Project #2
Survey on Programming Paradigms

[Standard Template Library]

제출일 : 2016.05.23
과목명 : 프로그래밍 언어
소　속 : 숭실대학교 컴퓨터학부
학　번 : 20102548
발표자 : 임성민

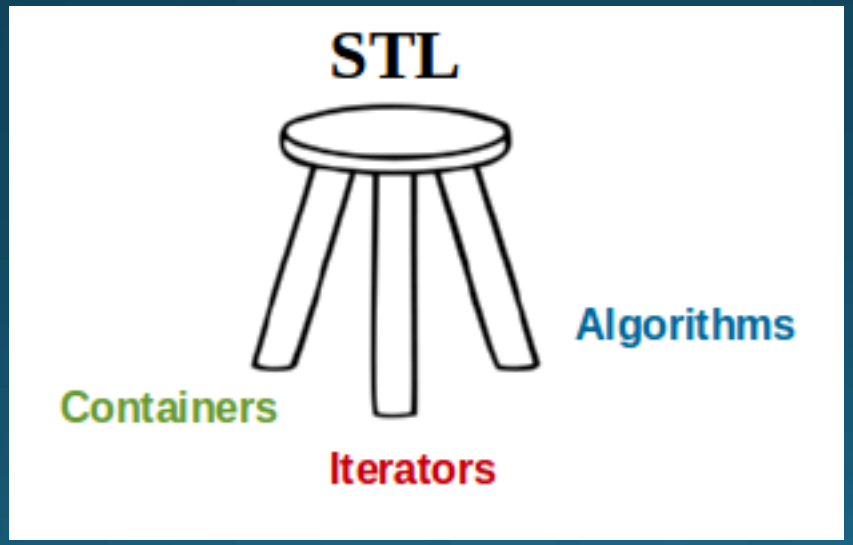
목차

STL이란 무엇인가?

- 표준 템플릿 라이브러리의 약자
- 영어로 Standard Template Library
- 일반적으로 많이 사용되는 자료구조와 알고리즘을 모은 라이브러리
- Linked list, Dynamic array, Tree, Stack, Queue 등의 자료구조와 Search, Sort 등의 알고리즘은 일반적으로 널리 사용되는데, 매번 구현이 비슷하므로 다시 만들어 쓰기가 귀찮고 번거롭다.
- 따라서 이러한 모든 종류의 자료형에 대해 일반적으로 동작하도록 잘 만든 것이 STL이다.

STL이란 무엇인가?

- Container, Iterator, Algorithms으로 구성



STL이란 무엇인가?

- C 표준 라이브러리와 비교

C 표준 라이브러리	C++ STL
printf, strcpy, atoi…	Vector, list, set, map…
함수 수준	템플릿 수준
기능	자료구조 + 알고리즘
일반화 프로그래밍 (X)	일반화 프로그래밍 (O)

왜 STL을 알아야하는가?

- Generic Programming(일반화 프로그래밍)
  I. STL은 지난 수년간의 Generic Programming 연구를 통해 얻은 산물이다.
  II. “재사용 가능하다” == “광범위하게 이곳 저곳에 두루 전용(Adaptation) 될 수 있으며 그러한 와중에도 효율(Efficiency, 성능)은 떨어지지 않는다”
  III. 높은 전용성(Adaptability)과 높은 효율성(Efficiency)이 다른 Generic Programming에 관한 연구들과 뚜렷한 차이점.
  IV. 따라서 STL은 프로그램의 유지 보수성, 확장성이 증가시키고 프로그래밍의 편의성을 제공한다.

왜 STL을 알아야하는가?

- 특히 게임 개발에서 C++은 절대적으로 강력하다.
  - 대부분의 게임 미들웨어 패키지는 C++로 만들어져있다.

