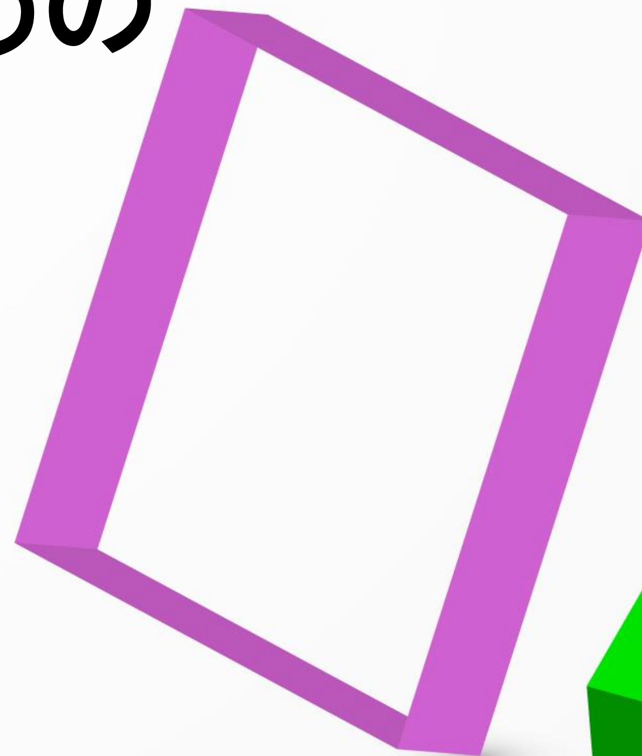
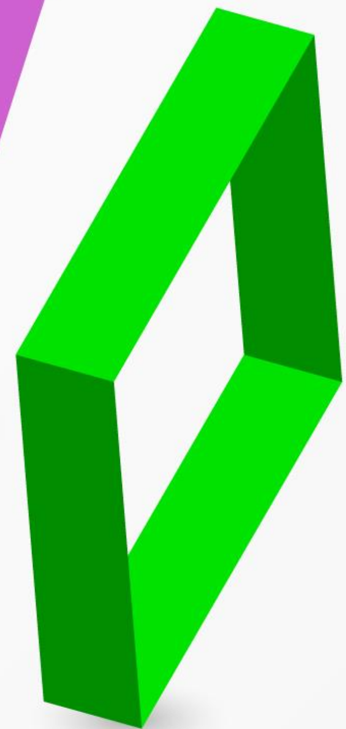
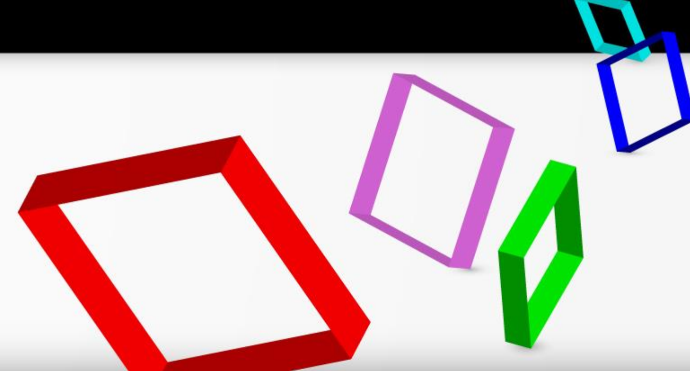
P-Review'19
「見やすいコードのための using」

