Standard Template Library (STL)

Computer Programming for Engineers (DSAF003-42)

Instructor:

Youngjoong Ko (nlp.skku.edu)

This Week

- Introduction
  - by Example: Vector Template Class
  - Introduction to STL
- Iterators
  - Constant, mutable, and reverse iterators
- Containers and adaptors
  - Sequential containers: vector
  - Associative Containers: set and map
  - Container adapters: stack and queue

INTRODUCTION BY EXAMPLE: VECTOR TEMPLATE CLASS

Introduction to Vectors

- Limitation of C-Arrays 배열은 크기 고정.
  - The size of static array is fixed, and should be known at compile time
  - Dynamic array needs malloc/free/new/delete, which needs to be handled with care.

- STL Vectors: "arrays that automatically grow and shrink" 자동으로 크기 축소/확장.
  - Array-like data structure dynamically resized during program execution
  - However, we do not care about the memory allocation/deallocation.
    메모리 할당/해제 신경 X

- Declared differently:
  - Syntax:
    - `std::vector<Base_Type>`
    - Produces "new" class for vectors with that type
  - Example declaration:
    - `std::vector<int> v;`

Vector Usage

```
std::vector<int> v;
```

- "v is vector of type int"
- Calls class default constructor: empty vector object created

기본생성자 호출.

- Usage
  - Indexing: indexed like arrays for access (e.g., v[0], v[1], v[k],...)
  - Adding elements: `push_back()`
  - Querying the count of elements: `size()`
  - Many other convenient member functions
    - https://www.cplusplus.com/reference/vector/vector/

Vector Example

- Display 7.7 Using a Vector (1 of 2)

```
#include <iostream>
#include <vector>
using namespace std;
int main( )
{
    vector<int> v;
    cout << "Enter a list of positive numbers.\n"
        << "Place a negative number at the end.\n";
    int next;
    cin >> next;
    while( next > 0)
    {
        v.push_back(next);
        cout << next << " added. ";
        cout << "v.size( ) = " << v.size( ) << endl;
        cin >> next;
    }
```

Vector Example

- Display 7.7 Using a Vector (1 of 2)

```
    cout << "You entered:\n";
    for (unsigned int i = 0; i < v.size( ); i++)
        cout << v[i] << " ";
    cout << endl;
    return 0;
}
```

```
Enter a list of positive numbers.
Place a negative number at the end.
2 4 6 8 -1
2 added. v.size = 1
4 added. v.size = 2
6 added. v.size = 3
8 added. v.size = 4
You entered:
2 4 6 8
```

Vector Efficiency

- Member function capacity()
  - Returns memory currently allocated 현재 할당된 메모리 반환
  - Not same as size()
  - Typically, capacity >= size
    - Automatically increased as needed 자동 증가
    - In practice, when capacity is not enough, the capacity is **doubled**.
      2배로 ↑

- If efficiency critical:
  - Can set behaviors manually

32만큼 메모리 할당

```
v.reserve(32);              // pre-set capacity to 32
v.reserve(v.size()+10);     // allocates 10 more elements
```

- Our first vector example (1_vector_basic.cpp)
  - We will vary the number of inserted items to see how capacity increases.

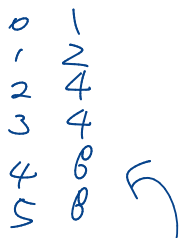
DEMO

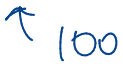
```
vector<int> v;
cout << "capacity: " << v.capacity() << endl;
int num[] = {0,1,2,3,4,5};

for(auto i : num) {
  v.push_back(i);
  cout << "after insert " << i << ", capacity: " << v.capacity() << endl;
}

for(auto i = 0; i < v.size(); i++) {
  cout << v[i] << endl;
}

cout << "capacity: " << v.capacity() << endl;
v.reserve(100);
cout << "capacity: " << v.capacity() << endl;
```





STANDARD TEMPLATE LIBRARY (STL)

Introduction

- Recall stack and queue data structures
  - We can create our own
    - In practice, we do not implement/re-invent stack/queues again.
  - A large collection of standard data structures exists.
  - Make sense to have standard portable implementations of them!
    - C++ has them! ➔ STL

- Standard Template Library (STL)
  - Software library for C++, having all such data structures
  - Four components: containers, adaptors, iterators, algorithms

