**STL 강의 정리(기말)**

1. **Iterator**

* 반복자처럼 행동하는 것은 반복자이다
* 반복자는 포인터를 추상화한 것이다.

ostream\_iterator<char> out{cout} : 화면의 맨 왼쪽 위치를 가져온다

out++, \*out : 출력 반복자는 ‘++’과 ‘\*’ 에서 아무것도 안 한다, 할 필요가 없다

//[문제] String 클래스(char 를 저장하는 컨테이너)가 sort 가능하도록 하려면?

// sort는 랜덤 반복자만 정렬해준다

// --> String\_iterator를 random\_access\_iterator 로 만들어주면 된다

// --> sort가 요구하는 기능만 코딩하여 해결

String s;

auto p = s.begin();

typeid(p).name()

//class String\_iterator 출력

typeid(iterator\_traits<String\_iterator>::iterator\_category).name()

//struct std::random\_access\_iterator\_tag 출력

* 밑은 클래스 반복자 코딩 에시

**class** **String\_iterator** : **public** std::iterator<std::random\_access\_iterator\_tag,char>

*// 2개만 적어도됨 나머지 3개는 디폴트 1.반복자 유형 2.자료형*

*// -> 이제 begin end가 캐릭터형이라는 것을 컴퓨터가 이해할 수 있음*

{

char\* p{**nullptr**};

**public**:

**using** iterator\_concept = std::contiguous\_iterator\_tag;

String\_iterator(char\* p) : p{ p } {}

bool **operator**!=(**const** String\_iterator& rhs) **const** {

**return** p != rhs.p;

}

*//21 4 29*

bool **operator**==(**const** String\_iterator& rhs) **const** {

**return** p == rhs.p;

}

*//21 4 29*

bool **operator**<(**const** String\_iterator& rhs) **const** {

**return** p < rhs.p;

}

String\_iterator& **operator**++() {

++p;

**return** \***this**;

}

*//21 4 29*

String\_iterator& **operator**--() {

--p;

**return** \***this**;

}

*//2021 4 29 &리턴*

char& **operator**\*() {

**return** \*p;

}

*//2021 4 29 읽기 전용 \**

char& **operator**\*() **const** {

**return** \*p;

}

*//21 4 29*

int **operator**-(**const** String\_iterator& rhs) **const** {

**return** p - rhs.p;

}

*//21 4 29*

String\_iterator **operator**+(int n) **const** {

**return** String\_iterator(p + n);

}

String\_iterator **operator**-(int n) **const** {

**return** String\_iterator(p - n);

}

};

*//2021. 5. 4*

**template**<>

**struct** **std**::iterator\_traits<String\_iterator>

{

**using** iterator\_category = std::random\_access\_iterator\_tag;

**using** value\_type = char;

**using** point = char\*;

**using** reference = char&;

**using** ptrdiff\_t = std::ptrdiff\_t;

};

*//2021.4.27 역방향 반복자*

**class** **String\_reverse\_iterator** {

char\* p{ **nullptr** };

**public**:

String\_reverse\_iterator(char\* p) : p{ p } {}

bool **operator**!=(**const** String\_reverse\_iterator& rhs) **const** {

**return** p != rhs.p;

}

String\_reverse\_iterator& **operator**++() {

--p;

**return** \***this**;

}

*//4.27 수정 char뒤에 & 붙임(읽고 쓰기 가능), return \*(p - 1) (왼쪽거를 리턴해)*

char& **operator**\*() **const** {

**return** \*(p-1);

}

};

**class** **String** {

**using** iterator = String\_iterator;

**using** reverse\_iterator = String\_reverse\_iterator;

iterator begin() {

**return** iterator{ p };

}

*//2021.4 .20 Add*

iterator end() {

**return** iterator{ p + num };

}

*//2021.5 .11 Add*

iterator cbegin() **const** {

**return** iterator{ p };

}

*//2021.5 .11 Add*

iterator cend() **const** {

**return** iterator{ p + num };

}

*//2021.4.20 Add*

reverse\_iterator rbegin(){

**return** reverse\_iterator{ p+num };

}

reverse\_iterator rend() {

**return** reverse\_iterator{ p };

}

bool **operator**==(**const** String& rhs) **const** {

*//2021 05 20 unordered\_set find 판정 == 사용*

++cnt;*//setCnt(getCnt() + 1); 안하려면 여기서해야함*

**if** (num != rhs.num)

**return** false;

**for** (int i = 0; i < num; ++i)

**if** (p[i] != rhs.p[i])

**return** false;

**return** true;

}

*//2021 05 11 길이순 정렬*

*//2021 05 18 찾기 실력 비교 위해 기본 정렬기준 변경 - 길이 오름차순*

bool **operator**<(**const** String& rhs) **const** {

**return** num < rhs.num;

}

}

int String::cnt;

1. **Generic Function**

제네릭 함수는 어떤 타입이든 가능해야함 (템플릿)

template<class Iter, class \_Ty>

Iter my\_find(Iter b,const Iter e,const \_Ty& c)

