

# 数据结构

## 第一讲




# 数据结构的发展

**1968年**

美国的唐.欧.克努开创

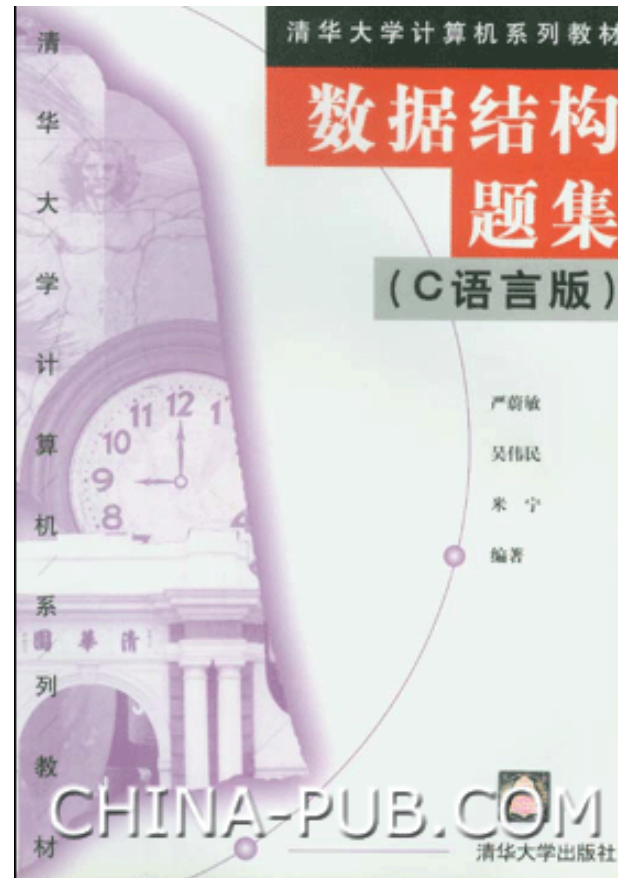
大型程序



**20年代70到80年代**

各种版本的数据结构著作出现

# 教材



- 【作者】 [严蔚敏;吴伟民 \[同作者作品\]](#)
- 【丛书名】 [清华大学计算机系列教材](#)
- 【出版社】 [清华大学出版社](#)

# 参考资料

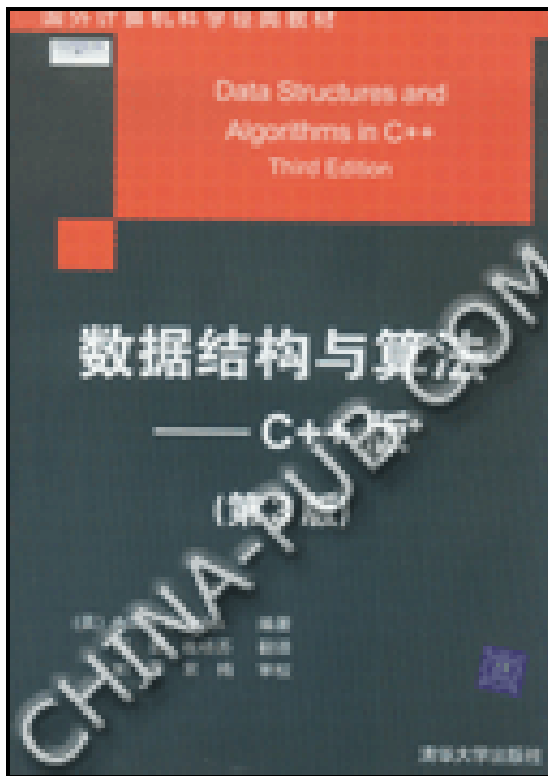


【作者】 李春保 苏光奎  
【出版社】 清华大学出版社  
北京科海电子出版社



【作者】 高一凡  
【出版社】 西安电子科技大学  
出版社

# 参考资料



【作者】 (美) Adam Drozdek  
【译者】 郑岩[同译者作品] 战晓苏



【作者】 殷人昆等  
【丛书名】 清华大学计算机系列教材

# 参考资料



【作者】 [徐孝凯](#)  
【出版社】 [清华大学出版社](#)



