条款15 尽可能使用constexpr

① const常量的不确定性(vscode可显示)

编译期常量

```
int e=10;  const int a=10;  const int b=1+2;
const int c=e;  const int d=get_value();  运行时常量
```

某些需要编译期常量的代码只能使用a,b

注:gcc的数组长度支持动态设置

② constexpr值:可以保证声明出的变量是编译期常量

所有constexpr对象都是const,但不是所有const都是constexpr

③ constexpr函数:C++11(最难用阶段)

普通函数

a.函数的返回值类型不能为void

b.函数体只能写return xxx,其中xxx必为常量/常量表达式

若有形参替换至expr中,expr必须仍为一个常量表达式

c.参数无需constexpr,以便函数退化

诡异的是 return x++/++x; 仍会报错(非常量表达式)

如果为constexpr函数传运行的值,或使用非常量值接收结果,

则constexpr退化为普通函数

构造函数

d.构造初始化列表中必须是常量表达式

e.构造体函数必须为空

f.所有和这个类相关的成员,析构函数必须为默认的

类成员函数

g.constexpr声明的成员函数具有const属性

④ C++14对constexpr函数的增强

打破a.b.e.g并且添加：函数可以修改生命周期和常量表达式相同的对象

⑤ if constexpr (since C++17)

void check( ) { if constexpr (sizeof(int)>sizeof(char)) {…} else {…} } 如果条件成立，else内的内容都不会编译

注：constexpr可让很多运算从运行时移动到编译期，带来的好处是模板参数可以通过constexpr计算出来 但constexpr会延长编译时间，并且去掉constexpr修饰，可能出现编译报错