# 第 1 章　程序设计基础

语言，是人与人进行交流沟通的工具。人与计算机通信也需要语言，为了使计算机进行各种工作，就需要有一套用以编写计算机程序的数字、字符和语法规则，由这些字符和语法规则组成计算机的各种指令（或各种语句），这些就是计算机所能接受的语言，称为计算机语言。

## 1.1　简单的 C/C++程序

C 语言是目前世界上广泛使用的高级语言，其结构紧凑、语言简洁，只有 32 个关键字，9 种控制语句；使用方便灵活，书写形式自由；数据类型完备，运算符丰富；允许直接访问物理地址，对硬件进行操作；生成目标代码质量高，程序执行效率高；可用于各种型号的计算机和操作系统。

C 语言是结构化和模块化的语言，它是面向过程的。在处理较小规模的程序时，程序设计者用 C 语言较为得心应手；但是当面对的问题比较复杂、程序规模比较大时，程序设计者往往感到力不从心。

C++语言的目的是在 C 语言的基础上实现面向对象编程的功能，从而实现程序代码更易维护、代码重用性更高、程序结构更清晰且具有 C 语言的高效性。在 C 语言的基础上新增了一些数据类型，如 Boolean（逻辑型）、Const（常量）等，并且使用内联函数、引用、重载、继承、虚函数等，实现面向对象的程序设计功能。

### 1.1.1　输出“Hello,World！”

【例 1-1】 使用 C 语言编写程序输出“Hello,World!”。

```
#include <stdio.h>
int main() /*主程序部分，main 为程序入口*/
{
   printf("Hello,World!\n"); //printf()为输出函数
   return 0;
}
```

运行结果：

```
Hello,World!
```

程序说明：

① #include <stdio.h>指示编译器在对程序进行预处理时，将文件 stdio.h（standard input and output，标准输入/输出）中的代码嵌入到程序中该指令所在的地方，这样该程序在正式编译时，就可以利用包含文件中的内容了。

② 每个 C 程序都必须有并且只有一个 main()主函数，main 的名字也不可被更改。一个 C 程序可以包含若干个函数，总是从主函数处开始执行，main()函数前面的 int 表示本函数返回值是“整型”的意思，也可用 void 表示没有返回值。

③ 花括号{}括起来的表示函数体，由语句组成。分号“;”是语句结束符，表示该语句结束。printf()为输出函数，符号“\n”表示换行，即输出完内容之后，将光标移到下一行开头（最

左边）处。

④ 符号“/*……*/”和“//”为注释语句，它只帮助阅读和理解程序，可以起到注释、标识、说明、指示等作用，以增加程序的可读性。这部分内容不参加程序的编译，对程序的功能无影响。“/*……*/”可以跨行或跨段注释，“//”只注释本行。

【例 1-2】 使用 C++语言编写程序输出“Hello,World!”。

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"Hello,World!"<<endl;
    return 0;
}
```

运行结果：

```
Hello,World!
```

程序说明：

① iostream 指标准输入/输出流；C++中为了避免名字定义冲突，特别引入了“名字空间”的定义，即 namespace，用于标识符的各种可见范围。C++标准程序库中的所有标识符都被定义于一个名为 std 的 namespace 中。

② cin 和 cout 是标准设备，用来实现数据的输入和输出。cin 一般代表键盘（输入），cout 一般代表显示器（输出）。

③ endl 类似于 C 语言中的\n，作用是换行（输出）。

## 1.1.2 求解并输出 *n* 的阶乘值

【例 1-3】 使用 C 语言编写一个循环程序，输入 *n* 值，输出 *n* 的阶乘（*n*!）值。

分析：所谓 *n* 的阶乘就是从 1 到 *n* 的累积，所以可以通过一个 for 循环语句，从 1 到 *n* 依次求积即可。

```
#include <stdio.h>
int main()
{
    int i,n,fa=1;
    scanf("%d",&n);
    for(i=1;i<=n;i++)  //for 循环语句，初值为 1，终值为 n，步长为 1
        fa=fa*i;
    printf("%d\n",fa);
    return 0;
}
```

运行结果（输入 5，输出 120）：

```
5
120
```

【例 1-4】 使用 C++语言编写一个循环程序，输入 *n* 值，输出 *n* 的阶乘（*n*!）值。

```
#include <iostream>
using namespace std;
int main()
{
    int i,n,fa=1;
    cin>>n;
    for(i=1;i<=n;i++)
        fa=fa*i;
    cout<<fa<<endl;
    return 0;
}
```

运行结果（输入 6，输出 720）：

```
6
720
```

### 1.1.3 使用函数实现求解 *n* 的阶乘值

【例 1-5】 使用 C 语言函数形式编写程序，输入 *n* 值，输出 *n* 的阶乘（*n*!）值。

```
#include <stdio.h>
int factorial(int n)//定义求 n 的阶乘值的函数
{
    int i,fa=1;
    for(i=1;i<=n;i++)
        fa=fa*i;
    return fa;  //返回函数值
}
int main()
{
    int n,f;
    scanf("%d",&n);
    f=factorial(n);  //调用函数
    printf("%d\n",f);
    return 0;
}
```

运行结果（输入 5，输出 120）：