Standard Containers in STL

- Simple containers
  - pair
  - tuple
- Sequence containers: ordered collections
  - `vector`: dynamic array
  - `list`: doubly linked list
  - `deque`: double-ended queue (adapted to stack and queue)
- Associative containers: unordered collections
  - `set`, `multiset`
  - `map`: dictionary (internally ordered by balanced binary tree)
  - `multimap`: similar to map, but with duplicate keys
  - `unordered_map`: dictionary with hash

Standard Adaptors in STL

- Adaptors
  - wrapping a common container to implement data structures

- Examples from `deque`
  - `stack`
  - `queue`

Efficiency of STL

- STL designed with efficiency as important consideration
  - Strives to be optimally efficient
- Example:
  - set, map elements stored in sorted order for fast searches
- Template class member functions:
  - Guaranteed maximum running time
  - Called "Big-O" notation, an "efficiency"-rating

ITERATORS

Iterators 반복자

포인터의 일반화

- Generalization of a pointer to STL containers/adaptors
  - Typically even implemented with pointer!

- "Abstraction" of iterators 반복자의 추상
  - Designed to hide details of implementation 구현 세부 정보 숨기기 설계
  - Provide consistent interface across different container classes

- Each container class has "own" iterator type 각자의 반복자 type 있음
  - Similar to how each data type has own pointer type

컨테이너에 저장된 원소를 참조할 때 접근
(Stack, Queue 에는 없음)

Manipulating Iterators

- Recall using overloaded operators:
  - `++, --, ==, !=, *`
  - So if p is an iterator variable, *p gives access to data pointed to by p

p가 가리킨 data에 액세스

- Vector template class
  - Has all above overloads
  - Also has members `begin()` and `end()`

첫번째 원소의 반복자 리턴 (실제 원소 X)

```
// return iterator for the first item in c
std::vector<int>::iterator it = c.begin();
// return iterator for after-last item in c
// e.g., for size-2 vector, end() indicates index 2
auto it2 = c.end();
```

end는 맨 마지막의 다음 번 원소임 (참조하면 런타임 에러 날 것)

Cycling with Iterators

- Recall cycling ability:
  - Using `begin()/end()`, we can write for-loop in a similar way used for arrays

```
for( auto p = c.begin(); p != c.end(); p++ )
   process(*p);   //*p is current data item
```

  - Powerful usage of `auto`!

- Keep in mind:
  - Each container type in STL has own iterator types
  - Even though they're all used similarly

Vector Cycling Example

- Display 19.1 Iterators Used with a Vector (1 of 2)

```
//Program to demonstrate STL iterators.
#include <iostream>
#include <vector>

int main( )
{
    vector<int> container;
    for (int i = 1; i <= 4; i++) container.push_back(i);
    cout << "Here is what is in the container:\n";
    vector<int>::iterator p;
```

Vector Cycling Example

- Display 19.1 Iterators Used with a Vector (2 of 2)

```
    for (p = container.begin( ); p != container.end( ); p++)
        cout << *p << " ";
    cout << endl;
    cout << "Setting entries to 0:\n";
    for (p = container.begin( ); p != container.end( ); p++)
        *p = 0;
    cout << "Container now contains:\n";
    for (p = container.begin( ); p != container.end( ); p++)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```

```
Here is what is in the container:
1 2 3 4
Setting entries to 0:
Container now contains:
0 0 0 0
```

Vector Iterator Types

- Iterators for vectors of integers are of type:

```
std::vector<int>::iterator
```

- Iterators for lists of integers are of type:

```
std::list<int>::iterator
```

Random Access

- Display 19.2
  - Bidirectional and Random-Access Iterator Use (1 of 3)

```
int main()
{
    vector<char> container;
    container.push_back('A');
    container.push_back('B');
    container.push_back('C');
    container.push_back('D');
    for (int i = 0; i < 4; i++)
        cout << "container[" << i << "] == "
             << container[i] << endl;
    vector<char>::iterator p = container.begin();

    cout << "The third entry is " << container[2] << endl;
    cout << "The third entry is " << p[2] << endl;
    cout << "The third entry is " << *(p + 2) << endl;
```

Three different notations for the same thing.

Random Access

- Display 19.2
  - Bidirectional and Random-Access Iterator Use (2 of 3)

