



땅울림 자구 스테디

4주차

1

Queue

2

Vector
List
Sequence

3

Vector

4

비주얼 tip

1. Queue

1 Queue

큐의 구현 방법

선형 큐 : <http://mailmail.tistory.com/33>

★원형 큐 : <http://mailmail.tistory.com/41>

1 Queue

1966_프린터 큐

1 Queue

문서들의 중요도 검사해서 순서에 맞게 출력

관찰)

1. 항상 가장 중요도가 큰 문서가 먼저 출력됨
2. 중요도가 같은 문서가 있을 수 있기 때문에, 큐가 순환할 때마다 바뀌는 M문서의 위치를 알고 있어야 함

1 Queue

풀이)

1. 큐의 front가 max중요도이면 pop, 아니면 큐를 순환
2. 큐의 front가 max인지 검사하기 위해 **정렬된** 배열 사용

1 Queue

예제) N : 5, M : 4, 중요도 : 1 2 9 1 1

큐의 front와 arr[cnt]이 같아질 때까지 큐를 순환

queue



arr



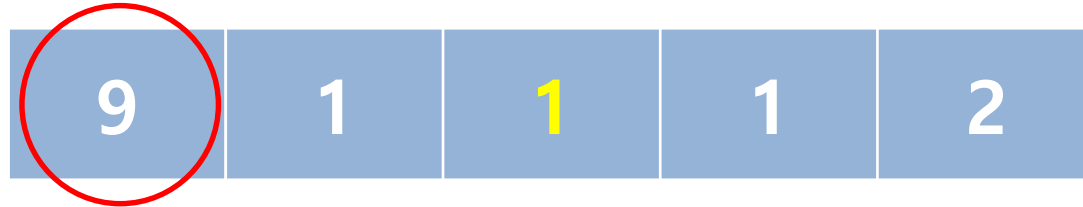
(내림차순으로 정렬)

↑
cnt = 1

1 Queue

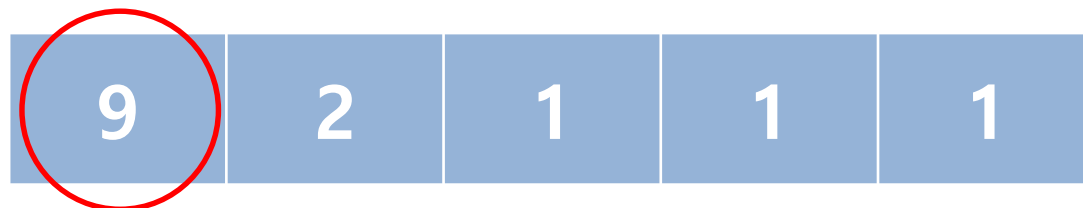
큐의 front와 arr[cnt]이 같으면 pop

queue



pop할때마다 cnt를 증가

arr



cnt = 1 cnt = 2

1 Queue

큐의 front와 arr[cnt] 같아질 때까지 순환

queue



arr



cnt = 2

1 Queue

큐의 front와 arr[cnt] 같아질 때까지 순환

queue



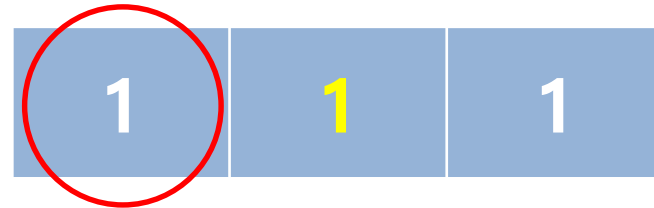
arr



cnt = 2

1 Queue

queue



arr

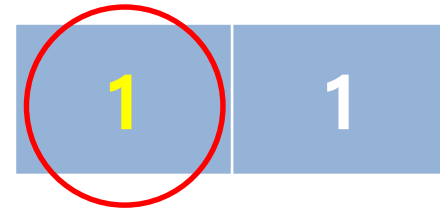


cnt = 3

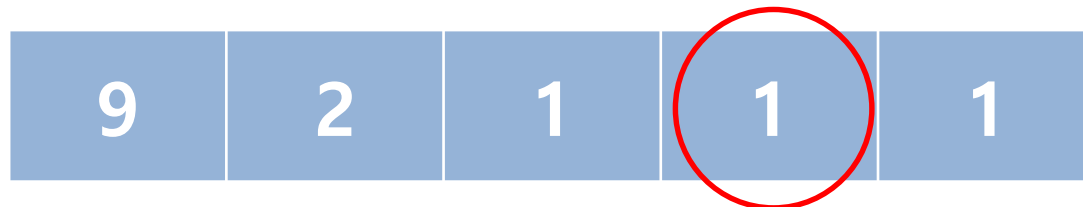
1 Queue

queue

정답 : 4(cnt)



