Lab 6

https://padlet.com/dcslabcp/0419-8v3iei1n9ifcvs5n

온라인 질문 사이트

STL
Standard Template Library

vector & list

1. STL이란

- Standard Template Library
- C++을 위한 라이브러리
- 프로그램에 필요한 자료구조와 알고리즘을 Template으로 제공
  - Template 덕분에 어떠한 데이터 타입도 사용이 가능하다.

Template이란? https://en.wikipedia.org/wiki/Template_(C%2B%2B)

- 장점
  - 일반화 지원 : 다양한 데이터 타입에 대해 동일한 코드 사용 가능
  - 안정성 : 표준이기 때문에 안정적이다.
  - 효율성 : 효율적으로 구현되어 있어 빠르게 동작한다.

2. STL 종류

- 시퀀스 컨테이너(Sequence Container)
  - 자료를 입력하는 순서대로 저장한다.
  - 삽입 및 삭제 속도가 빠르다. 검색 속도가 느리다.
  - 많지 않은 양의 자료 / 검색 속도가 중요하지 않은 경우 사용한다.
  - 종류 : array, **vector**, **list**, deque
- 정렬 연관 컨테이너(Associative Container)
  - 트리 구조로 이루어져 있다.
  - 삽입 및 삭제 속도가 느리다. 검색 속도가 빠르다.
  - 많은 양의 자료 / 빠른 검색이 중요한 경우 사용한다.
  - 종류 : **set**, multiset, map, multimap

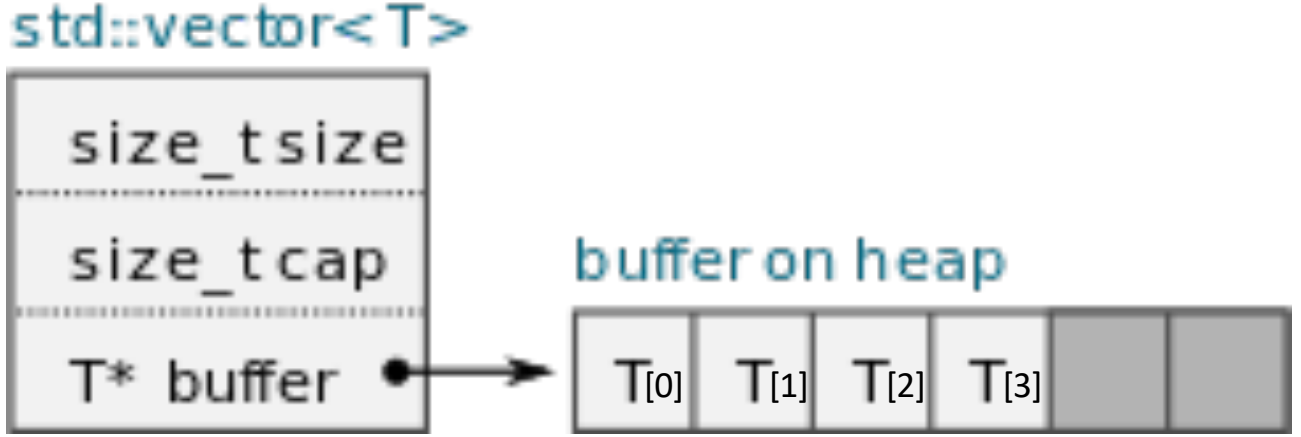
3. 벡터(vector) vs 리스트(list)

- 공통점
  - 저장할 개수가 가변적일때 사용한다.
  - 배열(array)에 비해 성능이 떨어짐
- 차이점

	원소들의 메모리 주소	중간 데이터 삽입/삭제	데이터 랜덤 접근
벡터	연속	어려움	저장 위치 알고 있으면 바로 접근
리스트	비연속	쉬움	저장 위치 알고 있어도 순차 접근