CD部 須合雅之

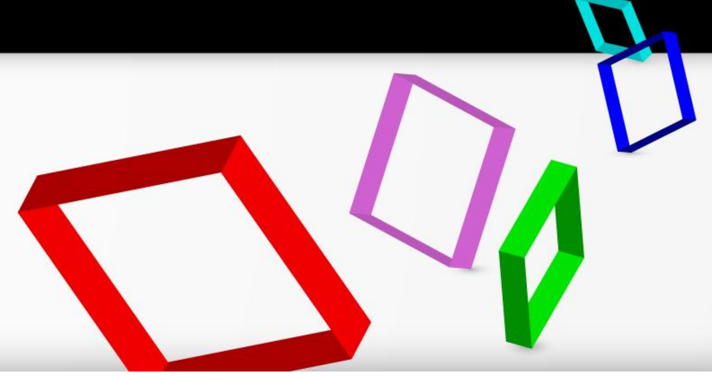



自己紹介



- 氏名　：　須合雅之
- 所属　：　コンテンツデザイン部
- 新卒1年目

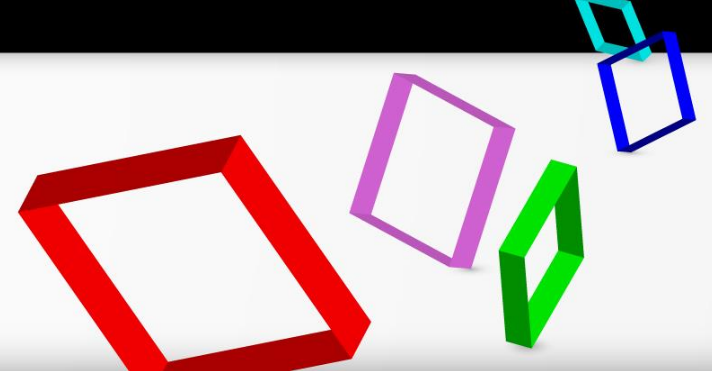
テーマの選定理由



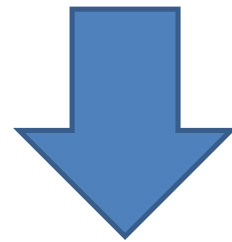
今回のテーマ･･･

「見やすいコードのためのusing」

テーマの選定理由

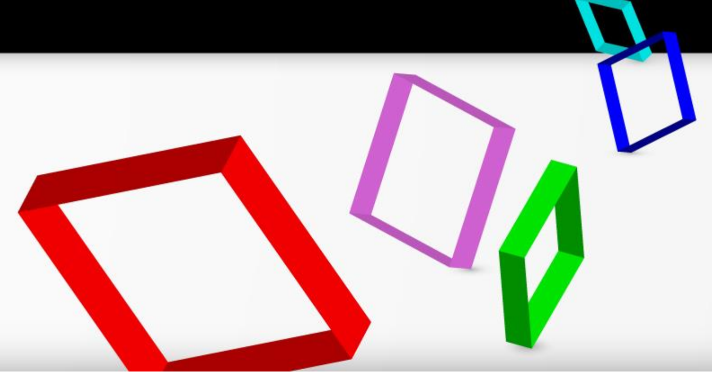


仕事で初めて触れたコードがとても見やすかった
見やすくするための手法にusingが使われていた



・usingを使ってコードを見やすくする手法
・usingがどんなことをしているか
後世のエンジニアに知ってもらいたい

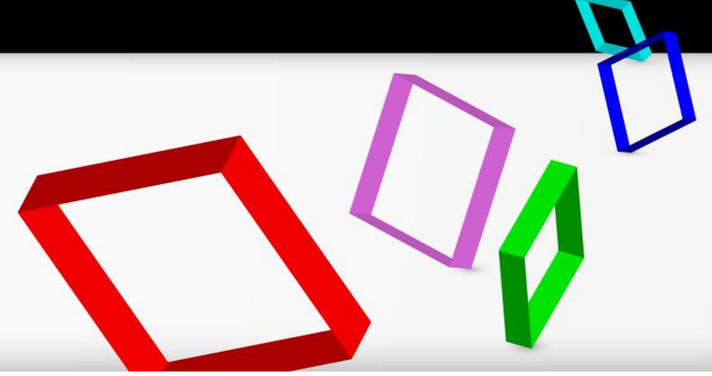
仕事で触れたコード



見やすいと感じた点

- 関数名、型名がわかりやすい
- 一つ一つの処理が簡潔
- 一行が長すぎない

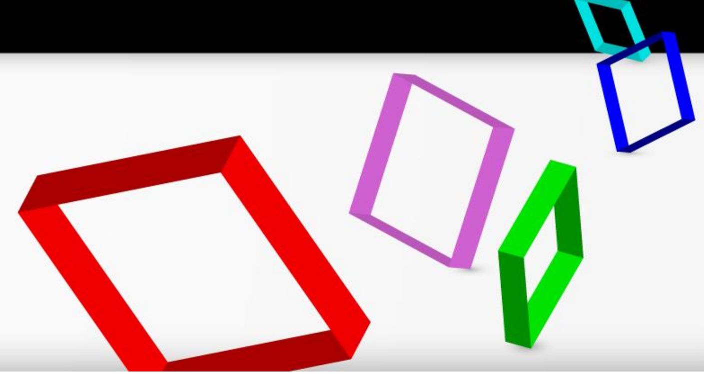
なぜ見やすさが大事か



コードが見づらいと、

読みたくない

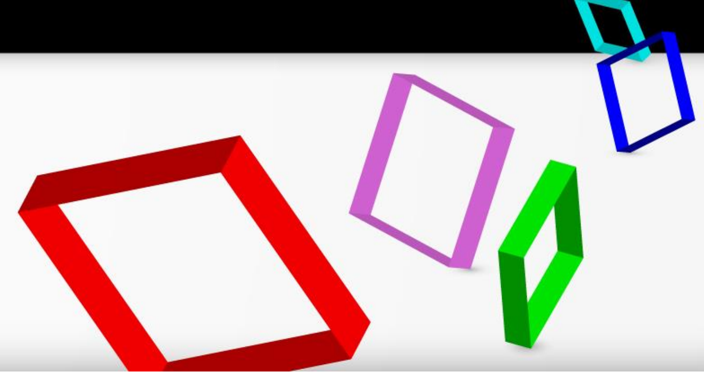
なぜ見やすさが大事か



※コード エディターの画面

```
C test.h ×
C test.h > ...
1
2
3  class MyProxy
4  {
5  public:
6      void loadPlayerData(const std::function<void()> &onSuccess, const std::function<void(const std::stri
7      void loadStageData(const std::function<void()> &onSuccess, const std::function<void(const std::strin
8      void loadItemData(const std::function<void()> &onSuccess, const std::function<void(const std::string
9      void loadCharacterData(const std::function<void()> &onSuccess, const std::function<void(const std::s
10 }
11
12
```

なぜ見やすさが大事か

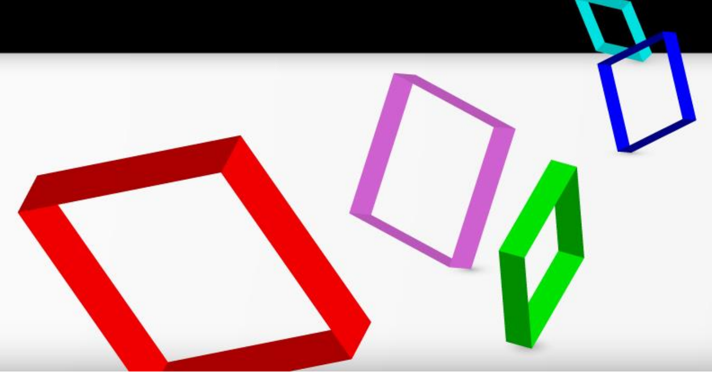


※コードエディターの画面

```
test.h
