี index กำกับ เขียน `s[k]` ไม่ได้
- เข้าใช้ข้อมูลผ่าน iterator จะได้เรียงลำดับ
- เก็บแล้ว เปลี่ยนค่าไม่ได้ (ต้องลบแล้วเพิ่มใหม่)

```
#include <set>
```

```
set<int> s = {3,1,2,6,7};  
s.insert( 5 );  
s.erase( 2 );
```

การสร้าง set

```
set<int>    s1;        // เก็บ int
set<double> s2;        // เก็บ double
set<string> s3;        // เก็บ string
set<vector<int>> s4;    // เก็บ vector ที่เก็บ int
auto s5 = set<int>();
```

ทุกกรณีข้างบนนี้ ได้ set ว่าง ๆ

s1.size(), s2.size(), ... มีค่าเป็น 0

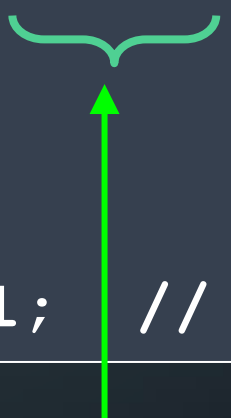
การให้ค่าเริ่มต้นกับ set

```
set<int>      s1 = {3, 5, 4, 1}; // initializer list
set<double> s2 = {3.1, 5.2, 4, 3};
set<string> s3 = {"Yes", "No", "Ok"};
set<vector<int>> s4 = {{}, {9}, {2,3}};
```

```
set<int>      s5( s1 ); // copy ข้อมูลจาก s1
auto         s6( s2 );
```

insert: เพิ่มข้อมูลใน set ไม่ต้องบอกตำแหน่ง

```
set<int> s;  
for (int e : {3,1,4,2,5,5,5,5}) {  
    s.insert(e);  
}  
cout << s.size() << endl; // 5
```



ถ้ามีข้อมูลที่จะเพิ่มอยู่แล้ว ก็ไม่เพิ่มให้

ตัวอย่าง : มีเลขอะไรบ้างในสตริง (ใช้ set)

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

set<char> int count_unique_digits(string t) {
    set<char> d;
    for (char c : t) {
        if ('0' <= c && c <= '9')
            d.insert(c);
    }
    return d;
}
```

ตัวอย่าง : มีเลขอะไรบ้างในสตริง (ใช้ vector)

ถ้าใช้ vector ต้องเสียเวลาค้น

```
#include <vector>
#include <algorithm>
using namespace std;
vector<char> count_unique_digits(string t) {
    vector<char> d;
    for (char c : t) {
        if ('0' <= c && c <= '9')
            if (find(d.begin(), d.end(), c) == d.end())
                d.push_back(c);
    }
    return d;
}
```


set ก็มี iterator ให้ใช้

- s.begin(), s.end(), ++itr, itr++, --itr, itr--
- s.begin() คือตำแหน่งแรก = ตำแหน่งที่มีค่าน้อยสุดใน set
- ถ้าเท่ากับ s.end() แสดงว่า ข้อมูลหมดแล้ว
- เข้าใช้ข้อมูลด้วย *itr
- ทำ itr + k และ itr - k ไม่ได้

```
set<int> s = {3,2,5,11,7,17,13};  
auto itr = s.begin();  
while (itr != s.end()) {  
    cout << *itr << ' ';  
    ++itr;  
}
```

เขียน itr += 1 ไม่ได้

ใช้ range-based for loop ลุขหยบข้อมูล

```
set<int> s = {4,1,2,6,5};  
for (auto itr = s.begin(),end=s.end(); itr!=end; ++itr) {  
    auto e = *itr;  
    ...  
}
```

```
set<int> s = {4,1,2,6,5};  
for (auto e : s) {  
    ...  
}
```

ทำได้เหมือน vector

เพิ่ม itr : เลื่อนไปข้อมูลที่มากขึ้น ๆ ๆ

```
set<int> s = {4,1,2,6,5};  
for (auto itr = s.begin(), end=s.end(); itr!=end; ++itr) {  
    cout << *itr << ' ';    1 2 4 5 6  
}
```

```
set<int> s = {4,1,2,6,5};  
for (auto e : s) {  
    cout << e << ' ';  
}    1 2 4 5 6
```

vector ได้ตามลำดับซ้ายไปขวา
set ได้ตามค่าน้อยไปมาก

เพิ่ม itr : เลื่อนไปข้อมูลที่มากขึ้น ๆ ๆ

```
set<char> s = {'Z', 'A', 'C', 'D', 'B'};  
for (auto e : s) {  
    cout << e << ' ';  
}
```

A B C D Z