4. 벡터(vector) 자료 구조

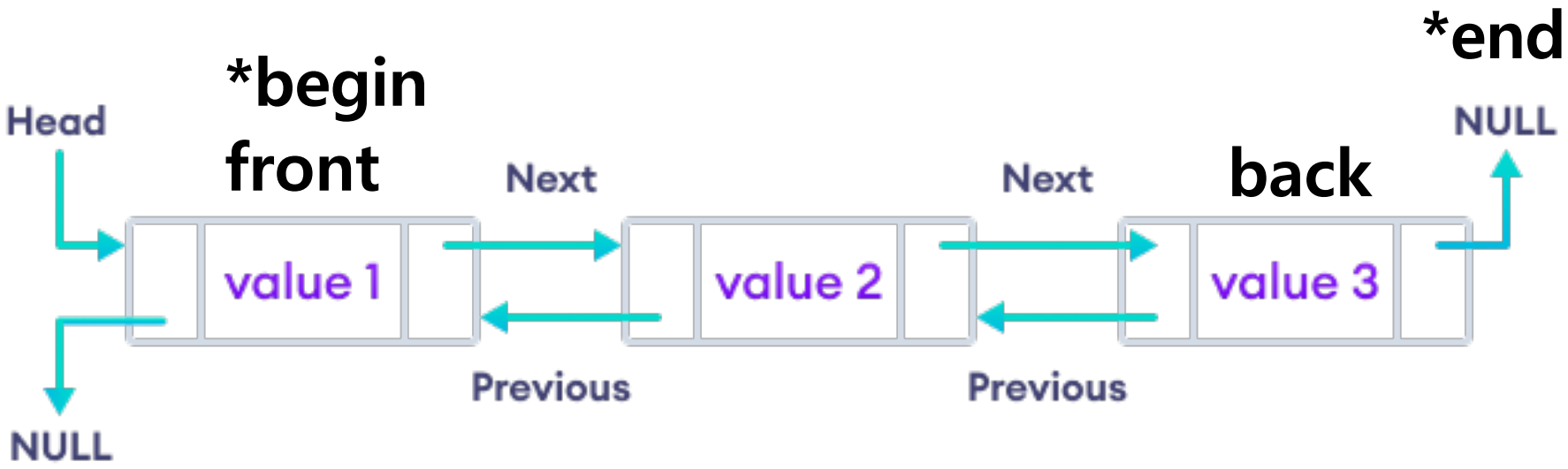
- std::vector 객체(instance)는 힙에 할당된 버퍼에 대한 포인터, 벡터의 크기와 용량을 추적하는 변수들을 포함한다.
- 벡터 capacity가 초과될 경우 새로운 메모리 주소에 buffer를 재할당



size=4 capacity=6

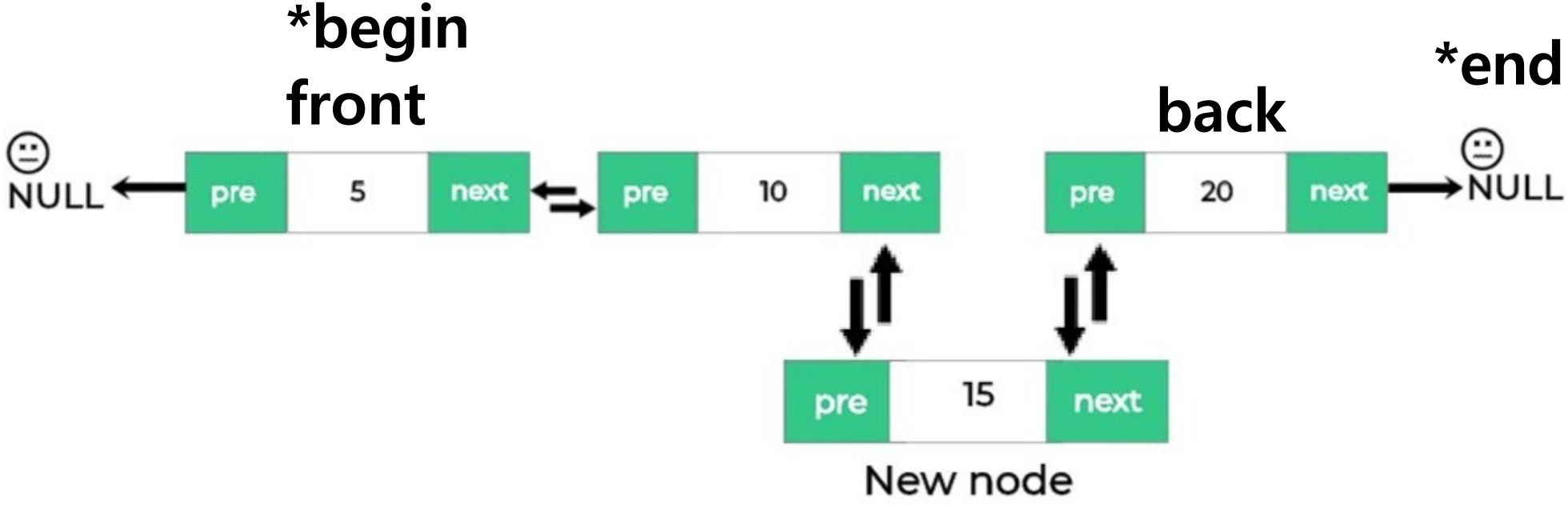
5. 리스트(list) 자료 구조

- double linked list와 동일한 자료 구조
- 각 원소에 이전 원소와 다음 원소의 주소를 포함하고 있다.



