**STL 강의 정리**

#include <filesystem> //파일 크기를 알아낼수있다 (file\_size())

cout << numeric\_limits<int>::max() << endl; //인트의 최댓값을 알아보자 21억

iostream헤더에 limits.h가 포함됨

sort사용시 람다함수를 이용하면 가독성이 증가한다(sort는 logN번 반복함

가독성만 증가 하는게 아니라 정렬하는 속도 또한 증가함 (컴파일러 최적화) - 함수의 인라인화)

레퍼런스는 포인터를 없애려고 만들었다? :원격으로 값을 기록하기위해 레퍼런스를 사용한다

임의의 값을 갖는 정수 1천만개를 만들어라

지역변수로 초기화시 안된다 - 기본 1MB로 제한되어 있음 프로젝트 속성 - 링커 - 시스템 - 스택 예약크기를 늘려주면 가능은 함

전역 데이터 (데이터영역) - 2GB까지 잡을 수 있음 (실제는 4GB이나 유저는 2GB까지 - x86기준)

ofstream out("int 10개.txt"sv);//s 는 스트링객체라고 명시해주는 것 sv스트링 뷰[[1]](#footnote-1) 객체라고 명시

//[문제] int 10000개를 1부터 10000까지 값으로 채워라

//파일에 "int 만개.txt"로 기록하라

//(질문) 하드디스크의 파일 크기를 확인하고 결과를 설명하라

// --> 40kb보다 더 큼 파일에 기록시 인간이 볼 수 있도록 숫자를 문자로 바꾸기 때문에 크기가 더 커지게 된다.

//엔터키를 강제 형변환하면서 2바이트로 바뀌었기때문에 40000 +295크기를 갖게됨 , ios::binary 를 사용하면 위에 한 짓을 못하게 막을 수 있음 ofstream out("int 만개.txt",ios::binary);

int a[10'000]; // 초기화 하냐 안하냐에 큰 차이가 있다 안하면 컴파일러가 똑똑해서 아무것도 안함

iota(begin(a), end(a), 1); //시작 부터 끝까지 1개씩 증가시키면서 저장함(iota :numeric 헤더에 포함)

read write 사용 예시 in.read((char\*)a, 10'000 \* sizeof(int)); / out.write((char \*)a, sizeof(int) \* 10'000);

int\* a = new int[num]; // [] 크기만큼 연속된 공간을 요청 (contiguous) 중간에 이미 쓰고 있는 공간이 있으면 안됨

try catch 사용

//메모리를 줄 수 없으면 예외를 던진다. 그러면 최소 터지는것을 막을 수 있다.

try{

a = new int[num];

}

catch (exception& e) {

cout << e.what() << endl;

}

// int\* a = new int[num] - > raw포인터, 바보 포인터

unique\_ptr<int[]> p (new int[num]);

// 날짜 요일 - 시스템 클럭, 시간간격 - 스테디 클럭, 하이레졸 클럭 - ?

chrono::steady\_clock::time\_point begin = chrono::steady\_clock::now();

auto begin = chrono::steady\_clock::now();

자료형이 뭔지 알겠다면 이제 auto로 짧고 편하게 쓰자

//기본단위는 ns 단위임 아래는 밀리초로 경과시간을 출력하는 방법

cout << "경과시간(밀리초) - "<< chrono::duration\_cast<chrono::milliseconds>(d).count() << endl;

// [문제] Dog 만 객체를 생성하라. name은 “Dog + 1” 부터 시작되는 숫자로, {Dog1,Dog2,Dog3,......Dog10000) id값은 [1,10만] 살 랜덤값으로 파일에 메모리 크기 그대로 기록한다

(확인) - 크기 == sizeof(Dog)\*10'000 파일에서 읽어 이름오름차순으로 정렬한후 제일 처음과 마지막을 출력한다.

