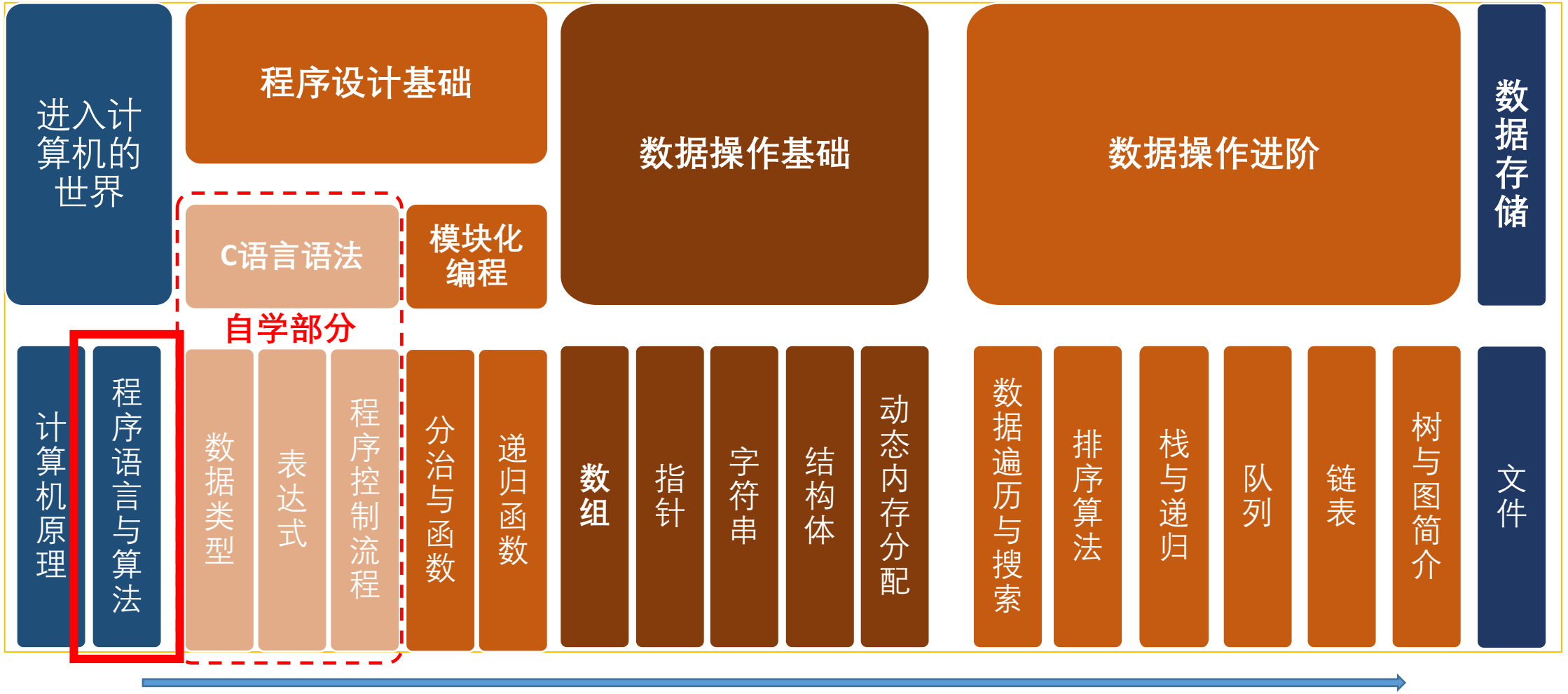
计算思维与实践

哈尔滨工业大学（深圳）
计算机科学与技术学院
大数据技术中心
张保权

课程内容安排



程序设计思想与数据操作方法融会贯通，内容由浅入深

第二讲 程序语言与算法-学习内容

- 2.1 程序语言概述
- 2.2 程序设计过程与开发环境
- 2.3 算法概念、设计与分析概述

2.1 程序语言概述

2.1.1 由机器语言到高级语言

2.1.2 计算机语言的发展

2.1.3 编写一个程序

2.1 程序语言概述

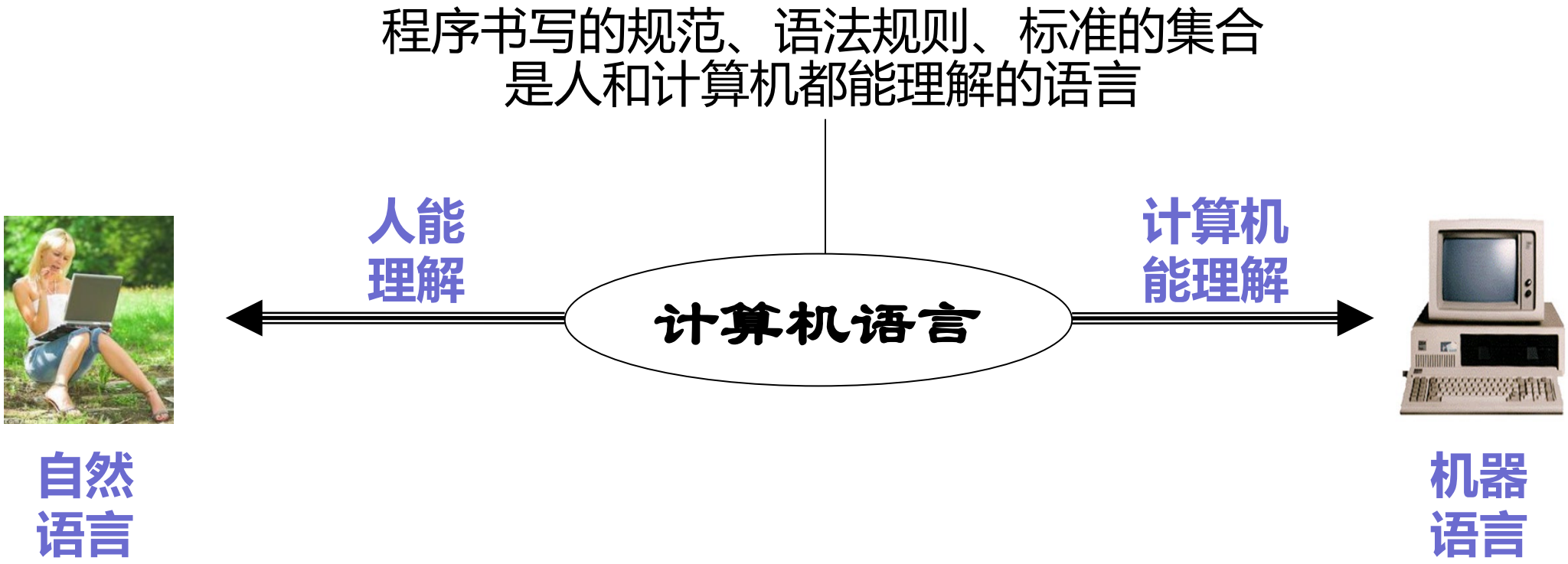
2.1.1 由机器语言到高级语言

2.1.2 计算机语言的发展

2.1.3 编写一个程序

2.1.1 由机器语言到高级语言-什么是计算机语言?

- **计算机语言**是人与计算机之间通讯的语言，人与计算机之间传递信息的媒介，人和计算机都能理解的语言，其由程序书写的**规范、语法规则、标准**的集合组成。



2.1.1 由机器语言到高级语言-机器语言回顾?

- **机器指令**：CPU可以直接分析并执行的指令，一般由0和1的编码表示。通常由操作码和操作数两部分组成，操作码指出该指令所要完成的操作，操作数指出参与运算的对象，以及运算结果所存放的位置等；
- **机器语言**：机器能够执行的所有指令的集合。
- 如果将机器语言作为计算机语言（即人去适配计算机），将会？

100001 10
00000111

100010 10
00001010

人要去学习

计算机语言
(机器语言)

直接可理解

注意：计算机可以直接运行程序只能是机器语言所编写的程序；其他语言编写的程序都需要转换成机器程序，计算机才能执行

2.1.1 由机器语言到高级语言-汇编语言

- 如何解决机器语言编写程序所存在的困难？即汇编语言
- 汇编语言：用助记符号书写程序的规范、语法规则、标准的集合
  - 用符号编写程序 ==》翻译 ==》机器语言程序;
  - 人们提供了用助记符编写程序的规范/标准，并将其视作为计算机语言；
  - 开发一个翻译程序（被称为汇编程序），将符号程序自动转换成机器语言程序。

操作码　　地址码

100001　1000000111

MOV　R0, 7

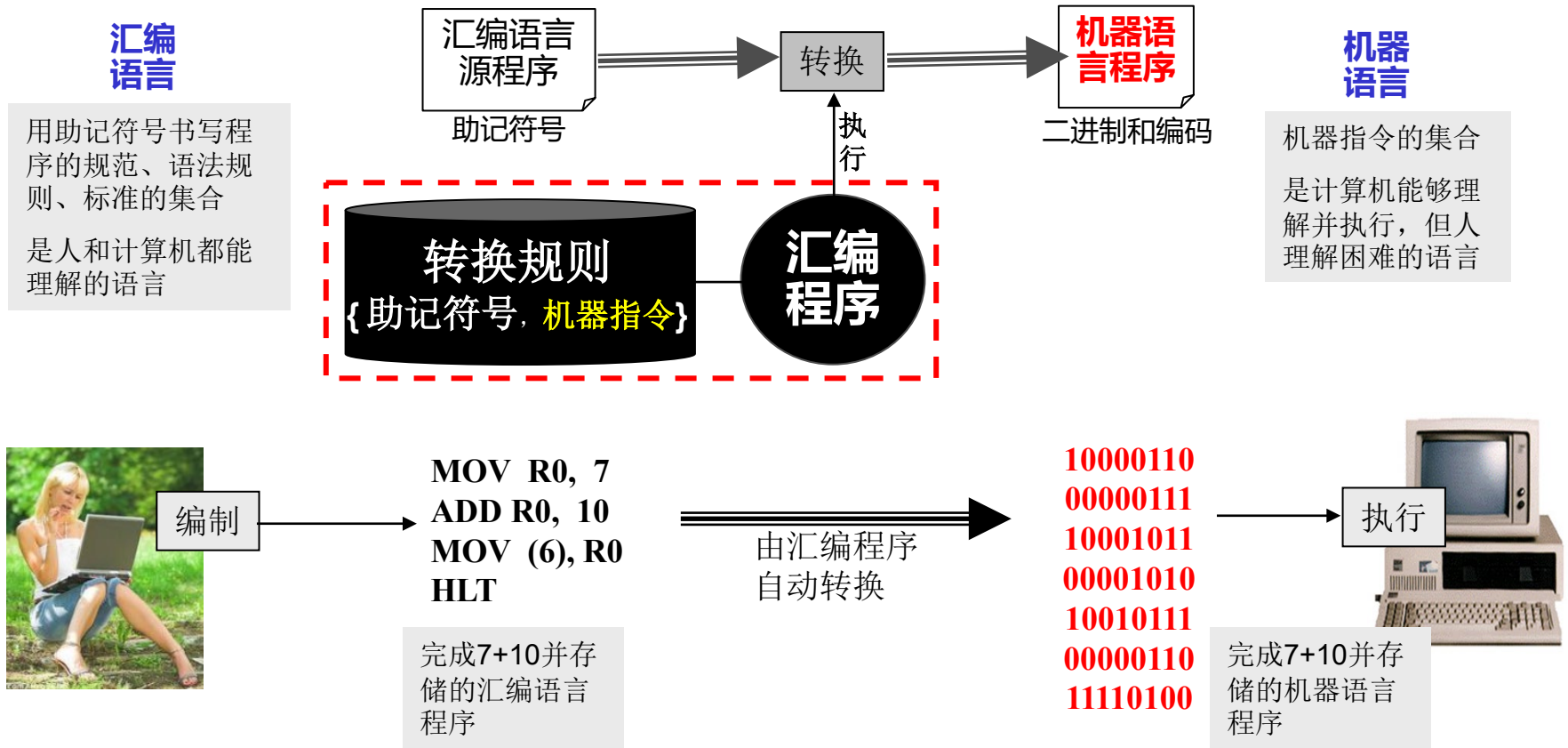
计算7+10并存储的汇编语言源程序

```
MOV  R0,  7
ADD  R0,  10
MOV  (6),  R0
HLT
```

用助记符号编写程序的语言，可与机器语言一一对应

2.1.1 由机器语言到高级语言-汇编语言

- **机器不能直接执行符号化程序（即汇编语言程序），怎么办？**
  - 汇编/翻译：利用汇编程序将汇编语言程序**汇编/翻译**成机器语言程序的过程
- **汇编程序**：将汇编语言程序翻译成机器语言程序的程序



