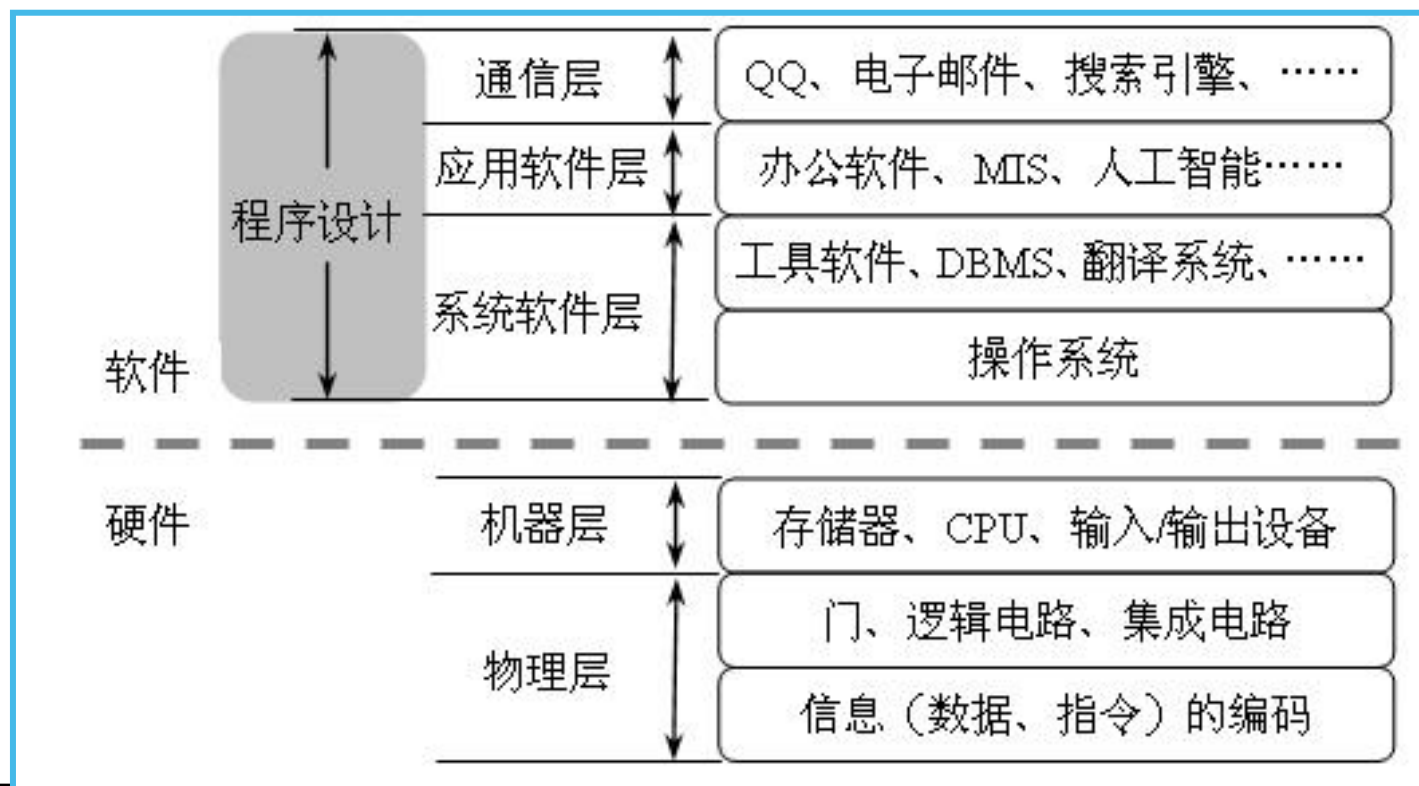


计算机科学概论

第 5 章 问题求解与程序设计

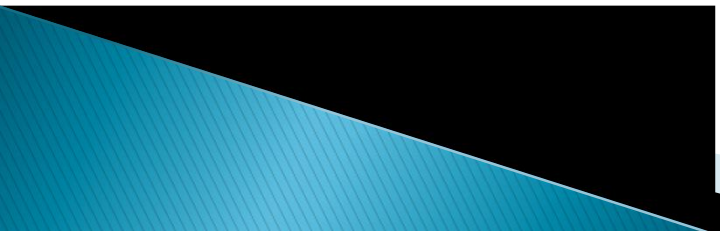
第 5章 问题求解与程序设计

程序设计在计算机系统的位置



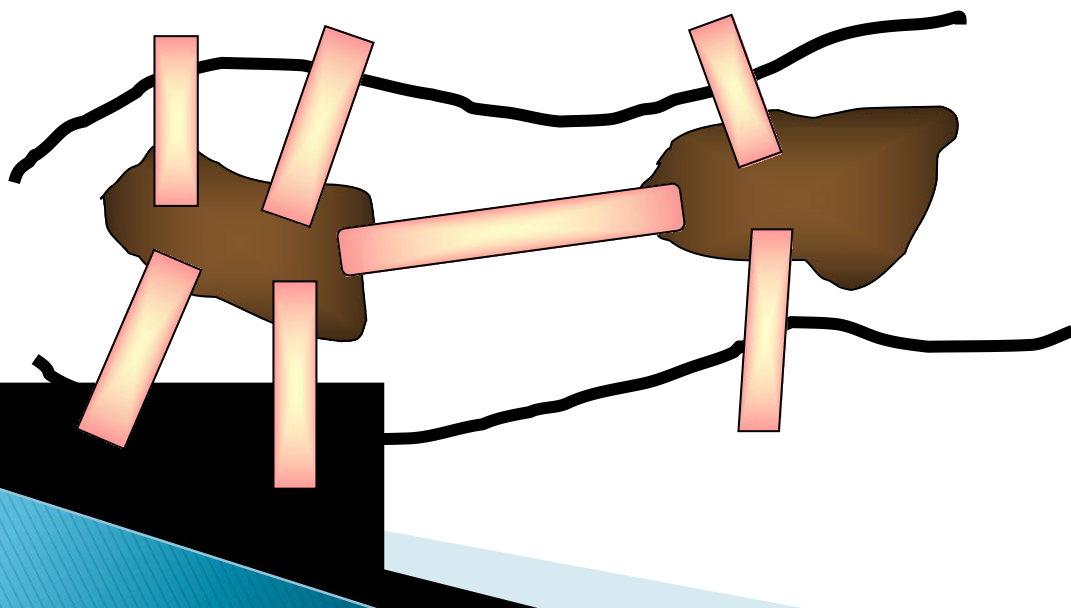
第 5 章 问题求解与程序设计

本章讨论的主要问题是：

1. 什么是程序？什么是程序设计？什么是程序设计语言？
 2. 程序是怎么设计出来的？程序设计的关键是什么？
 3. 计算机运行程序的过程就是对数据的加工处理过程，如何将数据存储到计算机的内存中？如何描述问题的处理方法和具体步骤才能让计算机“看懂”呢？
 4. 为了方便编写程序，现代程序设计普遍采用高级程序设计语言，高级语言程序如何转换为等价的机器指令？
- 

情景问题——七桥问题

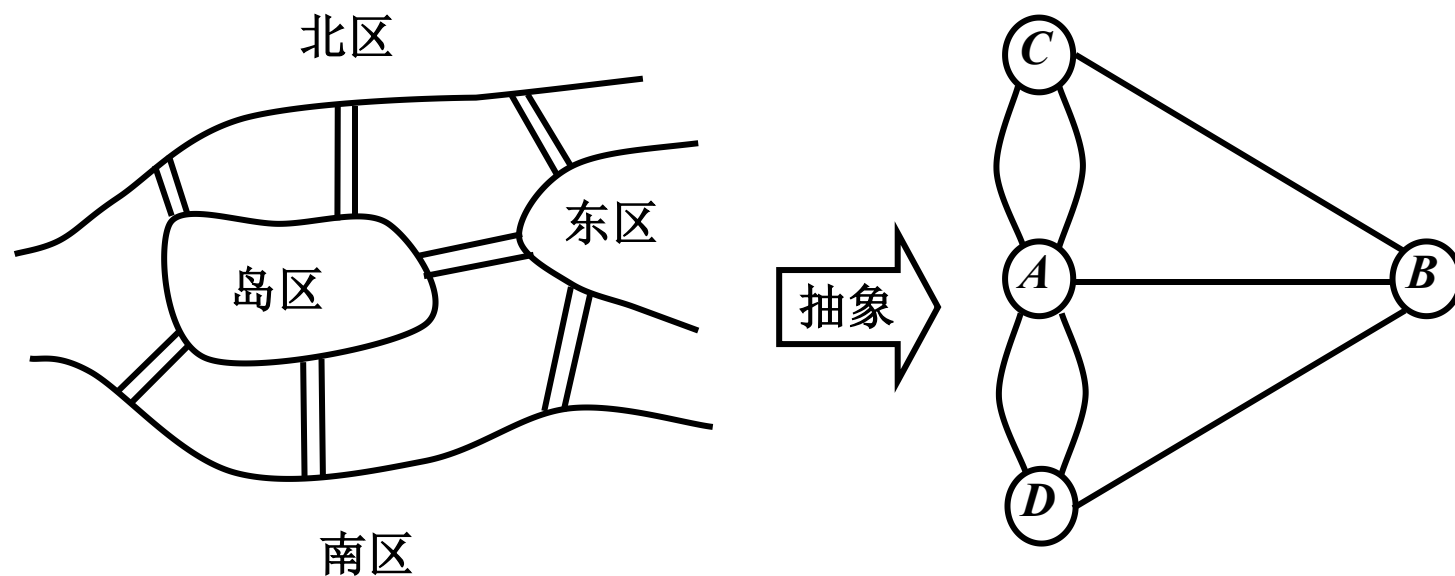
【问题】17世纪的东普鲁士有一座哥尼斯堡城（现在叫加里宁格勒，在波罗的海南岸），城中有一座岛，普雷格尔河的两条支流环绕其旁，并将整个城市分成北区、东区、南区 and 岛区4个区域，全城共有七座桥将4个城区连接起来，于是，产生了一个有趣的问题：一个人是否能在一次步行中穿越全部的七座桥后回到起点，且每座桥只经过一次。



欧拉

情景问题——七桥问题

【想法——抽象模型】 可以用A、B、C、D表示4个城区，用 7 条线表示 7 座桥，将七桥问题抽象为一个图模型。



欧拉回路： 如果一个图G中的一个路径包括每个边恰好一次，则该路径称为欧拉路径。如果一个回路是欧拉路径，则称为欧拉回路。

情景问题——七桥问题

【想法——基本思路】是否存在欧拉回路的判定规则是：

- (1) 如果通奇数桥的地方多于两个，则不存在欧拉回路；
- (2) 如果只有两个地方通奇数桥，可以从这两个地方之一出发，找到欧拉回路；
- (3) 如果没有一个地方通奇数桥，则无论从哪里出发，都能找到欧拉回路。

由上述判定规则得到求解七桥问题的基本思路：依次计算图中与每个节点相关联的边的个数（称为节点的**度**），根据度为奇数的节点个数判定是否存在欧拉回路。

情景问题——七桥问题

如何用计算机来解决这个问题？

- **【数据表示——数据结构】** 设邻接矩阵 $\text{arc}[n][n]$ 存储该图。
- **【数据处理——算法】** 算法用伪代码描述如下：

	A	B	C	D
A	0	1	2	2
B	1	0	1	1
C	2	1	0	0
D	2	1	0	0

伪代码

1. 通奇数桥的顶点个数 count 初始化为0;
2. 下标 i 从 $0 \sim n-1$ 重复执行下述操作:
 - 2.1 计算矩阵 $\text{arc}[n][n]$ 第 i 行元素之和 degree ;
 - 2.2 如果 degree 为奇数, 则 $\text{count}++$;
3. 如果 count 等于0或2, 则存在欧拉回路; 否则不存在欧拉回路;

