배열과 컨테이너

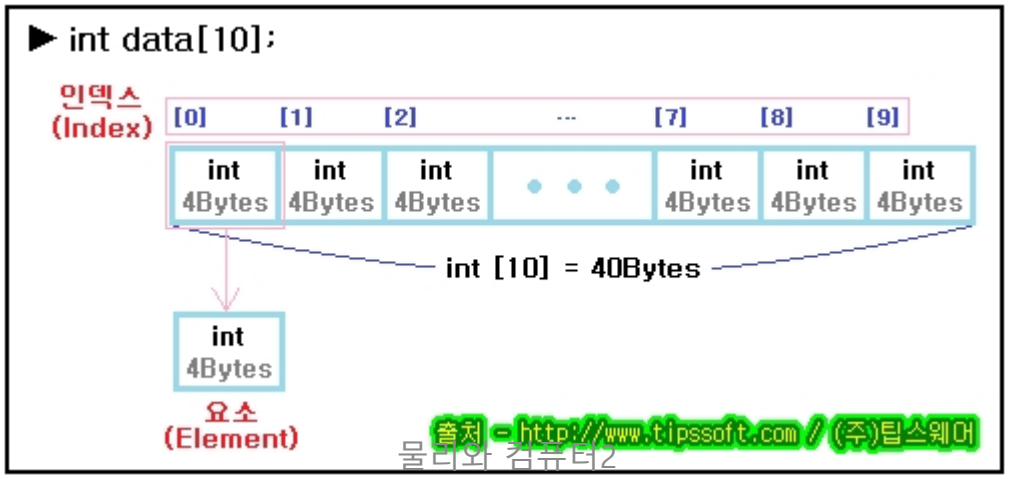
(Data Structure)

들어가기 전에

- 오늘의 소스코드를 받아봅시다.
  - 중요한 코드는 꼭 다른 곳으로 복사해두세요.
  - phycom2015-2 디렉토리에서,
    - git stash
    - git pull
  - 디렉토리가 사라졌다면,
    - git clone https://github.com/geonmo/phycom2015-2.git

C언어의 배열

- 배열이란?
  - 하나의 변수에 상수들의 묶음을 넣는 문법
  - 예) int data[10] = {1,2,3,4,5,6,7,8,9,10};
    std::cout<<a[0]<<std::endl;
  - [index]는 0부터 시작. 즉, data[0]~data[9]



컨테이너

- 배열의 단점
  - 배열이 처음 "선언"될 때 지정한 크기 이상의 데이터를 저장할 수 없다
- 컨테이너의 등장
  - 배열의 단점을 극복하고 보다 효율적인 데이터관리를 위해 자료구조(Data Structure)이론대로 설계된 배열들(실은 Class)

컨테이너의 종류

- 시퀀스 컨테이너
  - 컨테이너의 내용이 어떠한지의 고민 없이 저장하는 컨테이너
  - 데이터를 저장하고 관리하는 방법에 따라 아래와 같이 구분됨
    - vector, deque 그리고 list
- 연관 컨테이너
  - 내용이 어떠한지 고민(정렬)을 하는 컨테이너
    - set(하나만 저장) , map(짝을 지어 저장)

컨테이너의 기초 "vector"

- vector란?
  - 자료를 뒤에만 저장하고 뒤로만 빼내는 배열
  - 사용법은 배열과 거의 같음

6_container_1.cpp

```
int data[10]={1,2,3,4,5,6,7,8,9,10};
vector<int> vec_data;
vec_data.push_back( 1 );    // vec_data ={1};
vec_data.push_back( 2 );    // vec_data ={1,2};
cout<<"vector [1] : "<<vec_data[1]<<endl;
vec_data.pop_back();        // vec_data ={1};
cout<<"vec_data's size :
"<<vec_data.size()<<endl;

// vector의 내부내용 삭제
vec_data.clear();

// 이렇게 정의할 수도 있다.
vec_data.assign( data, data+10 );
cout<<"vector[9] : "<<vec_data[9]<<endl;
```

vector의 업그레이드 "deque"

