C++ <内置函数 inline-function>

平常调用函数时需要一定的时间和空间的开销

1、程序先执行函数被调用之前的语句；

2、流程的控制转移到被调用函数的入口处，同时进行参数传递

3、执行被调用函数中函数体的语句；

4、流程返回调用函数的下一条指令处，将函数返回值带回；

5、接着执行主调函数中未执行的语句。

这些动作都要花费一定的时间，如果有些函数需要频繁使用，则所用时间会很长，从而降低程序的执行效率，有些实用程序对效率是有要求的，要求系统响应时间短，这就希望尽量压缩时间的开销

C++提供一种提高效率的方法，即在编译时将所调用函数的代码直接嵌入到主调函数中，而不是将流程转出去。这种嵌入到主调函数中的函数称为内置函数（inline function），又称内嵌函数。

**指定内置函数的方法很简单，只需在函数首行的左端加一个关键字inline即可**

系统编译在遇到内置函数时，会直接使用内置函数体中的代码代替原本调用的部分，同时用实参代替形参，直接置换代码。

注意：可以在声明函数和定义函数时同时写inline关键字，也可以旨在其中一处声明inline，效果相同，都可以按内置函数处理

使用内置函数可以节省运行时间，但却增加了目标程序的长度。假设要调用10次内置函数，则在编译时先后10次将内置函数的代码复制并插入主调函数，这就增加了目标文件中主调函数的长度。因此一般只将规模很小（一般为5个语句以下）而使用频繁的函数声明为内置函数。在函数规模很小的情况下，函数调用的时间开销可能相当于甚至超过执行函数本身的时间，把它定义为内置函数，可大大减少程序运行时间。

内置函数中不能包括复杂的控制语句，如循环语句和switch语句。

应当说明∶对函数作inline声明，只是程序设计者对编译系统提出的一个建议，也就是说它是建议性的，而不是指令性的。并非一经指定为 inline，编译系统就必须这样做。编译系统会根据具体情况决定是否这样做，例如对前面提到的包含循环语句和switch语句的函数或一个递归函数是无法进行代码置换的，又如一个1000行的函数，也不大可能在调用点展开。此时编译系统就会忽略 inline 声明，而按普通函数处理。

归纳起来，只有那些规模较小而又被频繁调用的简单函数，才适合于声明为内置函数。

C++ [ 函数重载 (function overloading) ]

C++允许用同一个函数名定义多个函数，这些函数的参数个数和参数类型不同，这就是函数的重载。即对一个函数名重新赋予它新的含义，使一个函数名可以多用。

例如找到三个数中最大的数，分别考虑整数、双精度、长整型

重载函数除了允许参数类型不同以外，还允许参数的个数不同

例：

#include <iostream>

using namespace std;

int findMax(int a,int b){

return (((a > b) ? a : b) > c ? ((a > b) ? a : b) : c);

}

double findMax(double a,double b,double c){

return (((a > b) ? a : b) > c ? ((a > b) ? a : b) : c);

}

long findMax(long a,long b,long c){

return (((a > b) ? a : b) > c ? ((a > b) ? a : b) : c);

}

/\* 主函数 \*/

int main(){

cout << findMax(12,13) << endl;

cout << findMax(12.3,13.89,45.75) << endl;

cout << findMax(31483647,3143648,3147489) << endl;

return 0;

}

C++ [ 函数模板 (function template) ]

函数的重载可以实现一个函数名多用，将实现相同的或类似的功能的函数用同一个函数名来定义，但是在程序中仍然要分别定义每一个函数，函数体基本完全相同，只是形参的类型不同也要分别定义，能否对此再简化呢？

C++提供了函数模板(function template)。所谓的函数模板，实际上使建立一个通用函数，其函数类型和参数类型不具体指定，用一个虚拟的类型来代表。这个通用函数就称为函数模板。凡是函数体相同的函数都可以用这个模板来代替，不必定义多个函数，只需再模板中定义一次即可。在调用函数时系统会根据实参的类型来取代模板中的虚拟类型，从而实现了不同函数的功能。

修改之前的例子：

#include <iostream>

using namespace std;

template <typename T>

T findMax(T a,T b,T c){

return (((a > b) ? a : b) > c ? ((a > b) ? a : b) : c);

}

/\* 主函数 \*/

int main(){

cout << findMax(12,13,33) << endl;

cout << findMax(12.3,13.89,45.75) << endl;

cout << findMax(31483647,3143648,3147489) << endl;

return 0;

}

template <typename T> 是通用函数模板定义

template的含义是"模板",<>中先写关键字 typename 或 class ,后边跟一个类型参数T,类似标识符,是一个虚拟的类型名。

可以观察到，用函数模板比函数重载更方便，程序更加简洁。但是需要注意它只适用于函数的参数个数相同而类型不同，且函数体相同的情况，如果参数的个数不同，则不能用函数模板。

C++ [ 函数携带默认参数 ]

一般情况下，在函数调用时形参从实参那里取得值，因此实参的个数应与形参相同。有时多次调用同一函数时用同样的实参，C++提供简单的处理办法，给形参一个默认值，这样形参就不必一定要从实参取值了。

但是需要注意，被指定默认值的参数必须放在形参列表中的最右端，因为实参与形参是从左至右的顺序进行的。

例：求两个或三个整型数值的最大值

#include <iostream>

using namespace std;

int findMax(int a,int b,int c = INT\_MIN){

return (((a > b) ? a : b) > c ? ((a > b) ? a : b) : c );

}

/\* 主函数 \*/

int main(){

cout << findMax(12,22) << endl;

cout << findMax(78,66,324) << endl;

return 0;

}

使用带有默认参数的函数时有两点需要注意：

1、如果函数的定义在函数调用之前，则应在函数定义中给出默认值。如果函数的定义在函数调用之后，则在函数调用之前需要有函数声明，此时必须在函数声明中给出默认值，在函数定义时可以不给出默认值；

2、一个函数不能既作为重载函数，又作为有默认参数的函数。因为当调用函数时如果少些一个参数，系统无法判定时利用重载还是默认参数，出现二义性、无法执行。