【作者】 [黄国瑜 叶乃菁](#)  
【出版社】 [清华大学出版社](#)

# 绪论

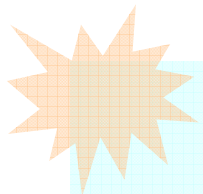
## 本章主要内容

**什么是数据结构**

**基本概念和术语**

**抽象数据类型的表示和实现**

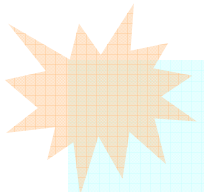
**算法和算法分析**



# 什么是数据结构

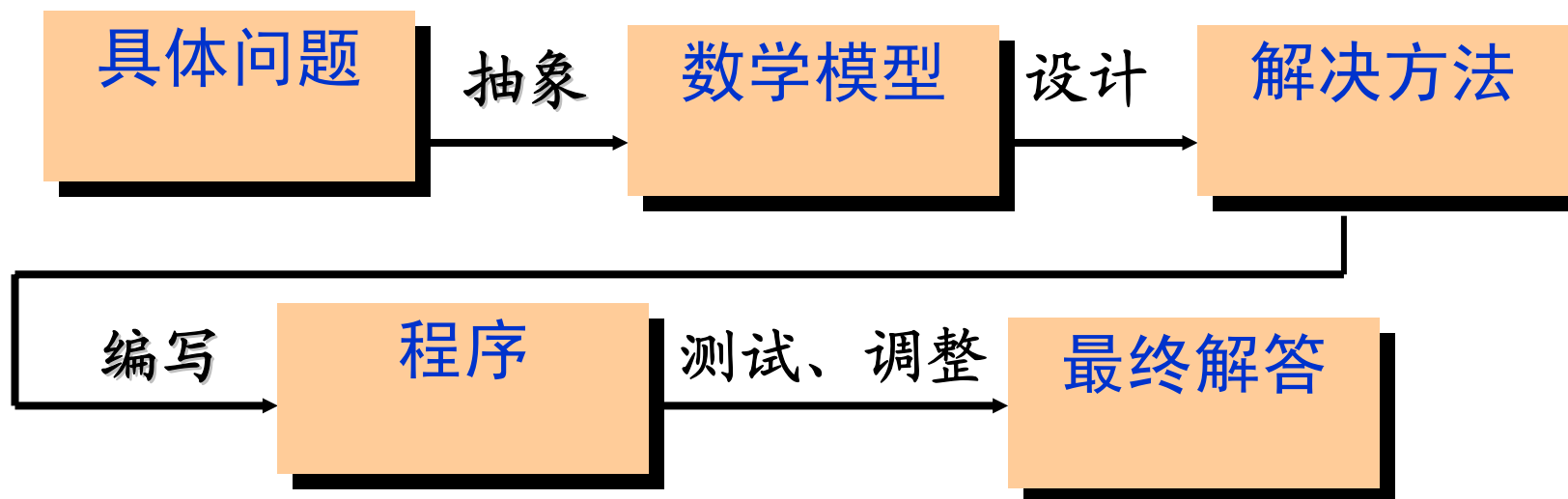
- ✓ 计算机是一门研究用计算机进行**信息表示和处理**的科学。
- ✓ 信息的表示和组织又直接关系到处理信息的程序的**效率**。
- ✓ **分析待处理的对象的特征及各对象之间存在的关系**，这就是数据结构这门课所要研究的问题。