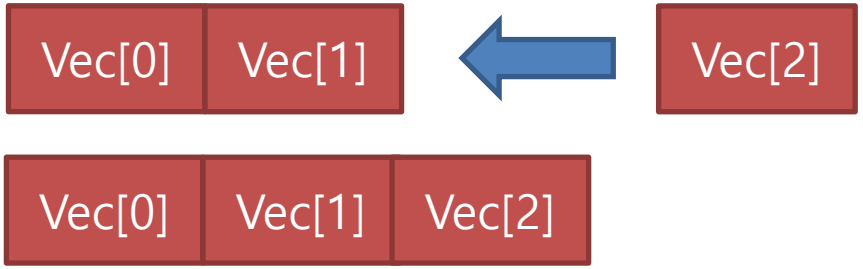
- deque란?
  - 자료를 앞,뒤에만 저장하고 앞,뒤로만 빼내는 배열
  - 사용법은 배열, vector와 거의 같음

6_container_1.cpp

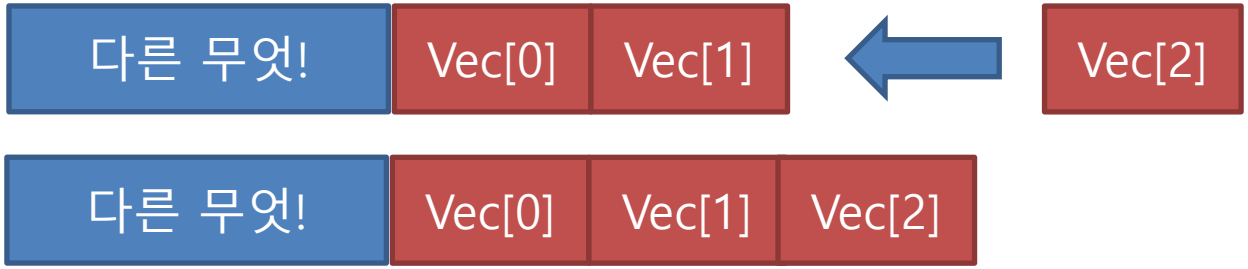
```
std::deque<int> deq_data;
deq_data.push_back(2);        // deq_data={2};
deq_data.push_front(1);       // deq_data={1,2};
deq_data.push_back(3);        // deq_data={1,2,3};
cout<<"deque [2] :" << deq_data[2]<<endl;
deq_data.pop_front();         // deq_data={2,3};
cout<<"deque [1] :" << deq_data[1]<<endl;
```

vector VS deque

- 도형으로 보는 vector와 deque
  - deque.push_back(something)

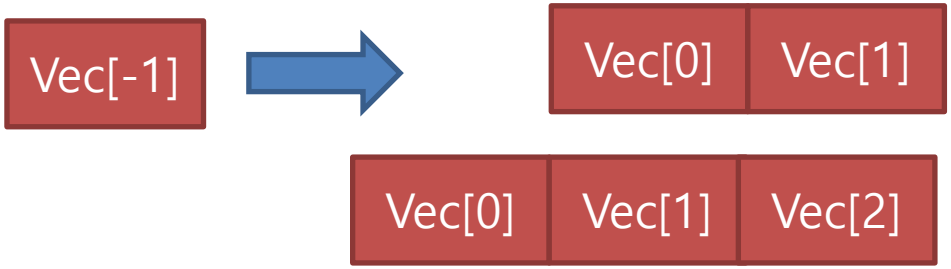


  - vector.push_back(something)

