



预定义符号

这些预定义符号都是语言内置的

```
__FILE__      //进行编译的源文件
__LINE__      //文件当前的行号
__DATE__      //文件被编译的日期
__TIME__      //文件被编译的时间
__STDC__      //如果编译器遵循ANSI C，其值为1，否则未定义
```

```
int main() {
    printf("%s\n", __FILE__); //进行编译的源文件
    printf("%d\n", __LINE__); //文件当前的行号
    printf("%s\n", __DATE__); //文件被编译的日期
    printf("%s\n", __TIME__); //文件被编译的时间
    //printf("%s\n", __STDC__); //VS环境下未定义__STDC__，说明Visual Studio并未完全
    遵循ANSI C
    printf("%s\n", __func__); //main
    return 0;
}
```

```
int main() {
    printf("%s\n", __FILE__); //进行编译的源文件
    printf("%d\n", __LINE__); //文件当前的行号
    printf("%s\n", __DATE__); //文件被编译的日期
    printf("%s\n", __TIME__); //文件被编译的时间
    //printf("%s\n", __STDC__); //VS环境下未定义__STDC__，说明Visual Studio并未完全遵循ANSI C
    printf("%s\n", __func__); //main
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
E:\Github_Gitee\C\Define\test.c
17
Oct 14 2022
18:51:47
main

E:\Github_Gitee\C\Define\x64\Debug\Define.exe (进程 17528)已退出，代码为 0。
按任意键关闭此窗口。 . . _
```

CSDN @期邈云汉

VS环境下未定义STDC，说明Visual Studio并未完全遵循ANSI C。



#define



#define 定义标识符

```
#define name stuff //名称；内容
```

```
#define MAX 1000
```

```

#define reg register          //为 register这个关键字，创建一个简短的名字
#define do_forever for(;;)    //用更形象的符号来替换一种实现
#define CASE break;case      //在写case语句的时候自动把 break写上。
// 如果定义的 stuff过长，可以分成几行写，除了最后一行外，每行的后面都加一个反斜杠(续行符)。
#define DEBUG_PRINT printf("file:%s\tline:%d\t \
                        date:%s\ttime:%s\n" ,\
__FILE__,__LINE__ ,      \
__DATE__,__TIME__ )

int main() {
    printf("%d\n", MAX);
    printf("%s\n", STR);
    do_forever; //执行到这里会死循环
    return 0;
}

```

在define进行定义时，最好不要在后面加上分号 `;`，替换时也会将分号替换，容易导致问题。

```

#define 1000;
int max = 0;
if (3 > 1)
    max = MAX;
else
    max = 0;
//报错

```

★ #define 定义宏

#define 机制包括了一个规定，允许把参数替换到文本中，这种实现通常称为宏（macro）或定义宏（define macro）

```
#define name( parament-list ) stuff
```

其中的 parament-list 是一个由逗号隔开的符号表，它们可能出现在stuff中。

```

//int Max(int x, int y) {
//    return (x > y ? x : y);
//}
//若采用宏定义
#define Max(x,y) (x>y?x:y)
int main() {
    int a = 10;
    int b = 5;
    int max = Max(a, b);
    printf("%d\n", max);
    return 0;
}

```

最终得到较大值10。找到了宏的标识，直接进行符号替换。

- 参数列表的左括号必须与name紧邻。

- 如果两者之间有任何空白存在，参数列表就会被解释为stuff的一部分

例如：

```
#define SQUARE(x) x*x

int main() {
    printf("%d\n", SQUARE(5));
    printf("%d\n", SQUARE(5 + 1)); //宏的参数是完全替换的，相当于 5 + 1 * 5 + 1 结果为11
    return 0;
}
```

做改进：

```
#define SQUARE(x) (x)*(x) //这样就不容易出错
//最好最外层也加上括号
```

考虑如下计算：

```
#define Double(x) (x)+(x)

int main() {
    int a = 5;
    printf("%d\n", (10 * Double(a))); //这里相当于 10 * (5) + (5) 结果为55
    return 0;
}
```