# 什么是数据结构

- 用计算机解决具体问题的步骤



# 数据结构的引入-1

## [实例1]

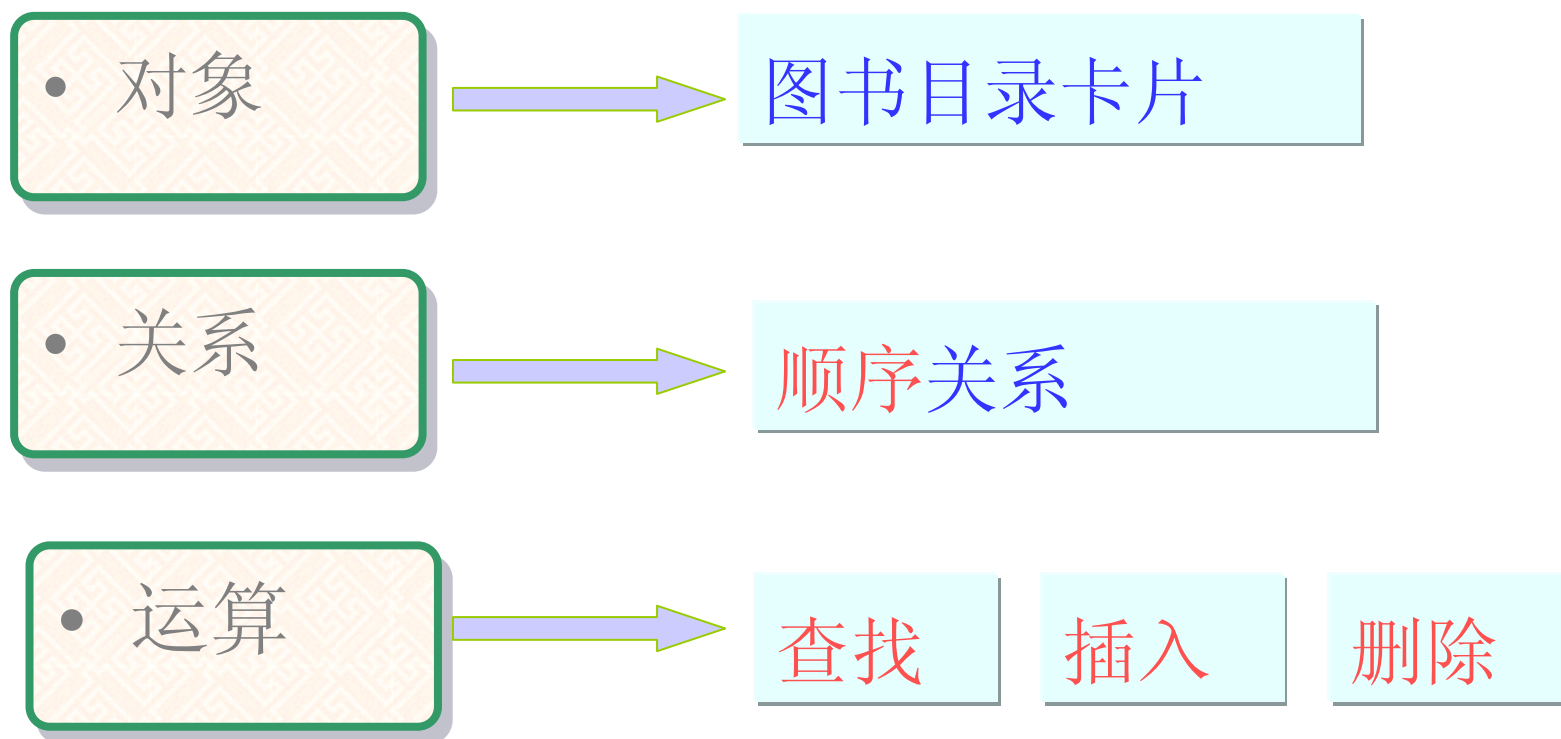
- 计算机管理图书目录问题
  - 从计算机管理图书目录问题抽象出来的模型  
即是包含图书目录的表和对表进行查找运算

。

| 书名     | 作者   | 登录号      | 分类号        | 出版日期      | 定价   |
|--------|------|----------|------------|-----------|------|
| Java语言 | 李晓 等 | 97000018 | 73.8792-99 | 1977/3/26 | 43.5 |
| UNIX系统 | 张 昊  | 96000129 | 73.874-126 | 1996/9/19 | 23.5 |
| ...    | ...  | ...      | ...        | ...       | ...  |

