

도대체 멤버변수 개수가 왜 필요한데?

- C++ 템플릿 메타프로그래밍을 이용한 정적분석 -

까뽀

이런 사람이 있었습니다...



머리좀감고다녀라

난 진짜 **까쁘**가 여러번 역설했지만

5

멤버변수를 왜 세야하는지 아직도
납득을 못하는중

5

오후 7:41

게임에는 핵이 존재한다.

- 핵이 존재하는 이유
 - 피격 판정을 클라에서 계산하기 때문
- 서버에서 하는 것
 - 데미지
- 피격 판정은 반응성을 이유로 클라이언트에서 하는 경우가 많음
- FPS에 유독 핵이 많은 이유도 이것

```
45
46 struct Data
47 {
48     public:
49         int delay{};
50         int attackCount{};
51         int damage{};
52
53         int range{};
54 };
~
```

가장 간단하고 확실한 방법 해시값 비교

- 구조체 내에 있는 데이터를 이용하여 해시값을 만든다.
- 이를 특정 타이밍마다 서버와 비교하게 될 경우 데이터 변조는 사실상 완전히 막힘

```
struct Data
{
public:
    int delay{};
    int attackCount{};
    int damage{};

    int range{};

public:
    int CalcHash() const
    {
        int hash = 0;
        hash = GetHash( hash, delay );
        hash = GetHash( hash, attackCount );
        hash = GetHash( hash, damage );
        hash = GetHash( hash, range );
    }
};
```

아 맞다 까먹었다



변수를 추가 할 때 해시에 추가 안함

- 이라면 이 변수는 해시검사에서 제외되고 변조해도 검출 불가

```
24
25 struct Data
26 {
27     public:
28         int delay{};
29         int attackCount{};
30         int damage{};
31
32         int range{};
33         int range2{};
34
35     public:
36     int CalcHash() const
37     {
38         int hash = 0;
39         hash = GetHash( hash, delay );
40         hash = GetHash( hash, attackCount );
41         hash = GetHash( hash, damage );
42         hash = GetHash( hash, range );
43         // ???
44     }
45 };
~
```

멤버변수의 개수를 알 수 있다면...

- 어쨌든 기존과 달라졌는지를 체크해서 컴파일 에러를 띄울 수 있지 않을까
- 처음에는 구조체의 크기로 이 변화를 감지하고자 시도
- 구조체 패딩 문제 때문에 실패

유레카

```
10 namespace impl
11 {
12     struct any_type
13     {
14         template<typename T>
15         constexpr operator T();
16     };
17
18     template <class T, class... Ts>
19     decltype( void( T{ { std::declval<T>() }... } ), std::true_type{} ) test_is_braces_constructible( std::size_t );
20
21     template <class, class...>
22     std::false_type test_is_braces_constructible( ... );
23
24     template<std::size_t N, class T, class... Ts>
25     struct sizeof_struct
26     : public std::conditional_t<decltype( test_is_braces_constructible<T, Ts...>( 0 ) )::value,
27         sizeof_struct<N + 1, T, any_type, Ts...>,
28         std::integral_constant<std::size_t, N>>
29     {
30     };
31 }
32
33 template<typename T, typename = std::enable_if_t<std::is_aggregate_v<T>>>
34 using sizeof_struct = typename impl::sizeof_struct<0, T, impl::any_type>;
35
36 template<typename T>
37 constexpr inline auto sizeof_struct_v = sizeof_struct<T>::value;
38
```




