

# C++ Basic

## Hello C++!



연세대학교 컴퓨터과학과 재학중인 퓨터 정강희입니다.

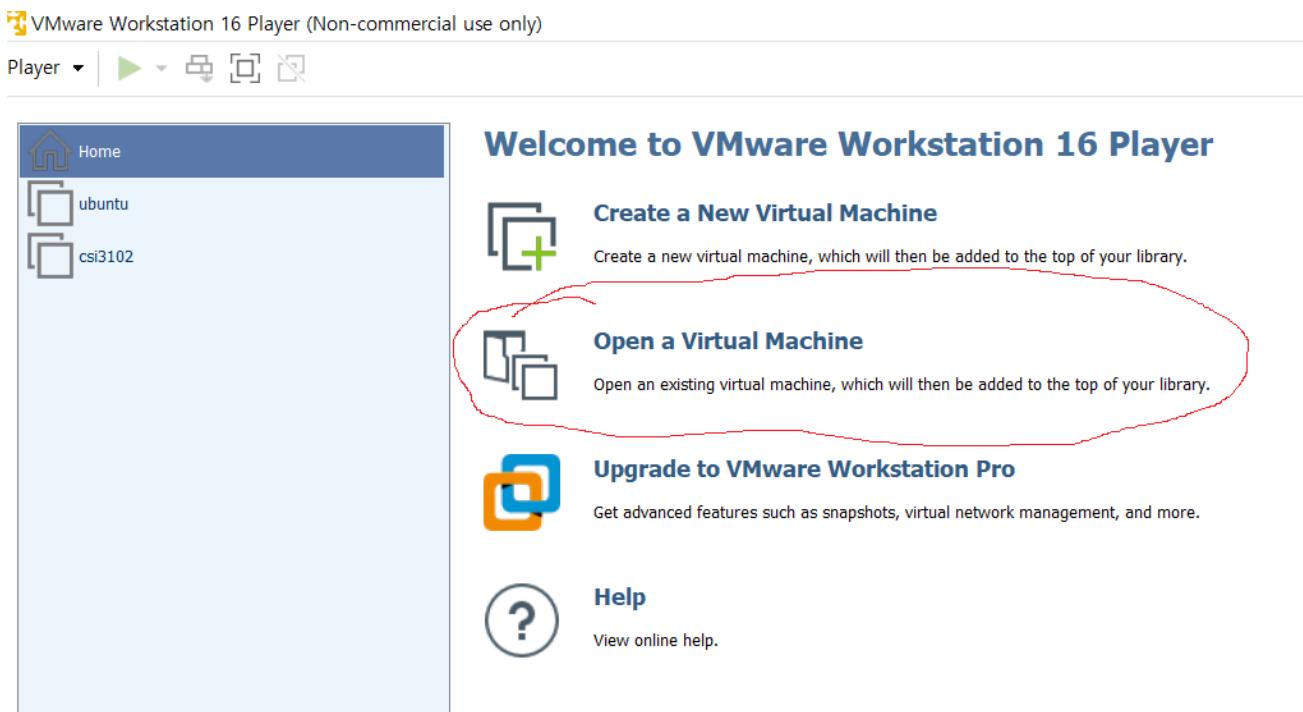
- (2018.05 – 2019.01) NC Fellowship program Game AI Track 참가 : pytorch와 강화학습을 이용한 Microchess AI 개발
- (2018.08) 2018 OSS개발자포럼 & 국민대학교 sw여름캠프 조교
- (2019.01) 2019 OSS개발자포럼 & 국민대학교 sw겨울캠프 조교
- (2019.07 – 2020.02) Unitsoft 코딩학원강사 : 영재고.과고.국제학교 학생반 강사
- (2020.08 – 2020.10) 2020 군장병 공개 SW 온라인 해커톤 참가 : 휴대폰 자동 비대면 반납 + 군인월급계산 앱 개발
- (2021.10 - 2021.12) 2021 국방기술을 활용한 창업경진대회 금상 : 원격 의료 서비스, 메디로그
- (2021.12 - ) Lablup DevOps 엔지니어 인턴
- (2021.12 - ) Elice 스쿨튜터

# ■ 수업 방식

# 개발환경설정

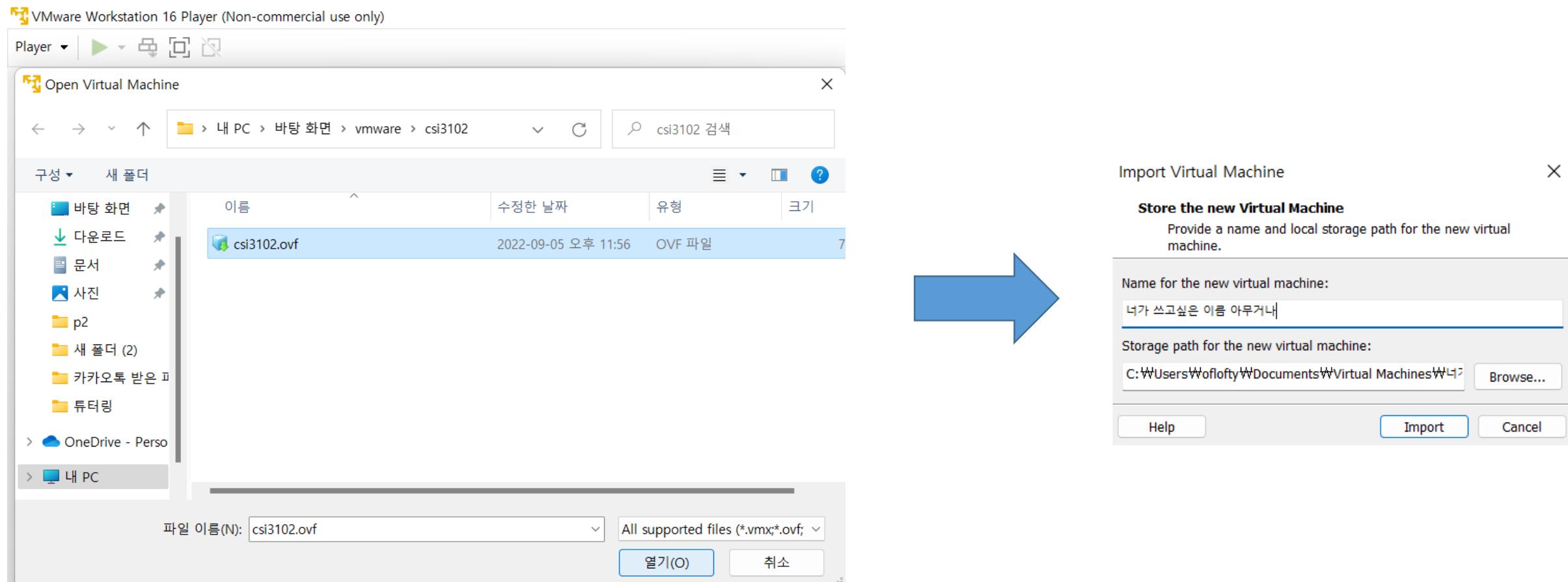
# Vmware 설치

1. <https://hpcp.yonsei.ac.kr/files/csi3102/vmware.zip>에 들어가서 이미지 파일 받기.
2. <https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html>에 들어가서 VMware Workstation Player를 다운받자.
3. 적당한 위치에 압축 풀고 VMware Workstation Player 실행한 다음에 open a virtual machine을 클릭



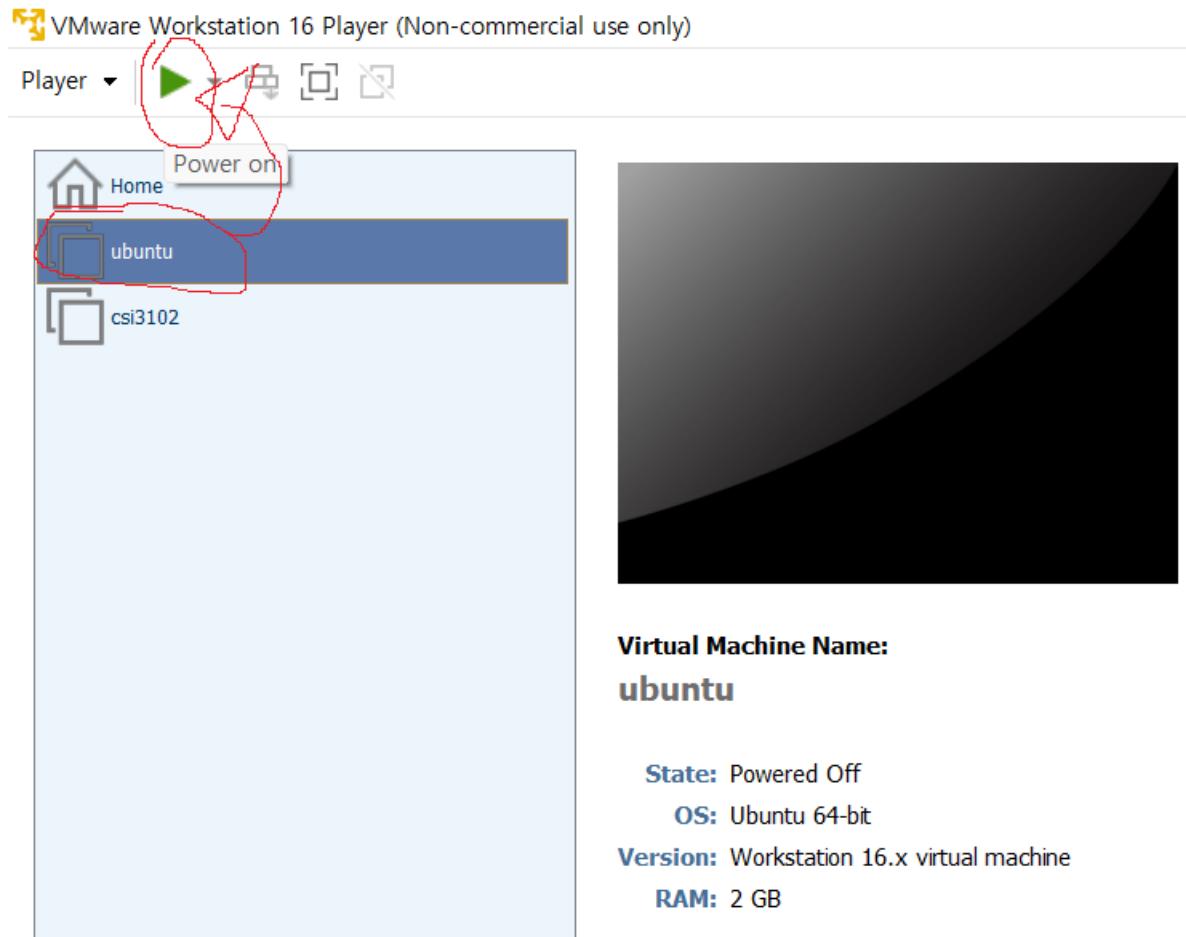
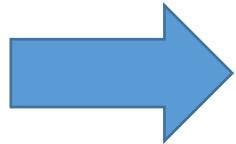
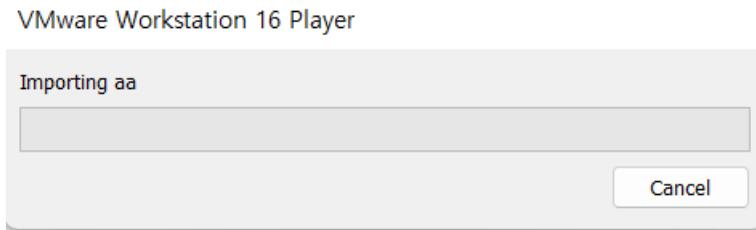
# Vmware 설치

4. 압축을 푼 파일에 있는 ovf 파일을 열자. 그 다음에 이 os에 붙여줄 이름을 너 맘대로 영어로 쓰자.



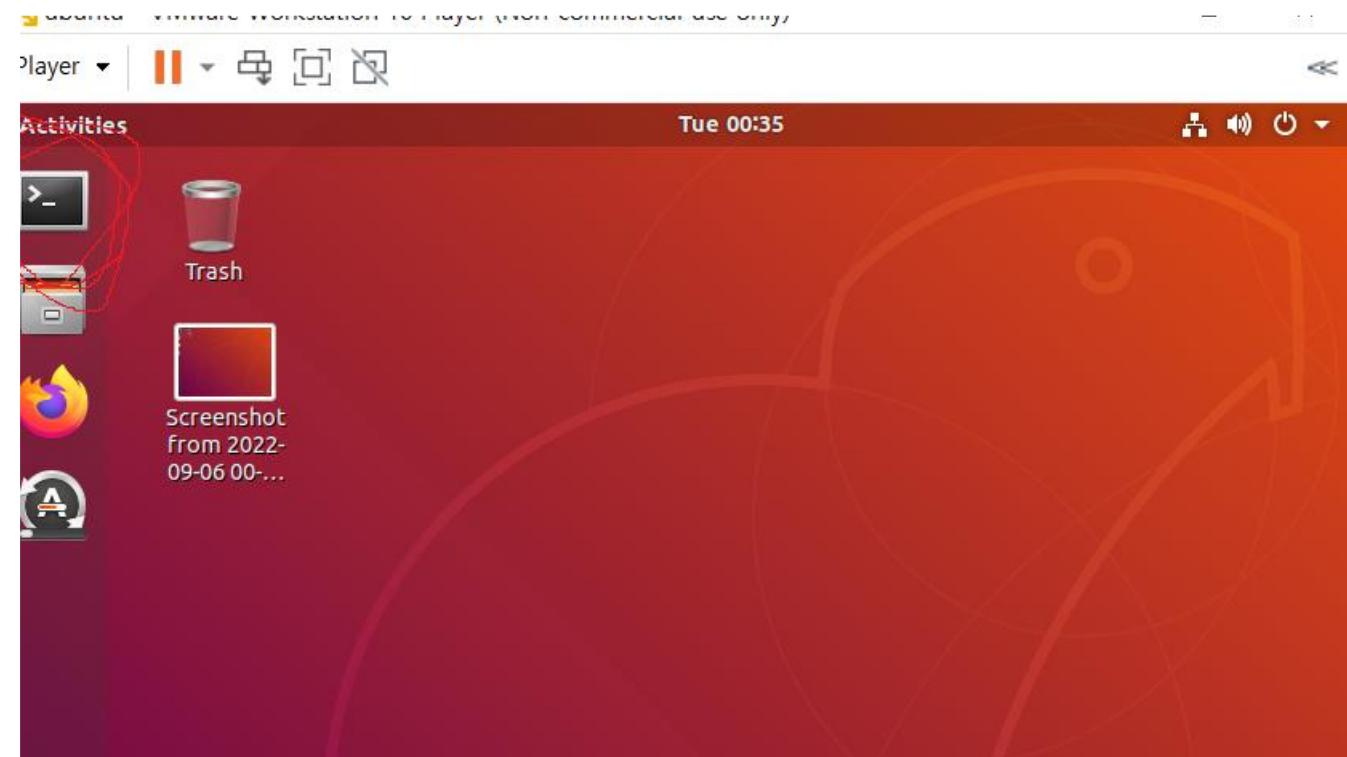
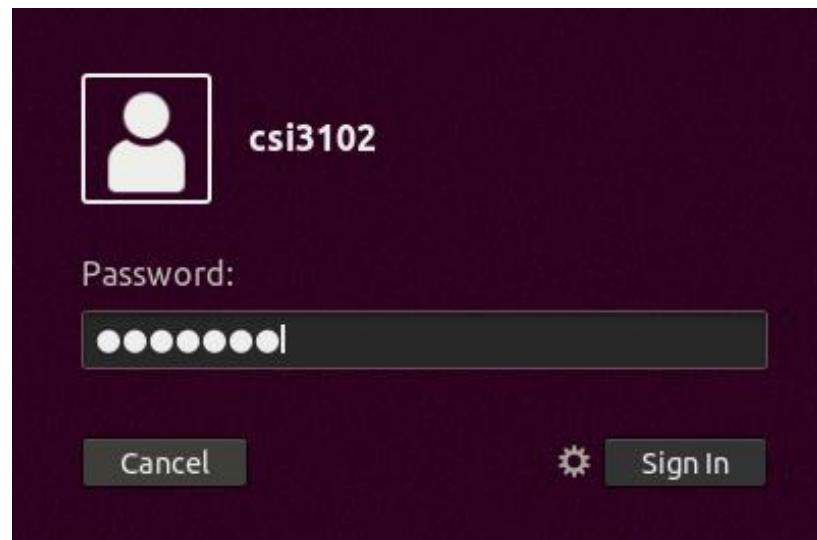
# Vmware 설치

5. 설치가 완료될때까지 기다리면 됨. 그 후에 설치된 가상환경의 우분투 이미지파일을 실행하자.



