第3章　控制流程

3.1　算　　法

一个程序应包括对数据的描述和对操作(数据处理)的描述。

(1) 对数据的描述：在程序中要指定数据的类型和数据的组织形式，即数据结构(data structure)。

(2) 对操作的描述：即操作步骤，也就是算法(algorithm)。数据是操作的对象，操作的目的是对数据进行加工处理。

作为程序设计人员，必须认真考虑和设计数据结构和操作步骤(即算法)。著名计算机科学家沃思(Nikiklaus Wirth)提出了一个公式：

数据结构 ＋ 算法 ＝ 程序

实际上，一个程序除了以上两个主要要素之外，还应当采用结构化程序设计方法进行程序设计，并且用某一种计算机语言表示。因此，可以这样表示：

程序＝算法＋数据结构＋程序设计方法＋语言工具和环境

以上等号后面的四个方面是一个程序设计人员所应具备的知识。这四个方面中，算法是程序的灵魂，是为解决一个问题而采取的方法和步骤，也可以说是在程序中解决“做什么”和“如何做”的问题。数据结构是程序的加工对象，语言是工具，编程还需要合适的方法。程序中的操作语句实际上就是算法的体现。不了解算法，就谈不上程序设计。

由于算法的重要性，本章将介绍算法的一些初步知识，以便介绍如何编写一个C语言程序。

3.1.1　算法概述

做任何事情都有一定的步骤。为解决一个问题而采取的方法和步骤，就称为算法。对于同一问题，可以有不同的解决方法和步骤。例如，求1＋2＋3＋…＋100，即 $\sum_{n=1}^{100} n$。有的人可能先进行1＋2，再加3，再加4，一直加到100。也有的人可能会采用这样的方法：$\sum_{n=1}^{100} n$ ＝100＋(99＋1)＋(98＋2)＋…＋(51＋49)＋50＝5050。当然也可以用其他的方法。一般来说，人们都希望用相对较为简单的方法和较少的步骤解决问题。因此，为了

更加有效地解决问题，不仅要考虑算法的正确性，还要考虑算法的质量，选择合适的算法。

本书介绍的只限于计算机算法，即计算机能够执行的算法。

计算机算法可分为如下两大类：

(1) 数值运算算法：主要目的是求解数值。例如求解方程的跟，求函数的定积分，求几何图形的面积，都属于数值运算的范围。

(2) 非数值运算算法：包括的范围很广，最常见的是事务管理领域。例如图书检索、人事管理等。

对于数值运算有现成的模型，可以运用数值分析方法，因此对于数值运算的算法研究比较深入，也相对比较成熟。但是计算机在非数值运算的应用比数值运算方面的应用更为广泛。

下面是一个简单的算法，希望读者能通过这样一个简单的算法，对算法即算法特性有一定的了解。

例 3.1 求 1×2×3×4×5。

最原始方法：

步骤 1：先求 1×2，得到结果 2。

步骤 2：将步骤 1 得到的乘积 2 乘以 3，得到结果 6。

步骤 3：将 6 再乘以 4，得 24。

步骤 4：将 24 再乘以 5，得 120。

这样的算法虽然正确，但十分繁复。如果要求 1×2×2×…×1000，则要 999 个步骤。步骤繁复，而且每一步都要直接利用上一步的数值结果(如 2，6，24 等)，也不方便。可以采用更加简便的通用算法。

改进的算法：

S1：使 t=1。

S2：使 i=2。

S3：使 t×i，乘积仍然放在变量 t 中，可表示为 t×i→t。

S4：使 i 的值+1，即 i+1→i。

S5：如果 i≤5，返回重新执行步骤 S3 以及其后的 S4 和 S5；否则，算法结束。最后得到的 t 的值，就是 5!的值。

如果计算 100!，只需要将 S5 中的 i≤5 改成 i≤100 即可。

如果要求 1×3×5×7×9×11，算法也只需要做很少的改动：

S1：1→t。

S2：3→i。

S3：t×i→t。

S4：i+2→t。

S5：若 i≤11，返回 S3，否则，结束。

可以看出，这种方法表示的算法具有通用性、灵活性。S3 到 S5 组成了一个循环，在实现算法的时候，多次运行 S3，S4，S5 等步骤，直到执行 S5 步骤时，判断乘数 i 已经超过限定值而结束程序。该算法不仅正确，而且是较好的算法，因为计算机是高速运算的自动

