OpenGL 경로 지정 + 코딩의 팁

~ 컴퓨터 그래픽스를 시작하는 꿀팁 ~

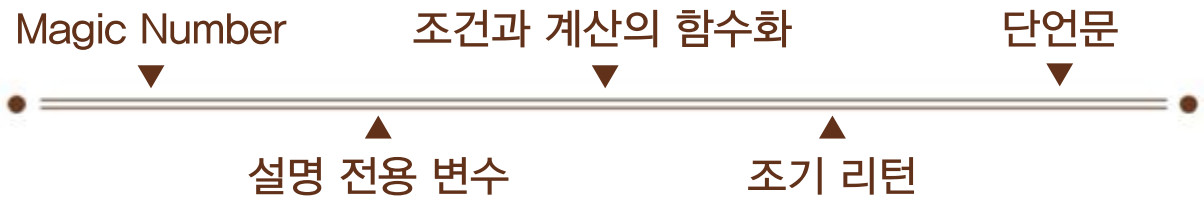
한국산업기술대학교
게임공학과

김도율

CONTENTS

1. OpenGL 경로 설정

2. 코딩의 팁



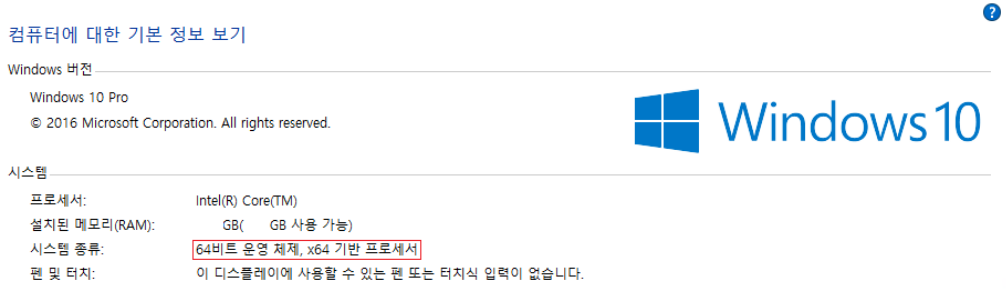
Bonus. 헤더파일에 넣어야 하는 것

Bonus. Visual Studio의 유용한 단축키

OpenGL 경로 지정

1. OpenGL 경로 지정하는 방법

1. glut_3.7.zip의 내용들을 적절한 곳에 압축 해제한다 :
   - ① glut.h
     - VSPATH\VC\include\gl\ 에 압축 해제
   - ② glut.lib & glut32.lib
     - VSPATH\VC\lib\ 에 압축 해제
   - ③ glut.dll & glut32.dll
     - [x86] C:\Windows\System32\ 에 압축 해제
     - [x64] C:\Windows\SysWOW64\ 에 압축 해제



준비물

1. glut_3.7.zip (https://www.opengl.org/ 에서 다운로드)
2. Visual Studio 설치 경로(이하 VSPATH)
   - ➢ 다음 페이지에서 설명

glut_3.7.zip의 내용물

1. glut32.lib
2. glut.lib
3. glut32.dll
4. glut.lib
5. glut.h

[참고]
배포할 때 실행 파일과 함께 glut.dll 과 glut32.dll 을 꼭 첨부하도록 한다.

1. VSPATH

[VS2015]

- C:\Program Files (x86)
  \Microsoft Visual Studio 14.0

[VS2017]

- C:\Program Files (x86)
  \Microsoft Visual Studio
  \2017
  \Community
  \VC
  \Tools
  \MSVC
  \14.11.25503

