第3章 控制语句与预处理命令

CHAPTER

结构化程序有顺序、分支和循环三种结构。针对分支结构和循环结构，C语言提供了相应的语句。另外，为了便于程序的书写、阅读、修改及调试，C语言提供了编译预处理功能。

3.1 分支语句

分支结构用分支语句来实现。C语言中提供的分支语句有两种：一种是if语句，另一种是switch～case语句。

3.1.1 if语句

if语句有以下四种格式：单分支格式、双分支格式、多分支格式和嵌套格式。

1. 单分支格式

一般形式为：

```
if(表达式) 语句
```

执行过程：先计算if后面的表达式，若结果为真(非0)，执行后面的语句；若结果为假(0)，不执行该语句。其流程图如图3.1所示。

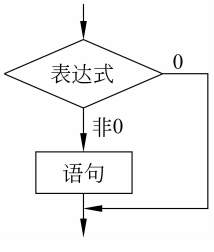

图3.1 单分支流程图

【例3.1】 输入一个整型数，输出该数的绝对值。

```
#include "stdio.h"
void main()
{ int a;
  scanf("%d",&a);
  if(a<=0)
    a=-a;
  printf("%d\n",a);
}
```

【运行结果】

```
-3↙
3
```

2. 双分支格式

一般形式为：

```
if(表达式)  语句1
else  语句2
```

执行过程：先计算 if 后面的表达式，若结果为真（非0），则执行语句1；否则执行语句2。其流程图如图3.2所示。

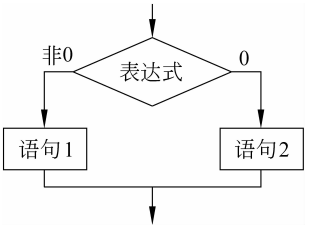

图3.2 双分支流程图

【例3.2】 输入两个整型数，将平方值较大者输出。

```
#include "stdio.h"
void main()
{ int a,b,max;
  scanf("%d%d",&a,&b);
  if(a*a>b*b)
    max=a;
  else
    max=b;
  printf("max=%d\n",max);
}
```

【运行结果】

```
2-3↙
max=-3
```

3. 多分支格式

一般形式为：

```
if(表达式1)  语句1
else  if(表达式2)  语句2
      else  if(表达式3)  语句3
                  ⋮
            else  if(表达式n)  语句n
                  else  语句n+1
```

执行过程：先计算表达式1，若表达式1的结果为真（非0），执行语句1，否则计算表达式2；若表达式2的结果为真（非0），执行语句2。以此类推，若n个表达式的结果都为假（0），则执行语句n+1。其流程图如图3.3所示。

由执行过程可知，这n+1个语句中只能有一个被执行。若n个表达式的值都为假，则执行语句n+1，否则执行第一个表达式值为真（非0）的表达式后面的语句。

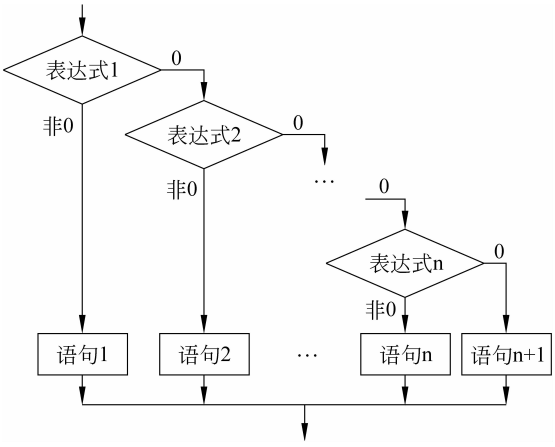

图 3.3　多分支流程图

【例 3.3】 输入一个百分制成绩，输出其对应的等级(90～100 为 A，80～99 为 B，70～79 为 C，60～69 为 D，0～59 为 E)。

```
#include "stdio.h"
void main()
{ int x; char y;
  scanf("%d",&x);
  if(x>=90) y='A';
  else if(x>=80) y='B';
       else if(x>=70) y='C';
            else if(x>=60) y='D';
                 else y='E';
  printf("y=%c\n",y);
}
```

