Item2: prefer const/enums/inlines to #define

#define RATIO 1.653 🡺 when error occurs, preprocessor error message just show something about 1.653 not RATIO

🡺prefer const double RATIO=1.653🡺const will ensure one copy

Note: char\*x=”dbssl” const char array, x=&array[0]

const char\* const Name=”xys”;🡺const std::string Name=”xys”;

string is more preferable to char\*

Note: static member of C++ class: no matter how many objects of the class are created, there is only one copy of static member.

Ex. Class GamePlayer{

static const int NumTurns=5;

int scores[NumTurns];

};🡺ensure at most one copy of the const by static declaration

Note:

Declaration: what the compiler needs, example:

extern int bar;

double f (int,double);

class f;

Definition: what the linker needs, example:

int bar;

double f(int r, double d){return r+d;}

class f{};

an identifier can be declared as often as you want, but it must be defined once. Ex.

Double f(int,double);

Double f(int,double);🡺 two lines are legal

Usually, C++ requires that a definition for anything you use, but class-specific static constants of integral type (ex. int, bool,char) are exception. In-class initialization is allowed only for integral types, only for constants.

Ex. class CostEstimate{  
 private:

static const double F;//declaration of static class constant; goes in header file

};

const double CostEstimate::F=1.35;//definition of static class constant; goes in implementation file

when you need the value of class constant during compilation of class[ex. compiler insists knowing the size of array during compilation]🡺 the enum hack

it is legal to take the address of a const

it is not legal to take the address of a #define

it is not legal to take the address of an enum

🡺advantages of enum hack:

1.never result in unnecessary memory allocation for const objects

2.fudamental of template metaprogramming

Ex. class GamePlayer{

enum{NumTurns=5};//make NumTurns a symbolic name for 5

int scores[NumTurns];//fine

};

Use template inline function to replace macro that looks like function

template<typename T>

inline void callWithMax(const T&a, const T&b){

if(a>b?a:b);

}// because we do not know what T is, we pass by reference to const

🡺obeys scope and access rules

Conclusion1: For simple constants prefer const objects or enum to #define🡺debug error message

Conclusion2: For function-like macro, prefer inline functions to #define🡺prevent parentheses problems

Ex. #define square(a) a\*a

Square(5) 🡺 5\*5 🡺25

Square(1+2) 🡺 1+2\*1+2 🡺5