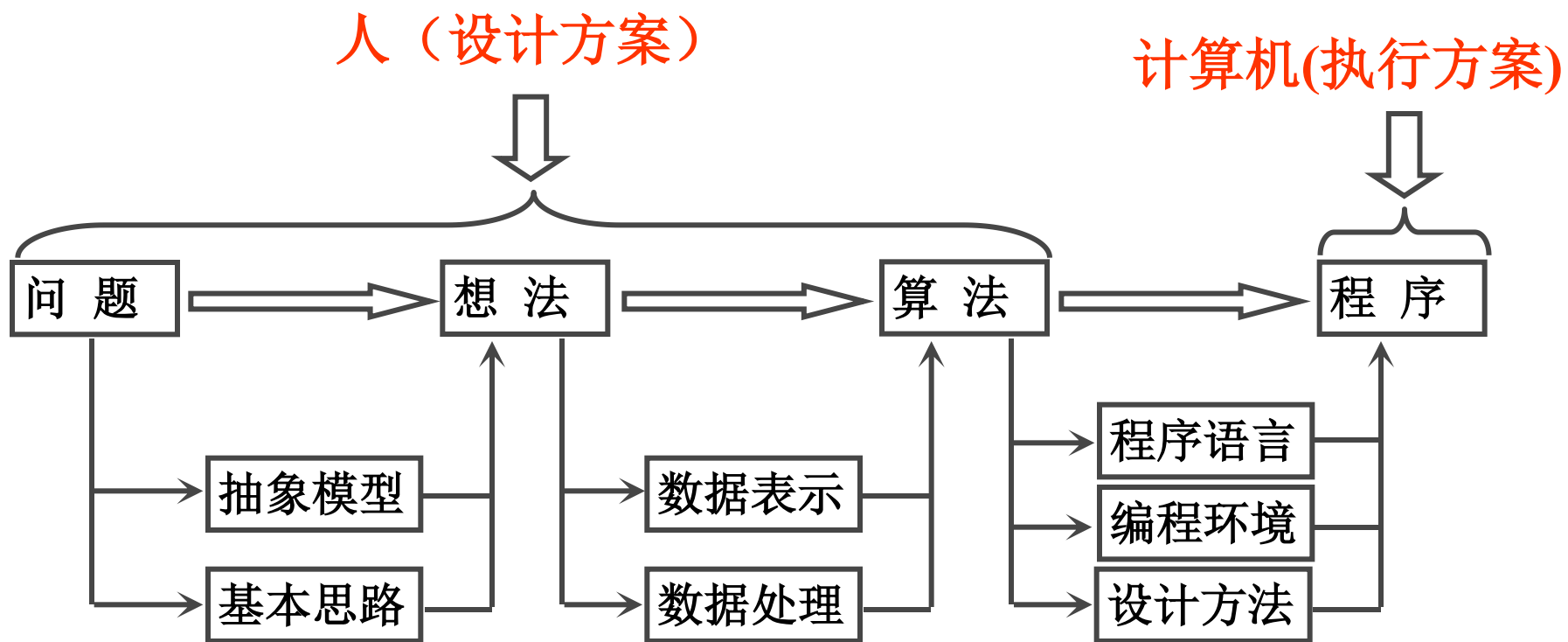
情景问题——七桥问题

【程序实现】 以下是用C语言编写的程序：

```
int EulerCircuit(int mat[10][10], int n)    //函数定义，二维数组作为形参
{
    int i, j, count = 0, degree;            //count累计通奇数桥的节点个数
    for (i = 0; i < n; i++)                //依次累加每一行的元素
    {
        degree = 0;                        // degree存储通过节点i的桥数，初始化为0
        for (j = 0; j < n; j++)            //依次处理每一列的元素
        {
            degree = degree + mat[i][j];    //将通过节点i的桥数求和
        }
        if (degree % 2 != 0)                //桥数为奇数
            count++;
    }
    return count;                           //结束函数，并将count返回到调用处
```

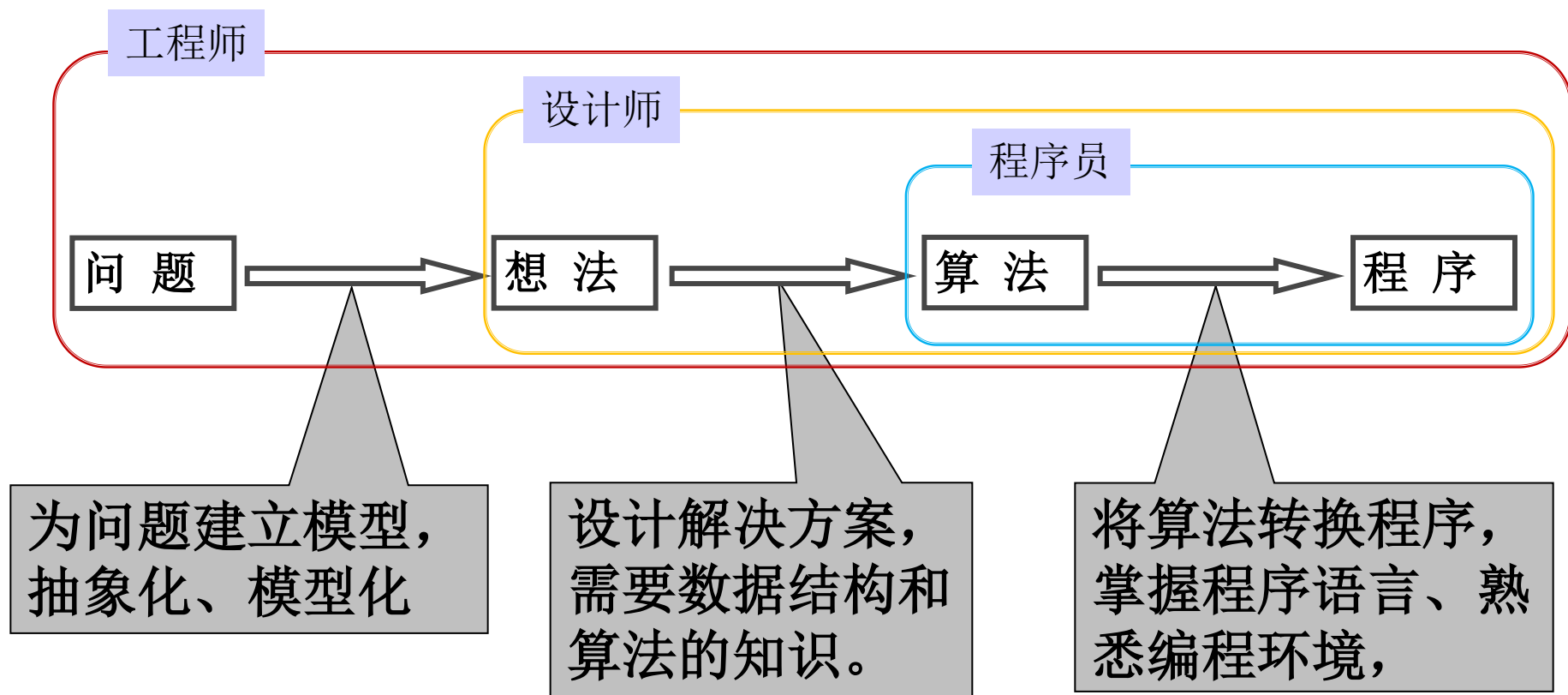

第 5章 问题求解与程序设计——程序设计

程序设计的一般过程



第 5章 问题求解与程序设计——程序设计

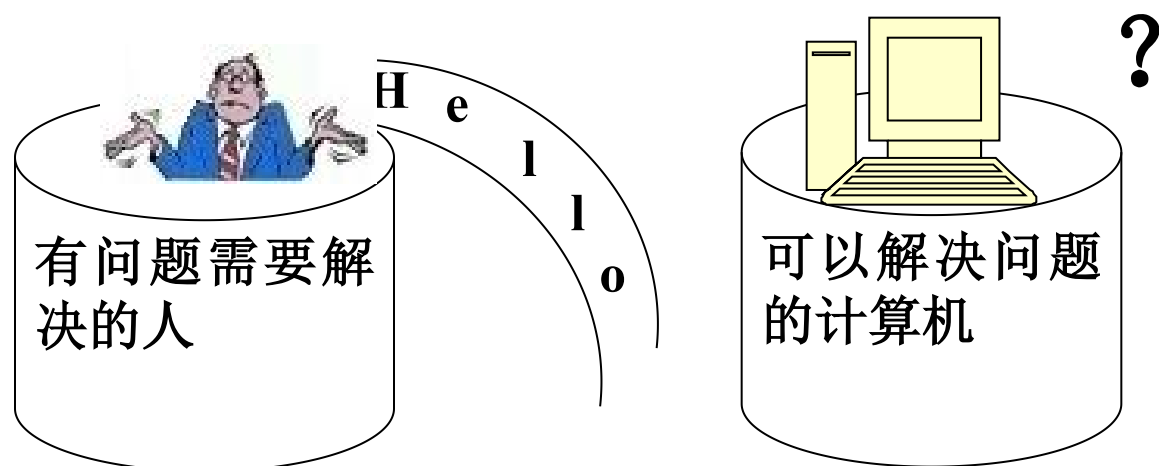
程序设计的一般过程



第 5 章 问题求解与程序设计——程序设计

理解程序

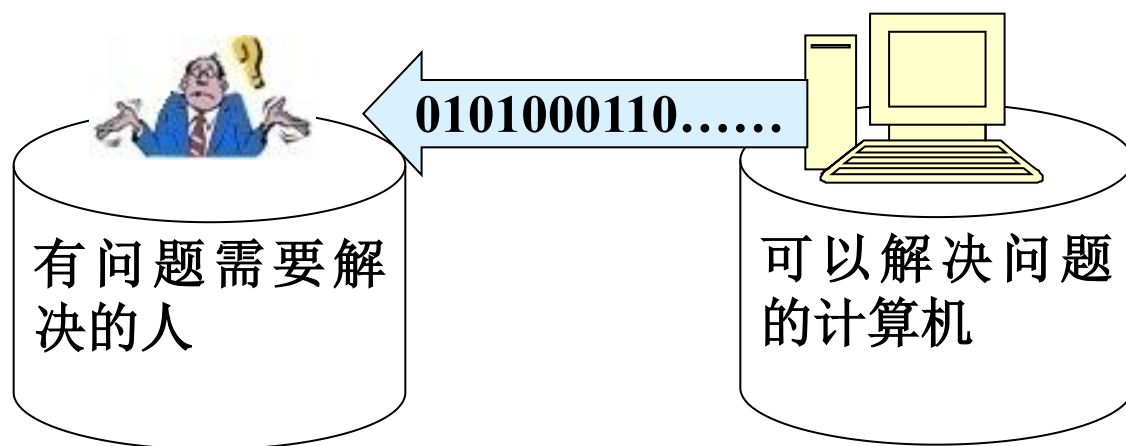
- 计算机是一个大容量、高速运转、但是没有思维的机器。
- 计算机只认识 0 和 1，听不懂人说的话——计算机如何接收人的指令？



第 5 章 问题求解与程序设计——程序设计

理解程序

- 计算机是一个大容量、高速运转、但是没有思维的机器。
- 计算机输出了 0 和 1 的编码，可是人看不懂——如何解释计算机的运算结果？

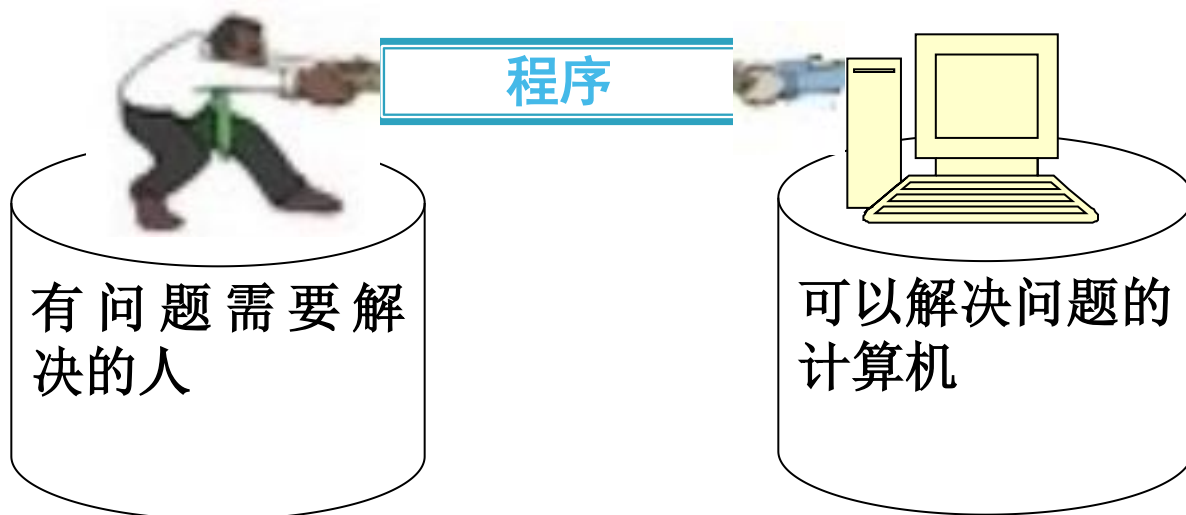


和计算机的交流？

第 5 章 问题求解与程序设计——程序设计

理解程序

程序是跨越这条鸿沟的桥梁，人要 and 计算机有效地交流，必须通过程序。



第5章 问题求解与程序设计——程序设计

- ▶ **程序**：是能够实现特定功能的一组**指令**序列的集合，是描述对某一问题的解决步骤。其中，指令可以是**机器指令**、**汇编语言**的语句，也可以是**高级语言**的语句，甚至还可以是用**自然语言**描述的指令。
- ▶ 用高级语言编写的程序称为**源程序**；用机器语言（或汇编语言）编写的程序称为**目标程序**；由二进制代码表示的程序称为**机器代码**。
- ▶ **程序设计**：是给出解决特定问题的程序的过程，是软件构造活动中的重要组成部分，程序设计往往以**某种程序设计语言为工具**，给出这种语言下的程序。
- ▶ 专业的程序设计人员常被称为**程序员**。