【运行结果】

```
88↙
y= B
```

4. 嵌套格式

if 语句可以嵌套，即在一个 if 语句中又可以包含一个或多个 if 语句，一般形式为：

```
if(表达式 1)
    if(表达式 2)  语句 1
    else  语句 2
else
    if(表达式 3)  语句 3
    else  语句 4
```

在缺省花括号的情况下，if 和 else 的配对关系是：从最内层开始，else 总是与它上面

最近的并且没有和其他 else 配对的 if 配对。

【例 3.4】 已知函数

$$y=\begin{cases}-1, & x<0\\ 0, & x=0\\ 1, & x>0\end{cases}$$

编写程序，输入 x 值，输出 y 值。

```
#include "stdio.h"
void main()
{ float x; int y;
  scanf("%f",&x);
  if(x>=0)
    if(x>0) y=1;
    else    y=0;
  else y=-1;
  printf("x=%f,y=%d\n",x,y);
}
```

【运行结果】

```
-2↙
x=-2.000000,y=-1
```

思考题：例 3.4 是否还可用其他方法实现？

使用 if 语句时应注意以下几点：

(1) if 后面圆括号内的表达式可以为任意类型，但一般为关系表达式或逻辑表达式。

(2) if 和 else 后面的语句可以是任意语句。

(3) if(x)与 if(x!=0)等价。

(4) if(!x)与 if(x==0)等价。

(5) if 语句的各分支格式都是 if 语句嵌套格式的特例。

3.1.2 switch～case 语句

虽然用 if 语句可以解决多分支问题，但如果分支较多，嵌套的层次就多，这样会使程序冗长，可读性降低。C 语言提供了专门用于处理多分支情况的语句——switch～case 语句，其一般形式为：

```
switch(表达式)
{ case 常量表达式 1:语句 1  [break;]
  case 常量表达式 2:语句 2  [break;]
        ⋮
  case 常量表达式 n:语句 n  [break;]
  [default :语句 n+1  [break;]]
}
```

swich～case 语句的执行过程是：首先计算 switch 后面圆括号中表达式的值，然后用其结果依次与各 case 后面的常量表达式的值进行比较；若相等，执行该 case 后面的语句。执行时，如果遇到 break 语句，就退出 switch～case 语句，转至花括号的下方，否则顺序往下执行。若与各 case 后面常量表达式的值都不相等，则执行 default 后面的语句。

【例 3.5】 用 swich～case 语句实现例 3.3。

```
#include "stdio.h"
void main()
{ int a;
  char y;
  scanf("%d",&a);
  switch(a/10)
  { case 10: y='A';break;
    case 9:y='A';break;
    case 8:y='B';break;
    case 7:y='C';break;
    case 6:y='D';break;
    default:y='E';break;
  }
  printf("y=%c\n",y);
}
```

【运行结果】

```
88↙
y=B
```

请读者比较例 3.3 和例 3.5。

说明：

(1) switch 后面的圆括号后不能加分号。

(2) switch 后面圆括号内表达式的值必须为整型、字符型或枚举型。

(3) 各 case 后面常量表达式的值必须为整型、字符型或枚举型。

(4) 各 case 后面常量表达式的值必须互不相同。

(5) 若每个 case 和 default 后面的语句都以 break 语句结束，则各个 case 和 default 的位置可以互换。

(6) case 后面的语句可以是任何语句，也可以为空，但 default 的后面不能为空。若为复合语句，则花括号可以省略。

(7) 若某个 case 后面的常量表达式的值与 switch 后面圆括号内表达式的值相等，就执行该 case 后面的语句；执行完后若没有遇到 break 语句，就不再进行判断，接着执行下一个 case 后面的语句。若想执行完某一语句后退出，就必须在语句最后加上 break 语句。

(8) 多个 case 可以共用一组语句。如例 3.5 中的程序段：

```
case 10:y='A';break;
case 9:y='A';break;
```

可以改为：

```
case 10:
case 9:y='A';break;
```

（9）switch～case语句可以嵌套，即一个switch～case语句中又含有switch～case语句。

（10）default可省略，当没有与表达式值相匹配的常量表达式时，直接退出switch～case语句。

思考题：多个case可以共用一组语句吗？

【例3.6】 switch～case语句的嵌套举例。

```
#include "stdio.h"
void main()
{ int x,y,a=0,b=0;
  scanf("%d%d",&x,&y);
  switch(x)
  { case 1: switch(y)
            { case 0: a++;break;
              case 1: b++;break;
            }
    case 2: a++;b++;break;
    case 3: a++;b++;
  }
  printf("a=%d,b=%d\n",a,b);
}
```