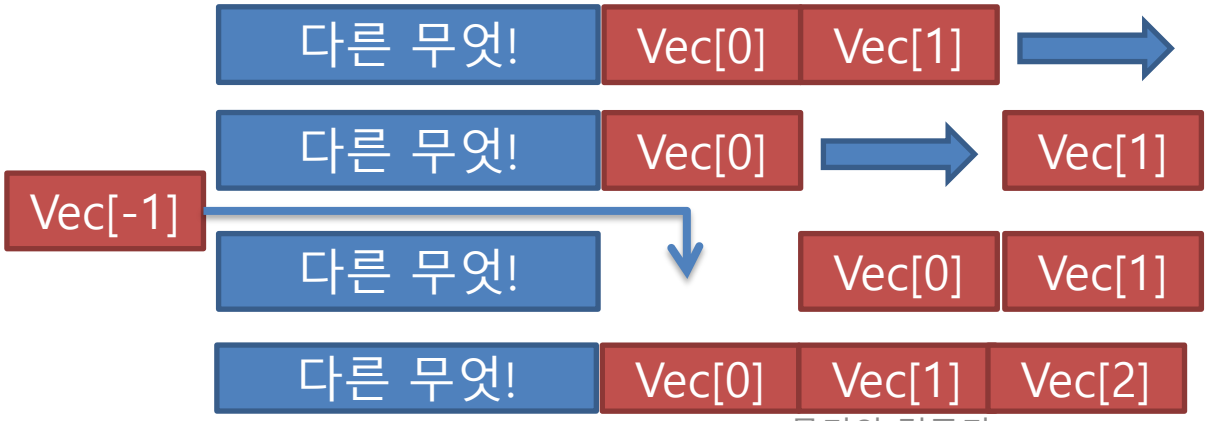


vector VS deque

- 도형으로 보는 vector와 deque
  - deque.push_front(something)

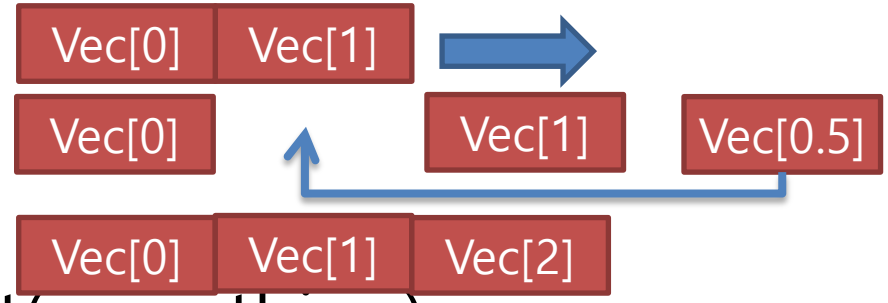


  - vector.push_front(something)



vector VS deque

- 도형으로 보는 vector와 deque
  - deque.insert(something)



  - vector.insert(something)

vector VS deque

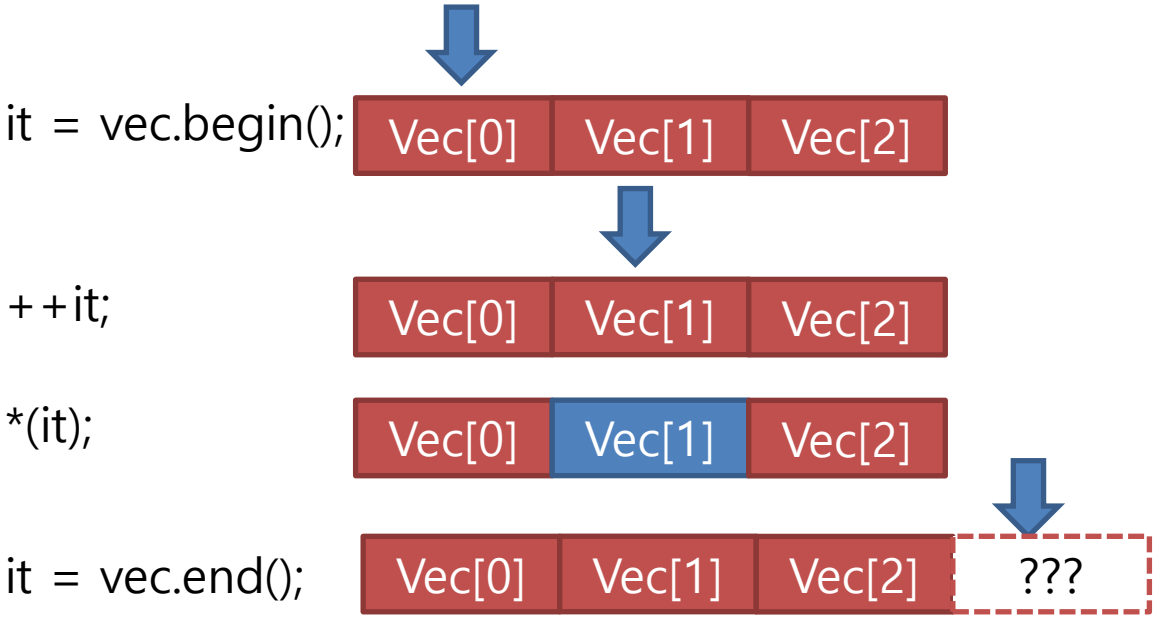
- Spec비교

내용	vector	deque
뒤쪽 삽입, 삭제	O	O
앞쪽 삽입, 삭제	X(느려짐)	O
중간 삽입, 삭제	X(느려짐)	X(느려짐)
무작위 접근(a[5])	O	O
기본 속도	빠름	느림