第 5 章 问题求解与程序设计——程序设计

程序设计的关键

- 程序设计的关键是数据表示和数据处理。
- **数据表示**完成的任务是从问题抽象出数据模型，并将该模型从机外表示转换为机内表示；
- **数据处理**完成的任务是对问题的求解方法进行抽象描述，即设计算法，再将算法的指令转换为某种程序设计语言对应的语句，转换所依据的规则就是某种程序设计语言的语法。

C: `printf("Hello, world!\n");`

C++: `cout << "Hello, World!\n";`

C#: `System.Console.WriteLine("Hello, world!");`

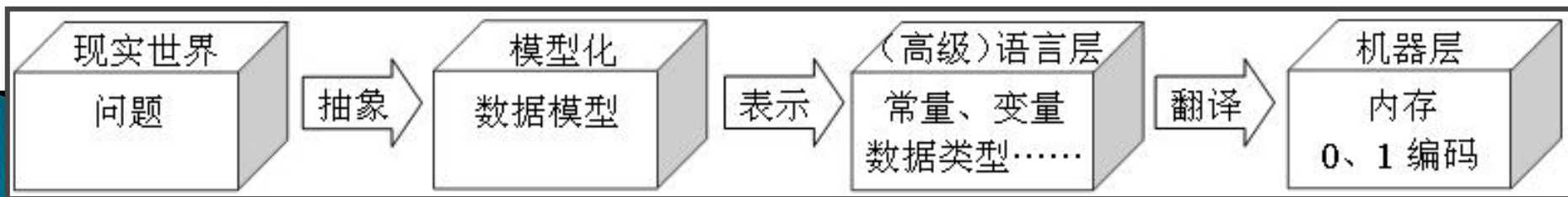
Python: `print("Hello, world!")`

Java: `System.out.println("Hello, world!");`

第5章 问题求解与程序设计——程序设计

程序设计的关键 - 数据表示

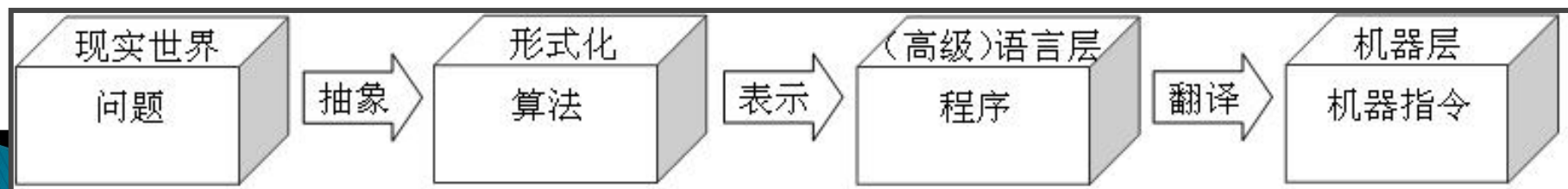
- **问题抽象**：计算机能够求解的问题一般可以分为数值问题和非数值问题，**数值问题**抽象出的数据模型通常是**数学方程**，**非数值问题**抽象出的数据模型通常是线性表、树、图等**数据结构**。
- **数据表示**意味着需要将抽象出的数据模型从**机外表示**转换为**机内表示**，也就是将数据模型存储到计算机的内存中，典型方法就是**用程序设计语言描述数据模型**。



第5章 问题求解与程序设计——程序设计

程序设计的关键-数据处理

- **程序设计**：问题的解决方案最终需要借助程序设计语言来表示，也就是将算法转换为程序，只有在计算机上能够运行良好的程序才能为人们解决特定的实际问题。
- 数据处理的核心是**算法设计**，一般来说，对不同求解方法的抽象描述产生了相应的不同算法，不同的算法将设计出不同的程序。



第5章 问题求解与程序设计——数据结构

数据结构的基本概念

- **数据**：所有能输入到计算机中并能被计算机程序识别和处理的符号集合。包括数值、字符、图形、图像、声音等。
- **数据元素**：数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。
- **数据结构（逻辑结构）**：相互之间存在一定关系的数据元素的集合，通常，数据元素之间具有以下三种基本关系：
 - （1）一对一的线性关系：线性结构；
 - （2）一对多的层次关系：树结构；
 - （3）多对多的任意关系：图结构。

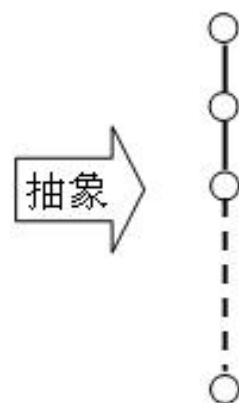
第5章 问题求解与程序设计——数据结构

数据结构的基本概念

例5.1 为学籍管理问题抽象数据模型。

学 号	姓 名	性 别	出生日期	政治面貌
0001	陆 宇	男	1986/09/02	团员
0002	李 明	男	1985/12/25	党员
0003	汤晓影	女	1986/03/26	团员
⋮	⋮	⋮	⋮	⋮

(a) 学生学籍登记表

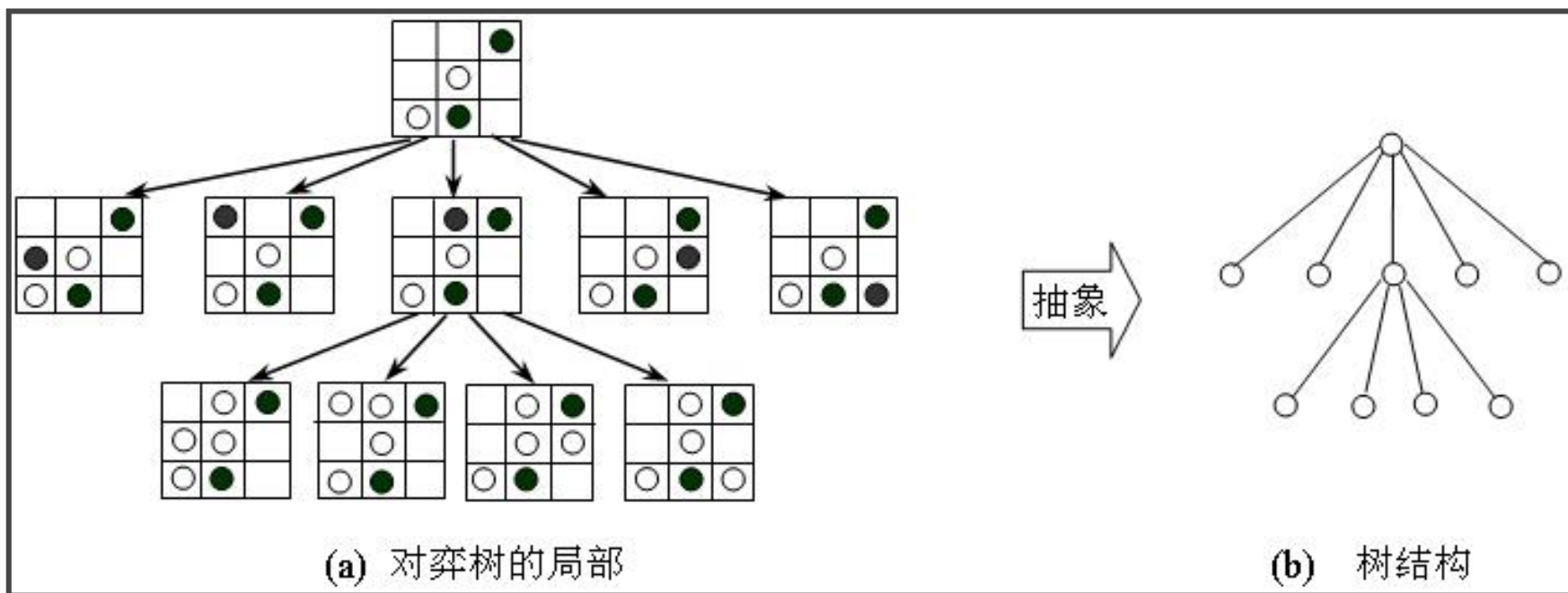


(b) 线性结构

第5章 问题求解与程序设计——数据结构

数据结构的基本概念

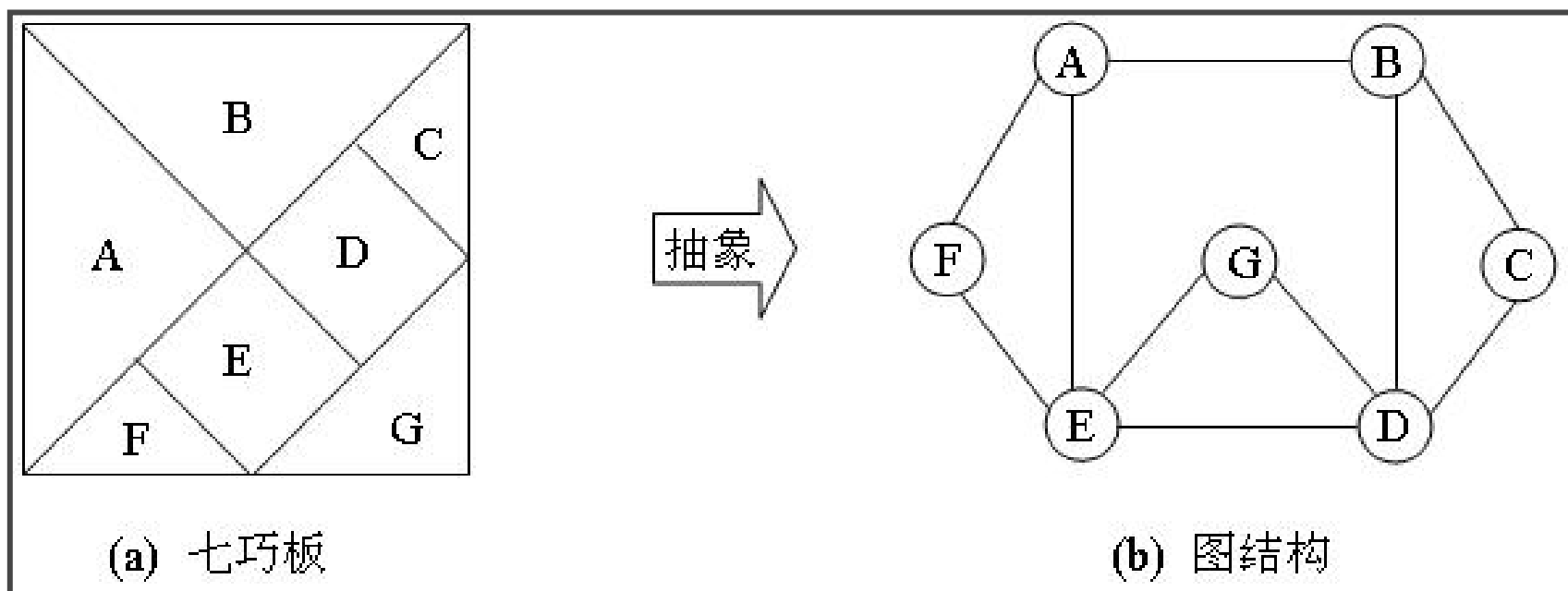
例5.2 为人机对弈问题抽象数据模型。



第5章 问题求解与程序设计——数据结构

数据结构的基本概念

例5.3 为七巧板涂色问题抽象数据模型。



七巧板涂色，要求相邻区域的颜色互不相同。

第5章 问题求解与程序设计——数据结构

抽象的数据类型

定义：抽象数据类型就是将数据与对该数据类型有意义的操作封装在一起的数据类型。然后用它封装数据和操作并对用户隐藏。

抽象数据类型的过程：

- 1、数据的定义
- 2、操作的定义
- 3、封装数据和操作。

通常用函数或类的实现。

主要的数据类型：

- 栈
- 队列
- 树
- 图

第5章 问题求解与程序设计——数据结构

抽象的数据类型——栈

栈的定义

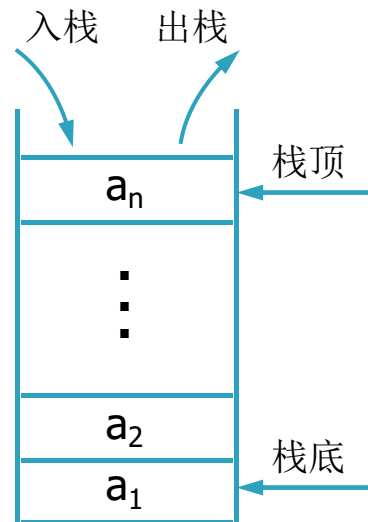
栈是限定仅在表的一端进行插入和删除操作的线性表，允许插入和删除的一端称为**栈顶**，另一端称为**栈底**。