arr



cnt = 4

1 Queue

위의 그림에서 노란색으로 표시한 숫자가 M
 M 이 q 의 `front`에 있으면서 가장 높은 중요도일 때 출력됨

하지만 큐를 순환시킬 때마다 M 의 위치가 변함

M 의 위치를 계속해서 변경시켜줘야함

1 Queue

한 번 순환 == 왼쪽으로 한칸씩 밀림
순환할 때마다 M을 1씩 감소시키다가
M이 -1이 되면 가장 오른쪽으로 옮겨지므로
 $M = q.size() - 1$

2. Vector, List, Sequence (교과서 ver.)

2 Vector

벡터(배열 기반)

일단은 배열하고 비슷하다.

배열과의 차이점 1) 중간에 삽입,삭제 제공

배열과의 차이점 2) 크기가 유동적(매우 중요)

2 Vector

벡터 함수

`at(int i)` : i 번째 인덱스의 값을 반환 ($v[i]$)

`set(int i, T data)` : i 번째 인덱스에 `data`를 대입 ($v[i] = data$)

`insert(int i, T data)` : i 번째 인덱스에 `data`를 삽입
(뒤에를 한 칸씩 밀고 끼워넣기)

`erase(int i)` : i 번째 인덱스를 지움(지우고 앞으로 한 칸씩 땡김)

`size()` : 벡터 크기

`empty()` : 벡터가 비었는지

2 Vector

함수들의 성능

배열과 거의 같음

at, set같은 건 $O(1)$

insert, erase는 $O(N)$

(환형으로 구현하면 $O(1)$, 하지만 그렇게 안함)

2 Vector

벡터의 가장 중요한 특징

크기가 유동적이다!

크기 N 인 배열 A 가 꽉 찼을 때 insert가 호출되면?

1. 크기 $2N$ 인 새로운 배열 B 를 생성
2. $0 \sim N-1$ 까지 $A[i]$ 를 $B[i]$ 에 복사
3. A 의 메모리를 해제하고 새로운 배열 B 를 사용

두 배씩 늘리는 게 가장 효율적이라고 알려져 있음.
(Array Doubling)

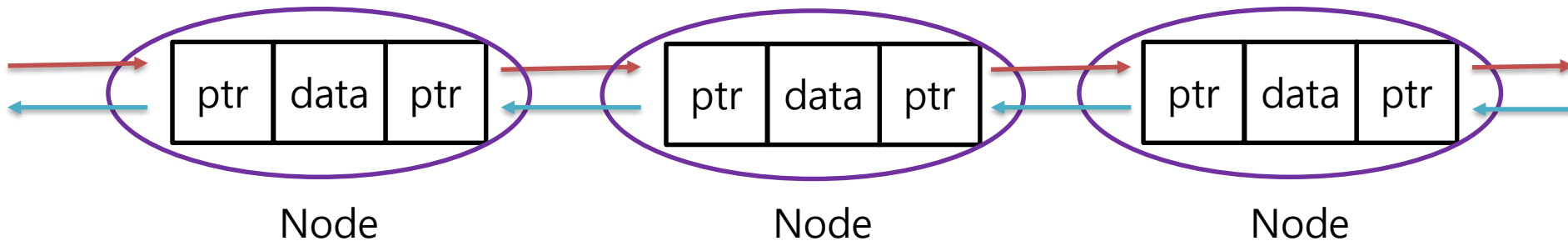
2^{List}

리스트(노드 기반)

이중 연결 리스트로 구현됨
양방향에서 탐색할 수 있음
중간에 삽입, 삭제가 자주 일어날 때 사용

2 List

이중 연결 리스트



장점 : 양방향 탐색 가능 (탐색, 삭제가 빠름)

