 const用法主要是防止定义的对象再次被修改,定义对象变量时要初始化变量

下面我就介绍一下几种常见的用法

1.用于定义常量变量,这样这个变量在后面就不可以再被修改

 const int Val = 10;

 //Val = 20; //错误,不可被修改

2. 保护传参时参数不被修改,如果使用引用传递参数或按地址传递参数给一个函数,在这个函数里这个参数的值若被修改,

则函数外部传进来的变量的值也发生改变,若想保护传进来的变量不被修改,可以使用const保护

 void  fun1(const int &val)

  {

     //val = 10; //出错

}

void fun2(int &val)

{

   val = 10; //没有出错

}

void main()

{

   int a = 2;

   int b = 2;

   fun1(a); //因为出错,这个函数结束时a的值还是2

   fun2(b);//因为没有出错,函数结束时b的值为10

}

如果只想把值传给函数,而且这个不能被修改,则可以使用const保护变量,有人会问为什么不按值传递,按值传递还需要把这个值复制一遍,

而引用不需要,使用引用是为了提高效率//如果按值传递的话,没必要加const,那样根本没意义

3. 节约内存空间,

 #define  PI  3.14 //使用#define宏

 const double Pi = 3.14 //使用const,这时候Pi并没有放入内存中

 double  a = Pi;  //这时候才为Pi分配内存,不过后面再有这样的定义也不会再分配内存

 double  b = PI;  //编译时分配内存

 double  c = Pi;  //不会再分配内存,

 double  d = PI;  //编译时再分配内存

const定义的变量,系统只为它分配一次内存,而使用#define定义的常量宏,能分配好多次,这样const就很节约空间

4.类中使用const修饰函数防止修改非static类成员变量

 class

{

 public:

  void fun() const //加const修饰

   {

     a = 10; //出错,不可修改非static变量

     b = 10; //对,可以修改

}

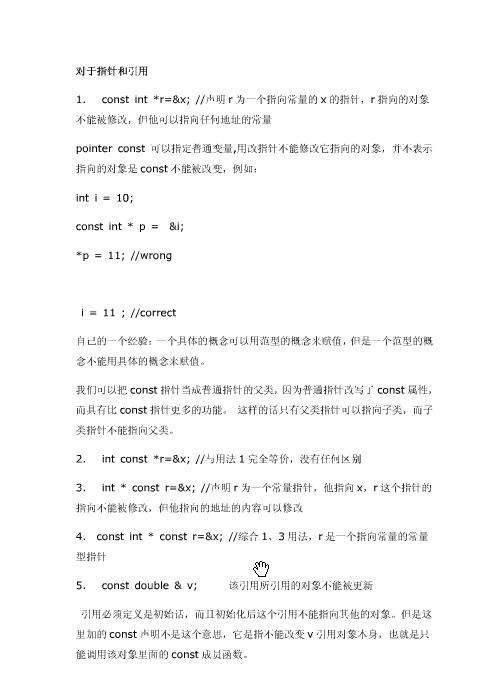
 private:

  int  a ;

  static int b;

}

5.修饰指针  
const int \*A; 或 int const \*A;  //const修饰指向的对象，A可变，A指向的对象不可变  
int \*const A; 　             //const修饰指针A， A不可变，A指向的对象可变   
const int \*const A;           //指针A和A指向的对象都不可变



6.修饰函数返回值,防止返回值被改变

  const int fun();

  接收返回值的变量也必须加const

  const int a = fun(); //接收的变量也要是const的,int a = fun()是错误的

7.修饰类的成员变量

  使用const修饰的变量必须初始化,在类中又不能在定义时初始化,

如;

class

{

private:

  int a = 10;

  const int b = 10;

  static const int c = 10;

//这样初始化都是错的,

}

初始化const int类型(没有static),在类的构造函数上初始化

Class Test

{

Public:

  Test():b(23) //构造函数上初始化b的值为23

   {

}

private:

     const int b ;

}

初始化staticconst int这个类型的(带有static的),在类的外面初始化

class Test

{

private:

  static const int c;

}

const int Test::c=10; //类的外部初始化c为10

8.const定义的对象变量只能作用于这个程序该C/C++文件,不能被该程序的其他C/C++文件调用,

 如file1.cpp中 const int val;

 在file2.cpp中, extern intval; //错误,无法调用,

要想const定义的对象变量能被其他文件调用,定义时必须使用extern修饰为

extern const int val;

非const变量默认为extern,要是const能被其他文件访问必须显示指定为extern