【运行结果】

```
1  0↙
a=2,b=1
```

3.2 循环语句

C语言中有三种循环语句：while语句、do～while语句和for语句。它们都是在条件成立时反复执行某个程序段，这个反复被执行的程序段称为循环体。循环体是否被继续执行要依据某个条件，这个条件称为循环条件。

3.2.1 while语句

while语句的一般形式为：

```
while(表达式)
  循环体
```

其中，表达式可以是任意类型，一般为关系表达式或逻辑表达式，其值为循环条件。循环体可以是任何语句。

while 语句的执行过程为：

(1) 计算 while 后面圆括号中表达式的值，若其结果为非 0，转(2)；否则转(3)。

(2) 执行循环体，转(1)。

(3) 退出循环，执行循环体下面的语句。

其流程图见图 3.4。

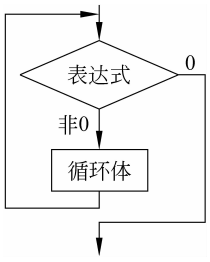

图 3.4　while 语句流程图

while 语句的特点：先判断表达式，后执行循环体。

【例 3.7】 从键盘上输入 10 个整数，输出偶数的个数及偶数和。

```
#include "stdio.h"
void main()
{ int i,n=0,sum=0,a;
  i=1;                          /*循环变量赋初值*/
  while(i<=10)                  /*循环条件为 i<=10*/
  { scanf("%d",&a);
    if(a%2==0)
    { n++;sum+=a;}
    i++;                        /*循环变量增值,使 i 趋于大于 10*/
  }
  printf("n=%d sum=%d\n",n,sum);
}
```

【运行结果】

```
1  2  3  4  5  6  7  8  9  10↙
n=5 sum=30
```

说明：

(1) 由于 while 语句是先判断表达式，后执行循环体，所以循环体有可能一次也不执行。

(2) 循环体可以是任何语句。如果循环体不是空语句，不能在 while 后面的圆括号后加分号(;)。

(3) 在循环体中要有使循环趋于结束的语句。如例 3.7 中的语句：i＋＋；。

3.2.2　do～while 语句

do～while 语句的一般形式为：

```
do  循环体 while(表达式);
```

其中，表达式可以是任意类型，一般为关系表达式或逻辑表达式，其值为循环条件。循环体可以是任意语句。

do～while语句的执行过程为：

(1) 执行循环体，转(2)。

(2) 计算while后面圆括号中表达式的值，若其结果为非0，转(1)；否则转(3)。

(3) 退出循环，执行循环体下面的语句。

其流程图见图3.5。

do～while语句的特点：先执行循环体，后判断表达式。

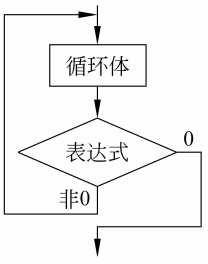

图3.5 do～while语句流程图

【例3.8】 计算整数n的值，使1＋2＋3＋…＋n刚好大于或等于500。

```
#include "stdio.h"
void main()
{ int n=0,sum;
  sum=0;                        /*循环变量赋初值*/
  do
  { n++;
    sum+=n;                     /*循环变量增值,使sum趋于500*/
  }while(sum<500);              /*循环条件为:sum<500*/
  printf("n=%d sum=%d\n",n,sum);
}
```

