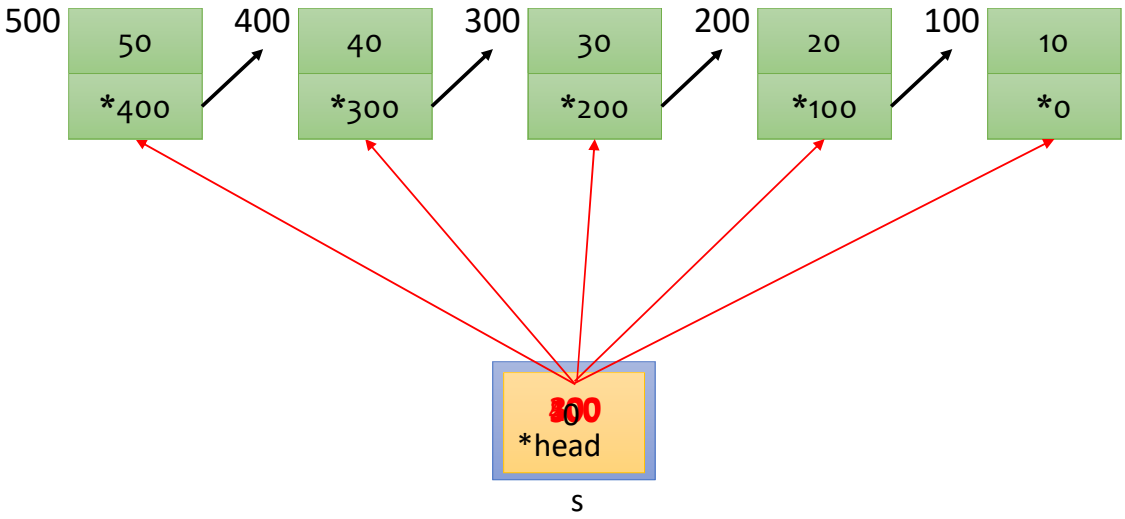
Container

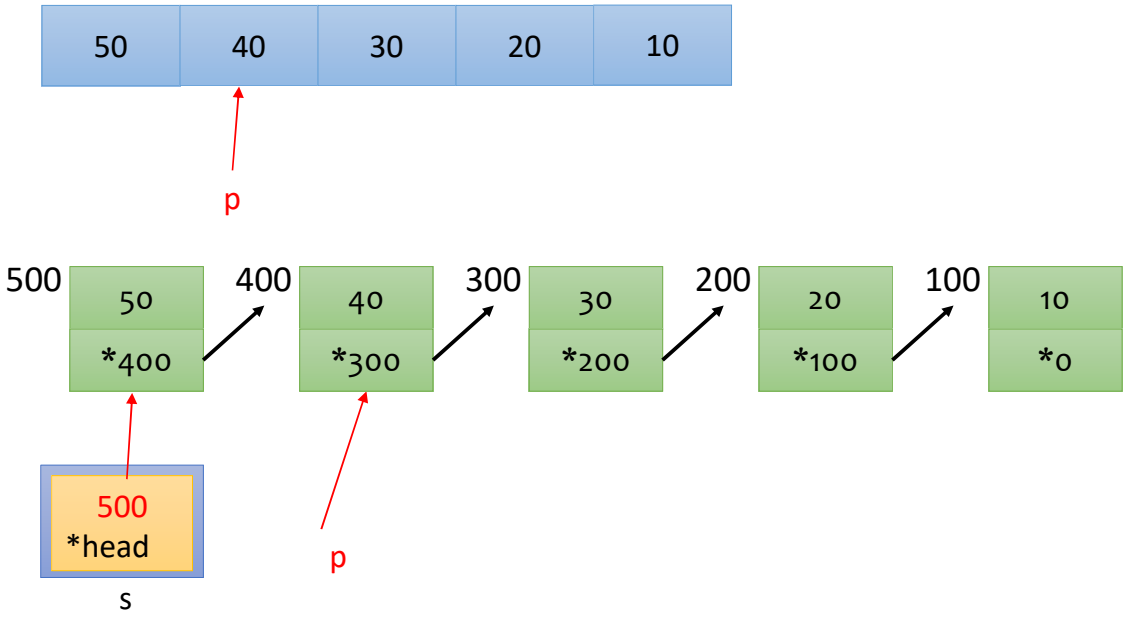
- Node 타입의 모양

- main 함수 실행 시의 메모리 구조



Container

- 앞에서 만든 efind()를 사용해서 slist 에서 값을 검색할 수 있을까 ?