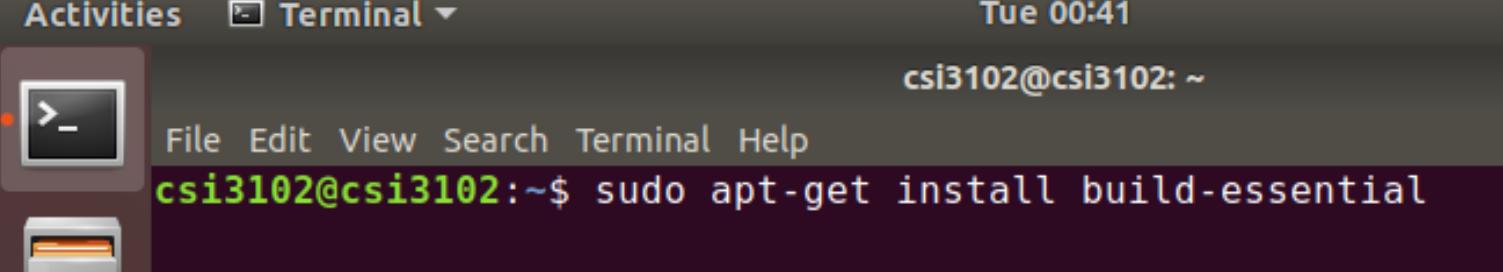
# Vmware 설치

6. 비밀번호는 위에 있는 csi3102를 그대로 치면 됨. 그러면 드디어 여러분은 윈도우가 아닌 우분투 환경에서 뭔가를 할 수 있게 되었어요!



# 필수 패키지 설치

6. 터미널을 켜서 이것저것 설치를 해줍시다.



```
Activities Terminal Tue 00:41
File Edit View Search Terminal Help
csi3102@csi3102:~$ sudo apt-get install build-essential
```

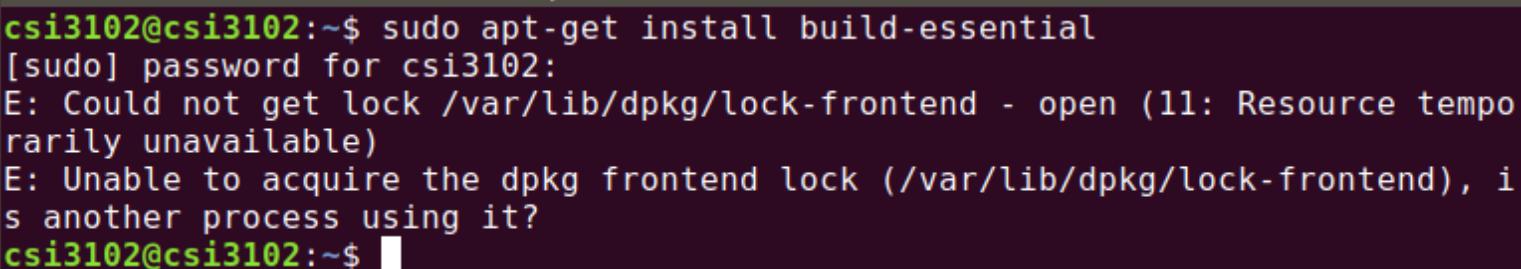
6-1. 만약 이렇게 에러가 발생한다면...우선 터미널 여시고 모든 프로세스를 죽여줍니다~!

1) sudo killall apt apt-get

만일 진행중인 프로세스가 없다라고 뜨면, 아래와 같이 하나하나씩 디렉토리를 삭제해주세요.

- sudo rm /var/lib/apt/lists/lock
- sudo rm /var/cache/apt/archives/lock
- sudo rm /var/lib/dpkg/lock\*

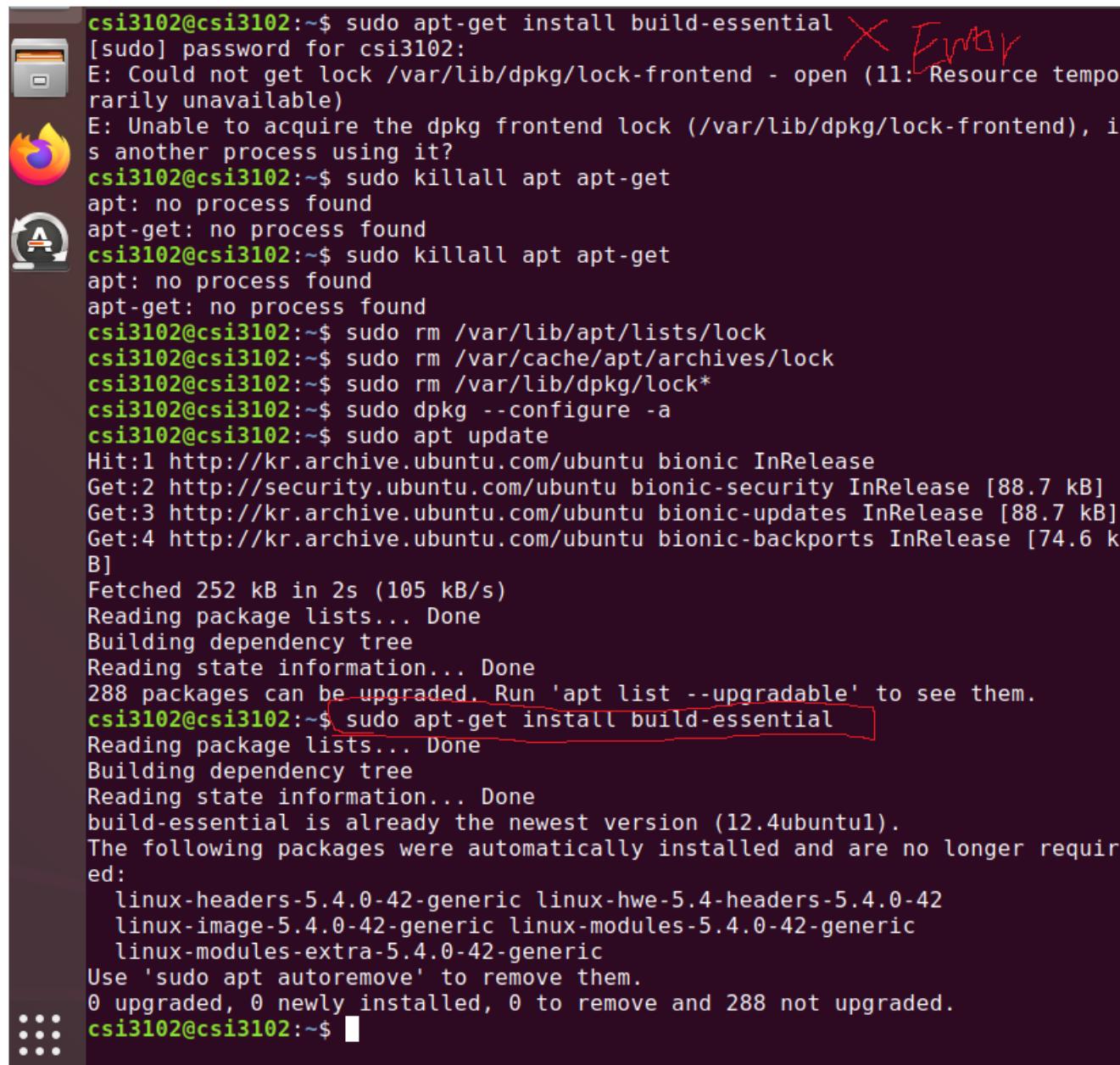
sudo dpkg --configure -a 를 하시고 sudo apt update



```
csi3102@csi3102:~$ sudo apt-get install build-essential
[sudo] password for csi3102:
E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource temporarily unavailable)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?
csi3102@csi3102:~$
```

# 필수 패키지 설치

6-2. 다시 sudo apt-get install build-essential를 실행해서 필수 패키지를 설치하자.



```
csi3102@csi3102:~$ sudo apt-get install build-essential
[sudo] password for csi3102:
E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource temporarily unavailable)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?
X Entry
csi3102@csi3102:~$ sudo killall apt apt-get
apt: no process found
apt-get: no process found
csi3102@csi3102:~$ sudo killall apt apt-get
apt: no process found
apt-get: no process found
csi3102@csi3102:~$ sudo rm /var/lib/apt/lists/lock
csi3102@csi3102:~$ sudo rm /var/cache/apt/archives/lock
csi3102@csi3102:~$ sudo rm /var/lib/dpkg/lock*
csi3102@csi3102:~$ sudo dpkg --configure -a
csi3102@csi3102:~$ sudo apt update
Hit:1 http://kr.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:3 http://kr.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://kr.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Fetched 252 kB in 2s (105 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
288 packages can be upgraded. Run 'apt list --upgradable' to see them.
csi3102@csi3102:~$ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
The following packages were automatically installed and are no longer required:
  linux-headers-5.4.0-42-generic linux-hwe-5.4-headers-5.4.0-42
  linux-image-5.4.0-42-generic linux-modules-5.4.0-42-generic
  linux-modules-extra-5.4.0-42-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 288 not upgraded.
csi3102@csi3102:~$ █
```

# 필수 패키지 설치

7. 그 후에 sudo apt-get install curl을 통해서 다른 필수 패키지를 설치하자.

```
csi3102@csi3102:~$ sudo apt-get install curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.4.0-42-generic linux-hwe-5.4-headers-5.4.0-42
  linux-image-5.4.0-42-generic linux-modules-5.4.0-42-generic
  linux-modules-extra-5.4.0-42-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libcurl4
The following NEW packages will be installed:
  curl libcurl4
0 upgraded, 2 newly installed, 0 to remove and 288 not upgraded.
Need to get 379 kB of archives.
After this operation, 1,053 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://kr.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcurl4
  amd64 7.58.0-2ubuntu3.20 [220 kB]
Get:2 http://kr.archive.ubuntu.com/ubuntu bionic-updates/main amd64 curl amd
  64 7.58.0-2ubuntu3.20 [159 kB]
Fetched 379 kB in 2s (161 kB/s)
Selecting previously unselected package libcurl4:amd64.
(Reading database ... 194923 files and directories currently installed.)
Preparing to unpack .../libcurl4_7.58.0-2ubuntu3.20_amd64.deb ...
Unpacking libcurl4:amd64 (7.58.0-2ubuntu3.20) ...
Selecting previously unselected package curl.
Preparing to unpack .../curl_7.58.0-2ubuntu3.20_amd64.deb ...
Unpacking curl (7.58.0-2ubuntu3.20) ...
Setting up libcurl4:amd64 (7.58.0-2ubuntu3.20) ...
Setting up curl (7.58.0-2ubuntu3.20) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.4) ...
csi3102@csi3102:~$
```

# ■ Vs code 설치

8. 마이크로소프트 GPG 키를 /etc/apt/trusted.gpg.d/ 에 다운로드

```
sudo sh -c 'curl https://packages.microsoft.com/keys/microsoft.asc |  
gpg --dearmor > /etc/apt/trusted.gpg.d/microsoft.gpg'
```

9. Visual Studio Code를 다운로드 받을 저장소 추가

```
sudo sh -c 'echo "deb [arch=amd64]  
https://packages.microsoft.com/repos/vscode stable main" >  
/etc/apt/sources.list.d/vscode.list'
```