2.1.1 由机器语言到高级语言-高级语言

- 能否利用类似自然语言作为编写程序的语言（即计算机语言）？
- **高级语言：**用类似自然语言书写程序的规范、语法规则、标准的集合
  - 人们提供了用类似自然语言方式、以语言为单位编写程序的规范/标准；
  - 开发一个翻译程序（被称为编译程序），其实现了将高级语言程序自动翻译成计算机可直接理解的机器语言程序功能。
- 编译程序：将高级语言源程序翻译成机器语言程序的程序

计算7+10并存储的汇编语言源程序

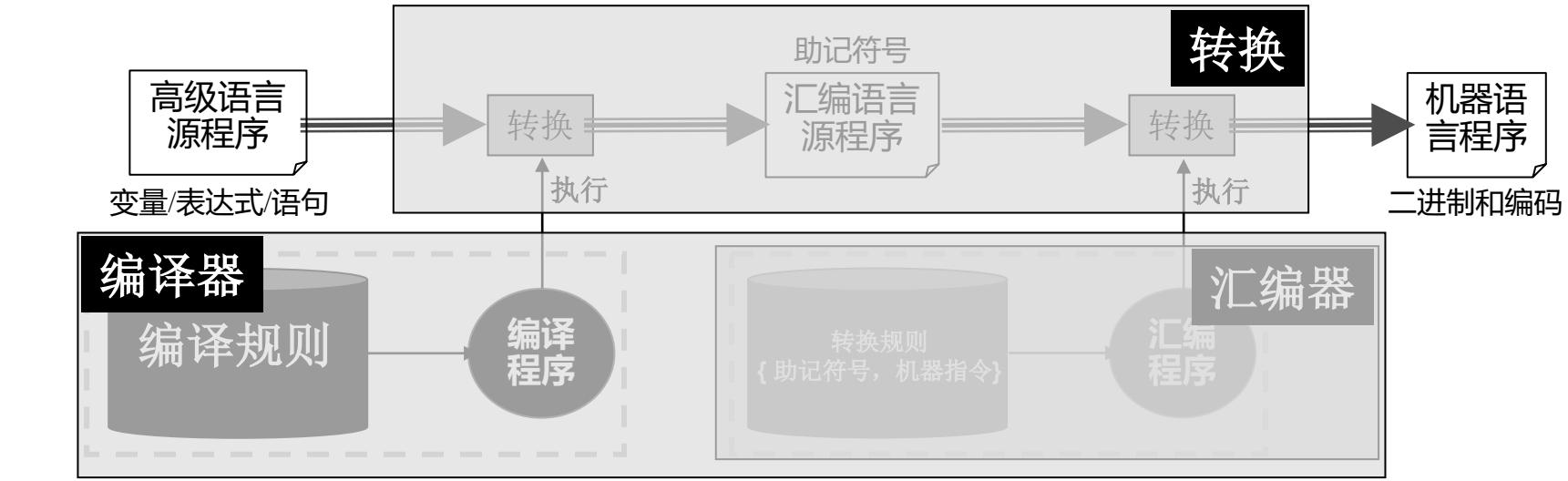
```
MOV  R0,  7
ADD  R0,  10
MOV  (6),  R0
HLT
```

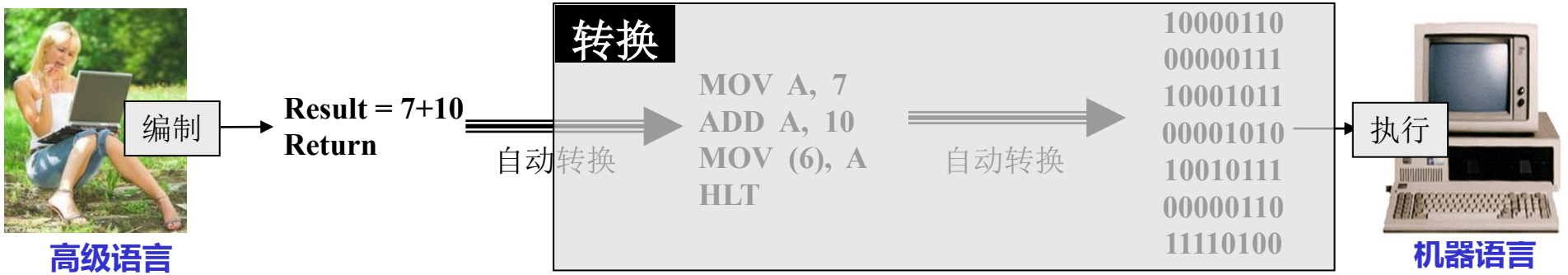
计算7+10并存储的高级语言(源)程序

```
Result = 7+10;
Return
```

2.1.1 由机器语言到高级语言-汇编语言VS高级语言

- 高级语言：机器无关性；一条高级语言语句往往可由若干条机器语言语句实现且不具有对应性；
- 汇编语言：机器相关性；汇编语言语句和机器语言语句有对应性。





2.1 程序语言概述

2.1.1 由机器语言到高级语言

2.1.2 计算机语言的发展

2.1.3 编写一个程序

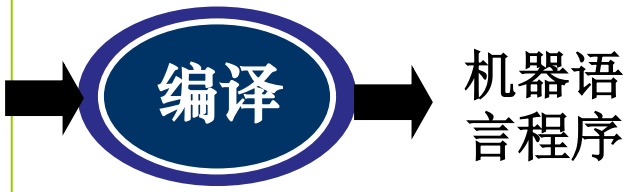
2.1.2 计算机语言的发展-高级语言（源）程序

符号化

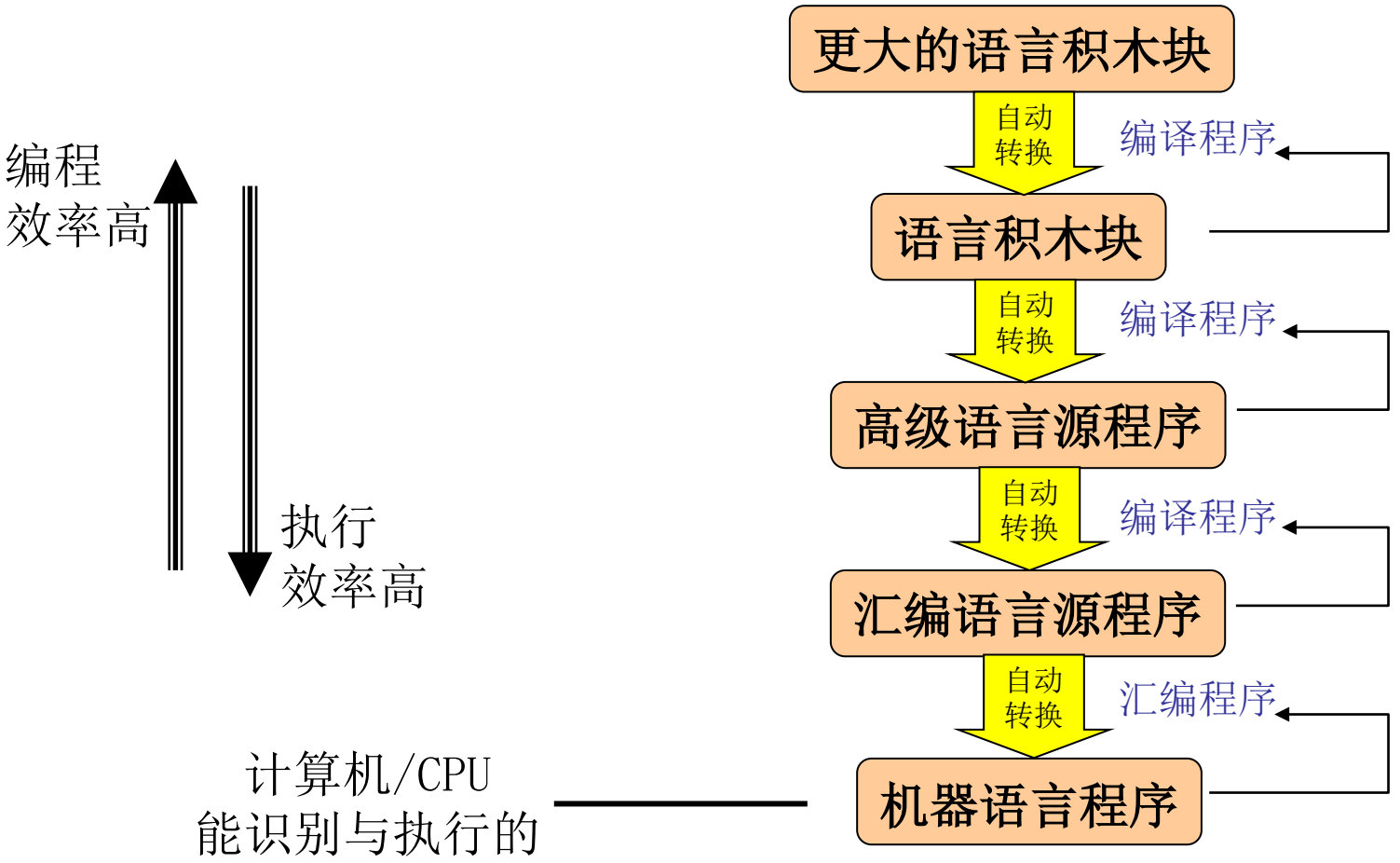
语句化

结构化

```
K = 0;
For  I =1 to 100 Step 1
{    If  I<=50 && I > 30
     { K =  K+I; }
}
```

