

auto

auto란?

- ▶ 선언 된 변수의 **초기화** 식에 따라 **해당 형식을 추론** 하도록 컴파일에 지시
- ▶ 타입 이름 철자 문제 및 오타 걱정 없이 보다 효율적으로 코딩이 가능하다

auto

```
auto a = 0;    //int
auto b = 1.0f; //float
auto c = 1.0;  //double
```

```
int a[] = {1, 2, 3};
for (auto value : a) //범위 기반 for 문[for(element : array)]
{
    cout << value << endl;
}
```

```
auto Add(int l, int r)
{
    return l + r;
}
int main()
{
    auto result = Add(1, 2);
}
```

template

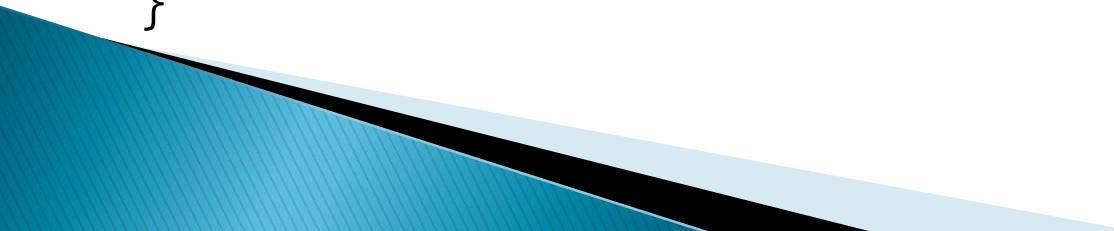
template란?

- ▶ 함수나 클래스를 개별적으로 다시 작성하지 않아도 여러 많은 자료형에서 사용할 수 있도록 만들어 놓은 틀
- ▶ 함수 템플릿과 클래스 템플릿으로 나누어 진다
- ▶ 형태
 - `template<class T>`
 - `template<typename T>`
 - 여기서의 `typename`과 `class`는 동일한 의미이다
 - 클래스 생성 시의 `class` 키워드와는 다른 것이며 혼동되지 않게 추가된 것이 `typename`이다

함수 템플릿(Function Template)

```
int sum(int l, int r)
{
    return l + r;
}
double sum(double l, double r)
{
    return l + r;
}
```

```
template<typename T>
T sum(T l, T r)
{
    return l + r;
}
```



함수 템플릿(Function Template)

```
template<typename T1, typename T2, , typename T3>  
auto sum(T1 a, T2 b, T3 c)  
{  
    return a + b + c;  
}
```

- 형식 매개 변수에 대한 실질적 제한은 없다

클래스 템플릿(Class Template)

//.h

```
template<typename T> class point
{
private:
    T _x, _y;
public:
    point(T, T);
    void set(T, T);
    T getX() const;
    T getY() const;
};
```

//.cpp

```
template<typename T> point<T>::point(T x, T y):_x(x), _y(y) {}
template<typename T> void point<T>::set(T x, T y)
{
    this->_x = x;
    this->_y = y;
}
template<typename T> T point<T>::getX() const { return this->_x; }
```


클래스 템플릿(Class Template)

```
//.h
template<typename T1>
class MyClass
{
    ...
public:
    template<typename T2>
    void MyFunc(T2);
};
```

```
//.cpp
template<typename T1> template<typename T2>
void MyClass<T1>::MyFunc(T2 param)
{
    ...
}
```

클래스 템플릿(Class Template)

▶ 주의 사항!!

- template class는 .h와 .cpp파일 나누는 것을 지원하지 않는다.
- .h에 정의 부문까지 포함하거나 main.cpp에 .cpp 파일을 include하여 쓰면 된다.