```
5
120
```

一个 C/C++程序可由一个主函数和若干个其他函数构成，由主函数调用其他函数，其他函数之间也可以互相调用。C/C++程序从 main()函数处开始运行，当 main()函数结束时，程序也就结束了。函数具有代码重用、提高编写效率和利于程序维护等诸多优点，读者可详细阅读第 5 章。

总结：

① C/C++程序是由一个或多个函数构成的，并且只能有一个主函数。

② 不管有多少个函数，程序执行都是从 main()函数开始的。在一个函数内，执行顺序是从上到下的。

③ 注释是从“//”开始的，具有增加可读性的作用。

④ 程序书写形式自由，一行内可以写多条语句，每条语句以“;”结束。

⑤ C 语言区分大小写字母。

## 1.2 算法

算法（Algorithm）是指对解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。算法是计算机科学最基本的概念，了解算法及其表示和设计方法是程序设计的基础和精髓。

### 1.2.1 算法的概念及表示方法

#### 1．算法及其特性

（1）什么是算法

算法就是一组有穷的规则，它规定了解决某个特定问题的一系列运算。通俗地说，为解决问题而采用的方法和步骤就是算法。

（2）算法的特性

① 确定性（Definiteness）。算法的每个步骤必须要有确切的含义，每个操作都应当是清晰的、无二义性的。

② 有穷性（Finiteness）。一个算法应包含有限的操作步骤且在有限的时间（人们可以接受的）内能够执行完毕。

③ 有效性（Effectiveness）。算法中的每个步骤都应当能有效地执行，并得到确定的结果。

④ 有零个或多个输入（Input）。在算法执行的过程中需要从外界取得必要的信息，并以此为基础解决某个特定问题。

⑤ 有一个或多个输出（Output）。设计算法的目的就是要解决问题，算法的计算结果就是输出。没有输出的算法是没有意义的。输出与输入有着特定的关系，通常，输入不同，会产生不同的输出结果。

（3）算法的分类

根据待解决问题的形式模型和求解要求，算法分为数值和非数值两大类。

① 数值运算算法：是以数学方式表示的问题求数值解的方法。例如，代数方程计算、线性方程组求解、矩阵计算、数值积分、微分方程求解等。通常，数值运算有现成的模型，这方面的现有算法比较成熟。

② 非数值运算算法：通常为求非数值解的方法。例如，排序、查找、表格处理、文字处理、人事管理、车辆调度等。

#### 2．算法的表示方法

设计出一个算法后，为了存档，以便将来算法的维护或优化，或者为了与他人交流，让他人能够看懂、理解算法，需要使用一定的方法来描述、表示算法。算法的表示方法很多，

常用的有自然语言、流程图和伪代码等。我们以求解 sum=1+2+3+4+5+…+(*n*–1)+*n* 为例，使用这三种不同的表示方法去描述解决问题的过程。

（1）自然语言（Natural Language）

用人们日常生活中使用的语言，如中文、英文、法文等来描述算法。中文描述上述从 1 开始的连续 *n* 个自然数求和问题的算法如下：

S1　确定一个 *n* 的值；

S2　假设等号右边的算式项中的初始值 *i* 为 1；

S3　假设 sum 的初始值为 0；

S4　如果 *i*≤*n*，则执行 S5，否则转出执行 S8；

S5　计算 sum 加上 *i* 的值后，重新赋值给 sum；

S6　计算 *i* 加 1，然后将值重新赋值给 *i*；

S7　转去执行 S4；

S8　输出 sum 的值，算法结束。

使用自然语言描述算法的优点是通俗易懂，没有学过算法相关知识的人也能够看懂算法的执行过程。但是，自然语言本身所固有的不严密性使得这种描述方法存在以下缺陷：

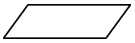
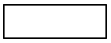
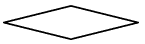
① 文字冗长，容易产生歧义性；

② 难以描述算法中的分支和循环等结构，不够方便、直观。

（2）流程图（Flow Chart）

流程图是最常见的算法图形化表达，有传统流程图和 N-S 流程图两种形式。传统流程图使用美国国家标准化学会（American National Standards Institute，ANSI）规定的一些图框、线条来形象、直观地描述算法处理过程。常见的流程图符号如表 1-1 所示。

**表 1-1　常见流程图符号**

| 符　号 | 名　称 | 作　用 |
|---|---|---|
|  | 开始符、结束符 | 表示算法的开始和结束符号 |
|  | 输入框、输出框 | 在算法过程中，表示从外部获取的信息（输入），然后将处理过的信息输出 |
|  | 处理框 | 在算法过程中，表示需要处理的内容，只有一个入口和一个出口 |
|  | 判断框 | 表示算法过程中的分支结构，菱形框的 4 个顶点，通常用上面的顶点表示入口，根据需要用其余的顶点表示出口 |
|  | 流程线 | 表示算法过程中流程的方向 |

使用流程图描述从 1 开始的连续 *n* 个自然数求和的算法如图 1-1 所示。

在使用过程中，人们发现流程线并不一定是必需的。随着结构化程序设计方法的出现，1973 年美国学者 I.Nassi 和 B.Shneiderman 提出了一种新的流程图形式，这种流程图完全去掉了流程线，算法的每步都用一个矩形框来描述，把一个个矩形框按执行的顺序连接起来，就是一个完整的算法描述。这种流程图用两位学者名字的第一个字母来命名，称为 N-S 流程图。

为了提高算法的质量，便于阅读理解，应限制流程的随意转向。为了达到这个目的，人们规定了三种基本结构，由这些基本机构按一定规律组成一个算法结构。

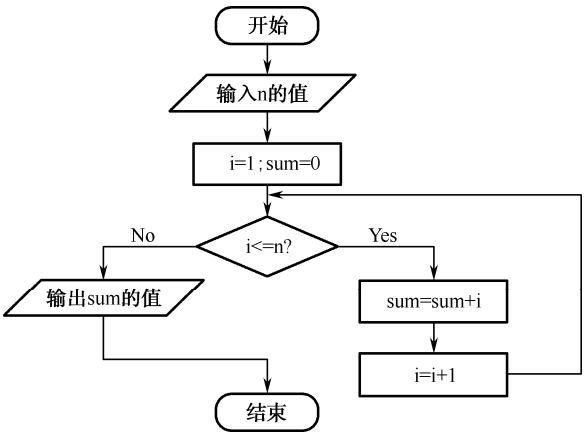



图 1-1　计算 1+2+3+…+*n* 的流程图

① 顺序结构：这是最简单的一种基本结构，各操作是按先后顺序执行的，如图 1-2 所示，图中操作 A 和操作 B 按照出现的先后顺序依次执行。

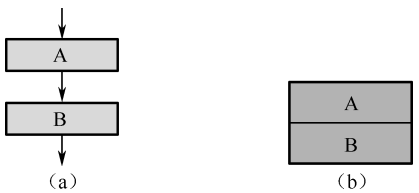



图 1-2　顺序结构

② 选择结构：又称分支结构，根据是否满足给定条件从两组操作中选择执行一种操作，某部分的操作可以为空操作。如图 1-3 所示，如果条件 P 成立则执行处理框 A，否则执行处理框 B。

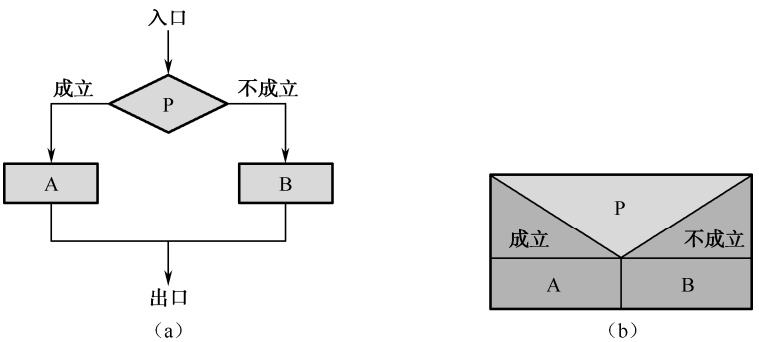



图 1-3　选择结构

③ 循环结构：又称重复结构，即在一定条件下，反复执行某部分的操作。循环结构又分为当型和直到型两种类型。

当型循环结构如图 1-4 所示，当条件 P 成立时，执行处理框 A，执行完处理框 A 后，再判断条件 P 是否成立，如果条件 P 成立，则再次执行处理框 A，如此反复，直至条件 P 不成立才结束循环。

直到型循环结构如图 1-5 所示，先执行处理框 A，再判断条件 P 是否成立，如果条件 P 不成立，则再次执行处理框 A，如此反复，直至条件 P 成立才结束循环。

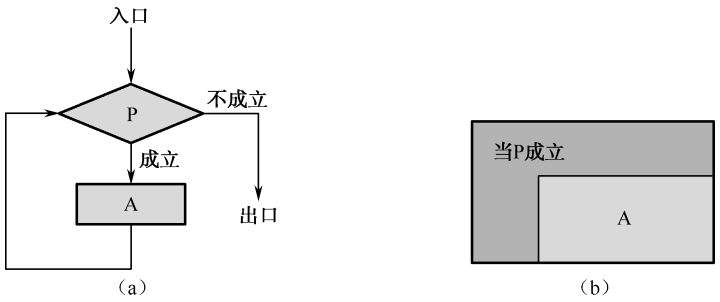



图 1-4　当型循环结构

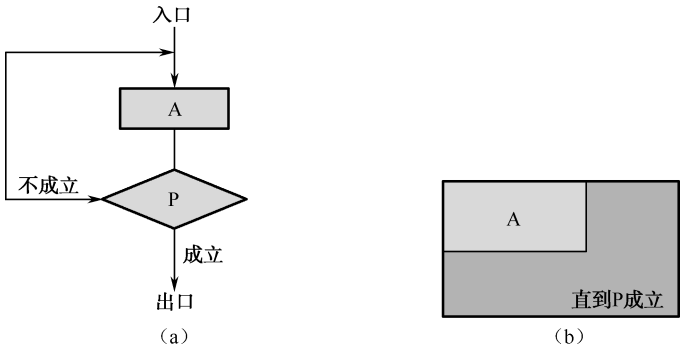



图 1-5　直到型循环结构

以上三种基本结构具有以下特点：

① 只有一个入口和一个出口；

② 结构内的每一部分都有机会被执行到；

③ 结构内不存在“死循环”（无终止的循环）。

（3）伪代码（Pseudocode）

伪代码是一种用来书写程序或描述算法时使用的非正式、透明的表述方法。它并非是一种编程语言，这种方法针对的是一台虚拟的计算机。伪代码通常采用自然语言、数学公式和符号来描述算法的操作步骤，同时采用计算机高级语言（如 C、Pascal、VB、C++、Java 等）的控制结构来描述算法步骤的执行。

使用伪代码描述从 1 开始的连续 *n* 个自然数求和的算法如下：