이.이게뭐노

any_type은 템플릿과 연산자 오버로딩으로 어떤 타입이든 가능하게 만든 구조체

```
10 namespace impl
11 {
12     struct any_type
13     {
14         template<typename T>
15         constexpr operator T();
16     };
17
18     template <class T, class... Ts>
19     decltype( void( T{ { std::declval<Ts>() }... } ), std::true_type{} ) test_is_braces_constructible( std::size_t );
20
21     template <class, class...>
22     std::false_type test_is_braces_constructible( ... );
23
24     template<std::size_t N, class T, class... Ts>
25     struct sizeof_struct
26     : public std::conditional_t<decltype( test_is_braces_constructible<T, Ts...>( 0 ) )::value,
27     sizeof_struct<N + 1, T, any_type, Ts...>,
28     std::integral_constant<std::size_t, N>>
29     {
30     };
31 }
32
33 template<typename T, typename = std::enable_if_t<std::is_aggregate_v<T>>>
34 using sizeof_struct = typename impl::sizeof_struct<0, T, impl::any_type>;
35
36 template<typename T>
37 constexpr inline auto sizeof_struct_v = sizeof_struct<T>::value;
38
```

```
struct any_type
{
    template<typename T>
    constexpr operator T();
};
```

```

template<std::size_t N, class T, class... Ts>
struct sizeof_struct
    : public std::conditional_t<decltype( test_is_braces_constructible<T, Ts...>( 0 ) )::value,
    sizeof_struct<N + 1, T, any_type, Ts...>,
    std::integral_constant<std::size_t, N>>
{
};

```

```

10 namespace impl
11 {
12     struct any_type
13     {
14         template<typename T>
15         constexpr operator T();
16     };
17
18     template <class T, class... Ts>
19     decltype( void( T{ { std::declval<Ts>() }... } ), std::true_type{} ) test_is_braces_constructible( std::size_t );
20
21     template <class, class...>
22     std::false_type test_is_braces_constructible( ... );
23
24     template<std::size_t N, class T, class... Ts>
25     struct sizeof_struct
26     : public std::conditional_t<decltype( test_is_braces_constructible<T, Ts...>( 0 ) )::value,
27     sizeof_struct<N + 1, T, any_type, Ts...>,
28     std::integral_constant<std::size_t, N>>
29     {
30     };
31 }
32
33 template<typename T, typename = std::enable_if_t<std::is_aggregate_v<T>>>
34 using sizeof_struct = typename impl::sizeof_struct<0, T, impl::any_type>;
35
36 template<typename T>
37 constexpr inline auto sizeof_struct_v = sizeof_struct<T>::value;
38

