SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

알고리즘

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day1-1. 알고리즘 학습을 하는 이유

목표

- 알고리즘을 왜 학습해야 하는 지 이해하고 열심히 연습한다.

우리는 왜? 알고리즘을 해야 할까?

알고리즘 학습을 하는 이유가 무엇일까?

- **문제 해결 능력을 기르기 위해**
- **사고력 증진을 위해**
- **...**

다양한 이유가 있을 수 있지만, SSAFY에서는 명확하다.

바로 취업을 위해서이다!

왜 알고리즘 문제로 평가를 진행할까?

원하는 기업에 자기소개서를 제출하면서,
그동안 진행한 프로젝트를 정리한 포트폴리오도 같이 제출하지만

포트폴리오만으로는 기업에서 원하는 인재를 채용하기엔
어려움이 있다!

- **직접 진행한 프로젝트인지**
- **어느 정도 지원자가 참여한 프로젝트인지 등등 판단하기 어렵다**

그래서 도입된 것이 바로 알고리즘 테스트

그 방대한 프로젝트 코드를 다시 짜라고 시킬 수도 없으니
간단한 문제를 제출하여 코드 테스트를 진행하는 것이다!

기업에서는 다양한 능력을 판단하고 싶다

- 실제로 코드를 구현할 수 있는 지
- 얼마나 최적화된 코드를 작성하는 지
- 알고리즘들을 이해하고 활용할 수 있는 지
- 모든 예외사항들을 고려할 수 있는 지

알고리즘 문제 유형 정리

기업용 코딩테스트 문제는 그래서 다음과 같은 유형을 가진다

명세서가 주어진다.

- 지원자는 해당 명세서를 읽고 문제상황을 인식하고 어떤 결과를 원하는 지 빠르게 파악한다.

1. 구현 여부
   - 명세서를 읽고 문제를 해결하기 위한 코드를 설계한 뒤 실제로 구현할 수 있는 능력 측정
2. 최적화 여부
   - 단순 설계만으로는 시간/메모리 이슈가 생기는 문제를 해결할 수 있는 최적화 능력 측정
3. 관계형 Data(비선형) 해결 가능 여부
   - 알고리즘들을 이해하고 활용 할 수 있는 지
   - 모든 예외사항들도 고려하여 코드를 작성할 수 있는 지
   - +@ (로드밸런싱)

구현 문제를 학습하기 위한 방법

- 푼다
- 또 푼다
- 연습만이 살 길이다!

최적화 문제를 학습하는 방법

- 푼다
- 또 푼다
- 아자!!!

설마 관계형 Data도?!

프로그래밍 문제를 잘 푸는 방법은 쉽다.
계속 푼다.

결론

4차 산업혁명 이 후, 블라인드 테스트가 도입된 이 후
벌써 수 년이 흘렀다.
알고리즘 문제 유형들도 점점 복잡해지고, 난이도가 상승하기 시작했다.

그렇다고 새로운 알고리즘 유형이 나오는 것은 아니다!
→ 쉽고 단순한 유형이 서로 얽혀 나오는 것!

반복 학습만이 해결할 수 있다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day1-2. SW문제해결

시간복잡도

알고리즘의 성능을 설명하는 지표

- **어떻게 확인할까?**

→ 명령문의 실행 빈도 수를 체크한다.

왜? 시간복잡도를 알아야 할까?

시간이 제한되어 있다.

- 기업은 정해진 시간을 주고 지원자들의 능력을 테스트해야 한다.

코딩테스트를 볼 때,

어떤 알고리즘은 선택할 수 있을 지 그 기준을 세울 수 있다.

- ex) N의 크기가 20~30, 순열을 활용하는 문제 → 가지치기를 활용할 수 있을까?
- ex) N이 10만 이상 최단 거리 알고리즘 → 다익스트라로 접근할까?

코딩테스트에서 문제 접근 방법

1. 문제를 파악한다.
2. 어떤 알고리즘을 사용할 수 있을 지 후보를 줄 세운다.
3. 각 방법의 알고리즘들의 시간복잡도를 체크한다.
4. 정해진 시간 내 들어오는 알고리즘을 선택한다.
5. 문제를 풀이한다.

빅오 표기법

다양한 시간 복잡도 체크 방법이 있다.

- Big-O (빅오) : 최악의 경우를 체크
- Big-Omega(빅오메가) : 최적의 경우를 체크
- Theta (쎄타) : Big-O, Big-Omega의 평균값 체크

하지만, 정해진 시간 내 통과를 해야 하기 때문에
최악의 성능을 체크할 수 있는 빅오 표기법을 사용한다.

[참고] Big-O 표기법

참고한다.

Big-O 표기법	명칭	설명	예제 알고리즘
O(1)	상수 시간	입력 크기와 상관없이 일정한 시간	배열
O(logn)	로그 시간	N의 크기와 상관 없이 N의 크기가 커질수록 비약적으로 성능이 향상	Binary Search
O(n)	선형 시간	N의 크기에 비례해서 시간 증가	선형 탐색(1중 for문)
O(nlogn)	로그-선형 시간	N의 크기는 500만 미만	sort(정렬)
O(n^2)	이차 시간	N 크기가 10000 정도	중첩 반복문
O(2^n)	지수 시간	N의 크기 25 미만	피보나치 (재귀)
O(n!)	계승 시간	N의 크기 10미만	순열

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day1-3. DAT

챕터의 포인트

- 배열
- DAT 소개
- DAT 사용하기

배열

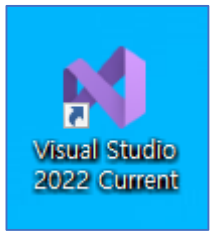
데이터를 저장하고 꺼내 온다.

