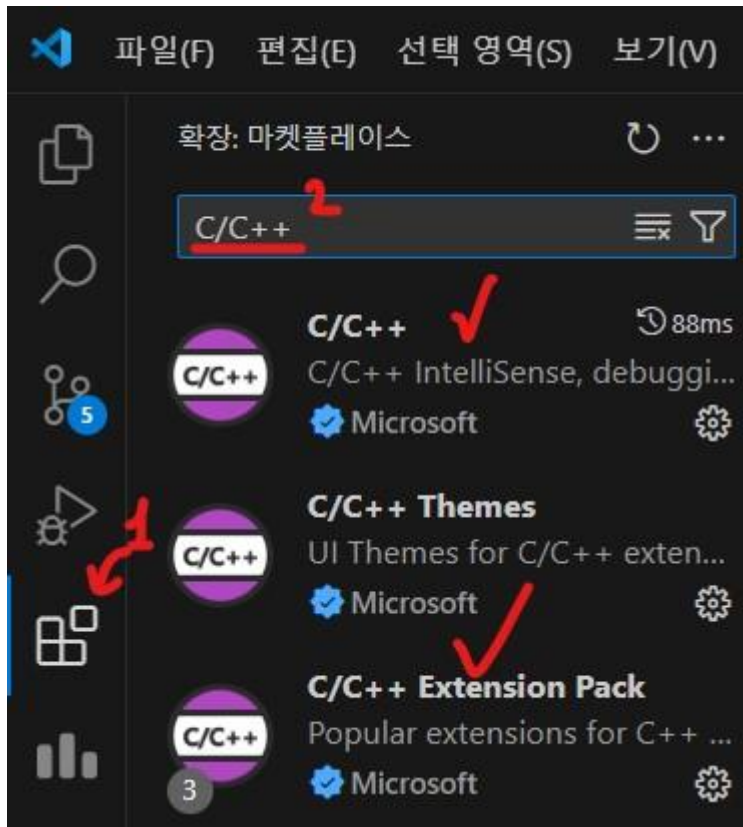


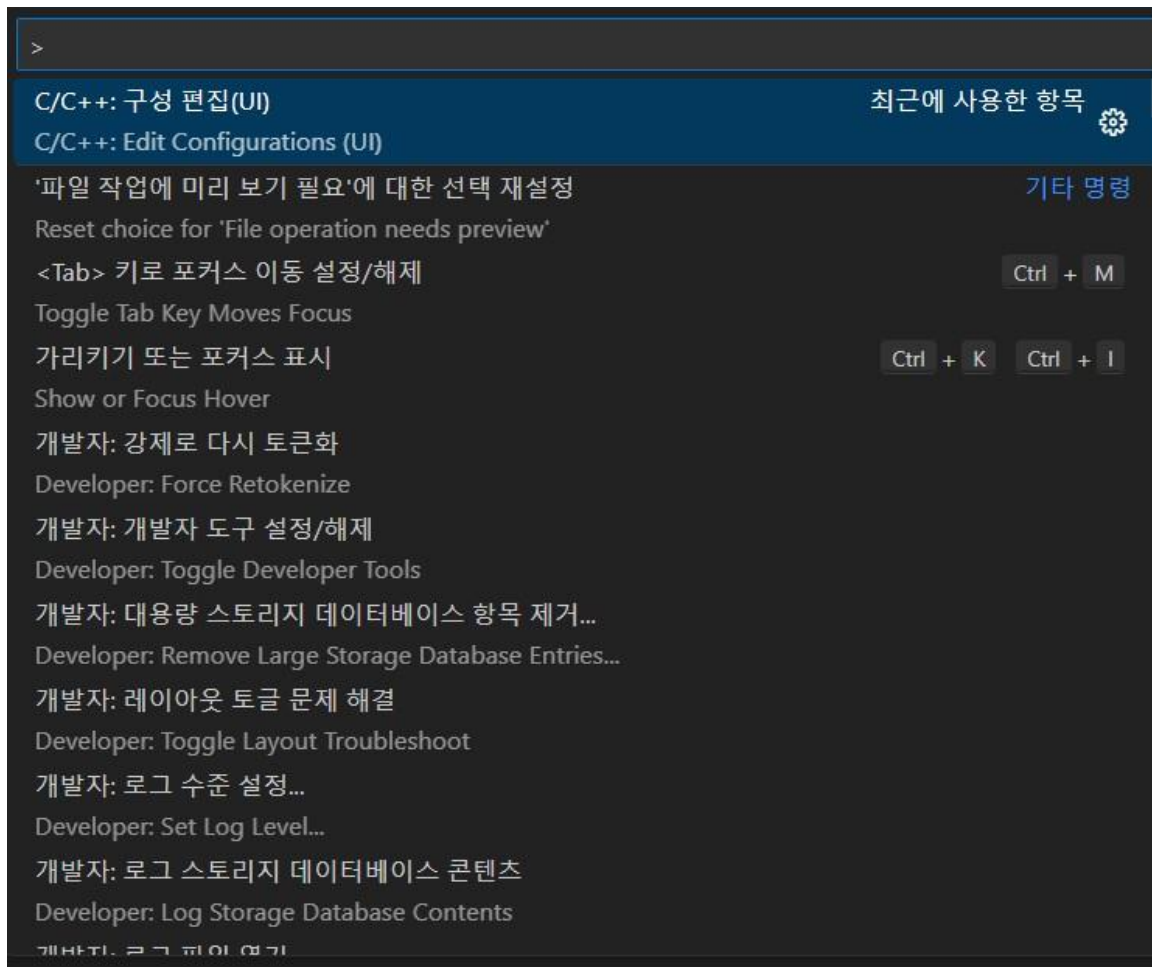
## 익스텐션 설치

C++을 실행하기 위해서는 추가로 익스텐션을 설치해야한다. 우선 설치해야 하는 익스텐션은 C/C++과 C/C++익스텐션 팩이다. 익스텐션에서 'C/C++'을 검색해 C/C++과 C/C++익스텐션 팩을 설치하자.



## 구성(Configuration)설정 및 테스트 환경 설정

익스텐션 설치가 완료되었다면, 빈 폴더를 생성하고 생성한 폴더를 연다. 그리고 F1을 눌러 구성 편집 UI(C/C++: Edit configurations (UI))를 선택하고, 아래와 같이 값을 변경한다.



F1 누르면 나오는 화면

### 구성 이름

구성을 식별하는 이름입니다. `Linux`, `Mac` 및 `Win32` 은(는) 해당 플랫폼에서 자동으로 선택되는 구성의 특수 식별자입니다.

편집할 구성 세트를 선택합니다.

Win32 ▼

구성 추가

### 컴파일러 경로

더 정확한 IntelliSense를 사용하도록 설정하는 데 사용되는, 프로젝트를 빌드하는 데 사용하는 컴파일러의 전체 경로입니다(예: `/usr/bin/gcc`). 확장에서는 컴파일러에 쿼리하여 IntelliSense에 사용할 시스템 포함 경로 및 기본 정의를 확인합니다.

컴파일러 경로를 지정하거나 드롭다운 목록에서 검색된 컴파일러 경로를 선택합니다.

C:/mingw64/bin/g++.exe ▼