```
S1  算法开始；
S2  输入 n 的值；
S3  i ← 1；              //为变量 i 赋初值
S4  sum ← 0；            //为变量 sum 赋初值
S5  do while i<=n        //当变量 i <=n 时，执行下面的循环体语句
S6  { sum ← sum + i；
S7  i ← i + 1； }
S8  输出 sum 的值；
S9  算法结束。
```

## 1.2.2 算法设计的基本方法

算法设计的任务是对各类问题设计出良好的算法及研究设计算法的规律和方法。针对一个给定的实际问题，要找出确实行之有效的算法，就需要掌握设计的策略和基本方法。

### 1．穷举法（Exhaustive Algorithm）

穷举法也称为枚举法、蛮力法，是一种简单、直接解决问题的方法。使用穷举法解决问题的基本思路是：依次穷举问题所有可能的解，按照问题给定的约束条件进行筛选，如果满足约束条件，则得到一组解，否则不是问题的解。将这个过程不断地进行下去，最终得到问题的所有解。

要使用穷举法解决实际问题，应当满足以下两个条件：

① 能够预先确定解的范围并能以合适的方法列举；

② 能够对问题的约束条件进行精确描述。

穷举法的优点是：比较直观，易于理解，算法的正确性比较容易证明；缺点是：需要列举许多种状态，效率比较低。

**【例 1-6】** 百钱买百鸡问题。我国古代数学家张丘建在《算经》中出了一道题：“鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一。百钱买百鸡，问鸡翁、母、雏各几何？”。意思是：某个人有 100 钱，打算买 100 只鸡。公鸡 5 钱一只，母鸡 3 钱一只，小鸡 1 钱 3 只。请编写一个算法，算出如何能刚好用 100 钱买 100 只鸡？

此题可用穷举法来解，以三种鸡的个数为穷举对象（分别设为 $x$，$y$ 和 $z$），以三种鸡的总数和买鸡用去的钱的总数为判定条件，穷举各种鸡的个数。按题意列出方程组如下：

$$x+y+z=100 \quad (1)$$

$$5x+3y+z/3=100 \quad (2)$$

上述方程组有三个未知数，两个方程式，所以 $x, y, z$ 有多组解。我们可以用“穷举法”把 $x, y, z$ 可能满足要求的组合列举出来，并判断是否符合要求，最后输出符合要求的组合。

假定 $x, y$ 的值已知，那么由方程（1）可求得 $z$ 值，而 $x, y$ 可能的取值都在 0～100 之间，所以可以用二重循环来组合它们，每个 $x, y$ 的组合都得到一个相应的 $z$ 值，即 $z=100-x-y$。若 $x, y, z$ 满足方程式（2），则输出 $x, y, z$，否则不输出。

该算法的流程如图 1-6 所示。