【运行结果】

```
n=32 sum=528
```

说明：

(1) do～while语句最后的分号(;)不可少，否则将出现语法错误。

(2) 循环体中要有使循环趋于结束的语句。如例3.8中的语句：sum＋＝n;。

(3) 由于do～while语句是先执行循环体，后判断表达式，所以循环体至少执行一次。

3.2.3 for语句

for语句的一般形式为：

```
for(表达式1;表达式2;表达式3)
    循环体
```

其中，循环体可以是任意语句。三个表达式可以是任意类型。一般来说，表达式1用于给某些变量赋初值，表达式2用来说明循环条件，表达式3用来修正某些变量的值。

for语句的执行过程为：

(1) 计算表达式1，转(2)。

(2) 计算表达式2，若其值为非0，转(3)；否则转(5)。

(3) 执行循环体,转(4)。

(4) 计算表达式 3,转(2)。

(5) 退出循环,执行循环体下面的语句。

其流程图见图 3.6。

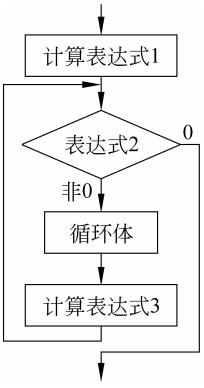

图 3.6　for 语句流程图

for 语句的特点:先判断表达式,后执行循环体。

【例 3.9】 计算 1～100 之间的整数和。

```
#include "stdio.h"
void main()
{ int i,sum;
  sum=0;
  for(i=1;i<=100;i++)
    sum+=i;     /* i=1 是为循环变量赋初值,i<=100 是循环
                   条件,i++是修正循环变量 */
  printf("sum=%d\n",sum);
}
```

【运行结果】

```
sum=5050
```

在 for 语句中,表达式 1 和表达式 3 经常使用逗号表达式,用于简化程序,提高程序运行效率。这也是逗号表达式的主要用途。如例 3.9 中的程序段:

```
sum=0;
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成:

```
for(i=1,sum=0;i<=100;sum+=i,i++);
```

在 for 语句中,在分号(;)必须保留的前提条件下,三个表达式的任何一个都可以省略,因此 for 语句又有如下省略形式。

1. for(;表达式 2;表达式 3) 循环体

表达式 1 省略。此时应在 for 语句之前给变量赋初值。如例 3.9 中的程序段:

```
for(i=1;i<=100;i++) sum+=i;
```

可以改写成:

```
i=1;
for(;i<=100;i++) sum+=i;
```

2. for(表达式 1;表达式 2;) 循环体

表达式 3 省略。此时应在循环体中修正循环变量。如例 3.9 中的程序段:

```
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成:

```
for(i=1;i<=100;)  {sum+=i;i++;}
```

3. for(表达式1；；表达式3）循环体

表达式2省略。此时认为表达式2的值始终为真，如果循环体中不包含break语句或goto语句，这时的循环无法终止，是死循环。如例3.9中的程序段：

```
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成：

```
for(i=1;;i++)  { if(i>100) break; sum+=i; }      /*break的功能见3.5.5*/
```

4. for(；表达式2；）循环体

表达式1和表达式3同时省略。此时应在for语句之前给变量赋初值，在循环体中修正循环变量。如例3.9中的程序段：

```
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成：

```
i=1;
for(;i<=100;)  {sum+=i;i++;}
```

这种情况完全等同于while语句。

5. for(；；表达式3）循环体

表达式1和表达式2同时省略。此时应在for语句之前给变量赋初值，在循环体中用break语句或goto语句退出循环。如例3.9中的程序段：

```
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成：

```
i=1;
for(; ;i++)  {if(i>100)  break; sum+=i;}
```

6. for(表达式1；；）循环体

表达式2和表达式3同时省略。此时应在循环体中修正循环变量，在循环体中用break语句或goto语句退出循环。如例3.9中的程序段：

```
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成：

```
for(i=1; ;)  { if(i>100)  break;  sum+=i; i++; }
```

7. for(；；）循环体

三个表达式同时省略。此时相当于：while(1）循环体。应在for语句之前给变量赋初值，在循环体中修正循环变量，在循环体中用break语句或goto语句退出循环。如例3.9中的程序段：

```
for(i=1;i<=100;i++)  sum+=i;
```

可以改写成：

```
i=1;
for(; ;)  { if(i>100)  break;  sum+=i; i++; }
```

说明：

(1) 若循环体不是空语句，不能在 for 语句的圆括号后加分号(;)。

(2) 表达式 1 或表达式 2 省略时，其后的分号不能省略，并且不能用其他符号代替。

(3) 要有使循环趋于结束的语句。

3.2.4　循环语句的嵌套

若一种循环语句的循环体中又有循环语句，则称为循环语句的嵌套。三种循环语句可以互相嵌套，并且可以嵌套多层。

【例 3.10】 输出九九表。

```
#include "stdio.h"
void main()
{ int i,j;
  for(i=1;i<=9;i++)
   {for(j=1;j<=i;j++)
      printf("%d*%d=%-3d",j,i,i*j);
    printf("\n");
   }
}
```