机器，实现循环轻而易举。

3.1.2 算法的特性

一个算法应该具有以下特点：

(1) 有穷性。一个算法应包含有限的操作步骤而不能是无限的。否则程序将陷入死循环，因而也不是有效的算法。

(2) 确定性。算法中每一个步骤应当是确定的，而不能是含糊的、模棱两可的。即算法中的每一个步骤的含义应当是唯一的，而不应当产生“歧义”，不应当被理解成不同的含义，而应该是明确的。

(3) 有零个或多个输入。所谓输入是指在执行算法时需要从外界取得必要的信息。例如，判断一个数是否是素数，此时这个被判断的数就是输入。也可以没有，或者有两个及两个以上的输入。

(4) 有一个或多个输出。算法的目的是求解，“解”就是输出。算法的输出，不一定就是计算机打印输出，一个算法得到的结果，就是算法的输出。没有输出的算法是没有意义的。

(5) 有效性。算法中每一个步骤应当能有效地执行，并得到确定的结果。

3.1.3 算法的表示方法

为了表示一个算法，可以用不同的方法。常用的方法有自然语言、传统流程图、结构化流程图、N-S 流程图、伪代码等。

例 3.1 中的算法就是用自然语言表示的。自然语言就是人们日常生活中使用的语言，可以是汉语、英语或其他语言。用自然语言表示通俗易懂，但是文字冗长，容易出现“歧义”。自然语言表示的含义往往不大严格，要根据上下文才能判断其准确含义。因此，除了很简单的问题，一般不用自然语言表示算法。

3.1.4 流程图

流程图是用一些图框表示各种操作。用图形表示算法，直观形象，易于理解。美国标准化协会 ANSI(American National Standard Institute)规定了一些常用的流程图符号，如图 3-1 所示。

图 3-1 中菱形框的作用是对一个给定的条件进行判断，根据给定的条件是否成立来决定如何执行其后的操作。它有一入口，两个出口。如判断条件 x 为奇数还是偶数，并打印出来，可以用图 3-2 表示。

连接点是用于将画在不同地方的流程线连接起来。在连接点的小圆圈中央写上标号，当有相同标号出现时，就表明这两个点是相互连接在一起的。实际上它们是同一个点，只是由于为了避免流程过长或者交叉，或者由于画不下，而将流程图分为两部分使流程图更加清晰，如图 3-3 所示。

对例 3.1 的算法用流程图表示，如图 3-4 所示。

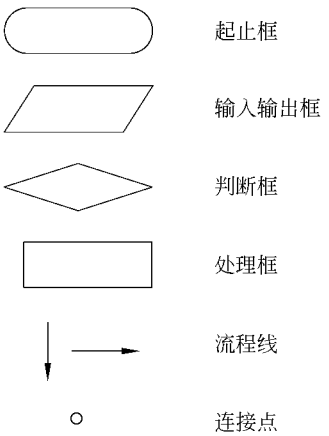

图 3-1　常用流程图符号

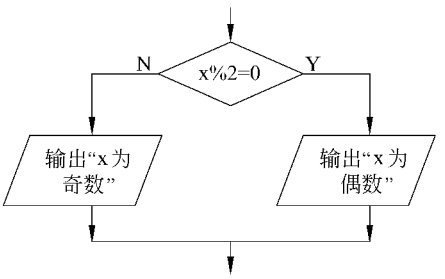

图 3-2　使用菱形框判断条件

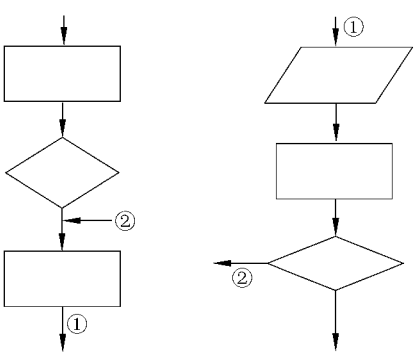

图 3-3　连接点的使用

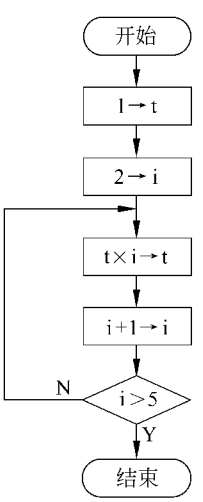

图 3-4　例 3.1 算法流程图

3.1.5　三种基本结构和改进的流程图

1. 传统流程图的弊端

传统的流程图用流程线指出各框的执行顺序，对流程线的使用没有严格的限制。因此，使用者可以不受限制地使流程随意转来转去，使流程图变得毫无规律。阅读者要花很大的精力去追踪流程，使人难以理解算法的逻辑。许多时候，流程图会变得如同乱麻一样，乱无头绪，如图 3-5 所示。