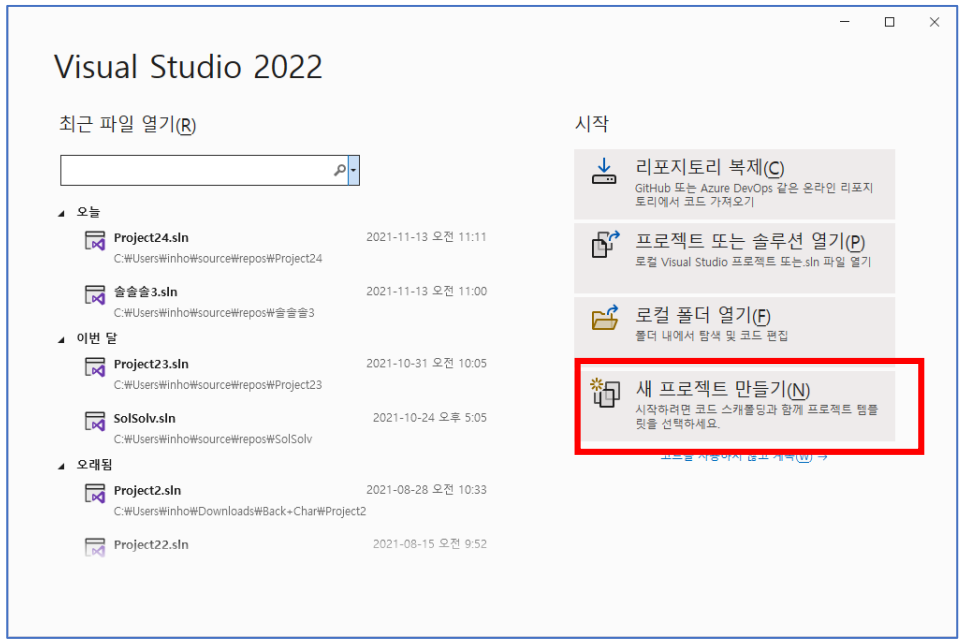
- **데이터가 메모리의 시작 주소를 기준으로 순서대로 연속해서 나열되어 있는 자료 구조**

```
int arr[100] = { 0 };
```

