C++
STANDARD TEMPLATE LIBRARY (STL)
(SORTING)

WASARA RODHETBHAI

DEPARTMENT OF COMPUTING • SILPAKORN UNIVERSITY

03/2023

การเรียงลำดับ (Sorting)

- วิธีการที่ใช้ในการจัดเรียงตัวเลขหรือ elements ภายใน array หรือ vector จากน้อยไปหามาก (ascending) หรือจากมากไปหาน้อย (descending)
- วิธีการเรียงลำดับสามารถทำได้หลากหลายวิธี หนึ่งในวิธีการเรียงลำดับคือ การใช้ฟังก์ชันการเรียงลำดับในไลบารี STL

ไลบารีฟังก์ชัน sort ( ) ใน STL

- ในการใช้ sort function จะต้องมีการ include ไลบารี algorithm

```
#include <algorithm>
```

รูปแบบ function ของ std::sort()

```
std::sort(ตำแหน่งเริ่มต้น, ตำแหน่งสุดท้าย, optional comparator ฟังก์ชันกำหนดการจัดเรียงลำดับ)
```

- ฟังก์ชัน sort เป็นฟังก์ชันที่ไม่มีการ return ค่ากลับ (void)
- หากไม่ใส่พารามิเตอร์ที่ 3 ฟังก์ชันจะถือว่าเป็นฟังก์ชัน std::less() ฟังก์ชันนี้จะคืนค่า Boolean โดยพิจารณาจากการเปรียบเทียบสองอาร์กิวเมนต์ โดย return true หากอาร์กิวเมนต์แรกน้อยกว่าอีกอาร์กิวเมนต์

เทคนิคการเรียงลำดับแบบผสม 3 แบบ (Introsort)

- ใน sort function ของ C++ ใช้การเรียงลำดับแบบ IntroSort ซึ่งถือเป็นการเรียงลำดับแบบผสม (hybrid sort)
- IntroSort เป็นการผสมผสานของ Quick Sort, Heap Sort และ Insertion Sort โดย
  - ในกรณีทั่วไป ฟังกชันเรียงลำดับจะใช้การเรียงลำดับด้วยวิธี Quicksort
  - แต่ในบางครั้งกรณี unfair conditions, โดยอาจมีการเรียงที่ใช้เวลามากกว่า N * log(N) ฟังก์ชันจะเปลี่ยนการเรียงลำดับเป็นวิธี Heap Sort
  - เมื่อข้อมูลมีจำนวนน้อย การเรียงลำดับจะใช้วิธี Insertion Sort

ความซับซ้อนของอัลกอริทึม sort()

- Average complexity ของฟังก์ชัน sort = N * log(N)

  เมื่อ N คือจำนวนข้อมูลใน array หรือ vector

EX 1.1 เรียงลำดับน้อยไปมาก (Ascending Order)-Array

```
int darr[5] = {30, 50, 40, 10, 20};
int len = sizeof(darr)/sizeof(darr[0]);
cout<<"Before sorting array : ";
for(int i=0; i<len; i++){
      cout<<" "<<darr[i];
}
sort(darr, darr + len);  //Sorting darr array
cout<<"\n\nAfter sorting array : ";
for(int i=0; i<len; i++){
      cout<<" "<<darr[i];
}
```

EX 1.2 เรียงลำดับน้อยไปมาก (Ascending Order)-vector

```
vector<int> dvec = {30, 50, 40, 10, 20};
cout<<"Before sorting vector : ";
for(auto i=dvec.begin(); i<dvec.end(); i++){
      cout<<" "<<*i;
}
sort(dvec.begin(),dvec.end());  //Sorting dvec
cout<<"\n\nAfter sorting vector : ";
for(auto i=dvec.begin(); i<dvec.end(); i++){
      cout<<" "<<*i;
}
```


การเรียงลำดับจากมากไปน้อย (Descending Order)

- std::sort(ตำแหน่งเริ่มต้น, ตำแหน่งสุดท้าย, greater<data_type>())
- ฟังก์ชัน std::greater() ซึ่งจะทำงานตรงกันข้ามกับฟังก์ชัน std::less() โดยจะเปรียบเทียบอาร์กิวเมนต์ 2 ตัวและ return True เมื่ออาร์กิวเมนต์แรกมีค่ามากกว่าอีกอาร์กิวเมนต์ มิฉะนั้นจะ return False

EX 2.1 เรียงลำดับมากไปน้อย (descending Order)-array

```
int darr[5] = {30, 50, 40, 10, 20};
int len = sizeof(darr)/sizeof(darr[0]);
cout<<"Before sorting array : ";
for(int i=0; i<len; i++){
      cout<<" "<<darr[i];
}
sort(darr, darr + len,greater<int>());  //Sorting darr
cout<<"\n\nAfter sorting array : ";
for(int i=0; i<len; i++){
      cout<<" "<<darr[i];
}
```


EX 2.2 เรียงลำดับมากไปน้อย (descending Order)-vector

```
vector<int> dvec = {30, 50, 40, 10, 20};
cout<<"Before sorting vector : ";
for(auto i=dvec.begin(); i<dvec.end(); i++){
      cout<<" "<<*i;
}
sort(dvec.begin(),dvec.end(),greater<int>());  //Sorting dvec
cout<<"\n\nAfter sorting vector : ";
for(auto i=dvec.begin(); i<dvec.end(); i++){
      cout<<" "<<*i;
}
```