# 数据结构的引入-1

## [实例1]



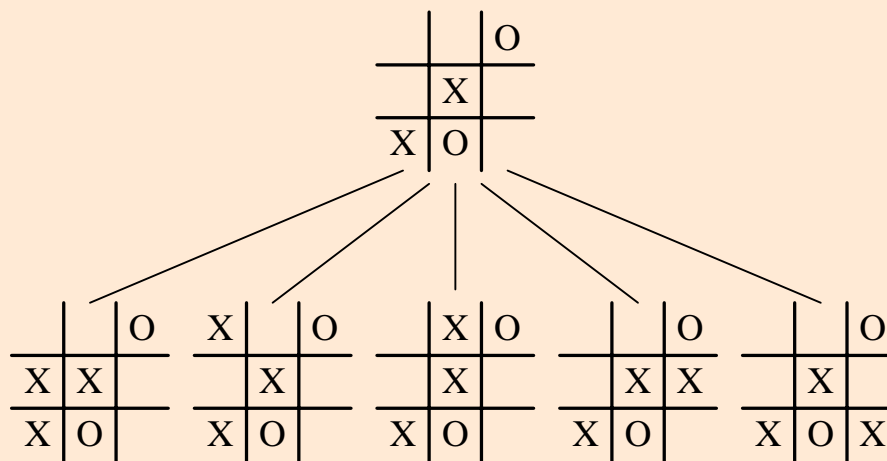
# 数据结构的引入-2

## [实例2]

- 计算机和人对弈问题

|   |   |   |
|---|---|---|
|   |   | O |
|   | X |   |
| X | O |   |

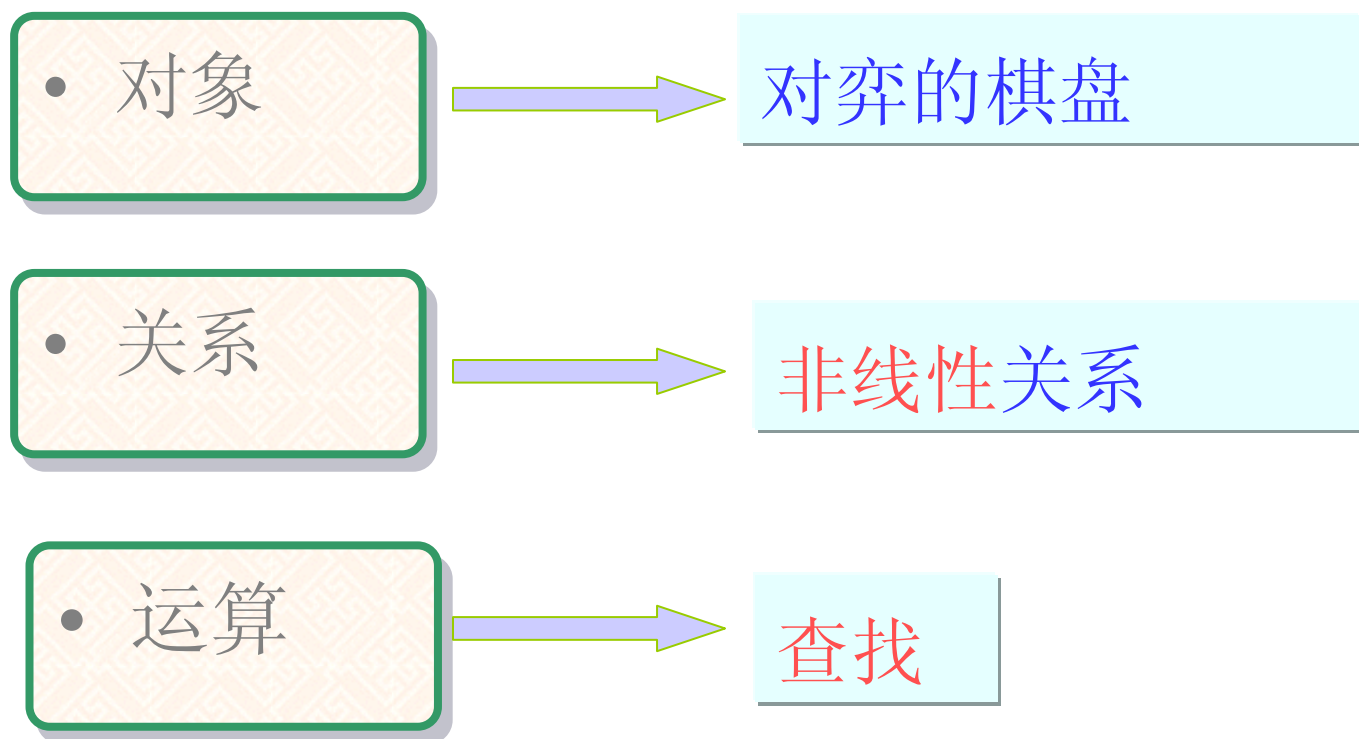
(a) 对弈树的数据元素



(b) 对弈树的一部分

# 数据结构的引入-2

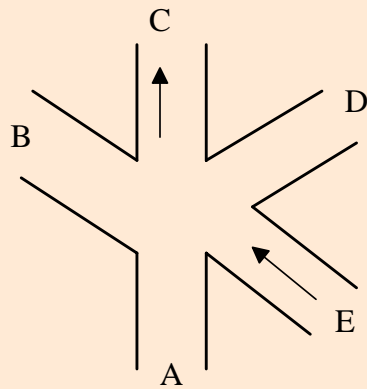
## [实例2]