    Gamebryo

    Havok

    FMOD

    SpeedTree,

    Unreal 엔진
  - C++은 gameplay 코드와 핵심엔진의 효율적인 작성을 위한 더욱 높은 수준의 추상화를 지원하기 때문.
  - 따라서 C++이 중요한 만큼 C++의 표준 라이브러리인 STL을 잘 알아야 한다.

왜 STL을 알아야하는가?

- 활용범위가 넓다.
  I. 비디오 게임
  II. 온라인 게임
  III. 스마트 폰 게임
  IV. 병렬 프로그래밍
  V. 안드로이드 NDK
  VI. 하드웨어 제어 프로그램
  VII. 고성능 프로그램
  VIII. 이외에 다양한 분야에서 사용

왜 STL을 알아야하는가?

- 기업 채용 지원자격

채용공고

입사지원 > 채용공고

제목	(주) 넥[redacted]코리아 2016년도 전문연구요원 모집				
직군/직렬/직무	게임프로그래밍 / 게임프로그래밍 / 게임프로그래머				
대상	신입	근무형태	일반직_병특	모집기간	16.01.11 ~ 채용시

지원하기

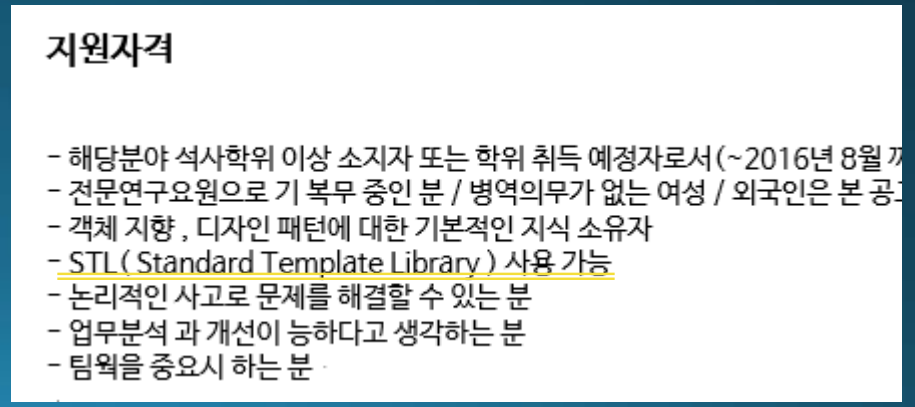
주요업무

1. 채용인원 : 전문연구요원 4명 (소속회사 : ㈜넥슨코리아)
2. 주요업무 : 게임 클라이언트 / 서버 개발

지원자격

- 해당분야 석사학위 이상 소지자 또는 학위 취득 예정자로서(~2016년 8월 까지) 전문연구요원 신규편입 대상자
- 전문연구요원으로 기 복무 중인 분 / 병역의무가 없는 여성 / 외국인은 본 공고의 대상에서 제외됩니다.
- 객체 지향 , 디자인 패턴에 대한 기본적인 지식 소유자
- STL (Standard Template Library) 사용 가능
- 논리적인 사고로 문제를 해결할 수 있는 분
- 업무분석 과 개선이 능하다고 생각하는 분
- 팀웍을 중요시 하는 분

제목	(주) 넥[redacted]코리아 2016년도 전문연구요원 모집
직군/직렬/직무	게임프로그래밍 / 게임프로그래밍 / 게임프로그래머



지원자격

- 해당분야 석사학위 이상 소지자 또는 학위 취득 예정자로서(~2016년 8월 까
- 전문연구요원으로 기 복무 중인 분 / 병역의무가 없는 여성 / 외국인은 본 공
- 객체 지향 , 디자인 패턴에 대한 기본적인 지식 소유자
- STL (Standard Template Library) 사용 가능
- 논리적인 사고로 문제를 해결할 수 있는 분
- 업무분석 과 개선이 능하다고 생각하는 분
- 팀웍을 중요시 하는 분

템플릿(Template)

- 사전적 의미
  - 형판(形板), 본보기, 틀
- C++에서 템플릿은 함수나 클래스 코드를 찍어내듯이 생산할 수 있도록 일반화(Generic) 시키는 도구
- 따라서 개별적으로 다시 작성하지 않고도 각기 다른 수많은 자료형에서 동작 할 수 있게 한다.
- 함수 템플릿, 클래스 템플릿 두가지가 있다.