```

```

template <class T, class... Ts>
decltype( void( T{ { std::declval<Ts>() }... } ), std::true_type{} ) test_is_braces_constructible( std::size_t );

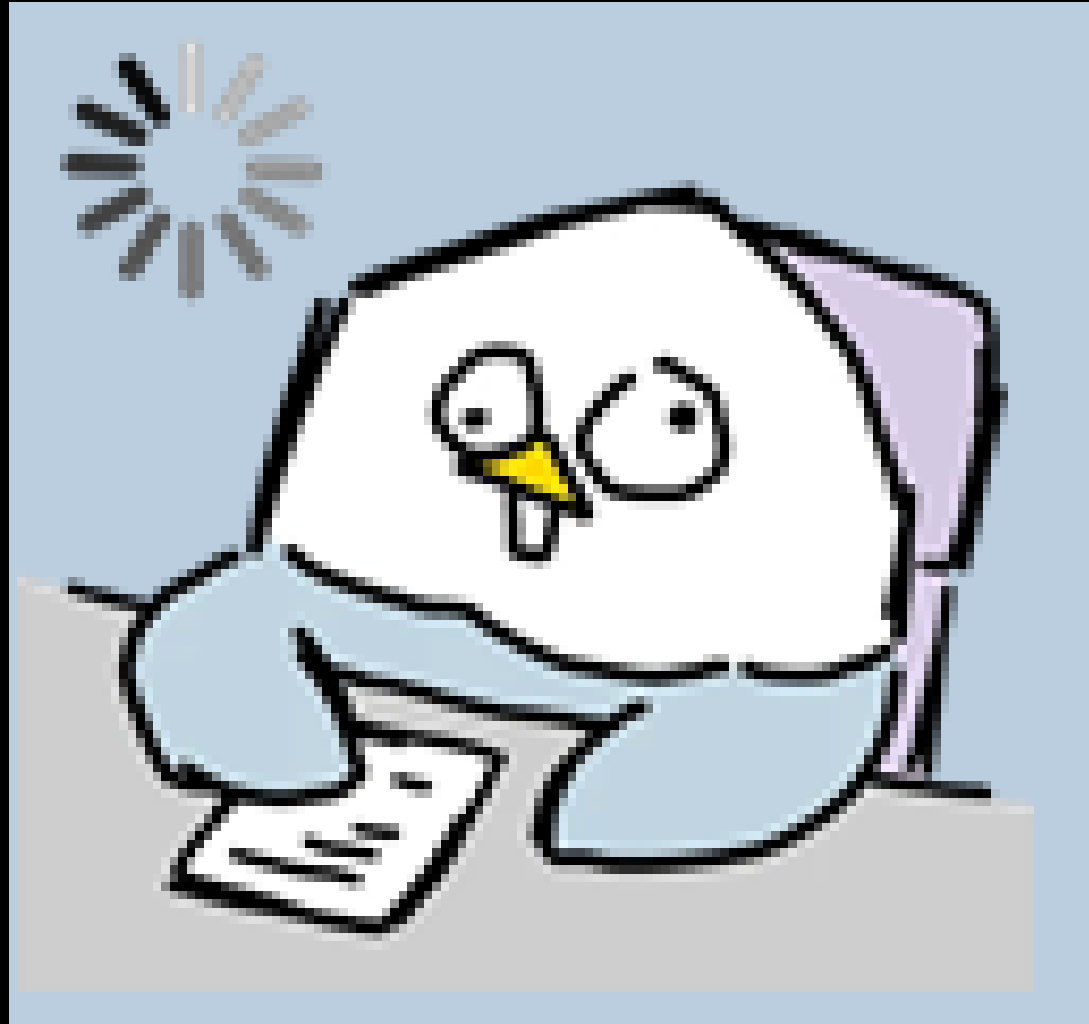
template <class, class...>
std::false_type test_is_braces_constructible( ... );

```

```

0 namespace impl
1 {
2     struct any_type
3     {
4         template<typename T>
5             constexpr operator T();
6     };
7
8     template <class T, class... Ts>
9     decltype( void( T{ { std::declval<Ts>() }... } ), std::true_type{} ) test_is_braces_constructible( std::size_t );
10
11     template <class, class...>
12     std::false_type test_is_braces_constructible( ... );
13
14     template<std::size_t N, class T, class... Ts>
15     struct sizeof_struct
16     : public std::conditional_t<decltype( test_is_braces_constructible<T, Ts...>( 0 )>::value,
17         sizeof_struct<N + 1, T, any_type, Ts...>,
18         std::integral_constant<std::size_t, N>>
19     {
20     };
21 }
22
23 template<typename T, typename = std::enable_if_t<std::is_aggregate_v<T>>>
24 using sizeof_struct = typename impl::sizeof_struct<0, T, impl::any_type>;
25
26 template<typename T>
27 constexpr inline auto sizeof_struct_v = sizeof_struct<T>::value;
28

```



어떤 원리인가

- C++ 에는 Aggregate한 타입이라는 게 존재
- {}로 초기화 가능한 것
- 컴파일이 안될 때까지 any_type을 넣어서 체크
- 컴파일이 안될 때 아까 false를 리턴하고 std::conditional에 의해 상속개수가 저장된 std::integral_constant를 상속 받게 됨

```
59 Data{ impl::any_type() }; // 컴파일 가능 N 1
60 Data{ impl::any_type(), impl::any_type() }; // 컴파일 가능 N 2
61 Data{ impl::any_type(), impl::any_type(), impl::any_type() }; // 컴파일 가능 N 3
62 Data{ impl::any_type(), impl::any_type(), impl::any_type(), impl::any_type() }; // 컴파일 가능 N 4
63 Data{ impl::any_type(), impl::any_type(), impl::any_type(), impl::any_type(), impl::any_type() }; // 컴파일 불가능 N 4인 상태로 std::integral_constant를 상속받고 끝
64
65
66
```