【运行结果】

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

3.2.5　break 语句和 continue 语句

1. break 语句

break 语句的一般形式为：

```
break;
```

break语句的功能：用于switch～case语句时，退出switch～case语句，程序转至switch～case语句下面的语句；用于循环语句时，退出循环体，程序转至循环体下面的语句。

【例3.11】 判断输入的正整数是否为素数，如果是素数，输出Yes，否则输出No。

```
#include "stdio.h"
void main()
{ int m,i;
  printf("m="); scanf("%d",&m);
  for(i=2;i<=m-1;i++)
    if(m%i==0) break;
  if(i>=m) printf("Yes");
  else printf("No");
}
```

【运行结果】

```
m=23↙
Yes
```

2. continue语句

continue语句的一般形式为：

```
continue;
```

continue语句的功能：结束本次循环，跳过循环体中尚未执行的部分，进行下一次是否执行循环的判断。在while语句和do～while语句中，continue把程序控制转到while后面的表达式处。在for语句中continue把程序控制转到表达式3处。

【例3.12】 计算1～100分别能够被2、4、8整除的整数个数。

```
#include "stdio.h"
void main()
{ int i,n2=0,n4=0,n8=0;
  for(i=1;i<=100;i++)
  { if(i%2)
      continue;     /*转至i++处*/
    n2++;
    if(i%4)
      continue;     /*转至i++处*/
    n4++;
    if(i%8)
      continue;     /*转至i++处*/
    n8++;
  }
  printf("n2=%d n4=%d n8=%d\n",n2,n4,n8);
}
```

【运行结果】

```
n2=50 n4=25 n8=12
```

说明：

(1) break 语句只能用于循环体和 switch～case 语句中。continue 只能用于循环体中。

(2) 用于循环体时，break 语句将整个循环终止，continue 语句只是结束本次循环。

(3) 在循环嵌套的情况下使用 break 语句时，仅仅退出包含 break 语句的最内层的那个循环语句的循环体；在 switch～case 语句嵌套的情况下使用 break 语句，仅仅退出包含 break 语句的最内层的 switch～case 语句。

3.2.6　goto 语句

goto 语句为无条件转移语句，其一般形式为：

goto 语句标号；

其中，语句标号是一种标识符，在 goto 语句所在的函数中必须存在，并且其后必须跟一个冒号(:)。冒号的后面可以为空，也可以是任何语句。语句标号表示程序在该点的地址。

goto 语句的功能：无条件地将程序控制转至语句标号处。

goto 语句的用途：一是与 if 语句一起实现循环，二是从循环嵌套的内层循环跳到外层循环外。

说明：

(1) 语句标号代表程序在该点的地址，使用 goto 语句只能实现在同一个函数内跳转，可以向前跳转，也可以向后跳转，但不能实现从一个函数跳转到其他函数。

(2) 只能从循环嵌套的内循环跳转到外循环，不能从外循环跳转到内循环。

(3) goto 语句使程序流程无规律，可读性差，不符合结构化原则，一般不宜采用，只有在不得已时才使用。

【例 3.13】 输入一组数，以 0 结束，求该组数据的绝对值之和。

```
#include "stdio.h"
void main()
{ int a,sum=0;
  loop1:                        /*语句标号*/
  scanf("%d",&a);
  sum+=a>0?a:-a;
  if(a!=0) goto loop1;          /*条件成立,转至语句标号 loop1 处*/
  printf("sum=%d",sum);
}
```

【运行结果】

```
1  2  -3  4  -5  0↙
sum=15
```

3.3 编译预处理

编译预处理是C语言编译系统的一个组成部分。所谓的编译预处理就是在对C源程序编译之前做一些处理，生成扩展的C源程序。C语言允许在程序中使用三种编译预处理命令，即宏定义、文件包含和条件编译。为了与C语言中的语句相区别，编译预处理命令以#开头。