为了提高算法的质量，使算法的设计和阅读方便，必须限制箭头的滥用，即不允许无规律的使流程随意转向，只能顺序地进行下去。但是算法总是要包括一些分支和循环，不可能全部由一个一个框顺序组成。所以，人们规定了几种基本结构，然后由这些基本结构按一定规律组成算法结构，整个算法的结构是由上至下地将各个基本结构顺序排列起来，

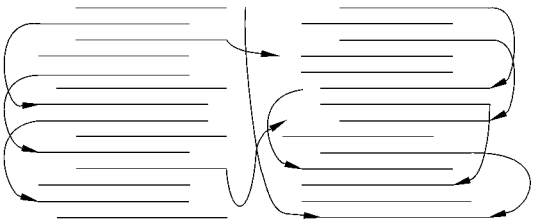

图 3-5　杂乱的流程示意

分支和循环都发生在各个基本结构之中，这样，算法的质量就得到了保证和提高。

2. 三种基本结构

结构化程序由三种基本结构组成，即顺序结构、选择结构、循环结构。这三种基本结构作为表示一个良好算法的基本单元。

1）顺序结构

顺序结构是最简单的一种基本结构，即按语句在程序中的先后顺序逐条执行。流程图如图 3-6 所示。虚线框中是一个顺序结构，其中 A 和 B 指定的操作是顺序完成的，即先执行完 A 所指定的操作，再完成 B 指定的操作。

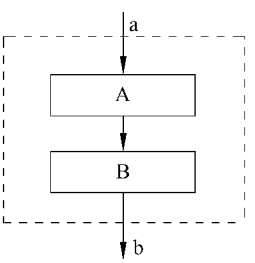

图 3-6　顺序结构

2）选择结构

选择结构也称为分支结构，即当执行该结构中的语句时，程序将根据不同的条件执行不同分支中的语句，流程图如图 3-7 所示。虚线框中是一个选择结构。此结构中必须包含一个判断框，根据给定的条件 p 是否成立而选择执行 A 框操作或 B 框操作。

注意：无论 p 条件是否成立，只能执行 A 框或 B 框之一，不可能既执行 A 框，又执行 B 框。无论走哪一条路径，最后都通过 b 出口脱离结构。A 框或 B 框中可以有一个是空的，即不执行任何操作，如图 3-8 所示。

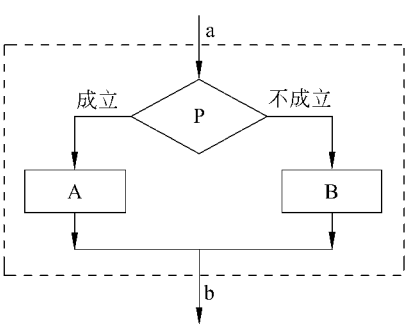

图 3-7　选择结构

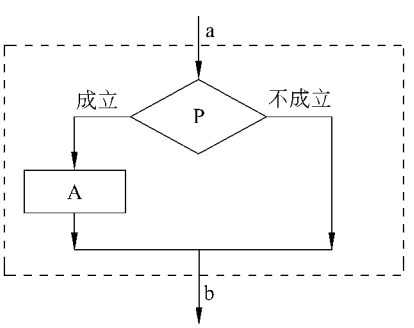

图 3-8　选择结构中的特殊情况

3）循环结构

循环结构又称为重复结构，即反复执行某一部分的操作。循环结构有两种类型：当

型循环和直到型循环。

(1) 当型(while 型)循环结构。

当判断条件成立时,重复执行某个操作。如图 3-9(a)所示。当给定的条件 p1 成立时,执行 A 框的操作,执行完 A 框的操作后,再判断条件 p1 是否成立,如果成立,则反复执行 A 框操作,直到某一时刻 p1 条件不成立为止,通过出口 b 脱离循环结构。

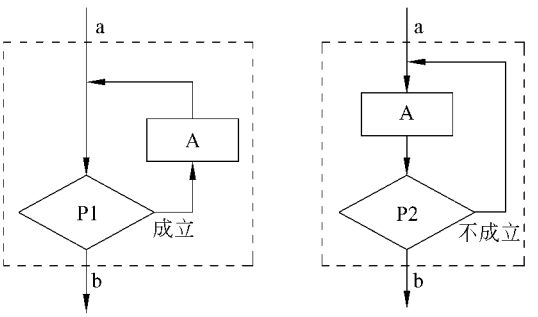

图 3-9 循环结构

(2) 直到型(until 型)循环。

重复执行某个操作,直到满足条件为止。如图 3-9(b)所示。先执行 A 框,然后判断给定的条件 p2 是否成立,如果不成立,则再执行 A,然后再对 p2 条件作判断,如果 p2 条件仍然不成立,反复执行 A,直到给定的 p2 条件满足为止,此时不在执行 A,脱离循环结构,顺序执行之后的操作。