주의사항

1. VS2017의 맨 마지막 경로 14.11.25503은 버전 번호로 바뀔 수 있다.

1. NuGet으로 OpenGL 설정하기

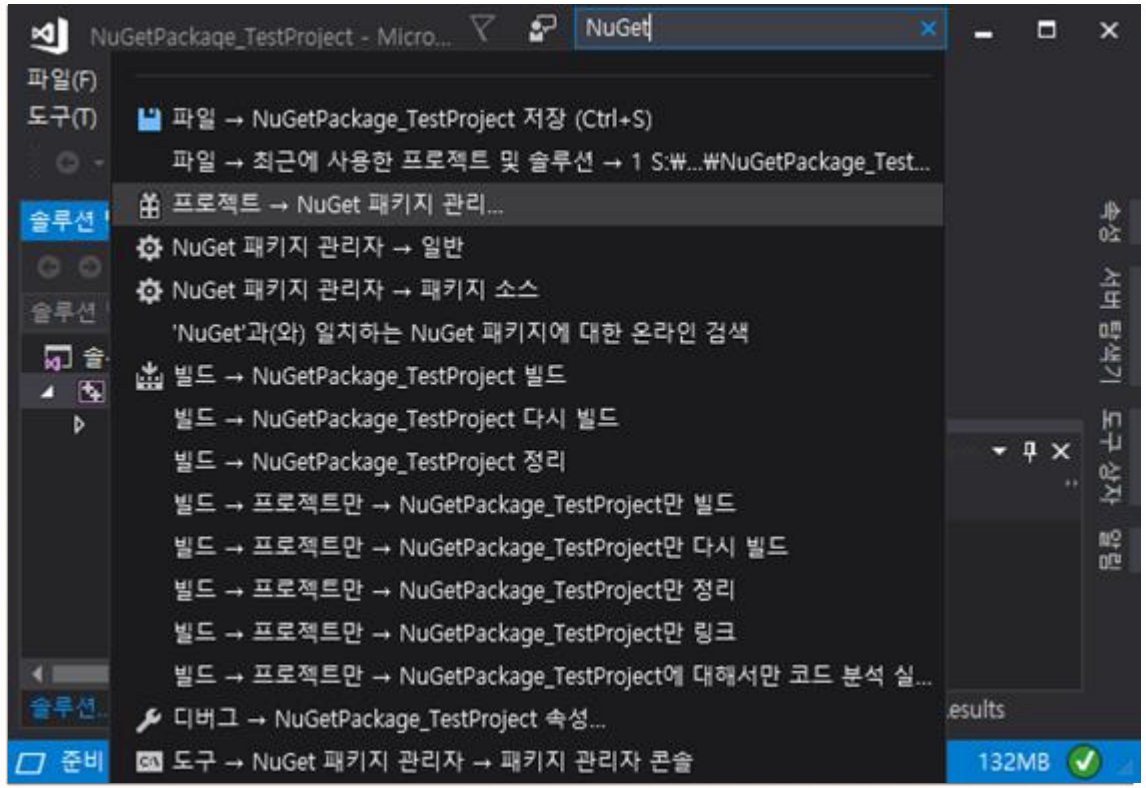
- NuGet?
  - NuGet은 프로젝트 솔루션에 라이브러리를 통합 하는 과정을 간소화 하는 오픈 소스 패키지 관리 시스템입니다. | MSDN
  - 간단히 말해 솔루션 단위로 한 번에 라이브러리를 설치해주는 플러그인
- 장점
  - 클릭 몇 번으로 라이브러리 설치 가능
  - 다른 솔루션에 영향 주지 않음(설치 꼬임 문제 ×)
- 단점
  - 솔루션마다 라이브러리의 용량이 포함됨
  - 새 솔루션을 만들 때마다 설치 필요