**단점 : 코드가 좀 더 복잡해짐
메모리를 더 많이 사용함**

잘 쓰면 더 튼튼하지만, 잘못쓰면 더 취약해짐

2 List

리스트 함수

Begin(), end() : 시작 위치와 끝 위치

insert(p, e) : 위치 p에 e를 삽입

Erase(p) : 위치 p의 원소를 삭제

insertFront(e) : 맨 앞에 e 삽입

insertBack(e) : 맨 뒤에 e를 삽입

removeFront() : 맨 앞 data 삭제

removeBack() : 맨 뒤에 data 삭제

2 Sequence

시퀀스(벡터 + 리스트)

벡터와 리스트를 합쳐놓은 것
배열로 구현하거나 이중 연결 리스트로 구현

2 Sequence

Operation	Array	List
size, isEmpty	1	1
atRank, rankOf, elemAtRank	1	<i>n</i>
first, last, before, after	1	1
replaceElement, swapElements	1	1
replaceAtRank	1	<i>n</i>
insertAtRank, removeAtRank	<i>n</i>	<i>n</i>
insertFirst, insertLast	1	1
insertAfter, insertBefore	<i>n</i>	1
remove	<i>n</i>	1

3. Vector

(실전 Ver.)

3 Vector

벡터 쓰는 이유

배열처럼 사용 가능

자동으로 초기화됨

크기를 유동적으로 정해줄 수 있음

맨 끝에 갖다붙이거나 갖다버리기 편함

순회하기 편함

+ 편리한 함수들 많이 사용 가능

3 Vector

데이터 삽입 방법

1. `v[1] = 2; v[4] = 7;` (배열이랑 똑같이)
2. `v.push_back(3);` (맨 끝에 갖다붙임)
3. `insert` (잘 안씀, 익숙해지면 가~~끔 사용)

3 Vector

벡터 선언 방법(#include <vector>)

타입

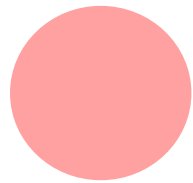
vector <int> v;

vector <int> v(N); ← 변수 사용 가능

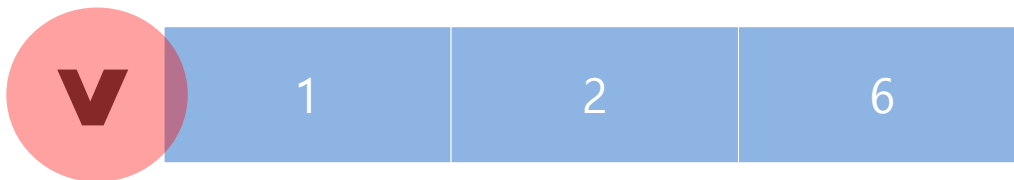
vector <int> v[5];

vector <vector <int>> v(N);

3 Vector

 : '핵'이라고 임시로 명명
(push_back 가능)

vector <int> v; 처음엔 아무것도 없고 이름만 가지고 있음
(메모리는 할당받음)



v.push_back(1);

v.push_back(2);

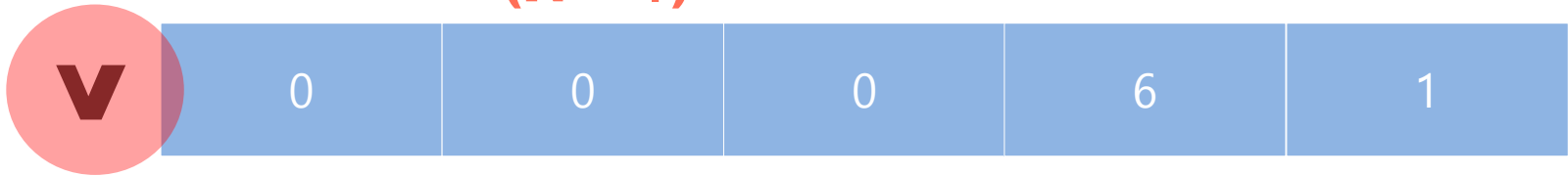
v.push_back(6);

v[4] = 7; Error!