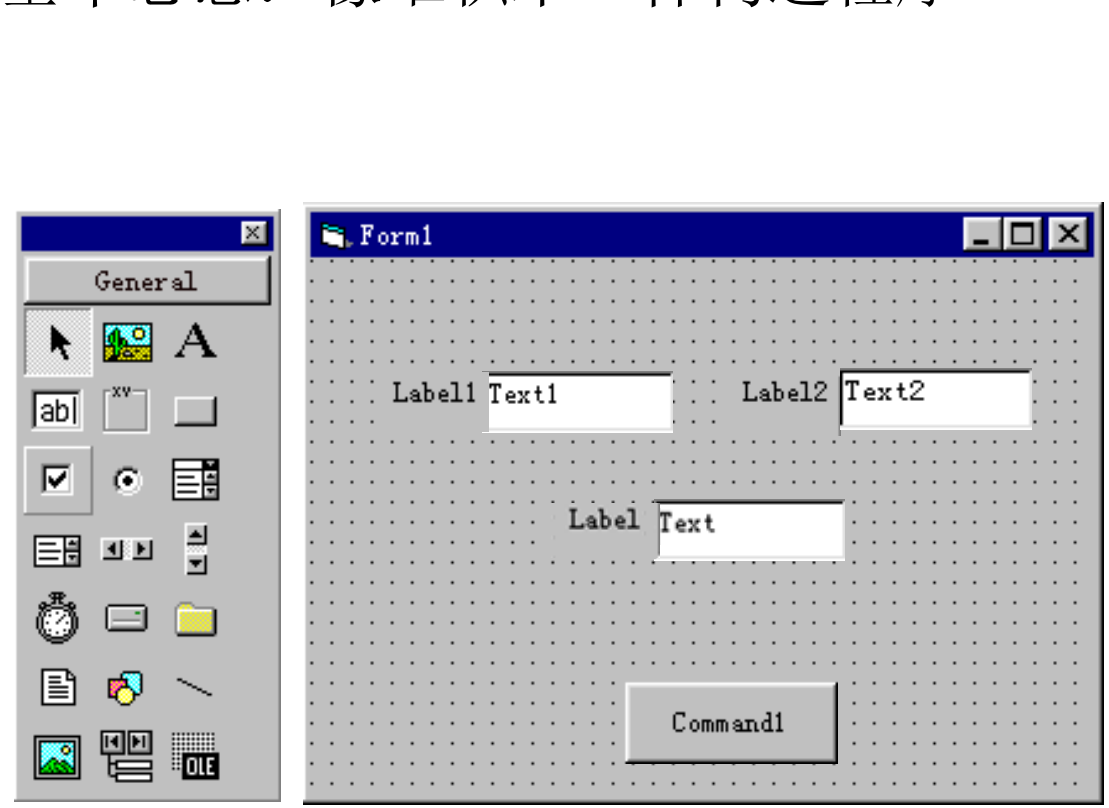


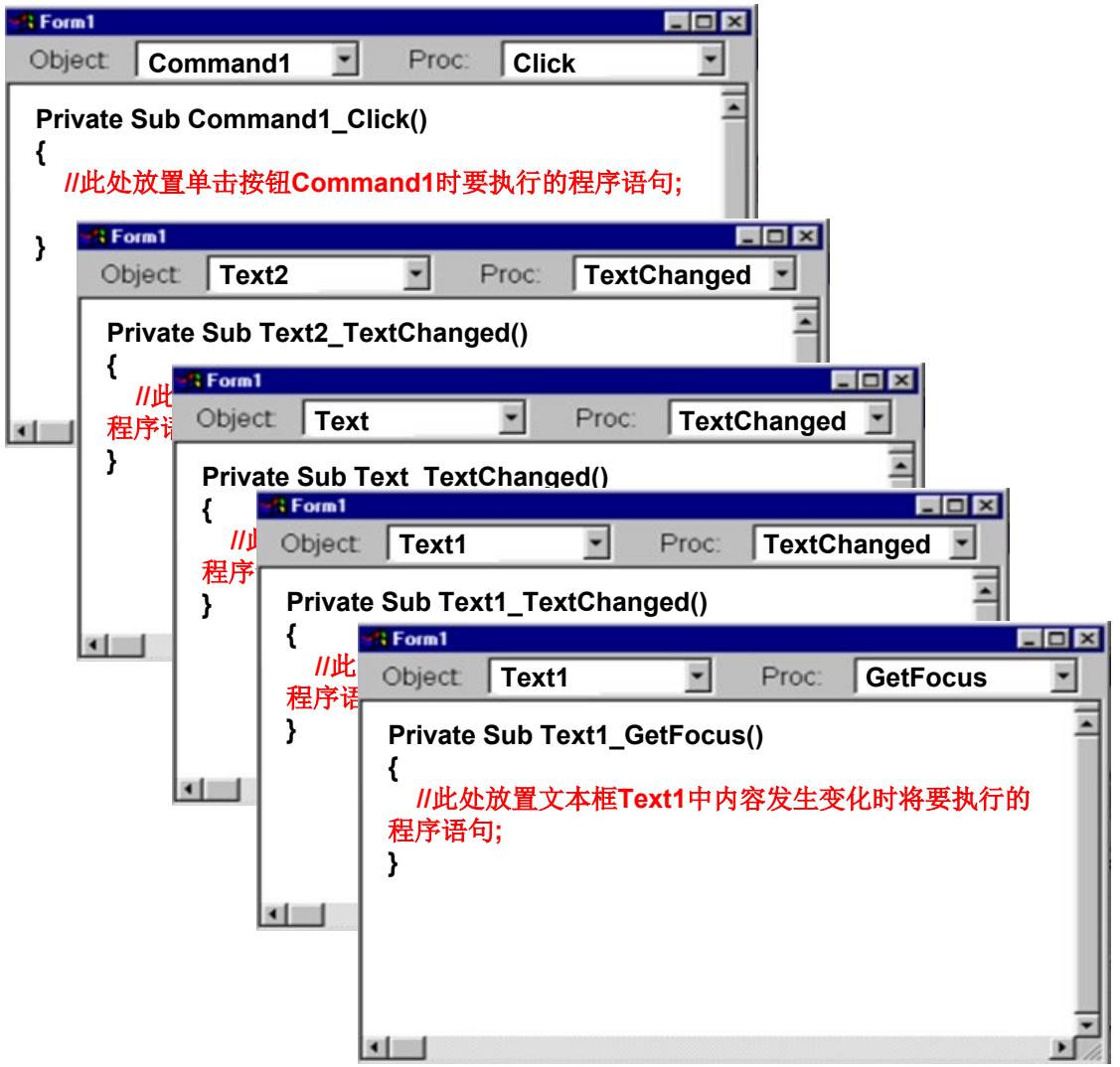
2.1.2 计算机语言的发展-计算机语言发展的基本思维



2.1.2 计算机语言的发展-可视化构造语言

- 基本思想：像堆积木一样构造程序





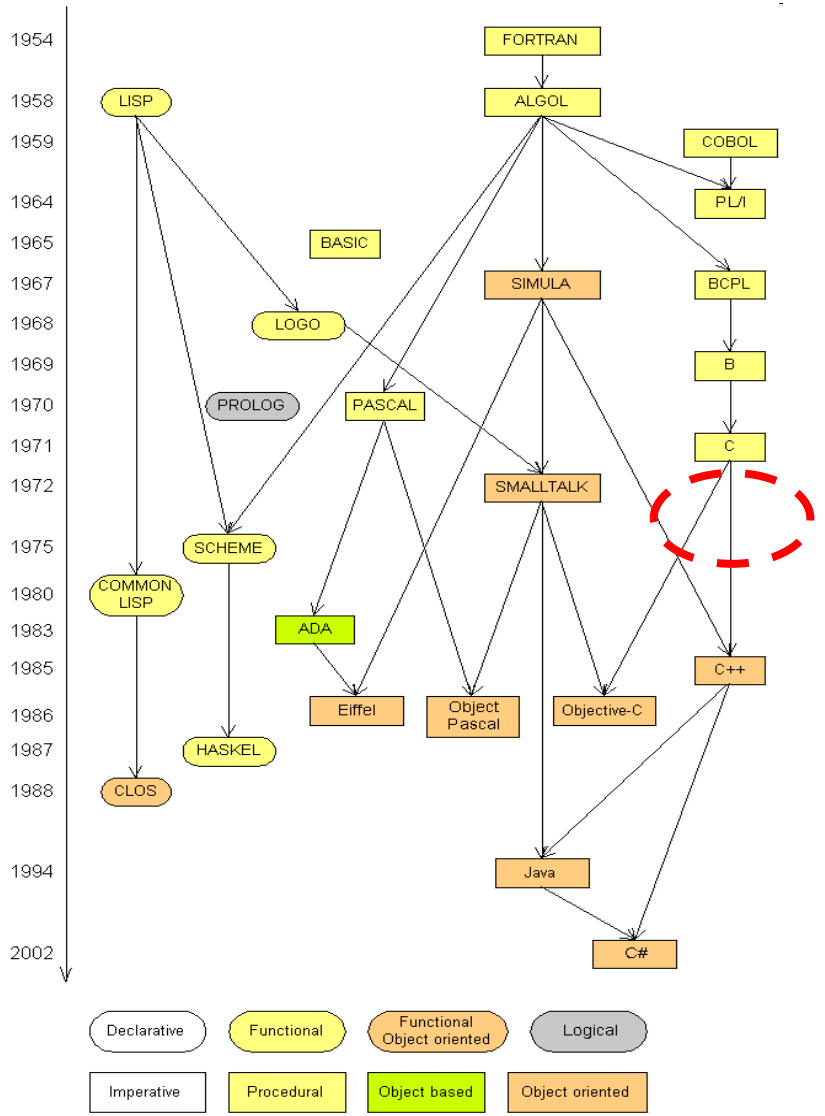
2.1.2 计算机语言的发展-计算机语言的发展历程

- 1954年约翰.巴克斯发明第一个高级语言FORTRAN；

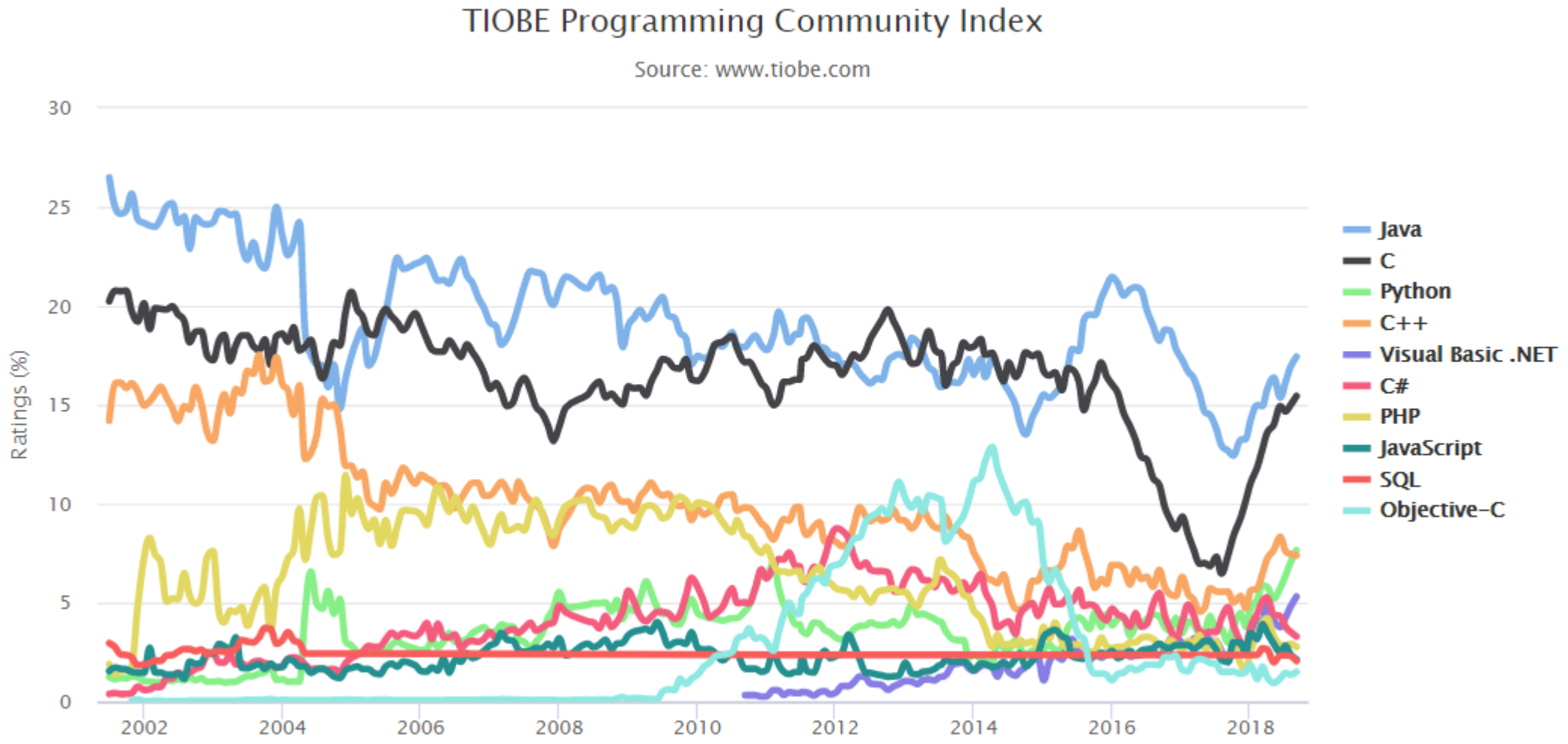
- 百家争鸣，据不完全统计2500种

http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm可看到其列表和简介

- **影响最大、寿命最长的非C语言莫属**



2.1.2 计算机语言的发展-编程语言受欢迎程度排行



http://www.tiobe.com/tpci.htm

2.1.2 计算机语言的发展-为什么要学C语言?

- 是一种高级语言
  - 并不“高级”，只是相对低级语言在一个高的级别上进行编程
  - 实际上是一种介于高级语言和低级语言之间的语言，运行效率高
  - 透过现象看本质，透过C语言窥探计算机底层的工作原理
- 很多流行语言、新生语言都借鉴了它的思想、语法
  - 从C++，到Java，再到C#
  - 学好C是学习这些流行语言的基础，了解程序设计的基本思想
  - 交流、笔试、面试时最常见的语言
- 历史悠久，战勋卓著
  - 诞生于20世纪70年代初、成熟于80年代
  - 很多重量级软件都是用C语言写的，维护已有的C代码
- 上天入地，无所不能
  - 几乎没有不能用C语言实现的软件
  - 没有不支持C语言的系统

2.1.2 计算机语言的发展-如何学好C语言?