  - 정리
    - 읽기 전용으로 쓸 때에는 vector를 쓰는 것이 빠름
    - 앞쪽에 데이터를 넣을 때가 있다면 deque가 더 빠름

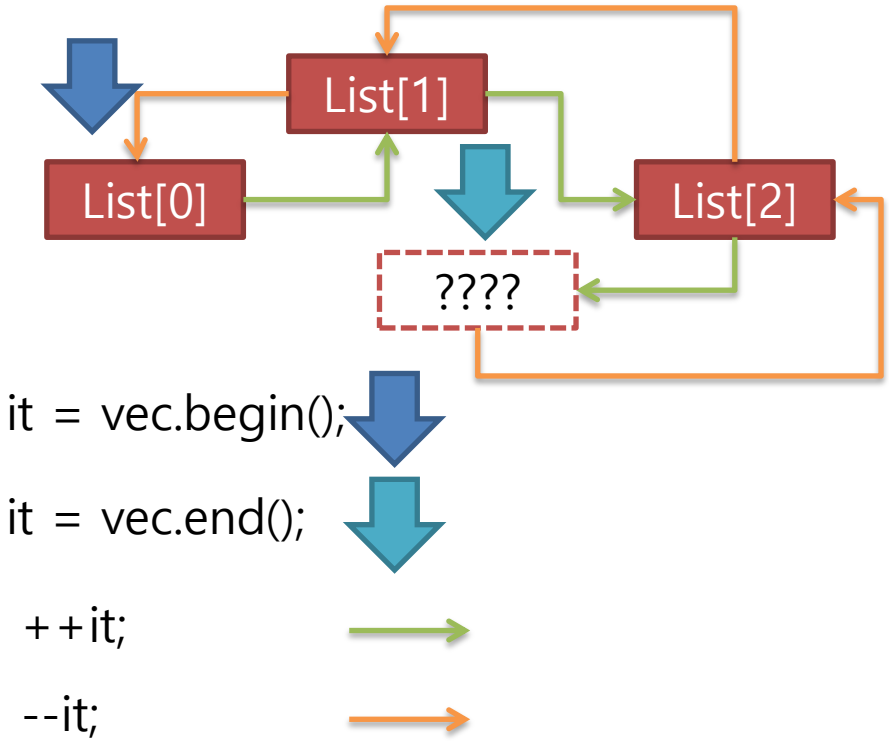
반복자(iterator)

- 반복자란?
  - 컨테이너에 저장된 원소를 가리키는 포인터
    - 다음 주에 배울 포인터랑 매우 비슷
      - 제한된 기능을 가진 포인터라고 보면 됨



삽입과 삭제에 특화된 "list"

- list란?
  - 자료의 저장을 일렬로 저장하는 것이 아니라 이전자료와 다음 자료의 위치를 같이 저장하여 관리
  - 예제 꼭 참고!
    - 12_iterator.cpp