5. 리스트(list) 삽입

- 새로운 원소의 이전 원소의 next 주소가 새로운 원소 주소
- 새로운 원소의 다음 원소의 prev 주소가 새로운 원소 주소



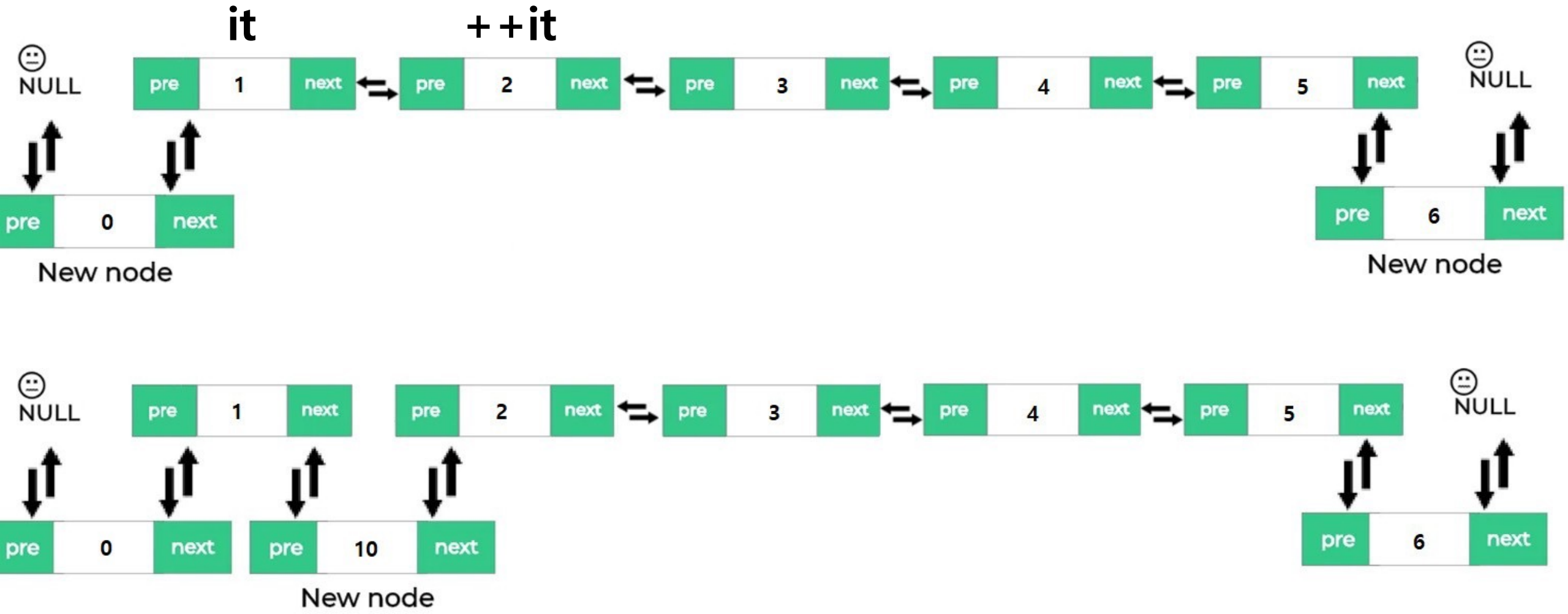
예제(list) 1

- list를 선언하고 초기화
- it는 list의 첫번째 원소로 초기화
- list back에 6 추가
- list front에 0 추가

- insert는 it 위치 앞으로 들어간다
- 10은
  list의 어느 자리에 들어갈까?

```
1   #include <iostream>
2   #include <list>
3
4   using namespace std;
5
6   int main(){
7       list<int> mylist = {1,2,3,4,5};
8       list<int>::iterator it;
9       it = mylist.begin();
10      mylist.push_back(6);
11      mylist.push_front(0);
12      ++it;
13      mylist.insert(it,10);
14      cout << "Size of mylist:" << mylist.size() << endl;
15      for (it=mylist.begin(); it!=mylist.end();++it)
16          cout << ' ' <<*it;
17      cout << '\n';
18  }
```