万事开头难

百折不挠

师傅领进门，出徒在个人

世界上没有从不练琴的钢琴家

练习、练习、再练习

持之以恒

2.1 程序语言概述

2.1.1 由机器语言到高级语言

2.1.2 计算机语言的发展

2.1.3 编写一个程序

2.1.3 编写一个程序

程序编写示例：不同计算机语言的程序表达

比较一下有什么差异?

```c
//C 语言的冒泡排序程序
void bubble_sort(int *lists, int count)
{    int i, j;
     for(i=0; i<count-1; i++)
   {   for(j=0; j<count-i; j++)
     {   if   (lists[j] < lists[j+1])
             {     int k = lists[j];
                   lists[j] = lists[j+1];
                   lists[j+1] = k;        }
     }
   }
}
```

```
'Visual Basic 语言的冒泡排序程序
Function bubble_sort(lists(1 to 100) as integer, count as integer) As integer
     Dim i as integer, j as integer, k as integer
     for(i=1 to count-1 Step1)
          for(j=1 to count-i Step 1)
               if   (lists(j) < lists(j+1))
                    k = lists(j);
                    lists(j) = lists(j+1);
                    lists(j+1) = k;
               end if
          next j
     next i
End Function
```

```python
# Python 语言的冒泡排序程序
def   bubble_sort(lists,count):
   for i in range(0, count):
      for j in range(0, count-i):
         if lists[j] < lists[j+1]:
            k=lists[j];
            lists[j]=lists[j+1];
            lists[j+1]=k;
   return lists
```

2.2 程序设计过程与开发环境

2.2.1 程序设计过程

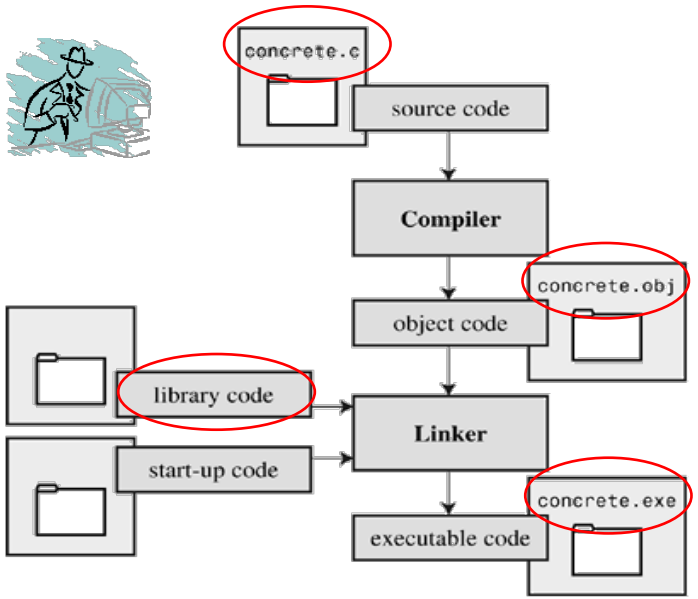
2.2.2 程序开发环境

2.2 程序设计过程与开发环境

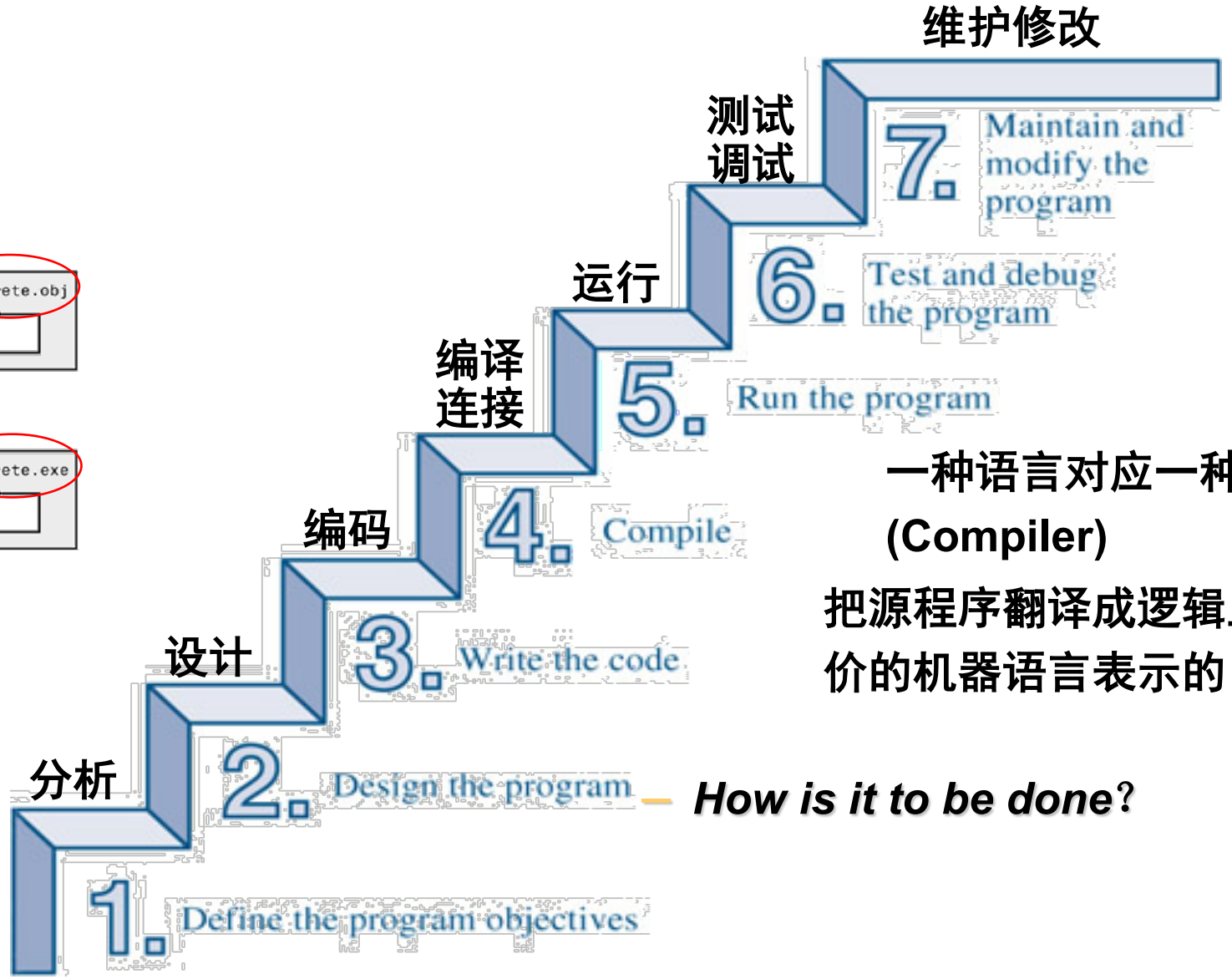
2.2.1 程序设计过程

2.2.2 程序开发环境

2.2.1 程序开发过程



把程序调用的库函数链接到目标程序中，生成可被OS执行的程序



一种语言对应一种编译器(Compiler)

把源程序翻译成逻辑上与之等价的机器语言表示的目标程序

2.2 程序设计过程与开发环境

2.2.1 程序设计过程

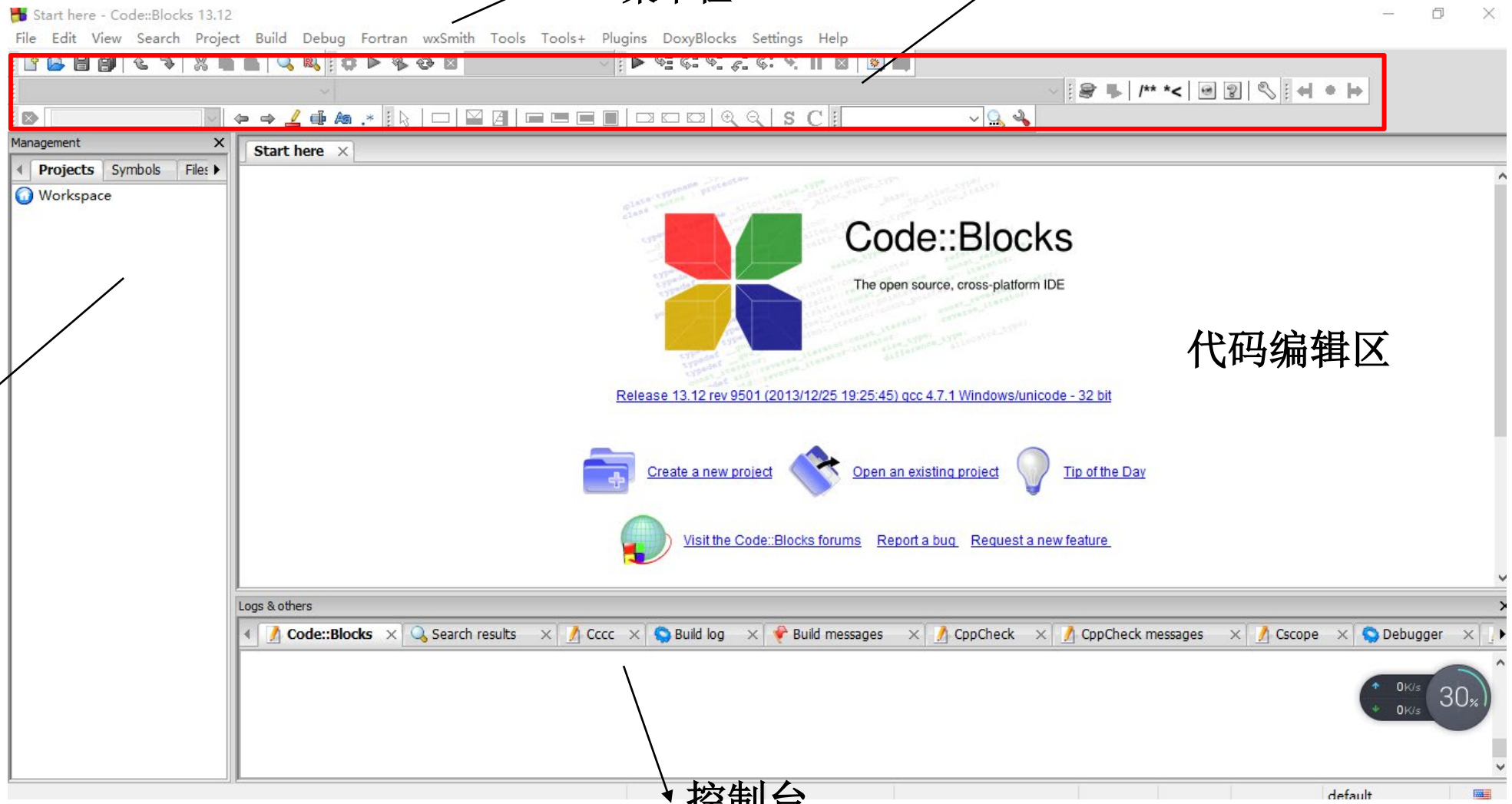
2.2.2 程序开发环境

2.2.1 程序开发环境

- IDE，全称为“集成开发环境”（Integrated Development Environment），是一种用于软件开发的应用程序，提供了一系列工具和功能来帮助开发者编写、测试和调试代码。
- 常见IDE：
  - Eclipse；
  - DevC++；
  - CodeBlocks
- 本课程以CodeBlocks开发环境为主