{

while (b != e) {

if (\*b == c) return b;

++b;

}

return e;

}

template<class Iter, class \_Pr>

Iter my\_find\_if(Iter b, const Iter e, \_Pr pr)

{

for (; b != e; ++b)

if (pr(\*b)) return b;

return e;

}

template<class Iter, class Pred>

int my\_count\_if(Iter beg, Iter end, Pred pr) {

int cnt{};

while(beg != end)

if (pr(\*beg)) {

cnt++;

beg++;

}

return cnt;

}

template <class X>

void f(X x)

{

if (contiguous\_iterator<X> )

cout << "연속 반복자" << endl;

else

cout << "연속된 메모리가 아닙니다" << endl;

}

template<class Iter>

ptrdiff\_t my\_distance(Iter beg, Iter end) {

int cnt{};

while (beg++ != end) {

++cnt;

}

return cnt;

}

copy(v.begin(), v.end(), ostream\_iterator<int>**{cout, " - "}); //구분자 추가**

template<class Iter1, class Iter2>

Iter2 my\_copy(Iter1 beg, Iter1 end, Iter2 dst)

{

while (beg != end)

{

\*dst++ = \*beg++;

}

return dst;

}

my\_copy(v.begin(), v.end(), back\_inserter(v2)); // back\_inserter를 써야 size가 늘어나서 범위기반 for문에서 출력이된다.

1. **Associative Container**

ifstream in("소스.cpp");

set<string> s; //중복 허용안됨 중복허용은 multiset사용하기

copy(istream\_iterator<string>{in}, {}, inserter(s, s.begin()));

multiset<char> s; //multiset은 buf를 빼고 buf는 공백까지 읽는것

template<class \_Ty>

class 사전오름차순 { //function object , callable type

public:

bool operator()(const \_Ty& a, const \_Ty& b) const { // const 함수로 지정해야함

return a < b;

}

};

//[문제] 소스.cpp 파일에 있는 단어와 그 단어의 출연횟수를 출력하시오

map<string, int> simap;

string str;

while (in >> str)

simap[str]++;

// 맵을 출력

for (auto& [단어,출현횟수] : simap)

cout << 단어 << "->" << 출현횟수 << endl;

//[문제] 소스.cpp 파일에 있는 문자(공백은 뺀다)를

// 출현횟수 기준 내림차순으로 출력하시오

map<char, int> simap;

while (in >> skipws >> str) { // 문자별로 횟수가 카운팅됨

if(isalpha(str))

simap[str]++;

}

map<int, char, greater<int>> ic;

// 맵을 바꿈

for (auto& [문자, 횟수] : simap) // i,40

ic[횟수] = 문자;

1. **Unordered Associative Container**

찾기 복잡도 O(1)

String이 Unordered - 컨테이너의 원소가 되려면?

template <>

struct hash<String> {

size\_t operator()(const String& s) const {

//s의 어떤값을 hash할거냐

return hash<int>()(s.size());

}

};

unordered\_set<String,hash<String> > us{ 10,20,30 };

//이거는 추가 거부

us.insert(20);

//기존 20 글자와 추가한것 서로 다른객체가 분명하다

//셋은 20글자와 추가한것을 같은 객체라고 판단하고 추가를 거부한다

// 원소가 같은것이라는 판단을 operator==으로 하지 않는다.

//셋은 operator <로 두 객체가 같은지 판단한다.

// if (!(x<y)&& !(y<x))

// x와 y는 같은 원소이다

// if !(20<20) && !(20< 20 ) )

// 셋의 같다는 판단을 동등성(equivalence) ---- < 로 같음을 판단

// 진짜 같음을 판단할 때 상등성(equality) ---- == 으로 같음을 판단

v.reserve(NUM);

for(int i =0; i < NUM; ++i)

v.emplace\_back(i+1);

// 찾을 String 객체를 array에 미리 넣어 놓는다

array<String, FNUM> toFind;

for (int i = 0; i < FNUM; ++i)

toFind[i] = v[sam(dre)];

toFind[0].setCnt(0);

{

//vector 찾기 성능 operator == 상등

for (int i = 0; i < FNUM; ++i)

if (find(v.begin(), v.end(), toFind[i]) != v.end());

cout << "예상 호출 횟수 - "<< FNUM \* NUM /2 << endl;

cout << "진짜 호출 횟수 - "<< toFind[0].getCnt() << endl;

//찾기 성능은 시간을 재면 안되고 호출 횟수로 판별해야한다

1. **Algorithm Function**

//[문제] v를 홀짝으로 분리하라.

auto p = partition(v.begin(), v.end(), [](const int& a) { return a & 1;});

// [문제] 소스 .cpp의 영문자를 다대문자로 변신시키자

ifstream is{"소스.cpp"};

std::transform(istreambuf\_iterator<char>{is},

istreambuf\_iterator<char>{},

ostreambuf\_iterator<char>{cout},

[](unsigned char c) -> unsigned char { return std::toupper(c); });