在宏替换内容上加上括号以保证得到预期的结果。

```
#define Double(x) ((x)+(x))
```

```
//#define Double(x) (x)+(x)
#define Double(x) ((x)+(x))
int main() {
    int a = 5;
    printf("%d\n", (10 * Double(a))); //这里相当于 10 * (5) + (5) 结果为55
    return 0;
}
```

Microsoft Visual Studio 调试控制台

100

E:\Github_Gitee\C\Define\x64\Debug\Define.exe (进程 10924) 已退出，代码为 0。
按任意键关闭此窗口。 . . _

CSDN @期邈云汉

所以用于对数值表达式进行求值的宏定义都应该用这种方式加上括号，避免在使用宏时由于参数中的操作符或邻近操作符之间不可预料的相互作用。

✳ 替换规则

在程序中扩展#define定义符号和宏时，需要涉及几个步骤。

1. 在调用宏时，首先对参数进行检查，看看是否包含任何由#define定义的符号。如果是，它们首先

被替换。

2. 替换文本随后被插入到程序中原来文本的位置。对于宏，参数名被他们的值所替换。
3. 最后，再次对结果文件进行扫描，看看它是否包含任何由#define定义的符号。如果是，就重复上
述处理过程。

注意：

1. 宏参数和#define 定义中可以出现其他#define定义的符号。但是对于宏，不能出现递归。
2. 当预处理器搜索#define定义的符号的时候，字符串常量的内容并不被搜索。

★ # 和

如何把参数插入到字符串中？

```
#define PRINT(FORMAT, VALUE) printf("the value of " #VALUE " is "FORMAT "\n",  
VALUE)  
  
//这里#可以把其后面的参数替换为字符串嵌入  
  
int main() {  
    printf("hello world\n");  
    printf("hello ""world\n"); //天然合为一个字符串(发现字符串是有自动连接的特点的)  
    int a = 100;  
    printf("The value a is %d\n", a);  
    int b = 20;  
    printf("The value b is %d\n", b);  
  
    //那么考虑能否将同一个功能的打印操作合并  
    PRINT("%d", a); //这里只有当字符串作为宏参数的时候才可以把字符串放在字符串中  
    /*代码中的 #VALUE 会预处理器处理为:  
       "VALUE" .*/  
  
    return 0;  
}
```

把宏对应的参数替换为字符串

```

#define PRINT(FORMAT, VALUE) printf("the value of " #VALUE " is "FORMAT "\n", VALUE)
//这里#可以把其后面的参数替换为字符串嵌入

int main() {
    printf("hello world\n");
    printf("hello ""world\n"); //天然合为一个字符串(发现字符串是有自动连接的特点的)
    int a = 100;
    printf("The value a is %d\n", a);
    int b = 20;
    printf("The value b is %d\n", b);

    //那么考虑能否将同一个功能的打印操作合并
    PRINT("%d", a); //这里只有当字符串作为宏参数的时候才可以把字符串放在字符串中
    /*代码中的 #VALUE 会预处理器处理为:
    |   "VALUE" .*/

    return 0;
}

```

Microsoft Visual Studio 调试控制台

```

hello world
hello world
The value a is 100
The value b is 20
the value of a is 100

```

E:\Github Gitee\C\Define\x64\Debug\Define.exe (进程 8364) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

CSDN @期邈云汉

##的作用

将分离的片段合成为一个符号

```

#define CAT(A,B) A##B
//把A和B组合成一个符号
int main() {
    int windows11 = 2022;

    printf("%d\n", CAT(Windows, 11));

}

```

```

#define CAT(A,B) A##B
//把A和B组合成一个符号
int main() {
    int Windows11 = 2022;

    printf("%d\n", CAT(Windows, 11));

}

```

Microsoft Visual Studio 调试控制台

```
2022
```

E:\Github Gitee\C\Define\x64\Debug\Define.exe (进程 16632) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