2.2.1 程序开发环境-CodeBlocks概述

点击桌面图标



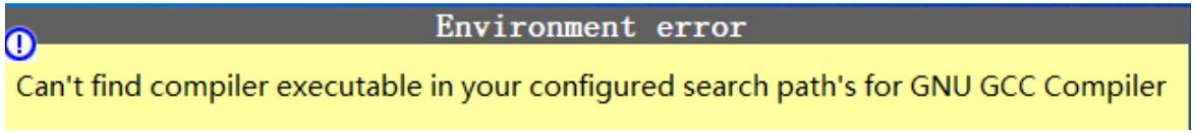
项目目录结构

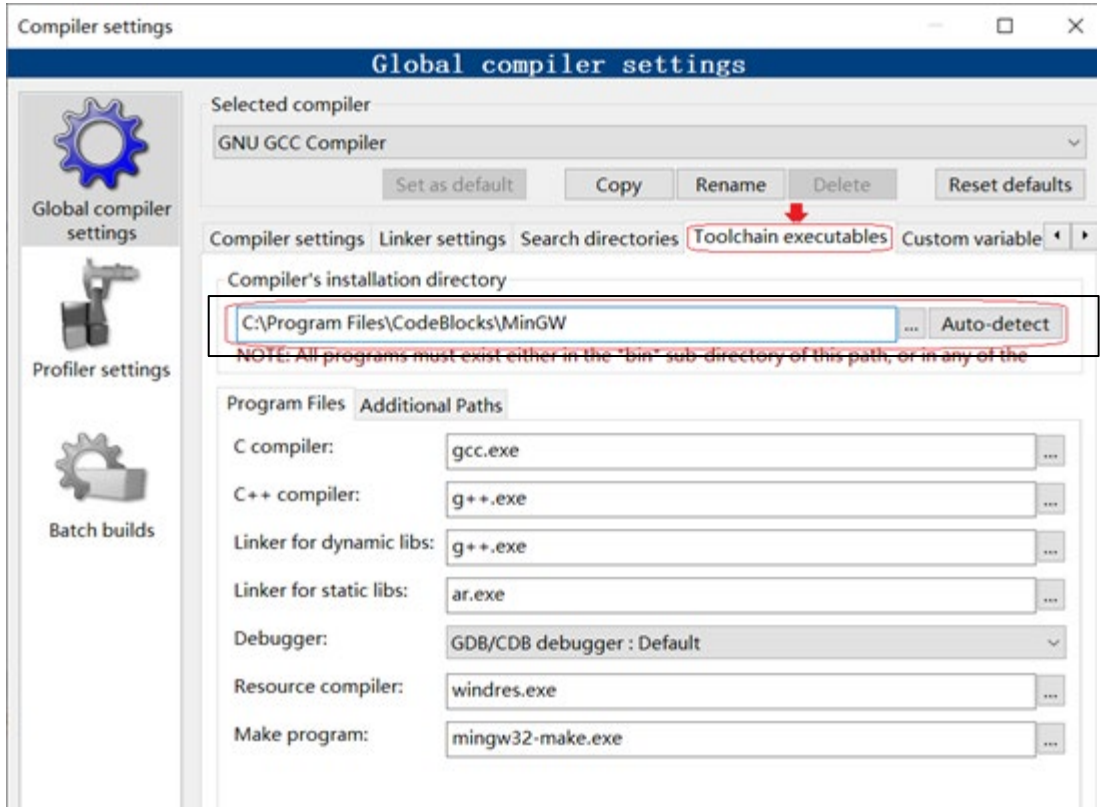
代码编辑区

控制台

2.2.1 程序开发环境-CodeBlocks概述

启动后若提示：

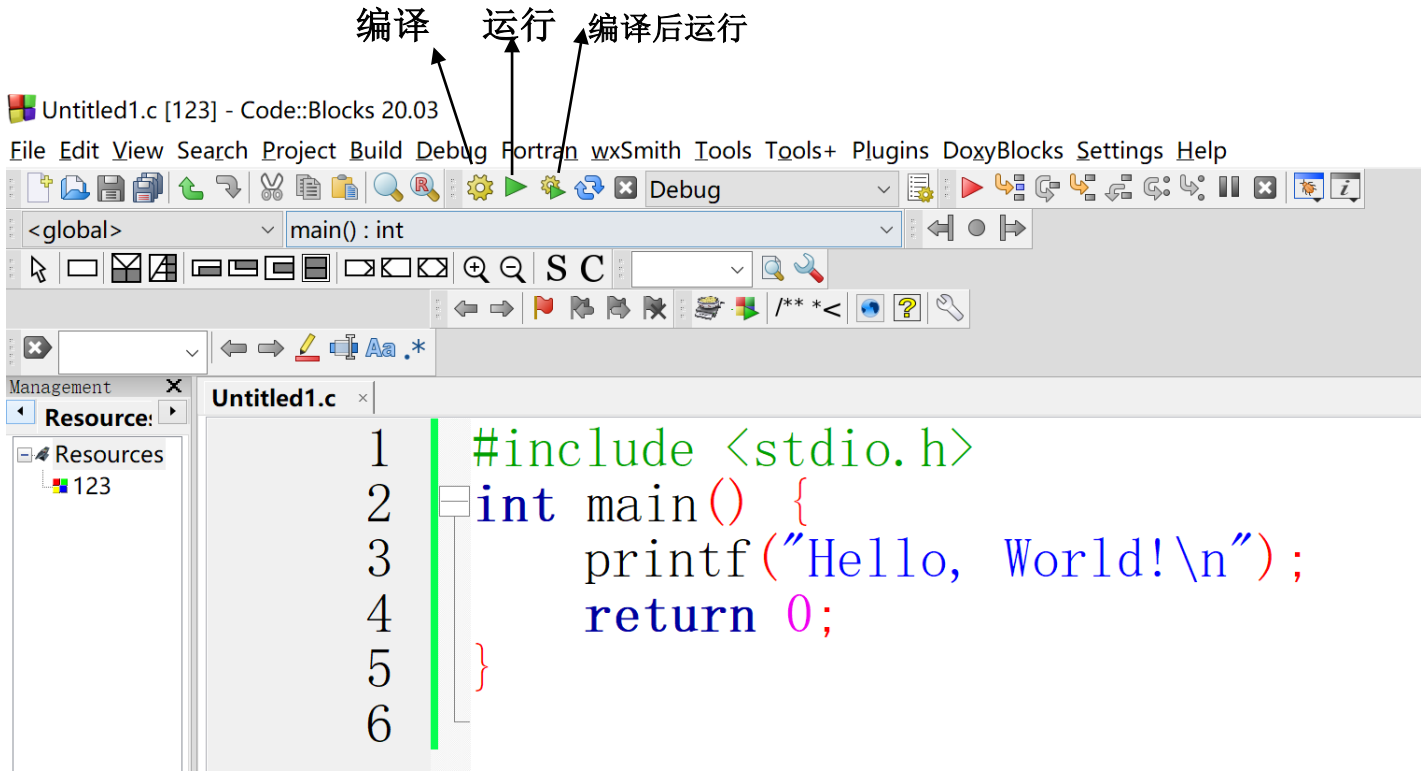




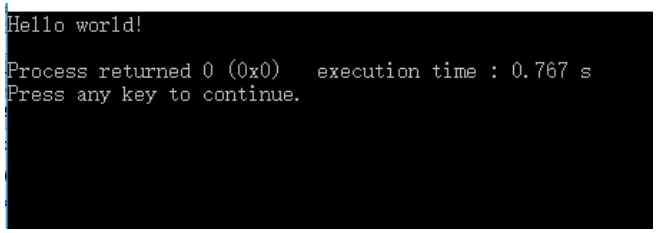
点击菜单栏setting->Compile,**指定编译器MinGW的位置**

2.2.1 程序开发环境-CodeBlocks概述

- 编译与运行
  - **编译**：将源码编译成可执行程序。点击工具栏齿轮按钮或点击Build菜单栏中的“Build”
  - **运行**：执行编译后的可执行文件。点击工具栏三角形按钮或点击Build菜单栏中的“Run”



