## C++ 中的 inline 用法

### 1、引入 inline 关键字的原因

在 c/c++ 中，为了解决一些频繁调用的小函数大量消耗栈空间（栈内存）的问题，特别的引入了 inline 修饰符，表示为内联函数。

栈空间就是指放置程序的局部数据（也就是函数内数据）的内存空间。

在系统下，栈空间是有限的，假如频繁大量的使用就会造成因栈空间不足而导致程序出错的问题，如，函数的死循环递归调用的最终结果就是导致栈内存空间枯竭。

下面我们来看一个例子：

## 实例

#include <stdio.h> //函数定义为inline即:内联函数

inline char\* dbtest(int a) {

return (i % 2 > 0) ? "奇" : "偶";

}

int main() {

int i = 0;

for (i=1; i < 100; i++) {

printf("i:%d 奇偶性:%s /n", i, dbtest(i));

}

}

上面的例子就是标准的内联函数的用法，使用 inline 修饰带来的好处我们表面看不出来，其实，在内部的工作就是在每个 **for** 循环的内部任何调用 dbtest(i) 的地方都换成了 (i%2>0)?"奇":"偶"，这样就避免了频繁调用函数对栈内存重复开辟所带来的消耗。

### 2、inline使用限制

inline 的使用是有所限制的，inline 只适合涵数体内代码简单的涵数使用，不能包含复杂的结构控制语句例如 while、for、switch，并且不能内联函数本身不能是直接递归函数（即，自己内部还调用自己的函数）。

### 3、inline仅是一个对编译器的建议

inline 函数仅仅是一个对编译器的建议，所以最后能否真正内联，看编译器的意思，它如果认为函数不复杂，能在调用点展开，就会真正内联，并不是说声明了内联就会内联，声明内联只是一个建议而已。

### 4、建议 inline 函数的定义放在头文件中

其次，因为内联函数要在调用点展开，所以**编译器必须随处可见内联函数的定义**，要不然就成了非内联函数的调用了。所以，这要求每个调用了内联函数的文件都出现了该**内联函数的定义**。

因此，将**内联函数的定义**放在**头文件**里实现是合适的，省却你为每个文件实现一次的麻烦。（模板函数与模板类也是必须在头文件中）

**声明跟定义要一致**：如果在每个文件里都实现一次该内联函数的话，那么，最好保证每个定义都是一样的，否则，将会引起未定义的行为。如果不是每个文件里的定义都一样，那么，编译器展开的是哪一个，那要看具体的编译器而定。所以，最好将**内联函数定义**放在**头文件**中。

### 5、类中的成员函数与inline

**定义**在类中的**成员函数**默认都是**内联的**，如果在类定义时就在类内给出函数定义，那当然最好。如果在类中未给出成员函数定义，而又想内联该函数的话，那在类外要加上 inline，否则就认为不是内联的。

class A { public:void Foo(int x, int y) { } // 自动地成为内联函数 }

将成员函数的定义体放在类声明之中虽然能带来书写上的方便，但不是一种良好的编程风格，上例应该改成：

// 头文件 class A { public: void Foo(int x, int y); } // 定义文件 inline void A::Foo(int x, int y){}

### 6、inline 是一种"用于实现的关键字"

关键字 inline 必须与函数定义体放在一起才能使函数成为内联，仅将 inline 放在函数声明前面不起任何作用。

如下风格的函数 **Foo** 不能成为内联函数：

inline void Foo(int x, int y); // inline 仅与函数声明放在一起void Foo(int x, int y){}

而如下风格的函数 **Foo** 则成为内联函数：

void Foo(int x, int y);inline void Foo(int x, int y) {} // inline 与函数定义体放在一起

所以说，inline 是一种"**用于实现的关键字**"，而不是一种"用于声明的关键字"。一般地，用户可以阅读函数的声明，但是看不到函数的定义。尽管在大多数教科书中内联函数的声明、定义体前面都加了inline 关键字，但我认为**inline不应该出现在函数的声明中**。这个细节虽然不会影响函数的功能，但是体现了高质量C++/C 程序设计风格的一个基本原则：**声明与定义不可混为一谈，用户没有必要、也不应该知道函数是否需要内联。**

### 7、慎用 inline

内联能提高函数的执行效率（提高效率体现在不需要新开辟函数的栈帧，保存本次函数调用的上下文），为什么不把所有的函数都定义成内联函数？如果所有的函数都是内联函数，还用得着"内联"这个关键字吗？   
内联是以**代码膨胀（复制）**为代价，仅仅省去了函数调用的开销，从而提高函数的执行效率。   
如果执行函数体内代码的时间，相比于函数调用的开销较大，那么效率的收获会很少。另一方面，每一处内联函数的调用都要复制代码，将使程序的总代码量增大，消耗更多的内存空间（甚至导致栈溢出）。

**以下情况不宜使用内联：**   
（1）如果函数体内的代码**比较长**，使用内联将导致**内存消耗代价较高**。   
（2）如果函数体内出现**循环**，那么执行函数体内代码的时间要比函数调用的开销大。**类的构造函数和析构函数容易让人误解成使用内联更有效。要当心**构造函数和析构函数可能会隐藏一些行为**，如"偷偷地"执行了**基类或成员对象**的构造函数和析构函数。所以**不要随便地将构造函数和析构函数的定义体放在类声明中（短小不复杂（不复杂的意思是，没有调用复杂的成员变量的构造函数）的没有父类的构造函数或者析构函数的函数定义可以放到类内）**。**一个好的编译器将会根据函数的定义体，自动地取消不值得的内联（这进一步说明了 inline 不应该出现在函数的声明中）。

编译器会根据函数的定义体的复杂程度，来判断是否采用认为设置的inline。

### 8、总结

内联函数并不是一个增强性能的灵丹妙药。只有当**函数非常短小**的时候它才能得到我们想要的效果；但是，如果函数并不是很短而且在很多地方都被调用的话，那么将会使得可执行体的体积增大（可执行程序体积变大）。   
**最令人烦恼的**还是当**编译器拒绝内联**的时候。在老的实现中，结果很不尽人意，虽然在新的实现中有很大的改善，但是仍然还是不那么完善的。一些编译器能够足够的聪明来指出哪些函数可以内联哪些不能，但是大多数编译器就不那么聪明了，因此这就需要我们的经验来判断。**如果内联函数不能增强性能，就避免使用它！**