# ## const

1. 修饰变量：在初始化是必须赋值

2. 修饰引用：

（1）引用类型必须与其所引用的对象的类型一致。==（但允许一个常量引用绑定到非常量对象上，即：不能通过该引用改变引用对象的值，但是该对象可以被其他途径改变）==

##### ```cpp

##### // const reference

##### int a=0;

##### int &b =a;

##### const int &c = a; // 常用引用

##### b = 2;

##### std::cout<<"a="<<a<<", b="<<b<<", c="<<c<<std::endl; // output 2 2 2

##### 

##### 

##### const int d = 5;

##### int &e = d; // error

##### ```

3. 修饰指针：

- **指向常量的指针**：==(1)仅仅要求不能通过该指针去改变对象的值，(2)但没有规定那个对象的值不能被其他途径改变==

##### ```cpp

##### int a = 2;

##### const int aa = 3;

##### int \*pa = &a;

##### const int \*pb = &a;

##### const int \*pbb = &aa;

##### std::cout <<"pb="<<\*pb<<", pbb="<<\*pbb<<std::endl;

##### 

##### 

##### int c = 4;

##### pb = &c;

##### pbb = &c;

##### std::cout <<"pb="<<\*pb<<", pbb="<<\*pbb<<std::endl;

##### 

##### 

##### c = 5;

##### std::cout <<"pb="<<\*pb<<", pbb="<<\*pbb<<std::endl;

##### 

##### ```

- **常量指针（const pointer）** ==\* 放在const 关键字之前==：是一个常量对象，所以必须要初始化，而且一旦初始化后，他的值就不能改变，他的值就是一个地址，就是说他里面存储的地址不能变，但其地址里的内容可以改变。