### 컴파일러 인수

사용된 포함 또는 정의를 수정하기 위한 컴파일러 인수입니다. `-nostdinc++`, `-m32` 등. 공백으로 구분된 추가 인수를 사용하는 인수는 배열에 별도의 인수로 입력해야 합니다(예: `--sysroot <arg>`의 경우 `"--sysroot"`, `"<arg>"` 을(를) 사용하세요).

줄당 하나의 인수입니다.

### IntelliSense 모드

MSVC, gcc 또는 Clang의 플랫폼 및 아키텍처 변형에 매핑되는 사용할 IntelliSense 모드입니다. 설정되지 않거나 `$(default)` (으)로 설정된 경우 확장에서 해당 플랫폼의 기본값을 선택합니다. Windows의 경우 기본값인 `windows-msvc-x64` (으)로 설정되고, Linux의 경우 기본값인 `linux-gcc-x64` (으)로 설정되며, macOS의 경우 기본값인 `macos-clang-x64` (으)로 설정됩니다. `$(default)` 모드를 재정의하려면 특정 IntelliSense 모드를 선택합니다. `<compiler>-<architecture>` 변형(예: `gcc-x64`)만 지정하는 IntelliSense 모드는 레거시 모드이며 호스트 플랫폼에 따라 `<platform>-<compiler>-<architecture>` 변형으로 자동으로 변환됩니다.

windows-gcc-x64 ▼

구성 편집 화면

변경하는 값은 컴파일러 경로를 `C:/mingw64/bin/g++.exe` 로, `intelliSense` 는 `windows-gcc-x64` 로 변경해주면 된다.