10. 추가 저장소에서 패키지 목록 가져오기

```
sudo apt-get update
```

11. VS Code 설치

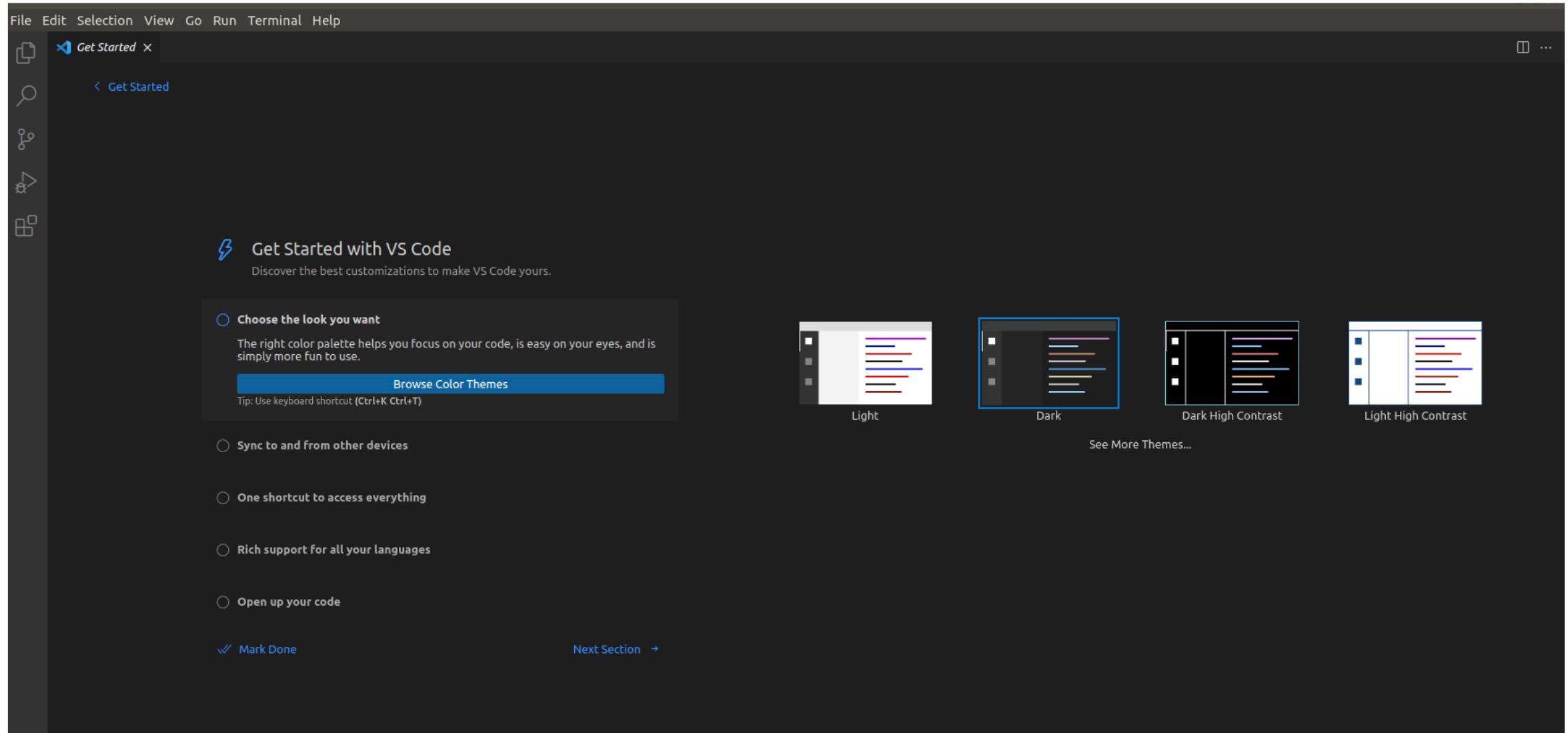
```
sudo apt-get install code
```

# VS code 설치

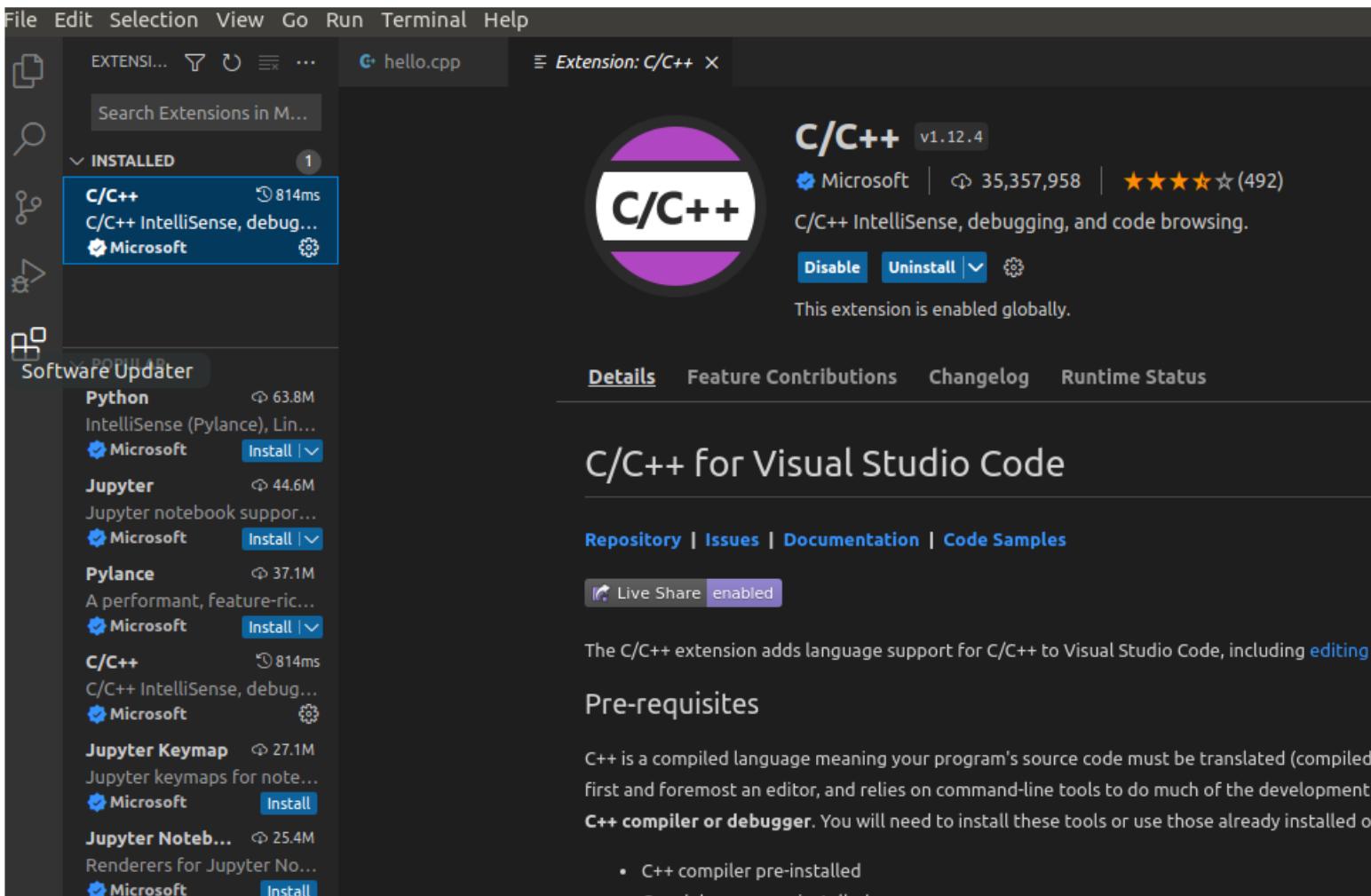
```
cs102@cs102:~$ sudo sh -c 'curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > /etc/apt/trusted.gpg.d/microsoft.gpg'
gpg: WARNING: unsafe ownership on homedir '/home/cs102/.gnupg'
      % Total    % Received % Xferd  Average Speed   Time     Time   Current
          Dload  Upload Total Spent   Left Speed
100    983  100    983    0     0  4292      0 --:--:--:--:--:-- 4311
cs102@cs102:~$ sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main" > /etc/apt/sources.list.d/vscode.list'
cs102@cs102:~$ sudo apt-get update
Get:1 https://packages.microsoft.com/repos/vscode stable InRelease [3,958 B]
Get:2 https://packages.microsoft.com/repos/vscode stable/main amd64 Packages [322 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:4 http://kr.archive.ubuntu.com/ubuntu bionic InRelease
Get:5 http://kr.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:6 http://kr.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Fetched 578 kB in 2s (346 kB/s)
Reading package lists... Done
cs102@cs102:~$ sudo apt-get install code
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.4.0-62-generic linux-hwe-5.4-headers-5.4.0-42
  linux-hwe-5.4-headers-5.4.0-62 linux-image-5.4.0-62-generic
  linux-modules-5.4.0-62-generic linux-modules-extra-5.4.0-62-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libvulkan1
The following NEW packages will be installed:
  code libvulkan1
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 85.4 MB of archives.
After this operation, 361 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://packages.microsoft.com/repos/vscode stable/main amd64 code amd64 1.71.0-1662018389 [85.3 MB]
```

# VS code 설치

## 12. 아아... 익숙한 화면이다...(아님 말고)



## 13. Extension page를 열어서 c/c++ Extension을 설치하자.



# C++ 개발환경 설정

14. 그 후에 바탕화면에 assn1이라는 폴더를 만들고 hello.cpp 파일을 만든 다음에 cd assn1을 입력해서 디렉토리를 이동한 후 g++ -c hello.cpp && g++ -o hello hello.o && ./hello 를 입력해서 컴파일한다. 잘 나오면 끝!

The screenshot shows the Visual Studio Code interface. The title bar says "Activities Visual Studio Code" and the status bar says "Tue 01:47" and "hello.cpp - Visual Studio Code".

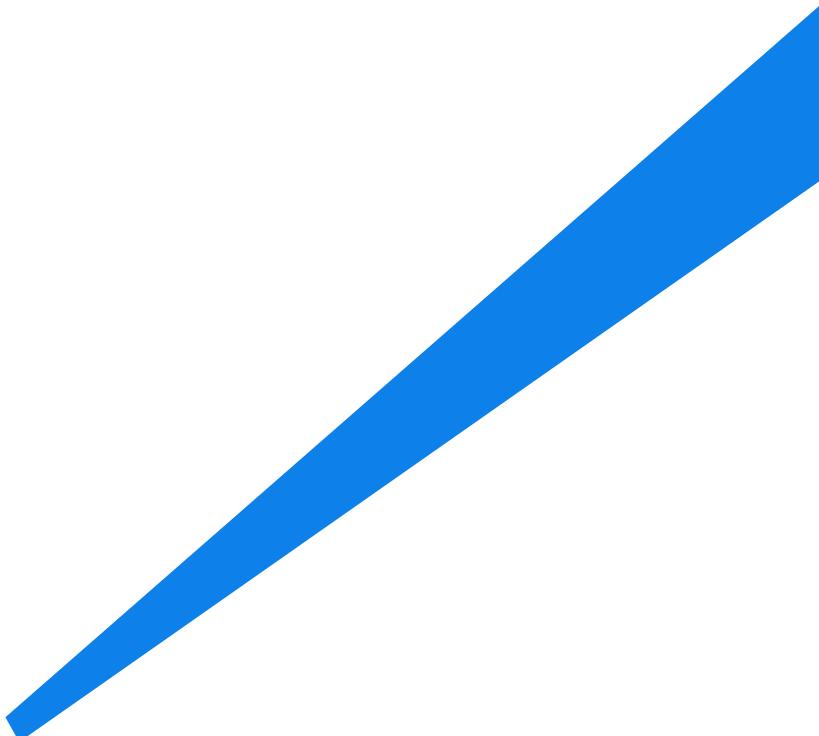
The left sidebar has icons for a terminal, file browser, search, and repository cloning. It also displays a message: "You have not yet opened a folder." with a "Open Folder" button, and instructions for opening a folder and cloning a repository.

The center is the code editor with the file "hello.cpp" open. The code is:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     cout<<"Hello, world!\n";
8
9     return 0;
10}
```

The bottom right is the terminal tab, which shows the command-line history:

- cs13102@cs13102:~/assn1\$ g++ -c hello.cpp
- cs13102@cs13102:~/assn1\$ g++ -o hello hello.o
- cs13102@cs13102:~/assn1\$ ./hello
- cs13102@cs13102:~/assn1\$ Hello, world!
- cs13102@cs13102:~/assn1\$ g++ -c hello.cpp && g++ -o hello hello.o && ./hello
- cs13102@cs13102:~/assn1\$ Hello, world!



C++?

# C++ 이란?

B 언어

C 언어



C++은 많은 [프로그래밍 언어](#) 중에 한 종류예요.

톰슨 씨가 만든 B 언어를 발전시켜서 C 언어,

그리고 C언어를 바탕으로 확장 시킨 C++이 탄생했다고 해요.

# C++ 왜 써요?

1. 이식성이 좋다.

C++로 작성한 프로그램을 어떤 컴퓨터에서 실행시켜도 작동해요.

2. 컴퓨터 제어가 강력하다.

컴퓨터가 가지고 있는 저장 장치에 직접 접근할 수 있어요.

## ■ 근데 난 왜 안 쓸까?

1. 제어가 강력한 만큼 **실수에 치명적이다**.

컴퓨터 **저장 장치를 직접 제어**해서 **실수**를 하면 **매우 큰 오류**가 날 수도 있어요.

2. 초보자가 배우기에 **난이도가 어렵다**.

**복잡한 프로그램**을 만들 수 있지만 **초보자**에게는 어렵다고 느껴질 수 있어요.

# I main 함수

가장 처음 시작하는  
메인 함수!

```
int n;  
void fun() {}  
int main() {  
}  
void func2() {}
```

프로그래머가 C++ 코드를 작성하고 “명령을 실행해!”

하고 명령을 하면 기능을 수행하기 위해

가장 처음 시작하는 부분이 바로 **메인 함수(main 함수)**에요.

# I main 함수

```
int main() {  
    // 실행할 코드 적는 곳..  
}
```

메인 함수의 구조는 이렇게 생겼어요.

int와 괄호() 안의 내용은 “함수” 파트에서 자세하게 배울 거예요.

중괄호[] 안에 실행하려는 코드를 적으면 실행시킬 수 있어요.

# I main 함수

```
int main() {  
    // 실행할 코드 적는 곳..  
}
```

메인 함수의 구조는 이렇게 생겼어요.

int와 괄호() 안의 내용은 “함수” 파트에서 자세하게 배울 거예요.

중괄호[] 안에 실행하려는 코드를 적으면 실행시킬 수 있어요.



입출력

# 입출력



우리가 보통 사용하는 프로그램은 사용자와 상호작용을 해요.

사용자의 **입력**을 받고, 입력에 따라 결괏값을 화면에 **출력**해줘요.

이것을 합쳐서 **입출력**이라고 해요.

# 입출력

#include를 통해  
라이브러리를  
넣어줘요.