```
set<pair<int,char>> s = {{3,'Z'}, {5,'A'}, {3,'C'}};  
for (auto p : s) {  
    cout << '(' << p.first << ','  
        << p.second << ") ";  
}
```

[3,C] [3,Z] [5,A]

ลด itr : เลื่อนไปข้อมูลที่น้อยลง ๆ ๆ

```
set<int> s = {4,1,2,6,5};  
for (auto itr = s.begin(), end=s.end(); itr!=end;){  
    cout << *(itr++) << ' ' ;  
}
```

1 2 4 5 6

```
set<int> s = {4,1,2,6,5};  
for (auto itr = s.end(), beg=s.begin(); itr!=beg;){  
    cout << *(--itr) << ' ' ;  
}
```

6 5 4 2 1

การจัดเก็บข้อมูลภายใน set รองรับการเลื่อนไปตำแหน่ง
ที่ค่ามากขึ้น หรือน้อยลง ได้อย่างรวดเร็ว

ตัวอย่าง: คำน้อยสุด ค่ามากสุดใน set

```
int get_min( set<int> s ) {  
    return *(s.begin());      // *s.begin() ก็ได้  
}  
  
int get_max( set<int> s ) {  
    return *(--s.end());      // *--s.end() ก็ได้  
}
```

การเปรียบเทียบ set

`s1 == s2`

`s1 < s2`

`s1 >= s2`

`...`

```
bool equals(set<int> s1, set<int> s2) {  
    int n = s1.size();  
    if (n != s2.size()) return false;  
    auto i1 = s1.begin();  
    auto i2 = s2.begin();  
    while (n--) {  
        if (*(i1++) != *(i2++)) return false;  
    }  
    return true;  
}
```

เปรียบเทียบตามลำดับข้อมูล
ที่เจอจาก iterator

ข้อมูลใน set เก็บเหมือนเรียงลำดับตลอดเวลา

```
set<int> s1 = {3,2,1};  
set<int> s2 = {1,4,2};  
cout << (s1 < s2) << endl; // 1  
s1.insert(4); s2.insert(3);  
cout << (s1 == s2) << endl; // 1  
  
vector<int> v1 = {1,2,3};  
vector<int> v2 = {3,1,2};  
cout << (v1 == v2) << endl; // 0
```


ข้อมูลใน set เก็บแล้วจะเป็น read-only

```
set<int> s = {4,1,2,6,5};  
*s.begin() = 99; // WRONG (read only)
```

```
set<int> s = {4,1,2,6,5};  
for (auto & e : s)  
    e += 7; // WRONG
```

```
set<vector<int>> s = {{1,2}, {9}};  
(*s.begin())[0] = 99; // WRONG
```

```
set<vector<int>> s = {{1,2}, {9}};  
(*s.begin()).push_back(99); // WRONG
```

ทำไม ?
จะได้เรียนในวิชา
โครงสร้างข้อมูล

การค้นข้อมูล : s.find(e)

คืน iterator ที่ชี้ข้อมูล e ที่พบ ใน s
แต่ถ้าไม่มี e ใน s จะคืน s.end()

```
set<int> s = {40,10,50,60,20};  
cout << (s.find(50) != s.end()) << endl;    // 1  
cout << (s.find(99) != s.end()) << endl;    // 0
```

อย่างค้นหาด้วย `find(s.begin(), s.end(), e)`

```
set<int> s;  
int N = 1000; // try N = 100000  
for (int i=0; i<N; ++i)  
    s.insert(i);  
int c = 0;  
auto begin = s.begin(), end = s.end();  
for (int i=0; i<N; ++i) {  
    c += (*find(begin, end, i) == i); // SLOW  
    c += (*s.find(i) == i); // FAST  
}  
cout << c << endl; // 1000
```

ทำไม ?
จะเรียนในวิชา
โครงสร้างข้อมูล

ตัวอย่าง : นับจำนวนสระ

```
set<char> vowels = {'A','E','I','O','U',  
                   'a','e','i','o','u'};  
  
string line;  
getline(cin, line);  
auto vend = vowels.end();  
int count = 0;  
for (char c : line) {  
    if (vowels.find(c) != vend)  
        ++count;  
}  
cout << count << endl;
```

ลบแบบที่ 1 (ลบตรงไหน) : s.erase(itr)

- ลบข้อมูลใน s ที่ itr ชี้
- คืน iterator ที่ชี้ข้อมูล "ถัดไป"
("ถัดไป" ของ e คือข้อมูลน้อยสุดที่มากกว่า e)