Visual Studio 실행

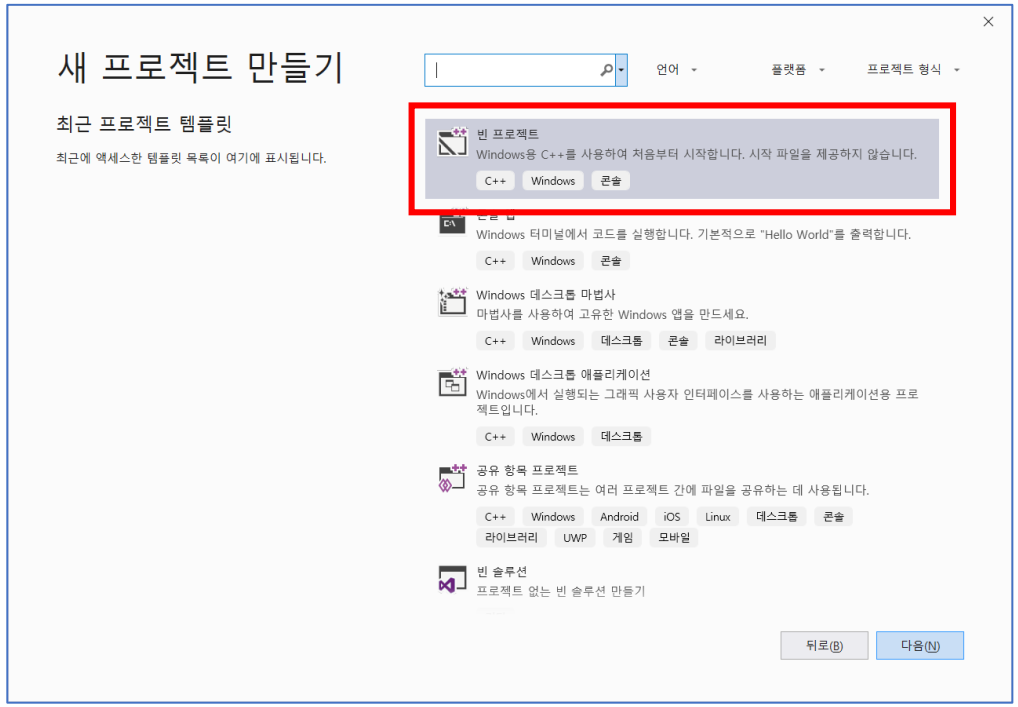
- **새 프로젝트 만들기 클릭**

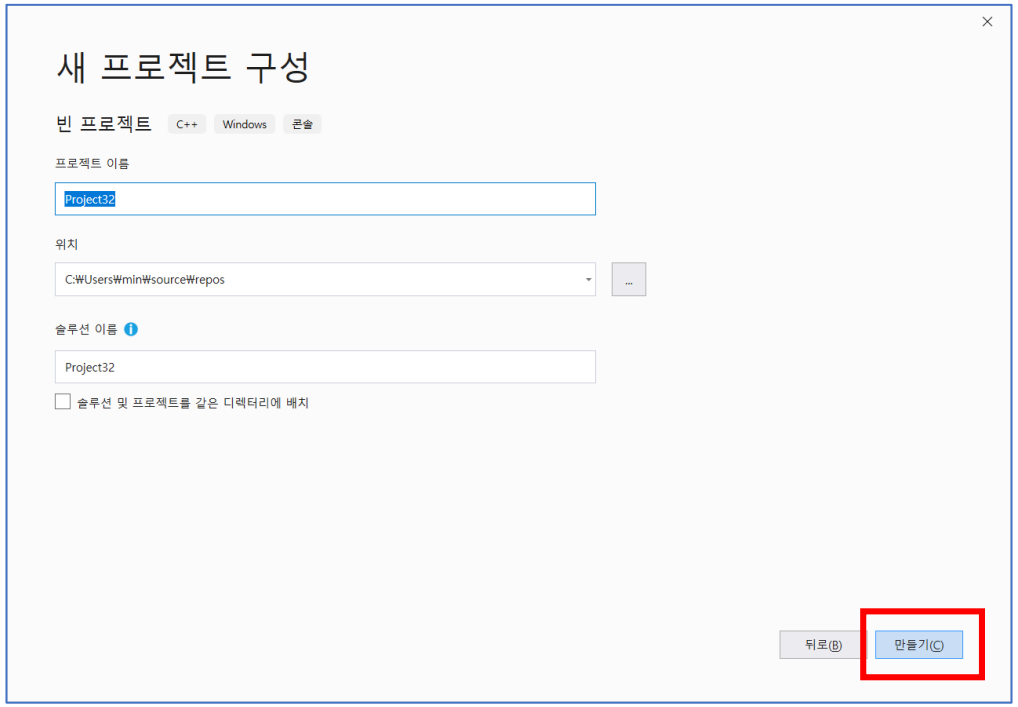




빈 프로젝트 만들기

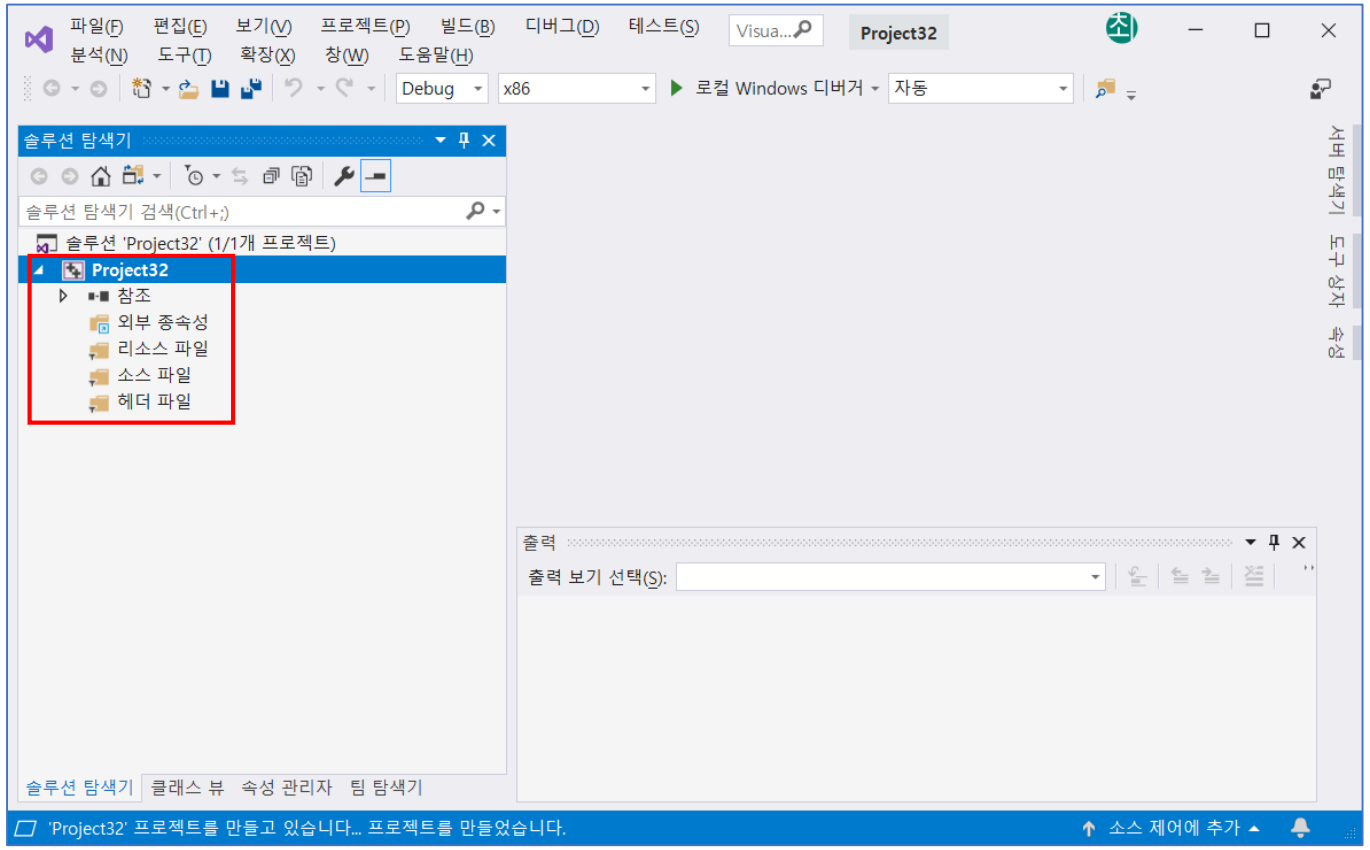
- **프로젝트 이름은 간단하게 작성하고 만들기 클릭**





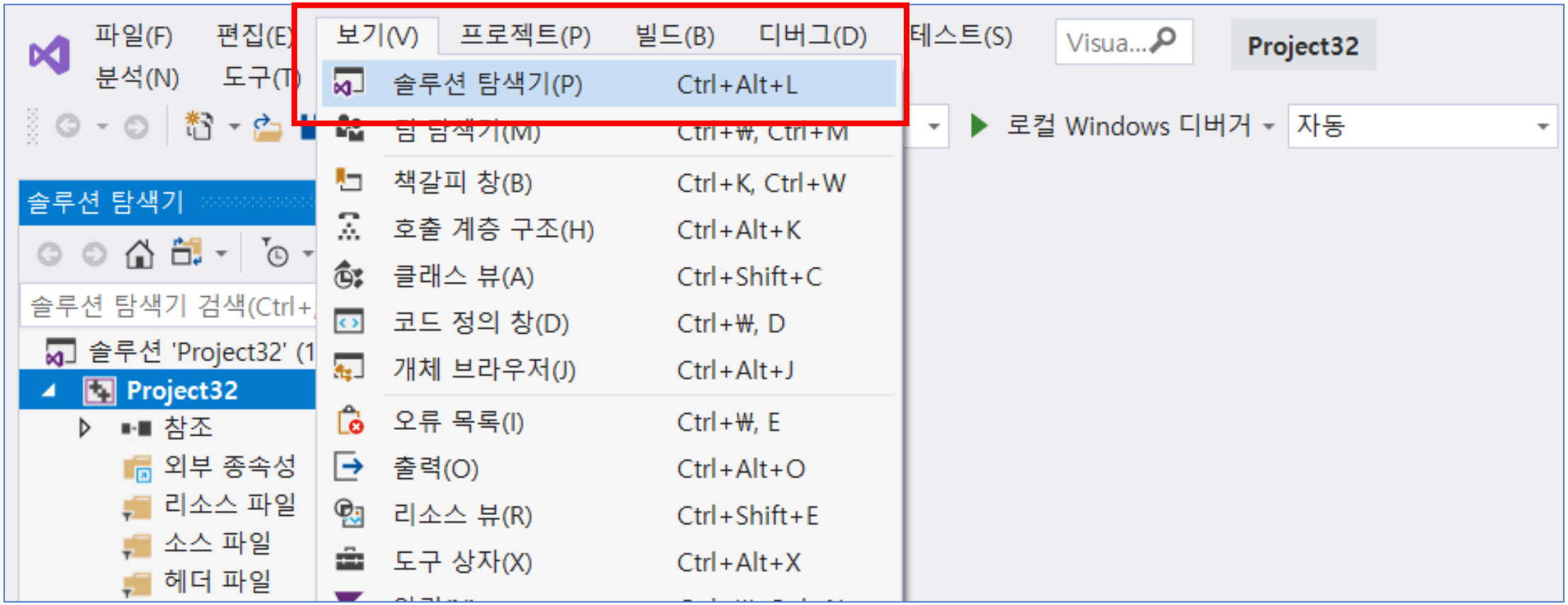
프로젝트 생성 완료

탐색기 탭 위치는 바뀔 수 있다

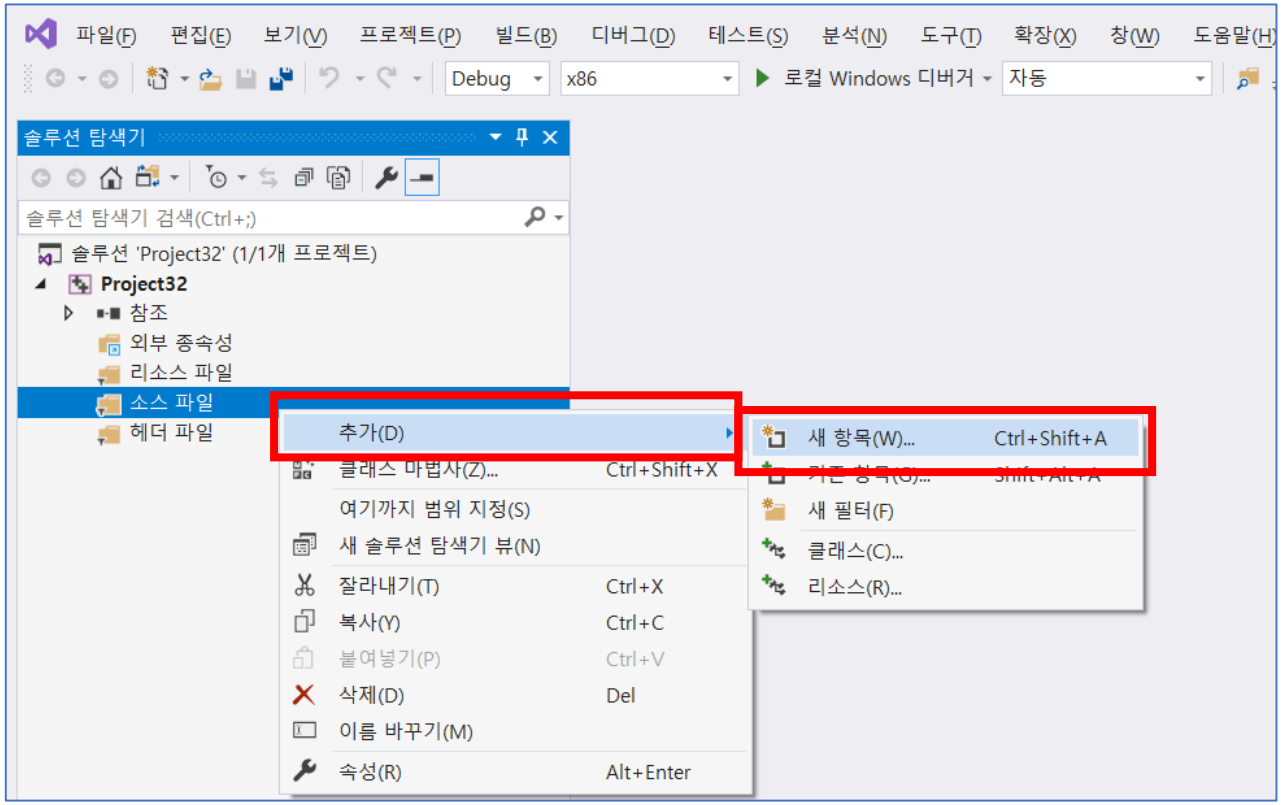


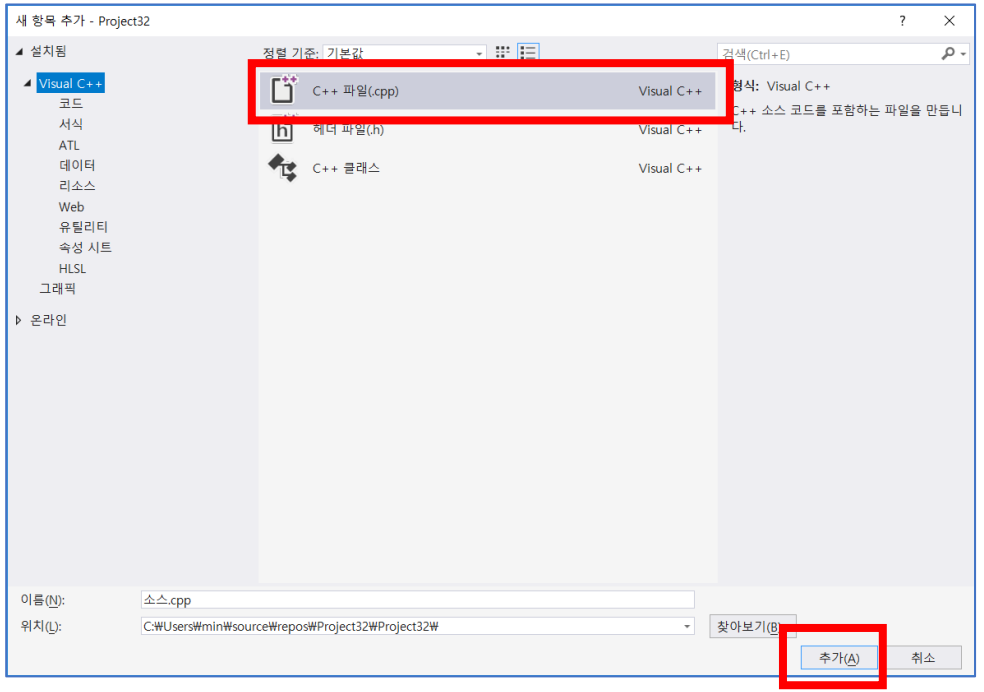
만약 솔루션탐색기가 보이지 않는다면

- **보기 → 솔루션 탐색기 클릭**



소스 파일에 마우스 우클릭 → C++ 파일 추가





코드를 작성한 뒤 빌드한다.

```
#include<iostream>
using namespace std;

int main() {
        int arr[100] = { 0 };
        for (int i = 0; i < 100; i++) {
                cout << arr[i];
        }
        return 0;
}
```

https://gist.github.com/hoconoco/96d4153d81f2243ee47594a8da24c108

배열을 다르게 사용한다.

단순하게 데이터를 저장하는 게 아니라

데이터를 정리해서 저장해서 관리할 것이다!

DAT 소개

배열 기반 자료 구조이다. 다음과 같은 특징을 갖는다.

1. 고정된 크기

- 테이블 크기는 키의 범위에 따라 미리 결정된다.
- ex) 키가 0에서 9까지라면 배열 크기는 10이 된다.

2. 빠른 검색 속도

- 시간 복잡도가 O(1)로, 특정 키의 존재 여부를 즉시 확인하거나 값을 가져올 수 있다.

3. 공간 낭비 가능성

- 키가 희소(sparse)한 경우, 메모리가 낭비될 수 있다.
- ex) 키가 1, 1000, 100000인 경우, 대부분의 인덱스가 비어 있게 된다.