### 2．递推法（Recurrence）

递推法是一种重要的算法设计思想。一般从已知的初始条件出发，依据某种递推关系，逐次推出所要求的各中间结果及最后结果。其中，初始条件可能由问题本身给定，也可能是通过对问题的分析与化简后确定。在实际应用中，题目很少会直接给出递推关系式，而是需要通过分析各种状态，找出递推关系式，这也是应用递推法解决问题的难点所在。递推算法可分为顺推法和逆推法两种。

（1）顺推法。从已知条件出发，逐步推算出要解决问题的方法。例如，斐波那契（Fibonacci）数列就可以通过顺推法不断推算出新的数据。

（2）逆推法。也称为倒推法，是顺推法的逆过程，该方法从已知的结果出发，用迭代表达式逐步推算出问题开始的条件。

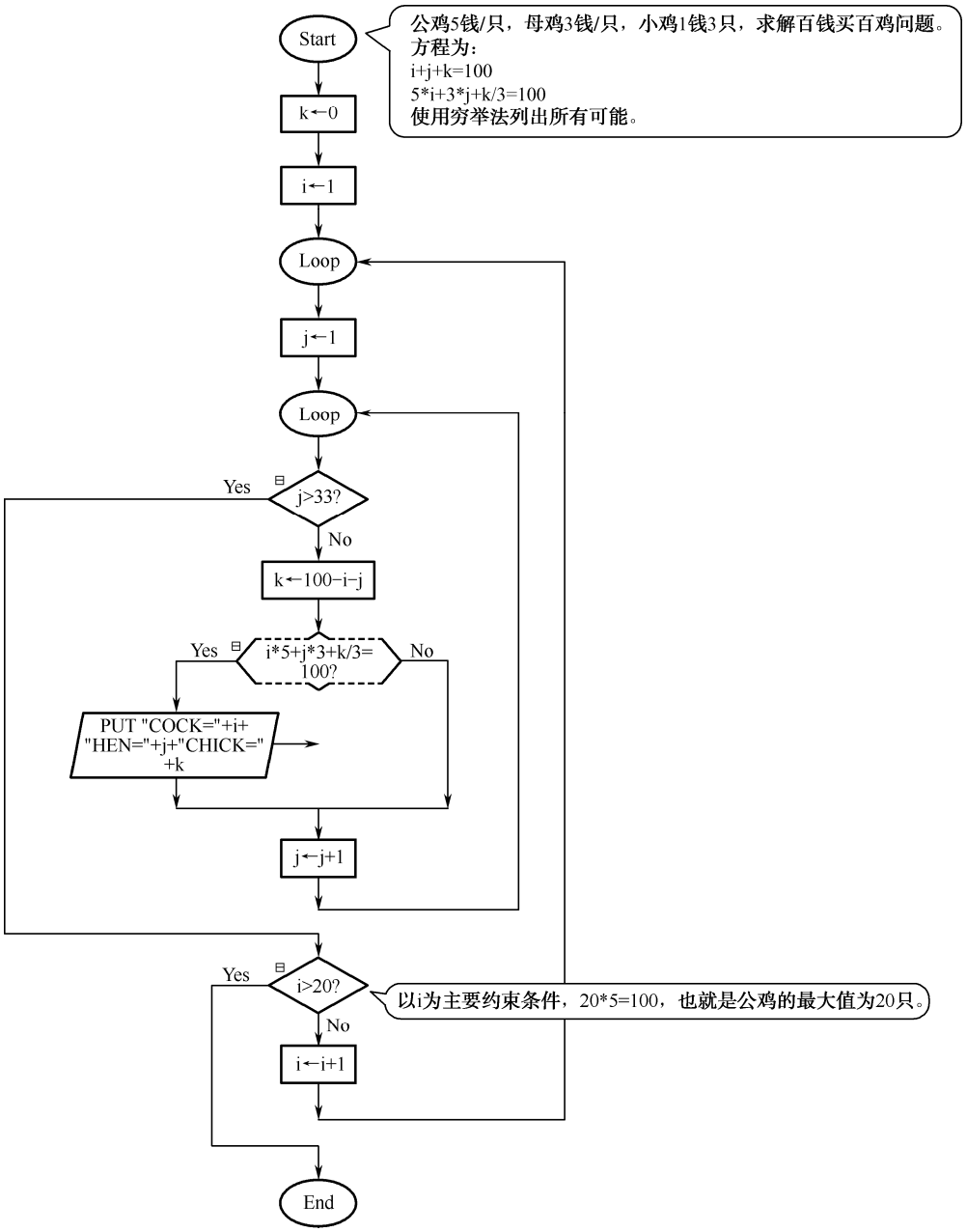



图 1-6　百钱买百鸡问题的流程图

**【例 1-7】** 使用顺推法解决 Fibonacci 数列问题。

Fibonacci 数列指的是这样一个数列：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…这个数列从第 3 项开始，每项都等于前两项之和。使用数学公式表示如下：

$$\begin{cases} f_1 = 1 & (n=1) \\ f_2 = 1 & (n=2) \\ f_n = f_{n-1} + f_{n-2} & (n \geqslant 3) \end{cases}$$

从以上的分析可知，Fibonacci 数列可使用递推算法来计算求得，图 1-7 就是使用顺推法解决 Fibonacci 数列前 12 项问题的流程图。

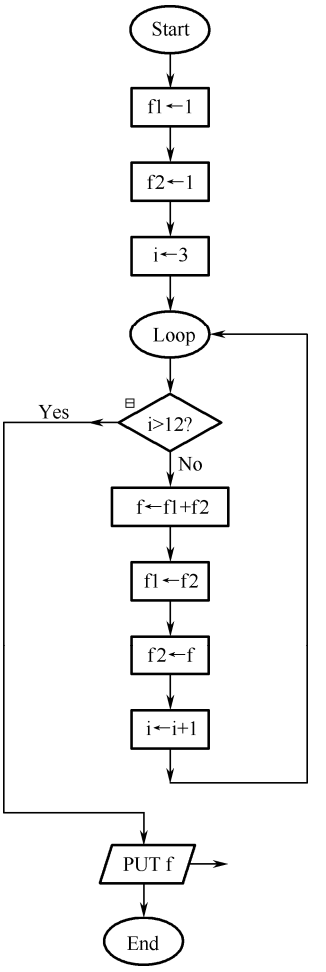



图 1-7　求解 Fibonacci 数列的流程图

**【例 1-8】** 使用逆推法解决猴子吃桃问题。猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，再多吃了一个。以后每天早上都吃了前一天剩下的一半多一个。到第 10 天早上想再吃时，只剩一个桃子了。问第一天共摘了多少桃子？

分析后可知，猴子吃桃问题的递推关系为：

$$S_n=\begin{cases}1 & (n=10)\\ 2(S_{n+1}+1) & (1\leqslant n<10)\end{cases}$$

在此基础上，以第 10 天的桃数作为基数，用以上归纳出来的递推关系设计出一个循环过程，将第 1 天的桃数推算出来，递推算法如图 1-8 所示。

### 3．排序（Sort）

将杂乱无章的数据元素，通过一定的方法按关键字顺序排列的过程称为排序。常见的基本排序算法有 5 类。

① 交换排序（Exchange Sort），如冒泡排序、快速排序等。

② 插入排序（Insertion Sort），如直接插入排序、二分插入排序等。

③ 选择排序（Selection Sort），如选择排序、推排序等。

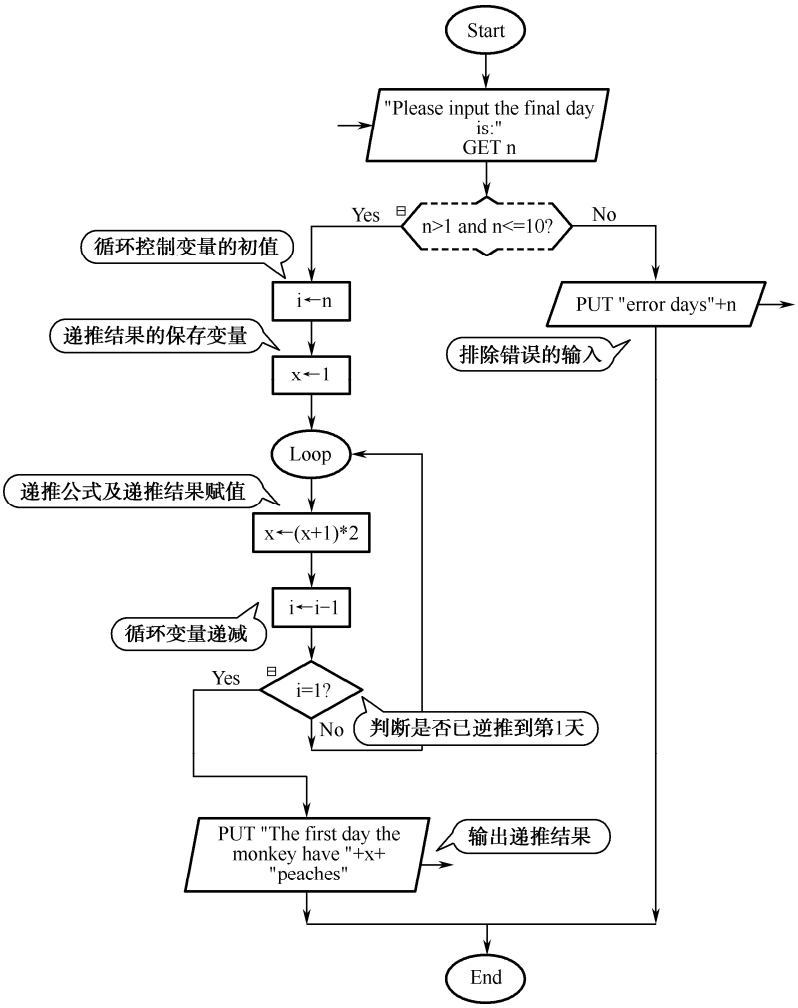



图 1-8　猴子吃桃问题的逆推算法

