C 语言宏定义

宏定义是预处理命令的一种，它允许用一个标识符来表示一个字符串。先看一个例子：

```
#include <stdio.h>

#define N 100

int main(){
    int sum = 20 + N;
    printf("%d\n", sum);
    return 0;
}
```

运行结果：

```
120
```

该示例中的语句 `int sum = 20 + N;`，`N` 被 `100` 代替了。

`#define N 100` 就是宏定义，`N` 为宏名，`100` 是宏的内容。在编译预处理时，对程序中所有出现的“宏名”，都用宏定义中的字符串去代换，这称为“宏代换”或“宏展开”。

宏定义是由源程序中的宏定义命令`#define` 完成的，宏代换是由预处理程序完成的。

宏定义的一般形式为：

```
#define    宏名    字符串
```

`#`表示这是一条预处理命令，所有的预处理命令都以#开头。`define` 是预处理命令。`宏名`是标识符的一种，命名规则和标识符相同。`字符串`可以是常数、表达式等。

这里所说的字符串是一般意义上的字符序列，不要和 C 语言中的字符串等同，它不需要双引号。

程序中反复使用的表达式就可以使用宏定义，例如：

```
#define M (n*n+3*n)
```

它的作用是指定标识符 M 来代替表达式(y*y+3*y)。在编写源程序时，所有的(y*y+3*y)都可由 M 代替，而对源程序编译时，将先由预处理程序进行宏代换，即用(y*y+3*y)表达式去替换所有的宏名 M，然后再进行编译。

将上面的例子补充完整：

```
1. #include <stdio.h>
2. 
3. #define M (n*n+3*n)
4. 
5. int main(){
6.     int sum, n;
7.     printf("Input a number: ");
8.     scanf("%d", &n);
9.     sum = 3*M+4*M+5*M;
10.    printf("sum=%d\n", n);
11.    return 0;
12.}
```

运行结果：

```
Input a number: 10↙
sum=1560
```

上面的程序中首先进行宏定义，定义 M 来替代表达式(n*n+3*n)，在 sum=3*M+4*M+5*M 中作了宏调用。在预处理时经宏展开后该语句变为：

```
sum=3*(n*n+3*n)+4*(n*n+3*n)+5*(n*n+3*n);
```

需要注意的是，在宏定义中表达式(n*n+3*n)两边的括号不能少，否则会发生错误。

如当作以下定义后：

```
#difine M n*n+3*n
```

在宏展开时将得到下述语句：

```
s=3*n*n+3*n+4*n*n+3*n+5*n*n+3*n;
```

这相当于：

$3n^2+3n+4n^2+3n+5n^2+3n$

这显然是不正确的。所以进行宏定义时要注意，应该保证在宏代换之后不发生错误。

对宏定义的几点说明

1) 宏定义是用宏名来表示一个字符串，在宏展开时又以该字符串取代宏名，这只是一种简单的替换。字符串中可以含任何字符，可以是常数，也可以是表达式，预处理程序对它不作任何检查，如有错误，只能在编译已被宏展开后的源程序时发现。

2) 宏定义不是说明或语句，在行末不必加分号，如加上分号则连分号也一起替换。

3) 宏定义必须写在函数之外，其作用域为宏定义命令起到源程序结束。如要终止其作用域可使用`#undef`命令。例如：

```
#define PI 3.14159

int main(){
    // Code
    return 0;
}

#undef PI

void func(){
    // Code
}
```

表示 PI 只在 main 函数中有效，在 func 中无效。

4) 宏名在源程序中若用引号括起来，则预处理程序不对其作宏代换，例如：

```
#include <stdio.h>
#define OK 100
int main(){
    printf("OK\n");
```

```
5.      return 0;
6.  }
```

运行结果：

OK

该例中定义宏名 OK 表示 100，但在 printf 语句中 OK 被引号括起来，因此不作宏代换，而作为字符串处理。

5) 宏定义允许嵌套，在宏定义的字符串中可以使用已经定义的宏名，在宏展开时由预处理程序层层代换。例如：

```
#define PI 3.1415926
#define S PI*y*y     /* PI 是已定义的宏名*/
```

对语句：

```
printf("%f", S);
```

在宏代换后变为：

```
printf("%f", 3.1415926*y*y);
```

6) 习惯上宏名用大写字母表示，以便于与变量区别。但也允许用小写字母。

7) 可用宏定义表示数据类型，使书写方便。例如：

```
#define UINT unsigned int
```

在程序中可用 UINT 作变量说明：

```
UINT a, b;
```

应注意用宏定义表示数据类型和用 typedef 定义数据说明符的区别。宏定义只是简单的字符串代换，是在预处理完成的，而 typedef 是在编译时处理的，它不是作简单的代换，而是对类型说明符重新命名。被命名的标识符具有类型定义说明的功能。

请看下面的例子：

```
#define PIN1 int *
```

```
typedef (int *) PIN2;
```

从形式上看这两者相似， 但在实际使用中却不相同。

下面用 PIN1，PIN2 说明变量时就可以看出它们的区别：

```
PIN1 a,b;
```

在宏代换后变成：

```
int *a,b;
```

表示 a 是指向整型的指针变量，而 b 是整型变量。然而：

```
PIN2 a,b;
```

表示 a、b 都是指向整型的指针变量。因为 PIN2 是一个类型说明符。由这个例子可见，宏定义虽然也可表示数据类型， 但毕竟是作字符代换。在使用时要分外小心，以避出错。