test.h > ...
1
2
3   class MyProxy
4   {
5   public:
6       void loadPlayerData(const std::function<void()> &onSuccess, const std::function<void(const std::stri
7       void loadStageData(const std::function<void()> &onSuccess, const std::function<void(const std::strin
8       void loadItemData(const std::function<void()> &onSuccess, const std::function<void(const std::string
9       void loadCharacterData(const std::function<void()> &onSuccess, const std::function<void(const std::s
10  }
11
12
```

画面に収まっていない

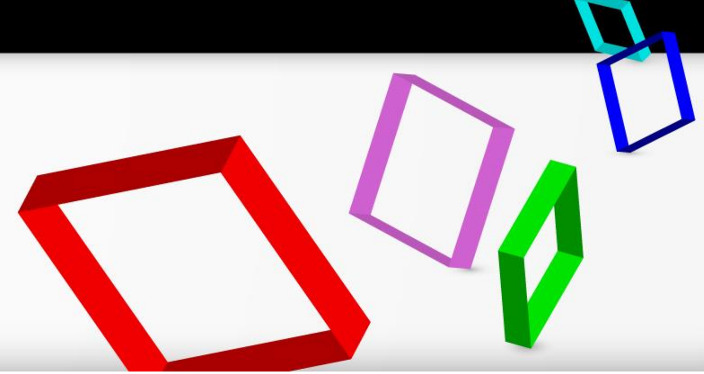
なぜ見やすさが大事か



見づらいコードを書いてしまうと
(読みたくないようなコードを書いてしまうと)
他人が読む際に解析に時間がかかり

<u>開発効率が悪くなってしまう</u>

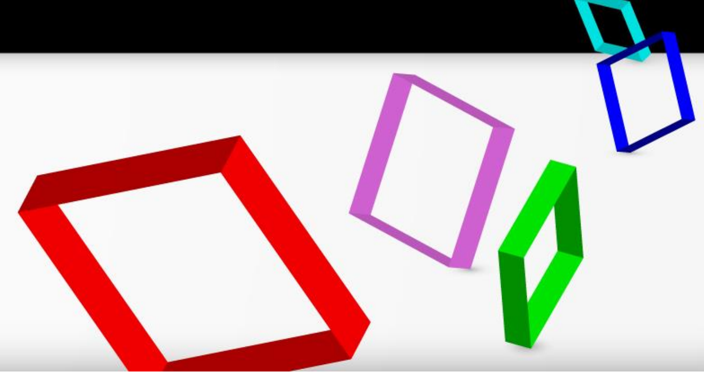
なぜ見やすさが大事か



※コード エディターの画面

```
C test.h ×
C test.h > ...
1   using TCallback = std::function<void()>;
2   using TOnError = std::function<void(const std::string &)>;
3
4   class MyProxy
5   {
6   public:
7       void loadPlayerData(const TCallback &onSuccess, const TOnError &onError);
8       void loadStageData(const TCallback &onSuccess, const TOnError &onError);
9       void loadItemData(const TCallback &onSuccess, const TOnError &onError);
10      void loadCharacterData(const TCallback &onSuccess, const TOnError &onError);
11  }
12
13
```