// [문제] sort 관련 알고리즘은 잘알고 쓰자

// sort와 stable\_sort

// sort - 1등부터 마지막까지 전부 정렬

// partial\_sort - 1등부터 10등까지

// nth\_element - 앞에서 10개

중복된 원소가 있을 경우 stable\_sort는 순서를 보장해준다

Ex) <int,char> , int순으로 정렬 1a 3a 2a 2b -> 1a 2a 2b 3a (stable\_sort)

//sort는 이게 보장이 안된다

// 자료구조의 데이터를 전달하게 섞는다.

shuffle(v.begin(), v.end(), mt19937(random\_device()())); //psedo – random

//partial sort

partial\_sort(v.begin(), v.begin() + 30, v.end());

//nth element

nth\_element(v.begin(), v.begin() + 30, v.end());

// [문제] 정렬되어 있는 자료를

// binary\_search

// upper\_bound

// lower\_bound

// equal\_bound

if (**binary\_search**(v.begin(), v.end(), c) == false) {

cout << c << " 는 없" << endl;

continue;

}

auto [low, upper] = **equal\_range**(v.begin(), v.end(), c);

cout << c << "는 " << upper - low << "개 있음" << endl;

//정렬되어있다면 조건에 해당하는 범위의 시작과 끝을 리턴함

// [문제] s의 가능한 조합을 모두 출력하시오 5!

while (**next\_permutation**(s.begin(), s.end())) {

cout << s << endl;

++cnt;

}

//모든 단어는 유일해?

auto p = unique(v.begin(), v.end());

if (p == v.end()) cout << " 모든 단어는 유일하다" << endl;

//정렬된 사전이 맞어?

cout <<"정렬된 사전이 맞어? " << boolalpha << is\_sorted(v.begin(), v.end())<< endl;

// 단어들 다운받고 컨테이너에 저장

//모두 몇단어인지? -> v.size()출력

//[문제] 사용자가 입력한 단어가 사전에 있는지 찾는다.

// 없으면 없는 단어입니다 출력

// 사전에 있다면 이 단어와 anagram 관계인 단어를 찾아 모두 출력한다

//[문제] 이 단어들에는 몇개의 anagram 쌍이 있을까?

// 애너그램 갯수기준 내림차순으로 정렬하여 파일 "애너그램개수순.txt"에 저장하라

vector<list<string>> sorted\_v;

sorted\_v.reserve(33000); //개수를 대략 알고있으니까 예약한다

//애너그램 발견

auto i = adjacent\_find(b, v.end(), [](const auto& a, const auto& b) {

return a.first == b.first;

});

if (i == v.end())

break;

//[i,j)는 anagram 구간

auto j = find\_if\_not(i+1, v.end(), [i](const auto& a) {

return a.first == i->first;

});

list<string> words;

for (auto x = i; x < j; ++x)

words.push\_back(x->second);

sorted\_v.push\_back(words);

b = j; //이후 와일문 반복

// 소팅해준후 출력

for (list<string>& o : sorted\_v) {

o.sort();

out << "[" << ++cnt << "] - ";

for (const string& o2 : o)

out << o2 << " ";

out << endl;

}

1. **range, format**

vector<int> v{ 1,3,5,7,9,2,4,6,8,10 };

ranges::sort(v);

for (int n : v

| views::reverse

//쓰기가 더편함(rbegin), 원본이 바뀌지 않음(algorithm->reverse())

| views::transform([](int& n) {return n \*= 10; })

//&붙이면 원본이 바뀜

)

cout << n << " ";

cout << endl;

auto r = v | views::filter([](int n) { return n & 1; }); //홀수만 뽑아오기

//#include <format>

//cout << format("{+06d}",120) << endl : +000120

format 함수는 string\_view 형태로 입력받고 string으로 리턴한다

첫번째 인자는 서석 문자열 두번째 인자는 포매팅할 값을 적는다

1. 서식문자열
   1. {} , value //value가 {}그대로 들어간다
   2. {0} {0} {1} , val,val2 //{}안의 숫자 순서대로 value가 들어감
   3. {:06},120 // “:” 문자 특수 표기 설정 이 경우 000120 이출력됨
   4. : 뒤에 올수있는 문법
      1. 첫번째 : 빈공간을 채울 문자
      2. 두번째 : 정렬방식
      3. 세번째 : 부호 표시방법
      4. 네번째 : 이진수나 16진수 표기시 접두사를 붙인다 // #붙이면 표기
      5. 5번째 : 6번째(width)앞에 0이있으면 첫번째,두번째가 0=인것처럼 작동
      6. 6번째 : 최소너비
      7. 7번째 : 수의 정밀도, 소수점 몇번째까지 표기할건지(정수도 포함,유효숫자) //.5, .3
      8. 8번째 : 타입, 해당 인수의 자료형과 진법을 표기 //자료형표기는 안해도됨 읽기 편하고 실수 줄이려고하는것