##### ```cpp

##### int a = 1;

##### const int b =2;

##### int \* const pa = &a;

##### int const \* const pb = &b;

##### 

##### std::cout <<"a="<<a<<", b="<<b<<std::endl;

##### \*pa = 3;

##### std::cout <<"a="<<a<<", b="<<b<<std::endl;

##### ```

const如果出现在星号左边，表示被之物是常量，如果出现在星号右边，表示指针自身是常量

1. ==顶层const（top level const）==：指针本身是一个对象，指针本身是一个const。

引申：const修饰的任意的对象是常量 如：const int a = 0；

2. ==底层const（low level const）：指针所指向的对象是一个const==

##### ```cpp

##### int a = 0;

##### const int b = 1; // 顶层

##### int \* const pa = &a; //顶层

##### const int \* pa2 = &a; //底层

##### const int \* const pb = &b; // 顶层 + 底层

##### const int &ra = a; //底层？？？

##### const int &rb = b; //底层？？？

// 对于顶层const, 拷贝对象没关系

##### a = b; //把顶层const b 拷贝给 非顶层const a

##### pa2 = pb; // 把pb的地址赋值给pa2, pa2是底层的，pb可以忽略顶层，只考虑底层

// 对于底层const, 必须具有相同的底层const资格，同时两个对象的数据类型必须要能转换

##### // int \*n = pa2; //错误 \*\*\*\*\*

```

6. 类的const 变量 只能调用const 方法，const 成员函数的目的是不改变类的成员变量

# ## constexpr 常量表达式

1. ==值不会改变，且编译过程中就能得到计算结果的表达式==

##### ```cpp

##### const int a = 1; //是常量表达式

##### const int b = a +1; //是常量表达式

##### 

##### int c = 0; //不是常量表达式

##### const int d = get\_size(); //不是常量表达式，运行时才知道

##### ```

2. C++11的规定，运行将变量声明为constexpr。目的是让编译器来检查验证该变量是不是一个常量表达式。以便虽然你想定义一个常量表达式，但是const int d = get\_size();却是在运行过程中值才被确定。

3. ==constexpr如果和指针一起用，只针对指针有效，对指针所指向的对象无效==。==即：constexpr的作用是把他所定义的对象置为顶层const==

##### ```cpp

##### const int a = 10; // 等价于 constexpr int a = 10;

##### constexpr int \* pa = &a //等价于 int \* const pa = &a

##### constexpr const int \* pb = &a; //等价于 const int \* const pb = &a

##### ```

4. constexpr 可以修饰类的成员函数，==被修饰的函数在执行期间不会改变对象的值.==所修饰的函数在其所有参数都是constexpr时，一定会返回constexpr

# ## 类型别名 与 指针混用

1. using 或 typedef

2. 容易出错的例子, ==不能把类型别名换成原来的代进去看==。方法是用const 与邻近的一层 括起来

##### ```cpp

##### typedef char \* CC;

##### const CC p1 = 0; // (1) p1是一个常量对象，不可改变，类型为CC，即指向char的指针，所以p1是个常量指针

**等价于 char\* const p1；p1所指向的对象是常量，所以p1存储单元格里的地址不能变**

**P1【地址】(不能变)->【char】**

##### const CC \*p2; // （2）p2是一个指针，const 表示不能通过p2去改变 其所指向对象的值，而他所指向的对象是CC

##### 等价于 char\* const p2

P2【地址】-> 【CC】- >【char】：能通过p2去改变 其所指向对象的值，但CC可以被其他改变

3. Iterator

STL的迭代器就是以指针塑造出来的，他的作用就像T\* 的指针，typedef T\* iterator. 所以我如果用const 修饰iterator ，其实等价于const Iterator = T\* const 即迭代器对象是常量. 所以如果想保住迭代器指向的对象的值不变，需要用const\_iterator

# ## const 和 constexpr

1. const 变量的初始化可以延迟到运行时，而 constexpr 变量必须在编译时进行初始化。

2. 所有constexpr对象都是const的，但是不是所有的const对象都是constexpr的。

# ## 成员函数的const

在设计类的时候，一个原则就是对于不改变数据成员的成员函数都要在后面加 const，而对于改变数据成员的成员函数不能加 const。所以 const 关键字对成员函数的行为作了更加明确的限定：

（1）有 const 修饰的成员函数（指 const 放在函数参数表的后面，而不是在函数前面或者参数表内），只能读取数据成员，不能改变数据成员；没有 const 修饰的成员函数，对数据成员则是可读可写的。

（2）除此之外，在类的成员函数后面加 const 还有什么好处呢？那就是==常量（即 const）对象可以调用 const 成员函数，而不能调用非const修饰的函数。==

## 1. 在C++中只有被声明为const的成员函数才能被一个const类对象调用。

## 2. const成员函数可以被对应的具有相同形参列表的非const成员函数重载

在这种情况下，类对象的常量性决定调用哪一个函数：

- const成员函数可以访问非const对象的非const数据成员，const数据成员，也可以访问const对象内的所有数据成员；