to\_string(cnt); // int값을 string을 바꾸는 함수 반대로 스트링을 인티저로 바꿀때 stoi

(static) - 전역 변수가되서 클래스 내부 메모리로 포함되지 않음, 사용하는 공간만 클래스 내부에 있음

void x();

x(); // 함수 호출 연산자 - > "()" 클래스내에서 오버로딩 가능

void operator()(void) {

cout << "class () operator overloading"<< endl;

}

void (\*f)(); // 스택에 생성 4바이트 크기

f = jump; //점프 함수의 번지수를 가르킴

f = slide; //사용 예시 게임에서 키설정을 바꿀때

typeid(“??”).name() typeid()는 자료형 클래스 객체 함수 거의 받을 수 있다. 해당하는 것의 정보를 가져오고 .name()함수사용시 해당 자료형이 무엇인지 출력해준다

//람다함수는 클래스로 만들어지고 컴파일러가 클래스 오퍼레이터를 만든다

function[[2]](#footnote-2)<void(int)> ct; //이 형태는 "function<void(int)>"모든 callabletype을 담을수 있다.

vector<string> v{ istream\_iterator<string>{cin},{} }; //키보드 입력의 끝까지 입력받는다 - 입력의 끝(Ctrl Z)

String(const String& other) : 복사생성자, memcpy

String& operator=(const String& other) :복사 할당 연산자 if (this != &other) {this->~String(); 자자기자신 아닐 때 소멸자 호출후 복사 생성과 같은 작업을 해줌

String(String&& other) : 이동 생성자, 자원을 가로챔 그대로 대입 p = other.p

String& operator=(String&& other) : 이동 할당 연산자 복사할당 처럼 자신이아닌지 확인후 대입한다 소멸자 호출은 없음 other는 nullptr과 0으로 멤버들을 초기화한후 마지막에 return \*this

move(s); 이동연산 사용법

//[Question] []이 있는데 array<T,N>은 왜 만들었나?

//뭘 사용하면 되나? “.” 입력 시 사용할 수 있는게 많다.

벡터는 프로그램이 실행될 때 크기가 변하는 동적 배열이다. Size와 capicity가 같을 때 만약 값이 추가된다면 재할당이 일어난다 벡터는 순서대로 소멸된다

벡터의 push\_back

v.emplace\_back(40); // 복사생성하지마 그자리에 생성해,

생성자가 없는거에 emplace back하는건 웃기는 짓이다 그럴땐 푸쉬백 써라

**+ 보충)** emplace\_back 쓰는 이유 임시 객체 안만들고 바로 컨테이너에 넣어 (생성자의 인자만 받아서 돌아가게 하는경우 쓰는거다 -> 디폴트생성자 코딩해줘야함)

vector 중간에 원소 추가 - 좋지않음 지우는것도 좋지않음

v.insert(v.begin() + 2, 3); // insert 첫번째함수 특정위치에 리스트를 넣는다, 세번째 특정위치에 카운트만큼 값을넣는다 네번째 특정위치에 값을넣는다

v.erase(remove\_if(v.begin(), v.end(), [](int value) {return !(value & 1); }), v.end());

erase\_if도 있음

//조건에따라 값을 지우기 remove\_if는 마지막위치를 반환한다

v.shrink\_to\_fit(); // capacity 조절함

//메모리가 연속된 자료구조를 사용하는 이유

// -- > O(1) access (시간복잡도)

deque 벡터보다 접근시간이 느리다, 그러나 더많은 값을 저장할 수있다

//무한하게 푸시백 하면 벡터는 얼마까지 저장할 수 있나? (.max\_size()이용)

// Vector Max – 136216567 // 단편화했을때 이런거지 한번에 크게잡으면 더 큰 크기를 잡을 수 있음

// Deque Max - 268435452

//이건 엄청난 사기

//왜 이 컨테이너를 만들지?

//이 컨테이너의 장점이 뭐지?

//span : meomory is contigous

template <typename T>

void print(span<T> s) {

for (int i = 0; i < s.size(); ++i)

cout << s[i] << " - ";

cout << endl;

}

// 메모리가 연속되어 있는 경우에 항상 실행될 수 있는 코드를 작성

* [], array, vector, string 잘됨 deque은 안된다? 역시 사기꾼인가?