图 3-10 和图 3-11 分别是当型循环和直到型循环应用的例子,其作用都是打印五个数:1,2,3,4,5。可以看出,对待一个问题,既可以用当型循环来处理,也可以用直到型循环来处理。

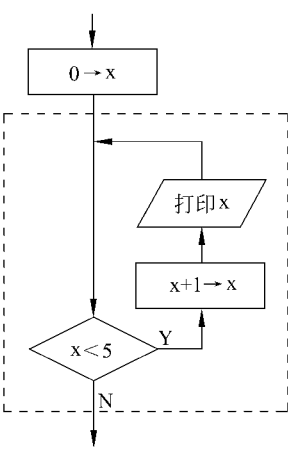

图 3-10 当型循环

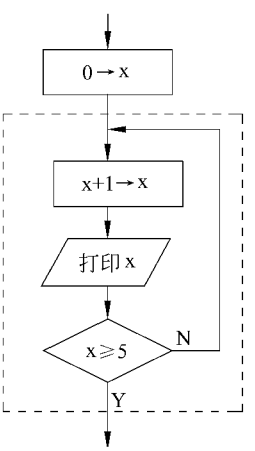

图 3-11 直到型循环

无论是顺序结构、选择结构还是循环结构,它们都有以下共同特点:

- 只有一个入口,在图 3-6～图 3-9 中,a 为入口点。

- 只有一个出口，在图 3-6～图 3-9 中，b 为出口点。
- 结构内的每一部分都有机会被执行到。
- 结构内不存在“死循环”。

由基本结构所构成的算法属于“结构化”算法，它不存在无规律的跳转，只在本基本结构内才允许存在分支。

3.1.6　N-S 流程图

1973 年，美国学者 I. Nassi 和 B. Shneiderman 提出了一种新型流程图：N-S 流程图。这种流程图完全省略了流程线，算法的每一步都用一个矩形框来描述，一个框一个框按执行顺序连接起来，组成一个大的框，来描述一个完整的算法。这种将全部算法写在一个矩形框中的流程图就成为 N-S 结构化流程图。

N-S 流程图用以下流程图符号表示。

(1) 顺序结构。按语句在程序中的先后顺序逐条执行。N-S 流程图如图 3-12 所示，其中 A 和 B 指定的操作是顺序完成的。

(2) 选择结构。根据给定的条件 p 是否成立而选择执行 A 框操作或 B 框操作。N-S 流程图如图 3-13 所示。

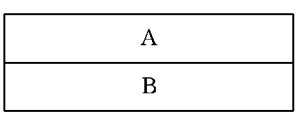

图 3-12　顺序结构的 N-S 流程图

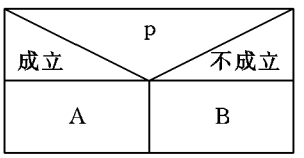

图 3-13　选择结构的 N-S 流程图

(3) 循环结构。

① 当型(while 型)循环结构的 N-S 流程图用图 3-14 表示。给定的条件 p1 成立时，执行 A 框的操作，反复执行 A 框操作，直到某一时刻 p1 条件不成立为止，脱离循环结构。

② 直到型(until 型)循环的 N-S 流程图用图 3-15 表示。先执行 A 框，然后判断给定的条件 p1 是否成立，如果不成立，反复执行 A，直到给定的 p1 条件满足为止，此时不再执行 A，脱离循环结构。

图 3-14　当型循环的 N-S 流程图

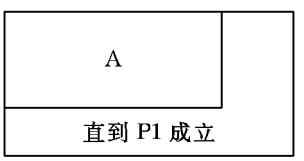

图 3-15　直到型循环的 N-S 流程图

用以上三种 N-S 流程图中的基本框，可以组成复杂的 N-S 流程图，表示完整的算法。图 3-12～图 3-15 中 A 框和 B 框的操作既可以是一个简单操作，也可以是三个基本结构之一，即它们由顺序结构、选择结构、循环结构组成。

例 3.1 的求 5！的算法如果用 N-S 流程图表示，如图 3-16 所示。

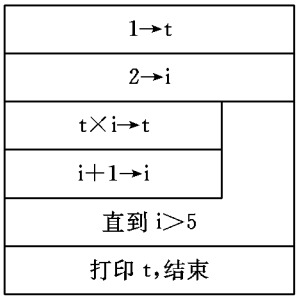

图 3-16 求 5! 算法的 N-S 流程图

3.2 顺序结构程序设计

已经知道,从程序流程的角度来看,程序可以分为三种基本结构,即顺序结构、选择结构、循环结构。这三种基本结构可以组成各种复杂的程序。

同其他高级语言一样,C 语言提供了多种语句来实现这些程序结构。C 语言的语句用来向计算机系统发出操作命令,一个实际的程序由若干条语句组成,每个语句用分号作为结束符。C 语言程序的执行部分是由语句组成的。程序的功能也是由执行语句实现的。