준비물

1. Visual Studio 2013 또는 그 이상의 버전

[참고]
Visual Studio 2017은 설치가 필요할 수 있음

1.
NuGet으로 OpenGL 설정하기



방법

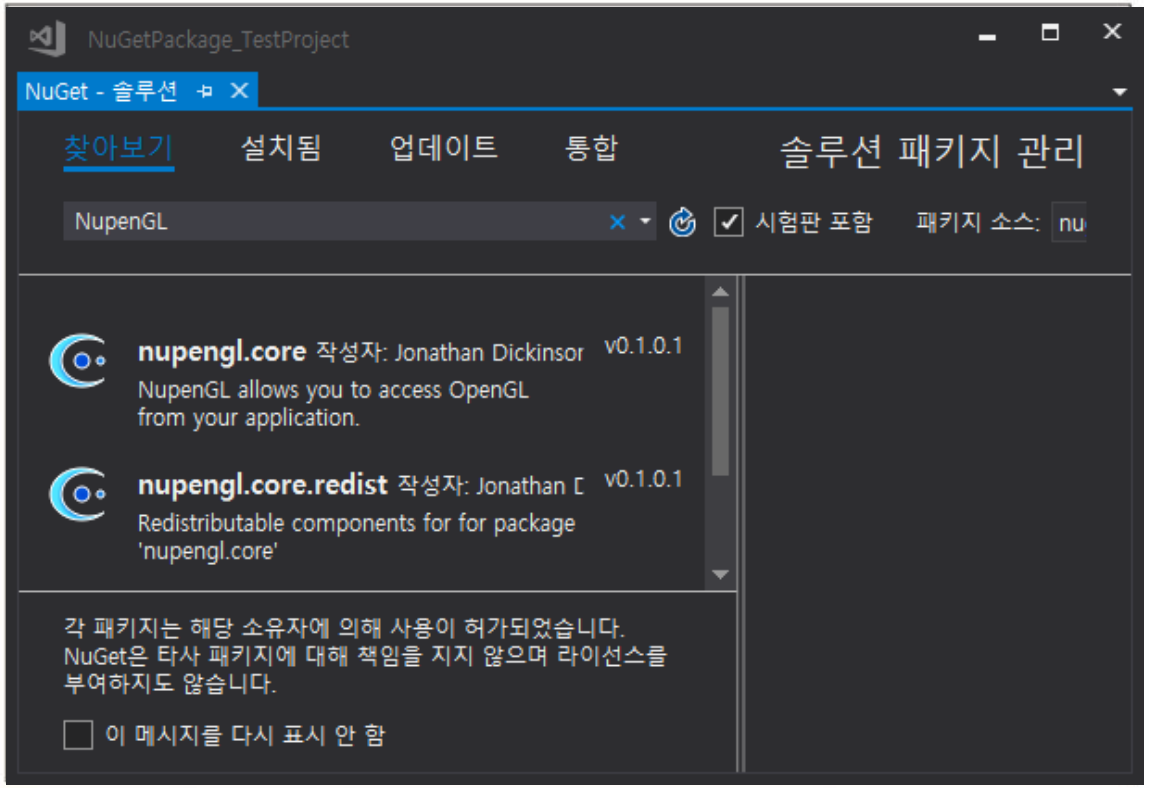
1. 우측 상단의 검색 바에서 NuGet 검색
2. 프로젝트 → NuGet 패키지 관리… 선택