栈的插入操作称为**入栈**，栈的删除操作称为**出栈**。不含任何数据元素的栈称为**空栈**。

特点：先进后出

栈的基本操作

操作	结果
int Length()	返回栈元素个数
bool Empty()	如栈为空，则返回true，否则返回false
void Clear()	清空栈
bool Push(T e)	插入元素e为新的栈顶元素
bool Top(T &e)	用e返回栈顶元素
bool Pop(T &e)	删除栈顶元素，并用e返回栈顶元素



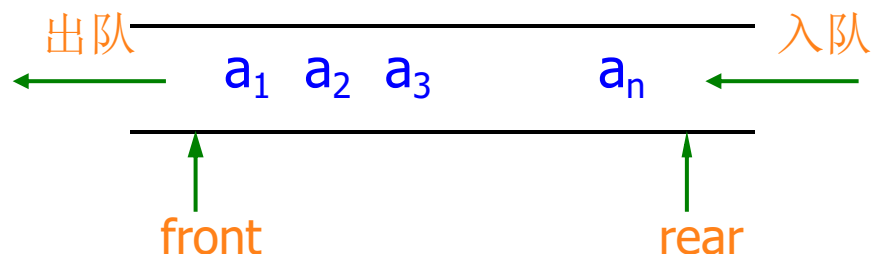
第 5 章 问题求解与程序设计——数据结构

抽象的数据类型——队列

队列的定义

队列是限定只允许在一端进行插入操作，在另一端进行删除操作的线性表。

- ◆ 允许删除（出队）的一端叫做**队头**
- ◆ 允许插入（入队）的一端叫做**队尾**
- ◆ 入队和出队的顺序相同
- ◆ 先进先出（FIFO）



操作	结果
int Length()	返回队列长度
bool Empty()	如队列为空，则返回true，否则返回false
void Clear()	清空队列
bool OutQueue (T &e)	删除队头元素，并用e返回其值
bool GetHead(T &e)	用e返回队头元素
bool InQueue(T e)	插入元素e为新的队尾

第5章 问题求解与程序设计——数据结构

抽象的数据类型——树

树的定义

树是具有相同特性的数据元素的集合，每个元素称为结点，同时包含一组有限的有向线段用来连接结点。若树是非空的，则存在一个根结点，树中的其它结点都可以沿着从根开始的唯一路径到达。

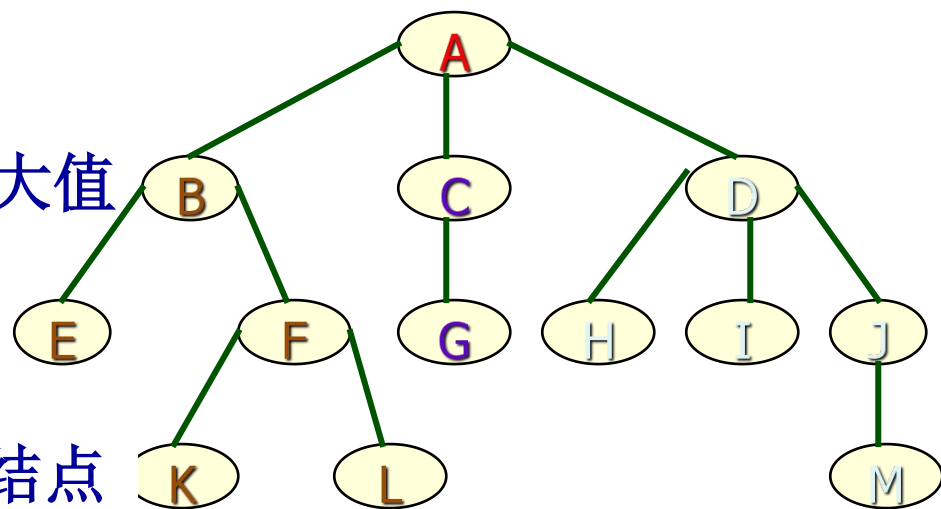
结点的度：分支的个数

树的度：树中所有结点的度的最大值

叶子结点：度为零的结点

分支结点：度大于零的结点

路径：从根到该结点所经分支和结点

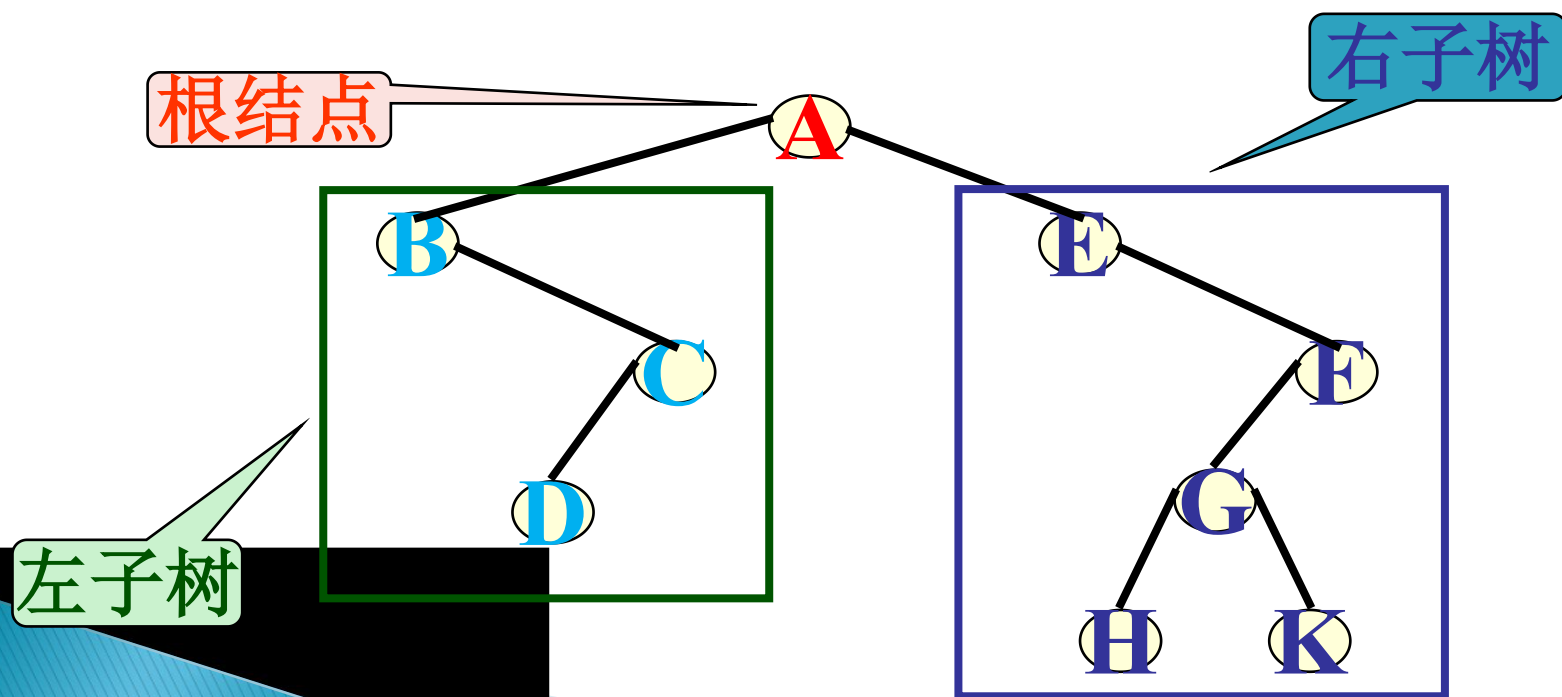


第 5 章 问题求解与程序设计——数据结构

抽象的数据类型——二叉树

二叉树的定义

二叉树或为空树，或是由一个根结点加上两棵分别称为左子树和右子树的、互不交叉的二叉树组成。



第 5 章 问题求解与程序设计——数据结构

抽象的数据类型——二叉树

➤ 二叉树的操作

包括建树、插入、删除、检索和遍历，这里我们只讲遍历。

➤ 二叉树的遍历

二叉树的遍历要求按照预定的顺序访问每一个结点且仅访问一次。常用的遍历次序是深度优先和广度优先。

➤ 深度优先遍历

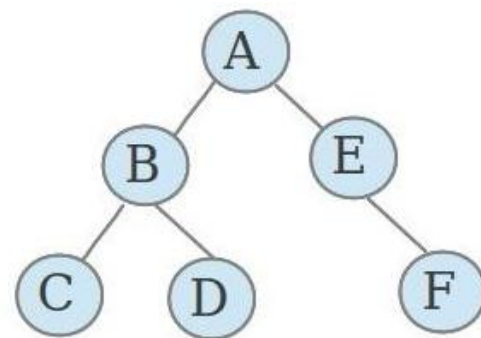
- 前序遍历：先访问根结点，再访问左子树，最后访问右子树。
- 中序遍历：先访问左子树，再访问根结点，最后访问右子树。
- 后序遍历：先访问左子树，再访问右子树，最后访问根结点。

第5章 问题求解与程序设计——数据结构

抽象的数据类型——二叉树

遍历的过程是一个迭代过程，以前序遍历为例，在任一给定结点上，可以执行三个操作：

- (1)访问结点本身；
- (2)遍历该结点的左子树；
- (3)遍历该结点的右子树。



前序遍历：ABCDEF
中序遍历：CBDAEF
后序遍历：CDBFEA

学习通练习1-4

第 5 章 问题求解与程序设计——数据结构

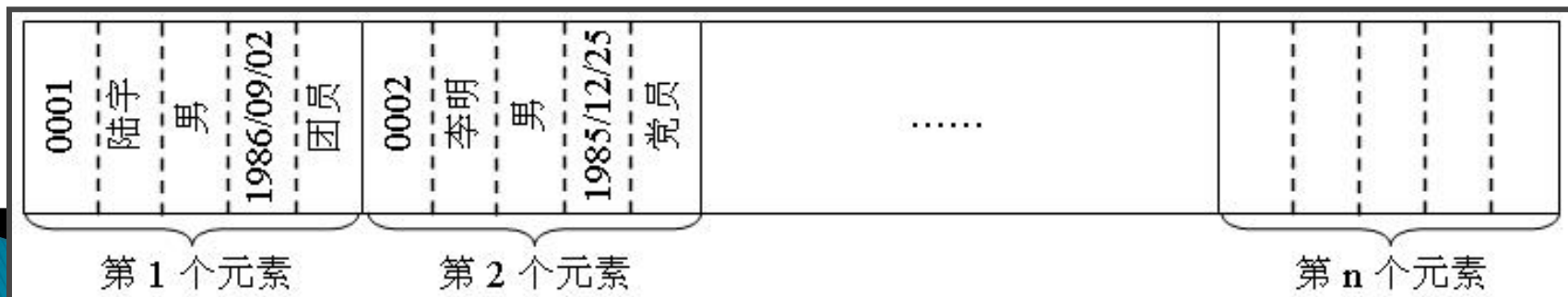
存储结构

- ▶ 为现实世界的问题建立数据模型后，还要将该模型存储在计算机的内存中，即将数据从机外表示转换为机内表示。
- ▶ 通常有两种存储表示方法：顺序存储和链接存储。
- ▶ **顺序存储的基本思想**是：用一组**连续**的存储单元**依次**存储数据元素，数据元素之间的逻辑关系由元素的**存储位置**来表示；
- ▶ **链接存储的基本思想**是：用一组**任意**的存储单元存储数据元素，数据元素之间的逻辑关系用**指针**来表示。

