Ⅰ. C++ STL

STL은 Standard Template Library로 C++을 위한 라이브러리입니다. 여러 자료구조를 포함하고 있고 그에 맞는 적절한 알고리즘이 모여 있기 때문에 용도에 맞게 빠르게 이용하면 됩니다. Template은 쉽게 말해서 자료형이 자유롭다고 보시면 됩니다. (JAVA, C#의 제네릭과 유사)

C++ STL은 기본적으로 3가지로 구분할 수 있습니다. 자료구조인 컨테이너(container), 컨테이너 내부의 원소를 다루는 반복자(iterator), 컨테이너에 대한 작업을 정의한 알고리즘(algorithm).

**1. 컨테이너(container)**

사용법은 # include <vector>와 같이 원하는 자료구조의 헤더파일을 추가해주면 됩니다. 전부 namespace std에 존재합니다.

**vector**: 동적배열 기반의 연결리스트입니다. (파이썬의 기본 리스트와 유사함, 기능적으로는 동일)

**stack**: 변수의 스코프, 함수 등을 통해 이미 충분히 익숙한 자료구조입니다. LIFO(Last-In First-Out) 구조로 들어오고 나가는 통로가 오직 1개입니다. 가장 마지막에 들어온 데이터가 나갈 수 있는 대상입니다. 웹페이지의 뒤로가기, 문서편집기의 되돌리기 기능 등으로 이용됩니다.

**queue**: FIFO(First-In-First-Out)구조입니다. 가장 먼저 들어온 것을 가장 먼저 내보내는 구조로, 번호표 시스템, 프린터 출력 시스템 등에 사용됩니다.

**deque**: 데큐는 stack과 queue의 특징을 모두 가지는 자료구조입니다. 내부적으로 동적배열로 구성되어 있으며 stack, queue는 따로 지정하지 않으면 제한된 deque의 형태로 구성되게 됩니다.

**list**: 메모리 참조 기반의 연결리스트입니다. 기본적으로 이중연결리스트로 구성되어 있으며 단일연결리스트를 쓰려면 **forward\_list**를 써야 합니다.

**set**: 집합은 유일한 데이터를 다루는데 유리한 자료구조입니다. 예를 들어 A란 집합에 1, 2, 3과 같은 형태는 저장할 수 있지만 1, 1, 1과 같은 형태는 저장되지 않습니다. (내부적으로 1만 저장됩니다.) set은 기본적으로 정렬된 형태의 집합입니다(내부적으로 레드블랙트리). 순서가 정렬되지 않은 형태인 **unordered\_set**이 따로 존재합니다(내부적으로 해시테이블).

**map**: 유일한 순서쌍 데이터에 대한 자료구조입니다. key를 통해 value를 찾는 자료구조로 탐색에 유리합니다 **unordered\_map**은 파이썬의 딕셔너리와 유사하고 기능적으로는 동일합니다(그냥 map은 레드블랙트리, unordered\_map은 해시테이블).

**priority\_queue**: 우선순위를 가지는 큐입니다. 내부적으로 vector 기반으로 만들어집니다. 우선순위를 사용자가 직접 구현할 수도 있고 데이터 삽입 시 저절로 우선순위를 유지합니다.

**\*pair**는 생략하겠습니다. 궁금하시면 찾아보세요.

\*중복키를 허용하는 **multiset**, **multimap**과 **unordered\_multiset**, **unordered\_multimap**도 존재합니다. 내부적으로는 중복키에 대해 새로운 단일연결리스트를 생성하는 방식을 사용합니다. 흔히 사용하는 자료구조는 아닙니다.

\*레드블랙트리는 삽입, 삭제, 탐색이 O(logN)이고 해시테이블은 삽입, 삭제, 탐색이 O(N)이지만 기대시간은 O(1)입니다. 또한 공간에 있어서 해시테이블은 낭비가 심합니다(기대시간을 유지하기 위해 거의 항상 실제 데이터의 2배 이상의 크기 유지). 레드블랙트리는 항상 정렬된 형태이지만 해시테이블은 정렬되지 않은 형태이며 정렬 자체가 불가능하기 때문에 unordered\_ 를 사용하게 됩니다.

**2. 반복자(iterator)**

반복자는 컨테이너에 접근하기 위해 만들어진 것이라고 말씀드렸는데, 언뜻 보면 포인터와 유사해보입니다. 그러나 똑같진 않고 컨테이너 접근이란 개념에 집중하면 좋을 것 같아요.

set<int> setA;

setA.insert(10);

setA.insert(7);

setA.insert(4);

setA.insert(2);

setA.insert(1);

for (set<int>::iterator iter = setA.begin(); iter != setA.end(); iter++)

cout << \*iter << '\n';

\*iter 로 마치 포인터 참조로 값을 알아보는 것을 알 수 있습니다. 참고로 map과 같은 자료구조는 데이터가 2개이기 때문에 iter->first, iter->second로 데이터에 접근할 수 있습니다. begin()이나 end()는 iterator를 다룰 수 있게 도와주는 함수들이고 모든 컨테이너에 존재합니다.

반복자는 이름은 반복자지만 실제로는 컨테이너의 원소 중 한 부분을 가리키는 거라고 했었죠. 그래서 특정 원소를 가리키고 작업을 하는 것이 편리할 때도 쓸 수 있습니다. 예를 들어 연결리스트와 관련된 문제 등에서 특정 노드를 가리켜야 할 때 쓰일 수 있겠죠.

위에는 반복자가 뭔지 소개했을 뿐이고 단순 반복은 아래가 훨씬 편합니다. auto는 자료형이 저절로 설정되는 자료형입니다. 타입 추론이라고도 하며 JAVA와 C#의 var과 유사합니다. C#에 존재하는 foreach와 같은 형태로 쓸 수 있어 직관적이고 사용하기 편리합니다.

for (auto i : setA)

cout << i << '\n';

**3. 알고리즘**

사실 실제로 데이터를 다룰 때 일일히 반복문을 사용하며 원소를 탐색하는 일은 드물겠죠. 뭔가 찾는 경우엔 위와 같이 set의 경우에는 setA.find()를 쓰면 되듯이 대부분의 컨테이너는 작업에 알맞은 알고리즘 함수를 가지고 있습니다. 위에서 쓰인 insert(), begin(), end() 모두 알고리즘 함수의 예시입니다.

함수가 너무 다양하기 때문에 현재 모두 외우기엔 무리가 있고, 먼저 문제를 보고 사용해야 하는 컨테이너를 정했다면 원하는 알고리즘을 택하는 형식으로 하는 게 좋을 것 같아요. (자료구조에 대한 이해가 충분하다면 그에 맞는 적절한 함수가 모두 구비되어 있다고 생각하시면 됩니다.)

[문제]

시작은 2022.00.00() 입니다. 마감 기한은 **2022.00.00() 23:59**입니다. 특별한 사정없이 마감 기한까지 제출을 못하면 경고를 부여받으므로 주의해주세요.

이번엔 보고서를 제출하는 문제입니다!

\*문제는 별도의 파일로 드리겠습니다.