CSDN @期邈云汉

★带副作用的宏参数

当宏参数在宏的定义中出现超过一次的时候，如果参数带有副作用，那么你在使用这个宏的时候就可

能出现危险，导致不可预测的后果。副作用就是表达式求值的时候出现的永久性效果。

```
x+1; //不带副作用
x++; //带有副作用
```

考虑宏：

```
//考虑宏的副作用
#define MAX(a, b) ( (a) > (b) ? (a) : (b) )
int main() {
    int x = 5;
    int y = 8;
    int z = MAX(x++, y++);
    printf("x=%d y=%d z=%d\n", x, y, z);
    //z = ((x++) > (y++) ? (x++) : (y++)); x = 6 y = 10 z = 9
}
```

```
//考虑宏的副作用
#define MAX(a, b) ( (a) > (b) ? (a) : (b) )
int main() {
    int x = 5;
    int y = 8;
    int z = MAX(x++, y++);
    printf("x=%d y=%d z=%d\n", x, y, z);
    //z = ((x++) > (y++) ? (x++) : (y++)); x = 6 y = 10 z = 9
}
```

Microsoft Visual Studio 调试控制台

x=6 y=10 z=9

E:\Github_Gitee\C\Define\x64\Debug\Define.exe (进程 15172) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

CSDN @期邈云汉

宏和函数的对比

宏通常被应用于执行简单的运算

```
#define MAX(a, b) ((a)>(b)?(a):(b))
```

那为什么不用函数来完成这个任务？

原因如下：

1. 用于调用函数和从函数返回的代码可能比实际执行这个小型计算工作所需要的时间更多。
所以宏比函数在程序的规模和速度方面更胜一筹。因为宏在预处理是直接替换符号的。
2. 更为重要的是函数的参数必须声明为特定的类型。显然，宏是不需要类型检查的。
所以函数只能在类型合适的表达式上使用。反之宏可以适用于整形、长整型、浮点型等可以用于>来比较的类型。
宏是类型无关的。

宏的缺点：

1. 每次使用宏的时候，一份宏定义的代码将插入到程序中。除非宏比较短，否则可能大幅度增加程序的长度。
2. 宏是无法调试的。
3. 宏由于类型无关，也就不够严谨。

4. 宏可能会带来运算符优先级的问题，导致程容易出现错。

宏有时候可以做函数做不到的事情。比如：宏的参数可以出现类型，但是函数做不到

例如考虑动态内存申请malloc:

```
#define MALLOC(num, type) (type*)malloc(num * sizeof(type))
int main() {
    int* p = (int*)malloc(10 * sizeof(int)); //申请10个整形空间
    //当需要申请不同类型的空间时
    //Malloc(10, int); //显然，函数不可以传类型，考虑宏
    MALLOC(10, int); //宏的参数可以出现类型，但是函数做不到
    return 0;
}
```

对比分析

项目	#define定义宏	函数
代码长度	每次使用时，宏代码都会被插入到程序中。除了非常小的宏之外，程序的长度会大幅度增长	函数代码只出现在一个地方，每次使用这个函数时，都调用那个地方的同一份代码
执行速度	更快	存在函数的调用和返回的额外开销，所以相对慢一些
操作符优先级	宏参数的求值是在所有周围表达式的上下文环境里，除非加上括号，否则邻近操作符的优先级可能会产生不可预料的后果，所以建议宏在书写的时候多些括号	函数参数只在函数调用时求值一次，它的结果值传递给函数。表达式的求值结果更容易预测
带有副作用的参数	参数可能被替换到宏体中的多个位置，所以带有副作用的参数求值可能会产生不可预料的结果	函数参数只在传参时求值一次，结果更容易控制
参数类型	宏的参数与类型无关，只要对参数的操作是合法的，它就可以使用于任何参数类型	函数的参数是与类型有关的，如果参数的类型不同，就需要不同的函数，即使他们执行的任务是相同的
调试	宏是不方便调试的	宏是不方便调试的
递归	宏是不能递归的	函数是可以递归的