第 5 章 问题求解与程序设计——数据结构

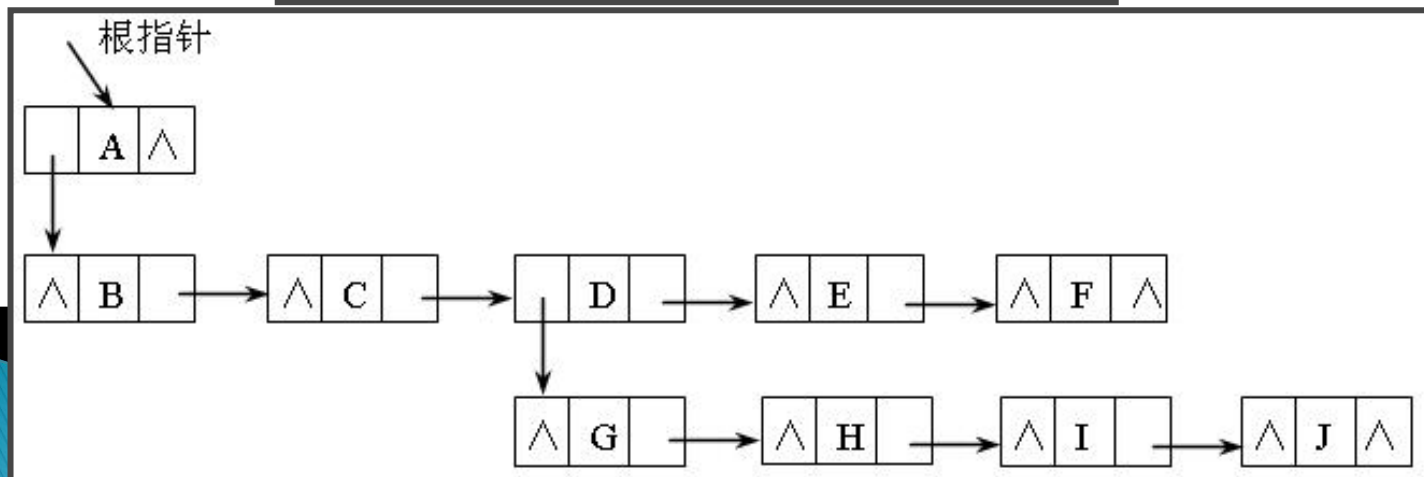
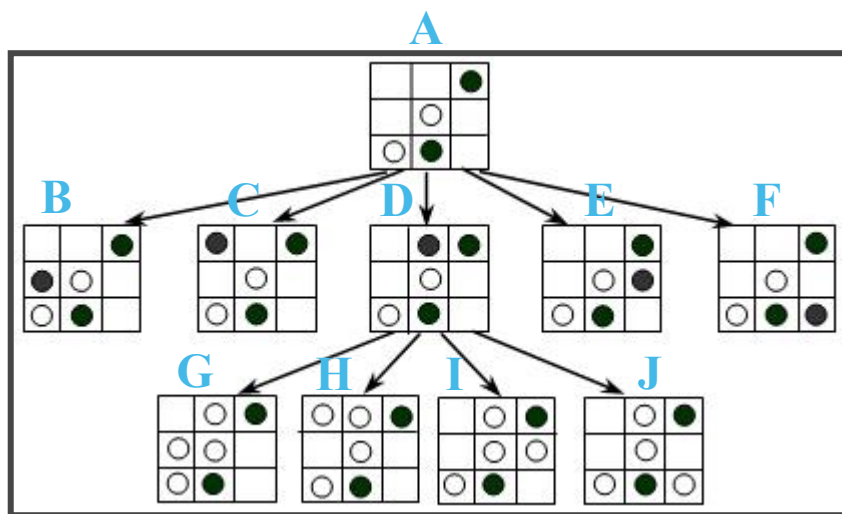
存储结构—顺序存储

学 号	姓 名	性 别	出生日期	政治面貌
0001	陆 宇	男	1986/09/02	团员
0002	李 明	男	1985/12/25	党员
0003	汤晓影	女	1986/03/26	团员
⋮	⋮	⋮	⋮	⋮



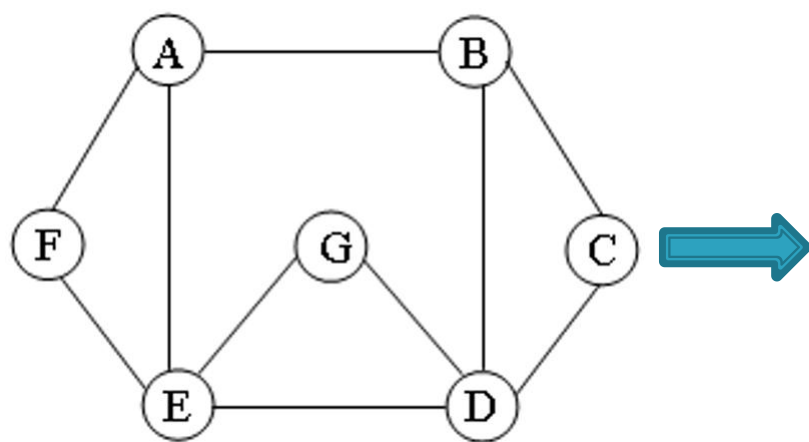
第 5 章 问题求解与程序设计——数据结构

存储结构--链式存储




第5章 问题求解与程序设计——数据结构

存储结构—邻接矩阵



图



	A	B	C	D	E	F	G
A	0	1	0	0	1	1	0
B	1	0	1	1	0	0	0
C	0	1	0	1	0	0	0
D	0	1	1	0	1	0	1
E	1	0	0	1	0	1	1
F	1	0	0	0	1	0	0
G	0	0	0	1	1	0	0

邻接矩阵

第5章 问题求解与程序设计——算法

算法的定义

■ **算法**（**Algorithm**）：对**特定问题**求解步骤的一种描述，是指令的**有限序列**。

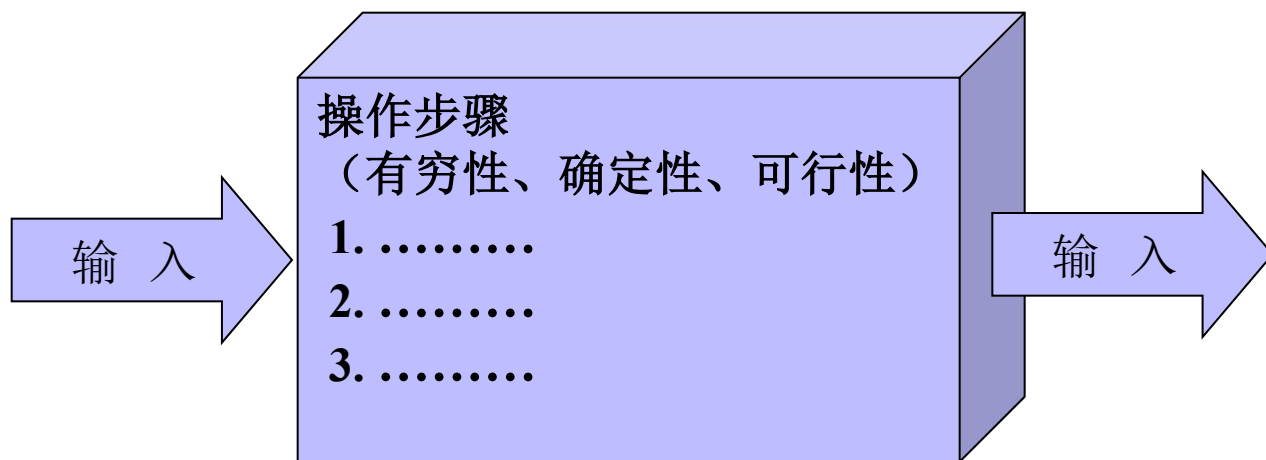
■ 算法的五大特性：

- (1) **输入**：一个算法有零个或多个输入。
- (2) **输出**：一个算法有一个或多个输出。
- (3) **有穷性**：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。
- (4) **确定性**：算法中的每一条指令必须有确切的含义，对于相同的输入只能得到相同的输出。
- (5) **可行性**：算法描述的操作可以通过已经实现的基本操作执行有限次来实现。

第 5 章 问题求解与程序设计——算法

算法的定义

■ **算法**（**Algorithm**）：对**特定问题**求解步骤的一种描述，是**指令**的**有限序列**。



第 5 章 问题求解与程序设计——算法

描述算法

算法的设计者在构思和设计了一个算法之后，必须清楚准确地将所设计的**求解步骤**记录下来，即**描述算法**。通常用伪代码来描述算法。

- **伪代码**是介于自然语言和程序设计语言之间的方法，保留了程序设计语言严谨的结构、语句的形式和控制成分，忽略了繁琐的变量说明，在抽象地描述算法时一些处理和条件允许使用自然语言来表达。至于算法中自然语言的成份有多少，取决于算法的**抽象级别**。
- 由于伪代码书写方便、格式紧凑、容易理解和修改，因此被称为**算法语言**。

第5章 问题求解与程序设计——算法

描述算法

例5.4 设计算法，在含有 n 个元素的集合中查找最大值元素。

解：设最大值为 \max ，可以假定第1个元素为最大值元素，依次将第2、3、……、 n 个元素与 \max 比较， \max 中保存的始终是每次比较后的最大值元素，算法用伪代码描述如下：

伪代码

step1: $\max \leftarrow$ 第1个元素；

step2: 初始化被比较元素的序号 $i \leftarrow 2$ ；

step3: 当 i 小于等于 n 时重复执行下述操作：

 step3.1: 如果第 i 个元素大于 \max ，则 $\max \leftarrow$ 第 i 个元素；

 step3.2: $i \leftarrow i + 1$ ；

step4: 输出 \max ；

第 5 章 问题求解与程序设计——算法

描述算法

例5.5 设计算法，实现用欧几里德算法查找最大公约数。

解：设两个自然数是 m 和 n 并满足 $m \geq n$ ，欧几里德算法的基本思想是将 m 和 n 辗转相除直到余数为0。例如， $m = 35$ ， $n = 25$ ， m 除以 n 的余数用 r 表示，计算过程如下：

除数 m	被除数 n	余数 r
35	25	10
25	10	5
10	5	0

当余数 r 为0时，被除数 n 就是 m 和 n 的最大公约数。

第5章 问题求解与程序设计——算法

描述算法

例5.5 设计算法，实现欧几里德算法。

伪代码

step1: $r \leftarrow m \bmod n$;

step2: 循环直到 r 等于0

 step2.1: $m \leftarrow n$;

 step2.2: $n \leftarrow r$;

 step2.3: $r \leftarrow m \bmod n$;

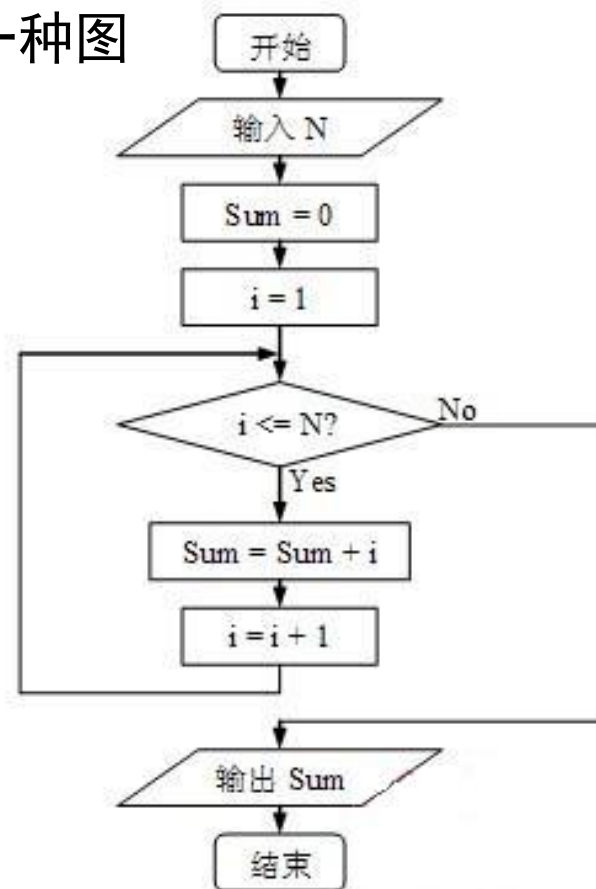
step3: 输出 n ;