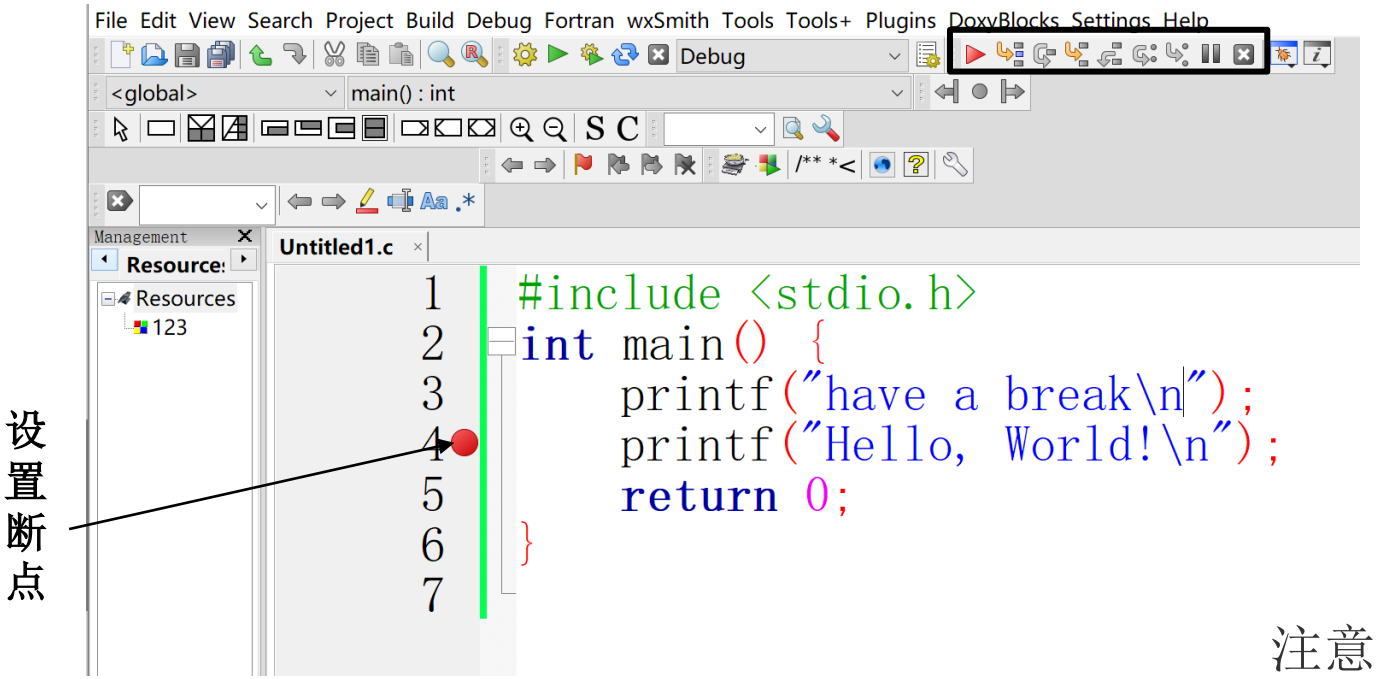
运行结果

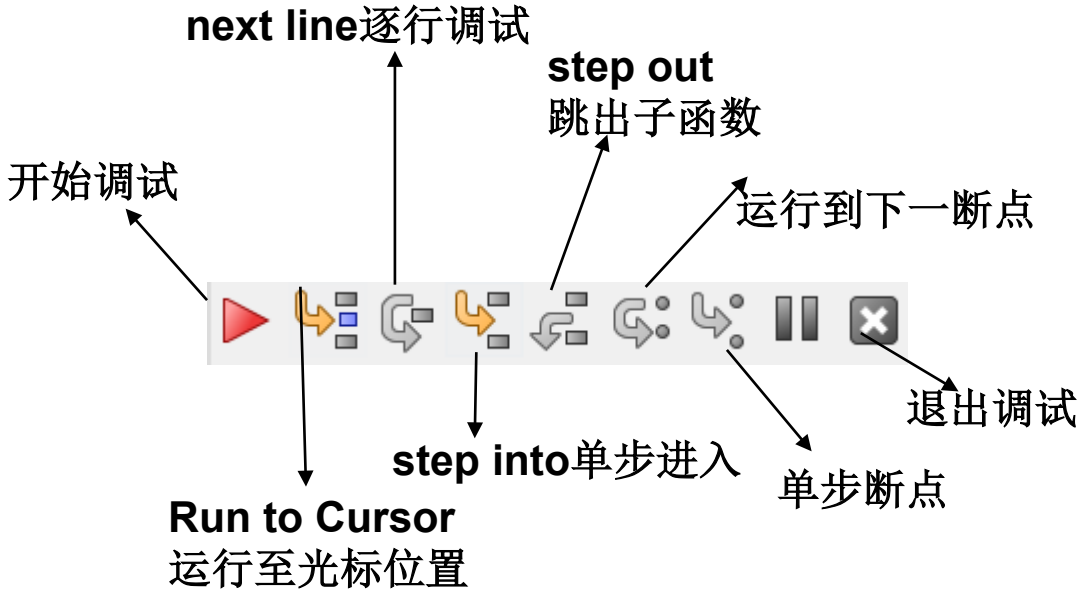


2.2.1 程序开发环境-CodeBlocks概述

- 调试程序

- **调试**：跟踪程序执行过程，调试问题代码





注意CodeBlocks调试器需要一个完整的项目才可以启动，单独的文件无法使用调试器。

2.3 算法概念、设计与分析概述

2.3.1 算法基本概念

2.3.2 算法设计过程概述

2.3.3 算法分析方法概述

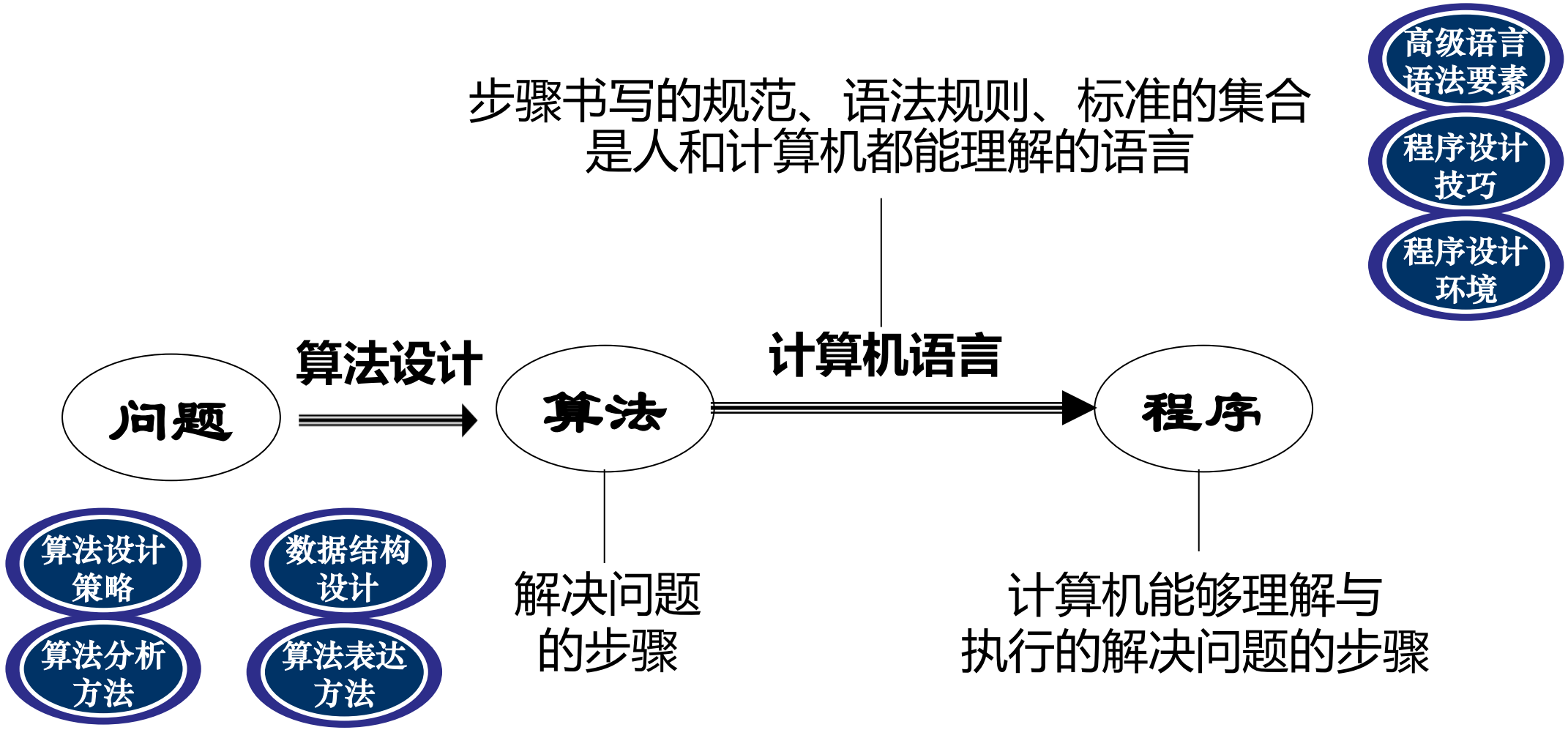
2.3 算法概念、设计与分析概述

2.3.1 算法基本概念

2.3.2 算法设计过程概述

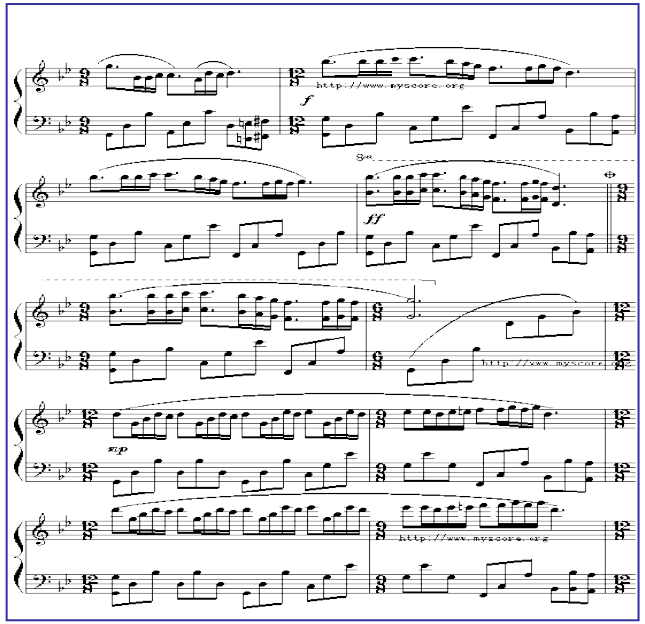
2.3.3 算法分析方法概述

2.3.1 算法基本概念-计算机如何求解一个实际问题?



2.3.1 算法基本概念-什么是算法?

- “算法”(Algorithm)一词源于阿拉伯数学家阿科瓦里茨米，其在公元825年写了著名的《波斯教科书》，书中概括了进行四则算术运算的计算规则。
- **算法**是一个有穷规则的集合，它用规则规定了解决某一特定类型问题的运算序列，或者规定了任务执行或问题求解的一系列步骤。



如音乐乐谱、太极拳谱等都可看作广义的算法

2.3.1 算法基本概念-什么是算法?

算法示例

寻找两个正整数的最大公约数的欧几里德算法

输入：正整数*M*和正整数*N*

输出：*M*和*N*的最大公约数(设M>N)

算法步骤：

Step 1. *M*除以*N*,记余数为*R*

Step 2. 如果*R*不是*0*，将*N*的值赋给*M*, *R*的值赋给*N*, 返回Step 1; 否则, 最大公约数是*N*, 输出*N*, 算法结束。

	M	N	R	最大公约数
具体问题	32	24		?
算法计算过程				
1	32	24	8	
2	24	8	0	8
具体问题	31	11		?
算法计算过程				
1	31	11	9	
2	11	9	2	
3	9	2	1	
4	2	1	0	1

2.3.1 算法基本概念-算法的五个特征

- **有穷性**：一个算法在执行有穷步规则之后必须结束。
- **确定性**：算法的每一个步骤必须要确切地定义，不得有歧义性。
- **输入**：算法有零个或多个的输入。
- **输出**：算法有一个或多个的输出/结果，即与输入有某个特定关系的量。
- **能行性**：算法中有待执行的运算和操作必须是相当基本的**(可以由机器自动完成)**，并能在有限时间内完成。

寻找两个正整数的最大公约数的欧几里德算法

输入：正整数*M*和正整数*N*

输出：*M*和*N*的最大公约数(设M>N)

算法步骤：

Step 1. *M*除以*N*,记余数为*R*

Step 2. 如果*R*不是*0*，将*N*的值赋给*M*, *R*的值赋给*N*, 返回Step 1; 否则, 最大公约数是*N*, 输出*N*, 算法结束。

基本运算：除法、赋值、逻辑判断

典型的“重复/循环”与“迭代”

2.3 算法概念、设计与分析概述

2.3.1 算法基本概念

2.3.2 算法设计过程概述

2.3.3 算法分析方法概述

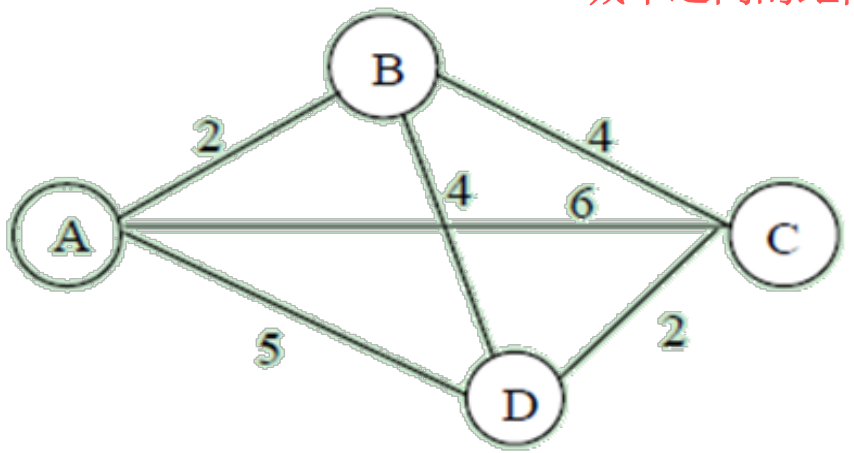
2.3.2 算法设计过程概述-以一个TSP问题为例介绍

什么是TSP问题

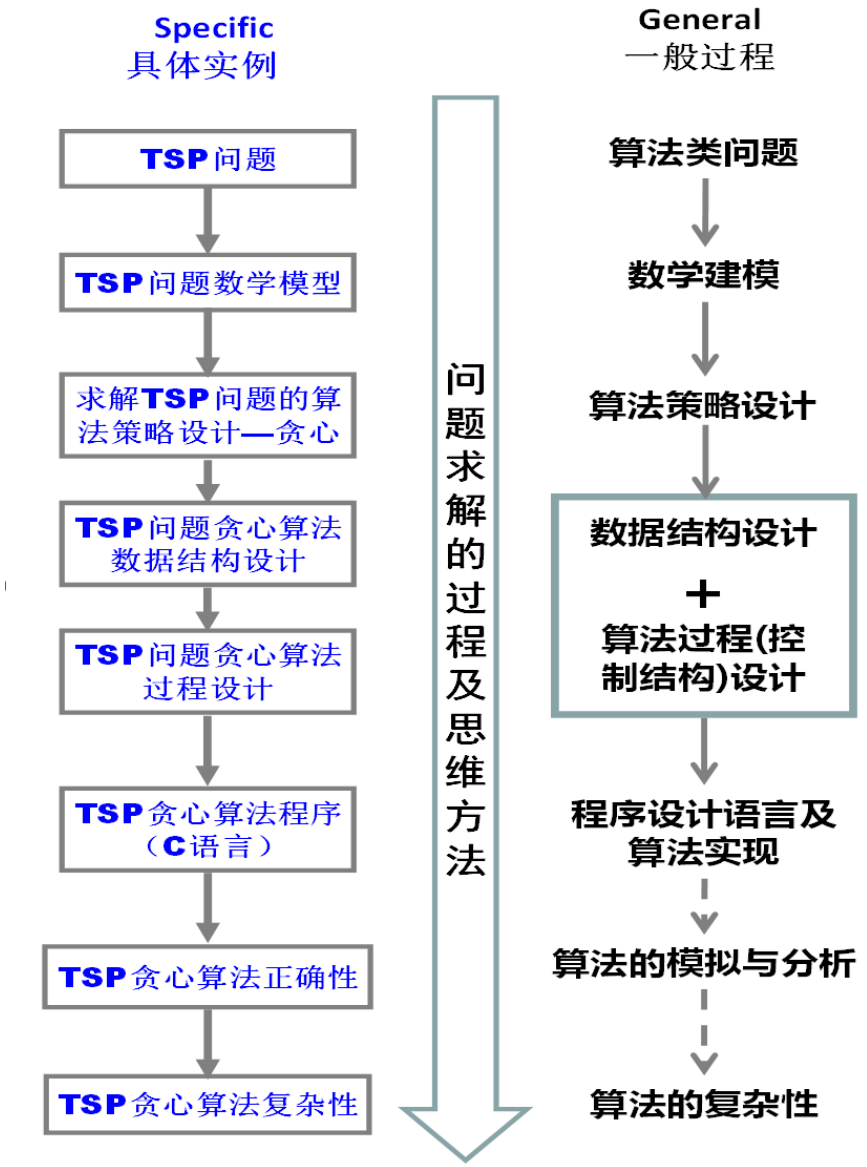
◆TSP问题(**Traveling Salesman Problem**，旅行商问题)，威廉哈密尔顿爵士和英国数学家克克曼T.P.Kirkman于19世纪初提出TSP问题.

◆**TSP问题**：有若干个城市，任何两个城市之间的距离都是确定的，现要求一旅行商从某城市出发必须经过每一个城市且只能在每个城市逗留一次，最后回到原出发城市，问如何事先确定好一条最短的路线使其旅行的费用最少。

城市之间的距离



2.3.2 算法设计过程概述-算法类问题求解框架



2.3.2 算法设计过程概述-第一步 数学建模

将TSP问题抽象为数学问题

输入： n个城市，记为V={$v_1, v_2, \dots, v_n$}，任意两个城市$v_i, v_j \in V$之间有距离$d_{v_i v_j}$

输出： 所有城市的一个访问顺序T={$t_1, t_2, \dots, t_n$}，其中$t_i \in V$，$t_{n+1} = t_1$,使得$min \sum_{i=1}^{n} d_{t_i t_{i+1}}$.

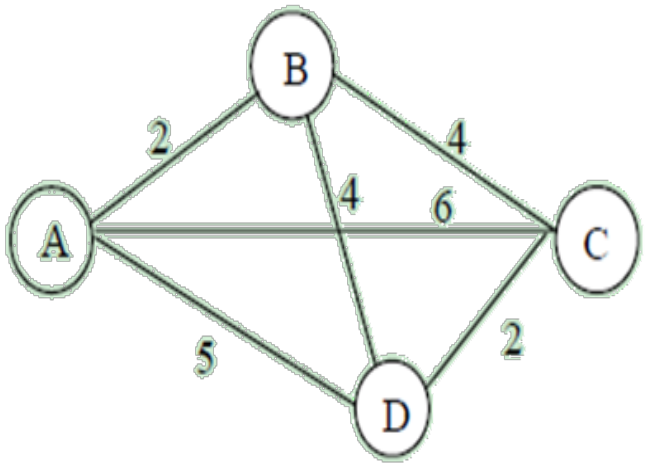
进一步简化

输入： 图中有n个节点，记为V={1, 2, …, n}，任意两个节点i，j之间的距离有$d_{i,j}$

输出： 所有节点的一个访问顺序 T={$t_1, t_2, \dots, t_n$}，其中$t_i \in V$，$t_{n+1} = t_1$,使得$\min \sum_{i=1}^{n} d_{t_i t_{i+1}}$.