```
set<int> s = {40,10,50,60,20};  
auto itr = s.begin(); // ชี้ที่ 10  
++itr;                // ชี้ที่ 20  
itr = s.erase(itr);   // ลบ 20, itr ชี้ที่ 40  
cout << *itr << endl; // 40
```

ลบแบบที่ 2 (ลบอะไร) : s.erase(e)

- ลบข้อมูล e ใน s ออก
- ถ้ามี e ให้ลบ คืน 1 ถ้าไม่มี คืน 0

```
set<int> s = {40,10,50,60,20};  
int r;  
r = s.erase(50);           // r = 1  
r = s.erase(99);           // r = 0  
for (auto e : s)  
    cout << e << ' ';    // 10 20 40 60
```

ตัวอย่าง: erase_range

```
void erase_range(set<int> &s, int low, int high) {
    auto itr = s.begin();
    while (itr != s.end() && *itr <= high) {
        if (low <= *itr && *itr <= high)
            itr = s.erase(itr);
        else
            ++itr;
    }
}

int main() {
    set<int> s = {4,1,2,5,9,3,7,8,6,0};
    erase_range(s, 2, 6);
    for (auto e : s)
        cout << e << ' ';
}
```

0 1 7 8 9

ตัวอย่าง: erase_range (แบบนิค)

```
void erase_range(set<int> &s, int low, int high) {  
    for (auto e : s) {  
        if (low <= e && e <= high)  
            s.erase(e);  
    }  
}  
  
int main() {  
    set<int> s = {4,1,2,5,9,3,7,8,0};  
    erase_range(s, 2, 6);  
    for (auto e : s)  
        cout << e << ' ';  
}
```

ดูเหมือนทำได้ แต่การเพิ่ม/ลบข้อมูล
ระหว่างการวนหัยข้อมูล อาจผิด

0 1 3 7 8 9

ผิด

บริการอื่น ๆ ของ set

- `s.empty()` ถ้าไม่มีข้อมูลคืนจริง ไม่ขึ้นคืนเท็จ
- `s.clear()` ลบข้อมูลทั้งหมด เหลือ 0 ตัว
- และอีกหลายบริการ (ที่ขอไม่ครอบคลุมในวิชานี้)

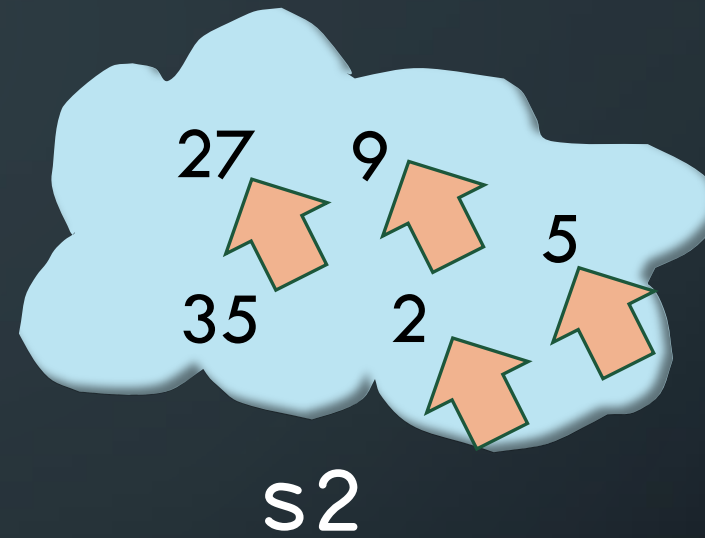
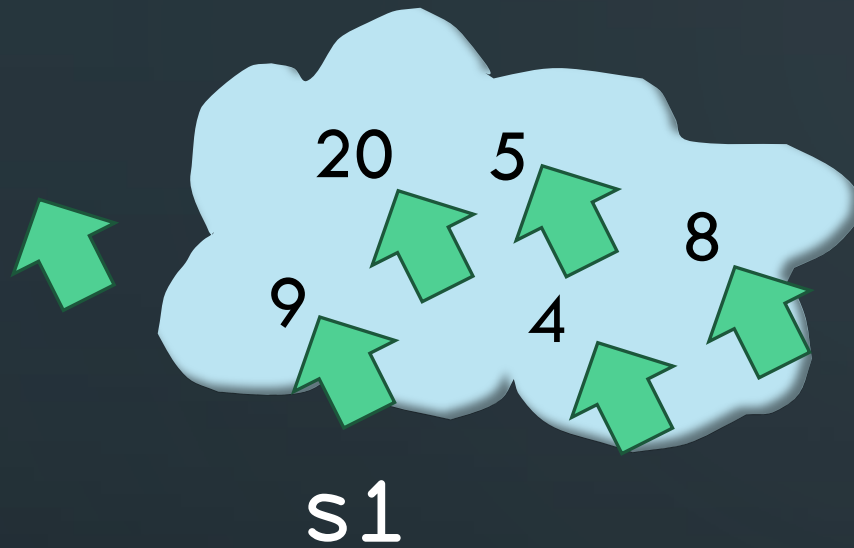
<https://cplusplus.com/reference/set/set/>

ตัวอย่าง: set intersection (แบบที่ 1)