第 5 章 问题求解与程序设计——算法

描述算法

流程图表示

- 用特定的图形加上说明来描述算法的一种图
- 优点
 - 能够清晰地看出算法的执行过程
 - 使用标准中规定流程线
- 基本元件
 - 矩形框
 - 菱形框
 - 箭头线
 - 圆角矩形
 - 平行四边形



第5章 问题求解与程序设计——算法

▶ 算法结构与表示

◦ 算法的控制结构

- 顺序结构

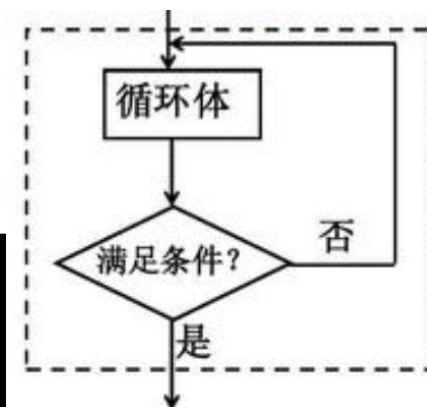
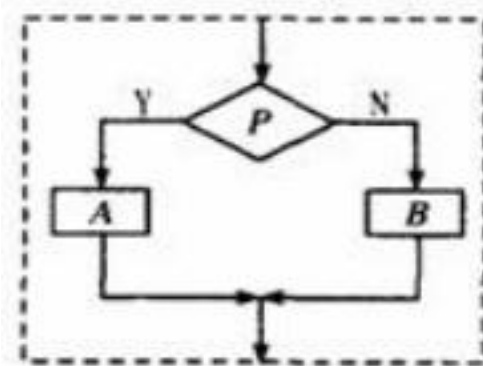
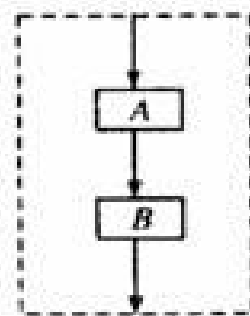
- 若干个依次执行的处理步骤组成的

- 条件分支结构

- 先根据条件做出判断，再决定执行哪一种操作的结构

- 循环结构

- 需要重复执行的同一操作的结构



第5章 问题求解与程序设计——算法

算法分析

算法分析：对算法所需要的计算机资源进行估算。

➤时间复杂度（**Time Complexity**）

➤空间复杂度（**Space Complexity**）

撇开与计算机软硬件有关的因素，影响算法时间代价的最主要因素是**问题规模**。

■**问题规模**：输入量的大小。一般来说，它可以从问题描述中得到。例如，找出100以内的所有素数，问题规模是100。

■一个显而易见的事实是：几乎所有的算法，对于规模更大的输入需要运行更长的时间。例如，找出10 000以内的所有素数比找出100以内的所有素数需要更多的时间。所以运行算法所需要的时间 T 是问题规模 n 的函数，记作 $T(n)$ 。

第 5 章 问题求解与程序设计——算法

算法分析

- 为了客观地反映一个算法的执行时间，可以用算法中**基本语句**的执行次数来度量算法的工作量。
- 基本语句**：执行次数与整个算法的执行次数成正比的操作指令。
- 基本语句对算法运行时间的贡献最大，是算法中最重要的操作。
- 分析算法的时间复杂度的基本方法是：找出所有语句中执行次数最多的那条语句作为**基本语句**，计算基本语句的**执行次数**，取其**数量级**放入大 O 中即可。
- 这种衡量效率的方法得出的不是时间量，而是一种**增长趋势的度量**。

第5章 问题求解与程序设计——算法

算法分析

例5.6 分析下列算法的时间复杂度。

伪代码

```
step1: max  $\leftarrow$  第1个元素;  
step2: 初始化被比较元素的序号  $i \leftarrow 2$ ;  
step3: 当  $i$  小于等于  $n$  时重复执行下述操作:  
    step3.1: 如果第  $i$  个元素大于 max, 则 max  $\leftarrow$  第  $i$  个元素;  
    step3.2:  $i \leftarrow i + 1$ ;  
step4: 输出 max;
```

解：基本语句是step3.1的比较语句，即第 i 个元素是否大于 max，该语句需要重复执行 $n - 1$ 次，因此，算法的时间复

第 5 章 问题求解与程序设计——程序语言

程序设计语言的发展

程序设计语言的发展是一个不断演化的过程，其根本的推动力是对抽象机制的更高的要求，以及对程序设计思想的更好的支持。

第一代程序设计语言（1GL）——机器语言

```
01010000101001010001010010101110101  
00001010010101010010100101000010100  
10100010100101011101010000101001010  
10111010100001010010101010010100101  
00001010010100010100000101001010001  
01001010111010100001000010100101000  
101001010111010100001.....
```

如果不小心弄错了一个二进制位，该如何找出来？

机器语言

- ▶ 机器语言：固定在计算机硬件中的二进制代码，可以由硬件直接执行。每种处理器都有自己专用的机器指令集合。计算机能够执行用机器语言编写的程序。

10110000 00000101	； 将5放进累加器acc中
00101100 00000110	； 累加器的值与6相加，
	结果仍然在累加器中
11110100	； 停机结束

机器语言

▶ 存在的问题：

- 用机器语言编写的程序难以阅读和理解
- 程序员必须记住复杂的指令系统，书写、辨认冗长的二进制代码
- 编出的程序只能在某个特定类型的计算机上运行
- 极大地限制了程序的质量和 application 范围

第5章 问题求解与程序设计——程序语言

程序设计语言的发展

程序设计语言的发展是一个不断演化的过程，其根本的推动力是对抽象机制的更高的要求，以及对程序设计思想的更好的支持。

第二代程序设计语言（2GL）——汇编语言

```
MOV BX, 12H  
SHL BX, 1  
MOV AX, BX  
SHL BX, 1  
SHL BX, 1  
ADD BX, AX  
.....
```

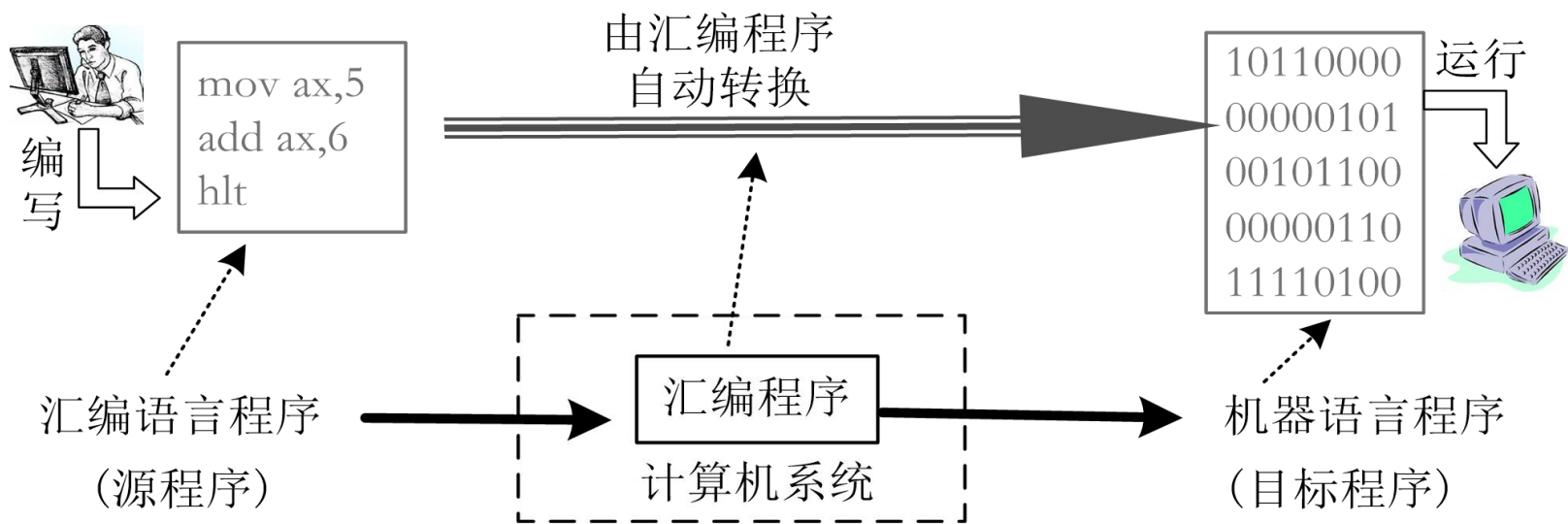
例如，ADD表示机器指令00000100。相对于机器语言，汇编语言简化了程序编写，而且不容易出错

汇编语言

- ▶ **汇编语言：**给每条机器指令分配一个助记符，即用一个与指令代码实际含义相近的英文单词、字母或数字等符号来代替指令代码（如用ADD来表示运算符‘+’）

- **缺点：**不同机器的汇编指令不同，**程序不能移植**；程序员需要记忆指令；对硬件直接操作。

```
mov ax,5   ; 将5放进ax寄存器中
add ax,6   ; ax的值与6相加,
           ; 结果仍然在ax中
hlt        ; 停机结束
```



第5章 问题求解与程序设计——程序语言

程序设计语言的发展

程序设计语言的发展是一个不断演化的过程，其根本的推动力是对抽象机制的更高的要求，以及对程序设计思想的更好的支持。

第三代程序设计语言（3GL）——高级语言

- **高级语言**：高级程序设计语言，相应地，机器语言和汇编语言称为**低级语言**，低级意味着要求程序员从机器的层次上考虑问题。
- **结构化程序设计语言**：BASIC、PASCAL、C等。
- **面向对象程序设计语言**：Java、C++、C#等。
- **可视化程序设计语言**：VB、Delphi、Visual C++等。
- **网络程序设计语言**：ASP、PHP和JSP等。

高级语言的概念

- ▶ 高级语言又称算法语言，与人类的思维和交流方式更接近，用表达各种意义的单词和公式，按照一定的语法规则来编写程序。它包括上千种语言，如Pascal、C、Python、C++、Java和VB.NET等。克服了汇编语言的缺陷，提高编写和维护程序的效率。

- ▶ 特点：

- 独立于计算机硬件结构，所编写的程序与在什么型号的计算

```
//FirstProg.cpp
main()
{
    int first,second,sum; //定义3个变量
    first=5;              //为第一个变量赋值
    second=6;             //为第二个变量赋值
    sum=first+second;     //将两个变量相加，放到第三个变量sum中
    cout<<sum;           //输出所求的和
}
```

高级语言

- ▶ 高级语言的**优点**是语句功能强，程序员编写的源程序比较短，容易学习，使用方便，可移植性好，便于推广和交流。
- ▶ **缺点**是编译程序比汇编程序复杂，而且编译出来的目标程序往往效率不高，目标程序的长度比有经验的程序员所编的同样功能的汇编语言程序要长一倍以上，运行时间也要长一些。

第5章 问题求解与程序设计——程序语言

程序设计语言的发展

程序设计语言的发展是一个不断演化的过程，其根本的推动力是对抽象机制的更高的要求，以及对程序设计思想的更好的支持。

第四代程序设计语言（4GL）——非过程式语言

- 4GL开发软件时只需要考虑“做什么”而不必考虑“如何做”，不涉及太多的算法细节，从而大大提高软件生产率。
- 到目前为止，使用最广泛的4GL是数据库查询语言，许多大型数据库语言如Oracle、Sybase、Informix等都包含有4GL成分。

第5章 问题求解与程序设计——程序语言

程序设计语言的发展

程序设计语言的发展是一个不断演化的过程，其根本的推动力是对抽象机制的更高的要求，以及对程序设计思想的更好的支持。

第五代程序设计语言（5GL）——知识型语言

- 5GL主要应用在人工智能研究上，典型代表是LISP语言和PROLOG语言。
- 目前，4GL和5GL的发展都不是很成熟，在效率、应用等方面都存在诸多问题，常用的程序设计语言仍然是3GL。

