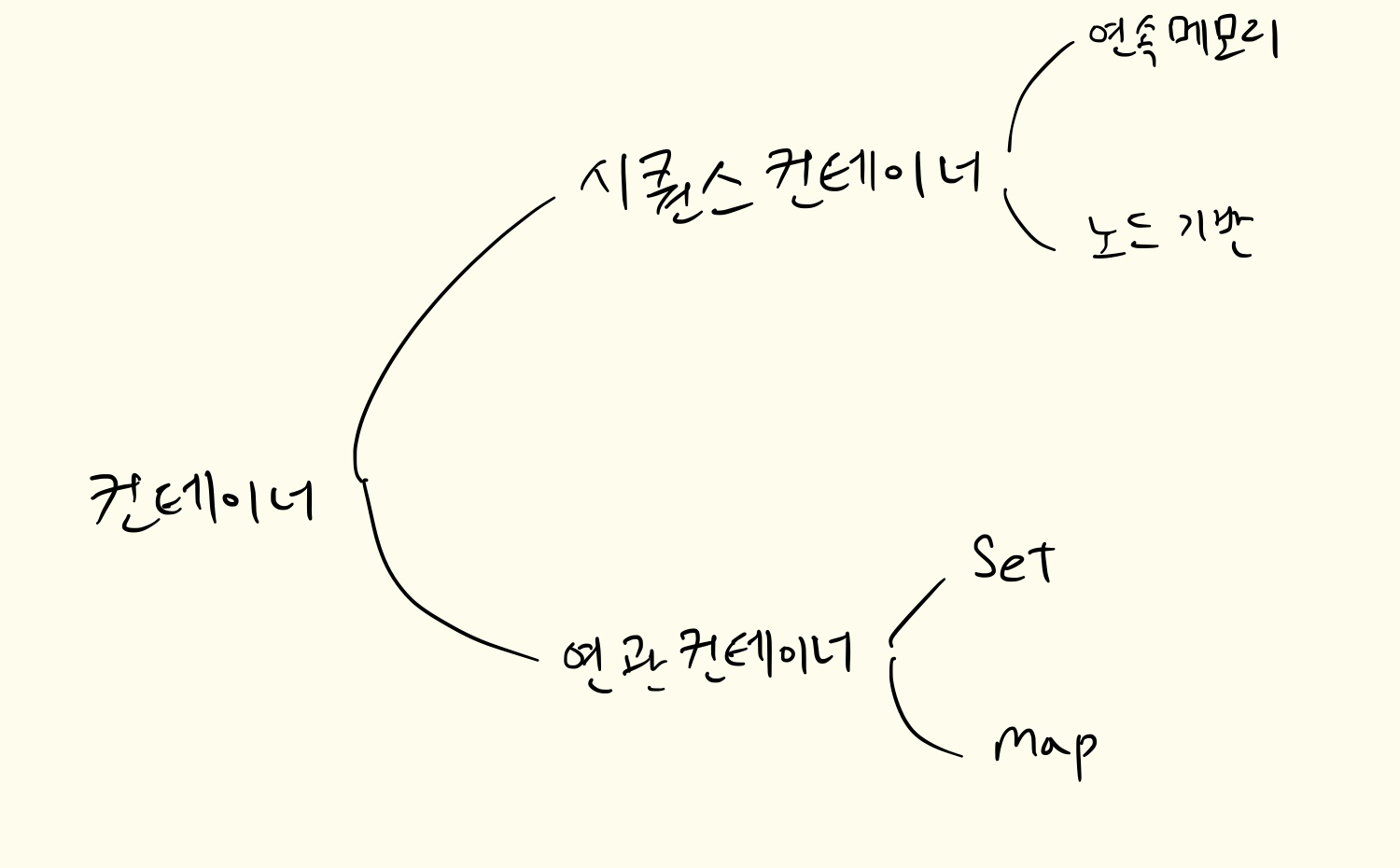
항목 2: “컨테이너에 독립적인 코드”라는 환상을 조심하자

STL은 일반화에 기초를 두고 만든 프로그래밍 장치 입니다. **일반화란 어떤 비슷한 성질을 가진 것들을 모두 같은 취급하여 사용할 수 있는 것**을 의미 합니다. 예를 들어 **배열**이란 데이터 집합은 **컨테이너로 일반화** 되었고, **함수**는 **알고리즘으로 일반화** 되었고, **포인터**는 **반복자로 일반화** 되었습니다.



