文本替换宏

预处理器支持文本宏替换。亦支持仿函数文本宏替换。

语法

#define *标识符 替换列表*(可选)	(1)	
#define *标识符*(*形参*) *替换列表*(可选)	(2)	
#define *标识符*(*形参*, ...) *替换列表*(可选)	(3)	(C++11 起)
#define *标识符*(...) *替换列表*(可选)	(4)	(C++11 起)
#undef *标识符*	(5)	

解释

#define 指令

#define 指令定义 *标识符* 为宏，即指示编译器以 *替换列表* 替换 *标识符* 的所有后继出现，这能可选地额外处理。若标识符已定义为任何类型的宏，则程序为病式，除非定义等同。

仿对象宏

仿对象宏以 *替换列表* 替换每次出现的被定义 *标识符* 。 #define 指令的版本 (1) 准确表现如此。

仿函数宏

仿函数宏以 *替换列表* 替换每次出现的被定义 *标识符* ，可选地接受一定量实参，然后替换 *替换列表* 中任何 *形参* 的对应出现。

仿函数宏语法类似函数调用语法：每个后随作为下个预处理记号的 `(` 的宏名实例引入为 *替换列表* 所替换的记号序列。序列以匹配的 `)` 记号终止，跳过中间的匹配左右括号对。

对于版本 (2) ，实参数量必须与宏定义中的形参数量相同。对于版本 (3,4) ，实参数量必须多于 (C++20 前)不少于 (C++20 起)形参数量（不计 ... ）。否则程序为病式。若标识符不在函数记号中，即它自身后无括号，则它完全不被替换。

#define 指令的版本 (2) 定义简单仿函数宏。

#define 指令的版本 (3) 定义有可变数量实参的仿函数宏。附加实参（是谓 *可变实参*）可用 __VA_ARGS__ 标识符访问，它会被与要被替换的标识符一起提供的实参替换。

#define 指令的版本 (4) 定义有可变数量实参的仿函数宏，但无常规实参。附加实参（是谓 *可变实参*）只能用 __VA_ARGS__ 标识符访问，它会被与要被替换的标识符一起提供的实参替换。

对于版本 (3,4) ， *替换列表* 可以含有记号序列 **__VA_OPT__ (*内容*)** ，若 __VA_ARGS__ 非空，则它会被 *内容* 替换，否则不展开成任何内容。

```
#define F(...) f(0 __VA_OPT__(,) __VA_ARGS__)
#define G(X, ...) f(0, X __VA_OPT__(,) __VA_ARGS__)
#define SDEF(sname, ...) S sname __VA_OPT__(= { __VA_ARGS__ })
F(a, b, c) // 被 f(0, a, b, c) 替换
F()        // 被 f(0) 替换
G(a, b, c) // 被 f(0, a, b, c) 替换
G(a, )     // 被 f(0, a) 替换
G(a)       // 被 f(0, a) 替换
SDEF(foo);       // 被 S foo; 替换
SDEF(bar, 1, 2); // 被 S bar = { 1, 2 }; 替换
```

(C++20 起)

注意：若仿函数宏包含不为匹配的左右括号对所保护的逗号（最常于模板实参列表中找到，如于 `assert(std::is_same_v<int, int>);` 或 `BOOST_FOREACH(std::pair<int,int> p, m)` ），则逗号被转译成宏实参分隔符，导致实参数量不匹配所致的编译失败。

保留宏名

包含标准库头文件的翻译单元不可 #define 或 #undef 声明于任何标准库头文件的名称。

使用标准库任何部分的翻译单元不可 #define 或 #undef 词法上等同于下列内容的名称：

- 关键字

- 有特殊含义的标识符
- 任何标准属性记号

(C++11 起)

除了可定义 likely 与 unlikely 为仿函数宏。 (C++20 起)

否则，行为未定义。

与 ## 运算符

仿函数宏中， *替换列表* 中标识符前的 # 运算符经通过标识符运行形参替换，并将结果以引号环绕，等效地创建字符串字面量。另外，预处理器添加反斜杠以转义环绕内嵌字符串字面量的引号，若它存在，并按需要双写字符串中的反斜杠。移除所有前导和尾随空白符，并将文本中部（但非内嵌字符串字面量中内）的任何空白符序列缩减成单个空格。此操作被称为“字符串化”，若字符串化的结果不是合法的字符串字面量，则行为未定义。