```
    cout << "Back to container[0].\n";
    p = container.begin( );
    cout << "which has value " << *p << endl;
    cout << "Two steps forward and one step back:\n";
    p++;
    cout << *p << endl;
    p++;
    cout << *p << endl;
    p--;
    cout << *p << endl;
    return 0;
}
```

p++ moves the iterator.
So, *p will show different Results.

Random Access

- Display 19.2
  - Bidirectional and Random-Access Iterator Use (3 of 3)

```
container[0] == A
container[1] == B
container[2] == C
container[3] == D
The third entry is C
The third entry is C
The third entry is C
Back to container[0].
which has value A
Two steps forward and one step back:
B
C
B
```

- Iterator (2_vector_cycling.cpp, 3_vector_random_access.cpp)
  - Cycling and Random Access


```
// Cycling
for (auto it = container.begin(); it != container.end( ); it++)
  cout << *it << " ";
cout << endl;

for (auto it = container.begin(); it != container.end( ); it++)
  *it = 0;
```

이것도 가능~

```
// Random Access
// Setting the last item with zero
// Why should we use --?
*--it = 0;
cout << *it << endl;

// Setting the second item with zero
// Why -2 is used here?
it[-2] = 0;
cout << *it << endl;
```

마지막의 한 칸 뒤 가리키니까 당겨서 마지막을 초기화

원래 1234인데 1230이 됨.

1230 ↑ 여기 가리키니까 두칸 앞으로 ㄱㄱ.

Iterator Classifications

- Forward iterators:
  - ++ works on iterator
- Bidirectional iterators:
  - Both ++ and -- work on iterator
- Random-access iterators:
  - ++, --, and random access all work with iterator

  세 다 작동
- These are kinds of “iterators”, not types!

CONTAINERS

Containers

- 순차 (vec. deque. list)
- 컨테이너 어댑터 (stack. queue. 우선순위 큐)
- 연관 (set. map. ...)

- Container classes in STL
  - Different kinds of data structures
  - Like lists, queues, stacks

각각 저장할 데이터 유형에 대한 매개변수 O.

- Each with parameter for particular data type to be stored
  - e.g., Lists of ints, doubles or myClass types

- Each has own iterators
  - One might have bidirectional, another might just have forward iterators

Sequential Containers

- Arranges list data 데이터 정렬.
  - 1st element, next element, … to last element

- Linked list is sequential container 링크드리스트는 순차컨테이너.
  - Earlier linked lists were "singly linked lists"
    - One link per node

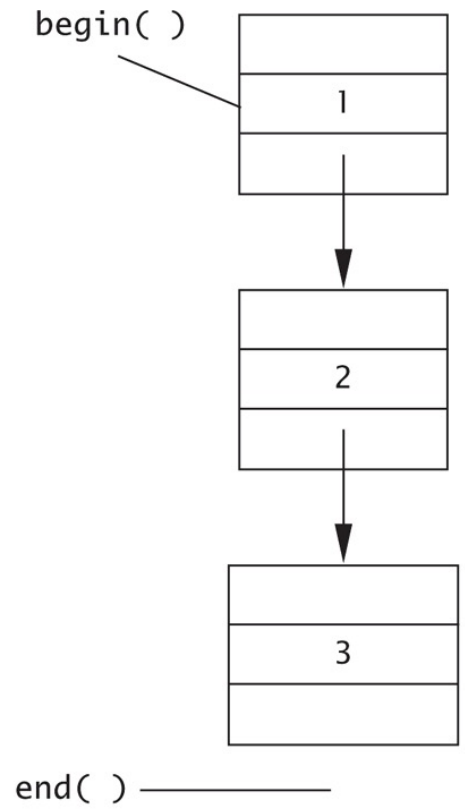
- STL has no "singly linked list" STL은 단방향 링크드리스트가 아님.
  - Only "doubly linked list": template class *list*

ONLY 양방향!!

Display 19.4 Two Kinds of Lists

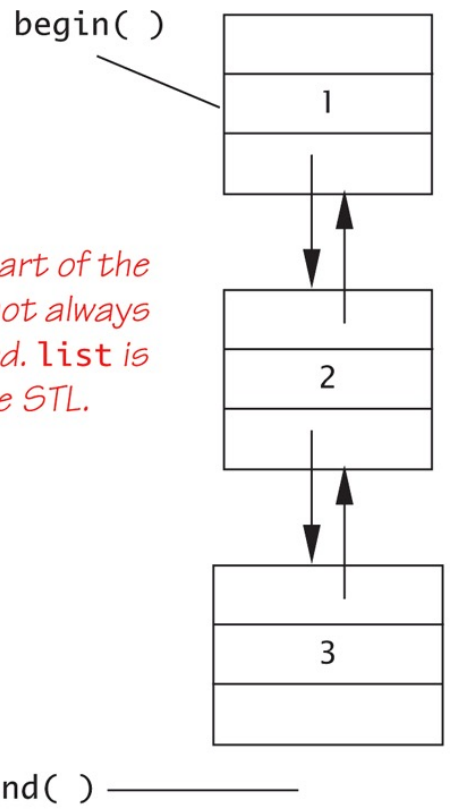
Display 19.4 Two Kinds of Lists

`slist`: *A singly linked list*
++ defined; -- not defined



`slist` *is not part of the STL and may not always be implemented.* `list` *is part of the STL.*

`list`: *A doubly linked list*
Both ++ and -- defined



list Template Class

- Display 19.5 Using the list Template Class(1 of 2)

