9 预处理命令

1.1 概述

在前面各章中，已多次使用过以“#”号开头的预处理命令。如包含命令#include，宏定义命令#define 等。在源程序中这些命令都放在函数之外，而且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的第一遍扫描(词法扫描和语法分析)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。

C语言提供了多种预处理功能，如宏定义、文件包含、条件编译等。合理地使用预处理功能编写的程序便于阅读、修改、移植和调试，也有利于模块化程序设计。本章介绍常用的几种预处理功能。

1.1 宏定义

在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语言中，“宏”分为有参数和无参数两种。下面分别讨论这两种“宏”的定义和调用。

1.1.1 无参宏定义

无参宏的宏名后不带参数。

其定义的一般形式为：

```
#define  标识符  字符串
```

其中的“#”表示这是一条预处理命令。凡是以“#”开头的均为预处理命令。“define”为宏定义命令。“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。

在前面介绍过的符号常量的定义就是一种无参宏定义。此外，常对程序中反复使用的表达式进行宏定义。

例如：

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

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。

【例 9.1】

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

上例程序中首先进行宏定义,定义 M 来替代表达式(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。如当作以下定义后：

```
#difine M y*y+3*y
```

在宏展开时将得到下述语句：

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

这相当于：

$$3y^2+3y+4y^2+3y+5y^2+3y;$$

显然与原题意要求不符。计算结果当然是错误的。因此在作宏定义时必须十分注意。应保证在宏代换之后不发生错误。

对于宏定义还要说明以下几点：

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，这只是一种简单的代换，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置换。
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
main()
{
  ……
}
```

```
#undef PI
f1()
{
  ……
}
```

表示 PI 只在 main 函数中有效，在 f1 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 9.2】

```
#define OK 100
main()
{
  printf("OK");
  printf("\n");
}
```

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。

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 STU struct stu
```

在程序中可用 STU 作变量说明：

```
STU body[5],*p;
#define INTEGER int
```

在程序中即可用 INTEGER 作整型变量说明：

```
INTEGER 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。

8) 对“输出格式”作宏定义，可以减少书写麻烦。

【例 9.3】中就采用了这种方法。

```
#define P printf
#define D "%d\n"
#define F "%f\n"
main(){
  int a=5, c=8, e=11;
  float b=3.8, d=9.7, f=21.08;
  P(D F,a,b);
  P(D F,c,d);
  P(D F,e,f);
}
```

1.1.1 带参宏定义

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。

带参宏调用的一般形式为：

宏名(实参表);

例如：

```
#define M(y) y*y+3*y      /*宏定义*/
   ……
k=M(5);                   /*宏调用*/
   ……
```

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

【例 9.4】

```
#define MAX(a,b) (a>b)?a:b
main(){
  int x,y,max;
  printf("input two numbers:    ");
```

```
  scanf("%d%d",&x,&y);
  max=MAX(x,y);
  printf("max=%d\n",max);
}
```

上例程序的第一行进行带参宏定义，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. 在带参宏定义中，形式参数不分配内存单元，因此不必作类型定义。而宏调用中的实参有具体的值。要用它们去代换形参，因此必须作类型说明。这是与函数中的情况不同的。在函数中，形参和实参是两个不同的量，各有自己的作用域，调用时要把实参值赋予形参，进行"值传递"。而在带参宏中，只是符号代换，不存在值传递的问题。

3. 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。

【例 9.5】

```
#define SQ(y) (y)*(y)
main(){
  int a,sq;
  printf("input a number:    ");
  scanf("%d",&a);
  sq=SQ(a+1);
  printf("sq=%d\n",sq);
}
```

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。

4. 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，因此结果是正确的。如果去掉括号，把程序改为以下形式：

【例 9.6】

```
#define SQ(y) y*y
main(){
  int a,sq;
  printf("input a number:    ");
  scanf("%d",&a);
  sq=SQ(a+1);
  printf("sq=%d\n",sq);
}
```

运行结果为：