④ 归并排序（Merge Sort），如归并排序、多相归并排序等。

⑤ 分布排序（Distribution Sort），如桶排序、基数排序等。

**【例 1-9】** 使用冒泡排序算法将 $n$ 个数从小到大排序。

已知一组无序数据 a[1], a[2], …, a[$n$]，需要将其按升序排列。首先比较 a[1]与 a[2]的值，若 a[1]大于 a[2]，则交换两者的值，否则不变。再比较 a[2]与 a[3]的值，若 a[2]大于 a[3]，则交换两者的值，否则不变。再比较 a[3]与 a[4]，其余类推，最后比较 a[$n$−1]与 a[$n$]的值。这样处理一轮后，a[$n$]的值一定是这组数据中最大的。再对 a[1]～a[$n$−1]以相同方法处理一轮，则 a[$n$−1]的值一定是 a[1]～a[$n$−1]中最大的。再对 a[1]～a[$n$−2]以相同方法处理一轮，其余类推。共处理 $n$−1 轮后，a[1], a[2] , …, a[$n$]就以升序排列了。降序排列与升序排列的方法相类似，若 a[1]小于 a[2]，则交换两者的值，否则不变，后面类推。总的来讲，每轮排序后最大（或最小）的数将移动到数据序列的最后，理论上总共要进行 $n(n-1)/2$ 次交换。

优点：稳定；缺点：慢，每次只能移动相邻两个数据。

冒泡排序算法如图 1-9 所示。

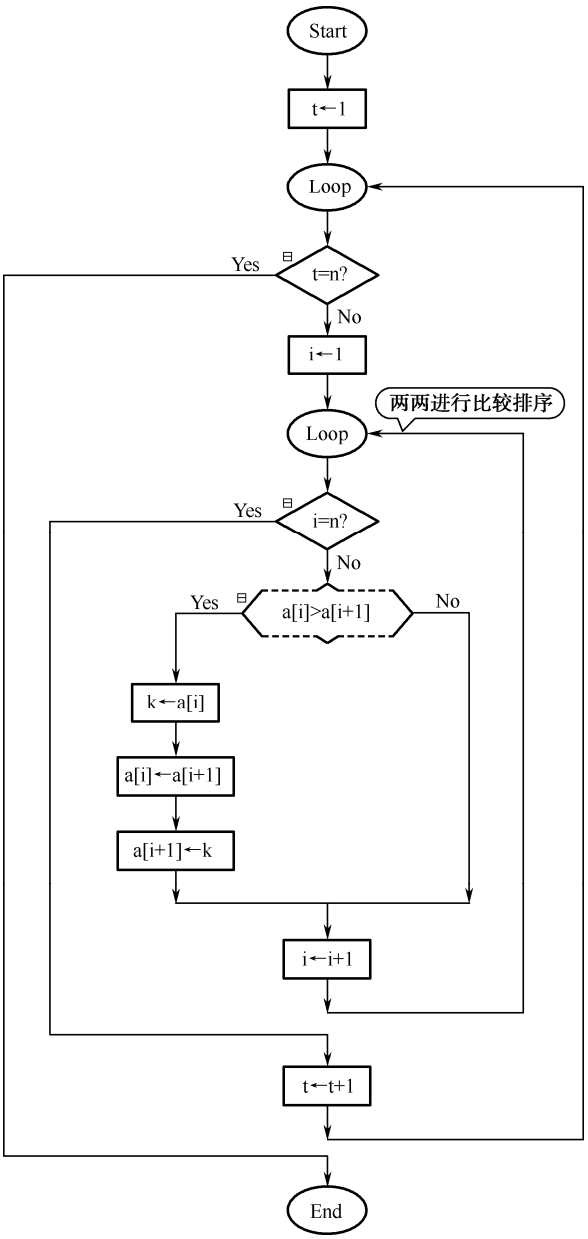



图 1-9　冒泡排序算法

#### 4．查找（Search）

在一些（有序的/无序的）数据元素中，通过一定的方法找出与给定关键字相同的数据元素的过程称为查找。基本的查找算法有以下 4 种：

① 顺序查找（Sequential Search）；

② 比较查找（Comparison Search），也称二分查找（Binary Search）；

③ 基数查找（Radix Search），也称分块查找；

④ 哈希查找（Hash Search）。

二分查找法也称为折半查找法，它充分利用了元素间的次序关系，采用分治策略，将 $n$

个元素分成个数大致相同的两半，取 a[$n$/2]与欲查找的 $x$ 进行比较，如果 $x$=a[$n$/2]则找到 $x$，算法终止；如果 $x$>a[$n$/2]，则需要在数组 a 的右半部继续搜索，直至找到 $x$ 为止，或得出关键字不存在的结论。

所以，二分法要求：

① 必须采用顺序存储结构；

② 必须按关键字大小有序排列。

二分查找的算法如图 1-10 所示，每执行一次都可以将查找空间减少一半，是计算机科学中分治思想的完美体现。其缺点是要求待查表为有序表，而有序表的特点则是插入和删除困难。因此，二分查找方法适用于不经常变动而查找频繁的有序列表。

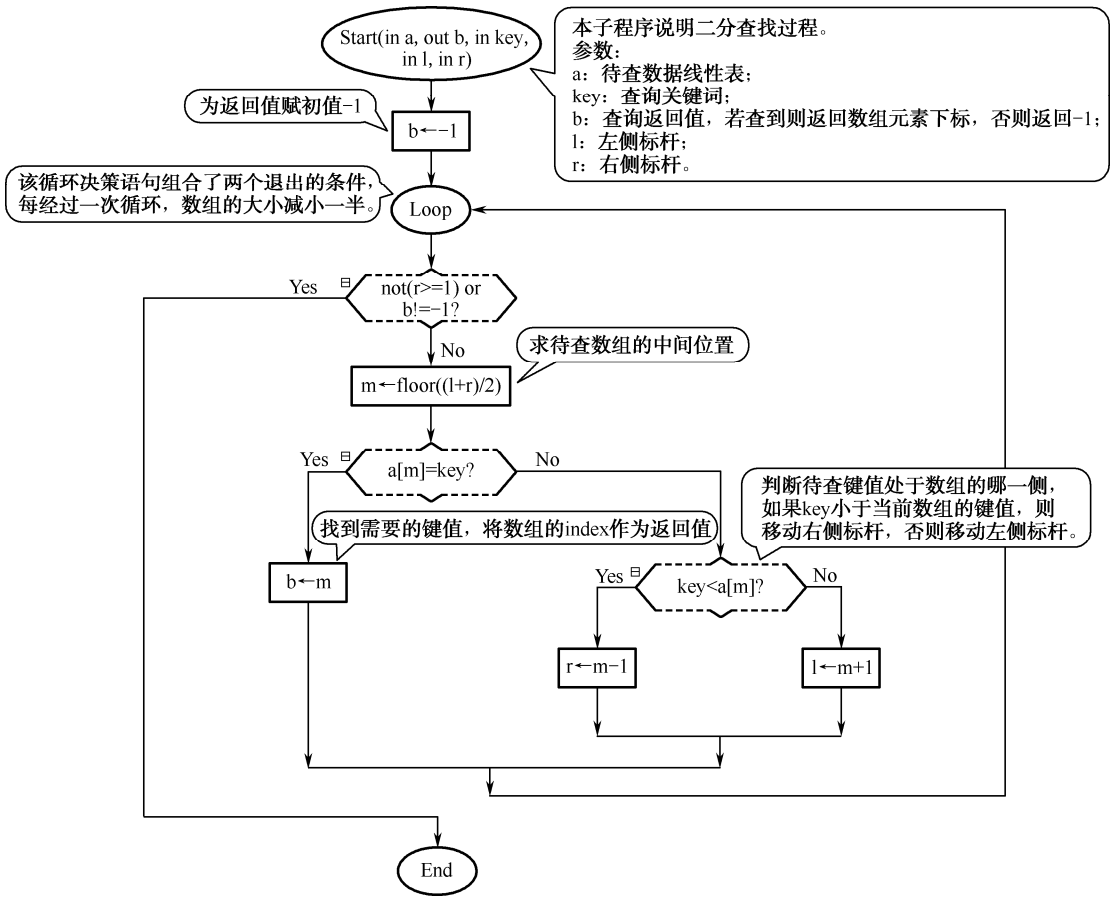



图 1-10　二分法查找算法