```
#include <iostream>
#include <list>
using std::cout;
using std::endl;
using std::list;

int main( )
{
    list<int> listObject;
    for (int i = 1; i <= 3; i++)
        listObject.push_back(i);

    cout << "List contains:\n";
    list<int>::iterator iter;
    for (iter = listObject.begin( ); iter != listObject.end( ); iter++)
        cout << *iter << " ";
    cout << endl;
```

list Template Class

- Display 19.5 Using the list Template Class(2 of 2)

```
    cout << "Setting all entries to 0:\n";
    for (iter = listObject.begin( ); iter != listObject.end( ); iter++)
       *iter = 0;
    cout << "List now contains:\n";
    for (iter = listObject.begin( ); iter != listObject.end( ); iter++)
       cout << *iter << " ";
    cout << endl;
    return 0;
}
```

```
List contains:
1 2 3
Setting all entries to 0:
List now contains:
0 0 0
```

- List (5_list.cpp)
  - Source code is shown in the prior slides.


```
// Random access is not defined.
//iter = listObject.begin();
//cout << iter[2] << endl; // Error
```

Container Adapters: stack and queue

- Container adapters are template classes
  - Implemented "on top of" other classes
- Example: stack template class implemented on top of deque template class by default
- Others: queue, priority_queue

기존의 컨테이너 기반으로 구현

Deque 위에 구현된 stack

반복자 X

Specifying Container Adapters

원래 기본 컨테이너 (ex. stack → deque) 있는데

- Adapter template classes have "default" containers underneath
  - But we can specify different underlying container 바꿀 수 있음 X
  - Examples:
    stack template class → any sequence container
    priority_queue → default is vector, could be others
- Implementing Example:

```
stack<int, vector<int>>
```

스택에 대한 벡터 기본 컨테이너 만듦.

  - Makes vector underlying container for stack

set Template Class

- Simplest container possible
- Stores elements without repetition 요소를 반복없이 저장
- Each element is own key 각 요소는 자체의 키
- Capabilities:
  - Add elements
  - Delete elements
  - Ask if element is in set
    set 안에 있는지.

class template

std::set <set>

```
template < class T,                        // set::key_type/value_type
           class Compare = less<T>,        // set::key_compare/value_compare
           class Alloc = allocator<T>      // set::allocator_type
           > class set;
```

Set

Sets are containers that store unique elements following a specific order.

In a set, the value of an element also identifies it (the value is itself the *key*, of type T), and each value must be unique. The value of the elements in a set cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.

Internally, the elements in a set are always sorted following a specific *strict weak ordering* criterion indicated by its internal comparison object (of type Compare).

set containers are generally slower than unordered_set containers to access individual elements by their *key*, but they allow the direct iteration on subsets based on their order.

Sets are typically implemented as *binary search trees*.

set Template Class Example

- Display 19.12 Program Using the set Template Class (1 of 3)

```
//Program to demonstrate use of the set template class.
#include <iostream>
#include <set>
using std::cout;
using std::endl;
using std::set;

int main( )
{
    set<char> s;
    s.insert('A');
    s.insert('D');
    s.insert('D');
    s.insert('C');
    s.insert('C');
    s.insert('B');
```

Set Template Class Example

- Display 19.12 Program Using the set Template Class (2 of 3)

