特征

> • 相当于把内联函数里面的内容写在调用内联函数处； • 相当于不用执行进入函数的步骤，直接执行函数体； • 相当于宏，却比宏多了类型检查，真正具有函数特性； • 编译器一般不内联包含循环、递归、switch 等复杂操作的内联函 数； • 在类声明中定义的函数，除了虚函数的其他函数都会自动隐式地当 成内联函数。

inline 使用

```
// 声明 1（加 inline，建议使用）
inline int functionName(int first, int second,...);
// 声明 2（不加 inline）
int functionName(int first, int second,...);
// 定义
inline int functionName(int first, int second,...) {/****/};
// 类内定义，隐式内联
class A {
 int doA() { return 0; } // 隐式内联
}
// 类外定义，需要显式内联
class A {
 int doA();
}
inline int A::doA() { return 0; } // 需要显式内联
```

编译器对 inline 函数处理步骤

> 1. 将 inline 函数体复制到 inline 函数调用点处；
> 2. 为所用 inline 函数中的局部变量分配内存空间；
> 3. 将 inline 函数的的输入参数和返回值映射到调用方法的局部变量 空间中；
> 4. 如果 inline 函数有多个返回点，将其转变为 inline 函数代码块 末尾的分支（使用 GOTO）。

优缺点

> • 内联函数同宏函数一样将在被调用处进行代码展开，省去了参数压 栈、栈帧开辟与回收，结果返回等，从而提高程序运行速度。 • 内联函数相比宏函数来说，在代码展开时，会做安全检查或自动类 型转换（同普通函数），而宏定义则不会。 • 在类中声明同时定义的成员函数，自动转化为内联函数，因此内联 函数可以访问类的成员变量，宏定义则不能。 • 内联函数在运行时可调试，而宏定义不可以。
>
> 缺点是每一个调用处都将代码展开，增加了空间开销。

虚函数可以是内联函数吗

> • 虚函数可以是内联函数，内联是可以修饰虚函数的，但是当虚函数 表现多态性的时候不能内联。 • 内联是在编译器建议编译器内联，而虚函数的多态性在运行期，编 译器无法知道运行期调用哪个代码，因此虚函数表现为多态性时 （运行期）不可以内联。 • inline virtual 唯一可以内联的时候是：编译器知道所调用的对象 是哪个类（如 Base::who()），这只有在编译器具有实际对象而不 是对象的指针或引用时才会发生。

```
#include <iostream>
using namespace std;
class Base
{
public:
  inline virtual void who()
  {
    cout << "I am Base\n";
  }
  virtual ~Base() {}
};
class Derived : public Base
{
public:
  inline void who() // 不写 inline 时隐式内联
  {
    cout << "I am Derived\n";
  }
};
int main()
{
  // 此处的虚函数 who()，是通过类（Base）的具体对象（b）来调用的，编译期间就能确
  定了，所以它可以是内联的，但最终是否内联取决于编译器。
  Base b;
  b.who();
  // 此处的虚函数是通过指针调用的，呈现多态性，需要在运行时期间才能确定，所以不能
  为内联。
  Base *ptr = new Derived();
  ptr->who();
  // 因为 Base 有虚析构函数（virtual ~Base() {}），所以 delete 时，会先调用派生
  类（Derived）析构函数，再调用基类（Base）析构函数，防止内存泄漏。
  delete ptr;
  ptr = nullptr;
  system("pause");
  return 0;
}
```