问题求解的基本思想： 在所有可能的访问顺序T构成的状态空间Ω上搜索使得$\sum_{i=1}^{n} d_{t_i t_{i+1}}$最小的访问顺序$T_{opt}$。

蛮干/遍历

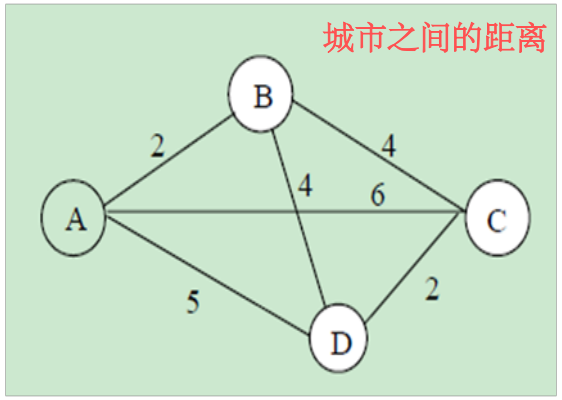


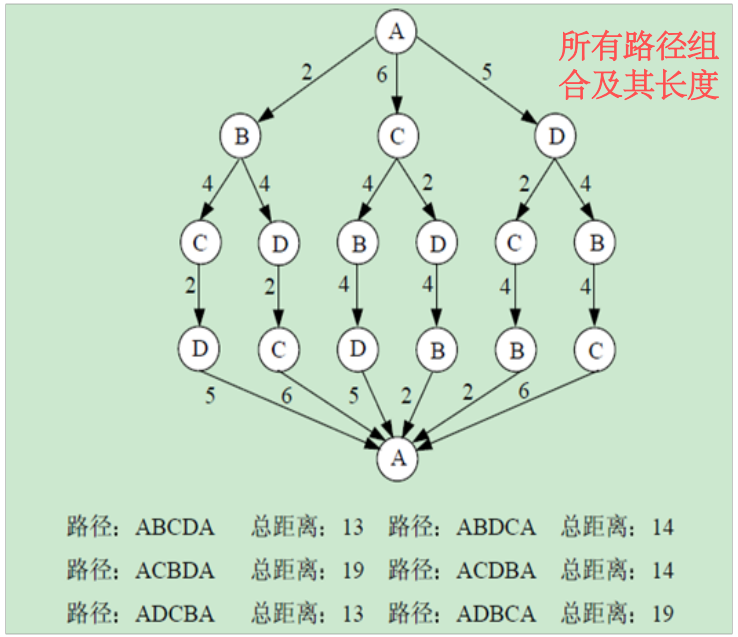
2.3.2 算法设计过程概述-第二步 算法策略设计

蛮干/遍历是基本的算法策略

- **遍历算法**

列出每一条可供选择的路线，计算出每条路线的总里程，最后从中选出一条最短的路线。





2.3.2 算法设计过程概述-第二步 算法策略设计

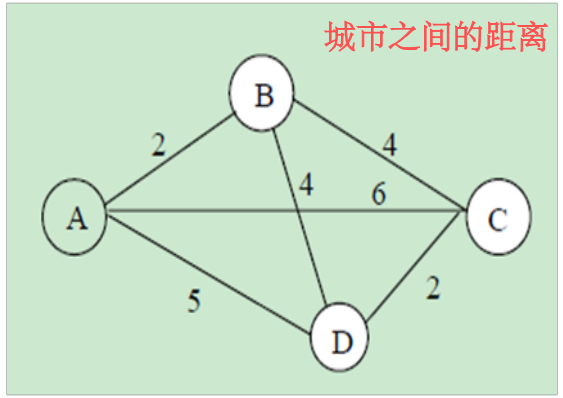
蛮干/遍历策略会带来怎样的问题?

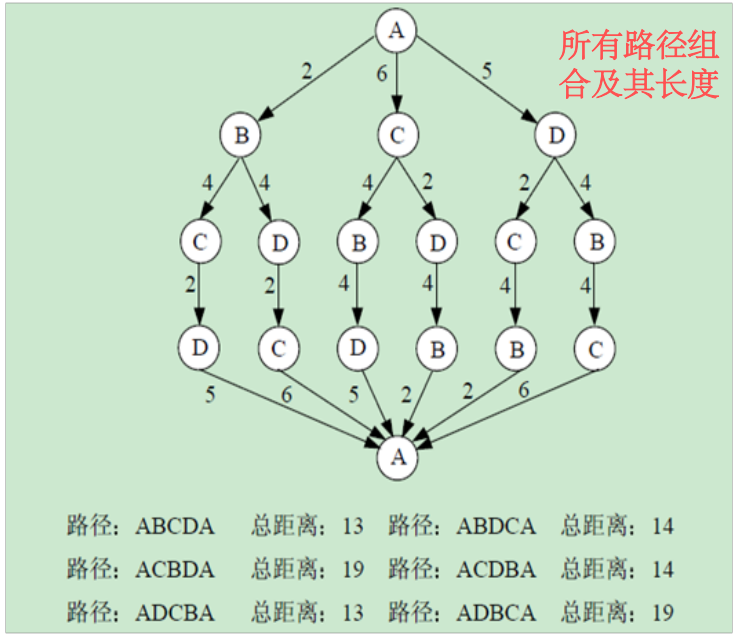
- **组合爆炸**

✓路径组合数目：(n-1)!

✓20个城市，遍历总数1.216×10^{17}

✓计算机以每秒检索1000万条路线的计算速度，需386年。





2.3.2 算法设计过程概述-第二步 算法策略设计

问题的难解性

◆TSP**问题的难解性**：随着城市数量**n**的上升，TSP问题的遍历计算量**(n-1)！** 剧增，计算资源将难以承受。

◆2001年解决了全球**15112**个城市的TSP问题，使用了美国Rice大学和普林斯顿大学之间互连的、速度为500MHz的Compaq EV6 Alpha 处理器组成的**110台计算机**，所有计算机花费的时间之和为**22.6年**。

An optimal TSP tour through Germany's 15 largest cities. It is the shortest among 43 589 145 600 possible tours visiting each city exactly once.

2.3.2 算法设计过程概述-第二步 算法策略设计

寻找其他策略降低计算量

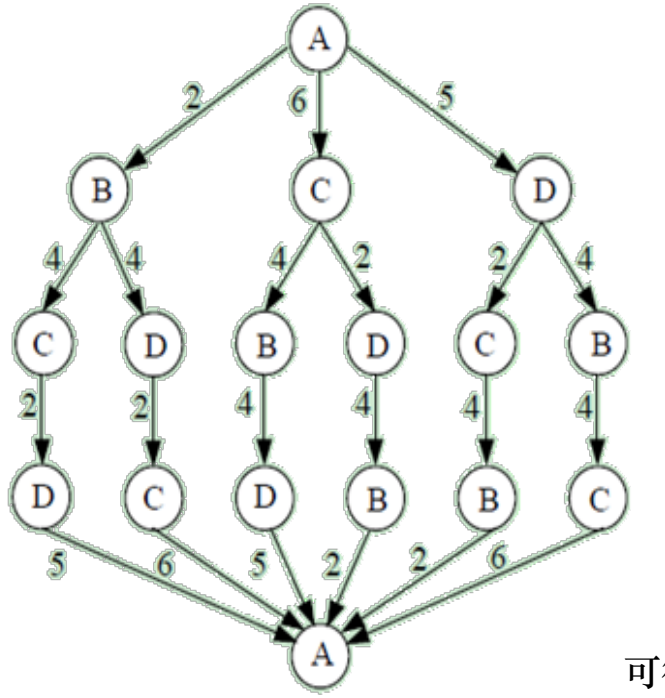
TSP问题，有没有其他求解算法呢？

◆**可行解:**满足所有约束条件的解

◆**最优解：** 在所有可行解中，能够使目标函数达到最优值（最大或最小值）的解

◆不同的算法设计策略：

- ✓ 遍历、搜索算法
- ✓ 分治算法
- ✓ 贪心算法
- ✓ 动态规划算法
- ✓ ……



可行解

最优解

路径：ABCDA 总距离：13　路径：ABDCA 总距离：14

路径：ACBDA 总距离：19　路径：ACDBA 总距离：14

路径：ADCBA 总距离：13　路径：ADBCA 总距离：19

TSP问题的可行解与最优解示意

2.3.2 算法设计过程概述-第二步 算法策略设计

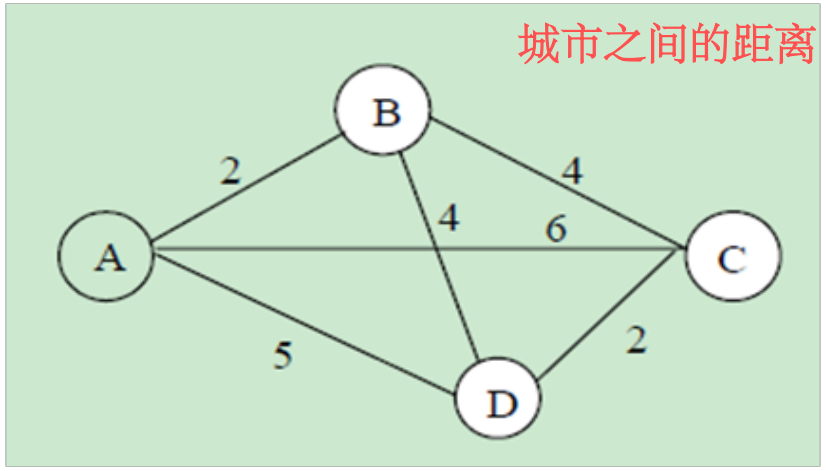
另一种策略：贪心算法

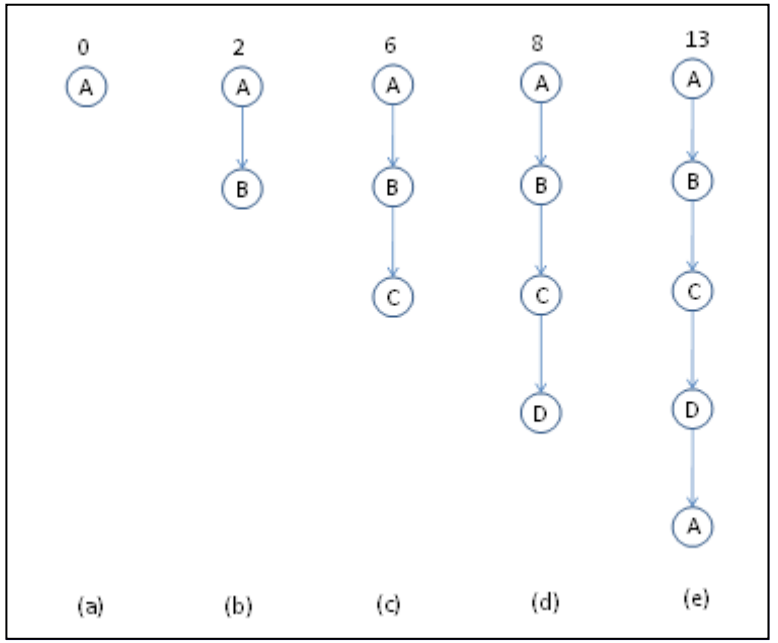
- **贪心算法**

基本思想“今朝有酒今朝醉”:**一定要做当前情况下的最好选择，否则将来可能会后悔，故名“贪心”。**

✓从某一个城市开始，每次选择一个城市，直到所有城市都被走完。

✓**每次在选择下一个城市的时候，只考虑当前情况，保证迄今为止经过的路径总距离最短。**



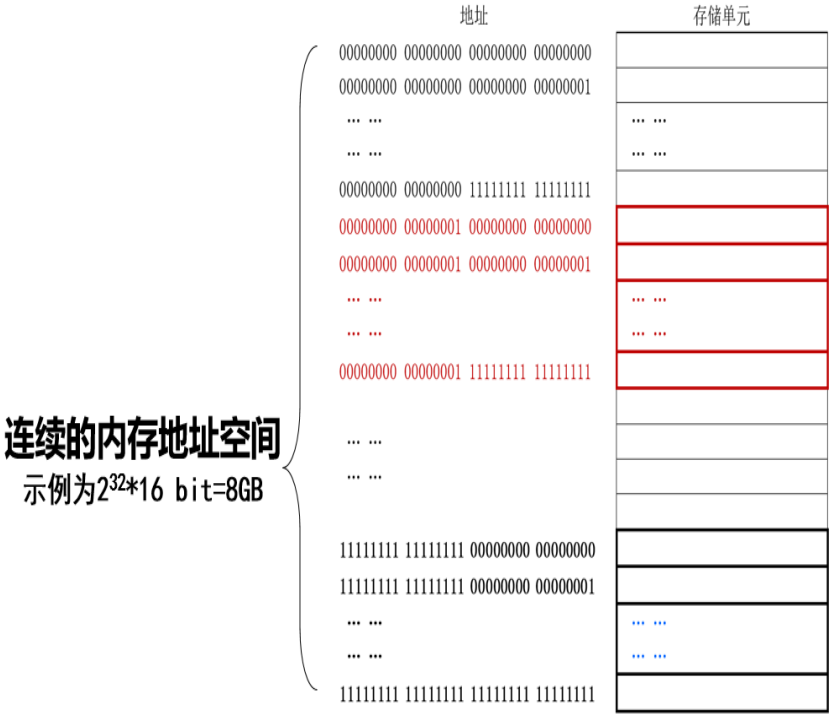


2.3.2 算法设计过程概述-第三步 算法数据结构及设计

为什么需要数据结构

- **数据结构**

指数据的逻辑结构、存储结构及其操作的总称，它提供了问题求解/算法的数据操纵机制。



2.3.2 算法设计过程概述-第三步 算法数据结构及设计

TSP问题的数据结构?