```
#include <iostream> // 입출력 기능 제공  
  
#include <vector>  
  
#include <string>
```

C++에는 **입출력**을 담당하는 **입출력 함수**가 있어요.  
입출력 함수를 쓰기 위해서는 **라이브러리**라는 것을  
코드에 작성해 줘야 해요.

# 입출력

포함시켜줘

cin, cout 을 담고  
있는 라이브러리!

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

다시 본론으로 넘어와서, **입출력**을 담당하는 **cin, cout**은  
**iostream**이라는 라이브러리에서 불러와서 사용할 수 있어요.  
그러면 밑에 추가로 있는 **namespace**는 무엇일까요?

# namespace

```
namespace std {  
    // cin 함수가 있어요.  
    // cout 함수가 있어요.  
}
```

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



**using namespace std**의 의미는 “std”라는 이름 공간

안에 있는 **입출력 함수(cin, cout)**를 사용하겠다는 의미예요.

# namespace

```
using namespace std;  
cout << “이름 공간 사용”;  
cin >> name;
```



```
std::cout << “이름 공간 미사용”;  
std::cin >> name;
```

# 입출력

```
cin >> 입력 받을 내용;
```

```
cout << “출력할 내용”;
```

입출력을 담당하는 **cin**과 **cout**은 위에 코드와 같이 각각  
**<<, >>** 과 함께 사용해요.

# 입출력

## 코드

```
#include <iostream>  
  
using namespace std;  
  
void main() {  
    int i;  
    cout << "숫자를 입력하세요\n";  
    cin >> i;  
}
```

## 결과 화면

```
숫자를 입력하세요  
// 입력할 때까지 기다려요.
```

예시와 같이 `cout <<` 을 사용해 출력해주고

`cin >>` 을 이용해 사용자 입력을 받을 수 있어요.

# 입출력

이스케이프 시퀀스	의미
\n	줄바꿈(개행)
\t	수평 탭
\b	백슬래시
\'	작은 따옴표
\"	큰 따옴표

`cout` 출력문에는 특수 문자를 사용해서 표현할 수 없는  
기능, 문자를 표현해줄 수 있어요.



변수

# ■ | 변수

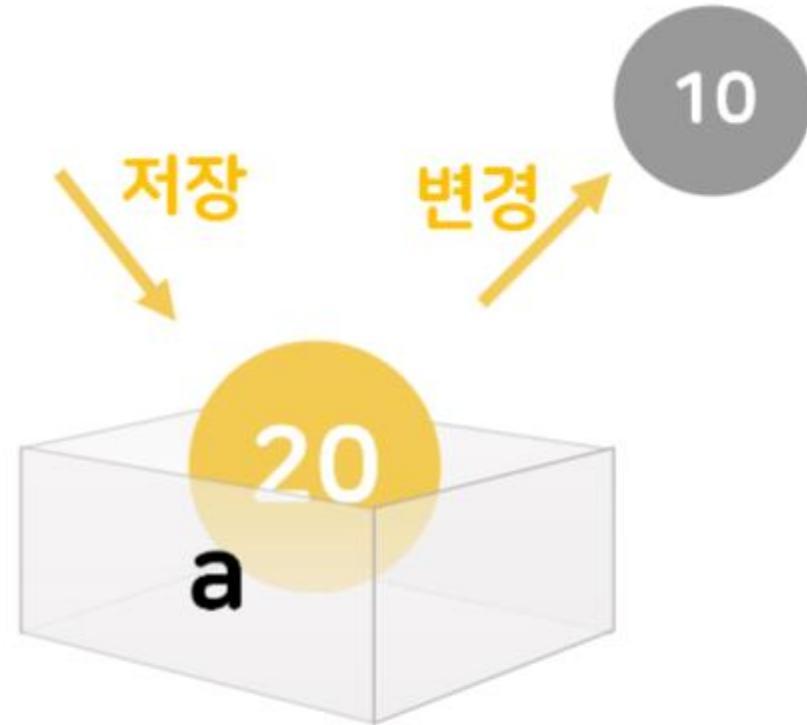
```
변수  
int a = 10;
```



변수는 값(데이터)을 저장할 수 있도록 프로그램에 의해  
특정 이름이 붙여진 공간이에요.

# ■ 변수

```
int a = 10;  
a = 20;
```



즉, 변수는 특정한 값을 **저장**하고 또는 값을 **변경할 수 있어요**.

## I 변수 이름 정하기

1. 변수 이름은 영문자, 숫자, 언더스코어(\_)로만 구성한다.
2. 변수는 숫자로 시작할 수 없다.
3. 공백을 포함할 수 없다.
4. C++에 이미 정의된 이름은 쓸 수 없다. (예: cout, int 등)

// 변수 선언의 잘못된 예

```
int 123abc = 12;
```

```
int int = 10;
```

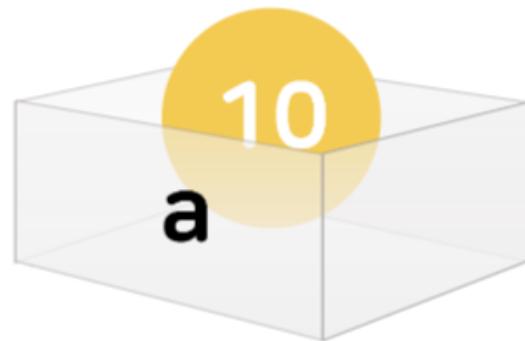
```
int hello elice = 50;
```

```
int !@#abc = 20;
```

# ■ 변수 선언, 초기화

10으로 변수를 초기화!

```
int a; // 변수 선언  
a = 10; // 변수 초기화
```



변수 선언을 하면 처음에는 들어 있는 값이 없어요.

최초로 변수에 값을 저장하는 행위를 변수 초기화라고 해요.

# ■ 변수 선언, 초기화



**초기화를 해주지 않은 변수**를 사용하려고 하면 어떻게 될까요?

초기화를 해주지 않으면 **쓰레기 값**이라는 이상한 값이 나와요.

# ■ | 변수의 범위(scope)

```
int main() {  
    { int a; }  
    cout << a; // 나도 쓸래!  
}
```

변수를 선언하고 초기화했다고  
어디서나 항상 저장된 값을 사용할 수 있는 것은 아니에요.

# ■ | 변수의 범위(scope)

중괄호 블록 안에서만  
사용할 수 있어요.

```
int main() {  
    int a = 10;  
    { int b = 5; }  
    cout << a; // 가능!  
    cout << b; // 오류!  
}
```

지역 스코프는 중괄호 안의 범위를 의미해요.

중괄호{} 안에서 선언된 변수는 중괄호 안에서만 사용할 수 있어요.

이렇게 선언된 변수는 지역 변수라고 불러요.

# ■ | 변수의 범위(scope)

변수 선언을 감싸는  
중괄호가 없어요.

```
int a = 19;  
  
int main() {  
  
    cout << a; // 나도 쓸래!  
  
}
```

전역 스코프는 어느 곳에서든 다 사용할 수 있어요.

전역 스코프로 선언된 변수는 전역 변수라고 불러요.

# ■ 주석

```
// 이것은 메인 함수예요. → 주석  
int main() {  
}
```

주석은 코드에 대한 이해를 돋는 설명을 적을 때

주석을 사용해서 메모를 해줘요.

실행할 때 주석은 무시가 된답니다.

한 줄 주석 : //

// 이것은 한 줄 주석이에요.

여러 줄 주석 : /\* \*/

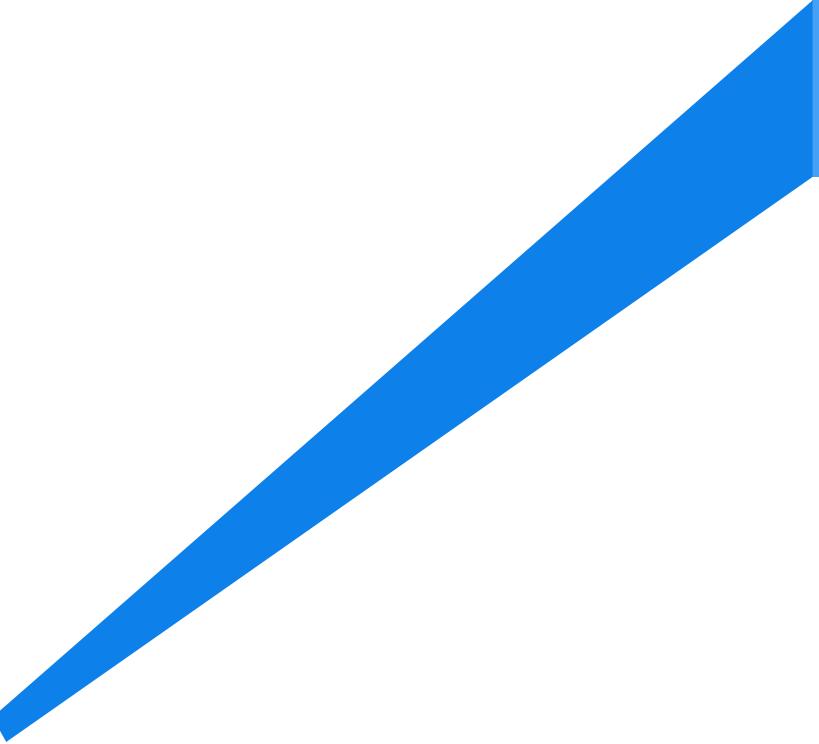
/\*

이건 여러 줄 주석이에요.  
여러 줄을 적을 수 있죠.

\*/

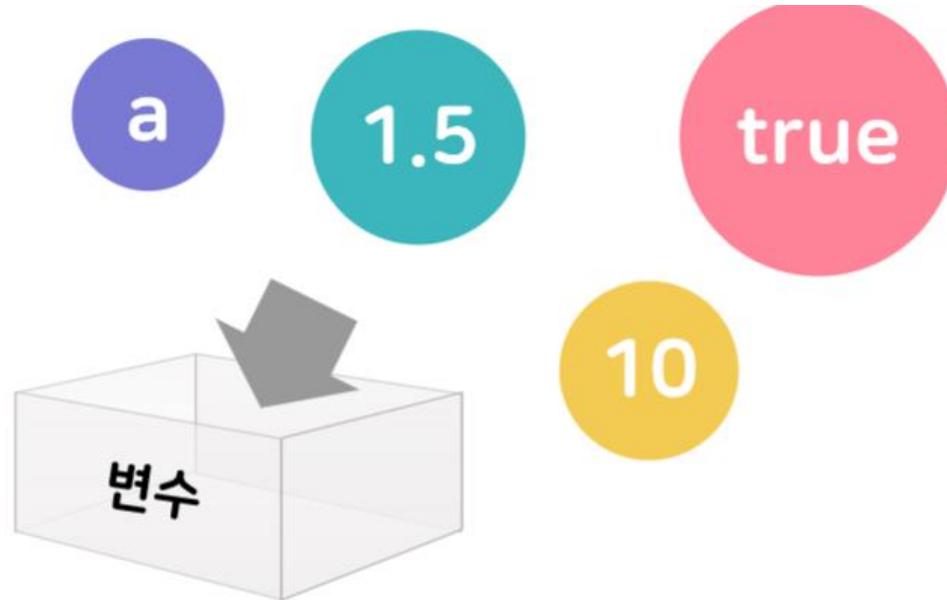
C++에서 주석을 사용하는 방법은 두 가지 방법이 있어요.

상황에 따라 적절하게 사용하면 돼요.



자료형

# 자료형



이제까지 변수에 정수만 담는 예시만 봤지만,  
**정수형** 외 변수에 **문자형**, **실수형**, 참, 거짓을 판단하는  
**논리형**까지 다양한 **자료형**을 넣을 수 있어요.

- 정수형(Integer Type)** : +,- 부호가 있고, 소수점이 없는 수의 자료형
- 실수형(Float Type)** : +,- 부호가 있고, 소수점이 있는 수의 자료형
- 문자형(Character Type)** : 문자 한 개를 표현할 수 있는 자료형
- 논리형(Boolean Type)** : 참(true), 거짓(false)을 표현할 수 있는 자료형

# | 정수 type

```
// 양수 음수 모두  
short a = -1;  
int b = 3;  
long c = -10;
```

```
// 양수만  
unsigned short a = 1;  
unsigned int b = 8;  
unsigned long c = 3;
```

+,- 부호가 있고, 소수점이 없는 수를 말해요.

정수형에는 **short, int, long**이 있고 **unsigned**를 붙이면  
더 큰 범위의 양수를 저장할 수 있어요.

# 정수 type

타입	범위
<b>short</b>	-32,768 ~ 32,767
<b>int</b>	-2,147,483,648 ~ 2,147,483,647
<b>long</b>	-2,147,483,648 ~ 2,147,483,647
<b>unsigned short</b>	0 ~ 65,535
<b>unsigned int</b>	0 ~ 4,294,967,295
<b>unsigned long</b>	0 ~ 4,294,967,295

각각의 정수형 타입들은 저장할 수 있는 **범위**가 있어요.

**변수 선언**을 할 때 **범위를 고려**해서 작성해야 해요.

# | 실수 type

```
float a = -1.345;  
double b = 3.12341;
```

실수형은 소수를 표현할 수 있고 정수보다 더 많은 수를  
표현할 수 있는 자료형이에요.

실수형으로는 **float**, **double** 두 종류가 있어요.

# | 실수 type

타입	범위
<b>float</b>	소수 6자리까지 표현
<b>double</b>	소수 15자리까지 표현

실수형 또한 표현할 수 있는 **범위가** 있어요.

실수형도 정수형처럼 **범위를 벗어난 수**를 입력하면

**오류**가 발생해요.

# 실수 type

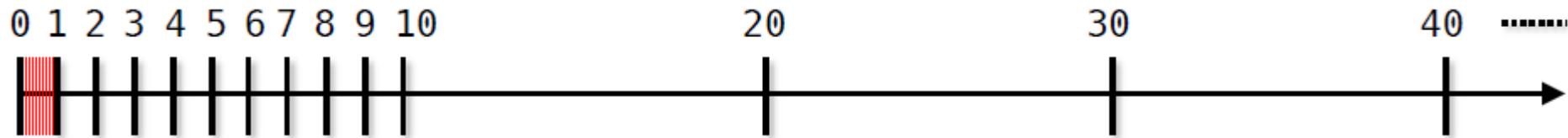
- 소수점 아래로 내려감 (ex : 1.31)

↑ 가수부  
↓ 지수부

- Rounding Error

```
>>> 1/3 + 1/3 + 1/3 + 1/3 + 1/3 + 1/3  
1.999999999999998
```

## Why?



# 문자형

```
char ch1 = 'a';  
char ch2 = '4';
```

**char형**은 따옴표(' ')를 사용해서 **한 개의 문자**를 저장해요.

위의 예시에서 'a', '4' 두 값은 실제로는 각각

**아스키코드**에 해당하는 **숫자**가 컴퓨터에 저장이 돼요.

# I boolean

```
bool flag1 = true;  
bool flag2 = false;
```

**논리형** 변수는 참(true)나 거짓(false) 중 한 가지 값만  
가질 수 있는 자료형이에요.

# I boolean

