항목 1: 적재적소에 알맞은 컨테이너를 사용하자

우선 STL 컨테이너를 분류하는 방법에 대해 소개해 드리겠습니다.

* **연속 메모리**

동적 할당된 하나 이상(보통 하나)의 메모리 단위에다가 데이터 요소를 저장해 두는 컨테이너 입니다. 각 메모리 단위(객체)는 하나 이상의 요소를 담고 있습니다. **새 요소가 삽입되거나 어떤 요소가 삭제 되면, 다른 요소들은 앞 혹은 뒤로 밀려나면서 새 요소가 삽입될 공간을 만들거나 지워진 공간을 메웁니다.** 이러한 “밀어내기” 때문에 수행 성능이 떨어질 수 있습니다. 종류로 vector, string, deque등 입니다.

* **노드 기반 컨테이너**

동적 할당된 하나의 메모리 단위에다가 하나의 요소만 저장합니다. **요소를 삽입 혹은 삭제했을 때 노드의 포인터만 영향을 받고 노드의 내용은 그대로 입니다.** 따라서, 삭제나 삽입이 일어났다고 해도 나머지 요소들이 밀려나거나 하지 않습니다. 종류로는 list, 표준 연관 컨테이너 모두가 노드 기반 컨테이너 입니다.

이제 “어떤 상황에 어떤 컨테이너를 쓰면 가장 좋을까?”에 대해 이야기 하겠습니다.

1. 컨테이너의 **아무 위치에 요소를 삽입할 수 있다면 시퀀스** 컨테이너가 좋습니다.
2. 컨테이너 내의 요소들의 **순서 결정에 직접 관여하고 싶다면 해쉬 컨테이너는 피해야** 합니다.
3. 표준 C++에 포함된 컨테이너를 사용해야 한다면 해쉬, slist, rope는 쓸 수 없습니다.
4. **임의 접근 반복자가 필요하다면 vector, deque, string** 입니다. 양방향 반복자가 필요한 경우에는 slist와 해쉬 컨테이너는 쓸 수 없습니다.
5. 요소 삽입, 삭제 할 때 다른 요소들이 **밀려나는 일이 없어야 한다면 연속 메모리 컨테이너는 쓸 수 없습니다.**
6. 컨테이너 내의 데이터가 **C의 데이터 타입과 메모리 배열 구조적으로 호환되어야 한다면 vector** 밖에 쓸 것이 없습니다.
7. **탐색 속도가 가장 중요하다면 가장 빠른 해쉬 컨테이너** 그 다음 빠른 정렬된 vector, 그 다음 표준 연관 컨테이너 중에 선택하세요.
8. 컨테이너의 **참조 카운팅을 신경 쓰고 싶지 않다면 string, rope 쓰지 말고 vector<char>**를 쓰세요.
9. **삽입과 삭제 동작이 안정적으로 되돌릴 수 있어야 한다면 노드 기반 컨테이너를** 사용하세요. 이런 동작이 여러 개의 요소에 대해 이루어져야 할 경우 list를 선택합니다.
10. **반복자, 포인터, 참조자 무효화(포인터가 가리키는 내용이 없어지는 일)되는 일을 최소화 해야 한다면 노드 기반 컨테이너를** 사용하기 바랍니다. 노드 기반 컨테이너는 노드 삽입과 삭제가 일어나도 기존의 반복자나 포인터 혹은 참조자가 삭제되지 않습니다. 반대로 연속 메모리 컨테이너는 전체적인 메모리 재할당이 빈번히 일어나기 때문에 반복자나 포인터, 참조자가 무효화되기 쉽습니다.
11. **임의 접근 반복자를 지원하는 시퀀스 컨테이너가 필요한데, 요소 삭제가 일어나지 않고 요소 삽입이 컨테이너의 끝에서 발생하면, 포인터와 참조자가 무효화되지 않는 것이 필요하다면 deque**가 가장 정답입니다.(deque는 요소 삽입이 끝에서 일어날 때 반복자만 무효화 됩니다. STL 컨테이너 중 포인터와 참조자를 무효화시키지 않고 반복자만 무효화 되는 것은 deque가 유일합니다.)
12. 11