C 语言中 define 的用法

define 是 C 语言中的预处理命令，它用于宏定义，可以提高源代码的可读性，为编程提供方便。

预处理命令以 “#” 号开头，如包含命令 #include，宏定义命令 #define 等。一般都放在源文件的前面，它们称为预处理部分。

所谓预处理是指在进行编译之前所作的工作。预处理是 C 语言的一个重要功能，它由预处理程序负责完成。当对一个源文件进行编译时，系统将自动引用预处理程序对源程序中的预处理部分作处理，处理完毕自动进入对源程序的编译。

宏的定义

在 C 或 C++ 语言源程序中允许用一个标识符来表示一个字符串，称为 “宏”。被定义为“宏” 的标识符称为 “宏名”。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完成的。宏代换是由预处理程序自动完成的。

在 C 或 C++ 语言中，“宏” 分为有参数和无参数两种。

无参数宏定义

无参数宏就是不带参数，其定义的一般形式为：

    #define 标识符 字符串

    “标识符” 为所定义的宏名。“字符串” 可以是常数、表达式、格式串等。

    例如：

    #define PI 3.14

它的作用是指定标识符 PI 来代替常数 3.14。在编写源程序时，所有用到 3.14 的地方都可用 PI 代替，而对源程序作编译时，将先由预处理程序进行宏代换，即用 3.14 去置换所有的宏名 PI，然后再进行编译。

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，这只是一种简单的代换，字符串可以是常数，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。

宏定义不是说明或语句 (它是预处理指令)，在行末不必加分号，如加上分号则连分号也一起置换。

下面举一个无参数宏替代常数的例子：

```
#define PI 3.14
#include <stdio.h>
int main()
{
float r = 1.0;
float area = PI*r*r;
printf("The area of the circle is %f",area);
return 0;
}
```

再举一个使用无参数宏替代字符串的例子：

```
#define M (y*y+3*y)
#include <stdio.h>
int main()
{
int s,y;
printf("input a number:");
scanf("%d",&y);
s = 3*M + 4*M + 5*M;
printf("s=%d\n",s);
```

```
return 0;
}
```

define M (y*y+3*y) 定义 M 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，而对源程序作编译时，将先由预处理程序进行宏代换，即用 (y*y+3*y)表达式去置换所有的宏名 M，然后再进行编译。

上例程序中首先进行宏定义，定义 M 表达式 (y*y+3*y), 在 s=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：s=3*(y*y+3*y)+4* (y*y+3*y)+5* (y*y+3*y); 但要注意的是，在宏定义中表达式(y*y+3*y) 两边的括号不能少。否则会发生错误。

带参数宏定义

C 语言允许宏带有参数。在宏定义中的参数称为形式参数，在宏调用中的参数称为实际参数。对带参数的宏，在调用中，不仅要宏展开，而且要用实参去代换形参。

带参数宏定义的一般形式为：

#define 宏名 (形参表) 字符串

在字符串中含有各个形参。

带参数宏调用的一般形式为：

宏名 (实参表)

例如：

#define M(y) y*y+3*y

....

k=M(5);

....

在宏调用时，用实参 5 去代替形参 y，经预处理宏展开后的语句为:

k=5*5+3*5

举一个具体例子：

```
#define MAX(a,b) (a>b)?a:b
#include <stdio.h>
int main()
{
int x,y,max;
printf("input two numbers:");
scanf("%d%d",&x,&y);
max = MAX(x,y);
printf("max=%d\n",max);
return 0;
}
```

上例程序的第一行进行带参数宏定义，用宏名 MAX 表示条件表达式 (a>b)?a:b，形参 a,b 均出现在条件表达式中。程序第七行 max = MAX(x, y) 为宏调用，实参 x,y，将代换形参 a,b。宏展开后该语句为： max = (x>y)?x:y; 用于计算 x,y 中的大数。

对于带参的宏定义有以下问题需要说明：

1. 带参宏定义中，宏名和形参表之间不能有空格出现。

例如把： #define MAX(a,b) (a>b)?a:b 写为： #define MAX (a,b) (a>b)?a:b 将被认为是无参宏定义，宏名 MAX 代表字符串 (a,b)(a>b)?a:b。