컨테이너는 시퀀스와 연관 컨테이너 타입으로 일반화 되었고 **각 타입에 속하는 컨테이너 끼리 비슷한 기능**을 가지고 있습니다. 시퀀스 컨테이너 내에서도 연속 메모리 컨테이너와 node 기반 컨테이너로 나뉘는데 **연속 메모리 컨테이너는 임의 접근 반복자를 쓸 수 있지만** node 기만 컨테이너는 쓸 수 없습니다. **node 기반 컨테이너는 push\_front와 push\_back을 지원하지만** 연관 컨테이너는 그렇지 않습니다. 그러나 **연관 컨테이너는 로그 시간 복잡도를 가진 lower\_bound, upper\_bound, equal\_range 멤버 함수를 쓸 수 있습니다.** 시퀀스 컨테이너에서는 지원하지 않습니다.

이처럼 컨테이너에는 각 종류마다 쓰임새가 다릅니다. **적재적소에 사용할 컨테이너를 정하는 것이 중요합니다.** 그러나 다음과 같이 생각 할 수도 있습니다. 모든 상황에 맞는 컨테이너를 만들어 사용하면 어떨까? 이를테면 vector를 쓰는 부분을 만들면서 언제든지 vector 대신 deque나 list를 쓸 수 있는 여지를 남겨 놓는 것입니다. 즉, 모든 종류의 컨테이너를 합쳐놓은 궁극의 컨테이너를 만들어 쓰겠다는 것입니다.

만들 수만 있다면 정말 좋은 컨테이너가 될 수 있겠지만 만들 수 없습니다. 각각의 컨테이너의 멤버 함수들이 자신의 컨테이너에 최적화 되어 있어서 (예를 들어 push\_back과 push\_front는 시퀀스 컨테이너에서만 지원 됩니다.) 멤버 함수를 사용 할 수 없게 됩니다.

또한 반복자, 포인터, 참조자를 무효화시키는 방식이 컨테이너마다 다르다는 것입니다. vector, deque, list를 통합시키는 컨테이너를 만들었다고 가정 하고 insert를 호출하게 되면 모든 것이 무효화 됩니다. deque::insert는 모든 반복자를 무효화 하기 때문입니다. 그 외에도 vector::insert는 모든 참조자와 포인터를 무효화 합니다.  
**STL 컨테이너는 각자 자신만의 장점과 단점을 가지고 있습니다. 서로 바꾸어서 쓸 수 있도록 설계되지 않았고, 감싼다고 해서 되는 것이 없습니다.**

경우에 따라 수시로 컨테이너 타입을 바꿀 수 밖에 없는 입장이라면, 변경을 조금 더 용이하게 해주기 위해 “캡슐화”를 사용합니다. 방법은 아주 간단합니다. **컨테이너와 반복자 타입에 대해 typedef를 거는 것입니다.**

예시 코드

*class Widget{ . . . }; // 클래스 정의*

*typedef vector<Widget> WidgetContainer; // 컨테이너 typedef 걸기*

*typedef WidgetContainer::iterator WCIterator; // 반복자 typedef 걸기*

*WidgetContainer VW; // 컨테이너 생성*

*Widget bestWidget; // 객체 생성*

*WCIterator iterBegin = VW.begin(); // 사용*

만약 typedef를 걸지 않았다면

vector<Widget>::iterator iterBegin = VW.begin( ); 이렇게 코드가 길어질 것입니다. 특수문자 “<”이 들어 있고 없고가 차이 납니다.

사실 typedef는 어떤 타입에 붙인 다른 이름에 불과하기 때문에, 이것으로 할 수 있는 캡슐화의 효과는 지극히 문자적으로 제한 됩니다. 절대 할 수 없는 기능(구현이 되어 있지 않거나 실행 하면 문제가 발생되는 것)을 실행 시키지 못하게 막는 장치는 가지고 있지 않습니다. **할 수 있는 기능만 고객에게 보여주고 싶다면 클래스를 만들고 private영역에 컨테이너를 숨기고 public영역에는 할 수 있는 기능만 함수로 만들어 두면 됩니다.**

이렇게 클래스를 사용하여 캡슐화를 꼼꼼히 잘 해두었다면 list로 만들었다가 문제가 생겨 vector로 바꿀 때 받는 영향이 최소화 됩니다.