**STL기초2 -변경 알고리즘**

**변경 알고리즘 :**

* 변경 알고리즘은 읽기 알고리즘과는 달리 컨테이너의 요소를 바꿀 수 있는 알고리즘이다.
* 요소의 값만 변경할 수 있을 뿐이지 컨테이너 자체에 대해서는 어떠한 조작도 하지 못한다는 점을 주의해야한다.
* 요소를 제거한다거나 새로운 요소를 삽입한다거나 컨테이너의 크기를 변경하는 것도 불가능하다.

🡺 왜냐하면 STL의 알고리즘들은 특수한 컨테이너에 소속되어 있는 것이 아니라 임의의 컨테이너와 함께 사용할 수 있도록 일반화되어 있기 때문이다.

\*copy : 지정한 구간을 복사하는데 주로 일부 요소들을 다른 컨테이너로 복사하고 싶을 때 사용한다.

OutIt copy(InIt first, InIt last, OutIt result);

BiIt copy\_backward(BiIt first, BiIt last, BiIt result);

* copy 함수는 first~last 사이의 모든 요소를 result반복자 위치 이후에 복사한다.
* 복사 목적지의 시작 위치는 result 반복자 하나로만 지정되며 길이는 원본과 같다고 가정하므로 result이후 last - first만큼의 기억 장소가 미리 확보되어 있어야 한다.
* 반복자는 같은 컨테이너의 다른 부분일 수도 있지만 다른 컨테이너의 반복자 구간끼리도 물론 복사할 수 있다

copy 예제

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

char src[]="1234567890";

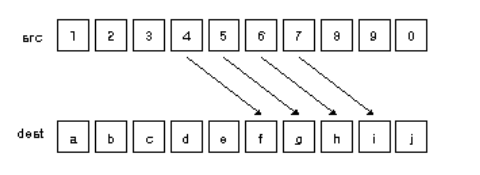
char dest[]="abcdefghij";

copy(&src[3],&src[7],&dest[5]);

puts(dest);

}

src의 3~7까지의 구간이 dest의 5 이후의 구간에 복사된다. 원본의 길이가 4이므로 목적지의 길이도 4이다.



* 실행 결과는 abcde4567j가 될 것이다. copy 함수는 이름이 의미하듯이 복사를 할 뿐이지 원본의 요소를 목적지에 삽입하는 것이 아니다. 따라서 result 반복자 이후부터 원본의 길이만큼 미리 확보되어 있지 않으면 결과는 예상할 수 없다.
* 위 예제의 복사 목적지를 dest[9]로 변경하면 배열 뒤쪽을 덮어쓰므로 아마도 다운될 것이다. 이럴 때 삽입 반복자를 사용하면 복사와 동시에 요소를 삽입할 수 있다

copy 예제2

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

using namespace std;

template<typename C> void dump(const char \*desc, C c) { cout.width(12);cout << left << desc << "==> ";

copy(c.begin(),c.end(),ostream\_iterator<typename C::value\_type>(cout," ")); cout << endl; }

int main()

{

char src[]="1234567890";

list<char> li;

copy(&src[3],&src[7],back\_inserter(li));

dump("list",li);

}

li 리스트는 빈 상태로 생성되었으므로 기억 장소를 전혀 확보하지 않고 있지만 back\_inserter 반복자가 대입 동작을 push\_back 삽입 동작으로 재정의하므로 복사되는 족족 리스트의 크기가 늘어난다

copy\_backward 예제

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

using namespace std;

template<typename C> void dump(const char \*desc, C c) { cout.width(12);cout << left << desc << "==> ";

copy(c.begin(),c.end(),ostream\_iterator<typename C::value\_type>(cout," ")); cout << endl; }

int main()

{

int i;

list<int> li,li2;

list<int>::iterator first,last,result,it;

for (i=0;i<10;i++) li.push\_back(i);

li2=li;

dump("복사전",li);

first=find(li.begin(),li.end(),2);

last=find(li.begin(),li.end(),7);

result=find(li.begin(),li.end(),3);

copy(first,last,result);

dump("copy",li);

first=find(li2.begin(),li2.end(),2);