- 배열과 slist 는 요소의 접근 방법과 이동 방식이 다르다.

	요소 간 이동	요소 접근
배열	++p	*p
slist	p = p->next	p->data

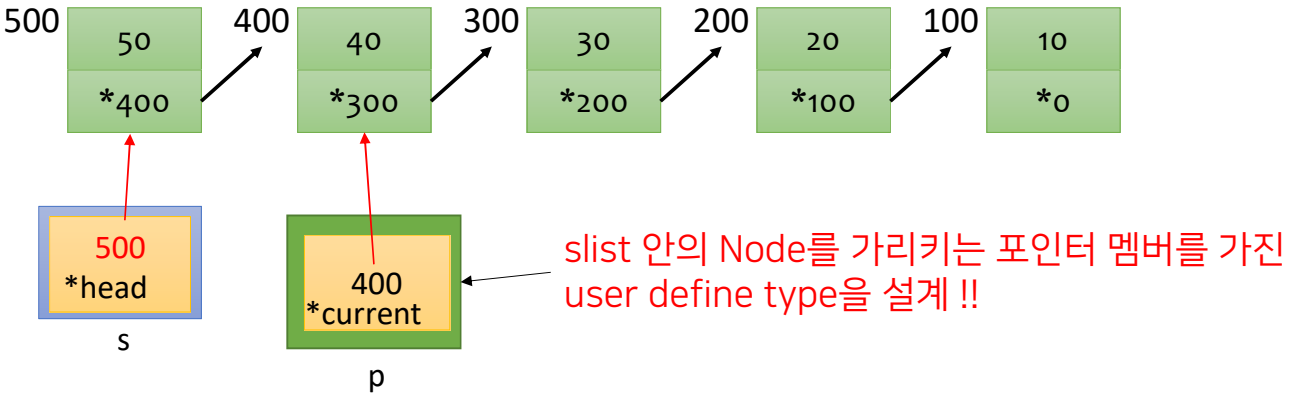
- efind를 사용해서 모든 선형 자료구조에서 값을 검색할 수 없을까 ?

- 반복자(iterator)의 개념

efind 와 slist의 결합

- efind 함수를 사용하기 위한 조건

기능	방법
요소 간 이동	전위형 ++
요소 에 접근	* 연산자
기타	!= 연산이 가능해야 한다.

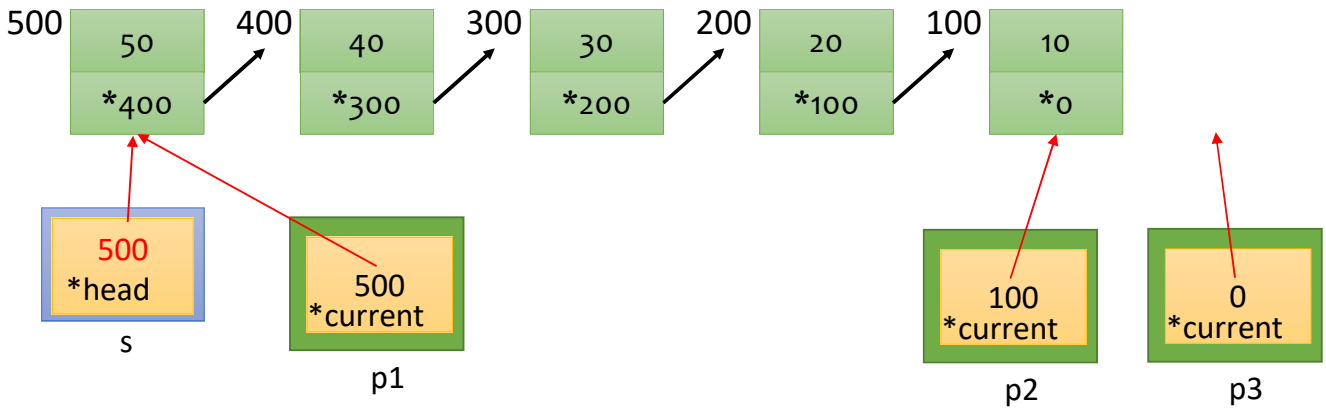


1. p가 raw pointer 라면 ++, * 연산시 동작 방식을 변경할 수 없다.
2. p가 user define type의 객체라면
   - ++p : operator++() 함수가 호출된다.
   - *p : operator*() 함수가 호출된다.