예제(list) 1


it
++it
NULL
pre
1
next
pre
2
next
pre
3
next
pre
4
next
pre
5
next
NULL
pre
0
next
pre
6
next
New node
New node
NULL
pre
1
next
pre
2
next
pre
3
next
pre
4
next
pre
5
next
NULL
pre
0
next
pre
10
next
pre
6
next
New node

추가 팁

① ②
```
for (it=mylist.begin(); it!=mylist.end();++it)
    cout << ' ' <<*it;
cout << '\n';
for (int e: mylist)
    cout << ' ' << e;
cout << '\n';
```

- 두개의 for 문은 동일하게 출력하지만 내부 동작은 다르다.

① it 에는 각 원소의 주소가 저장 (list 원소 데이터 읽기/쓰기 가능)
② e 에는 각 원소의 값이 저장 (list 원소 데이터 읽기만 가능)

잘못된 코드 예제

```
1   #include <iostream>
2   #include <list>
3
4   using namespace std;
5
6   int main(){
7       list<int> mylist;
8       list<int>::iterator it;
9       for (int i = 1; i <= 100; i++)
10          mylist.push_back(i);
11      for (it = mylist.begin(); it != mylist.end(); ++it){
12          if(*it % 3 == 0)
13              mylist.erase(it);
14      }
15      cout << mylist.size() << endl;
16  }
```

- 원래는 리스트에 int 1~100의 100개 원소를 삽입하고
- 값이 3의 배수인 원소들을 제거해서 size를 출력하고자 했다.
  - mylist.erase(it)는 mylist의 it 위치에 있는 원소를 제거한다.
- 코드를 실행시키면 버그가 발생한다. 문제가 무엇일까?

잘못된 코드 예제

```
1   #include <iostream>
2   #include <list>
3
4   using namespace std;
5
6   int main(){
7       list<int> mylist;
8       list<int>::iterator it;
9       for (int i = 1; i <= 100; i++)
10          mylist.push_back(i);
11      for (it = mylist.begin(); it != mylist.end(); ++it){
12          if(*it % 3 == 0)
13              mylist.erase(it);
14      }
15      cout << mylist.size() << endl;
16  }
```

- it 위치에 있는 원소를 삭제하고
- for 문으로 가서 다음 원소를 부르면?
- 다음 원소의 주소를 갖고 있던 it 위치의 원소가 삭제돼서 다음 원소의 주소를 못 찾는다.

잘못된 코드 예제

```
1   #include <iostream>
2   #include <list>
3
4   using namespace std;
5
6   int main(){
7       list<int> mylist;
8       list<int>::iterator it;
9       for (int i = 1; i <= 100; i++)
10          mylist.push_back(i);
11      for (it = mylist.begin(); it != mylist.end(); it++){
12          if(*it % 3 == 0){
13              it = mylist.erase(it);
14          }
15      }
16      cout << mylist.size() << endl;
17  }
```

- it 위치에 있는 원소를 삭제하고 다음 원소 위치를 return한다.
- for 문으로 가서 다음 원소를 부르면?
- 다다음원소로 넘어간다.

올바른 코드 예제

```
1   #include <iostream>
2   #include <list>
3
4   using namespace std;
5
6   int main(){
7       list<int> mylist;
8       list<int>::iterator it;
9       for (int i = 1; i <= 100; i++)
10          mylist.push_back(i);
11      for (it = mylist.begin(); it != mylist.end(); it++){
12          if(*it % 3 == 0){
13              it = mylist.erase(it);
14              it--;
15          }
16      }
17      cout << mylist.size() << endl;
18  }
```

- it 위치에 있는 원소를 삭제하고 다음 원소 위치를 return한다.
- for문으로 가기전에 이전 원소로 옮겨서
- for 문으로 가서 다음 원소를 부르면 순차적으로 호출할 수 있다.