구성 편집을 끝냈다면, 여기서 테스트 파일(1000.cpp)을 생성해 아래 코드를 추가한다.

```
#include <iostream>
```

```
using namespace std;
```

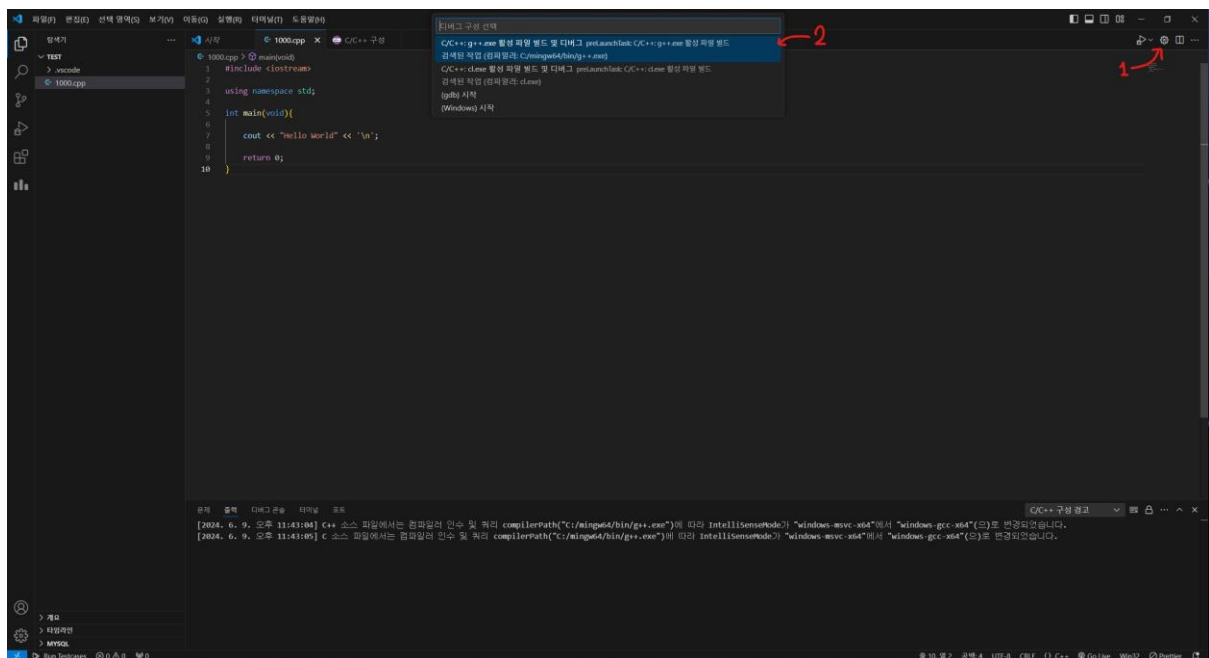
```
int main(void){
```

```
cout << "Hello World" << '\n';
```