3 Vector

변수도 사용 가능

vector <int> v(N); 처음부터 크기 N짜리로 만듦
(N = 4)



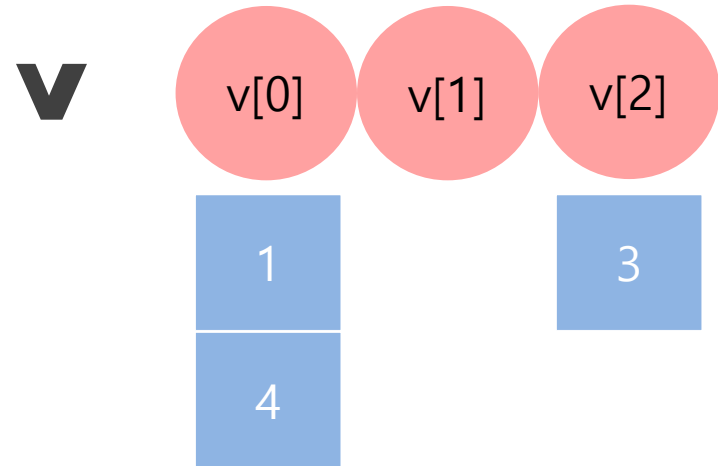
v.push_back(1);

v[3] = 6;

3 Vector

대괄호[]는 변수 사용 불가능

vector <int> v[3]; 크기 3짜리, 2차원 벡터 (행 3개)



v.push_back(1); Error!

v[2] = 6; Error!

v[0].push_back(1);

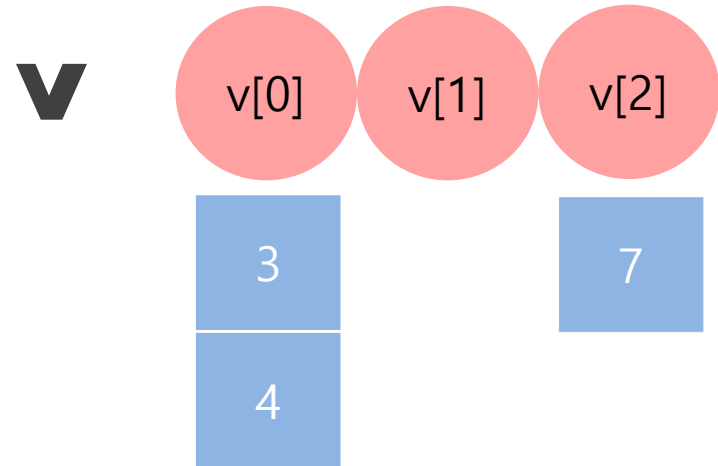
v[2].push_back(3);

v[0].push_back(4);

3 Vector

대괄호[]는 변수 사용 불가능

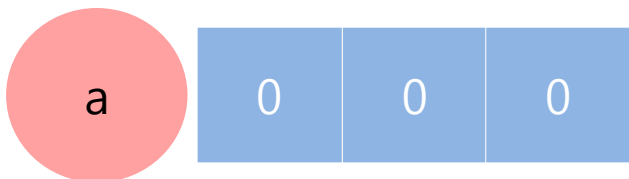
vector <int> v[3]; 크기 3짜리, 2차원 벡터



v[0][0] = 3;

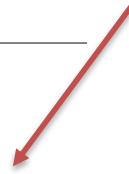
v[2][0] = 7;

vector <int> a(3);



3 Vector

대괄호[]는 변수 사용 불가능



```
vector <vector <int> > v(2);
```

크기 2짜리, 2차원 벡터(팩 2개)

v

v[0]

v[1]

3 Vector

자주하는 실수

```
int n;  
cin >> n;  
vector<int> v(n);  
int tmp;  
for (int i = 0; i < n; i++)  
{  
    cin >> tmp;  
    v.push_back(tmp);  
}
```

크기 선언해놓고 뒤에다 붙이기

```
int n;  
cin >> n;  
vector<int> v;  
for (int i = 0; i < n; i++)  
    cin >> v[i];
```

크기가 없는 상태에서 인덱스에 접근

3 Vector

자주 쓰는 함수

1. `v.push_back(e), v.pop_back()`
2. `v.size()` : `v`의 사이즈 리턴
3. `v.clear()` : 초기화(핵만 남음)
4. `sort(v.begin(), v.end())` : 정렬
5. `reverse(v.begin(), v.end())` : 뒤집
6. `v.resize(n)` : `v`의 크기를 다시 정해줌

3 Vector

auto(자료형을 자동으로 지정)

```
int arr[7] = { 0 };  
for (int i = 0; i < N; i++)  
    cout << arr[i];
```



```
vector<int> v(7);  
for (auto i : v)  
    cout << i;
```

```
for (map<int, string> ::iterator it = m.begin(); it != m.end(); it++)  
    cout << it->first << "\n";
```



```
for (auto i : m)  
    cout << i.first << "\n";
```