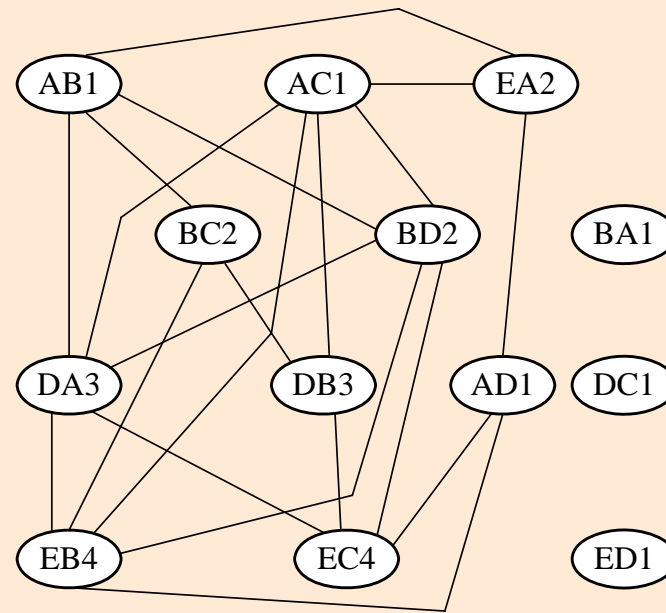
# 数据结构的引入-3

## [实例3]

- 多叉路口交通灯管理问题



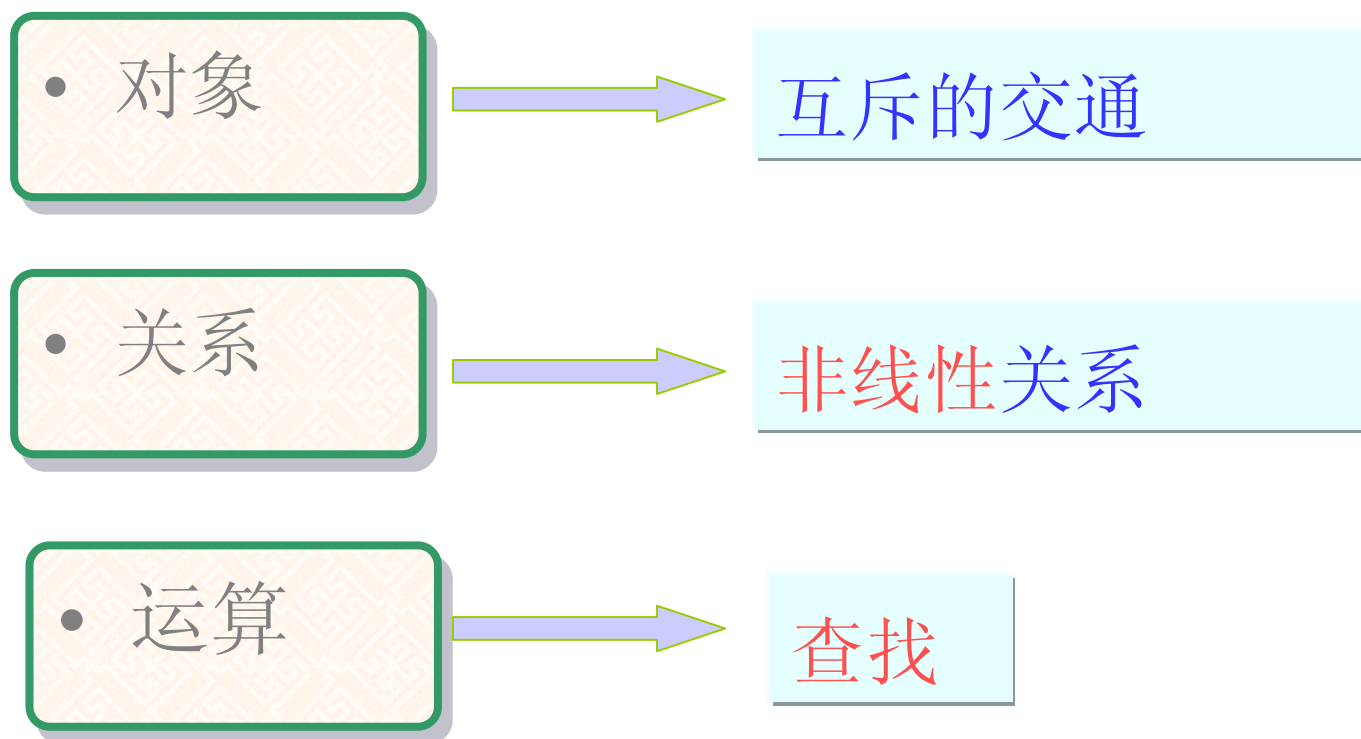
(a) 五叉路口

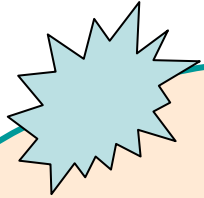


(b) 通路图

# 数据结构的引入-3

## [实例3]





数据结构：就是一门研究数值或非数值性程序设计中计算机操作的对象以及它们之间的关系和运算的一门学科。



# 数据结构的主要内容

## 研究内容

- 研究数据元素之间固有的客观联系，即数据的逻辑结构；
- 研究数据在计算机内部的存储方法，即为数据的存储结构，又称物理结构；
- 研究如何在数据的各种结构（逻辑的和物理的）上施加有效的操作或处理（算法）。

# 数据结构的基本概念

**数据(Data)**是描述客观事物的数、字符以及所有能输入到计算机中并被计算机程序处理的符号的集合。它是计算机程序加工的“原料”。

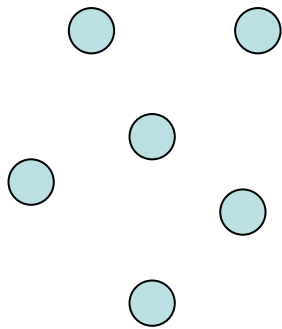
**数据元素(Data Element)**是数据的基本单位，即数据这个集合中的一个个体（客体）。有时一个数据元素可由若干个**数据项(Data Item)**组成，数据项是数据的最小单位。

**数据对象(Data Object)**具有**相同**特性的数据元素的集合，是数据的一个子集。例如，整数的数据对象是集合 $N = \{0, \pm 1, \pm 2, \pm 3, \dots\}$ ，字母字符的数据对象是集合 $C = \{A, B, \dots, Z\}$ 。