第5章 问题求解与程序设计——程序语言

程序设计语言的基本要素

自然语言与程序设计语言的类比：文章由段落、句子、单词和字母组成，类似地，程序设计语言的一个程序由模块、语句、单词和基本字符组成。

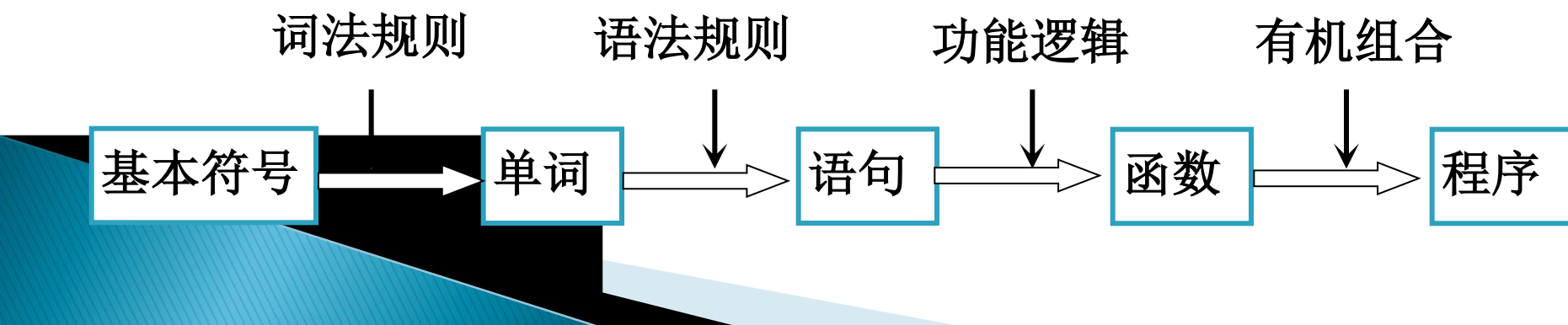
文章——程序

段落——模块

句子——语句

单词——单词

字母——基本字符



第5章 问题求解与程序设计——程序语言

程序设计语言的基本要素

- 程序设计语言由语法和语义两方面定义的。
- **语法**包括**词法**规则和**语法规则**，**词法规则**规定了如何从语言的基本符号构成词法单位（也称单词），**语法规则**规定了如何由单词构成语法单位（例如语句），这些规则是判断一个字符串是否构成一个形式上正确的程序的依据；
- **语义**规则规定了各语法单位的具体含义，程序设计语言的语义具有上下文无关性，程序文本所表示的语义是单一的、确定的。
- 从某种角度说，学习程序设计语言主要就是学习这些规则。

第5章 问题求解与程序设计——程序语言

实际上，所有程序设计语言的最终目的都是控制计算机按照人们的意愿去工作。

各种各样的程序设计语言具有共同的基本内容，无论哪一种程序设计语言，都是以

- 数据的表示（常量、变量、数据类型等）
- 数据的组织（数组、结构体、类等）
- 数据处理（赋值运算、算术运算、逻辑运算等）
- 程序的流程控制（顺序、分支、循环等）
- 数据传递（全局变量、函数调用、消息传递等）

为基本内容，只是不同的语言采用不同的方法实现上述基本内容，体现为不同的程序设计语言具有不同的表述格式（即语法规则）。

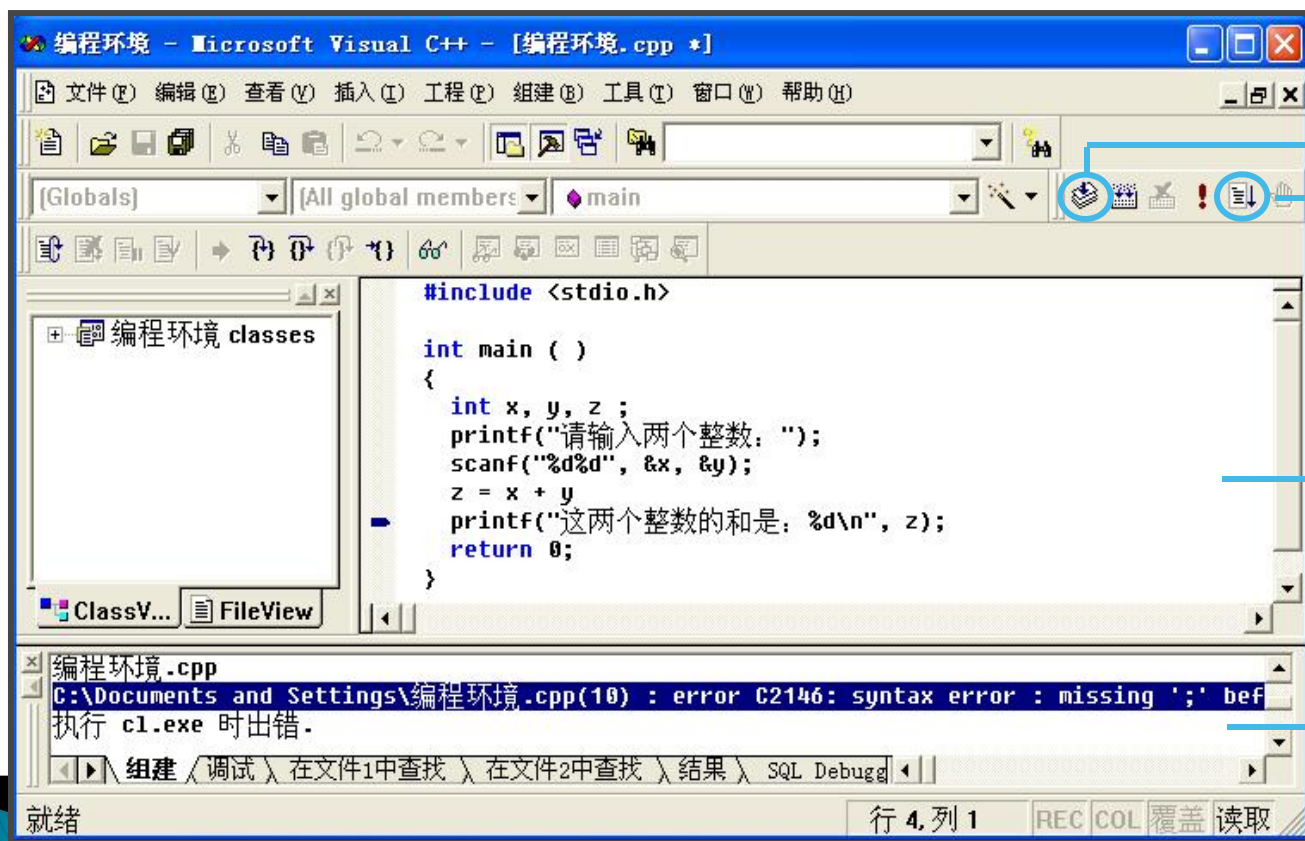
第 5 章 问题求解与程序设计——程序语言

程序设计的环境

- 程序设计的环境是指利用程序设计语言进行程序开发的编程环境。
- 目前的编程环境大都是交互式集成开发环境（Integrated Design Environment, **IDE**），包括程序编辑、程序编译、运行调试等功能。此外，还包括许多编程的实用程序。
- 熟练使用编程工具和环境，也是提高编程效率的因素之一，初学者应该尽快熟悉编程环境。

第5章 问题求解与程序设计——程序语言

程序设计的环境



编译按钮

执行按钮

编辑窗口

信息窗口

第 5 章 问题求解与程序设计——程序语言

程序设计的环境

VS支持创建各种类型的程序，如：ASP.NET、DHTML、JavaScript、Jscript、Visual Basic、Visual C#、Visual C ++、Visual F#，XAML及更多。



著名的Python IDE，还支持JavaScript、Node.js、CoffeeScript、TypeScript、Dart、CSS、HTML。



最初是一个Java IDE。现在扩展到支C/C ++，Java、Perl、PHP、Python、Ruby以及更多的语言。



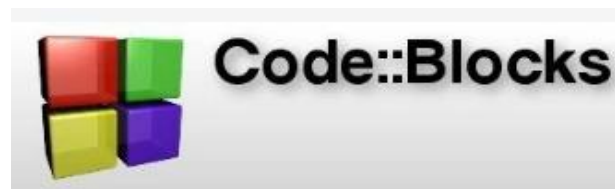
主要用于Java开发，支持Windows，Linux，OS X和Solaris等多种平台。



主要用于JAVA开发，用于Web和安卓移动应用程序开发。



支持三种语言C、C ++和Fortran语言。还支持很多预设和定制插件。



第 5 章 问题求解与程序设计——程序语言

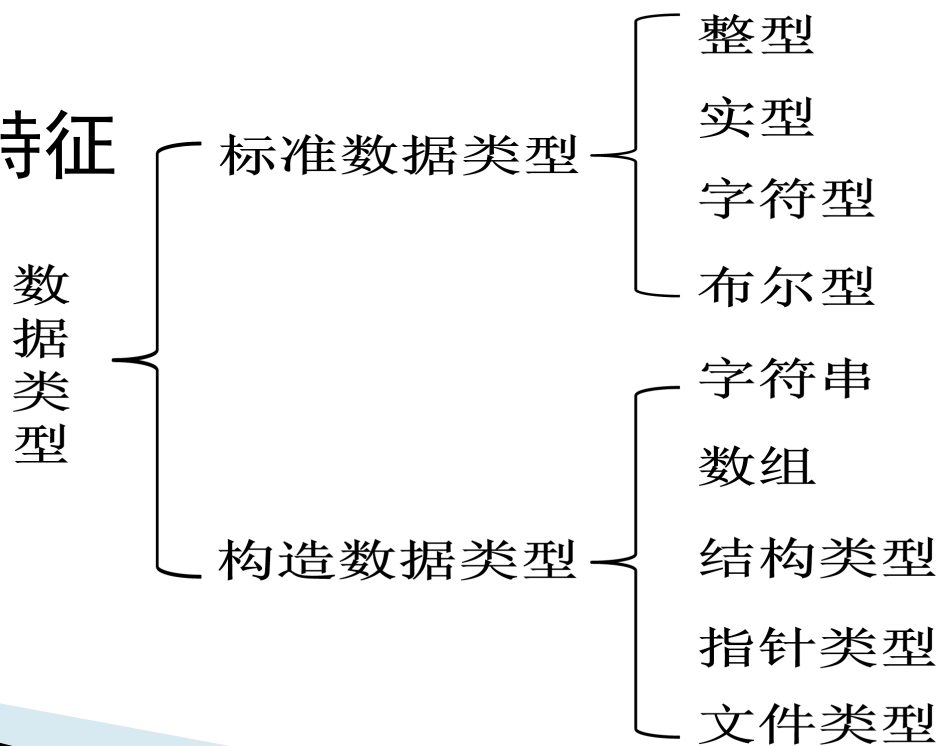
学习通练习5-8



程序设计语言的数据定义

- ▶ **数据类型**：具有相同取值范围的数据和能应用于这组数据的操作。
- ▶ 数据类型定义了数据的**取值范围**、**数据间的相互关系**以及建立在这些数据上的一些**操作**。

▶ 常见的数据类型及特征



程序设计语言的数据定义

熟练掌握取值范围

```
#include <stdio.h>
void main( )
{
    char c1 = 128; //溢出?
    unsigned char c2 = 256; //溢出?
    short s1 = 65535; //溢出?
    unsigned int s2 = 65537; //溢出?
    printf("%d,%d,%d,%d\n", c1, c2, s1, s2);
}
```