코드가 간결해짐

3 Vector

auto(10808 : 알파벳 개수)

4. Visual studio 설정 tip

4^{Tip}

**입력 데이터 txt파일로 입력받는 방법
(콘솔창에 일일이 안쳐도됨 ㄹㅇ 꿀팁)**

우클릭 => 속성

- 빌드(U)
- 다시 빌드(E)
- 정리(N)
- 보기(W)
- 분석(Z)
- 프로젝트만(J)
- SDK 버전 대상 다시 지정
- 여기까지 범위 지정(S)
- 새 솔루션 탐색기 뷰(N)
- 빌드 종속성(B)
- 추가(D)
- 클래스 마법사(Z)... Ctrl+Shift+X
- NuGet 패키지 관리...
- 시작 프로젝트로 설정(A)
- 디버그(G)
- 소스 제어(S)
- 잘라내기(T) Ctrl+X
- 붙여넣기(P) Ctrl+V
- 제거(V) Del
- 이름 바꾸기(M)
- 프로젝트 언로드(L)
- 솔루션 다시 검색(S)
- 파일 탐색기에서 폴더 열기(X)
- 속성(R) Alt+Enter

Visual Studio Solution Explorer showing a project named 'Algorithm'.

Right-click context menu items visible:

- 속성 (Properties)
- 삭제 (Delete)
- 이름 바꾸기 (Rename)
- 프로젝트 언로드 (Unload Project)
- 다시 빌드 (Rebuild)
- 정리 (Clean)
- 다시 빌드 (Rebuild)
- 빌드 (Build)

Background items visible in the Solution Explorer:

- Algorithm
- Algorithm
- Algorithm
- C:\Users\Wheo04\de

구성(C): 활성(Debug)

플랫폼(P): 활성(Win32)

구성 관리자(O)...

구성 속성

일반

디버깅

VC++ 디렉터리

▷ C/C++

▷ 링커

▷ 매니페스트 도구

▷ XML 문서 생성기

▷ 찾아보기 정보

▷ 빌드 이벤트

▷ 사용자 지정 빌드 단계

▷ 코드 분석

시작할 디버거:

로컬 Windows 디버거

명령	\$(TargetPath)
명령 인수	< input.txt < ***.txt 입력
작업 디렉터리	\$(ProjectDir)
연결	아니요
디버거 형식	자동
환경	
환경 병합	예
SQL 디버깅	아니요
Amp 기본 액셀러레이터	WARP 소프트웨어 액셀러레이터