なぜ見やすさが大事か

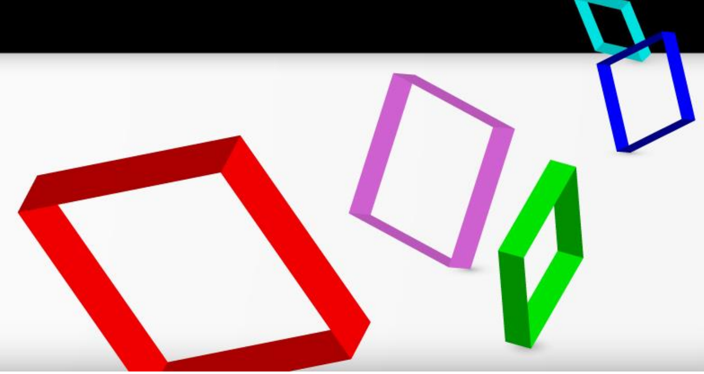


※コード エディターの画面

```
C test.h ×
C test.h > ...
1
2
3  class MyProxy
4  {
5  public:
6      void loadPlayerData(const std::function<void()> &onSuccess, const std::function<void(const std::stri
7      void loadStageData(const std::function<void()> &onSuccess, const std::function<void(const std::strin
8      void loadItemData(const std::function<void()> &onSuccess, const std::function<void(const std::string
9      void loadCharacterData(const std::function<void()> &onSuccess, const std::function<void(const std::s
10 }
11
12
```

読みたくない

なぜ見やすさが大事か



※コード エディターの画面

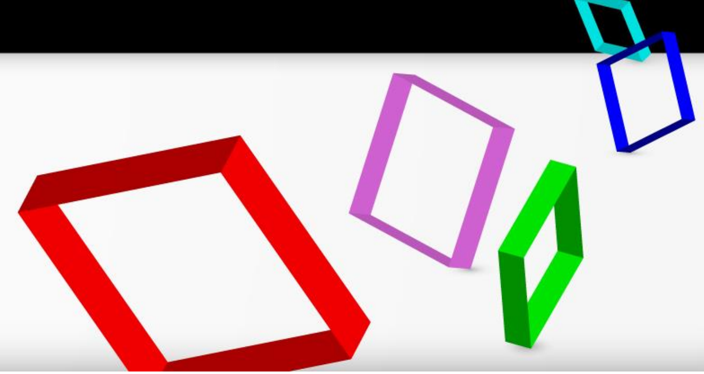
C test.h ×

C test.h > ...

```cpp
using TCallback = std::function<void()>;
using TOnError = std::function<void(const std::string &)>;

class MyProxy
{
public:
    void loadPlayerData(const TCallback &onSuccess, const TOnError &onError);
    void loadStageData(const TCallback &onSuccess, const TOnError &onError);
    void loadItemData(const TCallback &onSuccess, const TOnError &onError);
    void loadCharacterData(const TCallback &onSuccess, const TOnError &onError);
}

```

読めそう

なぜ見やすさが大事か



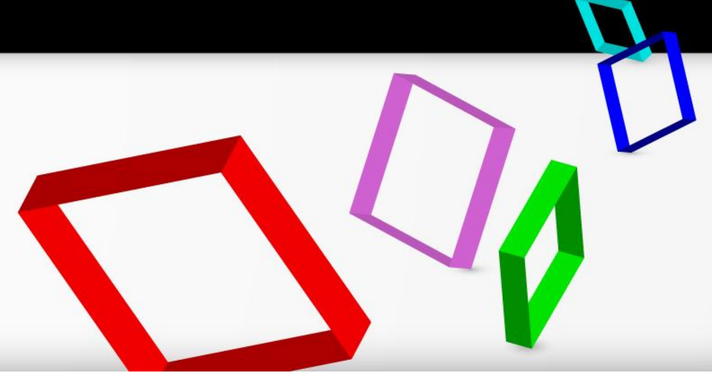
※コード エディターの画面

test.h

```
using TCallback = std::function<void()>;
using TOnError = std::function<void(const std::string &)>;

class MyProxy
{
public:
    void loadPlayerData(const TCallback &onSuccess, const TOnError &onError);
    void loadStageData(const TCallback &onSuccess, const TOnError &onError);
    void loadItemData(const TCallback &onSuccess, const TOnError &onError);
    void loadCharacterData(const TCallback &onSuccess, const TOnError &onError);
}

