**STL 기초 -1편**

-C++ 템플릿이란?

🡺 템플릿을 사용하여 프로그래밍 하는 것을 일반화(Generic) 프로그래밍이라고도 한다. 템플릿은 쉽게 얘기하면 함수나 클래스를 만들 수 있는 틀 이라고 생각하면 된다. 또한 특수화를 통해 템플릿을 따로 만들 수 있다.

\*template<> : 특수화

-Generic 과 Template의 차이점

🡺 C# 과 JAVA는 완전한 객체지향을 추구하는 언어이다. 그렇기 때문에 모든 타입은 Object 라는 최상위 클래스로부터 파생 되는데, 이 말은 모든 타입의 인스턴스는 Object의 인스턴스로 변환이 가능하다.

Generic 과 Template 는 다양한 타입에 대해서 컴파일 시간에 컴파일러가 자동적으로 코드를 생성 해 준다는 ㄱ념자체가 동일하지만 , 언어상 설계의 차이점이 Generic 과 Template 는 다르다

즉 Generic , Template 는 많은 차이점이 없다 (거의 없다고 해도 무방함)

\*find = 읽기 알고리즘은 컨테이너를 변경하지 않으며 컨테이너로부터 원하는 정보를 구하기만 하는 알고리즘이다. Find는 가장 기본적인 알고리즘인 검색기능을 제공

-예제

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

string names[]={"김정수","구홍녀","문병대",

"김영주","임재임","박미영","박윤자"};

vector<string> as(&names[0],&names[7]);

vector<string>::iterator it;

it=find(as.begin(),as.end(),"안순자");

if (it==as.end()) {

cout<< "없다" << endl;

} else {

cout<< "있다" << endl;

}

}

find는 구간을 순차 검색하며 요소를 비교할 때는 요소 타입의 == 연산자로 비교한다. 따라서 검색을 위한 특별한 조건은 필요없지만 컨테이너에 요소가 많으면 검색 속도는 느려진다

**\*\*find\_if = 완전히 일치하는 요소 뿐만 아니라 원하는 조건을 만족하는 요소도 검색 가능**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

bool HasYoung(string who)

{

return (who.find("영") != string::npos);

}

int main()

{

string names[]={"김정수","구홍녀","문병대",

"김영주","임재임","박미영","박윤자"};

vector<string> as(&names[0],&names[7]);

vector<string>::iterator it;

for (it=as.begin();;it++) {

it=find\_if(it,as.end(),HasYoung);

if (it==as.end()) break;

cout<< \*it << "이(가) 있다." << endl;

}

}

다음 함수를 변환 가능

bool HasYoung(string who)

{

return (who.find("영") != string::npos);

}

// 같은 역할을 하는 함수

/\*

struct HasYoung {

bool operator() (string who) const {

return (who.find(“영”) != string::npos);

}

};

It = find\_if(it , as.end(),HasYoung());

\*/

\*adjacent\_find 예제 : 구간을 순회하면서 앞뒤의 요소값이 같은 위치를 찾아 앞쪽 반복자의 위치를 리턴하며, 인접한 두 요소가 같은 것이 없으면 last가 리턴된다.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void main()

{

int ari[]={1,9,3,6,7,5,5,8,1,4};

vector<int> vi(&ari[0],&ari[9]);

vector<int>::iterator it;

it=adjacent\_find(vi.begin(),vi.end());

if (it != vi.end()) {

printf("두 요소가 인접한 값은 %d입니다.\n",\*it);

}

}

#include <iostream>

#include <algorithm>

using namespace std;

bool doublespace(char a, char b)

{

return (a==' ' && b== ' ');

}

void main()

{

const char \*str="기다림은 만남을 목적으로 하지 않아도 좋다.";

const char \*p,\*pend=&str[strlen(str)];

for (p=str;;p++) {

p=adjacent\_find(p,pend,doublespace);

if (p==pend) break;

cout<< p-str << "위치에 이중 공백이 있습니다." << endl;

}

}

\*find\_first\_of 예제 : 여러 개의 조건 중 하나라도 만족하는 요소를 검색하고 싶을 때 이 함수를 사용한다

🡺 임의 타입의 컨테이너에 대해 사용할 수 있으므로 훨씬 더 활용 범위가 넓으며 , strpbrk 와 유사하다.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void main()

{

int ar1[]={3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3,8,4,6,2,6,4,3};

int ar2[]={2,4,6,8,0};

int \*p=find\_first\_of(&ar1[0],&ar1[25],&ar2[0],&ar2[4]);

if (p!=&ar1[25]) {

printf("최초의 짝수는 %d번째의 %d입니다.\n",p-ar1,\*p);

}

}

두 개의 정수 배열을 선언하고 ar1에서 ar2에 있는 값 중 하나가 발견되는 최초의 지점을 찾는다. 예제에서 ar2에 모든 짝수를 나열해 두었으므로 ar1의 최초로 짝수인 수가 검색된다.

출력 결과 : 최초의 짝수는 2번째의 4입니다.