#undef

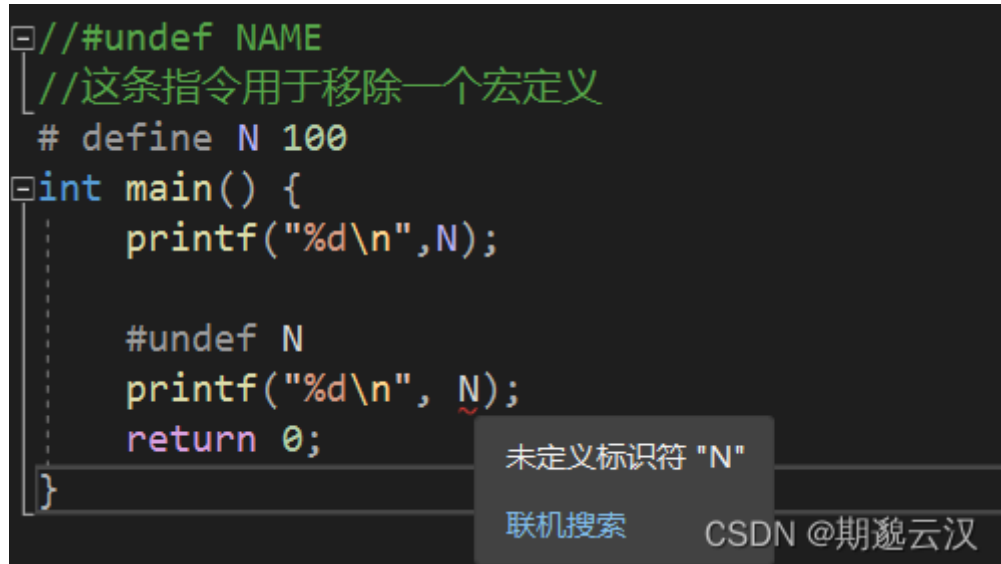
移除一个宏定义


```

//#undef NAME
//这条指令用于移除一个宏定义
# define N 100
int main() {
    printf("%d\n",N);

    #undef N
    printf("%d\n", N);
    return 0;
}

```



条件编译

许多C的编译器提供了一种能力，允许在命令行中定义符号。用于启动编译过程。

例如：当我们根据同一个源文件要编译出一个程序的不同版本的时候，这个特性有点用处。（假定某个程序中声明了一个某个长度的数组，如果机器内存有限，我们需要一个很小的数组，但是另外一个机器内存大些，我们需要一个数组能够大些。）

```

#include <stdio.h>
int main()
{
    int array [ARRAY_SIZE];
    int i = 0;
    for(i = 0; i < ARRAY_SIZE; i ++)
    {
        array[i] = i;
    }
    for(i = 0; i < ARRAY_SIZE; i ++)
    {
        printf("%d ",array[i]);
    }
    printf("\n");
    return 0;
}

```

在命令参数里我们可以为ARRAY_SIZE赋值。

编译指令：

```
//linux 环境演示  
gcc -D ARRAY_SIZE=10 programe.c
```

在编译一个程序的时候我们如果要将一条语句（一组语句）编译或者放弃是很方便的。因为我们有条件编译指令。

常用的条件编译：

```
1.  
#if 常量表达式  
//...  
#endif  
//常量表达式由预处理器求值。  
如：  
#define __DEBUG__ 1  
#if __DEBUG__  
//..  
#endif  
2. 多个分支的条件编译  
#if 常量表达式  
//...  
#elif 常量表达式  
//...  
#else  
//...  
#endif  
3. 判断是否被定义  
#if defined(symbol)  
#ifdef symbol  
#if !defined(symbol)  
#ifndef symbol  
4. 嵌套指令  
#if defined(OS_UNIX)  
#ifdef OPTION1  
    unix_version_option1();  
#endif  
#ifdef OPTION2  
    unix_version_option2();  
#endif  
#elif defined(OS_MSDOS)  
#ifdef OPTION2  
    msdos_version_option2();  
#endif  
#endif
```

```
//条件编译
int main() {
    #if 1 //条件为真, 参与编译
        printf("可以编译\n");
    #endif
    return 0;
}
```



```
//条件编译
int main() {
    #if 1 //条件为真, 参与编译
        printf("可以编译\n");
    #endif
    return 0;
}
```