```

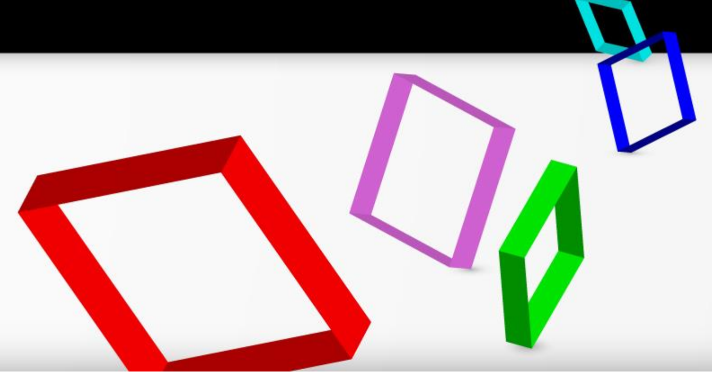
読めそう

usingでできること



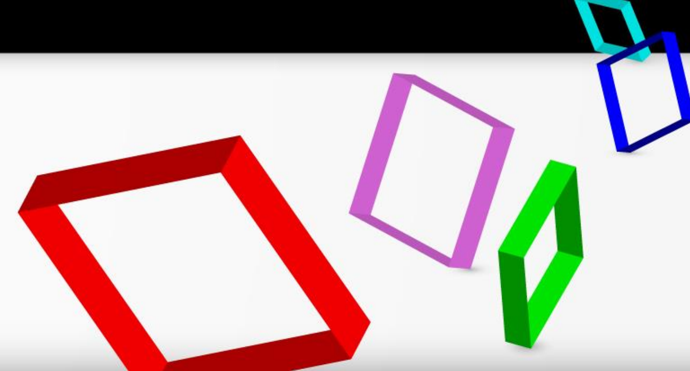
- 別名を付ける
- namespaceを省略

usingでできること



- 別名を付ける
- namespaceを省略

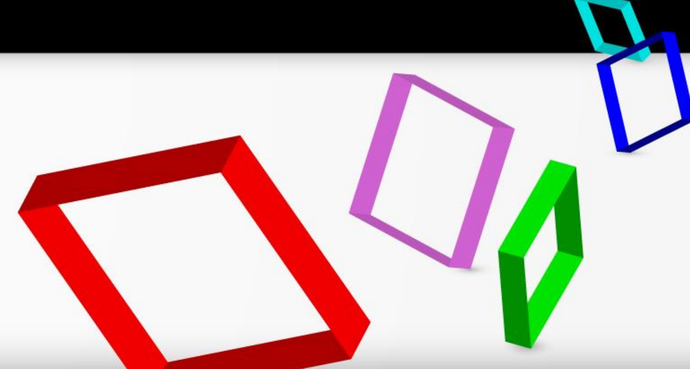
別名を付ける



usingを使う前のコード

```
struct BattleParamater{
    unsigned int Attack;
    unsigned int Defense;
    unsigned int MagicAttack;
    unsigned int MagicDefense;
}

unsigned int CalcDamage(unsigned int attack, unsigned int defence);
```

別名を付ける

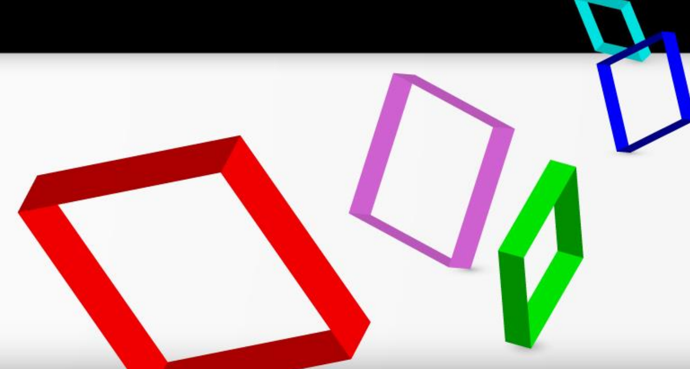


usingを使う前のコード

「unsigned int」をまとめたい

```
struct BattleParamater{
    unsigned int Attack;
    unsigned int Defense;
    unsigned int MagicAttack;
    unsigned int MagicDefense;
}

unsigned int CalcDamage(unsigned int attack, unsigned int defence);
```