또 다른 방법

1. 도구 → NuGet 패키지 관리자(N) →
   솔루션용 NuGet 패키지 관리… 선택

[참고]
Visual Studio 2017은 설치가 필요할 수 있음

1. NuGet으로 OpenGL 설정하기



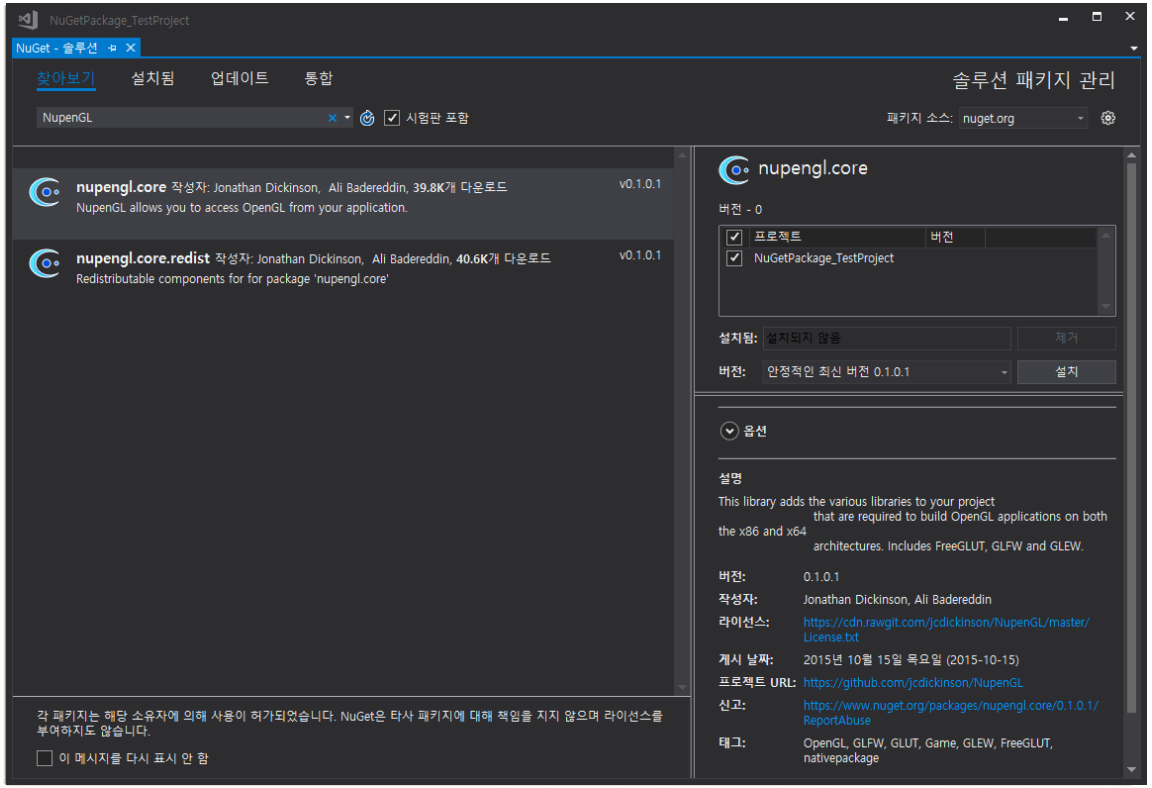
방법

1. [NuGet 솔루션 패키지 관리] 창에서 찾아보기 → nupengl 검색
2. nupengl.core 설치

nupengl

1. NuGet에서 설치 가능한 OpenGL 라이브러리.
2. ~~.redist는 redistribution(재배포)이며 nupengl.core 설치 시 같이 설치됨

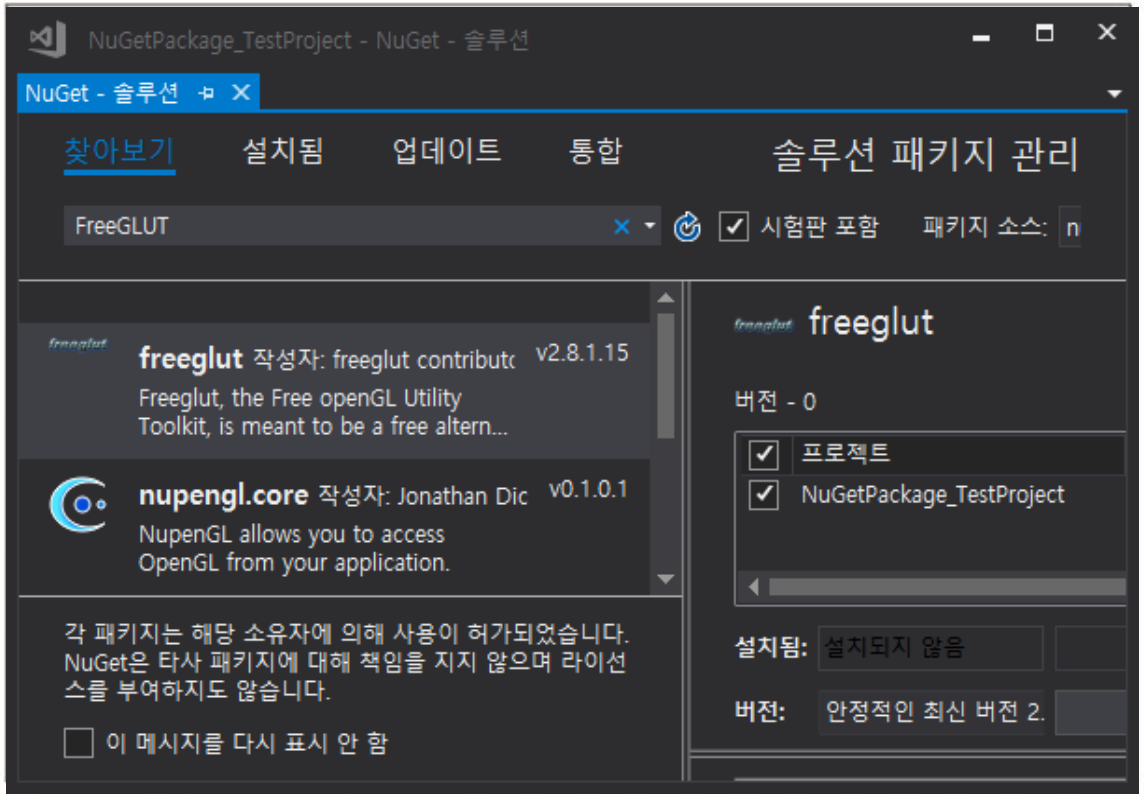
1. NuGet으로 OpenGL 설정하기



방법

1. 우측의 설치 클릭

1. NuGet으로 OpenGL 설정하기



방법

1. FreeGLUT도 동일하게 설치
2. 용량 약 40MB

코딩의 팁

2. 코딩의 팁

Magic Number

- **[Magic Number]** 변수에 대입하지 않고 사용하는 숫자
- iMazeWidth 와 iMazeHeight 는 미로의 크기이다.

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

constexpr int iMazeWidth = 10;
constexpr int iMazeHeight = 10;

int main(){
    int Maze[iMazeWidth][iMazeHeight];

    for(int x = 0; x < iMazeWidth; ++x)
        for (int y = 0; y < iMazeHeight; ++y)
        {
            // 초기화
            Maze[x][y] = 0;
        }
    // TODO : ...
}
```

반복할 때 숫자 대신 상수를 사용하면 추후 수정이 용이하다.

사용할 때

1. 숫자를 적는 거의 모든 곳
2. 의미를 가진 숫자

효과

1. 가독성이 좋아진다.