DAT의 주 용도

주로 사용하는 용도

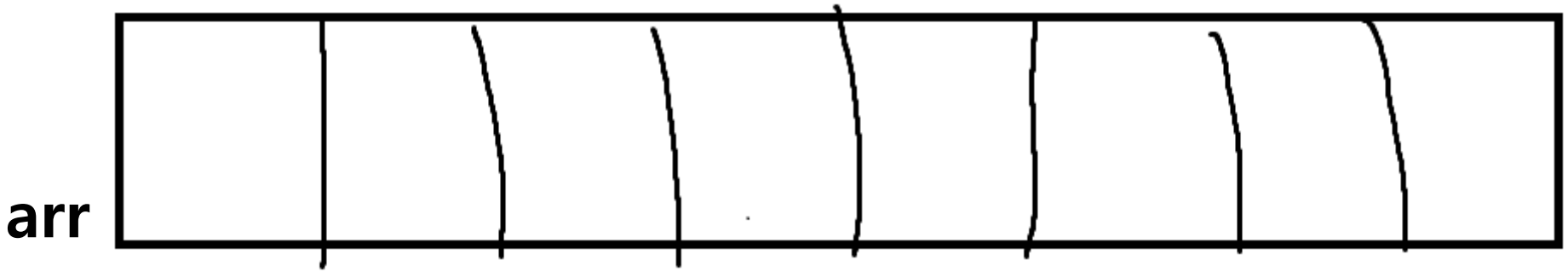
- 키의 존재 확인 (Presence Check)
- 정수 키와 값의 매핑
- 빈도 계산 (Frequency Counting)

매우 매우 중요하다.

어려운 내용이 아니다.

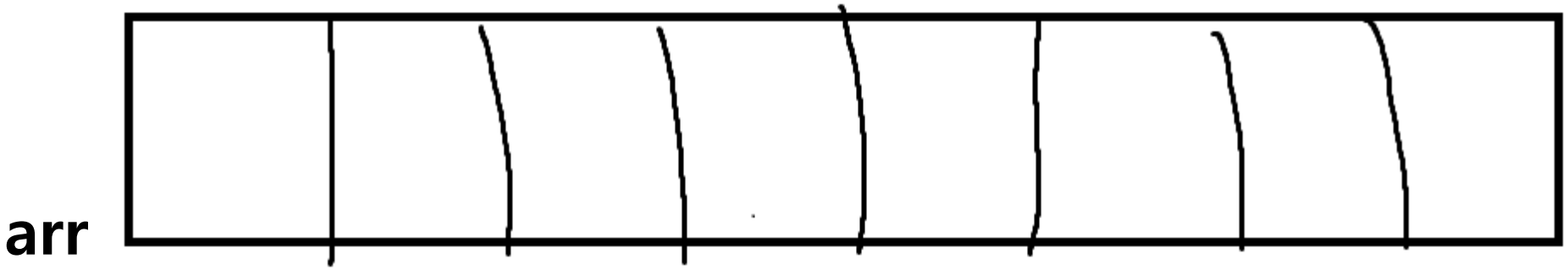
쉽게 인덱스에 의미를 부여하는 것이다!

배열이 있다



우리는 배열에 회원 정보를 저장할 것이다

- 이름
- 아이디
- 주민번호
- 전화번호
- 이메일
- 성별
- ..

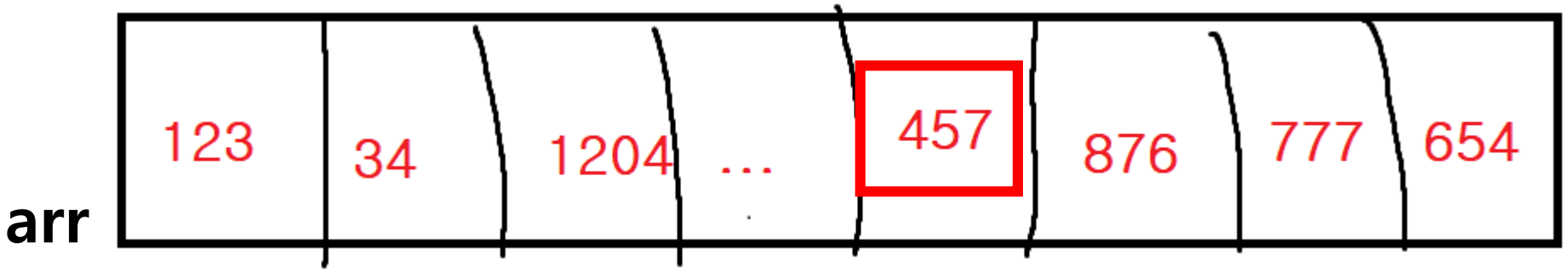


아이디로 유저들이 회원가입을 하고

아이디를 다음과 같이 배열에 저장했다

- 아이디는 정수로 1000만까지 사용하며,
- 배열의 크기가 1000만 이라 할 때,
- 457을 찾기 위해서 몇 번 데이터를 확인해야 할까? (최악의 경우)

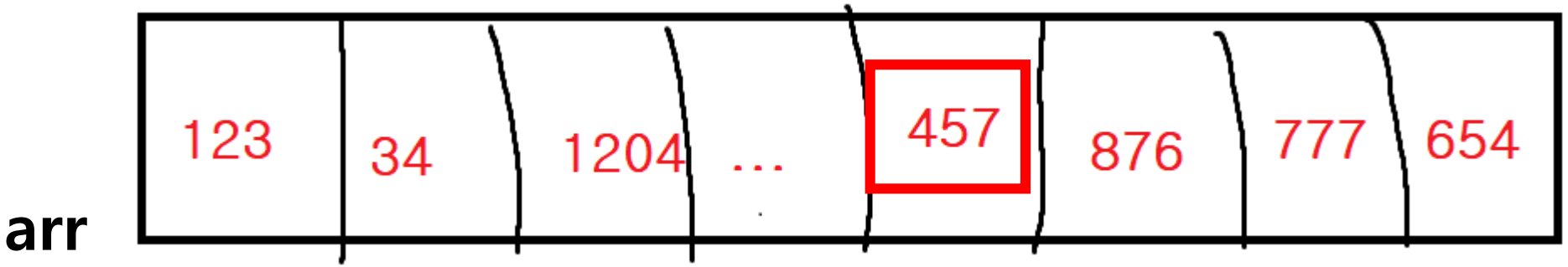
1000만 번 확인해야 한다 (매우 비효율적)



C/C++ 기준 1초당 1억 번 반복까지 안전

- 회원 정보를 1번 찾는 건 쉽다
- 만약 1000만 번의 회원 정보를 모두 조회해야 한다면?!

→ 데이터를 효율적으로 관리해야 할 필요가 있다



이제 아이디를 배열의 인덱스로 해서 데이터를 저장한다

- 0번 아이디의 정보는 arr[0] 에 저장한다
- 457번 아이디의 정보는 arr[457] 에 저장한다
- 검색 횟수 : 1회!