C 语言语句可分为以下五类:表达式语句、函数调用语句、控制语句、复合语句、空语句。

3.2.1 表达式语句

表达式语句由表达式加上分号";"组成。表达式语句是 C 语言中最基本的语句。

其一般形式为:

```
表达式;
```

执行表达式语句就是计算表达式的值。

例如:

```
s=pi*r*r;
a-b;
x+'a'-20;
```

表达式语句中最典型的就是用赋值表达式构成一个赋值语句,即在赋值表达式后面加上一个分号";"。其一般形式为:

```
变量=表达式;
```

例如:

```
x=y+z;
```

```
a='o';b='k';
```

赋值语句的功能是将赋值运算符“=”右边的表达式的值赋给赋值运算符左边的变量。2.7节中已经讲到，如果表达式的数据类型和变量的数据类型不同，表达式运算结果的类型将自动转换成变量的数据类型。

注意：赋值语句必须以“;”结束，否则就成了赋值表达式。赋值语句和赋值表达式的区别在于赋值表达式可以出现在任何允许表达式出现的地方，而赋值语句则不能。

例如：

```
if ((a=b)>0) y=a;
```

是合法的语句。其作用是，先进行赋值运算，将b的值赋给a，再判断a是否大于0，若大于，则执行y=a的赋值运算。在if语句中，“a=b”是赋值表达式，不是赋值语句。赋值表达式可以包含在其他表达式中。但是如果写成了

```
if ((a=b;)>0) y=a;
```

就是非法的语句了。赋值语句不能包含在其他表达式中。

3.2.2 函数调用语句

函数调用语句由一个函数调用加一个分号“;”构成。

其一般形式为：

```
函数名(实际参数表);
```

执行函数语句就是调用函数体并把实际参数赋予函数定义中的形式参数，然后执行被调函数体中的语句，求取函数值(在后面函数中再详细介绍)。函数调用以语句的形式出现，与前后语句之间的关系是顺序执行的。通过第2章的学习，已经知道许多C语言中的基本输入输出都是通过函数调用语句来实现的。

例如：

```
printf("C Program");          /*调用库函数,输出字符串*/
scanf("%d%d%d",&a,&b,&c);
printf("a=%d,b=%d,c=%d",a,b,c);
```

在使用C语言标准的函数输入输出库函数时，要用预编译命令#include <stdio.h>或#include "stdio.h"，将有关“头文件”包括到源文件中。

函数既可以是存在于函数库中的，也可是用户自己定义的。

例如：

```
sin(x);                       /*sin为math.h文件中的函数*/
fun(5);                       /*程序中用户自定义了一个函数fun;*/
```

3.2.3 控制语句

控制语句用于完成一定的控制功能，控制程序的流程，如选择控制，循环控制等。它

们由特定的语句定义符组成。

C语言有9种控制语句，可分成以下四类：

(1) 选择分支控制语句：if语句、switch语句。

(2) 循环控制语句：for语句、while语句、do…while语句。

(3) 结束控制语句：break语句、continue语句。

(4) 转向控制语句：goto语句、return语句。

具体如表3-1所示。

表3-1 C语言中的9种控制语句

语句种类	语句形式	作　用
选择分支控制语句	if() … else…	条件判断，分支语句
	switch(){…}	多分支选择语句
循环控制语句	for()	循环语句
	while()	循环语句
	do… while	循环语句
结束控制语句	break	终止执行switch或循环语句
	continue	结束本次循环语句
转向控制语句	goto	转向语句
	return	从函数返回语句

上面9种控制语句中，()表示其中是一个条件，…表示内嵌的语句。例如：if() … else…，具体可以表述为：

```
if (x%2==0) printf("x is even");
else printf("x is odd");
```

3.2.4 复合语句

把多个语句用括号{}括起来组成的一个语句称复合语句。复合语句又称为“块语句”，复合语句的语句形式如下：

```
{语句1;语句2;…;语句n;}
```

用一对花括号把若干个语句括起来组成一个语句组，在语法上复合语句被看成是一条语句，而不是多条语句。

例如：

```
{ a=2;
  b=3;
  c=a+b;
  printf("c=%d",c);
}
```