```
cout << "The set contains:\n";
set<char>::const_iterator p;

for (p = s.begin( ); p != s.end( ); p++)
   cout << *p << " ";
cout << endl;

cout << "Set contains 'C': ";
if (s.find('C')==s.end( ))
   cout << " no " << endl;
else cout << " yes " << endl;

cout << "Removing C.\n";
s.erase('C');
```

Set Template Class Example

- Display 19.12 Program Using the set Template Class (3 of 3)

```
    for (p = s.begin( ); p != s.end( ); p++)
        cout << *p << " ";
    cout << endl;
    cout << "Set contains 'C': ";

    if (s.find('C')==s.end( ))
        cout << " no " << endl;
    else cout << " yes " << endl;
    return 0;
}
```

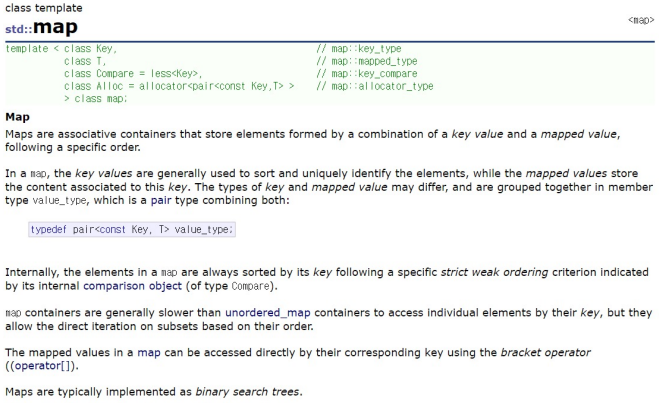
```
The set contains:
A B C D
Set contains 'C': yes
Removing C.
A B D
Set contains 'C': no
```

Map Template Class

- A function given as set of ordered pairs 순서쌍의 집합으로 주어진 func.
  - For each value *first*, at most one value *second* in map considering (*first*, *second*) pair
- Example map declaration:

```
map<string, int> numberMap;
```

- Can use [ ] notation to access the map [ ]로 액세스 가능
  - For both storage and retrieval 저장 & 검색용
- Stores in sorted order, like set
  - Second value can have no ordering impact



class template

std::**map** <map>

```
template < class Key,                                // map::key_type
           class T,                                  // map::mapped_type
           class Compare = less<Key>,                // map::key_compare
           class Alloc = allocator<pair<const Key,T> >    // map::allocator_type
           > class map;
```

Map

Maps are associative containers that store elements formed by a combination of a *key value* and a *mapped value*, following a specific order.

In a `map`, the *key values* are generally used to sort and uniquely identify the elements, while the *mapped values* store the content associated to this *key*. The types of *key* and *mapped value* may differ, and are grouped together in member type `value_type`, which is a `pair` type combining both:

```
typedef pair<const Key, T> value_type;
```

Internally, the elements in a `map` are always sorted by its *key* following a specific *strict weak ordering* criterion indicated by its internal comparison object (of type `Compare`).

`map` containers are generally slower than `unordered_map` containers to access individual elements by their *key*, but they allow the direct iteration on subsets based on their order.

The mapped values in a `map` can be accessed directly by their corresponding key using the *bracket operator* (`operator[]`).

Maps are typically implemented as *binary search trees*.

Map Template Class Example

- Display 19.14 Program Using the map Template Class(1 of 3)

```
#include <iostream>
#include <map>
#include <string>
using std::cout;
using std::endl;
using std::map;
using std::string;

int main( )
{
    map<string, string> planets;
    planets["Mercury"] = "Hot planet";
    planets["Venus"] = "Atmosphere of sulfuric acid";
    planets["Earth"] = "Home";
    planets["Mars"] = "The Red Planet";
    planets["Jupiter"] = "Largest planet in our solar system";
```

Map Template Class Example

- Display 19.14 Program Using the map Template Class(2 of 3)