템플릿(Template)

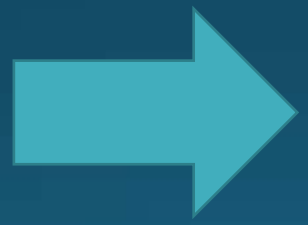
- 함수 템플릿
  - 여러 다른 자료형(int, long, float, double, class…)을 템플릿 인자로 받아, 함수 내부에서 활용할 수 있도록 한 것이다.
  - 즉, 여러 다른 자료형에 해당하는 여러 함수를 정의하지 않고 하나의 함수를 하나의 템플릿으로 표현 할 수 있다.

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

템플릿(Template)

- 함수 템플릿

```
int max(int a, int b)
 { return a > b ? a : b; }

double max(double a, double b)
{ return a > b ? a : b; }

float ...

long ....
```



```
template <typename Type>
Type max(Type a, Type b)
{ return a > b ? a : b; }
```

템플릿(Template)

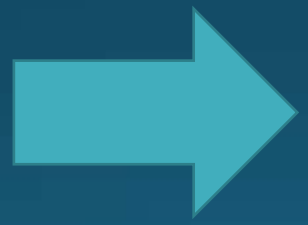
- 클래스 템플릿
  - 클래스 템플릿은 클래스를 템플릿 변수에 따라 생성할 수 있게 하는 기능이다.
  - 클래스 템플릿은 컨테이너의 용도로 많이 쓰인다.

```
template <typename type> class CLASS_NAME
{
    ..................................
};
```

템플릿(Template)

- 클래스 템플릿

```
class pair_int_char {
	public:
		int first;
		char second;
		pair_int_char(int x, char y) : first(x), second(y) {}
};

class pair_bool_double {
	public:
		bool first;
		double second;
		pair_bool_double(bool x, double y) : first(x), second(y) {}
};
```



```
template <typename T1, typename T2>
class pair {
	public:
		T1 first;
		T2 second;
		pair(T1 x, T2 y) : first(x), second(y) {}
};
```

템플릿(Template)

- 템플릿의 장점
  - 편의성
  - 코드의 재사용으로 SW 생산성, 유용성 증가
- 템플릿의 단점
  - 포팅(Porting)에 취약
  - 디버깅이 어려움
- 매크로와 차이점
  - 템플릿은 매크로와 유사한 면을 가지나, 매크로는 복잡한 함수나 클래스를 표현하는데 한계가 있다.
  - 또한 매크로는 타입 안정성이 확보되지 않아 부작용을 초래할 수 있으나, 템플릿은 타입을 검사하여 구체화 과정을 거치기 때문에 더 안전하다.

컨테이너(Container)

- 사전적 의미
  - 통, 그릇.
- STL 컨테이너는 타입이 같은, 즉 동질적인 객체의 집합을 저장하고 관리하는 역할을 한다.
- 즉, 똑같이 생긴 것들을 모아 놓는 장소로 쉽게 생각할 수 있다.
- 자료를 저장하는 방식, 삽입, 정렬, 삭제의 방식에 따라 여러가지가 있는데, 크게 3 가지로 분류된다.
  - 시퀀스 컨테이너
  - 연관컨테이너
  - 어댑터 컨테이너

컨테이너(Container)