```
set<int> intersection(set<int> &s1, set<int> &s2) {  
    set<int> s;  
    for (auto e : s1) {  
        if (s2.find(e) != s2.end())  
            s.insert(e);  
    }  
    return s;  
}
```

ลุยตรวจสอบสมาชิกของ s1 ที่ค้นพบใน s2

ตัวอย่าง: set intersection (แบบที่ 2)



การลบหีบข้อมูลจาก set
จะได้ข้อมูลจากน้อยไปมาก

ตัวอย่าง: set intersection (แบบที่ 2)

```
set<int> intersection(set<int> &s1, set<int> &s2) {  
    set<int> s;  
    auto i1 = s1.begin(), e1 = s1.end();  
    auto i2 = s2.begin(), e2 = s2.end();  
    while (i1 != e1 && i2 != e2) {  
        if (*i1 < *i2) {  
            ++i1;  
        } else if (*i2 < *i1) {  
            ++i2;  
        } else { // *i1 == *i2 is true  
            s.insert(*i1);  
            ++i1; ++i2;  
        }  
    }  
    return s;  
}
```

การลบหีบข้อมูลจาก set
จะได้ข้อมูลจากน้อยไปมาก

ตัวอย่าง: set intersection (แบบที่ 3)

```
void intersection(set<int> &s1, set<int> &s2) {  
    auto i1 = s1.begin(), e1 = s1.end();  
    auto i2 = s2.begin(), e2 = s2.end();  
    while (i1 != e1 && i2 != e2) {  
        if (*i1 < *i2) {  
            i1 = s1.erase(i1);  
        } else if (*i2 < *i1) {  
            ++i2;  
        } else {  
            ++i1; ++i2;  
        }  
    }  
    while (i1 != e1) i1 = s1.erase(i1);  
}
```

ผลลัพธ์อยู่ใน s1

ตัวอย่าง: Sieve of Eratosthenes

จำนวนเฉพาะทุกตัวที่มีค่าไม่เกิน N

```
s = { 2, 3, 4, 5, 6, 7, 8, 9,  
      10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  
      20, 21, 22, 23, 24, 25, 26, 27, 28, 29,  
      30, 31, 32, 33, 34, 35, 36, 37, 38, 39,  
      40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50 }
```

```
{ 4, 6, 8, 10, ..., 50}  
{ 6, 9, 12, 15, ..., 48}  
{ 8, 12, 16, 20, ..., 48}  
{ 10, 15, 20, 25, ..., 50}  
{ 12, 18, 24, 30, ..., 48}  
{ 14, 21, 28, 35, ..., 49}
```

ตัวอย่าง: Sieve of Eratosthenes

```
set<int> eratosthenes(int N) {  
    set<int> s;  
    for (int i=2; i<=N; ++i) {  
        s.insert(i);  
    }  
    for (int i=2, x=sqrt(N); i<=x; ++i) {  
        for (int k=2*i; k<=N; k+=i)  
            s.erase(k);  
    }  
    return s;  
}
```

```
{ 4, 6, 8, 10, ...}  
{ 6, 9, 12, 15, ...}  
{ 8, 12, 16, 20, ...}  
...
```

ลบด้วย iterator
น่าจะเร็วกว่า ลองคิดดู

set : สรุ

- เหมาะกับการเก็บข้อมูลที่ไม่ต้องการข้อมูลซ้ำ
- iterator วิ่งตามข้อมูลจากค่าน้อยไปมาก (หรือมากมาน้อย)
- ค้นหาได้เร็ว เพิ่มได้เร็ว ลบได้เร็ว แต่ห้ามเปลี่ยนค่าของข้อมูล
 - `s.find(e)`, `s.insert(e)`, `s.erase(e)`, `s.erase(itr)`
 - ใช้เวลาแปรตาม $\log_2(\text{ปริมาณข้อมูล})$
(จะได้เรียนในวิชาโครงสร้างข้อมูล)