城市映射为编号: A---1, B---2, C---3, D---4

城市间距离关系: 二维表

城市编号	1	2	3	4
1		2	6	5
2	2		4	4
3	6	4		2
4	5	4	2	

D

D[2][3]

城市的访问顺序：一维表

S	1	4	3	2
	S[1]	S[2]	S[3]	S[4]

{A->D->C->B->A}

2.3.2 算法设计过程概述-第四步 算法控制结构设计

算法的控制结构：类自然语言表达方法

- **算法与程序的基本控制结构**

✓**顺序结构**：“**执行A，然后执行B**”，是按顺序执行一条条规则的一种结构 。

✓**分支结构**：“**如果Q成立，那么执行A，否则执行B**”，Q是某些逻辑条件，即按条件判断结果决定执行哪些规则的一种结构 。

✓**循环结构**：控制指令或规则的多次执行的一种结构---迭代(iteration)

◆**循环结构又分为有界循环结构和条件循环结构。**

✓**有界循环**：“**执行A指令N次**”，其中N是一个整数。

✓**条件循环**：某些时候称为无界循环，“**重复执行A直到条件Q成立**”或“**当Q成立时反复执行A**”，其中Q是条件。

2.3.2 算法设计过程概述-第四步 算法控制结构设计

算法的控制结构：流程图表达方法

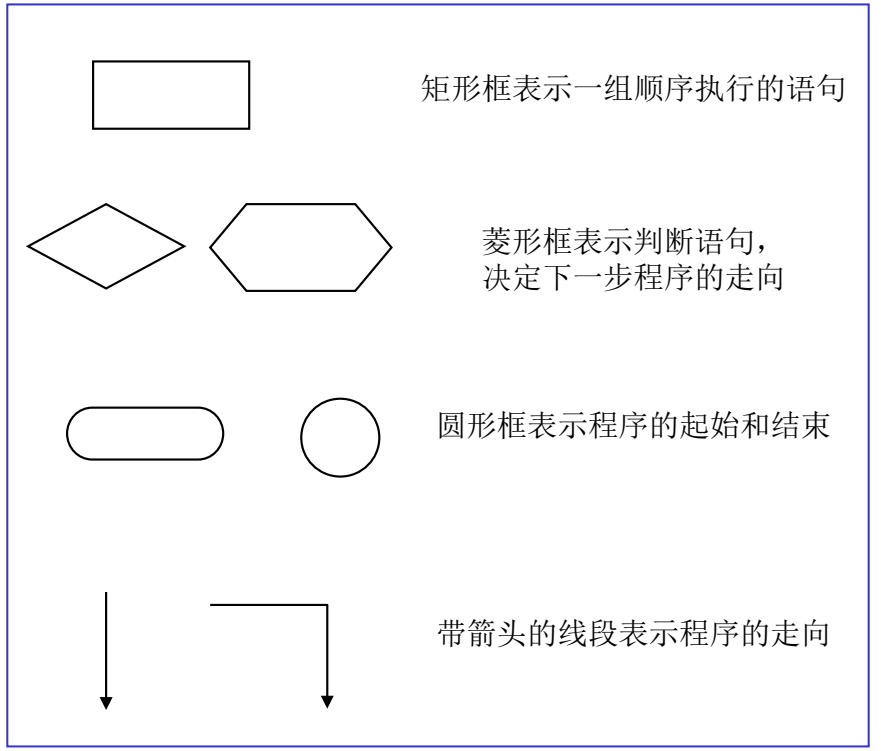
流程图的基本表示符号

✓**矩形框**：表示一组顺序执行的规则或者程序语句。

✓**菱形框**：表示条件判断，并根据判断结果执行不同的分支。

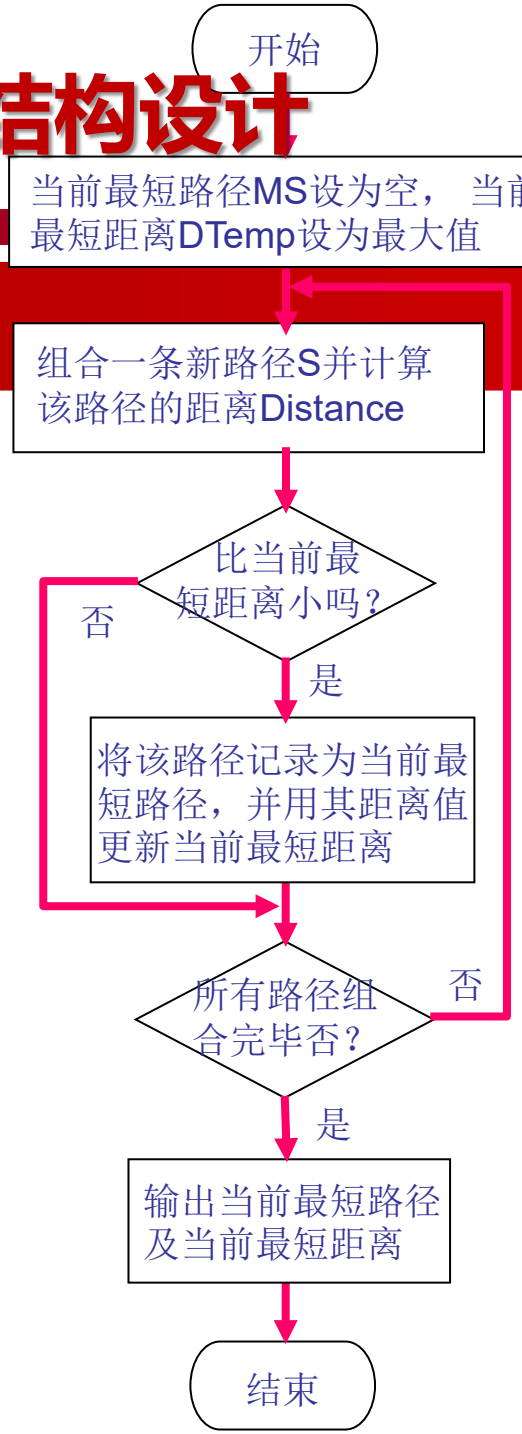
✓**圆形框**：表示算法或程序的开始或结束。

✓**箭头线**：表示算法或程序的走向。



2.3.2 算法设计过程概述-第四步 算法控制结构设计

求解TSP问题的遍历算法的表达



2.3.2 算法设计过程概述-第五步 程序设计

实现算法的程序：阅读与理解

```
main() {  …     寻找下一个未访问过的城市j,且其距当前访问城市S[I-1]距离最短
    K=1;   Dtemp=10000;
    do{  // K从1到n-1循环
        L=0;   Found=0;
        do{
            if(S[L]==K)
            {   Found=1;   break;   }
            else L=L+1;
         } while(L<I);     //L从1到I循环
        if (Found==0 && D[K][S[I-1]]<Dtemp)
        {   j=K;   Dtemp=D[K][S[I-1]];   }
        K=K+1;
    } while(K<n);                        //K从1到n-1循环
    S[I]=j; Sum=Sum+Dtemp;
    … }
```

- K从1,…,n-1是城市的编号
- I是至今已访问过的城市数
- S[I-1]中存储的是当前访问过的城市编号，要找第I个程序

S	1	4	3	2
	S[0]	S[1]	S[2]	S[3]

- D[K][S[I-1]]为城市K距当前访问过的城市的距离

2.3 算法概念、设计与分析概述

2.3.1 算法基本概念

2.3.2 算法设计过程概述

2.3.3 算法分析方法概述

2.3.3 算法分析方法概述-算法正确性分析

◆正确的算法

如果一个算法对于每一个输入都最终停止,而且产生正确的输出，那么它是正确的

◆不正确的算法

①不停止(在某个输入上)

②对所有输入都停止，但对某输入产生不正确结果

20世纪60年代，美国一架发往金星的航天飞机由于控制程序出错而永久丢失在太空中！

◆算法正确性证明

① 证明算法对所有输入都停止且对每个输入都产生正确结果

② 注意，通过调试程序，只能证明程序有错， 不能证明程序无错误!

2.3.3 算法分析方法概述-算法复杂度分析

◆时间复杂性

- 一个算法对特定输入的时间复杂性是该算法对该输入产生结果需要的操作数或“步”数
- 时间复杂性是输入大小的函数 我们假设每一步的执行需要常数时间，实际上每步需要的时间量可能不同。

- 大O表示法，记作O(f(n))，用于描述一个函数相对于另一个函数的增长率。
  例如f(n) = O(g(n)) 表示的含义是f(n)以g(n)为上界

2.3.3 算法分析方法概述-算法复杂度分析

◆空间复杂性：

- 一个算法对特定输入的空间复杂性是该算法对该输入产生结果所需要的存储空间大小

2.3.3 算法分析方法概述-算法复杂度分析

不同的算法时间复杂性量级

常见复杂度大小比较

$$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

$O(n^3)$	$O(3^n)$
0.2秒	1015年
n=60，假设计算机每秒计算百万次	

不同算法复杂度的直观比较

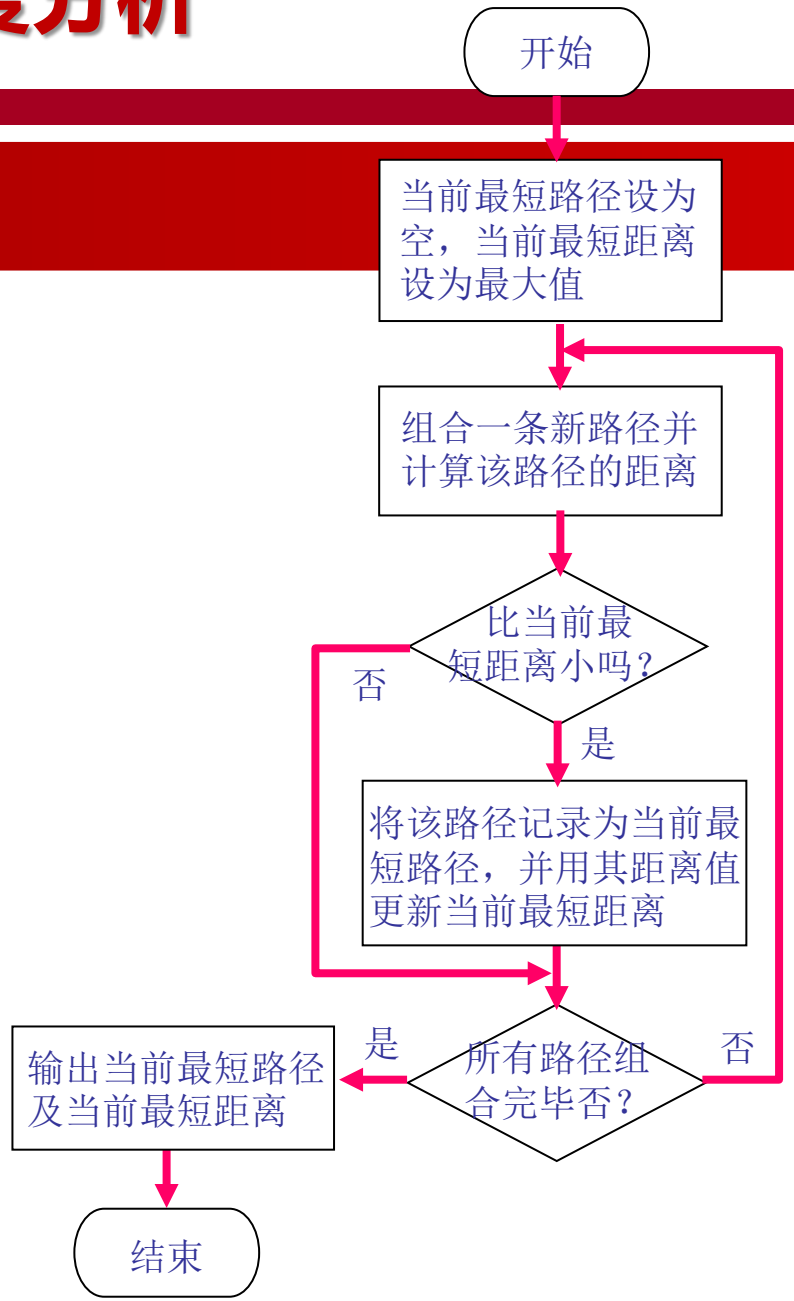
2.3.3 算法分析方法概述-算法复杂度分析

TSP问题遍历算法的复杂性分析

✓列出每一条可供选择的路线，计算出每条路线的总里程，最后从中选出一条最短的路线。

✓路径组合数目为(n-1)!

✓时间复杂度是O((n-1)!)



本讲重点

一、机器语言、汇编语言、高级语言定义

二、机器语言、汇编语言、高级语言程序执行过程

三、算法的概念与特征

四、算法类问题的求解框架

五、算法的时间复杂性