### 5．简单数论问题（Simple Number Theory）

数论的本质是对素数性质的研究，例如，哥德巴赫猜想是数论中存在最久的问题之一，哥德巴赫猜想可以陈述为：“任意一个大于 2 的偶数，都可表示成两个素数之和”。

**【例 1-10】** 试设计一个算法，求解 100 以内符合哥德巴赫猜想的偶数。

为了验证哥德巴赫猜想对 100 以内的正偶数都是成立的，要将整数分解为两部分，然后判断分解出的两个整数是否均为素数。若是，则满足题意；否则重新进行分解和判断。

图 1-11 显示了该算法的测素子程序，这是数论类算法的核心部分。

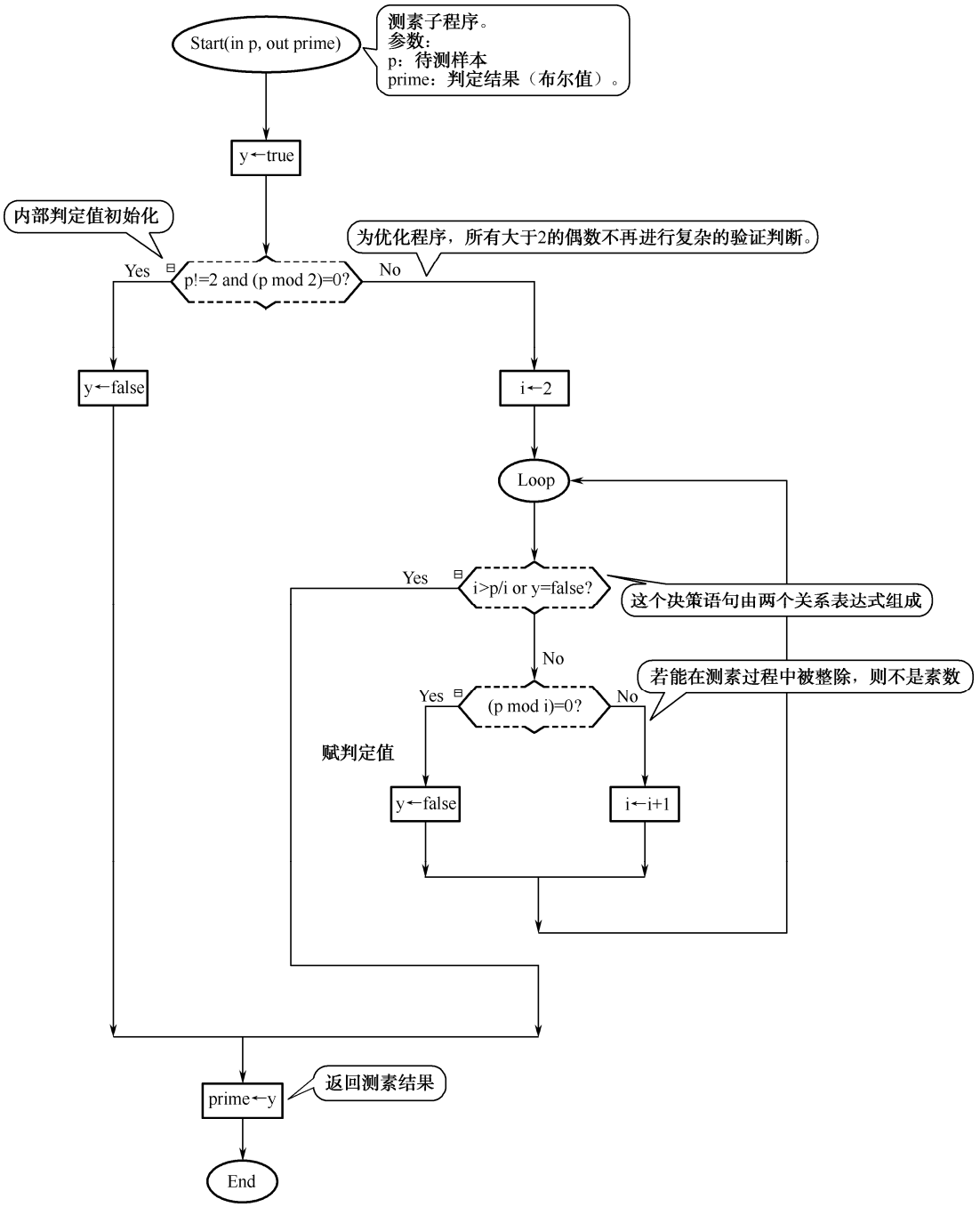



图 1-11　哥德巴赫猜想验证算法的测素子程序

图 1-12 是验证某个偶数是否符合哥德巴赫猜想的算法的 main()子图，主要包括用户交互，以及调用测素子程序判断两个数是否同为素数并输出计算结果的部分。

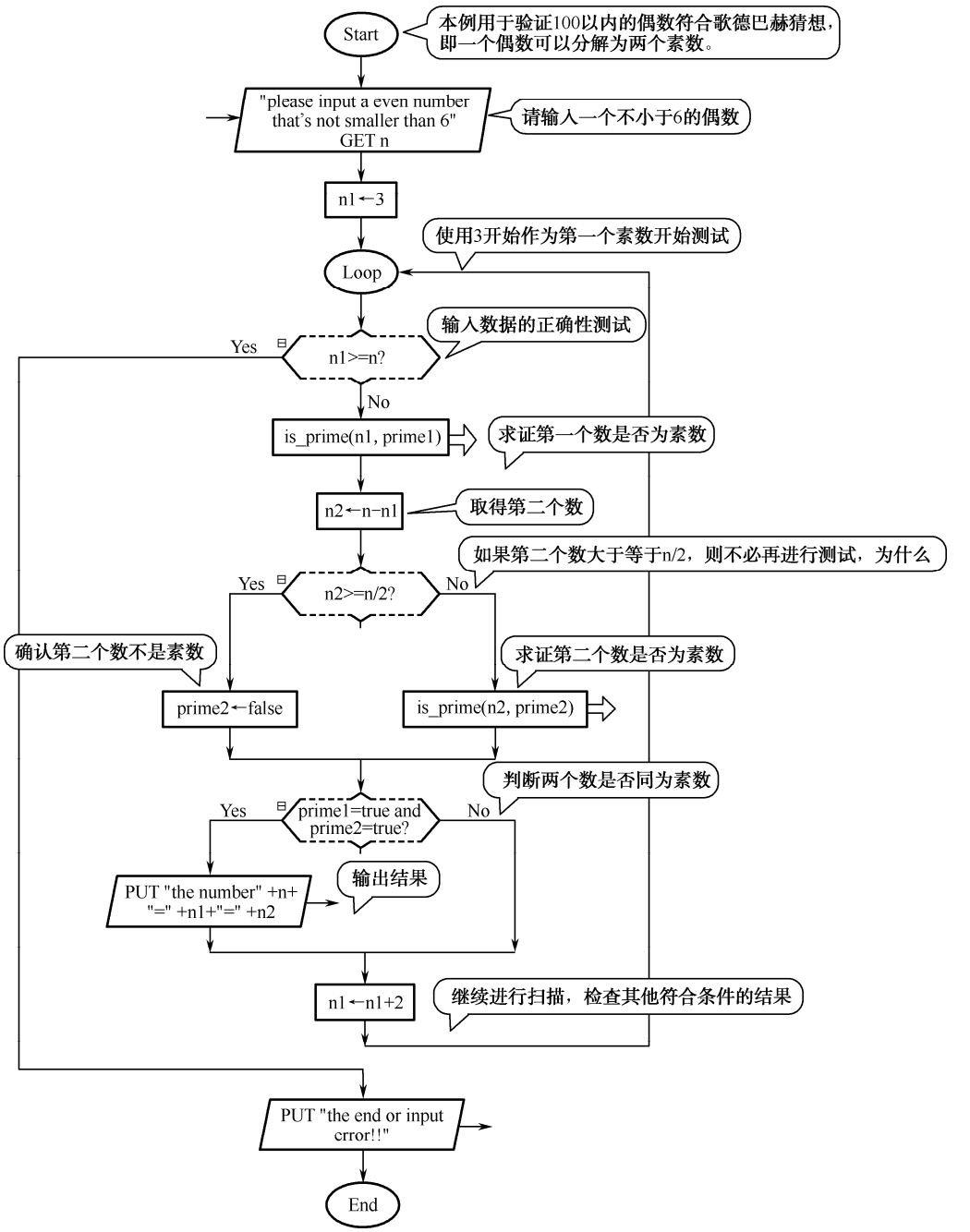



图 1-12　验证哥德巴赫猜想算法的 main()程序

## 1.3　程序与程序设计

一个程序是完成某一特定任务的一组指令序列，或者说，为实现某一算法的指令序列称为“程序”。

程序=数据结构+算法+程序设计方法+语言工具和环境

程序设计是指使用某种计算机语言，按照某种算法，编写程序的活动。程序设计往往以

某种程序设计语言为工具，给出基于这种语言的指令序列。程序设计过程包括分析、设计、编码、测试、排错等不同阶段。专业的程序设计人员常被称为程序员。

### 1.3.1 程序与程序设计语言

在《计算机软件保护条例》中将程序定义为：为了得到某种结果而可以由计算机等具有信息处理能力的装置执行的代码化指令序列，或者可被自动转换成代码化指令序列的符号化指令序列或者符号化语句序列。通俗地说，程序就是用计算机能够理解的“语言”写的“工作流程”，以便让计算机完成特定任务。

程序设计语言，通常称为编程语言，是指一组用来定义计算机程序的语法规则。简单地说，就是算法的一种描述。这种标准化的语言可以向计算机发出指令。依靠程序设计语言，人们把解决某个或者某类问题的算法，也可以说是步骤，告诉计算机，从而让计算机帮助人们解决人脑难以解决的问题。如果说计算机的硬件是身体，那么程序就是计算机的灵魂，而程序设计语言就是组成灵魂的各种概念和思想。用户能够根据自己的需求来安装不同程序，使计算机完成所需的功能，程序设计语言可以说是功不可没的。

程序设计语言的基础是一组记号和一组规则。程序设计语言一般都由三部分组成：语法、语义和语用。

语法就是在编写程序时所需要遵守的一些规则，也就是各个记号之间的组合规律。语法没有什么特殊含义，也不涉及使用者，但是编译器能够识别并编译的基础。

语义表示的就是程序的含义，也就是按照各种方法所表示的各个记号的特殊含义。程序设计语言的语义又包括静态语义和动态语义。静态语义是在编写程序时就可以确定的含义，而动态语义则必须在程序运行时才可以确定的含义。语义不清，计算机就无法知道所要解决问题的步骤，也就无法执行程序。