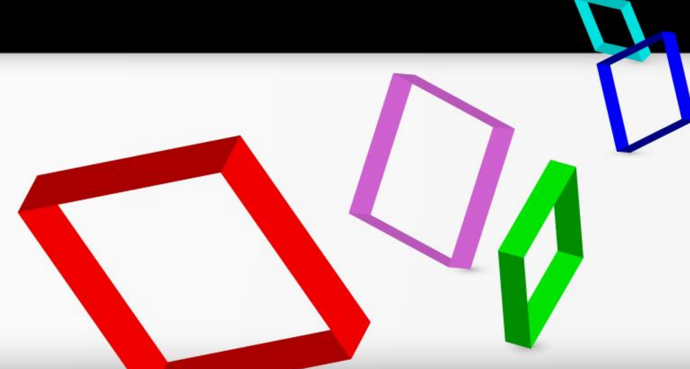
別名を付ける



usingを使ったコード

```
using uint = unsigned int;

struct BattleParamater{
    uint Attack;
    uint Defense;
    uint MagicAttack;
    uint MagicDefense;
}

uint CalcDamage(uint attack, uint defence);
```

別名を付ける

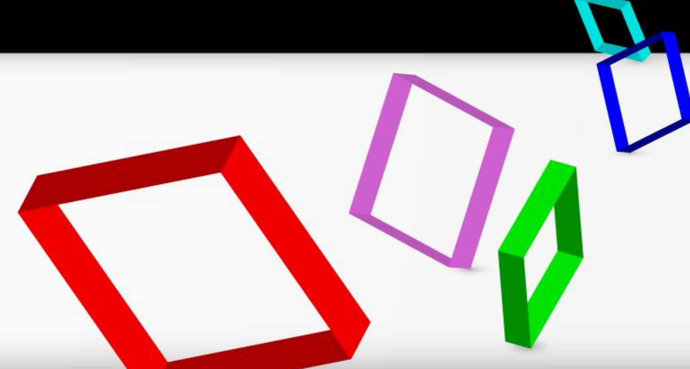


usingを使ったコード

「unsigned int」に「uint」という別名をつけた

```
using uint = unsigned int;

struct BattleParamater{
    uint Attack;
    uint Defense;
    uint MagicAttack;
    uint MagicDefense;
}

uint CalcDamage(uint attack, uint defence);
```

別名を付ける



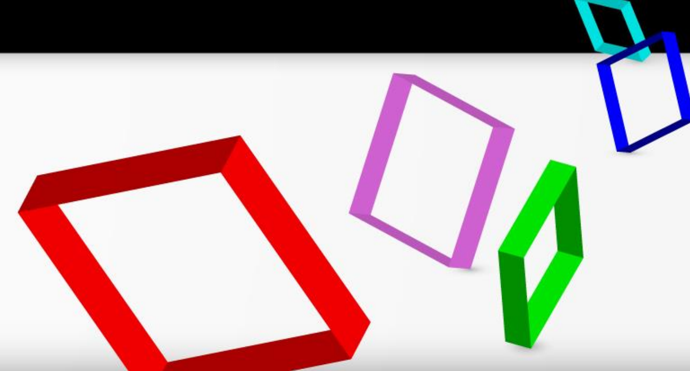
既存の型をusing を使って別名で宣言し直すことができる

式：using (新しい型名) = (古い型名)

よく見るusing

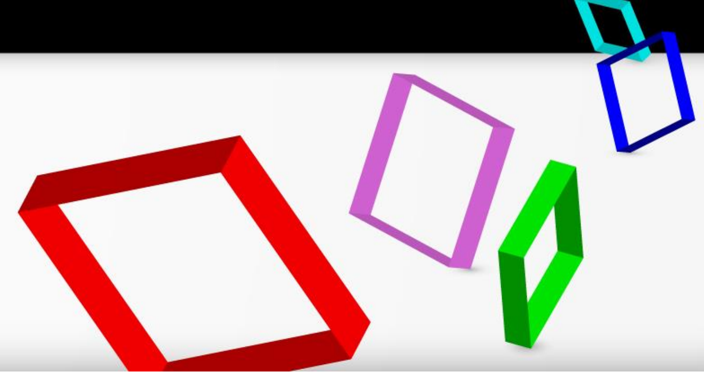
```
using s8  = signed char;        //  符号付き8ビット変数
using s16 = signed short;       //  符号付き16ビット変数
using s32 = signed int;         //  符号付き32ビット変数
using s64 = signed long long;   //  符号付き64ビット変数

using u8  = unsigned char;      //  符号無し8ビット変数
using u16 = unsigned short;     //  符号無し16ビット変数
using u32 = unsigned int;       //  符号無し32ビット変数
using u64 = unsigned long long; //  符号無し64ビット変数
```

別名を付ける



```
test.h
test.h > ...
1
2
3  class MyProxy
4  {
5  public:
6      void loadPlayerData(const std::function<void()> &onSuccess, const std::function<void(const std::stri
7      void loadStageData(const std::function<void()> &onSuccess, const std::function<void(const std::strin
8      void loadItemData(const std::function<void()> &onSuccess, const std::function<void(const std::string
9      void loadCharacterData(const std::function<void()> &onSuccess, const std::function<void(const std::s
10 }
11
12
```