실습(list) 1

- list를 선언하고 int 값 1~100의 100개 원소로 할당한다.
  - hint) 빈 list에 push_back해도 문제없음
- list에서 3의 배수 원소들은 삭제한다.
  - mylist.remove_if(predicate)는 predicate 조건문을 만족하는 원소를 삭제한다.
  - predicate 조건문을 함수로 만들어서 bool 값을 return하자.
- list의 원소 개수를 확인한다.
  - mylist.size()

실습(list) 1

```cpp
#include <iostream>
#include <list>

using namespace std;

int main(){
    list<int> mylist;
    list<int>::iterator it;
    for (int i = 1; i <= 100; i++)
        mylist.push_back(i);
    mylist.remove_if([](int num){return num % 3 == 0;});
    cout << mylist.size() << endl;
}
```

```cpp
#include <iostream>
#include <list>

using namespace std;

bool isMultiple3(int num){
    return num % 3 == 0;
}

int main(){
    list<int> mylist;
    list<int>::iterator it;
    for (int i = 1; i <= 100; i++)
        mylist.push_back(i);
    mylist.remove_if(isMultiple3);
    cout << mylist.size() << endl;
}
```

- 위, 아래 둘 다 가능하다.
- 위의 예제는 람다 함수라고 불리며
  [](매개변수){// 함수 동작}(호출 시 인자);
- 형식을 가지고있다.
- 특징으로 함수 이름이 없다.

실습(list) 2

- vector를 선언하고 int 값 1~100의 100개 원소로 할당 뒤 랜덤하게 섞는다.
- vector의 원소들을 하나씩 list에 오름차순으로 삽입한다.
- list의 원소를 순서대로 출력한다.
- 주석 부분을 지우고 적절한 코드로 채우세요.

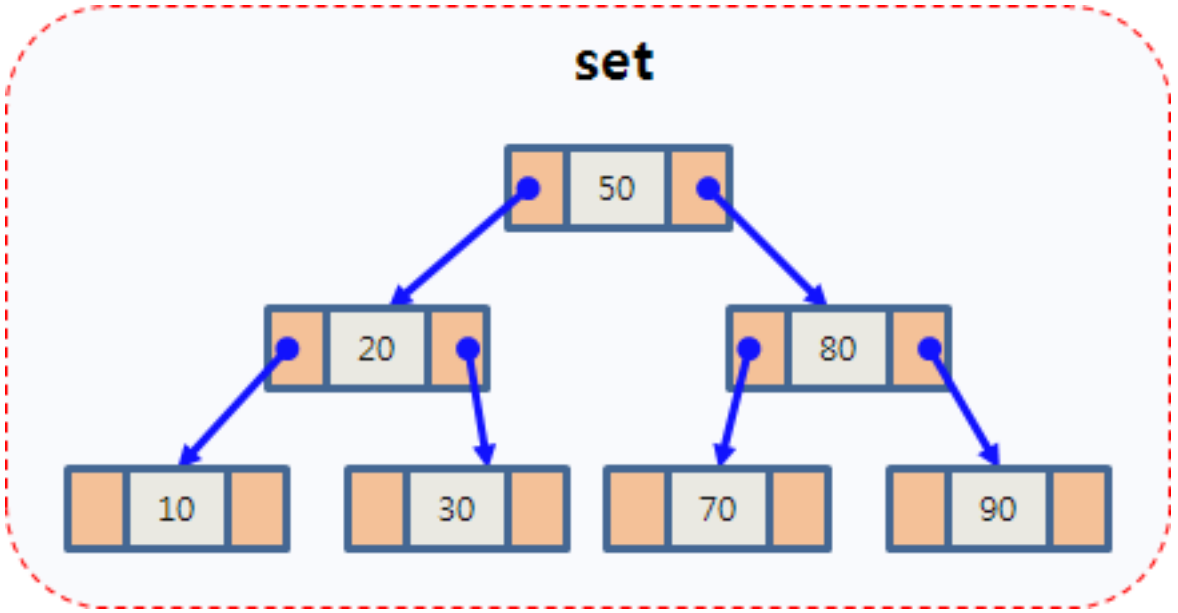
```cpp
#include <iostream>
#include <vector>
#include <random>
#include <algorithm>
#include <list>

using namespace std;

int main(){
    vector<int> v(100);
    for (int i=0; i<100; ++i)
        v[i]=i+1;
    random_device rd;
    mt19937 g(rd());
    shuffle(v.begin(), v.end(),g);

    list<int> mylist;
    list<int>::iterator it;

    for (int num: v){
        if (mylist.empty()){
            // mylist가 비어있을 경우
            continue;
        }
        if (num < mylist.front()){
            // num이 mylist의 맨 앞 원소보다 작을 경우
            continue;
        }
        if (num > mylist.back()){
            // num이 mylist의 맨 뒤 원소보다 클 경우
            continue;
        }
        for (it = mylist.begin(); it != mylist.end(); ++it){
            if(num < *it){
                // num이 mylist의 it 위치 원소보다 작을 경우
                break;
            }
        }
    }
    for (int e: mylist)
        cout << e << ' ';
    cout << endl;
}
```

