

for(;;)

- c++에서 auto를 사용하면
- for(auto, iterator)를 사용하면 vector< int > :: iterator it
- for(auto x: nums)에서 x는 nums의 요소 (cout << x)
- x는 const
- for(auto &x:nums)에서 x는 nums의 요소
- x는 const
- const_iterator (const_iterator)
- const_iterator (const_iterator)
- vector, deque, string, +=, <=, >=
- set, map, list, multiset, multimap

string

string

- string(); string str;
- string(const char *s); string s
- string(const string& str); string str
- string(int n, char c); string n

string

string str1, str2;

string

- str1 = "hello";
- str2 = str1;

string

- str1.assign("hello");
- str1.assign("hello", 3); hello를 str1

string

- str1 += "hello"; str1
- str1.append("hello"); str1
- str2.append(str1, int pos, int n); str1 pos n str2

string

- s.find(string str, int pos); s pos-1
- s.rfind(); s
- s.replace(int pos, int n, string str); s pos n str

string

- s.at(ASCII)

- = 0
- > 1
- < -1

- s.compare(str) s与str

string

- str[1] 下标为1的字符
- s.at(i) 下标为i的字符

字符串操作

- s.insert(pos, str); 在pos位置插入str
- s.erase(pos, n); 从pos位置开始删除n个字符

字符串子串

- s.substr(pos, n); 从pos位置开始取n个字符，返回string

字符串常量

s.size(); s.copy(); s.length(); s.assign(); s.append(); s.find(); s.rfind(); s.replace(); s.compare();
s.at(); s.insert(); s.erase();

vector(动态数组)

vector是C++标准库中的一个容器，用于存储同类型的元素。vector的底层实现是动态数组，可以动态地增加和减少元素。vector的常用操作包括：v.begin(), v.end(), v.rbegin(), v.rend(), v.size(), v.capacity(), v.empty(), v.resize(), v.push_back(), v.pop_back(), v.insert(), v.erase(), v.clear()等。

声明

- vector< int > v;
- vector(v.begin(), v.end()); 从v.begin()到v.end()的范围内初始化vector

初始化

- 初始化=使用vector
- .assign() 初始化
- 初始化{} 使用vector

常用操作

- .empty(); 是否为空 false 非空 true
- .capacity() 容量 >= .size() 大小
- .resize(int num, (int elem)); 将vector的大小调整为num，不足的部分用elem填充

增删改查

- .push_back(); 在末尾添加元素
- .pop_back(); 删除末尾元素
- .insert(pos, ele); 在pos位置插入ele
- .erase(pos); 删除pos位置的元素
- .clear(); 清空vector

迭代器

- 迭代器 arr[1];
- .front(); 返回第一个元素

- `.back()`; 返回最后一个元素

交换

- 交换两个 vector
- `.swap(vec)`;
- 交换 swap 两个 vector `vector< int >(v).swap(v) vector< int > (v)` 交换

保留

- `.reserve(int len)`; 将 len 保留到下一个 push_back 操作

其他

```
.begin(),.end().rbegin(),.rend() .assign(); .empty(); .capacity(); .size(); .resize(); .push_back();
.pop_back(); .insert(); .erase(); .front(); .back(); .reserve();
```

deque(双端队列)

双端队列 类似 vector

stack(栈)

栈 后进先出

操作

```
.push(); .pop(); .top(); .size(); .empty();
```

queue(队列)

队列 先进先出

操作

```
.push(); .pop(); .size(); .empty(); .front(); .back();
```

list(链表)

链表 类似 list 容器 可以随机访问 (通过 [] 操作)

操作

- `.assign()`; 将 list 容器赋值为 1,10
- `.swap()`;
- `list = list`

其他

- `.insert(pos,elem)`; 在 pos 位置插入 elem
- `.erase(pos)`; 删除 pos 位置
- `.remove(elem)`; 删除 elem 元素

其他

- `.reverse()`; 反转

- `.sort()`; `l1.sort()`; `l1`의 원소들이 오름차순으로 정렬된다

vector

```
.assign(); .swap(); .push_front(); .pop_front(); .push_back(); .pop_back(); .empty(); .size(); .resize();  
.insert(); .erase(); .clear(); .remove(); .reverse(); .sort();
```

set/multiset

- `set` (중복 불가)
- `multiset` (중복 가능)
- `set`의 원소들은 오름차순으로 정렬된다
- `multiset`의 원소들은 오름차순으로 정렬된다

set

- `set` = `multiset`의 특수한 경우

multiset

`set.insert()`

- `.erase()`;
 - `.erase(pos)` : `pos` 위치의 원소 삭제
 - `.erase(elem)` : `elem` 원소 삭제

find

- `.find(key)`; `key`가 있는 원소의 위치를 반환한다. 없으면 `end()`를 반환한다.
- `.count(key)`; `key`가 있는 원소의 개수를 반환한다.