```
bool flag1 = true;  
bool flag2 = false;  
  
cout << "true:" << flag1;  
cout << "false:" << flag2;
```

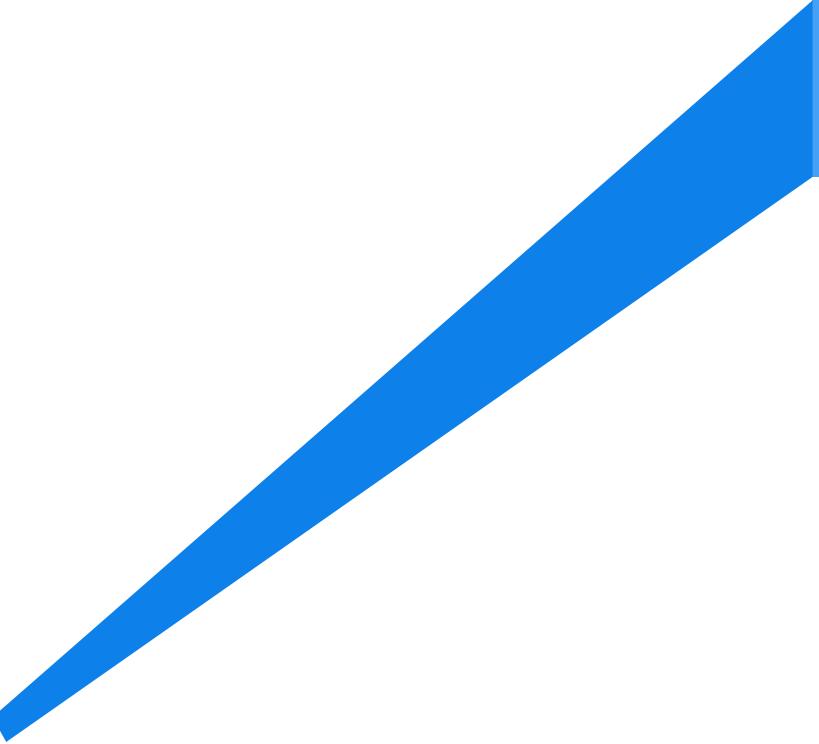
true로 저장하면  
기본은 1로 나와요.

true:1  
false:0

컴퓨터는 참, 거짓을 저장할 때 역시 숫자로 저장을 해요.

대부분의 프로그래밍 언어에서는 0은 거짓,

0이 아닌 모든 수는 참으로 간주해요.



연산자

# 연산자



연산자란 프로그램에서 더하기 빼기 등과 같은

연산을 처리하고 표현해주기 위한 기호예요.

연산을 받는 대상은 피연산자라고 해요.

# I 연산자

사칙 연산자	설명
+	덧셈
-	뺄셈
/	나눗셈
*	곱셈
%	나머지 연산*

**사칙 연산자**는 사칙 연산을 다루는 기본적인 연산자예요.

# 연산자

```
int a = 10, b = 5;  
  
int c;  
  
c = a + b; // 대입 c = 15
```

대입 연산자(=)을 사용해 사칙 연산자로 계산한 값을  
새로운 변수에 저장할 수도 있어요.

# 연산자

$$\text{num} = \boxed{2 + \boxed{6 / 2}}$$

$$\text{num} = \boxed{(2 + 6) / 2}$$

왼쪽 num에는 5가 저장되고, 오른쪽 num에는 4가 저장돼요.

사칙 연산자는 기본적인 사칙 연산의 우선순위 규칙을 따로요.

괄호()를 사용해서 우선순위를 정해줄 수 있어요.

# I 연산자

증감 연산자	설명
<code>++</code>	값을 1 증가
<code>--</code>	값을 1 감소

**증감 연산자**는 값을 1 증가 혹은 1 감소시켜주는 연산자예요.

**변수**의 앞에 또는 뒤에 붙여서 함께 사용해요.

# 연산자

전위 연산

② num2 = **++num1** ①

후위 연산

① num2 = **num1++** ②

증감 연산자의 위치는 연산 순서에 영향을 줘요.

전위 연산은 증감 연산자를 먼저 수행 후 연산을 하고,

후위 연산은 연산이 모두 끝난 후에 증감 연산자가 수행돼요.

# 연산자

```
int num1 = 5;  
int num2 = ++num1; //전위  
// num1 = 6, num2 = 6
```

증감 연산자를 변수 앞에 붙이는 **전위 연산자**는

num1의 **증감 연산자를 먼저 수행**하고,

num2에 **대입**을 해서 둘 다 6이 저장돼요.

# 연산자

```
int num1 = 5;  
int num2 = num1++; //후위  
// num1 = 6, num2 = 5
```

증감 연산자를 변수 뒤에 붙이는 **후위 연산자는**

num2에 **대입 연산자를 먼저 수행**하고

**증감 연산자를 수행**해서 num1에는 6, num2에는 5가 저장돼요.

## ■ 논리연산자

3이랑  $1 + 2$  는 같아요      ->      참(true)

7이랑  $5 + 5$  는 같아요      ->      거짓(false)

**논리 연산자**는 주어진 논리식을 판단해

**참(true)**과 **거짓(false)**을 결정해주는 연산자예요.

# ■ 논리연산자

논리 연산자	연산자 이름	설명
&&	AND 연산	논리식이 모두 참이면 참을 반환
	OR 연산	논리식 중 하나라도 참이면 참을 반환
!	NOT 연산	논리식의 참 거짓을 뒤집는 연산

# 논리연산자

&&

AND 연산자

입력		결과
0	0	0
1	0	0
0	1	0
1	1	1

AND 연산은 모두 참일 때만 참을 반환해주는 연산이에요.

# 논리연산자

||

OR 연산자

입력		결과
0	0	0
1	0	1
0	1	1
1	1	1

OR 연산은 둘 중 하나라도 참이면 참을 반환해주는 연산이에요.

# 논리연산자

!

NOT 연산자

입력	결과
0	1
1	0

NOT 연산은 참과 거짓을 반대로 뒤집어주는 연산이에요.

# I 논리연산자

```
bool a = 1 && 1; // a = true  
bool b = 1 && 0; // b = false  
bool c = 1 || 0; // c = true  
bool d = :0;      // d = true
```

# 비교연산자

비교 연산자	설명
<code>==</code>	왼쪽, 오른쪽이 같으면 참
<code>!=</code>	왼쪽, 오른쪽이 같지 않으면 참
<code>&gt;</code>	왼쪽이 오른쪽보다 크면 참
<code>&gt;=</code>	왼쪽이 오른쪽보다 크거나 같으면 참
<code>&lt;</code>	오른쪽이 왼쪽보다 크면 참
<code>&lt;=</code>	오른쪽이 왼쪽보다 크거나 같으면 참

비교 연산자는 피연산자 사이의 크기를 비교하는 연산자에요.

값의 크기를 비교해서 참, 거짓을 반환해요.

# 비교연산자

```
int num1 = 8;  
  
int num2 = 20;    비교 연산식  
  
cout << (num1 == num2);  
cout << (num1 <= num2);
```

```
0 // false  
1 // true
```

예시처럼 **두 수를 비교**하는 식을 만들어서  
참과 거짓을 반환하게 할 수 있어요.



배열

# 배열

따로따로 변수 선언해주기 너무 많아요!

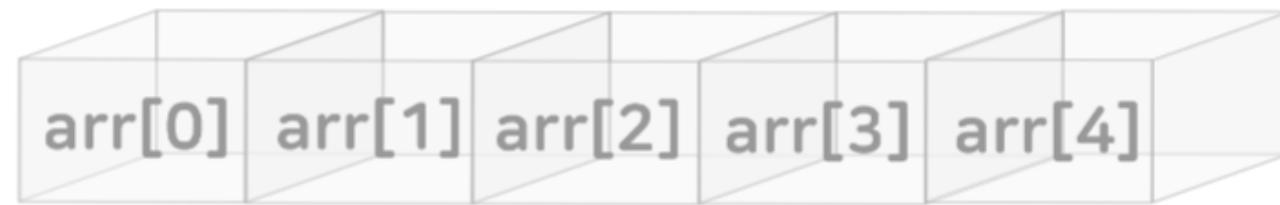


이 문제는 **배열(array)**이라는 것을 사용한다면  
매우 쉽게 **많은 양의 값**을 **저장**할 수 있어요.

같은 자료형끼리 집합!



int형 배열



배열은 같은 자료형의 변수들로 이루어진 집합을 말해요.

# 배열

배열의 원소예요. 배열 선언을 할 때  
어떤 자료형이 들어가는지 정해줘야 해요.



배열은 원소와 인덱스로 이루어져 있어요.

배열 안의 값 을 원소, 배열의 순서 를 나타내 주는 것이 인덱스예요.

# 배열

```
int arr[4];          // int형 4개  
float arr2[10];    // float형 10개  
char arr3[50];     // char형 50개
```

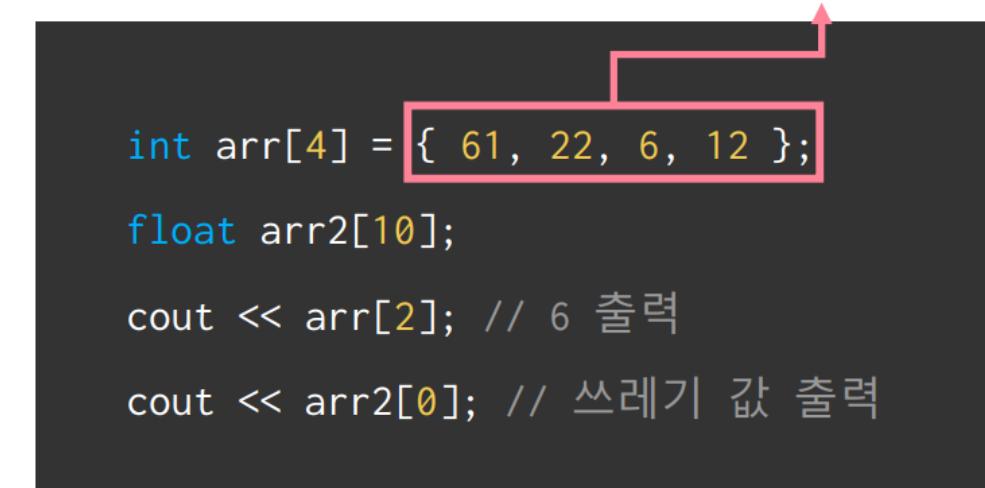
배열 선언의 다양한 예시들이예요.

배열은 대괄호[] 안에 정해준 크기만큼 공간이 생성돼요.

# 배열

배열의 크기만큼 초기화해주었어요.

```
int arr[4] = { 61, 22, 6, 12 };
float arr2[10];
cout << arr[2]; // 6 출력
cout << arr2[0]; // 쓰레기 값 출력
```



배열도 **초기화**를 하지 않으면 **쓰레기 값**이 생겨요.

배열을 초기화하는 방법은 **중괄호{}**를 통해서

배열의 개수만큼 초기화할 수 있어요.

# 배열

배열의 인덱스는 0부터!  
헷갈리면 안 돼요!!

```
int arr[4] = { 61, 22, 6, 12 };
cout << arr[0]; // 61
cout << arr[3]; // 6
cout << arr[4]; // 오류!!
```

배열에 들어있는 원소에 접근하려면  
몇 번째 인덱스인지 적어주어야 해요.  
단, 배열 크기를 벗어난 인덱스를 사용하면 오류가 발생해요.

# 배열

```
int arr[4] = { 61, 22, 6, 12 };

arr[1] = 0;

arr[2] = 0;      변경 되었어요!

// 배열 arr = {61, 0, 0, 12}
```

배열의 원소를 접근해서 해당 배열의 원소를  
변경해 줄 수도 있어요.

# 배열 특징

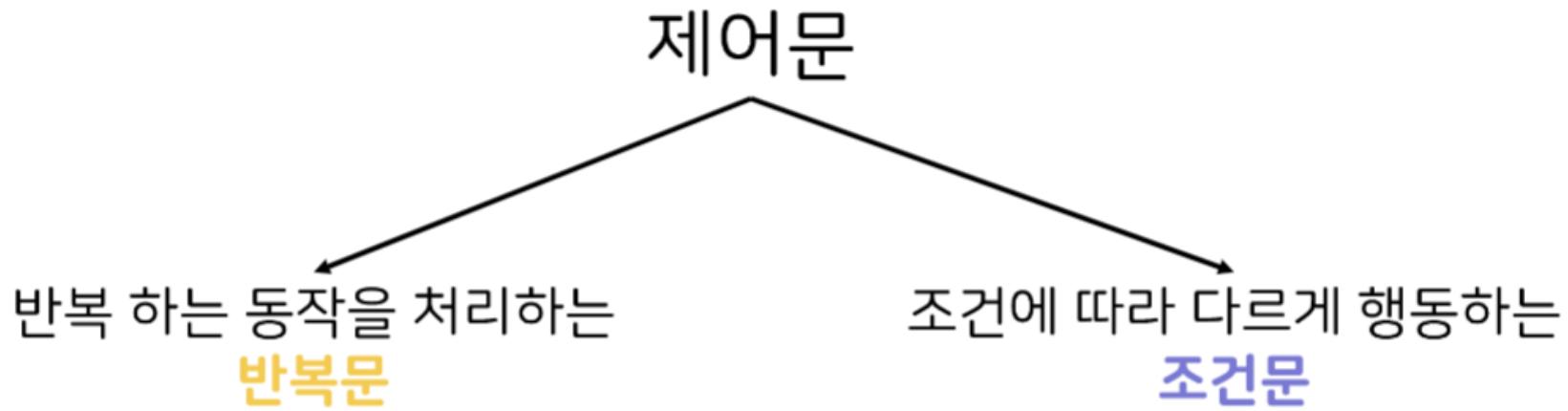
배열의 크기는 반드시 0 이상의 상수!

배열 인덱스는 0부터 시작!

배열 크기를 벗어난 인덱스를 사용하면 오류 발생!

# 조건문

# | 조건문



제어문을 사용하면 이렇게 여러 상황에 대해 처리할 수 있어요.

우리는 그중 조건문을 배워볼 거예요.

# 조건문

## if 문

```
if(/* a가 1이라면? */) {  
    // 행동  
}
```

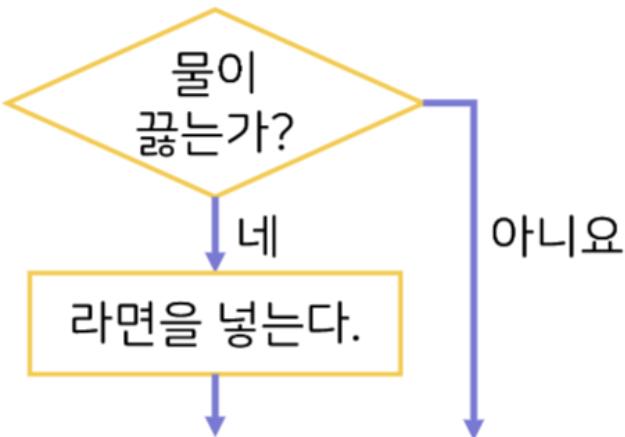
## switch 문