出现于 __VA_ARGS__ 前时，整个展开的 __VA_ARGS__ 被包在引号中：

```
#define showlist(...) puts(#__VA_ARGS__)
showlist();           // 展开成 puts("")
showlist(1, "x", int); // 展开成 puts("1, \"x\", int")
```

(C++11 起)

替换列表 中任何二个相继标识符间的 ## 运算符在二个标识符（首先未被宏展开）上运行形参替换，然后连接结果。此操作被称为“连接”或“记号粘贴”。只有一同组成合法记号的记号才可以粘贴：组成更长标识符的标识符、组成数字的数位，或组成 += 的运算符 + 与 =。不能以粘贴 / 与 * 创建注释，因为考虑文本宏替换前，注释就被移除了。若连接的结果不是合法记号，则行为未定义。

注意：一些编译器提供扩展，允许 ## 出现于逗号后及 __VA_ARGS__ 前，此情况下 ## 在存在可变实参时不做任何事，但在不存在可变实参时移除逗号：这使得可以定义如 fprintf (stderr, format, ##__VA_ARGS__) 的宏。

#undef 指令

#undef 指令取消定义 *标识符* ，即取消 #define 指令所作的 *标识符* 定义。若标识符未关联到宏，则忽略该指令。

预定义宏

下列宏名已预定义于任何翻译单元中。

__cplusplus	指代所用的 C++ 标准版本，展开成值 199711L(C++11 前)、201103L(C++11)、 201402L(C++14) 或 201703L(C++17) (宏常量)
__STDC_HOSTED__ (C++11)	若实现有宿主（运行在 OS 下）则展开成整数常量 1 ，若独立（不随 OS 运行）则展开成 0 (宏常量)
__FILE__	展开成当前文件名，作为字符串字面量，能以 #line 指令更改 (宏常量)
__LINE__	展开成源文件行号，整数常量，能以 #line 指令更改 (宏常量)
__DATE__	展开成翻译日期，形式为 "Mmm dd yyyy" 的字符串。若月的日数小于 10 则 "dd" 的首字符为空格。月份名如同以 std::asctime() 生成 (宏常量)
__TIME__	展开成翻译时间，形式为 "hh:mm:ss" 的字符串字面量 (宏常量)
__STDCPP_DEFAULT_NEW_ALIGNMENT__ (C++17)	展开成 std::size_t 字面量，其值为调用不具对齐的 operator new 所保证的对齐（更大的对齐将传递给具对齐重载，如 operator new(std::size_t, std::align_val_t) ） (宏常量)

实现可能所预定义下列附加宏名。

__STDC__	若存在则为实现定义值，典型地用于指示 C 一致性 (宏常量)
__STDC_VERSION__ (C++11)	若存在则为实现定义值 (宏常量)
__STDC_ISO_10646__ (C++11)	若 wchar_t 使用 Unicode ，则展开成 yyyymmL 形式的整数常量，日期指示受支持 Unicode 的最近版本 (宏常量)
__STDC_MB_MIGHT_NEQ_WC__ (C++11)	若基础字符集的宽字符编码可以不等于其窄编码，则展开成 1 ，如在为 wchar_t 使用 Unicode 的基于 EBCDIC 的系统上。 (宏常量)
__STDCPP_STRICT_POINTER_SAFETY__ (C++11)	若实现支持严格 std::pointer_safety 则展开成 1 (宏常量)

__STDCPP_THREADS__ (C++11)	若程序能拥有多于一个执行线程则展开成 1 (宏常量)

这些宏的值（除了 __FILE__ 和 __LINE__ ）在翻译单元间保持常量。试图重定义或取消定义这些宏导致未定义行为。

注意：在每个函数作用域内部，有个特殊的函数局域预定义变量，名为 __func__ (C++11 起) ，定义为保有实现定义格式函数名的静态字符数组。它不是预处理器宏，但它被与 __FILE__ 和 __LINE__ 一起使用，例如由 assert 。

(C++20 起)

