실습 13 - 14주차

학과 : 전자공학과 학번 : 2023104322 이름 : 현시온

- 과제는 pdf로 변환하여 제출(과제 문서 첫 줄에 학과/학번/이름 포함)
- 과제는 순서대로 작성하며, 문제와 설명을 모두 포함(형식이 맞지 않으면 감점)
- 프로그램을 작성하는 문제는 소스코드와 실행 결과를 모두 text로 붙여넣기(그림으로 포함하지 말 것)하고 코드 설명 및 결과에 대한 설명을 포함해야 함
- 문의 사항은 이메일(nize@khu.ac.kr) 또는 오픈 카톡방을 이용

1. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
#include <vector>
#include <list>


int main() {
 std::vector<int> v{ 10, 20, 30, 40, 50 };
 std::list<int> l{ 1, 2, 3, 4, 5 };
```

//10, 20, …, 50 (정수형)을 엘리먼트로 하는 벡터 v와 1, 2, …, 5 (정수형)을 엘리먼트로 하는 리스트 l 생성.

```
 for (std::vector<int>::iterator iter = std::begin(v); iter != std::end(v); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//정수형을 엘리먼트로 하는 벡터의 iterator iter을 v의 첫번째 엘리먼트의 주소부터 하여 마지막 엘리먼트의 주소까지 차례대로 옮겨가면서 iter의 대상, 즉 v의 엘리먼트들을 차례대로 출력한다.

```
 for (auto iter = std::begin(v); iter != std::end(v); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//iter의 타입을 v와 호환하도록 직접 설정해주지 않고, auto를 통해 직접 맞출 수 있도록 설정한다. 결과는 같다.

```
 for (auto iter = v.begin(); iter != v.end() ; iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//std::begin(v)와 v.begin()은 소속은 달라도 동일한 역할을 하기 때문에 결과는 같다.

```
 for (std::vector<int>::reverse_iterator iter = v.rbegin(); iter != v.rend(); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//iter의 타입이 reverse iterator이므로, 할당해줄 값 또한 reverse iterator여야 한다. 그러므로 rbegin과 rend라는 메소드를 써주어야 한다. Reverse iterator는 역순으로 컨테이너의 엘리먼트를 접근하기 위한 목적으로 사용하기 때문에, 결과는 v의 엘리먼트들을 역

순으로 출력하게 된다.

```
 for (auto iter = v.rbegin(); iter != v.rend(); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//iter에 할당되는 값인 v.rbegin()이 결국 reverse iterator와 호환되는 값이므로 iter의 타입은 자연스럽게 reverse iterator로 결정되어 바로 위 코드 부분과 같은 결과를 출력한다.

```
 for (auto iter = std::begin(l); iter != std::end(l); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//이번엔 인수를 리스트 l로 하여 스탠다드 메소드 begin과 end를 불렀다. 즉 이번 iter의 타입은 정수형을 엘리먼트로 하는 리스트의 iterator인 것이다. 나머지는 윗 코드들과 동일한 메커니즘으로 작동하기 때문에, 결과는 리스트 l의 엘리먼트들을 차례대로 출력하게 된다.

```
 for (auto iter = l.begin(); iter != l.end(); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//마찬가지로 스탠다드 메소드 begin과 컨테이너의 메소드 begin은 같은 역할을 하기에, 결과는 동일하다.

```
 for (auto iter = l.rbegin(); iter != l.rend(); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
```

//rbegin이 reverse iterator를 반환하는 메소드이므로 iter은 reverse iterator이고, 결과는 리스트 l의 엘리먼트들을 역순으로 출력하게 된다.

```
}
```

출력 결과:

```
10, 20, 30, 40, 50,
10, 20, 30, 40, 50,
10, 20, 30, 40, 50,
50, 40, 30, 20, 10,
50, 40, 30, 20, 10,
1, 2, 3, 4, 5,
1, 2, 3, 4, 5,
5, 4, 3, 2, 1,
```

2. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>


template <class container>
```

```
void Print(container c) {
 for (auto iter = c.begin(); iter != c.end(); iter++)
 std::cout << *iter << ", ";
 std::cout << std::endl;
}
```

//컨테이너 c를 파라미터로 하는 템플릿 함수 Print 선언. 이 함수가 호출될 때 입력 받은 인수가 컨테이너 자료형이면 그 자료의 엘리먼트들을 차례대로 출력한다.

```
int main() {
 std::vector<int> v1{ 10, 20, 30, 40, 50 };
 Print(v1);
```

//10, 20, …, 50을 엘리먼트로 하는 벡터 v1을 생성하고, 함수 Print를 호출하면 정상적으로 v1의 엘리먼트들이 차례대로 출력된다.

```
 v1.erase(v1.begin() + 3);
 Print(v1);
```

//erase 메소드를 통해 v1.begin() + 3에 위치한 엘리먼트, 즉 1+3=4번째 엘리먼트를 지우고 그 뒤의 엘리먼트들을 앞당긴다. 즉 Print를 호출하면 10, 20, 30, 50이 출력된다.

```
 v1.erase(v1.begin() + 2, v1.end());
 Print(v1);
 std::cout << std::endl;
```

//같은 방식으로 40이 지워진 v1에서 3번째 엘리먼트부터 마지막 엘리먼트까지 전부 지운다. 즉 Print를 호출하면 10, 20이 출력된다.