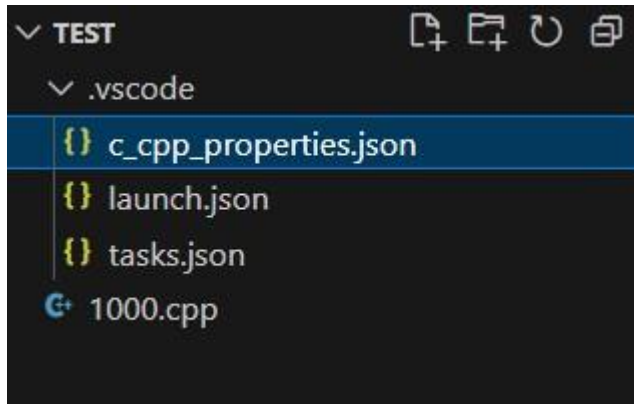
```
return 0;
```

```
}
```

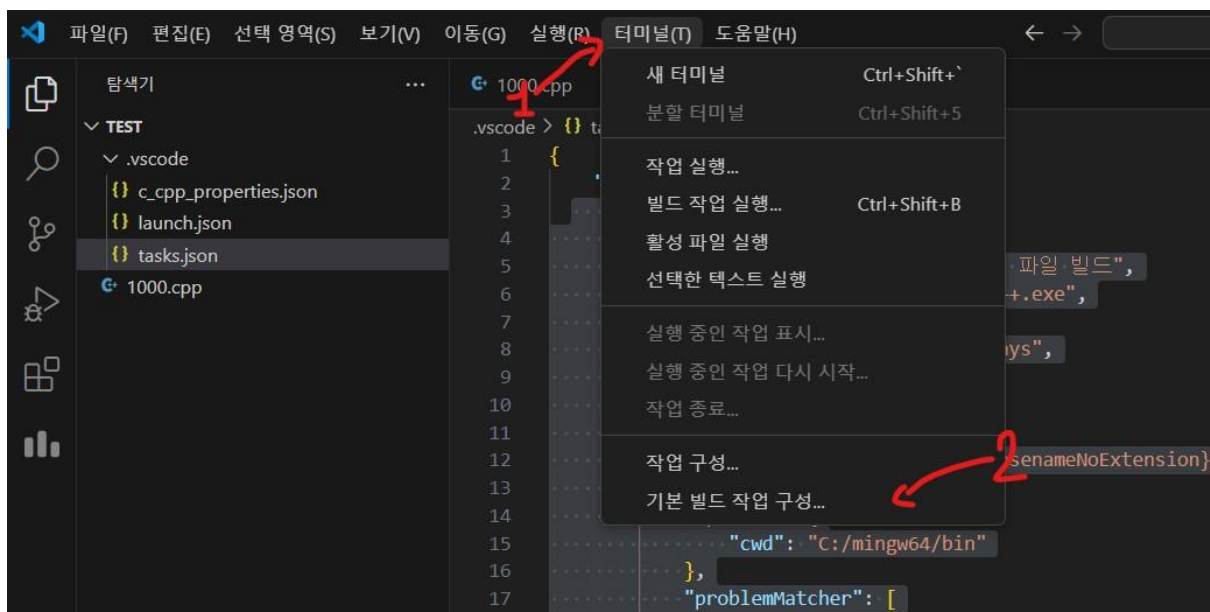
그런 다음, 테스트파일을 클릭 -> VSCode 오른쪽 상단에 위치해있는 톱니바퀴를 클릭 -> C/C++: g++.exe 활성 파일 및 디버거를 클릭한다.



이러면 아래와 같이 .vscode 폴더가 생성되면서 c\_cpp\_properties.json, launch.json, tasks.json 이 생성된다.



여기서 tasks.json 을 클릭 후 상단 바에 있는 터미널 -> 기본 빌드 작업 구성 -> C/C++: g++.exe 활성 파일 빌드를 선택한다.(이 작업을 해야 코드를 실행할 때마다 gcc, g++중 뭐로 빌드할건지 물어보지 않는다.)




이로써 모든 세팅은 준비가 되었다!

---

## 코드 실행(Run) 및 디버그(Debug) 테스트

이제 테스트만 하면 된다. 미리 만들어놓은 테스트 파일(**1000.cpp**)을 클릭하고 **F5** 를 누른다. 아래와 같이 **Hello World** 가 출력되었다면 정상적으로 실행되는 것이다.



```
PS C:\Users\zamps\OneDrive\바탕 화면\test> g++ 1000.cpp -std=c++11 -o 1000.exe
PS C:\Users\zamps\OneDrive\바탕 화면\test> .\1000.exe
Hello world
PS C:\Users\zamps\OneDrive\바탕 화면\test>
```

**F5** 가 정상적으로 실행되었다는 것은 실행/디버그 둘 다 정상 작동한다는 것이다. 하지만 보다 확실하게 디버그가 되는 지 알고싶기 때문에 디버그 테스트 파일을 하나 생성해서 **breakpoint** 가 작동하는 지 알아보자.(참고로, **F5** 는 디버그 단축키이며, 실행버튼은 VSCode 오른쪽 상단에 톱니바퀴 옆 재생버튼을 누르면 된다. 벌레 붙은 재생 버튼은 **F5** 와 같이 디버그를 실행한다.)