```
switch(a) {  
    // a가 1이라면? :  
    // 행동  
}
```

C++에서 제공하는 조건문에는 크게  
다양한 형태의 if 문과 switch 문 두 가지가 있어요.

# 조건문

## if 문 순서도

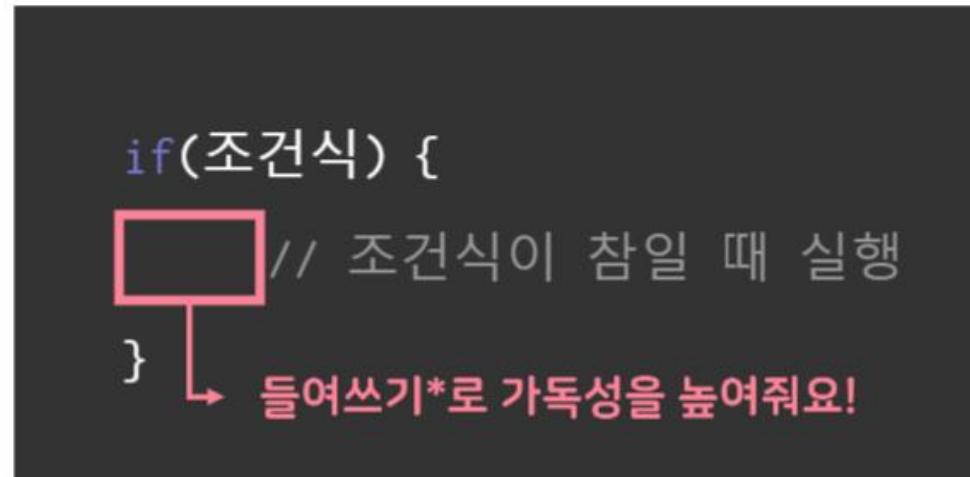


다음 순서도는 **가장 기본적인 if 문**의 순서도예요.

**주어진 조건**(물이 끓는다)가 **참**이면 라면을 넣고

**거짓**이면 아무것도 수행하지 않아요.

# | 조건문



**if** 문은 **중괄호{}**를 사용해서 블록을 만들어서  
조건식의 결과가 **참**일 때 실행하고 싶은 명령을 적어줘요.

# | 조건문

```
int a = 50;
if(a > 80) {           거짓
    cout << "a는 80 이상.";
}
cout << "프로그램 종료.";
```

프로그램 종료.

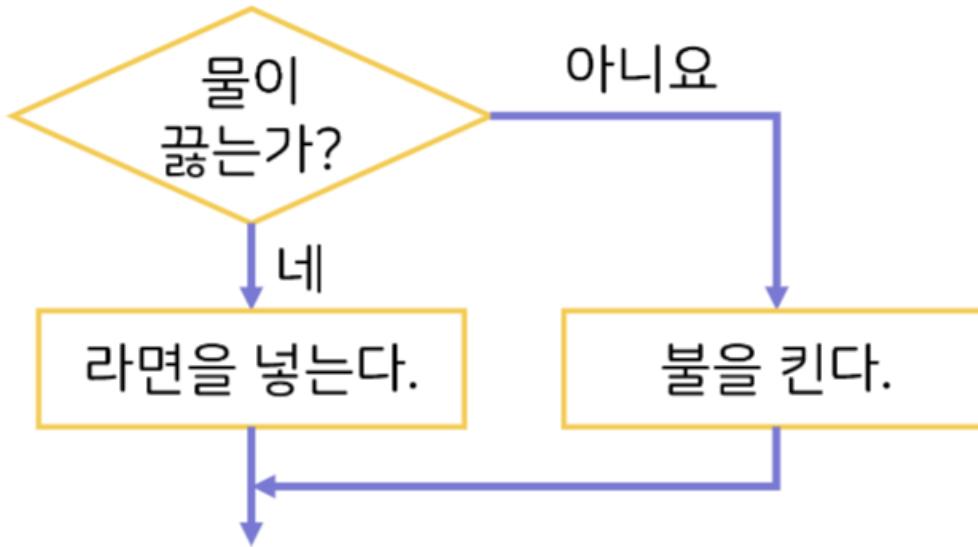
위의 조건식은 a가 80보다 크니? 라고 묻고 있어요.

a는 50이기 때문에 거짓을 반환해줘요.

그래서 if 문 안에 명령은 수행하지 않고 넘어가요.

# 조건문

else문  
순서도



else 문의 순서도예요.

if 문이 거짓일 때 하는 행동을 따로 추가해 줘요.

# | 조건문

```
if(조건식) {  
    // 조건식이 참일 때 실행  
}  
  
else {  
    // 조건식이 거짓일 때 실행  
}
```

**else 문**을 if 문 뒤에 추가해서 **거짓일 때 따로 수행하는 명령**을 적어줄 수 있어요.

# 조건문

```
int a = 50;  
if(a > 80) {—————거짓  
    cout << “a는 80 이상.”;  
}  
  
else {—————  
    cout << “a는 80 이하.”; ←  
}  
  
cout << “프로그램 종료.”;
```



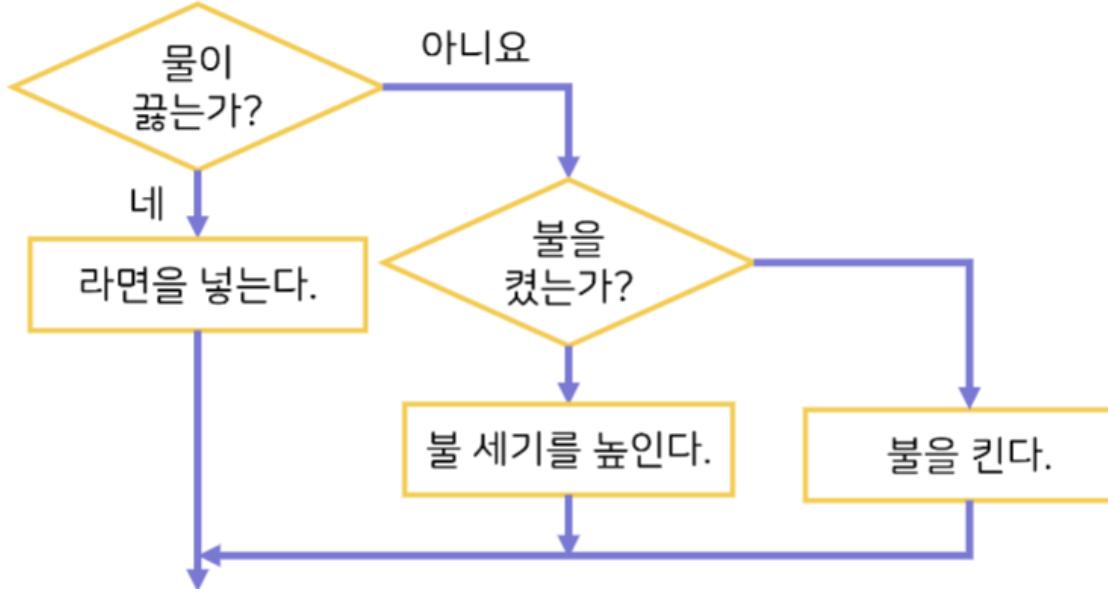
```
a는 80 이하.  
프로그램 종료.
```

조건식은 거짓이기 때문에 else 문 안의 명령을 수행해요.

그리고 else 문을 벗어나서 나머지 명령을 수행하죠.

# 조건문

else if 문  
순서도



else if 문의 순서도예요.

이 조건문을 사용해서 더 복잡한 조건들을 처리할 수 있죠.

# 조건문

else 문은 필수로  
써주지 않아도 괜찮아요!  
단, 있으면 한 번만 쓸 수 있어요.

```
if(조건식) {  
    // 조건식이 참일 때 실행  
}  
  
else if(조건식2) {  
    // 조건식2가 참일 때 실행  
}  
  
else {  
    // 모든 조건식에 대해 거짓일 때 실행  
}
```

else if 문을 통해 조건식을 더 추가해줄 수 있어요.

여러 개의 조건을 추가해서 더 복잡한 조건을 처리할 수 있어요.

# 조건문

```
int a = 50;  
if(a > 80) {  
    cout << "a는 80 이상.";  
}  
    참  
else if(a == 50) {  
    cout << "a는 50 이에요!";  
}
```

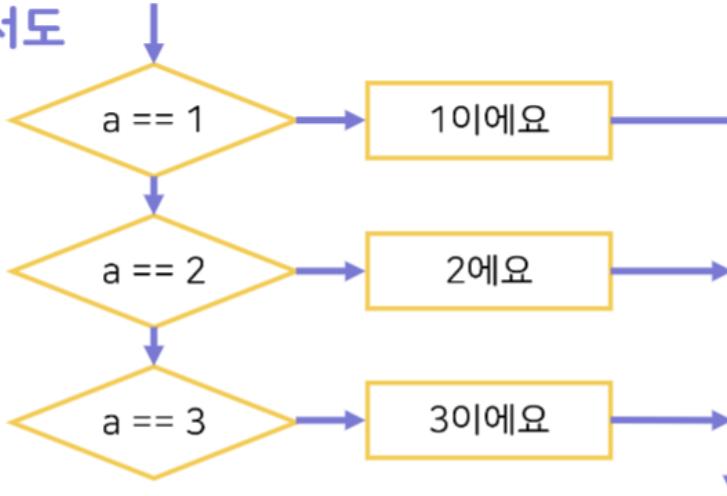


```
a는 50 이에요!
```

조건식 `a == 50`이 참이기 때문에 **else if 문**안의  
명령문을 실행해요.

# 조건문

switch 문 순서도



switch 문도 조건에 따라 다른 명령을 수행하는데

사용하는 조건문 이예요.

하는 일은 else 문 또는 else if 문과 비슷하게 동작해요.

# | 조건문

```
if(a == 1) {  
}  
  
else if(a == 2) {  
}  
  
else if(a == 3) {  
}
```

=

```
switch(a) {  
    case 1:  
        break;  
    case 2:  
        break;  
    case 3:  
        break;  
}
```

코드가 한눈에  
들어오지 않나요?

차이점이 있다면 **if 문**보다 **가독성이 좋아요.**

또 컴퓨터 내에서 **빠르게 동작**해요.

# 조건문

```
switch(조건 값(변수)) {  
    case 값1:  
        // 조건 값이 값1일 때 실행  
        break;  
    case 값2: case 끝에는 콜론(:)을 붙여줘야 해요.  
        // 조건 값이 값2일 때 실행  
        break;  
    ..  
}
```

**switch 문**은 괄호() 안의 **변수와 동일한 값을 갖는 case**로 가서 명령문을 실행해 줘요.

# 조건문

```
switch(조건 값(변수)) {  
    case 값1: break; break가 없어서 다음으로 넘어가요.  
    case 값2: break; // 조건 값이 값1 또는 값2 일 때 실행  
        break; 멈춰!  
    case 값3:  
    ...  
}
```

이때 case 마지막에 붙이는 **break**는 조건 검사를 멈추는 역할을 해요.

**break**가 없으면 **다음 케이스로 넘어가서**

원하지 않는 명령이 실행될 수 있어 주의해야 해요.

# | 조건문

```
switch(조건 값(변수)) {  
    case 값1:  
        break;  
    case 값2:  
        break;  
    default:  
        // 아무 case에도 속하지 않을 때 실행  
}
```

**default** 절은 조건 값이 아무 **case**에도 속하지 않을 때 실행이 돼요.

불필요하다면 작성하지 않아도 돼요.

# | 조건문

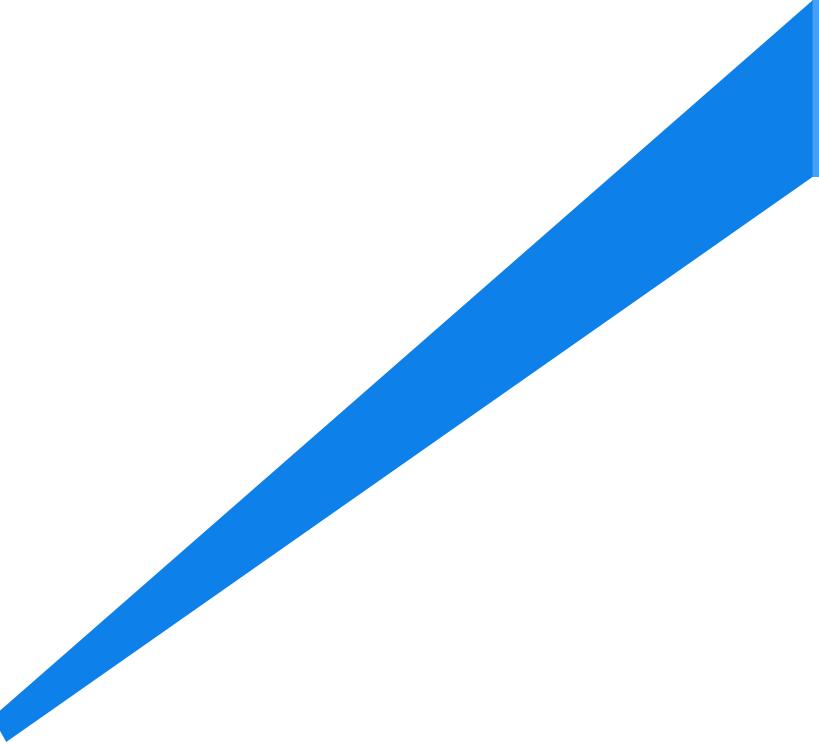
```
int a = 20;  
switch(a) {  
    case 10:  
        cout << "a는 10!";  
        break;  
    case 20:  
        cout << "a는 20!";  
        break;  
    case 30:  
        cout << "a는 30!";  
        break;  
}
```



```
a는 20!
```

**switch 문**을 사용한 예시예요.

상황에 따라서 **if 문**과 **switch 문**을 적절하게 사용하면 멋진 코드를 만들 수 있답니다!



반복문

# 반복문 : while

## while 문

```
while(조건식) {  
    // 조건식이 참일 때 실행  
}
```

## for 문

```
for(초기식; 조건식; 증감식) {  
    // 조건식이 참일 때 실행  
}
```

C++에서 **반복문**의 종류는 while 문과 for 문 두 가지가 있어요.  
이번 장에서는 **while 문**을 배워 볼 거예요.

# 반복문 : while

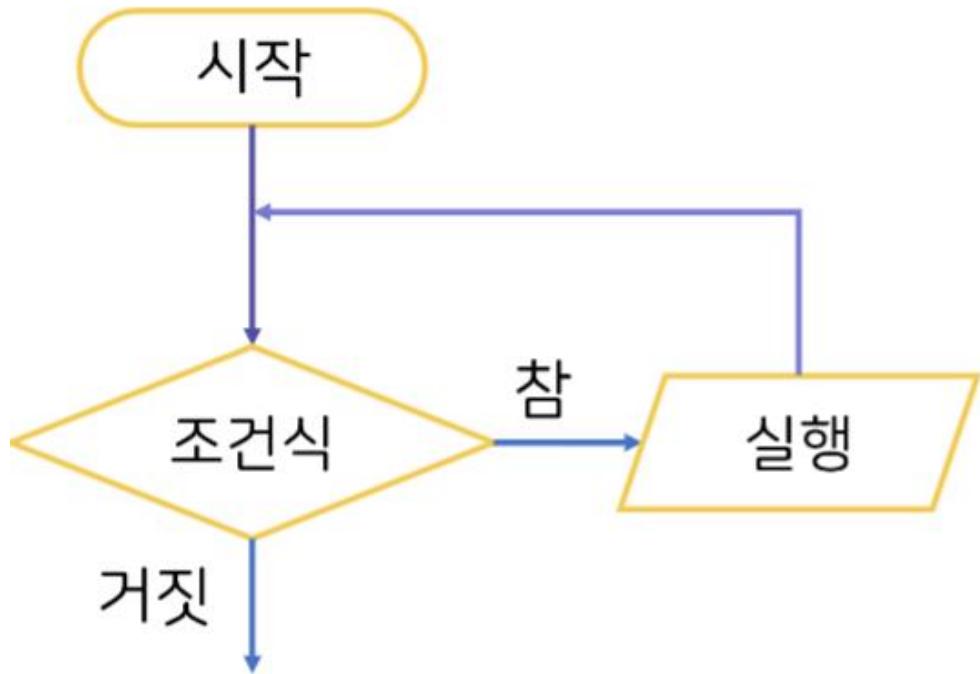
C++에서 while 문은 다음과 같이 사용해요.

while 문 안의 명령이  
한 줄일 때 중괄호를  
생략할 수 있어요.