- 非const成员函数只可以访问非const对象的任意的数据成员（不能访问const对象的任意数据成员）；（上述原因可详见C++Primer（5th）231页。 在默认情况下，this的类型是指向类类型非常量版本的常量指针，例如 Screen类中，this类型为 Screen \*const。<u>当在成员函数的后面加上const关键字时，隐式的将this指针修改为 const Screen \*const 即指向类类型常量版本的常量指针</u>。根据初始化原则，我们不能将一个常量指针赋值给一个非常量指针）

- 作为一种良好的编程风格，在声明一个成员函数时，若该成员函数并不对数据成员进行修改， 应尽可能将该成员函数声明为const成员函数。

# ## 3. const修饰的是谁？

const成员函数的写法有两种

##### ```cpp

##### 1、void fun(int a，int b) const{}

##### 2、void const fun(int a,int b){}

##### ```

这两种写法的本质是：void fun (const 类 \*this, int a,int b);

\*\*const修饰的不是形参a和b；const修饰的是属性this->a和this->b。与const所写的位置无关。\*\* 为什么？

==因为c++对类的this指针做了隐藏，\*\*本质上，const指针修饰的是被隐藏的this指针所指向的内存空间，修饰的是this指针。\*\*==

## \*\*总结：\*\*

1）const成员函数可以访问非const对象的非const数据成员、const数据成员，也可以访问const对象内的所有数据成员；

2）非const成员函数可以访问非const对象的非const数据成员、const数据成员，但不可以访问const对象的任意数据成员；

3）作为一种良好的编程风格，在声明一个成员函数时，若该成员函数并不对数据成员进行修改操作，应尽可能将该成员函数声明为const 成员函数。

4）如果只有const成员函数，非const对象是可以调用const成员函数的。当const版本和非const版本的成员函数同时出现时，非const对象调用非const成员函数。

## \*\*补充：\*\*

\> 类中的const成员变量都要放在初始化列表之中进行

\*\*const成员函数\*\*

\*\*bitwise const：\*\*只有在不更改对象的任何成员变量时才可以说const。 编译器强制实施这项检查。值得注意的是，==把一个成员函数声明为const可以保证这个成员函数不修改数据成员，但是，如果据成员是指针，则const成员函数并不能保证不修改指针指向的对象，编译器不会把这种修改检测为错误。例如==

\*\*logical constness:\*\* mutable

----

# 别人的博客摘要

# \*\*⑴C中的const，功能比较单一，较容易理解：\*\*

作用：被修饰的内容不可更改。

使用场合：修饰变量，函数参数，返回值等。（c++中应用场合要丰富的多）

特点： 是运行时const，因此不能取代#define用于成为数组长度等需要编译时常量的情况。同时因为是运行时const，可以只定义而不初始化,而在运行时初始化。如 const int iConst;。 另外，在c中，const变量默认是外部链接，因此在不同的编译单元中如果有同名const变量，会引发命名冲突，编译时报错。

# \*\*⑵c++中的const：\*\*

## \*\*a、非类成员const：\*\*

①const变量默认是内部连接的，因此在不同的编译单元中可以有同名的const 变量定义。

②编译时常量，因此可以像#define一样使用，而且因为上面一点，可以在头文件中定义const变量，包含的不同的cpp文件（编译单元）中使用而不引起命名冲突。

③编译器默认不为const变量分配内存，除非：1. 使用 extern 申明， 2：程序中有引用const 变量的地址。

④c++中临时对象/内置变量默认具有const属性。

## \*\*b、类中的const：\*\*

①\*\*\*与c语言中的const一样，只是运行时常量，不能作为数组维数使用，即不能取代#define\\*\*\*。

　在类中使用下面两种方式取代#define：

　　1：static const...

　　2 : enum{....}//enum 不占存储空间。

②\*\*类中的const 变量占用存储空间\*\*。

③类中的const成员变量\*\*需要在构造函数初始化列表中初始化\*\*。

④const 对象：在该对象生命周期内，\*\*必须保证没有任何成员变量被改变\*\*。\*\*const对象只能调用const成员函数\*\*。

⑤\*\*const成员函数\*\*： void fun() const ... 不仅\*\*能被const对象调用，也能被非const对象调用\*\*，因此，如果确认一个任何成员函数不改变任何成员变量，应该习惯性将该函数定义成const类型。

⑥如果一个对象被定义成const，那么该const对象“可能”会被放入到ROM当中，这在嵌入式开发当中有时非常重要。

# \*\*4、const和#define的区别\*\*

\*\*const分配内存的时机，是编译器编译期间，与#define相同\*\*

C++中的const常量类似于宏定义#define

\*const int c=5 等价于 #define c 5\*

1) 编译器处理方式不同 \*\*define宏是在预处理阶段展开。 const常量是编译运行阶段使用。\*\*

2) 类型和安全检查不同 \*\*define宏没有类型，不做任何类型检查\*\*，仅仅是展开。 \*\*const常量有具体的类型，在编译阶段会执行类型检查\*\*。

\*\*注意：尽量以const替换#define\*\*

# \*\*5、类成员中的const变量\*\*

\> 类中的const成员变量都要放在初始化列表之中进行

　　> const数据成员

　　> 引用数据成员

　　> 对象数据成员（内置类）

　　\*\*const成员函数\*\*

　　> void print() const => const 类名 \* const this

　　> 在其内部是不能修改数据成员

　　> 也不能调用非const成员函数

　　> const对象只能调用const成员函数,必须要提供一个const版本的成员函数

再深入探讨类的const成员和成员函数，参考：<https://www.cnblogs.com/cthon/p/9178701.html>

\*\*补充：\*\*

\> 类中的static数据成员需要在类之外进行初始化

　　> 被类或类创建的对象共享

　　> 全局/静态区

　　\*\*静态成员函数\*\*