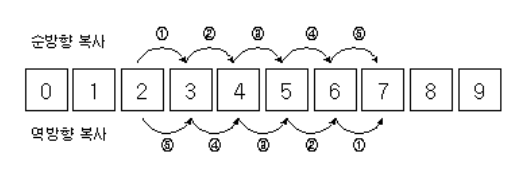
last=find(li2.begin(),li2.end(),7);

result=find(li2.begin(),li2.end(),8);

copy\_backward(first,last,result);

dump("back",li2);

}



* 순방향으로 복사할 때 first 위치의 2가 제일 먼저 3의 위치로 복사되는데 여기까지는 정상적이다. 그러나 다음 위치의 3을 4의 위치로 복사할 때는 이미 3이 앞쪽에 2의 의해 파괴되어 버렸으므로 3이 아닌 2가 복사되며 이후의 과정도 마찬가지이다. 원본이 복사되기 전에 먼저 변해 버리기 때문에 구간내의 모든 요소가 2로 변해 버리는 것이다
* 역방향으로 복사할 때는 원본이 먼저 복사되고 다른 값을 대입받으므로 이런 문제가 없다. 6을 7 자리에 먼저 대피해 놓고 5를 6자리에 복사하므로 안전하다. 역방향 복사 함수는 원형에 있어서도 다소 차이가 있는데 copy 함수는 순서대로 읽으므로 입력 반복자이기만 하면 되지만 copy\_backward함수는 반복자가 반대 방향으로 움직여야 하므로 양방향 반복자여야 한다
* result가 복사 목적지의 시작점이 아니라 끝다음점이어야 한다는 점도 다르다

copy\_backward 예제2

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

template<typename C> void dump(const char \*desc, C c) { cout.width(12);cout << left << desc << "==> ";

copy(c.begin(),c.end(),ostream\_iterator<typename C::value\_type>(cout," ")); cout << endl; }

int main()

{

int i;

vector<int> vi,vi2;

for (i=0;i<10;i++) vi.push\_back(i);

vi2=vi;

dump("복사전",vi);

copy(vi.begin()+2,vi.begin()+7,vi.begin()+3);

dump("copy",vi);

copy\_backward(vi2.begin()+2,vi2.begin()+7,vi2.begin()+8);

dump("back",vi2);

}

* 유독 벡터에 대해서만 복사 방향에 상관없이 잘 복사되는데 이는 구현 방식과 관련이 있다. 벡터는 요소들이 인접해 있는 구조를 가지고 있으므로 좀 더 빠른 복사를 위해 memmove 함수 또는 그에 준하는 메모리 복사 함수를 사용한다. 이 함수는 CPU의 메모리 이동 코드를 호출하는데 CPU가 복사 방향에 따라 순서를 적절하게 조정하기 때문에 겹치더라도 잘 동작하는 것이다.

\*\* 리스트의 경우는 이런 방향 자동 판단을 왜 할 수 없을까?

🡺 리스트의 노드는 메모리의 도처에 흩어져 있어 반복자만으로는 앞뒤 순서를 판단할 수 없으며 그래서 사용자가 수동으로 복사 방향을 선택해야 한다. 대부분의 컴파일러에서 벡터는 복사 방향에 상관없도록 구현되어 있지만 STL 스펙에 그렇게 구현해야 한다고 되어 있지는 않으므로 가급적이면 방향에 맞는 함수를 선택해서 사용하는 것이 바람직하다.

**void swap(T& x, T& y);**

* 스왑함수, 값 두개 스왑

**FwdIt2 swap\_ranges(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2);**

**🡺** first1~last1 사이를 first2구간과 바꾼다 , 다른 컨테이너의 반복자 구간끼리도 값을 교환 할 수 있되 동일 컨테이너 내의 교환인 경우 반복자 구간이 겹쳐서는 안된다.