```
 std::vector<int> v2{ 10, 0, 30, 0, 50 };
 Print(v2);
```

//10, 0, 30, 0 , 50을 엘리먼트로 하는 벡터 v2를 생성하고, 함수 Print를 호출하면 v2의 엘리먼트들이 차례대로 출력된다.

```
 auto remove_start = std::remove(v2.begin(), v2.end(), 0);
 Print(v2);
```

//remove 함수를 통해 v2의 모든 엘리먼트의 범위에서 0을 지운 결과의 크기만큼의 엘리먼트 범위까지만 v2에 덮어씌우고, remove_start에 지운 결과의 마지막 인덱스 + 1을 할당한다(end와 동일). 즉 Print를 호출하면 10, 30, 50, 그 다음엔 남은 값인 0, 50이 출력된다.

```
 v2.erase(remove_start, v2.end());
 Print(v2);
 std::cout << std::endl;
```

//remove_start는 크기가 3인 벡터의 end 값과 동일하다. 즉 erase 메소드를 통해 v2의 4번째 엘리먼트부터 마지막 엘리먼트까지 전부 지운다. 즉 Print를 호출하면 10, 30, 50이 출력된다.

```
 std::vector<int> v3{ 10, 0, 30, 0, 50 };
 Print(v3);
```

//처음 v2와 동일하게 v3를 생성하고 함수 Print로 출력한다.

```
 v3.erase(std::remove(v3.begin(), v3.end(), 0), v3.end());
 Print(v3);
 std::cout << std::endl;
```

//remove 함수가 v3를 10, 30, 50, 0 ,50으로 수정시키고, v3의 3번째 인덱스 주소 + 1을 반환한다. 그리고 이를 erase 메소드가 받아 v3의 4번째 엘리먼트부터 끝까지 지운다. 즉 Print로 출력하면 10, 30, 50이 출력된다.

```
 std::list<int> l{ 10, 0, 30, 0, 50 };
 Print(l);
```

//처음 v2와 동일한 엘리먼트를 가지는 리스트 l을 생성하고, 함수 Print를 호출한다. Print는 템플릿 함수이므로 인수가 컨테이너 자료형이라면 대부분 호환되어 엘리먼트들을 차례대로 호출할 수 있다.

```
 l.erase(std::remove(l.begin(), l.end(), 0), l.end());
 Print(l);
}
```

//remove 함수가 l을 10, 30, 50, 0 ,50으로 수정시키고, l의 3번째 인덱스 주소 + 1을 반환한다. 그리고 이를 erase 메소드가 받아 l의 4번째 엘리먼트부터 끝까지 지운다. 즉 Print로 출력하면 10, 30, 50이 출력된다. 이것은 벡터의 경우와 똑같이 작동하는 것을 보여주는데, remove와 erase 또한 우리가 만든 Print 처럼 컨테이너 자료형과 모두 호환 가능한 템플릿 함수 또는 메소드임을 예측할 수 있다.

출력 결과:

```
10, 20, 30, 40, 50,
10, 20, 30, 50,
10, 20,

10, 0, 30, 0, 50,
10, 30, 50, 0, 50,
10, 30, 50,

10, 0, 30, 0, 50,
10, 30, 50,

10, 0, 30, 0, 50,
10, 30, 50,
```

3. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
#include <numeric>

template <class iterator>
void Print(iterator begin, iterator end) {
 for (iterator iter = begin; iter != end; iter++)
```

```
 std::cout << *iter << ", ";
 std::cout << std::endl;
}
```

//iterator begin과 end를 파라미터로 하는 템플릿 함수 Print를 정의. begin과 end 직전에 위치한 모든 대상을 차례대로 출력하게 된다.

```
int main() {
 std::vector<int> v1{ 10, 20, 30, 40, 50 };
 Print(v1.begin(), v1.end());
 Print(v1.begin(), v1.end()-2);
```

//10, 20, …, 50을 엘리먼트로 하는 벡터 v1을 생성하고, v1의 begin과 end 메소드, 그리고 Print 함수를 통해 v1의 모든 엘리먼트들을 차례대로 출력할 수 있다. 한편 v1.end()-2는 (v1의 마지막 인덱스 + 1 – 2)인 인덱스의 주소를 가리키기 때문에 첫번째 엘리먼트부터 세번째 엘리먼트까지만 출력하게 한다.

```
 int sum = std::accumulate(v1.begin(), v1.end(), 0);
 std::cout << sum << std::endl;
```

//함수 accumulate를 통해 sum에는 v1의 모든 엘리먼트들을 더한 값이 할당된다. 즉 150이 출력된다.

```
 sum = std::accumulate(v1.begin(), v1.end()-2, 0);
 std::cout << sum << std::endl;
}
```

//위 코드를 통해 v1.end()-2가 무엇을 의미하는지 알았으므로, 함수 accumulate를 통해 v1의 첫번째 엘리먼트부터 세번째 엘리먼트까지만 더한 값이 sum에 할당될 것이라는 것을 알 수 있다. 즉 60이 출력된다.

출력 결과:

10, 20, 30, 40, 50,

10, 20, 30,

150

60