

## 1.4 函数增强\_物联网 / 嵌入式工程师 - 慕课网

“ 慕课网慕课教程 1.4 函数增强涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

### 4. 函数增强

```
int add(int a,int b)
{
    return (a + b);
}
```

```
add:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
@ link register save eliminated.
push    {r7}
sub sp, sp, #12
add r7, sp, #0
str r0, [r7, #4]
str r1, [r7]
ldr r2, [r7, #4]
ldr r3, [r7]
add r3, r3, r2
mov r0, r3
adds    r7, r7, #12
mov sp, r7
@ sp needed
ldr r7, [sp], #4
bx lr
```

```
_Z3addii:
    .fnstart
.LFB0:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
@ link register save eliminated.
push    {r7}
sub sp, sp, #12
add r7, sp, #0
str r0, [r7, #4]
str r1, [r7]
ldr r2, [r7, #4]
ldr r3, [r7]
add r3, r3, r2
mov r0, r3
adds    r7, r7, #12
mov sp, r7
@ sp needed
ldr r7, [sp], #4
bx lr
```

C 语言的编译器对函数名不会做任何修改，C++ 编译器编译完代码后，函数名会被处理成

\*\*“函数名 + 参数类型的形式”\*\*

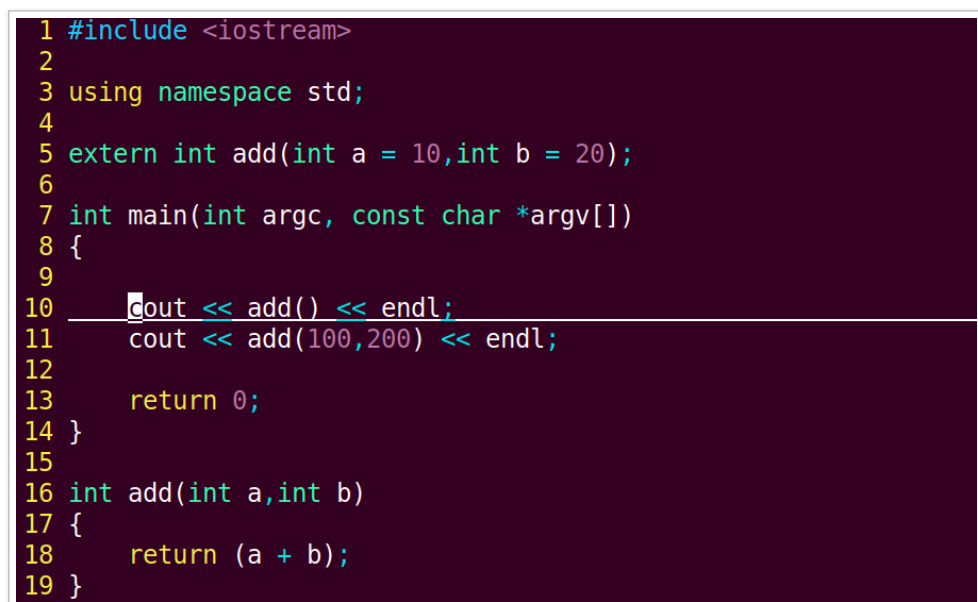
在相同的声明域中的函数名相同的，而参数表不同 \*\*(参数的个数或类型不同)\*\*，相互之间构成重载。 \*\*



在函数声明时为参数提供一个默认值，当函数调用时没有指定这个参数的值，编译器会自动使用默认值代替



当函数的定义和声明分开的时候，默认参数只能放在函数声明处



如果某个参数是默认值参数，那么它后面的参数必须都是默认参数

```
1 #include <stdio.h>
2
3 int add(int a = 10, int b)
4 {
5     return (a + b);
6 }
7
8 int main(int argc, const char *argv[])
9 {
10     printf("add() = %d\n", add());
11     return 0;
12 }
13
```

linux@ubuntu: ~/workdir/C++/quote

```
linux@ubuntu:~/workdir/C++/quote$ g++ function.cpp
function.cpp: In function 'int add(int, int)':
function.cpp:3:5: error: default argument missing for parameter 2 of '
t, int)'
   int add(int a = 10, int b)
       ^
linux@ubuntu:~/workdir/C++/quote$
```

在执行程序过程中如果要进行函数调用，则系统要将程序当前的一些状态信息存到栈中，同时转到函数的代码处去执行函数体语句，这些参数保存与传递的过程中需要时间和空间的开销，使得程序执行效率降低，特别是在程序频繁地进行函数调用以及函数代码段比较少时，这个问题会变得更为严重。

为了解决这个问题，C++ 引入了内联函数机制。就是将需要调用函数的代码，直接替换到调用函数的地方

```
extern int add(int a, int b) __attribute__((always_inline));
```

```
inline int add(int a, int b)
```

```
{
    return (a + b);
}
```

```
int main(void)
```

```
{
    int c = 0;

    c = add(5, 5);

    return 0;
}
```

编译器翻译之后的代码:

```

main:
    .fnstart
.LFB1:
    @ args = 0, pretend = 0, frame = 16
    @ frame_needed = 1, uses_anonymous_args = 0
    @ link register save eliminated.
    push    {r7}
    sub sp, sp, #20
    add r7, sp, #0
    movs    r3, #0
    str r3, [r7, #4]
    movs    r3, #5
    str r3, [r7, #8]
    movs    r3, #5
    str r3, [r7, #12]
    ldr r2, [r7, #8]
    ldr r3, [r7, #12]
    add r3, r3, r2
    str r3, [r7, #4]
    movs    r3, #0
    mov r0, r3
    adds    r7, r7, #20
    mov sp, r7
    @ sp needed
    ldr r7, [sp], #4
    bx lr

```

使用内联函数是一种用空间换时间的措施，若内联函数较长，且调用太频繁时，程序将加长很多\*\*。  
因此，通常只有较短的函数才定义为内联函数，对于较长的函数最好作为一般函数处理。\*\*

1. 观察如下代码，写出两个不同的重载函数, 完成浮点数和字符串操作

```

int my_swap(int a,int b);

#include <iostream>

int main(void)
{
    cout << calc(,100,200) << endl;

    return 0;
}

int calc(int a = 10,int b,int c)
{
    return (a + b + c);
}

```

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 beta, 点击查看详细说明

