(추가) STL Algorithm - Sorting

- sort(): 해당 컨테이너의 원소를 정렬함
- partial_sort(): 범위 내의 원소를 정렬함
- nth_element(): 컨테이너에서 지정한 원소가 n번째가 되도록 정렬함
- 참고 영상
  - https://youtu.be/2olsGf6JIkU?t=967

CppCon 발표 영상에서 해당 함수의 동작을 애니메이션으로 확인해 보세요!

Jonathan Boccara, "105 STL Algorithms in Less Than an Hour", 2018, CppCon
https://www.fluentcpp.com/getthemap/

STL Algorithm - Partitioning

- partition(): 조건식이 true가 되는 원소가 false가 되는 원소보다 앞에 있도록 정렬함
- partition_point(): 해당 위치 앞에 나온 원소가 조건식에 대해 모두 true이며, 해당 위치 뒤에 나온 원소가 모두 조건식에서 false가 되는 반복자를 반환함

STL Algorithm - Permutation

- rotate(): 주어진 컨테이너를 두 개로 나눠서 앞부분과 뒷부분의 위치를 변경함
- shuffle(): 주어진 컨테이너의 원의 순서를 무작위로 바꿈
- reverse(): 주어진 컨테이너의 순서로 반대로 변경함
- next_permutation(): 정렬된 컨테이너를 이용해서 사전 순서로 다음에 나오는 순열로 변환함
- prev_permutation(): 정렬된 컨테이너를 이용해서 사전 순서로 이전에 나오는 순열로 변환함

STL Algorithm – Numerical Processing

- count(): 주어진 컨테이너에서 특정 값의 원소나 조건식을 만족하는 원소의 개수
- accumulate(): 주어진 컨테이너의 모든 원소를 누적함 ($\sum n$, 합계는 + 연산자, $\prod n$, 시퀀스의 곱은 * 연산자)
  - std::accumulate(v.begin(), v.end(), 0);
  - std::accumulate(v.begin(), v.end(), 1, std::multiplies<int>());
- partial_sum(): 주어진 컨테이너의 각 원소에서 해당 원소의 앞에 나온 모든 원소의 이항 연산자 (더하기, 곱하기 등) 결과의 누적값의 컨테이너를 반환함
- inner_product(): 두 개의 컨테이너에서 위치가 같은 원소를 이항 연산(곱셈 등) 한 결과값을 또 다른 이항 연산(덧셈 등)에 적용해 결과를 누적하여 반환함
- 영상참고
  - https://youtu.be/2olsGf6JIkU?t=1398

STL Algorithm – Querying

- all_of(): 주어진 컨테이너의 모든 원소에 대해 조건식이 true 이면 true 를 반환함
- any_of(): 주어진 컨테이너의 원소 중 하나라도 조건식이 true 이면 true 를 반환함
- none_of(): 주어진 컨테이너의 모든 원소에 대해 조건식이 false이면 true를 반환함

STL Algorithm – Value Modifiers

- fill(): 주어진 컨테이너를 특정 값으로 채움
- generate(): 주어진 컨테이너에 함수의 반환값을 채움
  - istream_iterator<int>(cin);
  - generate_n(back_inserter(vec), size, [ ]( ) { return *(ist); });
- iota(): 주어진 컨테이너를 시작값에서 연속해서 증가하는 값으로 채움
- replace(): 주어진 컨테이너에서 지정한 값과 일치하는 원소를 다른 값으로 변경
- 참고영상
  - https://youtu.be/2olsGf6JIkU?t=2370

STL Algorithm – Movers

- copy(): 주어진 컨테이너의 원소를 다른 컨테이너로 복사함
  - ostream_iterator<int> ost(cout, " ");
  - copy(begin(vec), end(vec), ost);
- move(): 주어진 컨테이너의 원소를 다른 컨테이너로 복사함 (move semantics)
- swap_ranges(): 주어진 두 개의 컨테이너의 원소들을 서로 맞바꿈
- back_inserter(): push_back 동작을 지원하는 컨테이너의 끝 지점에 값을 입력함
  - copy_if(begin(vec), end(vec), back_inserter(vec2),
    [=](int i) { return i < N; });
- front_inserter(): push_front 동작을 지원하는 컨테이너의 시작 지점에 값을 입력함
- 참고영상
  - https://youtu.be/2olsGf6JIkU?t=2177

STL Algorithm – Structure Changers

- remove(): 주어진 컨테이너에서 특정 값과 일치하는 원소를 삭제함 (컨테이너의 크기는 변경되지 않음)
  - erase-remove idiom
  - vec.erase(remove(vec.begin(), vec.end(), 5), vec.end());
- erase() : 주어진 컨테이너에서 특정 값과 일치하는 원소를 모두 삭제함 (Since C++20)
- unique(): 주어진 컨테이너에서 연속적으로 중복되는 값 들 중에서 첫번째 원소를 제외하고 나머지를 제거함 (컨테이너의 크기는 변경되지 않음)
- 참고영상
  - https://youtu.be/2olsGf6JIkU?t=2469

- XXX_copy: XXX 연산 결과를 새로운 컨테이너에 복사함
- XXX_if : 조건에 맞는 원소에만 XXX 연산이 적용됨
- XXX-n: XXX 연산을 n번 수행함

STL Algorithm – Lonely Islands

- for_each(): 주어진 컨테이너의 각 원소별로 함수를 수행함
  - 함수 수행의 결과가 컨테이너와 상관 없이 일반적일 수 있으므로 side-effects 발생 가능성
- transform(): 주어진 컨테이너의 각 원소에 대해 함수를 수행한 결과를 그 원소의 위치에 저장함
  - transform(begin(vec), end(vec), back_inserter(results),
    []auto& i){ return i*i; });
- 참고영상
  - https://youtu.be/2olsGf6JIkU?t=2756

Good Design:
STL 알고리즘을 적절히 사용하자!

- STL 알고리즘은 코드를 더 표현적으로 만들 수 있다!
- for 루프문을 알맞은 algorithm 함수로 바꿔라!
- STL 알고리즘의 기술적인 면을 이해하자!
  - 알고리즘적 복잡도(complexity)
  - 사전/사후 전제조건
  - 구현을 봐라!
- 추상화가 잘 동작하는 것을 이해하라!
- 자신만의 알고리즘을 작성하라!

예제 : 개발자의 의도를 더욱 잘 표현하고, for 문을 알고리즘 함수로 변경할 수 있는 경우인가 ?

- Modern C++ 프로그래머로 가는 길의 작은 출발점
  - 1부터 10까지 합계를 구하는 프로그램

```
int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int sum = std::accumulate(begin(vec), end(vec), 0);
    std::cout << sum << '₩n';
}
```

  - N개의 원소에서 X보다 작은 수를 출력하는 프로그램

```
int main() {
    int N, X;  cin >> N >> X;
    // 표준 입력으로부터 n개의 원소를 벡터에 저장함
    vector<int> vec;
    generate_n(back_inserter(vec), N, [ ](){ return *(istream_iterator<int>{cin});});
    //벡터에서 조건을 만족하는 원소만 표준 출력으로 복사함
    ostream_iterator<int> ost(cout, " ");
    copy_if(begin(vec), end(vec), ost, [=](int i) { return i < X; });
}
```

Q&A