2. 개발 중 변경이 **매우** 용이하다.
3. 변수명을 잘 지을 경우 주석을 볼 필요도 없다.
4. const 변수 또는 #define 변수를 사용하여 상수처럼 사용할 수 있다.

[참고]

constexpr | 변수가 아닌 확실한 상수.　　※ const | 변수를 인자로 한 함수에서 const 속성을 부여할 수 있다.

2. 코딩의 팁

Magic Number

- MazeState는 미로의 각 블록이 어떤 종류인지 알려준다.

```
enum class MazeState {
      NONE  // 지나갈 수 있다.
    , WALL  // 지나갈 수 없다.
    , DOOR  // 다음 스테이지로 갈 수 있다.
};

int main(){
    MazeState Maze[iMazeWidth][iMazeHeight];

    for(int x = 0; x < iMazeWidth; ++x)
        for (int y = 0; y < iMazeHeight; ++y)
        {
            // 초기화
            Maze[x][y] = MazeState::NONE;
        }
    // TODO : ...
}
```

초기화 할 때 숫자 대신 상수를 사용하여 초기화 하면 수정이 용이하다.

사용할 때

1. 숫자를 적는 거의 모든 곳
2. 의미를 가진 숫자

효과

1. 어떤 상태를 구분하고 싶을 때 유용하다.
   - ① 길 : 지나갈 수 있다
   - ② 벽 : 지나갈 수 없다
   - ③ 문 : 다음 스테이지로 갈 수 있다
2. 숫자가 아닌 단어이기 때문에 가독성에 좋다.

[참고]
enum class | 범위가 한정되어 열거자(NONE, WALL 등)의 이름을 다른 범위에서 사용할 수 있다. 강제 형변환이 가능하다.

2. 코딩의 팁

설명 전용 변수

- 캐릭터가 대시(Dash) 중인지 검사하는 코드에서,

```
if ((speed >= 10.0f) && !(y > 0.0f) && (state != STATE_DAMAGE))
{
    // 생략
}
```

- 각 조건별로 변수를 만들어 이해를 돕는다.