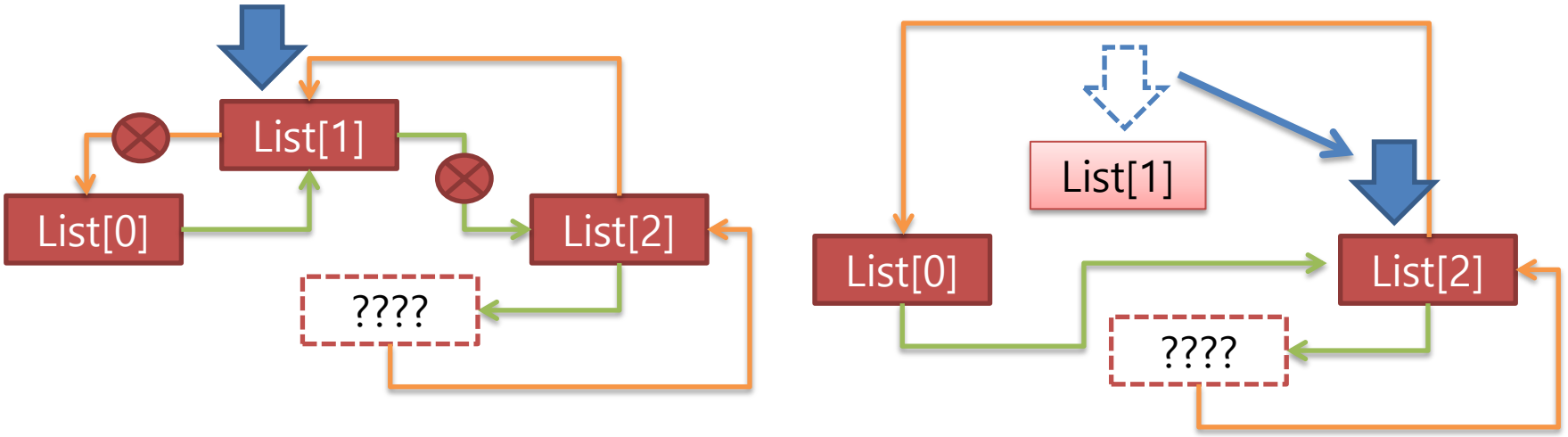


삽입과 삭제에 특화된 “list”

- list의 특징
  - 자료의 추가 및 삭제에 특화
  - 단, 임의 접근 사용 불가(애초에 지원 안함)
    - 예) list_data[1] X
  - 데이터 접근에 반복자를 사용하여 처리
    - 예) *(it), ++it, --it

삽입과 삭제에 특화된 "list"

- list의 삭제(erase)



반복자(iterator)

- 반복자 사용시 주의사항
  - vector, deque에서 반복자 사용시,
    - 값을 추가하거나 빼면 반복자 파괴됨
      - 추가 시 복원 불가. 새로 iterator를 만들어야 함
      - 단, 삭제 시 erase함수의 리턴값이 다음 원소를 가리키는 iterator이므로 사용 가능
  - list에서 반복자 사용시
    - 값을 빼면 해당 반복자 파괴됨
      - 단, erase함수의 리턴값이 다음 원소를 가리키는 iterator
  - 파괴된 반복자를 무시하고 사용하면 세그먼트 폴트 에러 발생

자동정렬 컨테이너 "set"

- set의 특징
  - 값을 임의대로 집어 넣어도 자동으로 정렬(정렬법을 알고 있을 경우)
  - 겹치는 값 자동 삭제
  - 겹치기를 허용하는건 multiset 이라고 따로 존재
  - 사용법은 list랑 비슷

12_iterator.cpp

```
std::set<int> set_data;
set_data.insert(3);
set_data.insert(6);
set_data.insert(4);
set_data.insert(2);
set_data.insert(7);

for( std::set<int>::const_iterator it = set_data.begin() ; it != set_data.end() ; ++it) {
    cout<<*(it)<<endl;
}
```

결과
set:
2
3
4
6
7

“map”!!

- map의 특징
  - 값을 std::pair로 짝을 지어 넣어줘야 함
  - 앞의 값을 first 혹은 key,
    뒤에 값을 second 혹은 value라고 부름
  - map을 익히기 위해
    - 예제를 통해 값을 넣는 법 3가지를 익히자
    - 반복자를 사용할 때는 어떻게 하는지도 익히자
    - find함수도 잘 써보도록 하자

18_map.cpp

```
std::map< std::string, int > students;

// map에 데이터 넣는법 #1
std::pair<std::string, int> p_student1 = std::make_pair< std::string, int>("철수",2210);
students.insert( p_student1);

// map에 데이터 넣는법 #2
students.insert( std::make_pair<std::string, int>("영희", 2211));

// map에 데이터 넣는법 #3
students["길동"] = 2410;

// 마치 임의 접근 한 것 처럼 쓸수 있다.(실은 그렇지 않지만,)
std::cout<<"연산자 [] 사용시"<<std::endl;
std::cout<<"철수 : "<<students["철수"]<<std::endl;
std::cout<<"영희 : "<<students["영희"]<<std::endl;
std::cout<<"길동 : "<<students["길동"]<<std::endl;
```