宏展开时，宏调用语句： max = MAX(x,y); 将变为： max = (a,b)(a>b)?a:b(x,y); 这显然是错误的。

2. 在宏定义中的形参是标识符，而宏调用中的实参可以是表达式。

```
#define SQ(y) (y)*(y)
#include <stdio.h>
int main()
{
int a,sq;
printf("input a number:");
scanf("%d",&a);
sq=SQ(a+1);
printf("sq=%d\n",sq);
return 0;
}
```

上例中第一行为宏定义，形参为 y。程序第七行宏调用中实参为 a+1，是一个表达式，在宏展开时，用 a+1 代换 y，再用 (y)*(y) 代换 SQ，得到如下语句： sq=(a+1)*(a+1); 这与函数的调用是不同的，函数调用时要把实参表达式的值求出来再赋予形参。而宏代换中对实参表达式不作计算直接地照原样代换。

3. 在宏定义中，字符串内的形参通常要用括号括起来以避免出错。在上例中的宏定义中 (y)*(y) 表达式的 y 都用括号括起来，结果是正确的。如果去掉括号，把程序改为以下形式：

```
#define SQ(y) y*y
#include <stdio.h>
int main()
{
int a,sq;
printf("input a number:");
scanf("%d",&a);
sq=SQ(a+1);
printf("sq=%d\n",sq);
return 0;
}
```

运行结果为：input a number:3

sq=7（我们期望的结果却是 16）。

问题在哪里呢? 这是由于代换只作符号代换而不作其它处理而造成的。 宏代换后将得到以下语句： sq=a+1*a+1; 由于 a 为 3 故 sq 的值为 7。这显然与题意相违，因此参数两边的括号是不能少的。有时候，即使在参数两边加括号还是不够的，请看下面程序：

```
#define SQ(y) (y)*(y)
#include <stdio.h>
int main()
{
int a,sq;
printf("input a number:");
scanf("%d",&a);
sq=160/SQ(a+1);
printf("sq=%d\n",sq);
return 0;
}
```

本程序与前例相比，只把宏调用语句改为： sq=160/SQ(a+1); 运行本程序如输入值仍为 3 时，希望结果为 10。但实际运行的结果如下：input a number:3　sq=160。

为什么会得这样的结果呢? 分析宏调用语句，在宏代换之后变为： sq=160/(a+1)*(a+1);a 为 3 时，由于 “/” 和“*”运算符优先级和结合性相同，则先作 160/(3+1)得 40，再作 40*(3+1)最后得 160。为了得到正确答案应在宏定义中的整个字符串外加括号，程序修改如下：

```
#define SQ(y) ((y)*(y))
#include <stdio.h>
int main()
{
int a,sq;
printf("input a number:");
scanf("%d",&a);
```

```
sq=160/SQ(a+1);
printf("sq=%d\n",sq);
return 0;
}
```

以上讨论说明，对于宏定义，保险的做法是不仅应在参数两侧加括号，也应在整个字符串外加括号。

4. 带参数的宏和带参函数很相似，但有本质上的不同，把同一表达式用函数处理与用宏处理两者的结果有可能是不同的。

下面举一个例子进行对比：

使用函数：

```
#include <stdio.h>
int SQ(int);
int main()
{
int i=1;
while(i<=5)
printf("%d\n",SQ(i++));
return 0;
}
int SQ(int y)
{
return((y)*(y));
}
```

使用宏：

```
#define SQ(y) ((y)*(y))
#include <stdio.h>
int main()
{
int i=1;
```

```
while(i<=5)
printf("%d\n",SQ(i++));
return 0;
}
```

在使用函数的例子中函数名为 SQ，形参为 Y，函数体表达式为 ((y)*(y))。在使用宏的例子中宏名为 SQ，形参也为 y，字符串表达式为 (y)*(y))。两例表面是相同的，函数调用为 SQ(i++)，宏调用为 SQ(i++)，实参也是相同的。但输出结果却大不相同，分析如下：