```
input a number:3
sq=7
```

同样输入 3，但结果却是不一样的。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。即使在参数两边加括号还是不够的，请看下面程序：

【例 9.7】

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

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：

【例 9.8】

```
#define SQ(y) ((y)*(y))
```

```
main(){
  int a,sq;
  printf("input a number:    ");
  scanf("%d",&a);
  sq=160/SQ(a+1);
  printf("sq=%d\n",sq);
}
```

以上讨论说明，对于宏定义不仅应在参数两侧加括号，也应在整个字符串外加括号。

5. 带参的宏和带参函数很相似，但有本质上的不同，除上面已谈到的各点外，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。

【例 9.9】

```
main(){
  int i=1;
  while(i<=5)
    printf("%d\n",SQ(i++));
}
SQ(int y)
{
  return((y)*(y));
}
```

【例 9.10】

```
#define SQ(y) ((y)*(y))
main(){
  int i=1;
  while(i<=5)
    printf("%d\n",SQ(i++));
}
```

在例 9.9 中函数名为 SQ，形参为 Y，函数体表达式为((y)*(y))。在例 9.10 中宏名为 SQ，形参也为 y，字符串表达式为(y)*(y))。 例 9.9 的函数调用为 SQ(i++)，例 9.10 的宏调用为 SQ(i++)，实参也是相同的。从输出结果来看，却大不相同。

分析如下：在例 9.9 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。而在例 9.10 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。

6. 宏定义也可用来定义多个语句，在宏调用时，把这些语句又代换到源程序内。看下面的例子。

【例 9.11】

```
#define SSSV(s1,s2,s3,v) s1=l*w;s2=l*h;s3=w*h;v=w*l*h;
main(){
  int l=3,w=4,h=5,sa,sb,sc,vv;
  SSSV(sa,sb,sc,vv);
  printf("sa=%d\nsb=%d\nsc=%d\nvv=%d\n",sa,sb,sc,vv);
}
```

程序第一行为宏定义，用宏名 SSSV 表示 4 个赋值语句，4 个形参分别为 4 个赋值符左部的变量。在宏调用时，把 4 个语句展开并用实参代替形参。使计算结果送入实参之中。

1.1 文件包含

文件包含是 C 预处理程序的另一个重要功能。

文件包含命令行的一般形式为：

```
#include"文件名"
```

在前面我们已多次用此命令包含过库函数的头文件。例如：

```
#include"stdio.h"
#include"math.h"
```

文件包含命令的功能是把指定的文件插入该命令行位置取代该命令行，从而把指定的文件和当前的源程序文件连成一个源文件。

在程序设计中，文件包含是很有用的。一个大的程序可以分为多个模块，由多个程序员分别编程。有些公用的符号常量或宏定义等可单独组成一个文件，在其它文件的开头用包含命令包含该文件即可使用。这样，可避免在每个文件开头都去书写那些公用量，从而节省时间，并减少出错。

对文件包含命令还要说明以下几点：

1. 包含命令中的文件名可以用双引号括起来，也可以用尖括号括起来。例如以下写法都是允许的：

   ```
   #include"stdio.h"
   #include<math.h>
   ```

   但是这两种形式是有区别的：使用尖括号表示在包含文件目录中去查找(包含目录是由用户在设置环境时设置的)，而不在源文件目录去查找；

   使用双引号则表示首先在当前的源文件目录中查找，若未找到才到包含目录中去查找。用户编程时可根据自己文件所在的目录来选择某一种命令形式。

2. 一个 include 命令只能指定一个被包含文件，若有多个文件要包含，则需用多个 include 命令。

3. 文件包含允许嵌套，即在一个被包含的文件中又可以包含另一个文件。

1.1 条件编译

预处理程序提供了条件编译的功能。可以按不同的条件去编译不同的程序部分，因而产生不同的目标代码文件。这对于程序的移植和调试是很有用的。

条件编译有三种形式，下面分别介绍：

1. 第一种形式：