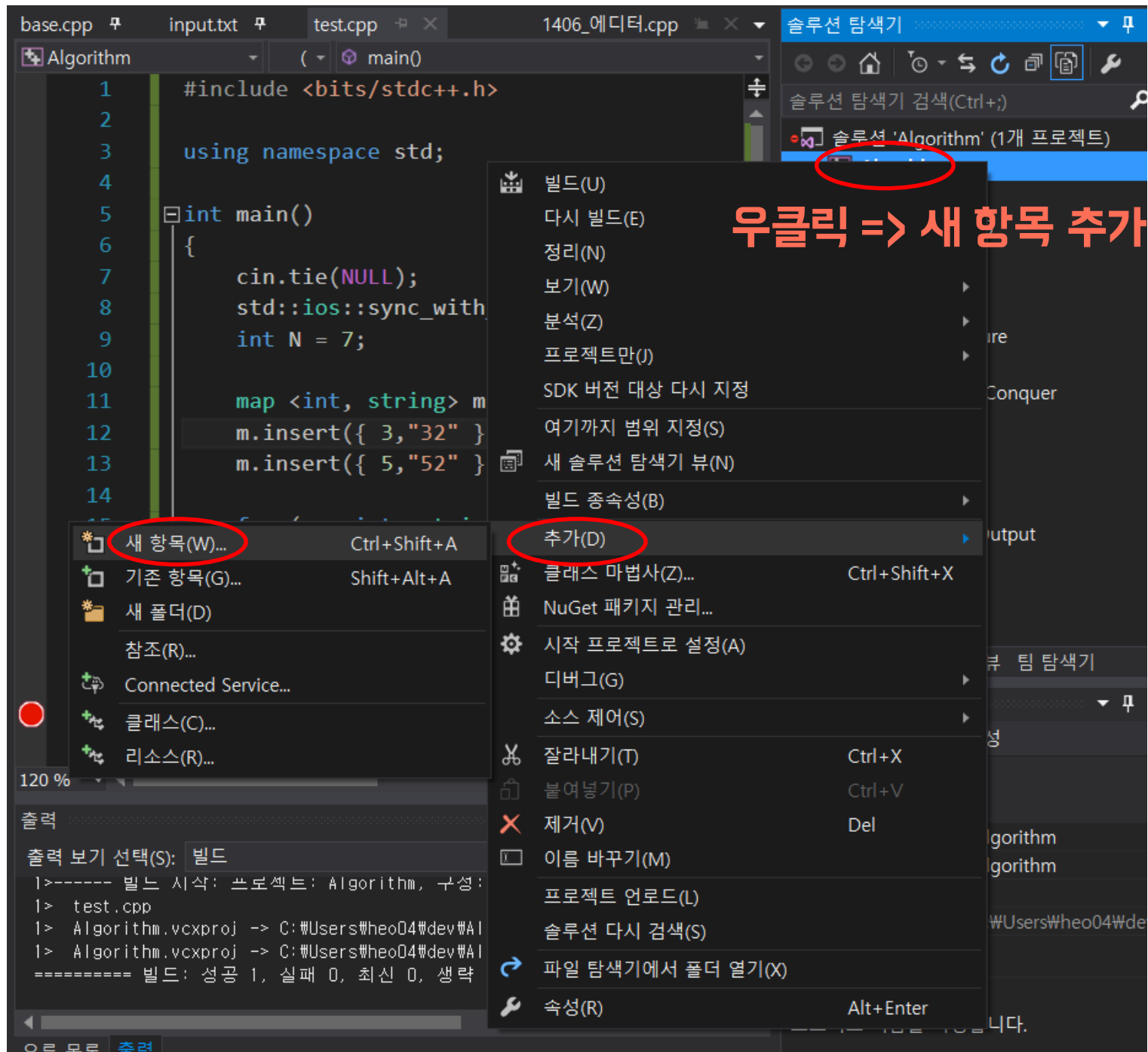
명령

실행할 디버그 명령입니다.

확인

취소

적용(A)



설치됨

Visual C++

- 코드
- 데이터
- 리소스
- Web
- 유틸리티
- 속성 시트
- 그래픽

온라인

정렬 기준: 기본값



설치된 템플릿 검색(Ctrl+E)



C++ 파일(.cpp)

Visual C++



헤더 파일(.h)

Visual C++

형식: Visual C++

C++ 소스 코드를 포함하는
다.

***.txt 파일 생성

[온라인으로 전환하거나 템플릿을 찾으려면 여기를 클릭하세요.](#)

이름(N):