이런 방식으로 저장하는 것을 바로 DAT 라 한다!

DAT 사용하기

[결론] DAT 대표적인 문제 유형

1. 1234234987245 중 어떤 숫자가 나왔는 가?
   - arr[숫자] = 나온 횟수 형식으로 DAT 에 저장
   - arr[str[index]] = 1;

2. 1234234987245 중 어떤 숫자가 몇 번씩 나왔는 가?
   - arr[숫자] = 나온 횟수 형식으로 DAT 에 저장
   - arr[str[index]] ++;

3. 1234234987245 중 어떤 숫자가 어느 위치에 나왔는 가?
   - 어느 위치에 존재하는 가까지 구현이 가능하다!

코드를 작성한다.

배열이 있다.

DAT 에 해당 배열의 요소가 있는 지 저장한다.

```
int arr[10] = { 1,2,5,6,4,4,5,7,8,4 };
int DAT[10];
int main() {


        for (int i = 0; i < 10; i++) {
                DAT[arr[i]]= 1;
        }
```

https://gist.github.com/hoconoco/68623e357106829db2587e7c7d322c5c

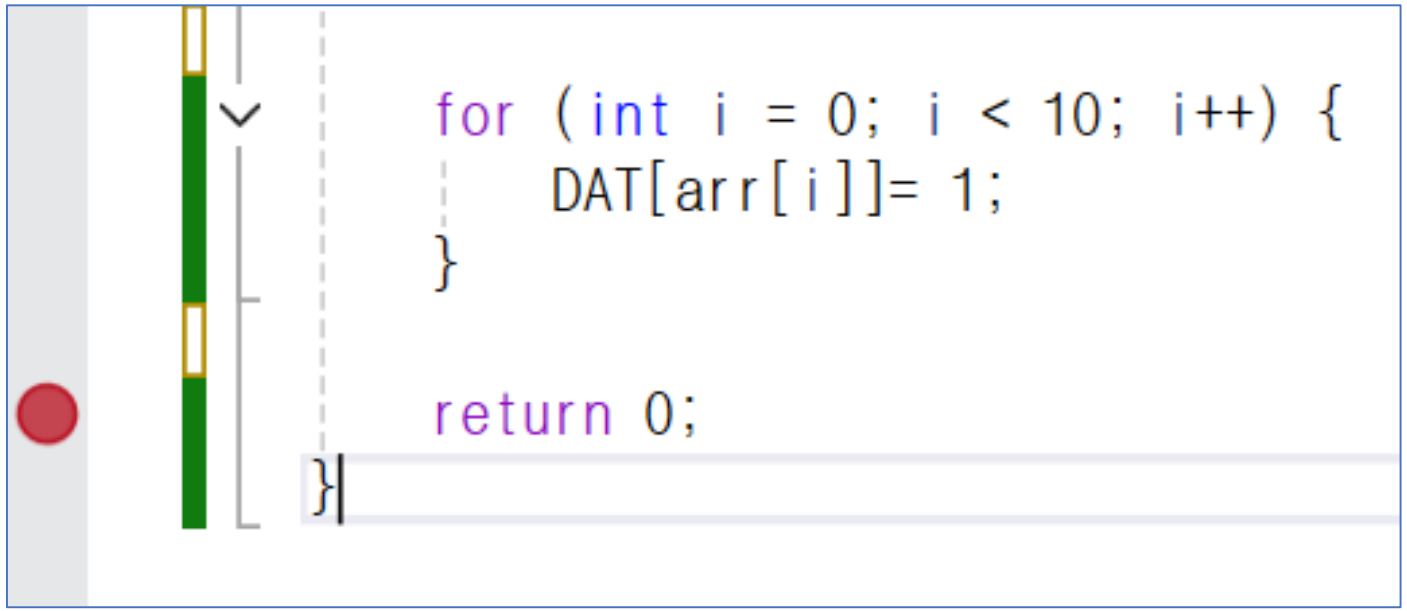
디버깅을 한다.

디버깅이란?

- 코드의 버그를 잡는 행위
- 코드를 한 줄씩 실행해가면서 그 결과를 확인한다.

- F5 : 디버깅
- F10 : 프로시저 단위 실행
- F9 : 브레이크 포인트

F9를 return 0; 에 한다.