```
#ifdef  标识符
  程序段 1
#else
  程序段 2
#endif
```

它的功能是，如果标识符已被 #define 命令定义过则对程序段 1 进行编译；否则对程序段 2 进行编译。如果没有程序段 2(它为空)，本格式中的#else 可以没有，即可以写为：

```
#ifdef  标识符
   程序段
#endif
```

【例 9.12】

```
#define NUM ok
main(){
  struct stu
  {
    int num;
    char *name;
    char sex;
    float score;
  }  *ps;
  ps=(struct stu*)malloc(sizeof(struct stu));
  ps->num=102;
  ps->name="Zhang ping";
  ps->sex='M';
  ps->score=62.5;
  #ifdef NUM
  printf("Number=%d\nScore=%f\n",ps->num,ps->score);
  #else
  printf("Name=%s\nSex=%c\n",ps->name,ps->sex);
  #endif
  free(ps);
}
```

由于在程序的第 16 行插入了条件编译预处理命令，因此要根据 NUM 是否被定义过来决定编译那一个 printf 语句。而在程序的第一行已对 NUM 作过宏定义，因此应对第一个 printf

语句作编译故运行结果是输出了学号和成绩。

在程序的第一行宏定义中，定义NUM表示字符串OK，其实也可以为任何字符串，甚至不给出任何字符串，写为：

```
#define NUM
```

也具有同样的意义。只有取消程序的第一行才会去编译第二个printf语句。读者可上机试作。

2. 第二种形式：

```
#ifndef 标识符
   程序段1
#else
   程序段2
#endif
```

与第一种形式的区别是将"ifdef"改为"ifndef"。它的功能是，如果标识符未被#define命令定义过则对程序段1进行编译，否则对程序段2进行编译。这与第一种形式的功能正相反。

3. 第三种形式：

```
#if 常量表达式
   程序段1
#else
   程序段2
#endif
```

它的功能是，如常量表达式的值为真(非0)，则对程序段1 进行编译，否则对程序段2进行编译。因此可以使程序在不同条件下，完成不同的功能。

【例9.13】

```
#define R 1
main(){
  float c,r,s;
  printf ("input a number:  ");
  scanf("%f",&c);
  #if R
    r=3.14159*c*c;
    printf("area of round is: %f\n",r);
  #else
    s=c*c;
    printf("area of square is: %f\n",s);
  #endif
}
```

本例中采用了第三种形式的条件编译。在程序第一行宏定义中，定义R为1，因此在条件编译时，常量表达式的值为真，故计算并输出圆面积。

上面介绍的条件编译当然也可以用条件语句来实现。但是用条件语句将会对整个源程序进行编译，生成的目标代码程序很长，而采用条件编译，则根据条件只编译其中的程序段1或程序段2，生成的目标程序较短。如果条件选择的程序段很长，采用条件编译的方法是十分必要的。

1.1 本章小结

1. 预处理功能是C语言特有的功能，它是在对源程序正式编译前由预处理程序完成的。程序员在程序中用预处理命令来调用这些功能。
2. 宏定义是用一个标识符来表示一个字符串，这个字符串可以是常量、变量或表达式。在宏调用中将用该字符串代换宏名。
3. 宏定义可以带有参数，宏调用时是以实参代换形参。而不是“值传送”。
4. 为了避免宏代换时发生错误，宏定义中的字符串应加括号，字符串中出现的形式参数两边也应加括号。
5. 文件包含是预处理的一个重要功能，它可用来把多个源文件连接成一个源文件进行编译，结果将生成一个目标文件。
6. 条件编译允许只编译源程序中满足条件的程序段，使生成的目标程序较短，从而减少了内存的开销并提高了程序的效率。
7. 使用预处理功能便于程序的修改、阅读、移植和调试，也便于实现模块化程序设计。