3.3.1 宏定义

1. 不带参数的宏定义

不带参数宏定义的一般形式为：

```
#define  宏名  宏体
```

其中，#define是宏定义命令，宏名是一个标识符，宏体是一个字符序列。

功能：用指定的宏名（标识符）代替宏体（字符序列）。

如例2.1中的宏定义：

```
#define  PI  3.1415926
```

该宏定义的作用是用指定的标识符PI来代替其后面的字符序列3.1415926，这样，在后续程序中凡是用到3.1415926这个字符序列的地方，都可用PI来代替（见例2.1）。由此看出，宏定义能用一个简单的标识符代替一个冗长的字符序列，以便于程序的书写、阅读和修改，非常有实际意义。

在编译预处理时，编译程序将所有的宏名替换成对应的宏体，用宏体替换宏名的过程称为宏展开，也叫宏替换。如例2.1中的程序，宏替换完成后将变成：

```
#include "stdio.h"
void main()
{ float r,l,s,v;
  printf("r="); scanf("%f",&r);
  l=2*3.1415926*r;
  s=3.1415926*r*r;
  v=4.0/3*3.1415926*r*r*r;
  printf("l=%f  s=%f  v=%f\n",l,s,v);
}
```

说明：

（1）宏名一般用大写，以便于阅读程序，但这并非规定，也可用小写。

（2）在宏定义中，宏名的两侧至少各有一个空格。

（3）宏定义不是C语句，不能在行尾加分号。如果加了分号，在预处理时连分号一起替换。一个宏定义要独占一行。

（4）宏定义的位置任意，但一般放在函数外。

(5) 取消宏定义的命令是 # undef,其一般形式为:

#undef　宏名

(6) 宏名的作用域为宏定义命令之后到该源文件结束,或遇到 # undef 结束。

(7) 在程序中,若宏名用双引号括起来,在宏替换时不进行替换处理。

(8) 宏可以嵌套定义,即在一个宏定义的宏体中可以含有前面宏定义中的宏名。在宏嵌套时,应使用必要的圆括号,否则有可能得不到所需的结果。

(9) 宏替换只是进行简单的字符替换,不作语法检查。

(10) 在一个源文件中可以对一个宏名多次定义,新的宏定义出现,就是对同名的前面宏定义的取消。

【例 3.14】 不带参数的宏定义应用举例。

```
#include "stdio.h"
#define N  4                          /*后面不能带分号*/
#define M  N+3                        /*宏嵌套,后面不能带分号*/
void main()
{ int a;
  a=M*N;                              /*宏替换后为:a=4+3*4;*/
  printf("N=%d,M=%d,",N,M);           /*"N=%d,M=%d,"中 N 和 M 不被替换。宏替换后为:
                                       printf("N=%d,M=%d,",4,4+3);*/
  printf("M*N=%d\n",a);               /*M 和 N 不被替换*/
  #undef M                            /*取消宏定义,M 不再代表 N+3*/
  #define  M  (N+3)                   /*重新宏定义,M 代表(N+3)*/
  a=M*N;                              /*宏替换后为:a=(4+3)*4;*/
  printf("N=%d,M=%d,",N,M);           /*"N=%d,M=%d,"中 N 和 M 不被替换。宏替换后为:
                                       printf("N=%d,M=%d,",4,(4+3));*/
  printf("M*N=%d\n",a);               /*M 和 N 不被替换*/
}
```

【运行结果】

```
N=4,M=7,M*N=16
N=4,M=7,M*N=28
```

2. 带参数的宏定义

带参数宏定义的一般形式为:

 # define　宏名(形参表列)　宏体

其中,# define 是宏定义命令,宏名是一个标识符,形参表列是用逗号隔开的一个标识符序列,序列中的每个标识符都称为形式参数,简称形参。宏体是包含形参的一个字符序列。

例如:

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

s是宏名，a、b是形参，a＞b? a:b是宏体。

在程序中使用带参数宏的一般形式为：

宏名(实参表列)

