ITEM 2

Date:

Prefer const, enums to #define

or

#define ASPECT_RATIO 1.653

→ The name ASPECT_RATIO is never seen by compiler, it gets removed by preprocessor and may not get entered in symbol table.

→ It is confusing if you get error during compilation since error msg may refer to 1.653 and not ASPECT_RATIO

const double AspectRatio = 1.653

→ It is entered in symbol table.

→ use of const yields to smaller code bcoz pre processor blind substitution of macro/#define result in multiple copies of 1.653 in object code but use of const will result in only one copy.

Date:.....

Class, Static, Const

=> To make sure there is atmost one copy of constant, make it static members.

=> #define don't respect scope

=> class A

① { private:
    static const double AMember;
    ...
};

② const double A::Amember = 1.36;

① declaration of static class, constant goes in header file.

② Defination of static class constant goes to imp file.

But what if compilers want to know the size of array at compilation time ??? ...

Date:.....

Prefer Enum to #define.

```
class GamePlayer {
private:
    enum { NumTurns = 5 };
    int scores[NumTurns];
    ....
};
```

=> enum behaves some way more than like #define then const

=> its not legal to take address of enum like #define.

=> If we don't want anyone to get reference / pointer to one of integral constants, enum is great way to enforce.

Date:.....

Prefer inline template to #define.

```
#define CALL_MAX(a,b) f((a>b)?(a):(b))
int a=5, b=0
CALL_MAX(++a,b); // a incremented twice
CALL_MAX(++a,b+10); // a incremented once
template <typename T>
inline void callMax(const T& a, const T& b)
{
    f(a>b ? a:b);
}
```

-> since we don't know what is T, we pass by reference to const.