//()생성자 {}초기값 설정

List – reverse()함수 담겨있는 값을 역순으로 바꾼다 근데 굳이 이렇게 할 필요가 있나?

for (auto i = cont.crbegin(); i != cont.crend(); ++i) 이렇게 읽으면 되지;;

리스트 초기화 동시에 파일로 읽기 list<char> cont(istreambuf\_iterator<char>{in}, {}); //공백도 읽으려면 buf 추가

List - count 함수

//[Problme] ".cpp" 'e' 가몇개있나?

cout <<count(cont.begin(), cont.end(), 'e') << endl;

//[Problme] ".cpp" 알파벳 소문자 가몇개있나?

cout << count\_if(cont.begin(), cont.end(),

[](const char& a) { return islower(a); //lower 판별함수 }) << endl;

//[Problme] ".cpp" #include 가몇개있나?

cout << count\_if(cont.begin(), cont.end(),

[](const string& a) { return a == "#include"s; }) << endl;

List - find함수

//[Problme] ".cpp" using이라는 string이 존재하는지? 있다면 몇 번째 string인지 출력하라

auto position = find\_if(cont.begin(), cont.end(),

[](const string& s) {return s == "using"s;});

cout << "위치는 " << distance(cont.begin(), position) << endl;

cout << "위치는 " << distance(cont.begin(), position) + 1 <<"번째 단어"<< endl;

distance()로 몇번째 단어인지 출력할 수 있다.

List - merge함수

//[Problem] con1 + con2 로 새로운 리스트 생성,use merge

con3.merge(con);

merge함수는 합칠 때 오름차순으로 합치려고 노력한다

List - sort함수를 갖고 있다

con.sort([](const int& a, const int& b) { return a > b; });//내림차순도가능

List - unique함수 (중복된거 지우기)

list<int> con{ 5,1,3,9,7,7,7,7,7,7,7,7,7,7,7,7,7};

con.unique();

List - splice함수

특정위치에 값삽입

list<int> con1{ 1,3,5 };

list<int> con2{ 2,4,6 };

con1.splice(++con1.begin(), con2, con2.begin());

1 2 3 5 출력됨

// Iterator

// 디자인패턴중 이터레이터 구현한것

// 반복자처럼 행동하는 것은 반복자

// 반복자는 포인터를 추상화한것

cout << typeid(iterator\_traits<vector<int>::iterator>::iterator\_category).name() << endl; //반복자 타입을 출력하는 방법

//[Problem] 전달된 박복자의 종류를 출력하는 함수를 작성하라

template<typename Iter> // 템플릿은 정의와선언을 분리하지않는다

void show(const Iter&)

{

cout << typeid(iterator\_traits<Iter>::iterator\_category).name() << endl;

}

istream\_iterator<char> ii{ cin };

show(ostream\_iterator<char>{cout});

show(forward\_list<char>{}.begin());

show(list<char>::iterator());

show(deque<char>::iterator());

show(vector<char>::iterator());

cout << typeid(string::iterator::iterator\_concept).name() << endl;

cout << typeid(array<char,0>::iterator::iterator\_concept).name() << endl;

cout << typeid(vector<char>::iterator::iterator\_concept).name() << endl;

//[Problem] String begin() 과 end() 반복자를 코딩하기

class String\_iterator {

char\* p{nullptr};

public:

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

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

return p != rhs.p;

}

String\_iterator& operator++() {

++p;

return \*this;

}

char operator\*() const {

return \*p;

}

};

//[Problem] String Class 역방향 반복자인 rbegin과 rend를 제공하여

// 역방향 억세스가 가능하도록 하라

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

--p;

return \*this;

}

이름 바꾸고 여기 내부에서만 사기치면 끝

1. #include<string\_view> [↑](#footnote-ref-1)
2. #include <functional> [↑](#footnote-ref-2)