멤버변수를 추가하고 해시함수에 추가 안 하면 바로 컴파일 에러

```
45
46 struct Data
47 {
48 public:
49     int delay{};
50     int attackCount{};
51     int damage{};
52
53     int range{};
54     int range2{};
55
56 public:
57     int CalcHash() const;
58 };
```

```
39 template<typename T, typename... Types>
40 const int GetHash_Raw( int nCrc, Types...args )
41 {
42     static_assert( sizeof_struct_v<T> == sizeof...( args ), "fuck" );
43     return ( GetHash( nCrc, args ) + ... );
44 }
45
46 int Data::CalcHash() const
47 {
48     int hash = 0;
49     return GetHash_Raw<Data>( hash, delay, attackCount, damage, range );
50 }
51
52 int BombData::GetHash() const
```

110 % 문제 검색되지 않음

110 % 문제 검색되지 않음

오류 목록

전체 솔루션 1 오류 0 경고 0/1 메시지 빌드 + IntelliSense

코드	설명
C2338	fuck

프로젝트 ConsoleApp1

누군가 구조체 타입의 멤버를 추가한다면

```
0
7 struct BombData
8 {
9     int damage{};
10    int range{};
11    std::vector<int> preset;
12 };
13
14 struct Data
15 {
16     public:
17         int delay{};
18         int attackCount{};
19         int damage{};
20
21         int range{};
22         BombData ball;
23
24     public:
25         int CalcHash() const;
26 };
~
```


각 구조체에 GetHashCode를 추가적으로 구현

- Vector같은 애들때문에 구조체별로 따로 구현해줄 필요가 있음
- 근데 GetHashCode가 필요한 구조체인지 아닌지는 어떻게 구분해서 함수를 호출하지...?
- 그리고 이것도 누군가가 추가하는 걸 까먹으면 어떡하지??

아까 있었던 내용중 컴파일 에러를 이용한 오버로딩을 사용

```
5
6 template<typename T, typename = std::enable_if_t<std::is_member_function_pointer< decltype( &T::GetHash ) >::value>>
7 int GetHash( int key, const T& t )
8 {
9     return t.GetHash();
10 }
11
12 template<typename T, typename... Types>
13 int GetHash( int key, const T& t, const Types&... tps )
14 {
15     static_assert( std::is_trivially_copyable_v<T>, "need GetHash" );
16     static_assert( !std::is_pointer_v<T>, "fuckkkkk" );
17     int value = key;
18     for ( int i = 0; i < sizeof( T ); ++i )
19     {
20         char* ptr = ( char* )&t;
21         value += ptr[ i ];
22         value *= 29;
23     }
24     return value;
25 }
26
27 template<typename T>
28 int GetHash( int key, const std::vector<T>& vec )
29 {
30     int value = key;
31     for ( auto& t : vec )
32     {
33         value = GetHash( value, t );
34     }
35     return value;
36 }
37
```

```
45
46 int Data::CalcHash() const
47 {
48     int hash = 0;
49     return GetHash_Raw<Data>( hash, delay, attackCount, damage, range, ball );
50 }
51
52 int main()
53 {
54 }
55
56 // 프로그램 실행: <Ctrl+F5> 또는 [디버그] > [디버깅하지 않고 시작] 메뉴
57 // 프로그램 디버그: <F5> 키 또는 [디버그] > [디버깅 시작] 메뉴
58
```

110 % 문제 검색되지 않음

-- INSERT --

✖ C2338 need GetHash

드디어 반드시 해시체크에 제대로 추가해야 컴파일 일이 되는 코드 완성

- 사람은 누구나 언젠가 실수를 한다
- 이러한 실수를 미연에 방지할 수 있는 방어적 코딩을 하자
- 컴파일타임에 이런걸 분석해서 막을 수 있는 C++하실?