语用表示了构成语言的各个记号和使用者的关系，涉及符号的来源、使用和影响。语用的实现涉及语境问题。语境是指理解和设计程序设计语言的环境，包括编译环境和运行环境。

从程序设计语言发展的历程看，程序设计语言大致分为机器语言、汇编语言和高级语言，它们之间的比较见表 1-2。

#### 1．机器语言（Machine Language）

机器语言是由二进制数的 0 和 1 代码指令构成的，不同的 CPU 又有不同的指令系统。尽管机器语言可以直接被计算机识别，但这种语言却非常难以编写，难以修改，难以维护。因为人们习惯于十进制数，所以用机器语言编写程序异常困难。因此，这种语言并不利于推广。

#### 2．汇编语言（Assembly Language）

汇编语言也是面向机器的程序设计语言，具有很强的功能性，可以利用计算机硬件的所有特性，并能直接控制硬件的语言。汇编语言是机器语言的指令化，虽然汇编用语言也和机器语言一样，存在难学难用、容易出错、维护困难等缺点，但相对于机器语言，汇编语言更易于读/写、调试和修改，汇编程序翻译成的机器语言程序的效率高。在实际应用中，某些高级语言无法胜任的工作，也可以利用汇编语言来实现。汇编语言虽然还是一种面向机器的低级语言，但更能发挥出硬件的特性。

### 3．高级语言（High-level Language）

高级语言种类繁多，如目前流行的 C/C++，Java，C#，VB.NET 等语言，这些语言的语法、命令格式都各不相同。高级语言是相对机器语言、汇编语言等低级语言来说的。虽然高级语言种类多，每种语言都有各自的语法与命令格式，但高级语言最大的优点是在形式上接近自然语言和算术语言，概念上接近人们使用的概念。这样的特点使得高级语言很容易进行编写、修改和维护，通用性强，易于学习。因此，高级语言是一种面向用户的语言，即使不是程序员，也可以编写程序。高级语言并不能为计算机所识别，需要编译器的帮助。编译器既是编写程序的工具，也充当人和计算机交流的“翻译”。它可以将人们所编写的高级语言转化为计算机所能识别的语言。和汇编语言相比，高级语言并不能直接控制硬件。所以，尽管高级语言好用，但它现在并不能完全取代汇编语言。程序设计语言比较见表 1-2。

**表 1-2　程序设计语言比较**

| | 机器语言 | 汇编语言 | 高级语言 |
|---|---|---|---|
| 定义 | 二进制代码 | 反映指令功能的助记符 | 独立于机器的算法语言（表达式） |
| 硬件识别 | 可识别（唯一） | 不可识别 | 不可识别 |
| 是否可直接执行 | 可直接执行 | 不可，需要汇编、连接 | 不可，需要编译/解释、连接 |
| 特点 | 面向机器<br>占用内存少<br>执行速度快<br>使用不方便 | 面向机器<br>占用内存少<br>执行速度快<br>较为直观<br>与机器语言一一对应 | 面向问题/对象<br>占用内存大<br>执行速度相对慢<br>标准化程度高<br>便于程序交换<br>使用方便 |