การเรียงลำดับตามเงื่อนไขที่กำหนด (User-Defined Order)

- เป็นการกำหนดฟังก์ชันเงื่อนไขการเรียงลำดับลงไปในอาร์กิวเมนต์ที่ 3 ของ sort
- โดยฟังก์ชันดังกล่าวต้อง return ค่าเป็น Boolean (True/False)
- ยกตัวอย่างเช่น การจัดเรียงข้อมูลในอาร์เรย์โดยพิจารณาจากเศษที่เหลือเมื่อหารด้วย 10 (ใช้ตัวดำเนินการ '%') จากน้อยไปมาก

ตัวอย่างที่ 3

```
bool My_func( int a, int b){
      return (a%10)<(b%10);
}
int main() {
      int darr[5] = {28, 46, 64, 92, 71};
      int len = sizeof(darr)/sizeof(darr[0]);
      sort(darr, darr + len, My_func); //Sorting darr array
}
```

Lambda expressions

- ตั้งแต่ C++11 เป็นต้นมา มีการใช้แลมบ์ดานิพจน์ในการเขียนโปรแกรม C++ ซึ่งเป็นการเขียนฟังก์ชันไว้ในบรรทัดเดียว โดยไม่ต้องการการประกาศหรือแม้แต่การระบุประเภทการส่งคืน
- สามารถใช้นิพจน์แลมบ์ดาที่เรากำหนดขึ้นเองเพื่อกำหนดลำดับของการจัดเรียง ซึ่งสามารถทำได้โดยกำหนดนิพจน์หนึ่งบรรทัดเป็นพารามิเตอร์ที่สามของฟังก์ชัน sort()

EX: Lambda expressions

```
vector<int> Vec{5,4,7,6,2,8,9,1,3};
sort (Vec.begin(), Vec.end(), [](int a, int b) { return a<b; });
for (auto elm: Vec){
    cout << elm <<" ";
}
```

Patial Sort

- partial_sort() function ใช้สำหรับการเรียงลำดับเพียงบางส่วนของข้อมูลทั้งหมด
- นำไปประยุกต์ใช้ เช่น การค้นหาค่าที่มากที่สุดหรือน้อยที่สุด โดยไม่ต้องเรียงลำดับข้อมูลทั้งหมด เป็นต้น

Ex: partial_sort

```
vector<int> dvec = {30, 50, 40, 10, 200};
cout<<"Before sorting vector : ";
for (auto i:dvec) {
      cout<< i <<" ";
}
partial_sort(dvec.begin(),dvec.begin()+2,dvec.end());
for (auto i:dvec) {
      cout<< i <<" ";
}
```

ความแตกต่างของ partial_sort กับ sort

```
vector<int> v = { 59, 45, 70, 68, 13, 51, 30 }, v1;
int i;
v1 = v;
// Using std::partial_sort
partial_sort(v.begin(), v.begin() + 2, v.end());
// Using std::sort()
sort(v1.begin(), v1.begin() + 2);
```

```
v = 13 30
v1 = 45 59
```


EX: Sort a String

```
string str;
cout << "Enter a string: ";
cin >> str;
sort(str.begin(), str.end());
cout << "The sorted string is: " << str;
```

```
Enter a string: sun
The sorted string is: nsu
```

EX: Sort ชื่อตามลำดับตัวอักษร

```
vector<string> nvec = { "Ton", "Harry", "Sam", "Ann", "Aek" };
cout<<"Before sorting vector : ";
for (auto em:nvec)
    cout << em << " ";

sort(nvec.begin(), nvec.end());

cout<<"\n\nAfter sorting vector : ";
for (auto em:nvec)
    cout <<em << " ";
```

```
Before sorting vector : Ton Harry Sam Ann Aek

After sorting vector : Aek Ann Harry Sam Ton
```


Sort Array of Struct

```
struct Student
{
      string name;
      int score;
};
bool compareScore(Student a, Student b)
{
            return a.score > b.score;
}
```

```
      int n = 5;
      Student a[n];
      // Details of Student 1
      a[0].name = "Steve";
      a[0].score = 80;
      // Details of Student 2
      a[1].name = "Sunny";
      a[1].score = 95;
      /*Bla Bla Bla*/
      sort(a, a + 5, compareScore);
```

ฟังก์ชัน is_sorted

- ไว้ใช้ในการตรวจสอบว่าค่าข้อมูลเรียงแล้วหรือยัง
- Return ค่าเป็น True (กรณีเรียงแล้ว)/False (กรณียังไม่ได้เรียง)
- มีรูปแบบดังนี้

std::is_sorted(ตำแหน่งเริ่มต้น, ตำแหน่งสุดท้าย, optional comparator ฟังก์ชันกำหนดการจัดเรียงลำดับ)

EX: is_sorted

```
int a[] = { 35, 66, 75, 12, 43 };
int range = 3;
if (is_sorted(a, a + range))
      cout << "Sorted in the range of " << range;
else
      cout << "Not Sorted in the range of " << range;
```

EX: is_sorted

```
bool ignore_case(char a, char b)
{
   return (tolower(a) <= tolower(b));
}
bool check_if_sorted(string str)
{
  return is_sorted(str.begin(), str.end(), ignore_case);
}
```


EX: is_sorted (II)

```
int main()
{
    string str = "aNT";
    if (check_if_sorted(str))
      cout << str <<" is sorted order.";
    else
      cout << str <<" is not sorted order.";
}
```