在使用函数的例子中，函数调用是把实参 i 值传给形参 y 后自增 1。然后输出函数值。因而要循环 5 次。输出 1～5 的平方值。而在使用宏的例子中，宏调用时，只作代换。SQ(i++) 被代换为 ((i++)*(i++))。在第一次循环时，由于 i 等于 1，其计算过程为：表达式中前一个 i 初值为 1，然后 i 自增 1 变为 2，因此表达式中第 2 个 i 初值为 2，两相乘的结果也为 2，然后 i 值再自增 1，得 3。在第二次循环时，i 值已有初值为 3，因此表达式中前一个 i 为 3，后一个 i 为 4，乘积为 12，然后 i 再自增 1 变为 5。进入第三次循环，由于 i 值已为 5，所以这将是最后一次循环。计算表达式的值为 5*6 等于 30。i 值再自增 1 变为 6，不再满足循环条件，停止循环。从以上分析可以看出函数调用和宏调用二者在形式上相似，在本质上是完全不同的。

"\","#","#@" 和 "##"

在用 #define 定义时 , 斜杠 ("\") 是用来续行的，"#" 用来把参数转换成字符串，是给参数加上双引号。"##" 则用来连接前后两个参数，把它们变成一个字符串,"#@" 是给参数加上单引号。下面的例子会使您很容易理解。

```
#define Conn(x,y) x##y
#define ToChar(a)  #@a
#define ToString(x) #x
int n = Conn(123,456); 结果就是 n=123456;
```

> char* str = Conn("asdf", "adf") 结果就是 str = "asdfadf";
> char a = ToChar(1); 结果就是 a='1';
> char* str = ToString(123132); 就成了 str="123132";

为什么需要”#”,” #@” 和”##” 这三个操作符呢？原因如下：

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。如下：

```
#define OK 100
#include <stdio.h>
int main()
{
printf("OK");
printf("\n");
return 0;
}
```

上例中定义宏名 OK 表示 100，但在 printf 语句中 OK 被引号括起来，因此不作宏代换。程序的运行结果为：OK, 这表示把“OK” 当字符串处理。

同样，宏名在源程序中若用单引号括起来，则预处理程序也不对其作宏代换。

宏定义的嵌套

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
对语句： printf("%f",s);
在宏代换后变为： printf("%f",3.1415926*y*y);
```

结束语

使用宏代替一个在程序中经常使用的常量，这样该常量改变时，不用对整个程序进行修改，只修改宏定义的字符串即可，而且当常量比较长时，我们可以用较短的有意义的标识符来写程序，这样更方便一些。举一个大家比较熟悉的例子，圆周率π是在数学上常用的一个值，有时我们会用 3.14 来表示，有时也会用 3.1415926 等，这要看计算所需要的精度，如果我们编制的一个程序中要多次使用它，那么需要确定一个数值，在本次运行中不改变，但也许后来发现程序所表现的精度有变化，需要改变它的值，这就需要修改程序中所有的相关数值，这会给我们带来一定的不便，但如果使用宏定义，使用一个标识符来代替，则在修改时只修改宏定义即可，还可以减少输入 3.1415926 这样长的数值多次的情况，我们可以如此定义 #define pi 3.1415926，既减少了输入又便于修改，何乐而不为呢？

另外，使用带参数的宏定义可完成函数调用的功能，又能减少系统开销，提高运行效率。正如 C 语言中所讲，函数的使用可以使程序更加模块化，便于组织，而且可重复利用，但在发生函数调用时，需要保留调用函数的现场，以便子函数执行结束后能返回继续执行，同样在子函数执行完后要恢复调用函数的现场，这都需要一定的时间，如果子函数执行的操作比较多，这种转换时间开销可以忽略，但如果子函数完成的功能比较少，甚至于只完成一点操作，如一个乘法语句的操作，则这部分转换开销就相对较大了，但使用带参数的宏定义就不会出现这个问题，因为它是在预处理阶段即进行了宏展开，在执行时不需要转换，即在当地执行。宏定义可完成简单的操作，但复杂的操作还是要由函数调用来完成，而且宏定义所占用的目标代码空间相对较大。所以在使用时要依据具体情况来决定是否使用宏定义。

全文完