```
const bool isJump   { y > 0.0f };
const bool isDamage { state == STATE_DAMAGE };
const bool isFast   { speed >= 10.0f };
const bool isDash   { isFast && !isJump && !isDamage };
if (isDash)
{
    // 생략
}
```

사용할 때

1. 조건문에서 검사해야 할 조건이 많을 때
2. 계산의 중간 값이 여러 번 사용될 필요가 있을 때

효과

1. 주석 없이도 이해가 가능하다.
2. 한 번에 봐야 할 코드가 적어지므로 가독성에 좋다.
3. 변수이기 때문에 해당 값이 필요할 때 재활용할 수 있다.

[참고]
대입 연산자(=)로 초기화하는 대신 중괄호 초기화(균일 초기화)를 사용하여 등호 비교 연산자(==)와 구분할 수 있다.

2. 코딩의 팁

조건식 및 계산식의 함수화

- 설명 전용 변수의 코드를 함수로 분리할 경우

```
bool isJump(float y)        { return y > 0.0f; }
bool isFast(int speed)      { return speed > 10; }
bool isDamage(State state)  { return state == STATE_DAMAGE; }

bool isDash(float speed, float y, State state)
{
    // 한 줄마다 정보량이 최대한 적은편이 읽기 쉽다.
    // 복합 조건으로 작성된 코드를 분리하면 조건식을 간단하게 만들 수 있다.

    if (isJump(y))          return false;
    if (isDamage(state))    return false;
    if (!isFast(speed))     return false;
    return true;
}
```

사용할 때

1. 특정 조건을 재사용할 수 있을 때
2. 둘 이상의 변수를 사용해서 계산할 필요가 있을 때

효과

1. 함수명으로 함수의 동작을 짐작할 수 있다.
2. 함수에 들어가는 코드 자체가 줄어들어 한 눈에 함수 전체를 살필 수 있다.
3. 사양 변경 시 함수 단위로 수정할 수 있기 때문에 유지보수에 유리하다.

[참고]
오로지 해당 함수에서만 사용하는 조건/계산일 경우, 람다(lambda) 또는 설명 전용 변수를 사용하라.

2. 코딩의 팁

조기 리턴

※ **동일한 코드입니다.**

의식의 흐름 기법을 사용한 코드

```
int bonus(int time, int hp)
{
    int result = 0;
    if (time <= 50)
    {
        if (hp >= 30)
            result = 1000;
        else
            result = 500;
    }
    else if (time <= 100)
        result = 200;
    return result;
}
```

VS

```
int bonus(int time, int hp)
{
    if (time > 100) return 0;
    if (time > 50) return 200;
    if (hp < 30) return 500;
    return 1000;
}
```

조기 리턴 기법을 사용한 코드

사용할 때

1. 특정 조건이 만족해야 실행해야 하는 함수일 때

효과

1. 코드 블록이 줄어 가독성이 좋아진다.
2. 조건에 따라 함수를 바로 끝내기 때문에 불필요한 연산을 하지 않는다.

[참고]
의식의 흐름 기법 | 의식 가는 대로 짜는 코드. 이렇게 코딩하면 안 돼요!

2. 코딩의 팁

단언문Assert

- **assert.h** 헤더파일의 assert 함수를 사용한다.
- 함수의 인자로 조건문을 넘기면 조건문이 **거짓**일 경우에 경고 메시지를 띄우고 프로그램을 강제로 종료한다.