실습(list) 2

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

using namespace std;

int main(){
    vector<int> v(100);
    for (int i=0; i<100; ++i)
        v[i]=i+1;
    random_device rd;
    mt19937 g(rd());
    shuffle(v.begin(), v.end(),g);

    list<int> mylist;
    list<int>::iterator it;

    for (int num: v){
        if (mylist.empty()){
            mylist.push_back(num);
            continue;
        }
        if (num < mylist.front()){
            mylist.push_front(num);
            continue;
        }
        if (num > mylist.back()){
            mylist.push_back(num);
            continue;
        }
        for (it = mylist.begin(); it != mylist.end(); ++it){
            if(num < *it){
                mylist.insert(it, num);
                break;
            }
        }
    }
    for (int e: mylist)
        cout << e << ' ';
    cout << endl;
}
```

6. set 자료구조

- 값이 정렬 되어있어 빠르게 값을 찾아낼 수 있다.
- 저장 데이터의 중복이 허용되지 않는다. (multiset은 중복 허용)



예제(set)

```cpp
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main(){
    vector<int> v = {9,7,2,3,4,2,2,4,8,1,9};
    set<int> s;
    pair<set<int>::iterator,bool> ret;

    for(int i : v) {
        ret = s.insert(i);
        if(ret.second == 0) cout << "중복된 값 : "<< i << endl;
    }
    cout<<"set : ";
    for(int i : s) cout << i << " ";
    cout << endl;
}
```

- set이 중복 데이터를 허용하지 않는 특징을 이용하는 예시
- 임의의 int vector를 선언하고 vector의 원소들을 set에 넣으면서 중복된 원소들은 삽입 실패
- s.insert(i)는 iterator와 성공여부를 return한다.
  두번째 값은 입력에 성공하면 1, 실패하면 0을 return 한다.

응용 문제(list) : 요세푸스 문제

- 1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있다.
- 자연수 K(≤ N)가 주어진다.
- 순서대로 K번째 사람을 제거한다.
- 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다.
- 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다.

- N=7, K=3 이라고 가정하고 코드를 작성한다.
- 출력은 제거된 순서인 3 6 2 7 5 1 4 이다.

```cpp
#include <iostream>
#include <list>

using namespace std;

int main(void) {
    int N = 7, K = 3;
    list<int> mylist;

    for (int i = 1; i <= N; i++)
        mylist.push_back(i);
    list<int>::iterator it = mylist.begin();

    while (!mylist.empty()){









}
```

응용 문제(list) : 요세푸스 문제

- iterator를 K-1 번 순차적으로 이동시킨다.
- list는 end 다음 요소를 begin으로 인식하지 않기때문에 end에서 begin으로 iterator를 직접 옮겨줘야 한다.
- iterator가 가리키는 요소를 제거하면 iterator가 다음 요소로 이동한다.
- iterator가 가리키는 요소가 end라면 iterator를 begin으로 옮긴다.

```cpp
#include <iostream>
#include <list>

using namespace std;

int main(void) {
    int N = 7, K = 3;
    list<int> mylist;

    for (int i = 1; i <= N; i++)
        mylist.push_back(i);
    list<int>::iterator it = mylist.begin();

    while (!mylist.empty()){
        /*
        iterator를 K-1번 순차적으로 이동한다.
        이동할때 iterator가 end에 있으면 begin으로 옮긴다.
        iterator가 가리키는 요소를 제거한다.
        만약 iterator가 end에 있으면 begin으로 옮긴다.
        */
    }
}
```

응용문제(list)

```cpp
#include <iostream>
#include <list>

using namespace std;

int main(void) {
    int N = 7, K = 3;
    list<int> mylist;

    for (int i = 1; i <= N; i++)
        mylist.push_back(i);
    list<int>::iterator it = mylist.begin();

    while (!mylist.empty()){
        for (int i = 1; i < K; i++){
            ++it;
            if(it == mylist.end()){
                it=mylist.begin();
            }
        }
        cout << *it << " ";
        it = mylist.erase(it);
        if (it == mylist.end())
            it = mylist.begin();
    }
}
```