1. const与define。两者都可以用来定义常量，但是const定义时，定义了常量的类型，所以更精确一些。#define只是简单的文本替换，除了可以定义常量外，还可以用来定义一些简单的函数，有点类似内置函数。const和define定义的常量可以放在头文件里面。（小注：可以多次声明，但只能定义一次）

　　2.》》const与指针和引用。

　　（a）const与指针。

　　先来看看下面的几种定义：

　　int me；

　　const int \* p1=&me；//p1可变，\*p1不可变，但是不能用\*p1来修改，p1可以转向

　　int \* const p2=&me；//p2不可变，\*p2可变，此时允许\*p2来修改其值，但是p2不能转向。

　　const int \*const p3=&me；//p3不可变，\*p3也不可变，此时既不能用\*p3来修改其值，也不能转向

　　第一个const的意思是对p1来讲它指向的就是const常量，虽然me不是，但是对p1来说就是。

　　（b）指针和引用的的区别很简单，就是引用更简洁，更安全。因为引用声明是必须初始化。 引用更接近const指针，一旦与某个变量关联，就将一直效忠于他。

　　（c）const指针可以接受const和非const地址，但是非const指针只能接受非const地址。所以const指针的能力更强一些，所以尽量多用const指针，这是一种习惯。

　　3.》》（a）const与函数。由于 c，所以经常把函数的形参类型设为const，而且多为const 引用。但是这里有一个限制，不能把不是左值的地址传递给引用。（左值包括变量，数组元素，结构成员，引用，被解除引用的指针等）。 形参是const类型的，说明该函数将不会修改其值，该函数便为const函数是。

　　（b）const与类成员函数。先看看下面这段代码：

　　const Stock land =Stock("hyd");

　　land.show();

　　land 是常量，但是类成员函数show无法保证不修改land，所以编译器将拒绝执行该段代码。除非你能保证show像const函数一样，但这需要另外一种语法，即：

　　void show() const;(声明)

　　void Stock::show() const{}(定义)。

　　（二）详细介绍，注重细节

　　C中常用:“ #define 变量名 变量值”定义一个值替代,然而却有个致命缺点:缺乏类型检测机制,这样预处理在C++中成为可能引发错误的隐患,于是引入const.

　　const使用:

　　1. 用于指针的两种情况:const是一个左结合的类型修饰符.

　　int const \*A; //A可变,\*A不可变

　　int \*const A; //A不可变,\*A可变

　　2.限定函数的传递值参数:

　　void function(const int Var); //传递过来的参数在函数内不可以改变.

　　3.限定函数返回值型.

　　const int function(); //此时const无意义

　　const myclassname function(); //函数返回自定义类型myclassname.

　　4限定函数类型.

　　void function()const; //常成员函数, Const成员函数不能改变对象的成员函数。

　　例如：

　　int Point::GetY()

　　{

　　return yVal;

　　}

　　这个函数被调用时，不改变Point对象，而下面的函数改变Point对象：

　　void Point:: SetPt (int x, int y)

　　{

　　xVal=x;

　　yVal=y;

　　}

　　为了使成员函数的意义更加清楚，我们可在不改变对象的成员函数的函数原型中加上const说明：

　　class Point

　　{

　　public:

　　int GetX() const;

　　int GetY() const;

　　void SetPt (int, int);

　　void OffsetPt (int, int);

　　private:

　　int xVal, yVal;

　　};

　　const成员函数应该在函数原型说明和函数定义中都增加const限定：

　　int Point::GetY() const

　　{

　　return yVal;

　　}

　　class Set {

　　public:

　　Set (void){ card = 0; }

　　bool Member(const int) const;

　　void AddElem(const int);

　　//...

　　};

　　bool Set::Member (const int elem) const

　　{

　　//...

　　}

　　非常量成员函数不能被常量成员对象调用，因为它可能企图修改常量的数据成员：

　　const Set s;

　　s.AddElem(10); // 非法: AddElem不是常量成员函数

　　s.Member(10); // 正确

　　\*\*\*\*\*\*\*但构造函数和析构函数对这个规则例外，它们从不定义为常量成员，但可被常量对象调用（被自动调用）。它们也能给常量的数据成员赋值，除非数据成员本身是常量。

　　为什么需要const成员函数？

　　我们定义的类的成员函数中，常常有一些成员函数不改变类的数据成员，也就是说，这些函数是"只读"函数，而有一些函数要修改类数据成员的值。如果把不改变数据成员的函数都加上const关键字进行标识，显然，可提高程序的可读性。其实，它还能提高程序的可靠性，已定义成const的成员函数，一旦企图修改数据成员的值，则编译器按错误处理。

　　const成员函数和const对象

　　实际上，const成员函数还有另外一项作用，即常量对象相关。对于内置的数据类型，我们可以定义它们的常量，用户自定义的类也一样，可以定义它们的常量对象。例如，定义一个整型常量的方法为：

　　const int i=1 ；

　　同样，也可以定义常量对象，假定有一个类classA，定义该类的常量对象的方法为：

　　const classA a(2)；

　　这里，a是类classA的一个const对象，"2"传给它的构造函数参数。const对象的数据成员在对象寿命期内不能改变。但是，如何保证该类的数据成员不被改变呢？

　　为了确保const对象的数据成员不会被改变，在C++中，const对象只能调用const成员函数。如果一个成员函数实际上没有对数据成员作任何形式的修改，但是它没有被const关键字限定的，也不能被常量对象调用。下面通过一个例子来说明这个问题：

　　class C

　　{

　　int X;

　　public:

　　int GetX()

　　{

　　return X;

　　}

　　void SetX(int X)

　　{

　　this->X = X;

　　}

　　};

　　void main()

　　{

　　const C constC;

　　cout<<constC.GetX();

　　}

　　如果我们编译上面的程序代码，编译器会出现错误提示：constC是个常量对象，它只能调用const成员函数。虽然GetX( )函数实际上并没有改变数据成员X，由于没有const关键字限定，所以仍旧不能被constC对象调用。如果我们将上述代码中：

　　int GetX()

　　改写成：

　　int GetX()const

　　再重新编译，就没有问题了。

　　const成员函数的使用

　　const成员函数表示该成员函数只能读类数据成员，而不能修改类成员数据。定义const成员函数时，把const关键字放在函数的参数表和函数体之间。有人可能会问：为什么不将const放在函数声明前呢？因为这样做意味着函数的返回值是常量，意义完全不同。下面是定义const成员函数的一个实例：

　　class X

　　{

　　int i;

　　public:

　　int f() const;

　　};

　　关键字const必须用同样的方式重复出现在函数实现里，否则编译器会把它看成一个不同的函数：

　　int X::f() const

　　{

　　return i;

　　}

　　如果f( )试图用任何方式改变i或调用另一个非const成员函数，编译器将给出错误信息。任何不修改成员数据的函数都应该声明为const函数，这样有助于提高程序的可读性和可靠性。