```
planets["Saturn"] = "Has rings";
planets["Uranus"] = "Tilts on its side";
planets["Neptune"] = "1500 mile-per-hour winds";
planets["Pluto"] = "Dwarf planet";

cout << "Entry for Mercury - " << planets["Mercury"]
   << endl << endl;

if (planets.find("Mercury") != planets.end( ))
   cout << "Mercury is in the map." << endl;
if (planets.find("Ceres") == planets.end( ))
   cout << "Ceres is not in the map." << endl << endl;

cout << "Iterating through all planets: " << endl;
```

Map Template Class Example

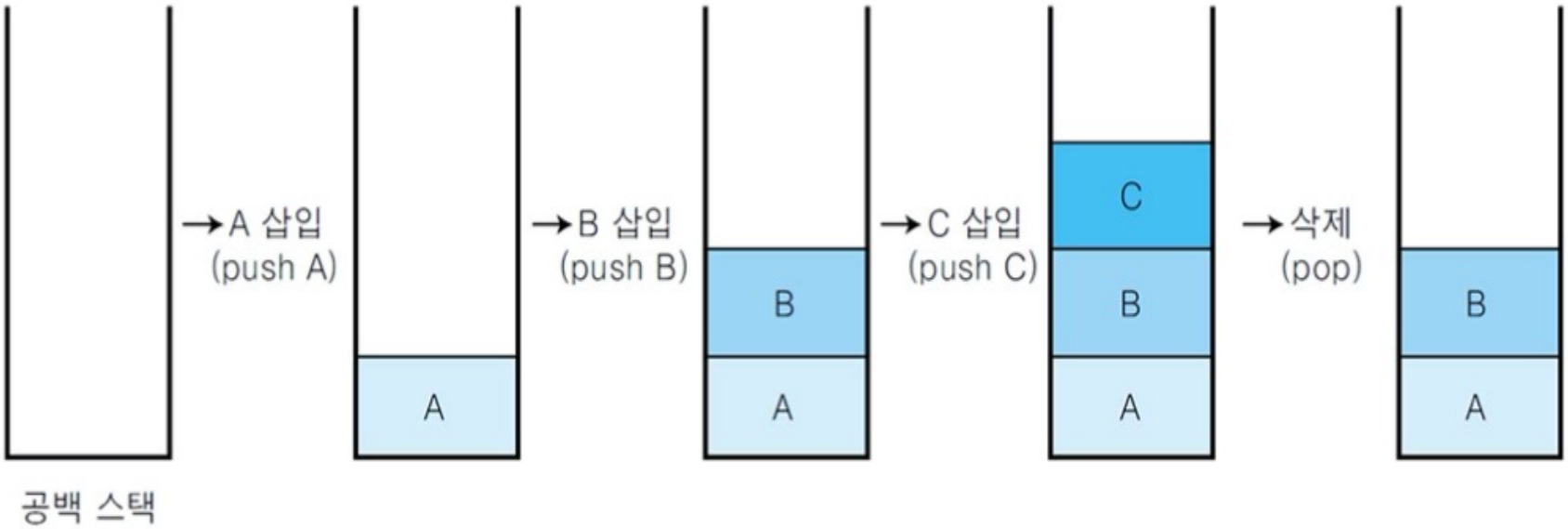
- Display 19.14 Program Using the map Template Class(2 of 3)

```
    map<string, string>::const_iterator iter;
    //The iterator will output the map in order sorted by the key.
    for (iter = planets.begin( ); iter != planets.end( ); iter++)
    {
        cout << iter->first << " - " << iter->second << endl;
    }
    return 0;
}
```

키 기준으로 정렬된 상태로 출력.

```
Entry for Mercury - Hot planet
Mercury is in the map.
Ceres is not in the map.
Iterating through all planets:
Earth - Home
Jupiter - Largest planet in our solar system
…
```

Stack



Stack (6_stack.cpp)


```
using std::stack;
using std::vector;
int main( )
{
    stack<char, vector<char>> s; // We change the default container from deque to vector.
    char next; cin.get(next);    // cin.get is used for accepting spaces and new line.
    while (next != '\n') {
        s.push(next); cin.get(next);
    }
    while ( ! s.empty( ) ) {
        cout << s.top( ); s.pop( );
    }
    cout << endl;

    // Stack has no iterator
    //auto it = s.begin();
    //cout << *it << endl;
    return 0;
}
```

기본 바꿈

공백 & 새줄 받아들이는데 사용

input 문자열이 reverse 돼서 나옴.

반복자 지원 X