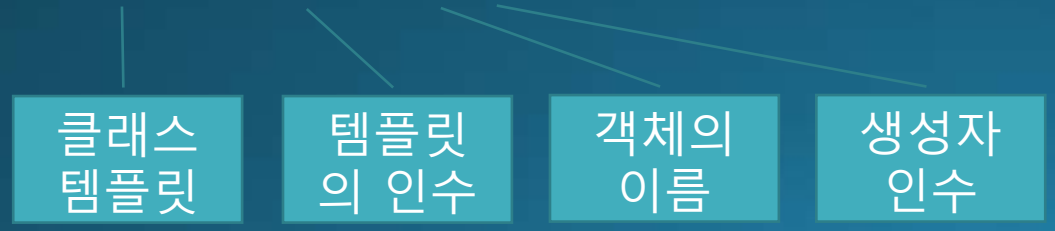
1. 시퀀스 컨테이너(Sequence Container)
   - 자료의 선형적인 집합이며 자료를 저장하는 기본 임무에 충실한 가장 일반적인 컨테이너이다.
   - 삽입된 자료를 무조건 저장하며 입력되는 자료에 특별한 제약이나 관리 규칙은 없다. 사용자는 시퀀스의 임의 위치에 원하는 요소를 마음대로 삽입, 삭제할 수 있다
     - Vector
     - List
     - Deque

컨테이너(Container)

① Vector

- 동적 배열
- 요소의 개수에 맞게 자동으로 메모리를 재할당하여 크기를 신축적으로 늘릴 수 있는 배열.
- 템플릿 기반이므로 요소의 타입에 무관한 배열을 만들 수 있다.
- 배열은 정적크기를 가지므로 상수로 크기를 정의해야 하지만, 벡터는 런타임에 결정되므로 변수로도 정의가 가능.

- vector<T> Name(n);
- vector<int> ar(5);



컨테이너(Container)

② List

- 이중 연결 리스트
- 동일한 자료의 집합을 관리한다는 측면에서는 벡터와 같다.
- 따라서 인터페이스가 동일하므로, 사용 방법이 같다.
- 순회를 통해서 원하는 요소에 접근할 수 있기 때문에 [] 연산자를 지원하지 않으며 따라서 상수시간에 액세스 할 수 없다.
- 하지만 벡터보다 삽입 삭제가 용이하고 빠르다.
- 읽기 속도가 중요하면 벡터를 선택하는 것이 좋고 삽입, 삭제가 빈번하면 리스트가 더 낫다.

컨테이너(Container)

③ Deque

- 양쪽 끝이 있는 큐
- 양끝에서 자료를 삽입, 삭제 할 수 있다.
- 조작의 위치에 따라 약간의 속도차만 있을 뿐이므로 벡터와 데크는 기능정으로 완전히 대체 가능하다.
- 벡터보다 앞쪽의 삽입, 삭제가 빠르지만, 나머지 연산들은 벡터보다 느리다.
- 따라서 뒤쪽에 추가만 한다면 벡터가 탁월한 선택이며 양쪽에서 삽입, 삭제가 발생하면 데크가 적합하다.

컨테이너(Container)

2. 연관 컨테이너(Associative Container)
   - 자료를 무조건 저장하기만 하는 것이 아니라 일정한 규칙에 따라 자료를 조직화하여 관리하는 컨테이너이다.
   - 정렬이나 해시 등의 방법을 통해 삽입되는 자료를 항상 일정한 기준(오름차순, 해시 함수)에 맞는 위치에 저장해 놓으므로 검색 속도가 빠른 것이 장점이다.
     - Set
     - Map

컨테이너(Container)