| 定位 | 低级语言，极少使用 | 低级语言，很少使用 | 高级语言，种类多，常用 |

未来程序设计语言将更加简洁，人们不需要描述具体的算法，只需要告诉计算机要做什么就可以，计算机则根据人们的要求自动生成一个算法。在某种意义上，这样的计算机已经具备了智能。每个人都可以根据自己的需要来设计出最适合的程序，社会发展也必将成为一个智能化的社会。

本教材主要介绍 C 语言，也涉及 C++语言。C++是一种高效、实用的程序设计语言，它既可以进行过程化程序设计，也可以进行面向对象程序设计，是 C 语言的超集。C++语言由两部分组成：① 过程语言 C 部分；② 类和对象部分，是面向对象程序设计（OOP）的主体，如图 1-13 所示。


图 1-13　C++语言的组成

## 1.3.2　程序设计语言处理过程

计算机只能直接识别和执行用机器语言编制的程序，为使计算机能识别用汇编语言和高

级语言编制的程序，要有一套预先编制好的起翻译作用的翻译程序，把它们翻译成机器语言程序，这个翻译程序称为语言处理程序。被翻译的原始程序（用汇编语言或高级语言编制而成）称为源程序，翻译后生成的程序称为目标程序。按照不同的翻译处理方法，翻译程序有三类：汇编程序、解释程序和编译程序。

（1）汇编程序（Assembler）：从汇编语言到机器语言的翻译程序。

汇编语言处理过程包括汇编、连接和执行三个阶段，如图 1-14 所示。

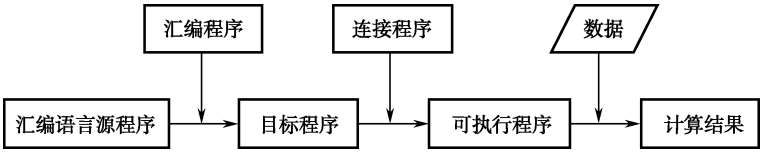



图 1-14　汇编语言处理过程

汇编语言源程序通过汇编程序翻译成目标程序；目标程序不能被直接执行，还需要把目标程序与库文件或其他目标程序通过连接程序形成计算机可执行程序；操作系统将连接后生成的可执行程序装入内存后开始执行，在这期间一般需要输入要处理的数据，执行后得到计算的结果。

（2）解释程序（Interpreter）：按源程序中指令（或语句）的执行顺序，逐条翻译并立即执行相应功能的处理程序。

解释程序对源程序从头到尾逐句扫描、逐句分析；若解释时没有发现错误，则将该语句翻译成一个或多个机器语言指令，然后立即执行这些指令；若发现错误，将会立即停止翻译，报错并提醒用户更正代码。解释程序不生成目标程序和可执行程序，如图 1-15 所示。

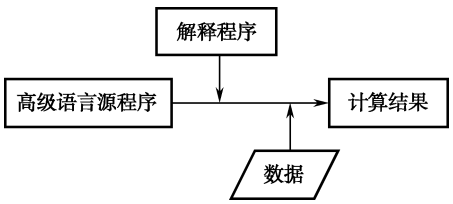



图 1-15　高级语言解释处理过程

（3）编译程序（Compiler）：从高级语言到机器语言或汇编语言的翻译程序。

编译方式的高级语言处理过程与汇编语言处理过程基本相同，如图 1-16 所示。

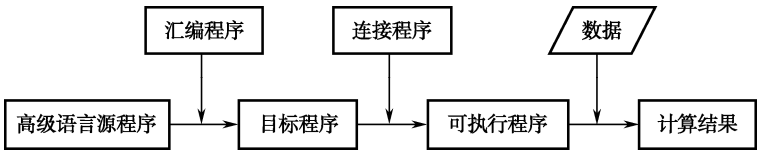



图 1-16　高级语言编译处理过程

在处理高级语言时，有些使用解释方式，如 BASIC、Python 语言等；而另外一些则使用编译方式，如 C、C++语言等，两种翻译方式比较见表 1-3。