input.txt

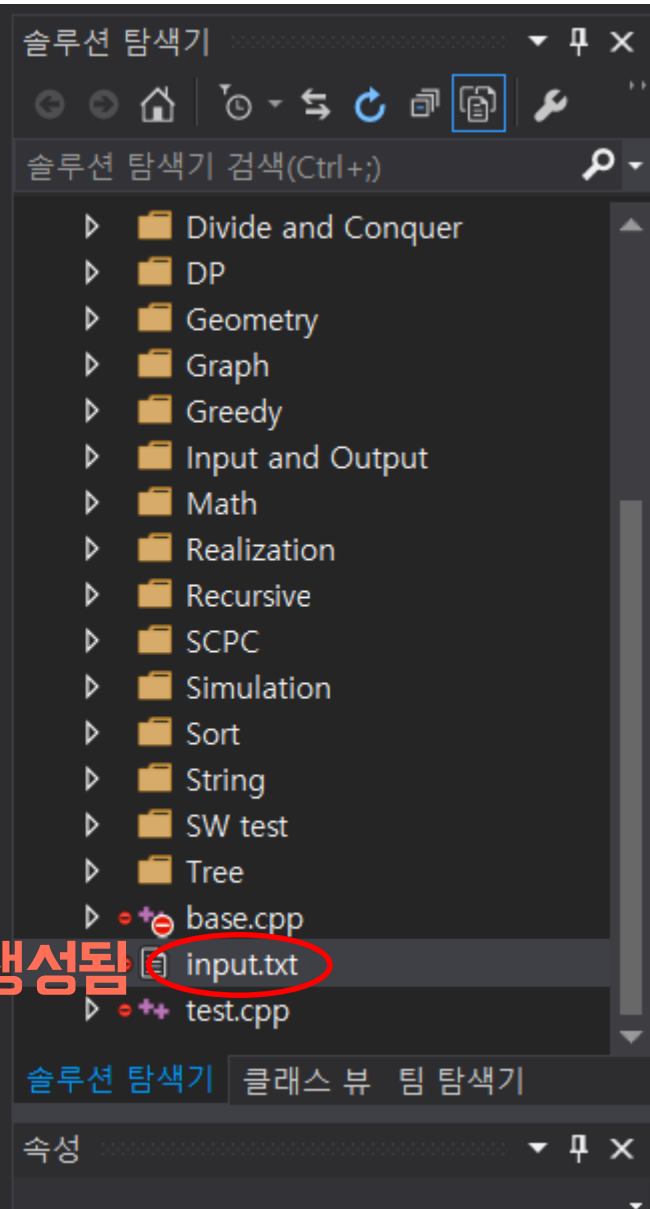
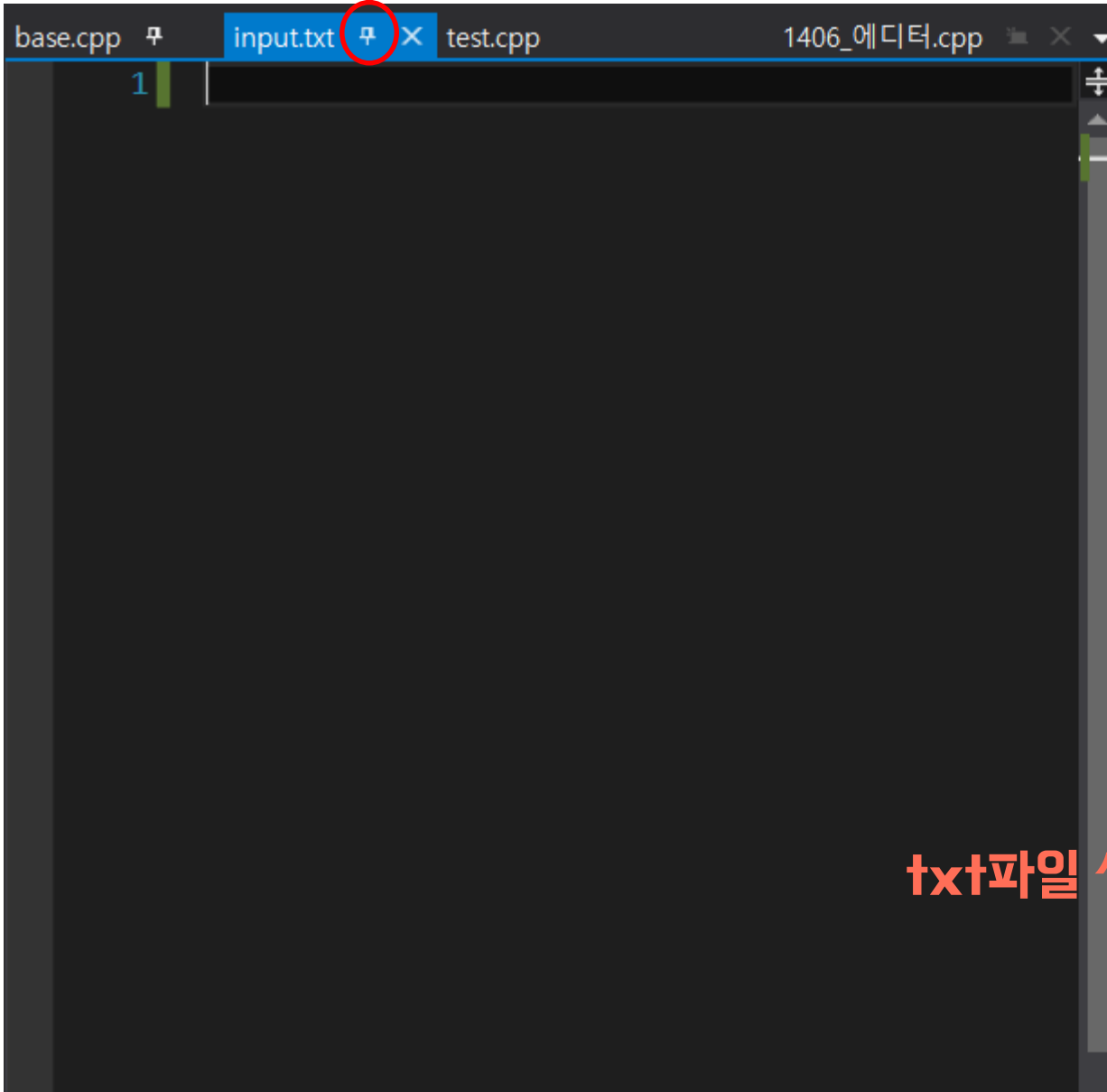
위치(L):