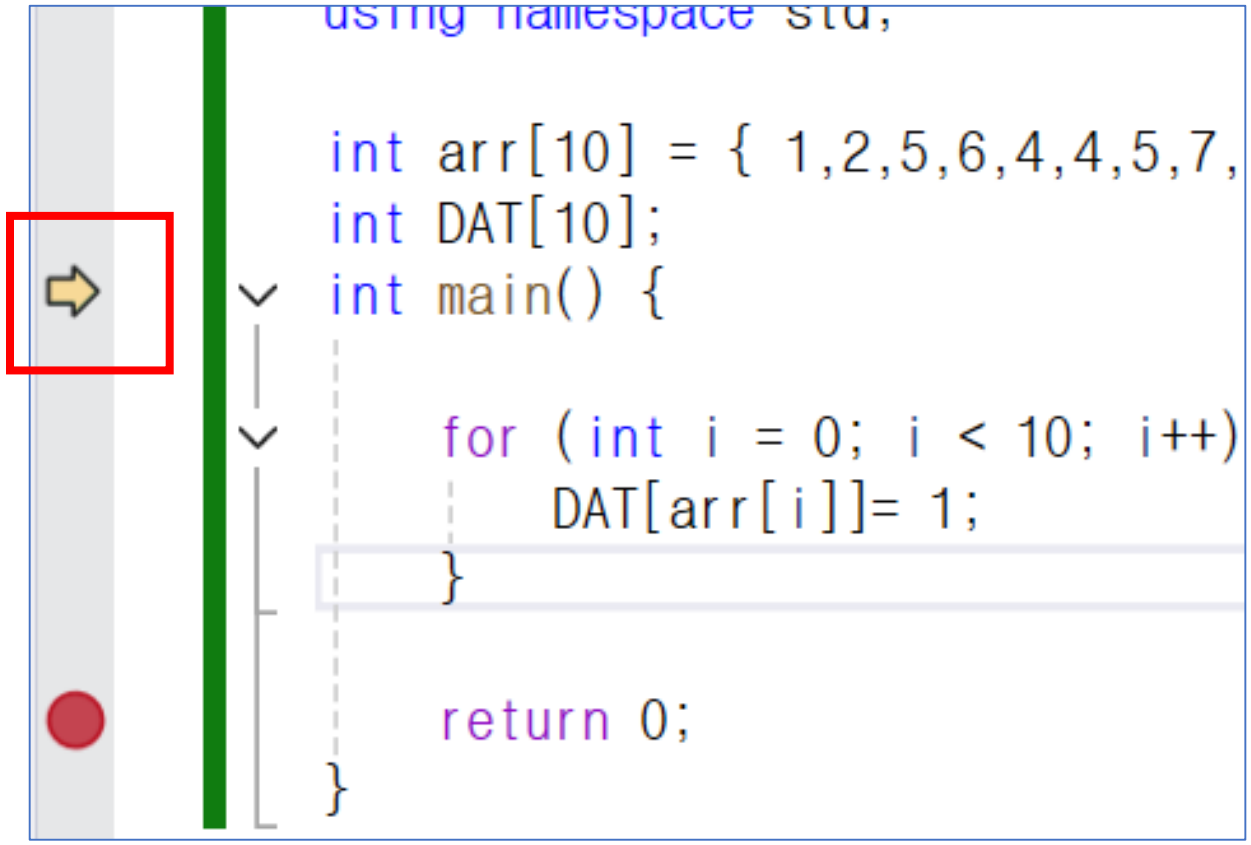


F10 키 입력

키보드의 F10키를 누른다.

F10키를 누르면 화살표가 한 줄씩 내려간다.

- **실제로 CPU가 실행하는 모습을 볼 수 있다.**



조사식 창 추가

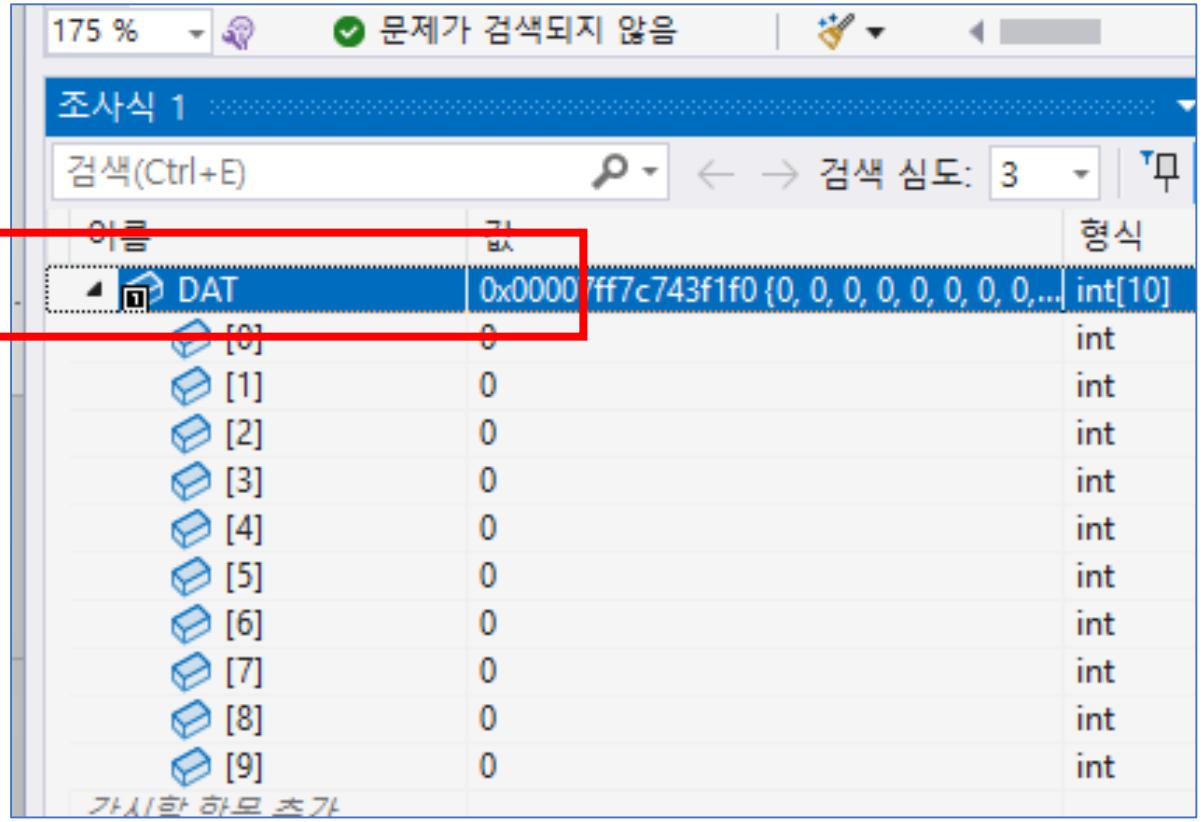
조사식 창을 추가해서

특정 변수나 배열의 값을 추적할 수 있다.