Microsoft Visual Studio 调试控制台

可以编译

CSDN @期邈云汉


多分支条件编译

```
int main() {
    #if 3>2
        printf("成立\n");
    #elif 2>5
        printf("2>5\n");
    #else
        printf("都不成立\n");
    #endif

    return 0;
}
```

//多分支条件编译

```
int main() {  
#if 3>2  
    printf("成立\n");  
#elif 2>5  
    printf("2>5\n");  
#else  
    printf("都不成立\n");  
#endif  
  
    return 0;  
}
```

 Microsoft Visual Studio 调试控制台

成立

CSDN @期邈云汉

判断是否被定义

```
int main() {  
#if defined MAX  
    printf("MAX 已被定义\n");  
#endif  
  
#ifdef MAX  
    printf("MAX 已被定义\n");  
#endif  
//两者完全等价  
  
//下面两种也是完全等价  
#if !defined M  
    printf("M 未定义\n");  
#endif  
  
#ifndef M  
    printf("M 未定义\n");  
#endif  
  
    return 0;  
}
```

```

//判断是否被定义
//#if defined(symbol)
//#ifdef symbol

//#if !defined(symbol)
//#ifndef symbol
int main() {
#if defined MAX
    printf("MAX 已被定义\n");
#endif

#ifdef MAX
    printf("MAX 已被定义\n");
#endif
//两者完全等价

//下面两种也是完全等价
#if !defined M
    printf("M 未定义\n");
#endif

#ifndef M
    printf("M 未定义\n");
#endif
}

```

Microsoft Visual Studio 调试控制台

```

MAX 已被定义
MAX 已被定义
M 未定义
M 未定义

```

E:\Github_Gitee\C\Define\x64\Debug\Define.
按任意键关闭此窗口. . .

CSDN @期邈云汉



文件包含

`#include` 指令可以使另外一个文件被编译。就像它实际出现于 `#include` 指令的地方一样。

这种替换的方式很简单：

预处理器先删除这条指令，并用包含文件的内容替换。

这样一个源文件被包含10次，那就实际被编译10次。要防止代码冗余包含。

- 本地文件包含

```
#include "filename"
```

查找策略：先在源文件所在目录下查找，如果该头文件未找到，编译器就像查找库函数头文件一样在标准位置查找头文件。

如果找不到就提示编译错误

Linux环境的标准头文件的路径：

```
/usr/include
```

VS环境的标准头文件的路径：

```
C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include
//这是VS2013的默认路径
```

可以在自己的安装目录下查找，或者使用文件搜索工具[Everything](#)来查找。

- 库文件包含

```
#include <filename.h>
```

查找头文件直接去标准路径下去查找，如果找不到就提示编译错误。

对于库文件也是可以使用“”的形式包含

但是这样做查找的效率就低些，当然这样也不容易区分是库文件还是本地文件了

★ 嵌套文件包含

单个头文件被多个层次的文件反复包含，那么希望一个头文件只被包含一次，可以使用条件编译。

解决头文件重复包含

```
#ifndef __TEST_H__    //未定义，那么就定义该头文件
#define __TEST_H__
//头文件的内容
#endif    //__TEST_H__
```

第一次未定义，那么就定义该头文件，包含完成到第二次要包含时第一行不成立，直接失败，后续就不会再执行。

或者使用预处理指令：

```
#pragma once
```

以避免头文件的重复引入。

考虑问题：

1. 头文件中的 `ifndef/define/endif` 是干什么用的？ 防止重复包含，造成冗余
2. `#include <filename.h>` 和 `#include "filename.h"` 有什么区别？ `<>` 直接去库目录查找；`""` 包含查找两次

==An==