resize

- `resize`

pair

- `set`의 원소들이 `pair` 타입일 때 `set< pair< int, int > >`를 사용한다.
- `first` : 첫 번째 원소

make_pair

- `pair< type, type > p (value1, value2);`
- `pair< type, type > p = make_pair(value1, value2);`

first

- `.first()`; `pair`의 첫 번째 원소
- `.second()`; `pair`의 두 번째 원소

second

set의 원소들이 pair 타입일 때



alt text

```
set< pair< int, int > > s; s.insert(make_pair(1, 2)); s.insert(make_pair(2, 3)); s.insert(make_pair(3, 4));
```

set

```
.insert(); .size(); .empty(); .swap(); .clear(); .erase(); .find(); .count();
```

map/multimap

- map 저장하는 pair
- pair 저장하는 key(값) 저장하는 value(값)
- key 저장하는 value
- map 저장하는 key(값) (value 저장)
- multimap 저장하는 key
- map key 저장

map

- map< int , int > m;

map

- m = map
- m.insert(pair< int , int >)(key,value);

map

- .insert();
 - m.insert(pair< int ,int >)(key,value);
 - m.insert(make_pair(key,value));
 - m[key] = 20; key 저장하는 m[key] value 저장
- .erase();
 - .erase(key); key 저장하는 value
 - .erase(pos); 저장하는 value
- .erase();

map

- .find(); 저장

map

- set

map

map map

- s.begin()->first(second);
- (*s,begin()).first(second);
- key m[key];

map

.size(); .empty(); .swap(); .clear(); .insert(); .erase(); .find(); .count();

map()

- 저장
- 저장()

map

- bool
- operator()

- `operator()` 메서드
- `operator()` 메서드

예제

functional 예제

- `plus`
- `minus`
- `multiplies`

예제

- `plus< T >`: 덧셈
- `minus< T >`: 뺄셈
- `multiplies< T >`: 곱셈
- `divides< T >`: 나눗셈
- `modulus< T >`: 모듈로
- `negate< T >`: **negate** 메서드

```
plus< int > v; // v(50,10); // v(50,10)
```

예제

- `bool equal_to< T >`: `==`
- `bool not_equal_to< T >`: `!=`
- `bool greater< T >`: `>`
- `bool greater_equal< T >`: `>=`
- `bool less< T >`: `<`
- `bool less_equal< T >`: `<=` **sort** 메서드

예제

- `bool logical_and< T >`: `&&`
- `bool logical_or< T >`: `||`
- `bool logical_not< T >`: `!`

STL 예제

`< algorithm >` `< functional >` `< numeric >`

예제

- `for_each` 메서드
 - `for_each(beg, end, _func);`
 - `beg`: 시작 위치 `end`: 종료 위치 `_func`: 함수 (함수 객체)
 - `_func` 메서드
- `transform` 메서드
 - `transform(beg1, end1, beg2, _func);`
 - `beg1`: 시작 위치 `end1`: 종료 위치 `beg2`: 시작 위치 `_func`: 함수 (함수 객체)
 - `transform` 메서드

예제

- `find` 메서드
 - `find(beg, end, value)`
 - `beg`: 시작 위치 `end`: 종료 위치 `value`: 값

- 返回迭代器, 返回end
- 返回迭代器
- find_if
 - find_if(beg, end, _prec)
 - beg: 迭代器 end: 迭代器 _prec: 谓词 (返回 bool)
- adjacent_find
 - adjacent_find(beg, end)
 - 返回迭代器
 - beg: 迭代器 end: 迭代器
- binary_search
 - 返回 bool
 - binary_search(beg, end, value)
 - beg: 迭代器 end: 迭代器
 - 返回 bool
- count
 - count(beg, end, value)
 - beg: 迭代器 end: 迭代器 value: 值
 - 返回 int const
- count_if
 - count_if(beg, end, _prec)
 - beg: 迭代器 end: 迭代器 _prec: 谓词 (返回 bool)

排序

- sort; 排序
 - 排序
- random_shuffle; 随机打乱
 - random_shuffle (beg, end, value)
 - beg: 迭代器 end: 迭代器
- merge; 合并
 - merge(beg1, end1, beg2, end2, dest)
 - beg1: 迭代器1 end1: 迭代器1 beg2: 迭代器2 end2: 迭代器2 dest: 迭代器
 - 返回迭代器
- reverse; 反转
 - reverse(beg, end)
 - beg: 迭代器 end: 迭代器

替换

- copy; 复制
 - merge(beg, end, dest)
 - beg: 迭代器 end: 迭代器 dest: 迭代器
- replace; 替换
 - replace(beg, end, oldvalue, newvalue)
 - beg: 迭代器 end: 迭代器 oldvalue: 旧值 newvalue: 新值
- replace_if;
 - replace_if(beg, end, _pred, newvalue)
 - beg: 迭代器 end: 迭代器 _pred: 谓词 newvalue: 新值

- swap;
 - swap(c1,c2)
 - c1: 1 c2: 2 1 2 swap

1 2 3 4 5 6

- 1 < numeric >
- accumulate; 1 2 3 4 5 6
 - accumulate(beg,end,value)
 - beg: 1 end: 6 value: 1
- fill; 1 2 3 4 5 6
 - fill(beg,end,value)
 - beg: 1 end: 6 value: 1

1 2 3 4 5 6

1 2 3 4 5 6

- set_intersection; 1 2 3 4 5 6
 - set_intersection(beg1,end1,beg2,end2,dest)
 - beg1: 1 end1: 6 beg2: 2 end2: 5 dest: 1 2 3 4 5 6
 - 1 2 3 4 5 6 (size)
 - 1 2 3 4 5 6
- set_union; 1 2 3 4 5 6
 - set_union(beg1,end1,beg2,end2,dest)
 - beg1: 1 end1: 6 beg2: 2 end2: 5 dest: 1 2 3 4 5 6
 - 1 2 3 4 5 6 (size)
 - 1 2 3 4 5 6 (size)
- set_difference; 1 2 3 4 5 6
 - set_difference(beg1,end1,beg2,end2,dest)
 - beg1: 1 end1: 6 beg2: 2 end2: 5 dest: 1 2 3 4 5 6
 - 1 2 3 4 5 6 (size)
 - 1 2 3 4 5 6 (size)
 - 1 2 3 4 5 6 v1 v2 v1