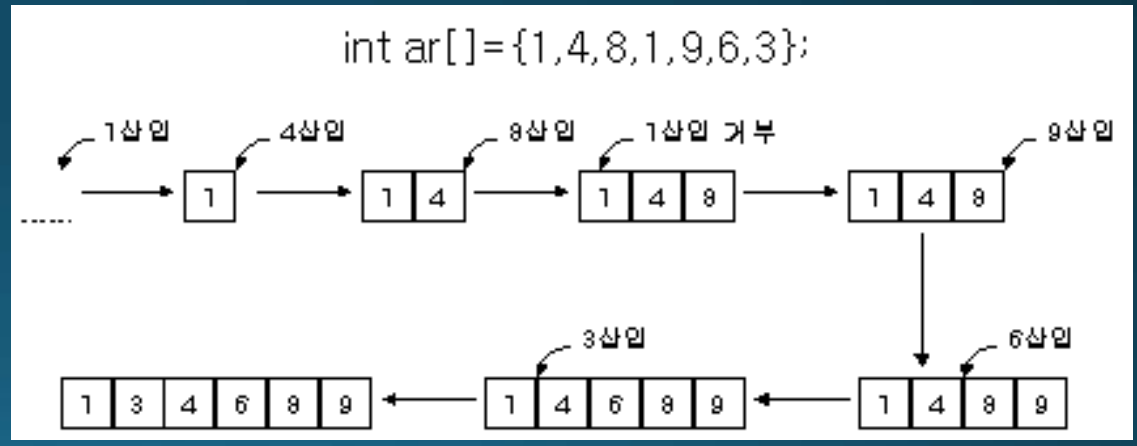
① Set

- 단어 뜻 그대로 집합을 의미.
- 동일한 타입의 데이터를 모아 놓은것.
- 저장하는 데이터 자체가 키로 사용되며 값은 저장되지 않는다.
- 동일 타입의 집합이라는 면에서 벡터와 같지만, 아무 위치에나 삽입되는 것이 아니라 정렬된 위치에 삽입된다는 차이점이 있다.
- 따라서 검색 속도가 아주 빠르다.
- 키의 중복을 허용하지 않지만 멀티셋은 중복을 허용한다.

컨테이너(Container)

① Set

```
#include <iostream>
#include <set>
using namespace std;
template<typename C> void dump(const char *desc, C c)
{ cout.width(12);cout << left << desc << "==> ";
        copy(c.begin(),c.end(),ostream_iterator<typename
C::value_type>(cout," ")); cout << endl; }
void main()
{
        int ar[]={1,4,8,1,9,6,3};
        int i;
        set<int> s;
        for (i=0;i<sizeof(ar)/sizeof(ar[0]);i++) {
                s.insert(ar[i]);
        }
        dump("원본",s);
        set<int> s2=s;
        dump("사본",s2);
        const char *str="ASDFASDFGHJKL";
        set<char> sc(&str[0],&str[13]);
        dump("문자셋",sc);
}
```



컨테이너(Container)

② Map

- 셋이 키의 집합만을 관리하는데 반해 맵은 키와 값의 쌍을 관리.
- 셋과 마찬가지로 정렬된 상태로 요소를 저장.
- 따라서 검색속도가 빠르다.
- 셋과 마찬가지로 키의 중복을 허용하지 않지만 멀티 맵은 중복을 허용한다.
- STL 컨테이너 중에서 벡터 다음으로 자주 사용되는 컨테이너이다.

컨테이너(Container)