其中，实参表列是用逗号隔开的表达式(常量和变量是特殊表达式)。

例如：

```
s(3,4)
```

编译预处理时，用宏体中的字符序列从左向右替换。如果不是形参，则保留；如果是形参，则用程序语句中相应的实参替换。这个过程叫宏展开，也叫宏替换。

例如：

```
s(3,4)宏替换后为:3>4?3:4
```

在使用带参数的宏定义时，应注意以下几点：

(1) 定义带参数的宏时，宏名和右边的圆括号“(”之间不能加空格，否则就成了不带参数的宏定义。如有下列宏定义：

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

则宏名为s，宏体为 (a,b) a＞b? a:b

(2) 为了正确进行替换，一般将宏体和各形参都加上圆括号。

(3) 若实参是表达式，则宏展开之前不求解表达式，宏展开之后进行真正编译时再求解。

【例 3.15】 带参数的宏定义应用举例。

```
#include "stdio.h"
#define X 3                              /*不带参数的宏定义*/
#define Y 4                              /*不带参数的宏定义*/
#define M(a,b) a>b?a:b                   /*带参数的宏定义*/
void main()
{ int val,x,y;
  scanf("%d%d",&x,&y);
  val=M(X,Y);                            /*实参为常量,宏展开后为: val=3>4?3:4*/
  printf("M=%d,",val);                   /*M不被替换*/
  val=M(x,y);                            /*实参为变量,宏展开后为:val=x>y?x:y*/
  printf("M=%d,",val);                   /*M不被替换*/
  val=M(x+3,y+4)*M(x+3,y+4);             /*实参为表达式,宏展开后为:
                                  val=x+3>y+4? x+3:y+4*x+3>y+4? x+3:y+4*/
  printf("M=%d,",val);                   /*M不被替换*/
  #undef M(a,b)                          /*取消宏定义*/
  #define M(a,b) ((a)>(b)?(a):(b))       /*重新宏定义*/
  val=M(x+3,y+4)*M(x+3,y+4);             /*实参为表达式,宏展开后为:
              val=((x+3)>(y+4)?(x+3):(y+4))*((x+3)>(y+4)?(x+3):(y+4))*/
```

```
    printf("M=%d\n",val);                /*M 不被替换*/
}
```

【运行结果】

```
3  4↙
M=4,M=4,M=6,M=64
```

3.3.2　文件包含

文件包含的一般格式为：

```
#include  "文件名"
```

或

```
#include < 文件名>
```

其中，#include 是文件包含命令，文件名是被包含文件的文件名。

如在例 2.14 中使用的文件包含：

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

功能：将指定的文件内容全部包含到当前文件中，替换 #include 命令位置。

处理过程：编译预处理时，用被包含文件的内容取代该文件包含命令；编译时，再对“包含”后的文件作为一个源文件进行编译。

若编译预处理前 f1.c 和 f2.c 的内容如图 3.7(a)所示，则编译预处理后 f1.c 和 f2.c 的内容如图 3.7(b)所示。

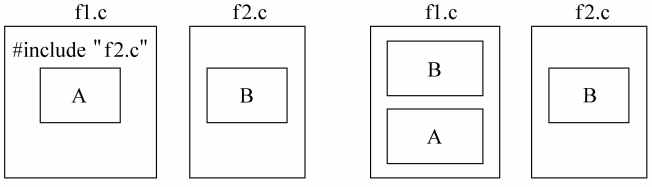

图 3.7　文件包含的预处理

两种文件包含格式的区别：

#include　"文件名"：系统先在当前目录搜索被包含的文件，若没找到，再到系统指定的路径去搜索。

#include　＜文件名＞：系统直接到系统指定的路径去搜索。

被包含文件的类型通常是以“.h”为扩展名的头文件(或称“标题文件”)和以“.c”为扩展名的源程序文件；该文件既可以是系统提供的，也可以是用户自己编写的。

在 Turbo C 中，常用的系统提供的头文件及说明见表 3.1。

表 3.1 常用头文件及说明

头文件名	说　明	头文件名	说　明
stdio.h	标准输入输出头文件	dos.h	DOS接口函数头文件
string.h	字符串操作函数头文件	ctype.h	字符操作头文件
math.h	数学函数头文件	graphics.h	图形函数头文件
conio.h	屏幕操作函数头文件	stdlib.h	常用函数头文件

需要注意的是：动态地址分配函数在 stdlib.h 中，VC++ 中没有头文件 graphics.h。若在 VC++ 环境下使用图形函数，则需下载并安装适用于 VC++ 的 graphics.h（http://www.easyx.cn/downloads/）。