```
while(조건식) {  
    // 행동  
    // 행동  
}
```

괄호 안의 조건식이 참(true)인 동안 내용을 반복하고  
거짓(false)일 때 반복을 멈춰요.

# 반복문 : while



```
while(조건식)  
    // 실행
```

while 문을 시작하면 바로 조건식을 확인한 후  
조건의 참, 거짓에 따라 행동을 할지 말지 결정을 해요.

# 반복문 : while

```
int i = 0;  
while(i < 5) {      // 조건식  
    cout << i << " ";  
    i++;            // i 값을 1 증가  
}
```



```
0 1 2 3 4
```

위 예시에서 **조건식**은  $i < 5$  이에요.

$i$ 의 값이 0부터 4일 때까지 참(true)니까 **반복**되요.

위에 예시는 0 1 2 3 4가 출력이 되고 **while 문**을 종료하게 되겠죠?

# 반복문 : while

```
while(guess != answer) {      // 조건식
    cout << "실패!";
    cin >> guess;           // 답 입력
}
cout << "성공!";
```

위의 예시에서는 예측(guess)이 정답(answer)과 다를 때 조건식이 참이라 정답을 맞히면 while 문을 종료하는 예시예요.

## 반복문 : while

```
while(1) {  
    // 참일 때 실행  
}
```

만약 위 예시처럼 **조건식**이 **항상 참**이면 어떻게 될까요?

# 반복문 : while

맞아요! 반복문 안을 무한히 돌면서 동작을 해요.

우리는 이것을 무한 루프(**Infinite Loop**)라고 부르기로 약속했어요.



그렇다면 이 무한 루프를 멈추게 혹은 계속하게  
제어할 수 있는 방법은 없을까요?

# 반복문 : while

## break

```
while(조건식) {  
    if(조건식)  
        break;  
}
```

## continue

```
while(조건식) {  
    if(조건식)  
        continue;  
}
```

반복문을 동작 단계에서 제어할 수 있는 방법은

반복문을 멈추는 **break** 문과 현시점에서 중지하고

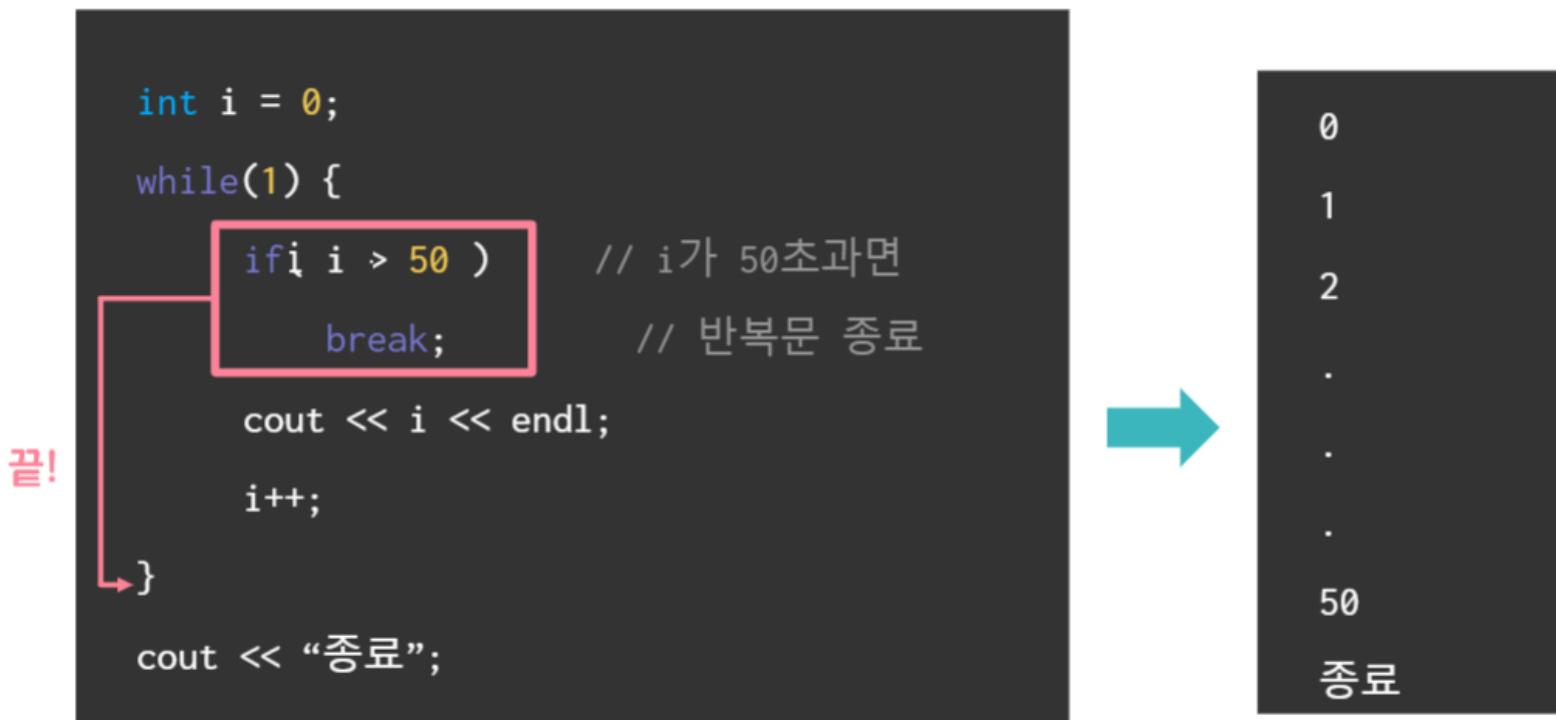
다시 반복문의 처음으로 돌아가는 **continue** 문이 있어요.

# 반복문 : while

```
while(조건식) {  
    if(조건식)  
        break; // 종료  
    끝!  
} // 반복문 종료 후 실행
```

먼저 **반복문**을 끝내는 **break 문**을 사용하면  
반복문이 끝나는 바로 **다음 명령어**를 실행해요.

# 반복문 : while



다음 `while` 문은 50에서 한 번 더 돌면 51이 돼요.

그럼 if 문이 참이 되어 `break` 문에서 `while` 문을 종료하게 되죠.

# 반복문 : while

무시하고 계속!

```
while(조건식) {  
    if(조건식)  
        continue;  
    // 다음 명령 무시  
}  
// 다음 명령
```

continue 문은 해당 명령이 내려지면, 반복문 안에 있는  
continue 문 이후의 명령을 무시하고 다음 조건식 판단으로 넘어가요.

# 반복문 : while

```
int i = 0;  
  
while(i < 51) {  
    if(i == 30) { // i가 30이면  
        i++;  
        continue; // 다음 반복  
    }  
    cout << i;  
    i++;  
}
```



```
28  
29  
// 30은 없어요!  
31  
32  
. .  
50
```

변수 i가 30일 때 if 문이 참이 되고 **continue** 문을 만나게 돼요.  
따라서, 출력문이 동작하지 않고 바로 **다음으로 넘어가요**.

# 반복문 : while

1. 반복문 while 안 조건식에 항상 참이 들어간다면 어떻게 될까요?
2. 반복문 실행 중에 break 문을 만나면 어떻게 될까요?
3. 반복문 실행 중에 continue 문을 만나게 되면 어떻게 될까요? ↘

# 반복문 : for

## while 문

```
while(조건식) {  
    // 조건식이 참일 때 실행  
}
```

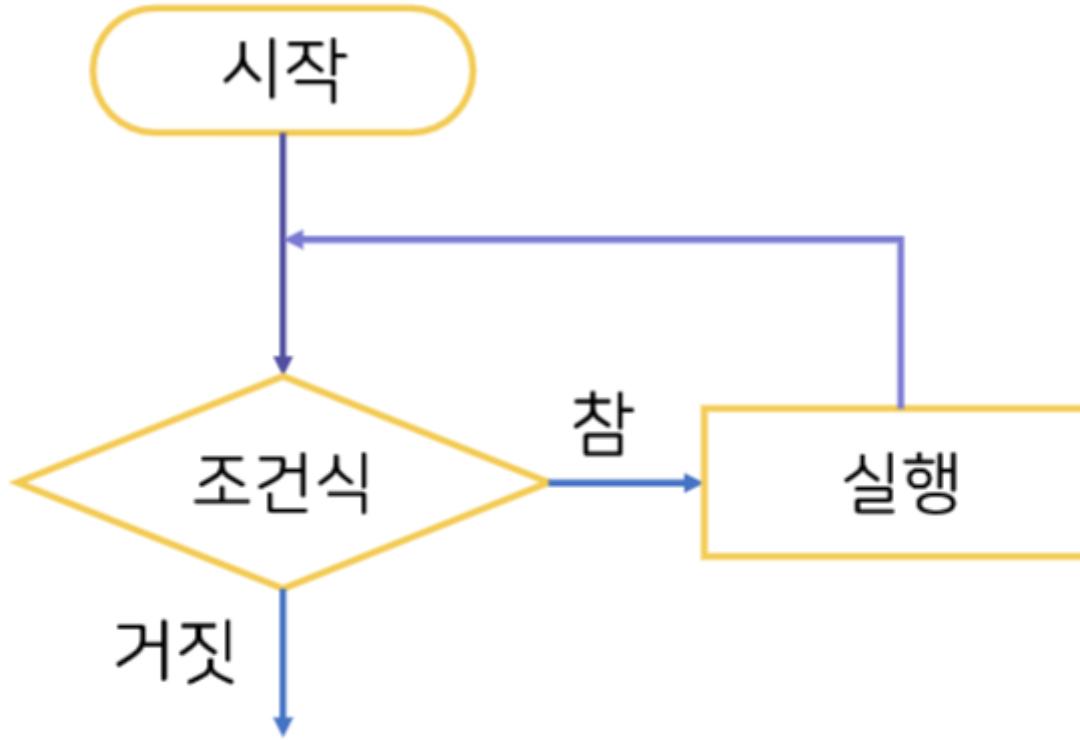
## for 문

```
for(초기식; 조건식; 증감식) {  
    // 조건식이 참일 때 실행  
}
```

지난 시간에는 반복문 중에 **while 문**을 배웠어요.

이번 시간에는 다른 반복문인 **for 문**에 대해서 알아봐요.

## 반복문 : for



for 문도 while 문처럼 반복을 위해 사용하는 명령문이에요.

# 반복문 : for

```
세미 콜론으로 식을 구분해줘요!  
for(초기식; 조건식; 증감식) {  
    // 조건식이 참일 때 실행  
}
```

**for 문**은 다음과 같은 **초기식**, **조건식**, **증감식** 세 개의 요소로 구성되어 있어요.

# 반복문 : for

초기식

```
for(int i = 0; i <= 5; i++) {
```

// 조건식이 참일 때 실행

```
}
```

초기식은 **for 문 처음 시작시** 변수를 초기화해주는 식을 적어줘요.

단, 초기화해준 변수는 **지역 변수라 for 문 밖**에서 사용하면 안 돼요!

# 반복문 : for

```
조건식  
for(int i = 0; i <= 5; i++) {  
    // 조건식이 참일 때 실행  
}
```

**for 문의 조건식**은 while 문의 조건식이랑 똑같아요.

**매 반복마다 조건식을 확인**해 참, 거짓으로 반복을  
계속할지 말지 결정해주는 역할을 해요.

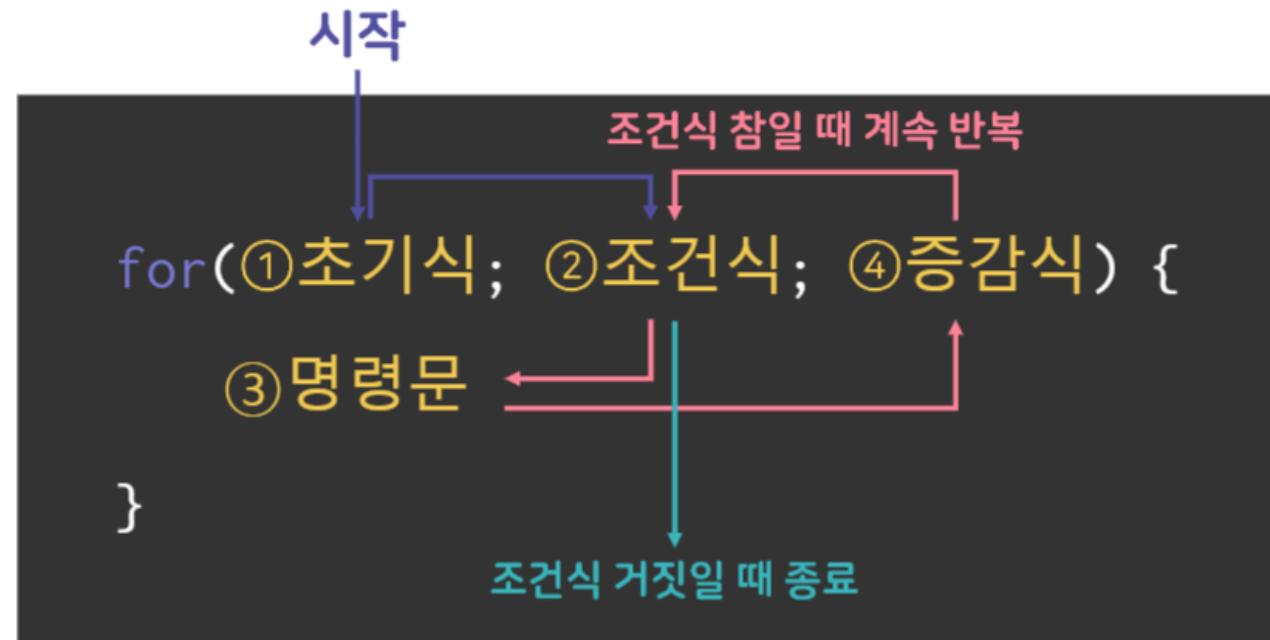
# 반복문 : for