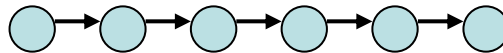
# 数据结构的基本概念

数据结构(Data Structure)是相互间存在一种或多种特定关系的数据元素的集合。

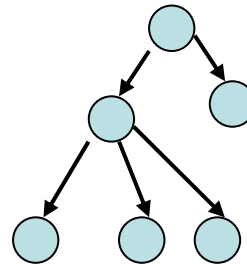
数据相互的关系称为结构( Structure)，也被称为逻辑结构。



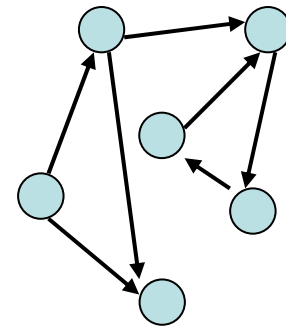
集合



线性



树



图（网）

# 数据结构的基本概念

数据结构(Data Structure) 的形式定义,  
数据结构是一个二元组

$$\text{Data-Structure} = (D, R)$$

其中,  $D$ 是数据元素的集合,  $R$ 是 $D$ 上关系的集合。

## [例]

复数是一种数据结构  $\text{Complex}=(C,R)$

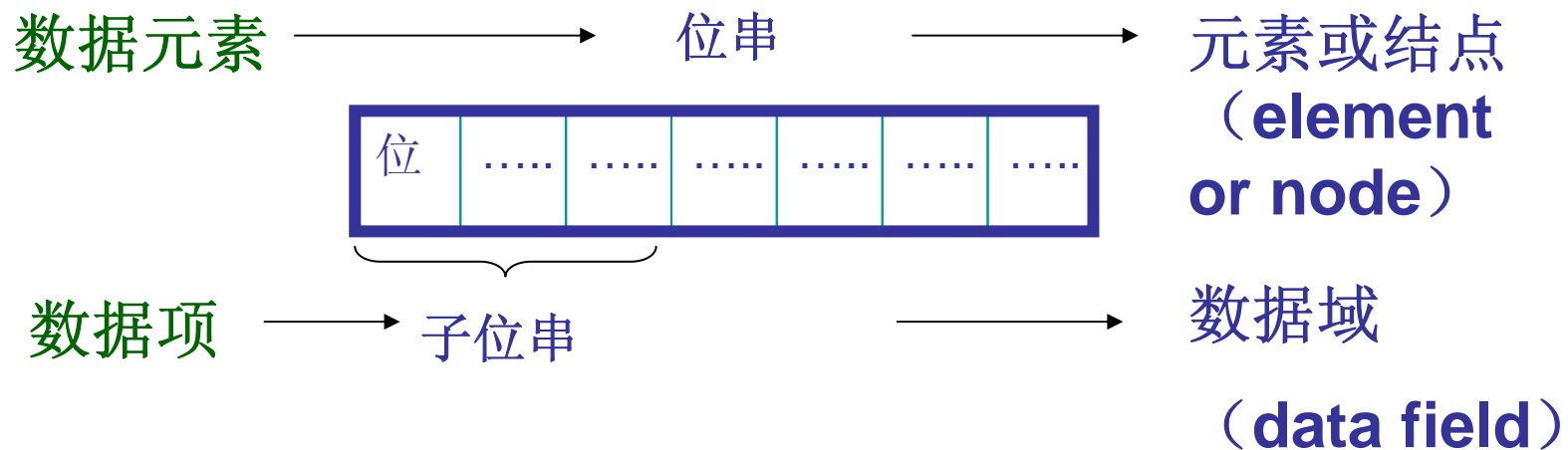
其中:  $C$ 含有两个实数的集合 $\{c1,c2\}$ ;

$R=\{P\}$ ,  $P$ 是定义在集合 $C$ 上的一种关系 $\{<c1,c2>\}$ ,  
其中 $C$ 是有序偶 $<c1,c2>$ 表示 $c1$ 是复数的实部,  $c2$ 是复数的虚部。