语言特性测试宏

下列宏预定义于每个翻译单元。

> 本节未完成
> 原因：在对应特性中描述

__cpp_aggregate_bases	展开成整数字面量 201603L (宏常量)
__cpp_aggregate_nsdmi	展开成整数字面量 201304L (宏常量)
__cpp_alias_templates	展开成整数字面量 200704L (宏常量)
__cpp_aligned_new	展开成整数字面量 201606L (宏常量)
__cpp_attributes	展开成整数字面量 200809L (宏常量)
__cpp_binary_literals	展开成整数字面量 201304L (宏常量)
__cpp_capture_star_this	展开成整数字面量 201603L (宏常量)
__cpp_char8_t	展开成整数字面量 201811L (宏常量)
__cpp_conditional_explicit	展开成整数字面量 201806L (宏常量)
__cpp_constexpr	展开成整数字面量 201603L (宏常量)
__cpp_decltype	展开成整数字面量 200707L (宏常量)
__cpp_decltype_auto	展开成整数字面量 201304L (宏常量)
__cpp_deduction_guides	展开成整数字面量 201703L (宏常量)
__cpp_delegating_constructors	展开成整数字面量 200604L (宏常量)
__cpp_enumerator_attributes	展开成整数字面量 201411L (宏常量)
__cpp_fold_expressions	展开成整数字面量 201603L (宏常量)
__cpp_generic_lambdas	展开成整数字面量 201304L (宏常量)
__cpp_guaranteed_copy_elision	展开成整数字面量 201606L (宏常量)
__cpp_hex_float	展开成整数字面量 201603L (宏常量)
__cpp_if_constexpr	展开成整数字面量 201606L (宏常量)
__cpp_inheriting_constructors	展开成整数字面量 201511L (宏常量)
__cpp_impl_destroying_delete	展开成整数字面量 201806L (宏常量)
__cpp_impl_three_way_comparison	展开成整数字面量 201711L (宏常量)
__cpp_init_captures	展开成整数字面量 201304L (宏常量)
__cpp_initializer_lists	展开成整数字面量 200806L (宏常量)
__cpp_inline_variables	展开成整数字面量 201606L (宏常量)
__cpp_lambdas	展开成整数字面量 200907L (宏常量)
__cpp_namespace_attributes	展开成整数字面量 201411L (宏常量)
__cpp_noexcept_function_type	展开成整数字面量 201510L

	(宏常量)
__cpp_nontype_template_args	展开成整数字面量 201411L (宏常量)
__cpp_nontype_template_parameter_auto	展开成整数字面量 201606L (宏常量)
__cpp_nontype_template_parameter_class	展开成整数字面量 201806L (宏常量)
__cpp_nsdmi	展开成整数字面量 200809L (宏常量)
__cpp_range_based_for	展开成整数字面量 201603L (宏常量)
__cpp_raw_strings	展开成整数字面量 200710L (宏常量)
__cpp_ref_qualifiers	展开成整数字面量 200710L (宏常量)
__cpp_return_type_deduction	展开成整数字面量 201304L (宏常量)
__cpp_rvalue_references	展开成整数字面量 200610L (宏常量)
__cpp_sized_deallocation	展开成整数字面量 201309L (宏常量)
__cpp_static_assert	展开成整数字面量 201411L (宏常量)
__cpp_structured_bindings	展开成整数字面量 201606L (宏常量)
__cpp_template_template_args	展开成整数字面量 201611L (宏常量)
__cpp_threadsafe_static_init	展开成整数字面量 200806L (宏常量)
__cpp_unicode_characters	展开成整数字面量 200704L (宏常量)
__cpp_unicode_literals	展开成整数字面量 200710L (宏常量)
__cpp_user_defined_literals	展开成整数字面量 200809L (宏常量)
__cpp_variable_templates	展开成整数字面量 201304L (宏常量)
__cpp_variadic_templates	展开成整数字面量 200704L (宏常量)
__cpp_variadic_using	展开成整数字面量 201611L (宏常量)

示例

运行此代码

```
#include <iostream>

// 制造函数工厂并使用它
#define FUNCTION(name, a) int fun_##name() { return a;}

FUNCTION(abcd, 12)
FUNCTION(fff, 2)
FUNCTION(qqq, 23)

#undef FUNCTION
#define FUNCTION 34
#define OUTPUT(a) std::cout << "output: " #a << '\n'

int main()
{
    std::cout << "abcd: " << fun_abcd() << '\n';
    std::cout << "fff: " << fun_fff() << '\n';
    std::cout << "qqq: " << fun_qqq() << '\n';
    std::cout << FUNCTION << '\n';
    OUTPUT(million);               // 注意缺少引号
}
```

输出：

```
abcd: 12
fff: 2
qqq: 23
34
output: million
```

参阅

文本替换宏 的 C 文档