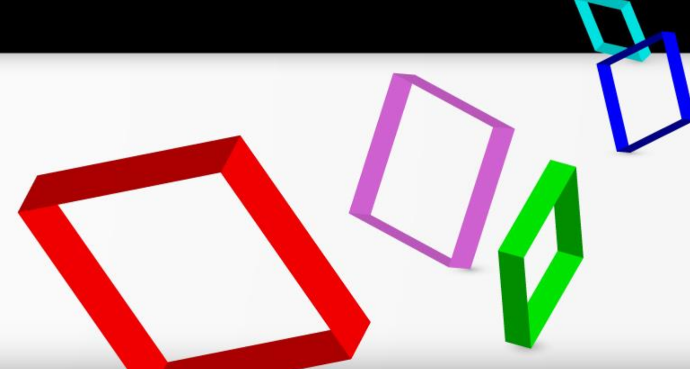
別名を付ける



test.h

```cpp
using TCallback = std::function<void()>;
using TOnError = std::function<void(const std::string &)>;

class MyProxy
{
public:
    void loadPlayerData(const TCallback &onSuccess, const TOnError &onError);
    void loadStageData(const TCallback &onSuccess, const TOnError &onError);
    void loadItemData(const TCallback &onSuccess, const TOnError &onError);
    void loadCharacterData(const TCallback &onSuccess, const TOnError &onError);
}

```

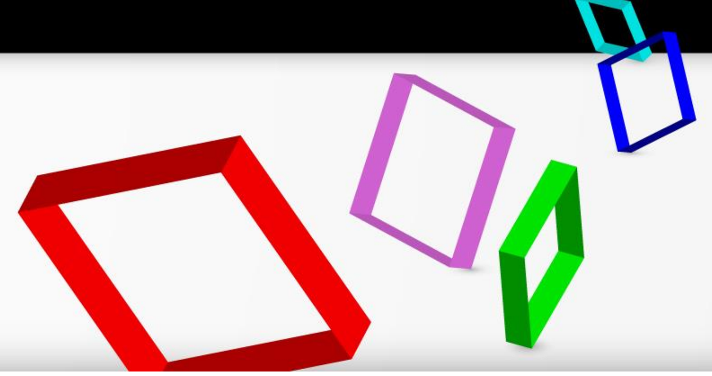
別名を付ける



C#の場合、クラスに対してだけでなく、namespaceに対しても別名を付けることができる

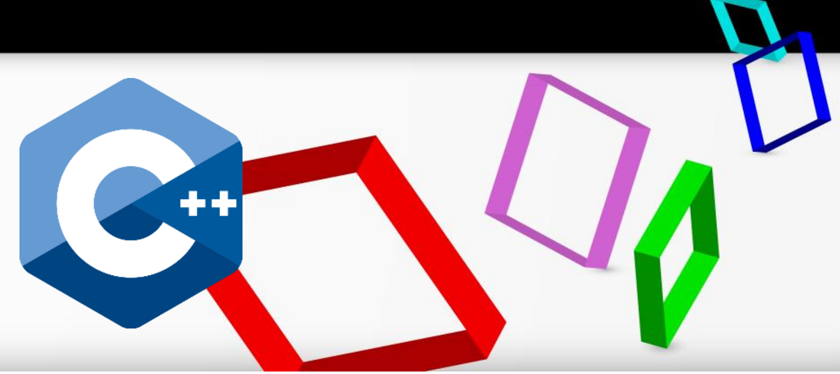
```
namespace PC{
    namespace MyCompany{
        namespace Project{
            public class MyClass { }
        }
    }

    using Project = PC.MyCompany.Project;
    class Program{
        void Main(){
            var mc = new Project.MyClass();
        }
    }
}
```

