C 预处理器 & 宏

Serendipity 已于 2023-10-14 22:29:50 修改

目录

一.C 预处理器

预处理器不是 编译器 的组成部分，但是它是编译过程中一个单独的步骤。简言之，C 预处理器只不过是一个文本替换工具而已，它们会指示编译器在实际编译之前完成所需的预处理。我们将把 C 预处理器（C Preprocessor）简写为 CPP。

1.1 预处理器指令

下面列出了所有重要的预处理器指令：

指令	描述
#define	定义宏
#include	包含一个源代码文件
#undef	取消已定义的宏
#ifdef	如果宏已经定义，则返回真
#ifndef	如果宏没有定义，则返回真
#if	如果给定条件为真，则编译下面代码
#else	#if 的替代方案
#elif	如果前面的 #if 给定条件不为真，当前条件为真，则编译下面代码
#endif	结束一个 #if……#else 条件编译块
#error	当遇到标准错误时，输出错误消息
#pragma	使用标准化方法，向编译器发布特殊的命令到编译器中

如果指定的标识符已定义，则值为真（非零）。如果指定的标识符未定义，则值为假（零）。

1.2 预处理器实例

```
//define定义标识符常量
#define MAX 1000

//这些指令告诉 CPP 从系统库中获取 stdio.h，并添加文本到当前的源文件中。下一行告诉 CPP 从本地目录中获取 myheader.h，并添加内容到当前的源文件中。
#include <stdio.h>
```

```c
#include "myheader.h"

//这个指令告诉 CPP 取消已定义的 FILE_SIZE，并定义它为 42。
#undef  FILE_SIZE
#define FILE_SIZE 42

//这个指令告诉 CPP 只有当 MESSAGE 未定义时，才定义 MESSAGE。
#ifndef MESSAGE
   #define MESSAGE "You wish!"
#endif

//这个指令告诉 CPP 如果定义了 DEBUG，则执行处理语句。在编译时，如果您向 gcc 编译器传递了 -DDEBUG 开关量，这个指令就非常有用。它定义了 DEBUG，您可以在编译期间随时
#ifdef DEBUG
   /* Your debugging statements here */
#endif
```

1.3 预定义宏

宏	描述
__DATE__	当前日期，一个以 "MMM DD YYYY" 格式表示的字符常量。
__TIME__	当前时间，一个以 "HH:MM:SS" 格式表示的字符常量。
__FILE__	这会包含当前文件名，一个字符串常量。
__LINE__	这会包含当前行号，一个十进制常量。
__STDC__	当编译器以 ANSI 标准编译时，则定义为 1。

让我们来尝试下面的实例：

```c
#include <stdio.h>
```

```
int main()
{
    printf("File :%s\n", __FILE__ );
    printf("Date :%s\n", __DATE__ );
    printf("Time :%s\n", __TIME__ );
    printf("Line :%d\n", __LINE__ );
    printf("ANSI :%d\n", __STDC__ );
    return 0;
}

//输出结果
File :test.c
Date :Jun 2 2012
Time :03:36:24
Line :8
ANSI :1
```

1.4 预处理器运算符

1.4.1 宏延续运算符（\）

一个宏通常写在一个单行上。但是如果宏太长，一个单行容纳不下，则使用宏延续运算符（\）。

```
#define  message_for(a, b)  \
    printf(#a " and " #b ": We love you!\n")
```


1.4.2 字符串常量化运算符（#）

在宏定义中，当需要把一个宏的参数转换为字符串常量时，则使用字符串常量化运算符（#）。在宏中使用的该运算符有一个特定的参数或参数列表。

```
#include <stdio.h>

#define  message_for(a, b)  \
```

```
    printf(#a " and " #b ": We love you!\n")
int main(void)
{
   message_for(Carole, Debra);
   return 0;
}

//输出结果：Carole and Debra: We love you!
```

1.4.3 标记粘贴运算符（##）

宏定义内的标记粘贴运算符（##）会合并两个参数。它允许在宏定义中两个独立的标记被合并为一个标记。

```
#include <stdio.h>

#define tokenpaster(n) printf ("token" #n " = %d", token##n)

int main(void)
{
   int token34 = 40;

   tokenpaster(34);
   return 0;
}

//输出结果：token34 = 40
```

二.#define 定义常数和宏

2.1 #define定义常数

```
#define 标识符 值
```

这个值可以是数值、字符或者表达式。

2.2 #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或定义宏。

2.2.1 宏（带参数）的声明

```
#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。
注意：
1.参数列表的左括号必须与name紧邻。
2.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。

例子：如果你在程序中定义下列的宏

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

如果此时你的程序中出现：

```
SQUARE( 5 );
```

则#define会用5 * 5代替上列的语句。

2.2.2 不规范警告！！！

```
1 int num = 5;
2 printf("%d\n" ,SQUARE( num + 2) );
```


这个代码是不是一眼就觉得答案应该是49

其实不然，答案应该是17。很简单，原因如下：

首先宏的使用本质上是替换，所以输出语句实际上是：

```
printf("%d\n" ,num + 2 * num + 2 );
```

由此我们可以看出，宏的使用纯纯就是替换，那解决方案就很简单了，如下：（只需要加一个括号就行了）

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

2.2.3 新例巩固

```
#define DOUBLE(x) (x) + (x)
```

这次定义的时候我们就加上了括号对吧，但是这个定义实际上还是有些许小问题。

```
1 int x = 5;
2 printf("%d\n" ,10 * DOUBLE(x));
```

看到这个代码，你会不会认为答案是100，但实际的输出结果只有55，这是因为替换后变成了：

```
printf("%d\n" ,10 * 5 + 5);
```

可以看到跟数学一样，乘除的优先级是要大于加减的，但是同样可以通过加括号解决：

```
#define DOUBLE(x) ((x) + (x))
```

综上所述：我们在定义宏的时候，需要通过加括号来让代码满足我们的需求。