输出结果会是什么？

```
-128, 0, -1, 65537
Press any key to continue.
```

char:-128~127

unsigned char:0~255

Short:-32768~32767

Unsigned int:0~ $2^{32}-1$

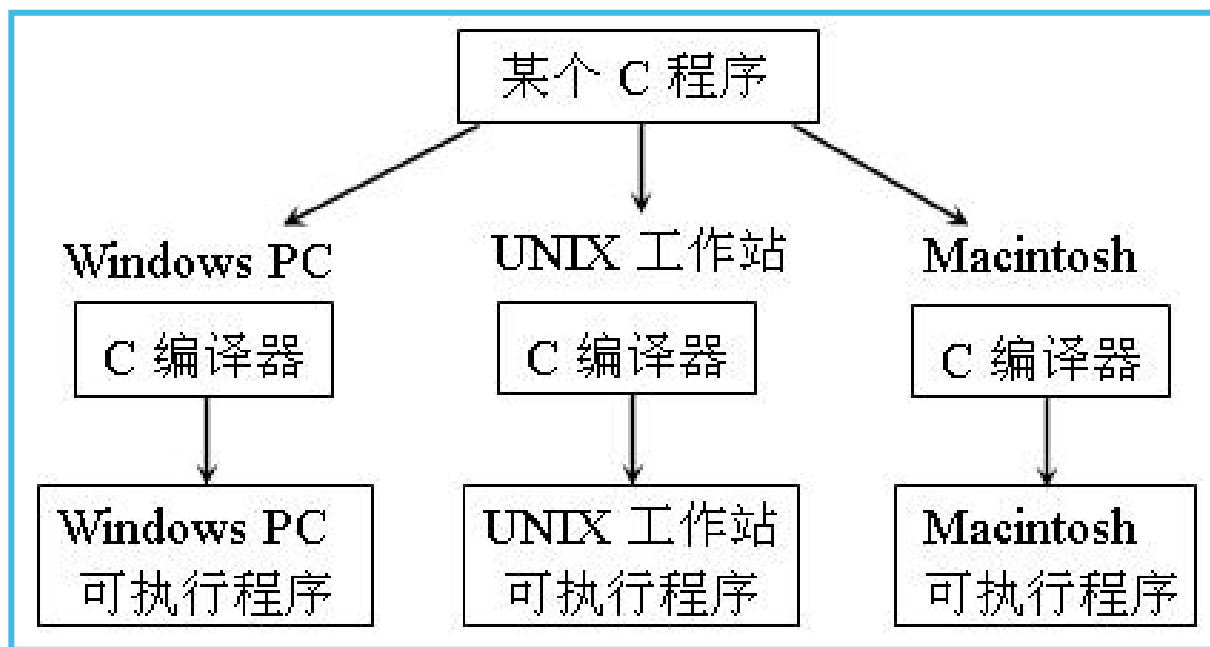
第 5 章 问题求解与程序设计——翻译程序

翻译程序的工作方式

- 利用高级语言编写的程序不能直接在计算机上执行，因为计算机只能执行二进制的机器指令，所以，必须将高级语言编写的程序（称为**源程序**）转换为在逻辑上等价的机器指令（称为**目标程序**），实现这种转换的程序称为**翻译程序**。
- 不同的程序设计语言需要有不同的翻译程序。
- 同一种程序设计语言在不同类型的计算机上也需要配置不同的翻译程序。

第 5 章 问题求解与程序设计——翻译程序

翻译程序的工作方式

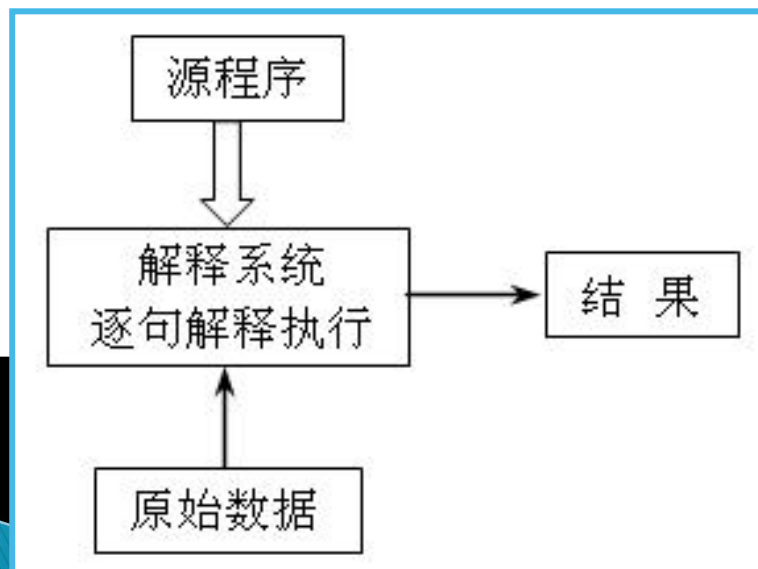


同一种程序设计语言在不同类型的计算机上也需要配置不同的翻译程序。

第5章 问题求解与程序设计——翻译程序

翻译程序的工作方式

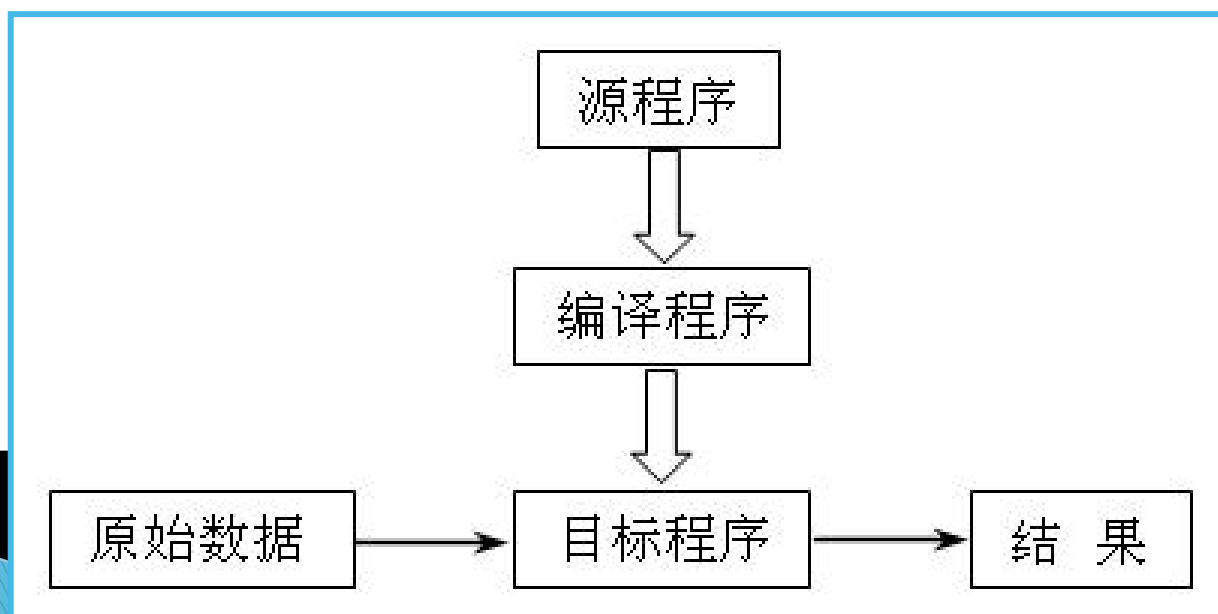
- **解释方式**一般是翻译一句执行一句，在翻译过程中，并不把源程序翻译成一个完整的目标程序，而是按照源程序中语句的顺序逐条语句翻译成机器可执行的指令并立即予以执行。
- 由于解释方式不产生目标代码，所以，**源程序的执行不能脱离其解释环境**，并且每次运行都需要重新解释。
- BASIC语言和JAVA语言都具有逐条解释执行程序的功能。



第5章 问题求解与程序设计——翻译程序

翻译程序的工作方式

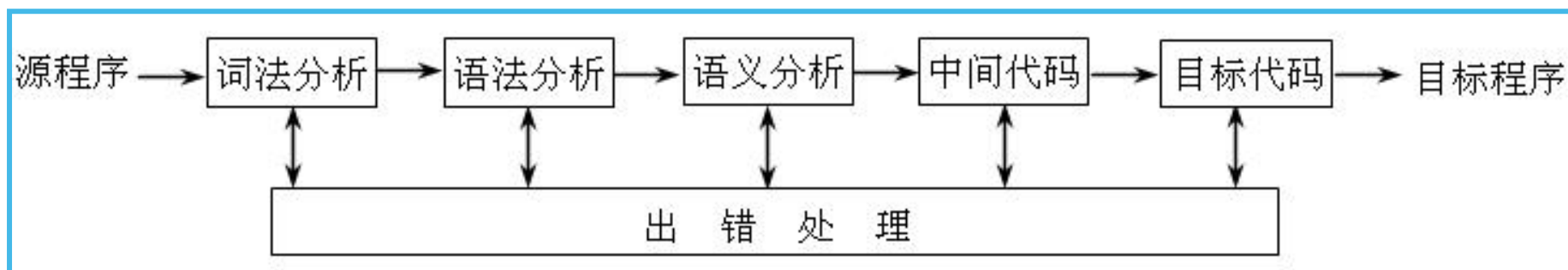
- **编译方式**是一个整体理解和翻译的过程，先由编译程序把源程序翻译成目标程序，然后再由计算机执行目标程序。
- 由于编译后形成了可执行的目标代码，所以，**目标程序可以脱离其语言环境独立执行**。C、C++语言都是编译型语言。



第 5 章 问题求解与程序设计——翻译程序

编译程序的基本过程

- 编译程序是把源程序翻译成目标程序，因此，编译程序需要根据源语言的具体特点和对目标程序的具体要求来设计。
- 如同自然语言的翻译，编译程序的翻译规则是源语言的语法规则和语义规则。



第5章 问题求解与程序设计——翻译程序

编译程序的基本过程

词法分析的任务是对源程序进行扫描和分解，按照词法规则**识别**出一个个的单词，如关键字、标识符、运算符等，并将单词**转化**为某种机内表示。

```
double  area  =  10  ;
```

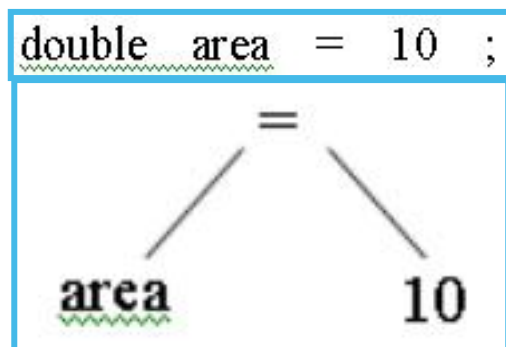
① ② ③ ④ ⑤

其中，单词①是关键字，单词②是标识符，单词③是运算符，单词④是常量，单词⑤是分隔符。

第 5 章 问题求解与程序设计——翻译程序

编译程序的基本过程

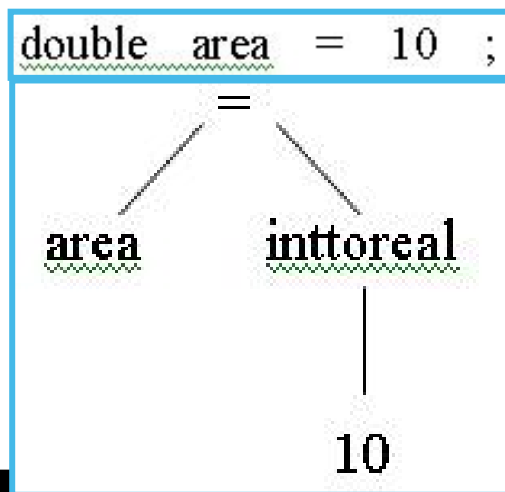
语法分析是编译程序的核心部分，它的任务是对词法分析得到的单词序列按照语法规则分析出一个个的语法单位，如表达式、语句等。



第5章 问题求解与程序设计——翻译程序

编译程序的基本过程

- **语义分析**的任务是检查程序中语义的正确性，以保证单词或语法单位能有意义地结合在一起。
- 语义分析的一个重要任务是**类型检查**，即对每个运算符的运算对象，检查它们的类型是否合法。



为什么计算机不能直接将一个整数和一个小数相加？

第5章 问题求解与程序设计——翻译程序

编译程序的基本过程

- 为了降低编译的难度，一般先将源程序转换为某种中间形式，然后再转换为目标代码。
- **中间代码**是复杂性介于源语言和机器语言之间的一种表现形式，常用的中间代码有三元式（运算符，运算对象1，运算对象2）、四元式（运算符，运算对象1，运算对象2，结果）、逆波兰式（运算对象1，运算对象2，运算符）等。
- Java程序的编译过程只到中间代码阶段，然后再由安装在计算机上的Java虚拟机把中间代码形式的Java程序转换为特定机器上的目标程序，以实现Java程序的跨平台特性。

第 5 章 问题求解与程序设计——翻译程序

编译程序的基本过程

- **目标代码**的任务是将中间代码转换为特定机器的目标程序，这涉及到计算机硬件系统功能部件的使用、机器指令的使用、存储空间的分配以及寄存器的调度等。
- 显然，高级语言和计算机的多样性为目标代码生成的理论研究和实现技术带来很大的复杂性。

源文件.cpp



编译



目标文件.obj

```
010100001010010  
101010010100100  
000010001000010  
01111010101001  
0011010.....
```

第 5章 问题求解与程序设计——回答问题

学完本章，你将如何回答下列问题：

1. 什么是程序？什么是程序设计？什么是程序设计语言？
 2. 程序是怎么设计出来的？程序设计的关键是什么？
 3. 计算机运行程序的过程就是对数据的加工处理过程，如何将数据存储到计算机的内存中？如何描述问题的处理方法和具体步骤才能让计算机“看懂”呢？
 4. 为了方便编写程序，现代程序设计普遍采用高级程序设计语言，高级语言程序如何转换为等价的机器指令？
- 