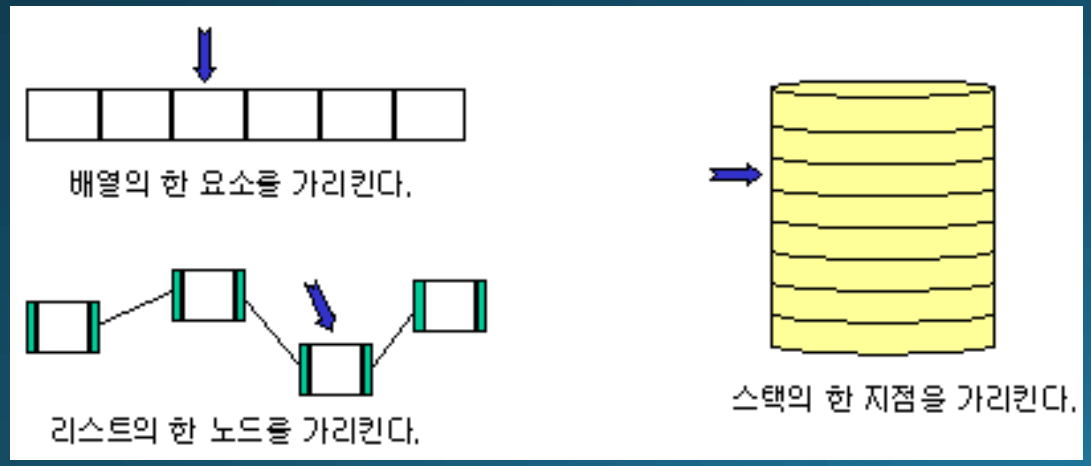
3. 어댑터 컨테이너(Adapter Container)
   - 자료를 미리 정해진 일정한 방식에 따라 관리하는 것이 특징이다.
   - 스택, 큐, 우선 순위 큐 세 가지가 있는데 스택은 항상 LIFO의 원리로 동작하며 큐는 항상 FIFO의 원리로 동작한다. 자료를 넣고 빼는 순서를 외부에서 마음대로 조작할 수 없으며 컨테이너의 규칙대로 조작해야 한다.
     - Stack
     - Queue
     - Priority Queue

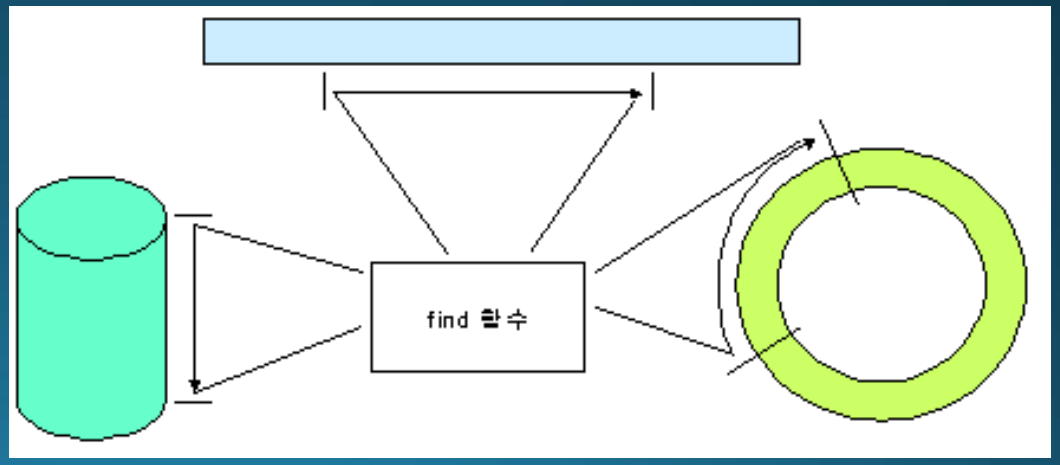
컨테이너(Container)

```
int main()
{
   int vertices, edges, v1, v2, weight;
   printf("Enter the Number of Vertices -\n");
   scanf("%d", &vertices);

   printf("Enter the Number of Edges -\n");
   scanf("%d", &edges);

   vector< list< pair<int, int> > > adjacencyList(vertices + 1);

   printf("Enter the Edges V1 -> V2, of weight W\n");

   for (int i = 1; i <= edges; ++i) {
      scanf("%d%d%d", &v1, &v2, &weight);

      // Adding Edge to the Directed Graph
      adjacencyList[v1].push_back(make_pair(v2, weight));
   }
   printf("\nThe Adjacency List-\n");
   // Printing Adjacency List
   for (int i = 1; i < adjacencyList.size(); ++i) {
      printf("adjacencyList[%d] ", i);

      list< pair<int, int> >::iterator itr = adjacencyList[i].begin();

      while (itr != adjacencyList[i].end()) {
         printf(" -> %d(%d)", (*itr).first, (*itr).second);
         ++itr;
      }
      printf("\n");
   }

   return 0;
}
```