3. 반복자 (iterator)
   - 포인터와 유사하게 동작하는 객체(++로 이동하고, * 로 요소 접근), 연속된 메모리가 아닌 자료구조(컨테이너)의 요소를 열거할 때 배열과 동일한 방식으로 사용 가능 해 진다.

- 반복자(iterator) 만들기

핵심 개념



1. 내부적으로 slist 안의 Node를 가리키는 멤버가 있어야 한다.
   - Node<T>* current 멤버 데이터

2. efind 로 전달하려면 3가지 연산자가 재정의 되어야 한다.
   - ++, *, !=
   - == 도 제공

3. slist 안에 자신의 시작과 마지막 다음을 가리키는 반복자를 리턴 하는 멤버 함수 제공
   - begin : 처음 노드를 가리키는 iterator 리턴
   - end : 마지막 다음 노드(0)를 가리키는 iterator 리턴.

- 반복자(iterator) 만들기

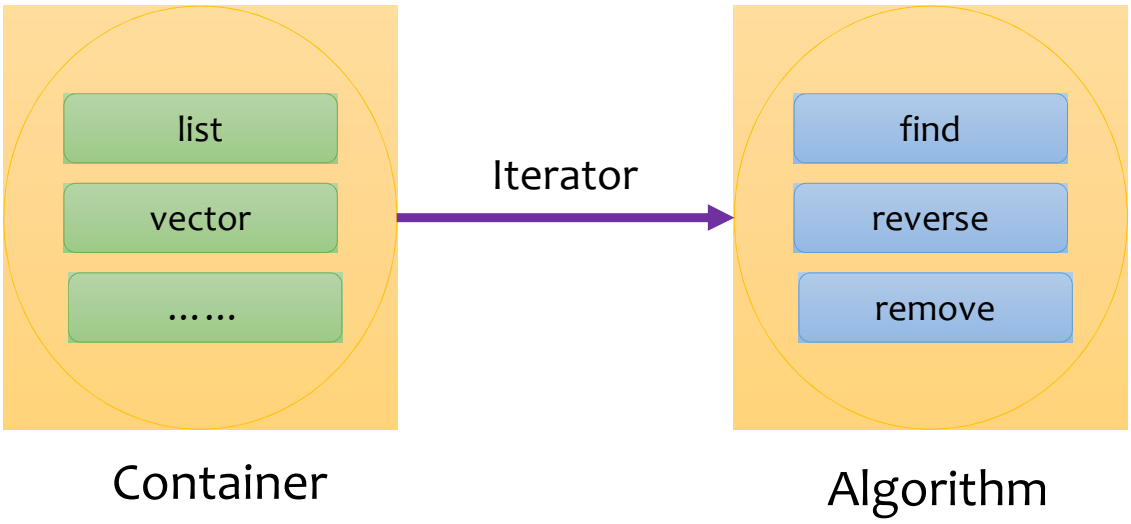
핵심 개념

1. 반복자 타입 이름
   - 각 컨테이너의 반복자 이름을 “iterator” 라는 약속된 이름으로 typedef (또는 using) 으로 제공한다.
   - 반복자의 타입 이름은 “컨테이너이름::iterator” 으로 접근

2. 반복자 덕분에 efind 는 배열 뿐 아니라 모든 선형 자료 구조에서 “선형 검색” 을 수행 할 수 있다.

핵심 개념

- STL 구조



1. 템플릿을 사용하지 않고 find, reverse, remove 를 멤버 함수로 만든다면
   - 3개 컨테이너 * 3개 함수 * 3개 타입 = 27 개 함수가 필요

2. 템플릿을 사용해서 find, reverse, remove 를 멤버 함수로 만든다면
   - 3개 컨테이너 * 3개 함수 = 9 개 함수가 필요

3. find, reverse, remove 를 멤버 함수가 아닌 일반 함수로 만든다면
   - 모든 컨테이너에 동작하는 3개의 함수가 필요
   - 모든 컨테이너의 요소를 동일한 방식으로 접근할 수 있어야 한다. – 반복자가 필요하다.

4. STL은 자료구조와 알고리즘이 분리된 라이브러리
   - 알고리즘 함수는 자신이 어떤 자료구조에 대해 연산을 수행하는지 모른다.