- **디버그 → 창 → 조사식 → 조사식1 클릭**

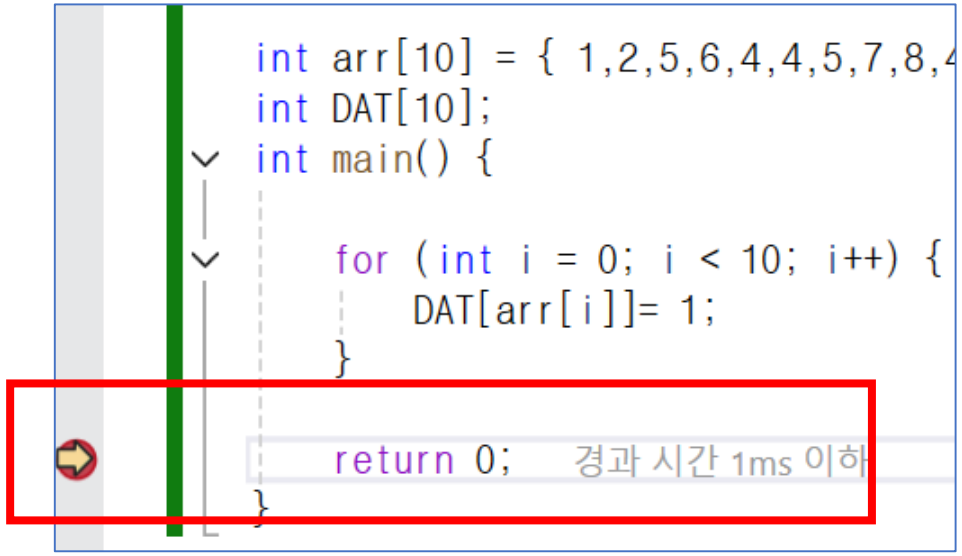
조사식에 원하는 변수 or 배열 이름을 입력

- **이제 F10을 누를 때마다 DAT에 값을 확인할 수 있다.**



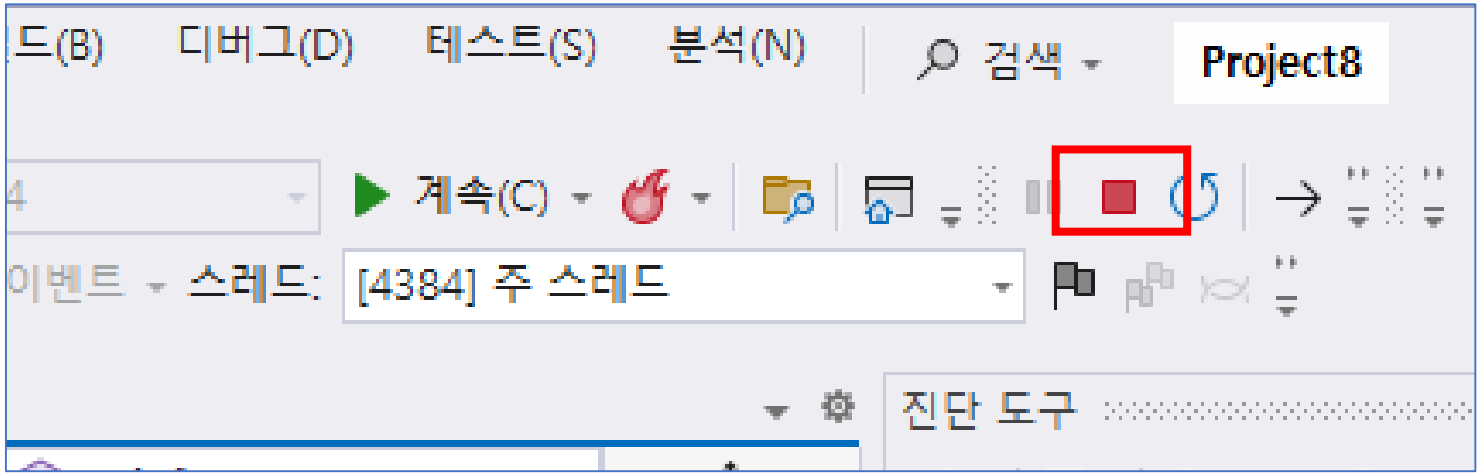
물론, 빠르게 처리도 가능하다

F10을 누르면 프로시저(실행 단위) 하나씩 실행하지만,
너무 코드가 길 경우,
브레이크 포인트를 걸고
해당 지점까지 F5(디버깅) 를 입력해서 한번에 실행할 수 있다



디버깅을 중단 한 뒤
처음부터 다시 진행한다.

어떻게 해당 코드로 DAT를 사용하는 지 확인한다.



코드를 작성하고 디버깅한다.

배열이 있다.

DAT 에 해당 배열의 요소가 몇 개 있는 지 저장한다.

```
int arr[10] = { 1,1,5,2,2,4,5,7,8,4 };
int DAT[10];
int main() {

        for (int i = 0; i < 10; i++) {
                DAT[arr[i]]++;
        }
```

https://gist.github.com/hoconoco/6e44c83cdeac37db29dce02b285d9554

주어진 문자열에서
A~Z 까지 문자가 존재하는 지 코드 작성하기

- **DAT 사용하기**

```
char str[100] = "SSAFYMINCODING";
```

1/0 으로 존재 여부 판단 가능

```
char str[100] = "SSAFYMINCODING";

int DAT[256] = { 0 };
//index : 문자 -> value : 존재하는 가?

for (int i = 0; str[i]; i++) {
        //str[i] 문자가 존재한다고 기록
        DAT[str[i]] = 1;
}
```

https://gist.github.com/hoconoco/703a4d7fefdfb94a5377447f0747a176

주어진 문자열에서
A~Z 문자가 각각 몇 개씩 존재하는 지 코드 작성하기

- **DAT 사용하기**

```
char str[100] = "SSAFYMINCODING";
```

++ 사용하기

```
char str[100] = "SSAFYMINCODING";

int DAT[256] = { 0 };
//index : 문자 -> value : 개수 측정

for (int i = 0; str[i]; i++) {
        //str[i] 문자가 <<한 개 더!>> 존재한다고 기록
        DAT[ str[i] ]++;
}
```

https://gist.github.com/hoconoco/b351cd82677e55a064e3af5099a9a2c4

주어진 문자열에서
두 쌍으로 이뤄진 문자가 각각 몇 개씩 존재하는 지 코드 작성하기

- DAT 사용하기

```
char str[100] = "SSAFYMINCODING";
```

DAT를 2차원으로 구현한다!

```
char str[100] = "SSAFYMINCODING";

//두 자리로 이뤄진 것을 찾는 것은 DAT를 2차원으로 구현하면 된다.
int DAT[256][256] = {0};
//index : 문자 -> value : 개수 측정

for (int i = 1; str[i]; i++) {
        DAT[ str[i-1] ][ str[i] ]++;
}
```

https://gist.github.com/hoconoco/bae95fb111690752b4a2a2cdcc3094ce

주어진 문자열에서
각 문자가 몇 번 인덱스에서 존재했는 지 코드 작성하기

- DAT 사용하기

```
A의 위치: 2
S의 위치: 0 1
Y의 위치: 4
Z의 위치: 없음
```

```
char str[100] = "SSAFYMINCODING";
```

DAT를 2차원으로 구현한다!

```
// DAT를 2차원 배열로 선언 (256개의 문자, 최대 100개의 위치 저장 가능)
int DAT[256][100] = { 0 }; // DAT[i][j]: i번 문자, j번째 위치
int count[256] = { 0 };    // 각 문자의 위치 개수를 저장할 배열

// 문자열 순회하며 각 문자의 위치 기록
for (int i = 0; str[i]; i++) {
    int ch = str[i]; // 현재 문자
    DAT[ch][count[ch]] = i; // ch 문자의 count[ch] 번째 위치에 i 저장
    count[ch]++; // 해당 문자의 개수 증가
}
```

https://gist.github.com/hoconoco/35b6e80cd1b1c4cc471630254d3f30fa

[결론] DAT 대표적인 문제 유형

1. 1234234987245 중 어떤 숫자가 나왔는 가?
   - arr[숫자] = 나온 횟수 형식으로 DAT 에 저장
   - arr[str[index]] = 1;

2. 1234234987245 중 어떤 숫자가 몇 번씩 나왔는 가?
   - arr[숫자] = 나온 횟수 형식으로 DAT 에 저장
   - arr[str[index]] ++;

3. 1234234987245 중 어떤 숫자가 어느 위치에 나왔는 가?
   - 어느 위치에 존재하는 가까지 구현이 가능하다!

DAT의 유형을 다뤄봤으니 이제 문제를 풀이한다.

[참고] DAT 범위를 벗어난다면?

DAT로 처리할 수 없는 범위를 벗어나게 되면
나중에 나올 HASH 자료구조를 사용한다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미