반복자(Iterator)

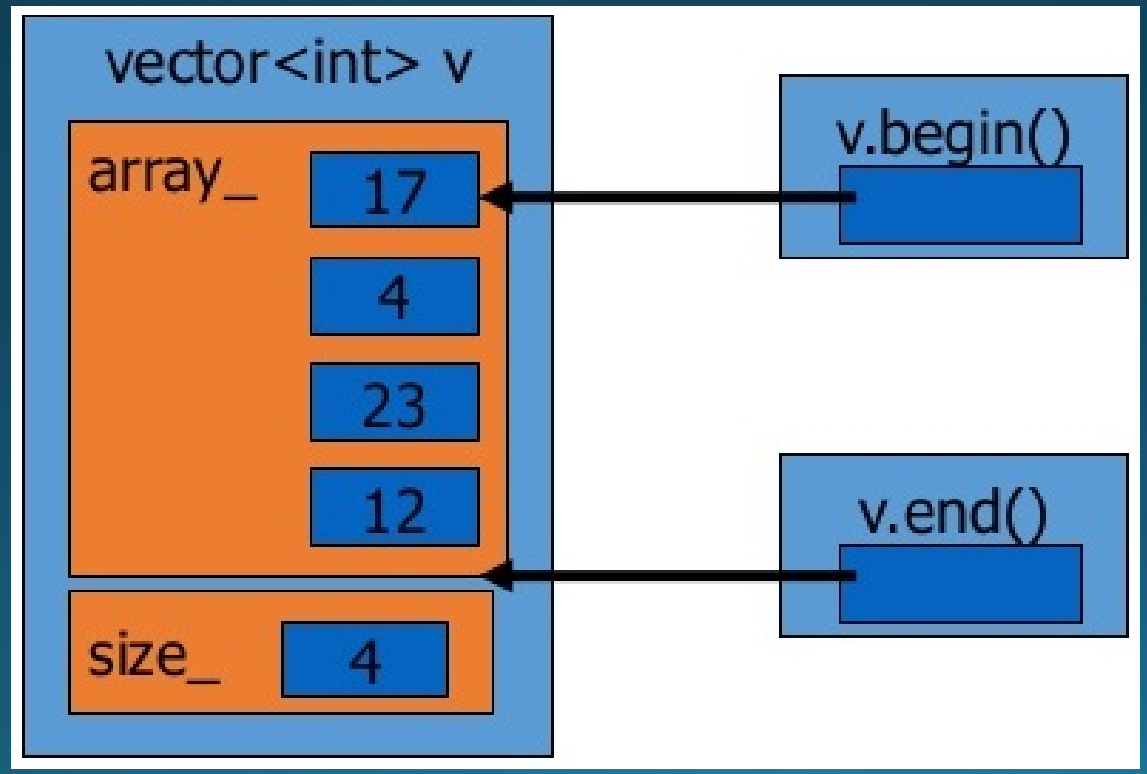
- 컨테이너의 한 지점을 가리키는 객체.
- 포인터보다 더 일반화된 개념.
- 컨테이너의 종류와 내부구조에 상관없이 한 요소를 가리킨다.
- 컨테이너와 알고리즘을 연결하는 매개체





반복자(Iterator)

- 멤버함수인 begin() 과 end()는 컨테이너의 처음지점과 끝 지점을 리턴한다.



반복자(Iterator)

기호	의미
*	현재 반복자가 참조하고 있는 아이템을 리턴.
++	반복자를 다음 아이템으로 이동시킨다.
--	반복자를 이전 아이템으로 이동시킨다.
==	두 반복자가 동일한지 비교한다.
!=	두 반복자가 다른지 비교한다.