usingでできること



- 別名を付ける
- namespaceを省略

namespaceを省略

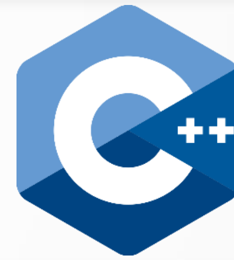


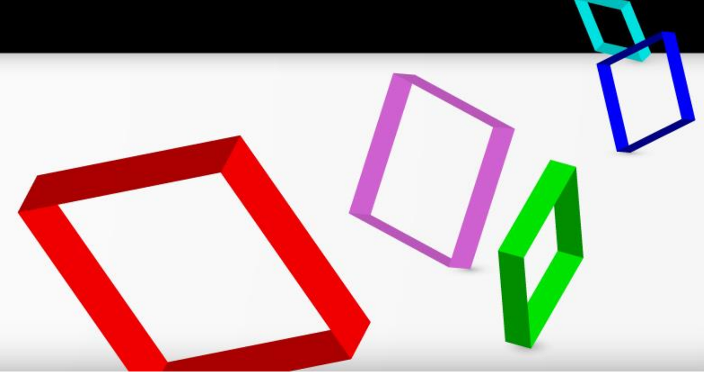
普通に使うと…

毎回「std::」付けるのは面倒、長くなる

```
#include <string>
#include <iostream>
#include <vector>

void Func()
{
    std::vector<std::string> month = {"むつき", "きさらぎ", "やよい"};

    for (int i = 0; i < month.size(); i++)
        std::cout << month[i] << std::endl;
}
```

namespaceを省略



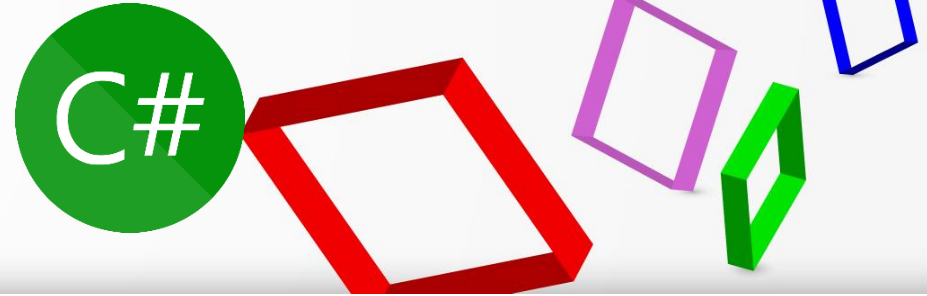


using namespaceで、指定したnamespaceを直接使える

「std::」が無くなり、見やすくなった

```
#include <string>
#include <iostream>
#include <vector>

using namespace std;

void Func()
{
    vector<string> month = {"むつき", "きさらぎ", "やよい"};

    for (int i = 0; i < month.size(); i++)
        cout << month[i] << endl;
}
```

namespaceを省略



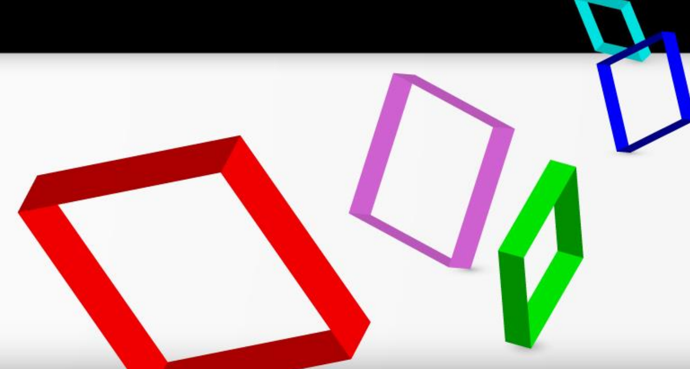
using未使用の場合

```
class Program{
    static void Main(){
        System.Console.WriteLine("√(3*3 + 4*4) = " + System.Math.Sqrt(3*3 + 4*4));
        System.Console.WriteLine("√(5*5 + 6*6) = " + System.Math.Sqrt(5*5 + 6*6));
    }
}
```

usingを使った場合

```
using System.Console;
using System.Math;
class Program{
    static void Main(){
        WriteLine("√(3*3 + 4*4) = " + Sqrt(3*3 + 4*4));
        WriteLine("√(5*5 + 6*6) = " + Sqrt(5*5 + 6*6));
    }
}
```

まとめ

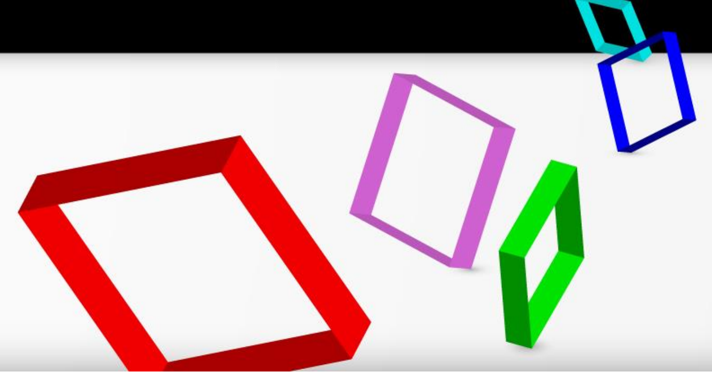


見づらいコードは読みたくない

見やすいコードのために、usingで

・別名を付ける

・namespaceを省略する



読もうと思えるコードを
目指しましょう