C:\Users\heo04\dev\Algorithm\Algorithm\Data Structure

찾아보기(B)...

추가(A)

압정 박아서 고정시켜놓고



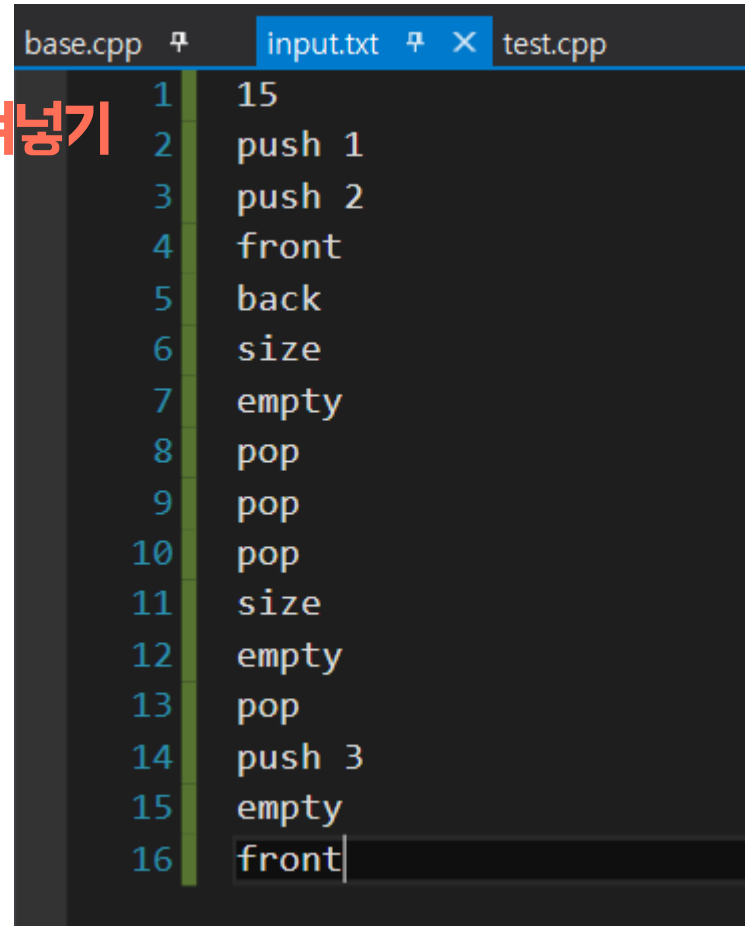
txt파일 생성됨

입력파일 복사

예제 입력 1 복사

```
15
push 1
push 2
front
back
size
empty
```

=> 붙여넣기



The screenshot shows a code editor with three tabs: 'base.cpp', 'input.txt', and 'test.cpp'. The 'input.txt' tab is active, displaying a list of 16 operations for a stack. The operations are: 1. 15, 2. push 1, 3. push 2, 4. front, 5. back, 6. size, 7. empty, 8. pop, 9. pop, 10. pop, 11. size, 12. empty, 13. pop, 14. push 3, 15. empty, and 16. front. The line numbers 1 through 16 are visible on the left side of the editor.

```
1 15
2 push 1
3 push 2
4 front
5 back
6 size
7 empty
8 pop
9 pop
10 pop
11 size
12 empty
13 pop
14 push 3
15 empty
16 front
```

base.cpp input.txt test.cpp 1406 에디터.cpp

Algorithm (main()

```
62 int n;
63 cin >> n;
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88 }
```

C:\WINDOWS\system32\cmd.exe

```
1
2
2
0
1
2
-1
0
1
-1
0
3
계속하려면 아무 키나 누르십시오 . . .
```

코드파일로 와서
Ctrl + F5

=> 자동으로 결과값 출력



감사합니다.

Made by 규정