반복자(Iterator)

```
#include <vector>
#include <cstdio>
using namespace std;
int main{
int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);

vector<int>::iterator iter = v.begin() //벡터의 반복자, 벡터의 첫번째 요소를 가리키도록 한다.
printf("First element of v = %d₩n", *iter);  //반복자의 역참조
iter++;                                      //반복자를 다음으로 이동
iter= v.end()-1;                             //반복자를 마지막으로 이동
}
```

알고리즘(Algorithm)

- 기능에 따라 읽기, 변경, 정렬, 수치 4가지로 분류.
- STL의 주요 알고리즘들은 전역 함수로 제공되며, 임의의 컨테이너에 대해 똑같은 방법으로 적용할 수 있다.
- 대부분 <algorithm.h> 헤더에 있다.
- find, for_each, copy, generate, sort, merge, accumulate 등이 있다.

알고리즘(Algorithm)

- find함수 : 컨테이너에 특정값이 존재하는지 찾는다.
- sort함수 : 퀵정렬을 사용한다.
- reverse함수 : 지정한 구간의 요소들 순서를 뒤집는다.
- count함수 : 조건에 맞는 요소의 개수를 센다.
- for_each함수 : 각 요소에 대해 지정한 작업을 한다.
- equal함수 : 구간이 일치하는지 비교한다.
- search함수 : 일치하는 부분 구간을 검색한다.
- swap함수 : 컨테이너를 교환한다.
- merge함수 : 구간을 병합하여 새로운 구간으로 복사한다.

알고리즘(Algorithm)

- Sort & Merge

```
#include <list>
int arr1[] = {6, 4, 9, 1, 7};
int arr2[] = {4, 2, 1, 3, 8};
list <int> l1(arr1, arr1+5);
list <int> l2(arr2, arr2+5);
l1.sort();  // l1 = {1,4,6,7,9}
l2.sort();  // l2 = {1,2,3,4,8}
l1.merge(/2);  // l1 = {1,1,2,3,4,4,6,7,9}, l2 = {}
```

평가

- 템플릿 기반이므로 함수와 클래스가 타입마다 매번 구체화되어 이에 따라 코드가 비대해질 수 있으며 가독성과 작성력이 떨어진다.
- 또한 STL로 작성한 코드는 가독성이 심하게 떨어지는데, 템플릿 자체가 익숙하지 않은 문법이고, 템플릿 클래스의 타입명이 길기 때문이다. 또한 다중으로 템플릿이 중첩되면 해석이 더욱 어려워진다.
- 또한 STL를 익숙하게 사용하기 위해 기본적인 지식이 많이 요구 되며 확실히 이해하지 않았을 경우 신뢰성에 문제가 생기기 때문에 이를 완전히 배우기 위한 비용이 높은 편이다.

결론

- STL은 일반화를 지원하며, 표준이므로 이식성이 좋다. 또한 소스가 공개되어 있어서 이를 분석하여 확장이 가능하다는 장점이 있다. 다양한 표준 자료구조와 알고리즘을 제공한다.
- 스텍, 큐, 우선순위 큐 같은 자료구조를 직접 구현할 필요 없이 가져와서 쓸 수 있고 멤버함수 까지 사용할 수 있다는 편리함이 학부생의 입장으로서 가장 큰 장점이라고 생각한다.
- 또한 공개되어 있는 좋은 코드들은 대부분 C++로 작성되어 있으며 STL를 많은 경우 사용하고 있기 때문에 STL을 알고 있으면 다양한 오픈 소스를 쉽게 참고 할 수 있다는 장점이 있다.

참고자료

- 네이버 블로그 (http://blog.naver.com/madplay/220187782482)
- Thinking About: C++ STL 프로그래밍 (최흥배, 한빛출판네크워크)
- STL 튜토리얼 레퍼런스 가이드 2판 (Musser, Derge, Saini 공저, 정승진 역, 인포북)
- 위키백과 (https://ko.wikipedia.org/wiki/%ED%85%9C%ED%94%8C%EB%A6%BF_(C%2B%2B))
- Software Engineering(소프트웨어 공학 연구소) (http://soen.kr/)