```
증감식  
for(int i = 0; i <= 5; i++) {  
    // 조건식이 참일 때 실행  
}
```

for 문이 매 반복을 끝내고 조건식을 확인하기 전

반복적으로 값을 계산해주고 싶으면 증감식 위치에  
연산식을 적어줘요.

# 반복문 : for



화살표를 통해 **for 문**이 어떻게 동작하는지

시각적으로 한눈에 알아봐요!

# 반복문 : for

```
for(;;) {  
    // 무한 루프  
}
```

초기식, 조건식, 증감식은 필수로 있어야 하는 것은 아니에요.

이렇게 아무것도 적지 않았을 때는 무한 루프에 빠져요.

# I while vs for

## while 문

```
while(조건식) {  
    // 조건식이 참일 때 실행  
}
```

## for 문

```
for(초기식; 조건식; 증감식) {  
    // 조건식이 참일 때 실행  
}
```

우리는 이로써 C++에서 다루는 **반복문**을 모두 배웠어요.

그렇다면 마지막으로 이 두 반복문의 **차이점**을 알아볼까요?

# I while vs for

써주어야 할 것이 더 많아요!

```
for(초기식; 조건식; 증감식) {  
    // 조건식이 참일 때 실행  
}
```

**for 문의 가장 큰 특징은 자체적으로  
초기식, 조건식, 증감식을 가지고 있다는 점이에요.**

# I while vs for

아하! 0부터 5까지  
반복하는구나!



```
for(i = 0; i <= 5; i++) {  
    // 조건식이 참일 때 실행  
}
```

**for 문**은 이러한 특징 덕분에 **특정 정수 범위를 반복**할 때  
초기식, 조건식, 증감식을 써서 **매우 직관적**으로 사용할 수 있어요.

# I while vs for

```
while(조건식) {  
    // 조건식이 참일 때 실행  
}
```

반면 **while** 문은 **조건식만** 써주고  
그 조건식의 결과가 거짓이 될 때 반복문을 나올 수 있었죠.

# I while vs for

정답을 맞힐 때까지  
해야 되는구나!



```
while(guess != answer) {  
    // 조건식이 참일 때 실행  
}
```

**while** 문 같은 경우는 범위보다는  
원하는 조건이 나올 때까지 반복하는 것에 유용하게 사용해요.

# I while vs for

## while 문

```
while(조건식) {  
    // 조건식이 참일 때 실행  
}
```

## for 문

```
for(초기식; 조건식; 증감식) {  
    // 조건식이 참일 때 실행  
}
```

이제 반복문의 종류에 대해 완벽하게 알게 되었어요.

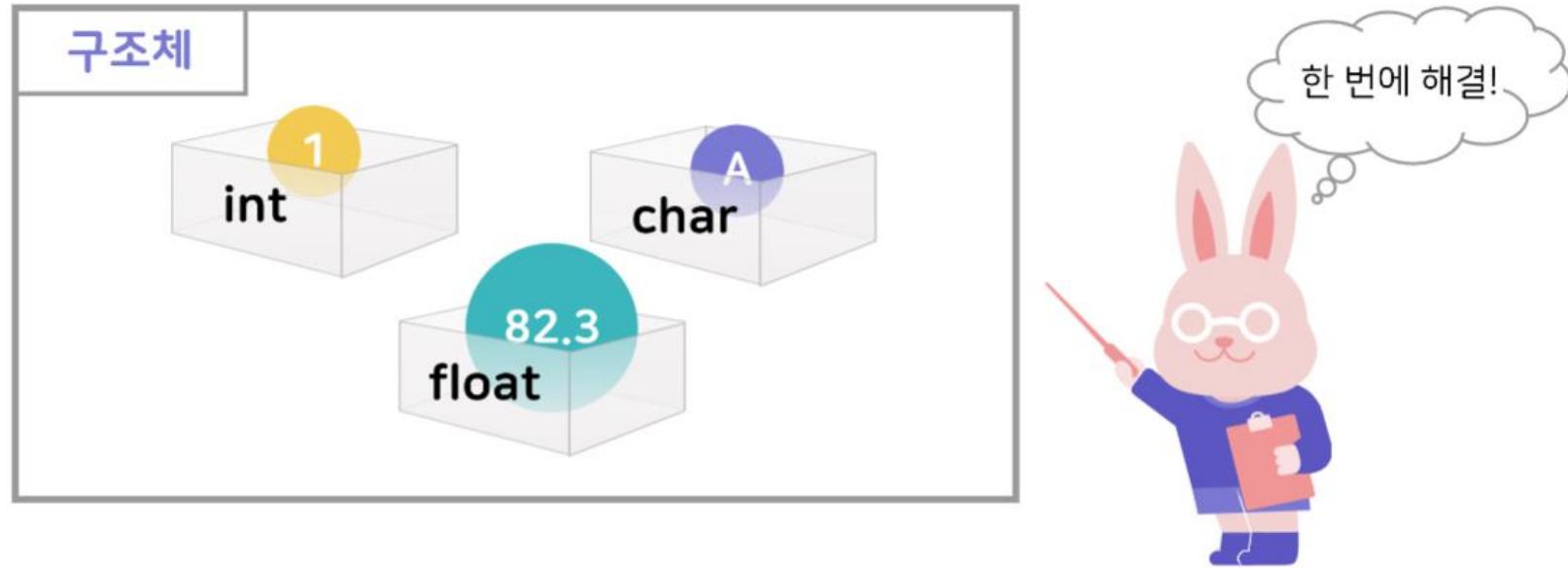
두 개의 반복문을 **적재적소에 사용**하도록 해보세요!

# I while vs for

1. for 문을 구성하는 요소들이 어떤 것이 있을까요?
2. 어떤 상황에 for 문을 사용하는 것이 편할까요?
3. for 문을 while 문으로 변경할 수 있나요?

# 구조체

# 구조체



구조체를 사용하면 **여러 자료형을 함께 저장할 수 있어요.**

**구조체**는 배열보다 더 복잡한 값들을 **한 번에 정의**해 줄 수 있는  
특징이 있답니다.

# 구조체

변수처럼 구조체 이름을 지어줘요.

```
struct Student{  
    int number; // 번호  
    float grade; // 성적  
    char name[20]; // 이름  
};
```

구조체는 **struct 키워드**를 이용해서 정의할 수 있어요.

struct 키워드와 지어줄 **구조체 이름**을 함께 적고

**중괄호{}** 안에 구조체를 구성하는 **변수**들을 적어줘요.

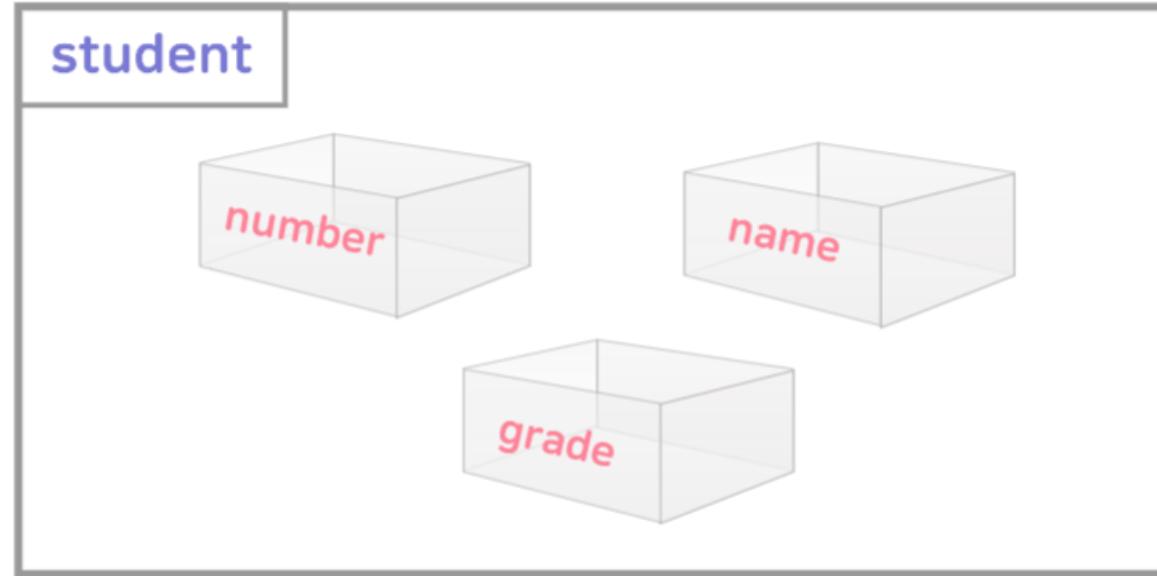
# 구조체

멤버 변수

```
struct Student {  
    int number;      // 번호  
    float grade;    // 성적  
    char name[20];  // 이름  
};
```

구조체를 구성하는 **변수**들은 구조체의 **멤버**라서  
**멤버 변수**라고 해요.

# 구조체



이제 **나만의 자료형**인 `student` 구조체를 만들었으니까,  
변수를 선언하고, 초기화해봐야겠죠?

# 구조체

```
struct Student {  
    int number;  
    float grade;  
    char name[20];  
} st1; // 선언
```

구조체 정의와 **동시에 선언**

```
struct Student {  
    ...  
};  
int main(){  
    (struct) Student st1; // 선언  
}
```

기본 자료형처럼 **별도로 선언**  
(struct 키워드 생략 가능)

구조체 변수를 **선언하는 방법**은 두 가지가 있어요.

선언했다면 역시 **초기화**도 해야겠죠?

# 구조체

```
struct Student {  
    int number;  
    float grade;  
    char name[20];  
}; // 구조체 선언
```

```
int main() {  
    Student st1 = { 1, 80.4, "길동"}  
} // 구조체 초기화
```

구조체의 **초기화**는 배열과 비슷하게 **중괄호{}**를 사용해요.

구조체의 멤버 변수 순서에 맞게 **초기화**할 수 있어요.

(구조체 변수 이름) . (멤버 변수 이름)



이 구조체 "의" 멤버 변수에 접근한다! 라고 생각하면 쉬워요.

이제 구조체를 사용해야겠죠?

구조체의 멤버 변수에 접근할 때에는

점(.) 연산자를 이용해서 접근할 수 있어요.

# 구조체

```
int main() {  
    Student st1 = { 1, 30.4, "길동"}  
    cout << st1.number; // 1  
    cout << st1.grade; // 30.4  
    cout << st1.name; // 길동  
}
```

Student 구조체 각각의 **멤버 변수 number, grade, name**을 사용하려면 이렇게 **점 연산자**를 사용해 변수를 이용할 수 있어요.

# 구조체

```
int main() {  
    Student st1 = { 1, 80.4, “길동”}  
    cout << st1.grade; // 80.4  
  
    st1.grade = 74.8;  
    cout << st1.grade; // 74.8  
}
```

변경 되었어요!

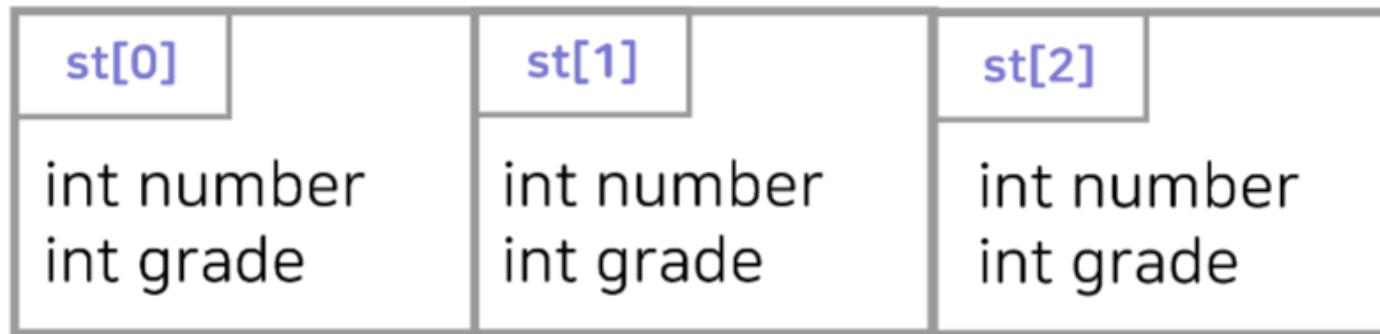
구조체 멤버 변수의 값을 변경할 수도 있어요.

# 구조체

```
int main() {  
    Student st[3]; // 구조체 배열  
    st[0].number = 20; // 첫 번째 학생의 번호  
    st[0].grade = 74.8; // 첫 번째 학생의 성적  
    st[0].name = “길동”; // 첫 번째 학생의 성적  
}
```

구조체로도 배열을 만들 수 있어요!

# 구조체



구조체 배열을 생성하면

같은 구조체들이 여러 개 생기는 것이죠.

# 구조체

```
int main() {  
    Student st[2] = {{1, 80.4, "길동"}, {2, 90, "철수"}};  
    cout << st[0].grade; // 80.4  
    cout << st[1].grade; // 90  
}
```

구조체 배열을 초기화 해주려면

중괄호{} 안에 배열 개수만큼 중괄호{}로 초기화 해줘야 해요.

# 구조체

```
int main() {  
  
    Student st[3];  
  
    st[0].number = 20; // 첫 번째 학생의 번호  
    st[0].grade = 74.8; // 첫 번째 학생의 성적  
  
}
```

구조체 배열 원소의 멤버 변수들도 역시  
저장, 변경이 가능해요.



**THANK YOU**