使用文件包含的目的是避免程序的重复书写，特别是能够使用系统提供的诸多可供包含的文件。

说明：

（1）一个#include 命令只能指定一个被包含文件，用文件包含可实现文件的合并连接。

（2）一个#include 命令要独占一行。

（3）文件包含可以嵌套，即在一个被包含文件中又可以包含另一个文件。

（4）被包含的文件必须存在，并且不能与当前文件有重复的变量、函数及宏名等。

3.3.3 条件编译

所谓条件编译就是对源程序的一部分指定编译条件。若条件满足则参加编译，否则不参加编译。一般情况下，程序清单中的程序全部都应参加编译。但是在大型应用程序中，可能会出现某些功能不需要的情况，这时就可以利用条件编译来选取需要的功能进行编译，以便生成不同的应用程序，供不同的用户使用。此外，条件编译还可以方便程序的逐段调试，简化程序调试工作。条件编译有三种形式。

1. 形式 1

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

作用：若标识符已经被定义过（一般使用#define 命令定义），则对程序段 1 进行编译，否则对程序段 2 进行编译。其中#else 和后面的程序段 2 可以省略。省略时，若标识符已被定义过，则编译程序段 1，否则不编译程序段 1。

2. 形式 2

```
#ifndef  标识符
    程序段 1
[#else
```

```
    程序段 2]
#endif
```

作用：若标识符未被定义过，则对程序段 1 进行编译，否则对程序段 2 进行编译。其中 # else 和后面的程序段 2 可以省略。省略时，若标识符未被定义过，则编译程序段 1，否则不编译程序段 1。

3. 形式 3

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

作用：若表达式的值为真，则对程序段 1 进行编译，否则对程序段 2 进行编译。其中 # else 和后面的程序段 2 可以省略。省略时，若表达式的值为真，则对程序段 1 进行编译，否则不编译程序段 1。

【例 3.16】 条件编译应用举例。

```
#include "stdio.h"
#define FLAG 1
void main()
{ int a,b,m;
  scanf("%d%d",&a,&b);
  #if FLAG
    m=a>b?a:b;
  #else
    m=a>b?b:a;
  #endif
  printf("m=%d\n",m);
}
```

【运行结果】

```
3  4↙
m=4
```

上述程序在编译时，由于开始定义的符号常量 FLAF 的值为 1，语句 m＝a＞b? a:b;参加编译，语句 m＝a＞b?b:a;不参加编译。如果将程序清单中的宏定义命令：# define FLAG 1 改为 # define FLAG 0，则语句 m＝a＞b?b:a;参加编译，语句 m＝a＞b?a:b;不参加编译。

3.4　程序设计举例

【例 3.17】 输入一个带符号的整型数，输出该数的位数。

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

```
void main()
{ int x,y,m=0;
  scanf("%d",&x); /* 在 VC++中可测 10 位,准确位 9 位 */
  y=x>=0?x:-x;
  while(y)
  { m++;y/=10;}
  printf("%d is %d bit number\n",x,m);
}
```

【运行结果】

```
23↙
23 is 2 bit number
```

【例 3.18】 利用下列公式计算 π 的近似值。

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots\pm\frac{1}{2n-1}\quad\left(\text{精度要求为}\frac{1}{2n-1}<10^{-4}\right)$$

```
#include "stdio.h"
#include "math.h"                    /* 程序中用到求绝对值函数 fabs() */
void main()
{ int n=1,t=1;
  float pi=0;
  while(fabs(t*1.0/n)>=1e-4)         /* 控制循环的条件是当前项的精度 */
   {
    pi+=t*1.0/n;                     /* 将当前项累加到 pi 中 */
    t=-t;                            /* 得到下一项的符号 */
    n+=2;                            /* 得到下一项的分母 */
   }
  printf("pi=%.2f\n",4*pi);          /* 输出 π 的近似值 */
}
```

【运行结果】

```
pi=3.14
```

需要注意的是，求实型数据的绝对值用 fabs()函数，求整型数据的绝对值用 abs()函数。

【例 3.19】 输出 Fibonacci 数列的前 40 项，Fibonacci 数列为：

$$F_n=\begin{cases}1, & n=1\\ 1, & n=2\\ F_{n-1}+F_{n-2}, & n\geqslant 3\end{cases}$$

```
#include "stdio.h"
void main()
{ int i;
  long f1=1,f2=1;            /* 从第 24 项开始超出了整型数的范围,必须定义为长整型 */
  for(i=1;i<=20;i++)
```