```
void display_week(int week)
{
    static const char* name[] = {
        "sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
    assert(0 <= week && week <= 6);
    cout << name[week] << endl;
}
```

사용할 때

1. 어떠한 경우라도 거짓이 되면 안 되는 조건일 때
2. 조건을 만족하지 않으면 동작을 보장하지 못할 때

효과

1. Release 모드에서 동작하지 않는다.
2. 개발 중에 어느 위치, 어떤 이유로 종료했는지 알려주기 때문에 디버깅에 매우 유용하다.

[참고]
C++11부터 _DEBUG_ERROR(문자열) 이라는 매크로를 제공한다. 이 매크로는 무조건 중단하지만 프로그램을 종료하지는 않는다.

헤더파일에 넣어야 하는 것

Bonus.

헤더 파일에 넣어야 하는 것

- #include 의 동작 원리 : **해당 헤더 파일을 붙여 넣는다.**
  ∴ **중복이 있으면 안 되는** 함수 정의나 변수는 헤더에 넣으면 안 된다.

- 대부분 소스 파일은 헤더 파일을 가지는 것을 권장한다.
  - 이름.cpp – 이름.h 와 같이 같은 이름으로 짓는 것을 권장.
  - 선언문은 대부분 헤더 파일에 작성한다.
  - 헤더 파일에서는 되도록 include하지 않는다
    - 헤더 파일에서 필요한 헤더는 소스 파일에서 미리 선언하는 게 좋다.
    - 헤더 파일에서 include 해야만 하는 경우 :
      1. 상속을 받은 **부모클래스가 다른 헤더파일**에 있을 경우
      2. 다른 헤더 파일에 정의된 객체를 사용할 때

넣어**야 하는** 것

1. 함수, 구조체, 클래스의 선언문(서명 뒤 ;)
2. inline 함수의 정의
3. template 함수의 정의

```
int func();
```

넣으면 **안 되는** 것

1. 변수
2. 함수 정의(서명 뒤 {})
3. 필요하지 않은 헤더 파일

```
int func() { return 1; }
```

[참고]
헤더 파일은 선언을 통해 짝이 되는 소스 코드에 어떤 함수가 있는지 한 눈에 볼 수 있게 하는 편이 좋다.

Visual Studio의 유용한 단축키

Bonus.

Visual Studio의 유용한 단축키

1. **[Alt 키를 누른 채 마우스 클릭]**
   - 마우스 커서가 있는 위치에 캐럿이 위치하고 그 위치에서 편집이 가능하다.
2. **[Alt + 키보드 방향키]**
   - 현재 캐럿이 있는 위치부터 키보드 방향키대로 범위가 선택된다.
3. **[Alt + 마우스 클릭 및 드래그]**
   - 마우스를 클릭한 위치부터 범위 선택이 된다.

4. **[범위를 선택하지 않고 Ctrl + X]**
   - 줄 전체를 잘라낸다.
5. **[범위를 선택하지 않고 Ctrl + C]**
   - 줄 전체를 복사한다.
   - 붙여 넣을 때 범위를 선택하지 않을 경우, 캐럿이 위치한 줄을 한 줄 내리고 붙여 넣는다.
6. **[Ctrl + ↑/↓]**
   - 스크롤 이동과 동일한 기능
7. **[Ctrl + →/←]**
   - Visual Studio가 인식하는 단어 단위로 이동
   - Shift를 누르고 하면 단어 단위로 범위 선택

Bonus.

Visual Studio의 유용한 단축키

8. [Alt + ↑/↓]
   - 캐럿이 위치한 줄을 위/아래 줄과 바꾼다.
9. [Ctrl + Space Bar]
   - 인텔리센스(IntelliSense)를 연다.
10. [블록 안에서 Ctrl + M, Ctrl + M]
    - 캐럿이 위치한 블록을 열거나 닫는다.
11. [단어를 선택하고 Ctrl + R, Ctrl + R]
    - 선택한 단어(클래스, 변수 등 의미를 가진 단어)를 다른 단어로 바꾼다.
12. [Ctrl + H]
    - 바꾸기(선택 범위, 현재 문서, 현재 블록, 현재 프로젝트, 현재 솔루션 / 대소문자 일치, 단어 일치)
13. [Ctrl + S]
    - 저장.
14. [F1]
    - 도움말을 연다.(기본 MSDN) 단어를 선택하고 F1을 누르면 해당 단어에 관련한 MSDN이 있으면 그곳으로 연결한다.
15. [F5]
    - 디버깅 모드로 실행한다.
16. [F9]
    - 현재 캐럿이 위치한 줄에 중단점을 찍는다.

Q & A