# 数据结构的基本概念

数据结构在计算机中的表示称为数据的**物理结构**，也被称为**存储结构**。它包括**数据元素的表示**和**关系的表示**。

## ➤ 数据元素的表示



# 数据结构的基本概念

## ➤ 存储结构的描述

**顺序映像：**借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系

**非顺序映像：**借助指针表示数据元素之间的逻辑关系

“一维数组”

顺序存储结构

链式存储结构

指针

[例] 复数  $z=3.0-2.3i$

|      |       |
|------|-------|
|      | ..... |
| 0415 | 3.0   |
| 0417 | -2.3  |
|      | ..... |

|      |      |
|------|------|
|      | .... |
| 0415 | -2.3 |
|      | .... |
| 0611 | 3.0  |
| 0613 | 0415 |
|      | .... |

# 数据结构的基本概念

## ➤数据类型（**data type**）

一个值的集合和定义在这个值集上的一组操作的总称。

在C语言中

数据类型：基本类型和构造类型

基本类型：整型、浮点型、字符型

构造类型：数组、结构、联合、指针、枚举型、自定义

{ 原子类型：值不可分解（整型、浮点型）  
结构类型：由若干成分按某种结构组成（数组）

# 数据结构的基本概念

## ➤抽象数据类型（**abstract data type, ADT**）

一个数学模型以及定义在该模型上的一组操作。抽象了数学特性；并且范围更广。

- 原子类型：值不可分解
- 固定聚合类型：其值由确定数目的成分按某种结构组成
- 可变聚合类型：值的数目是不确定的

**ADT用三元组表示（D，S，P）**



# 数据结构的基本概念

## ➤ ADT的定义形式

ADT抽象数据类型名{

    数据对象: <数据对象的定义>

    数据关系: <数据关系的定义>

    基本操作: <基本操作的定义>

} ADT抽象数据类型名

基本操作名（参数表）

    初始条件: <初始条件的描述>

    操作结构: <操作结果的描述>

# 数据结构的基本概念

[例]

ADT Triplet {

数据对象:  $D=\{e1,e2,e3|e1,e2,e3 \in ElemSet\}$

数据关系:  $R1=\{<e1,e2>,<e2,e3>\}$

基本操作:

InitTriplet(&T,v1,v2,v3)

操作的结果: 构成了三元组T, 元素e1,e2,e3  
分别被赋以参数v1,v2,v3的值。

.....

} ADT Triplet

} ADT抽象数据类型名

# 抽象数据类型的表示与实现

## ■ 预定义的常量和类型:

```
# define TRUE 1  
typedef int Status;
```

## ■ 数据结构（存储结构）的表示:

用类型定义typedef 表示;

# 抽象数据类型的表示与实现

## ■ 赋值语句:

变量名=表达式;

变量名1=变量名2=.....=表达式;

## ■ 选择语句:

✓ 条件语句:

If (表达式)

语句;

else

语句;

✓ 开关语句:

Switch (表达式) {

Case 值1: 语句1; break;

.....

default: 语句n+1;

}

# 抽象数据类型的表示与实现

## ■ 循环语句:

✓ For语句

for (赋初值表达式; 条件; 修改表达式) 表达式)  
语句;

✓ while语句:

while (条件) 语句;

## ■ 结束语句:

✓ 函数结束: return;

✓ Case结束 : break;

✓ 异常结束 : exit;

# include 预处理器指示符

## preprocessor include directive

预处理器指示符用# 号标识，这个符号将放在程序中该行的最起始一列上。

**#include** 指示符读入指定文件的内容，它有两种格式

**#include <some\_file.h>**

**#include "my\_file.h"**

如果文件名用尖括号< 和> 括起来，表明这个文件是一个工程或标准头文件，查找过程会检查预定义的目录

# C实例

```
#include <stdio.h>
```

```
void swap(int*,int*);
```

```
void main()
```

```
{
```

```
    int a=10,b=20;
```

```
    int*p1=&a,*p2=&b;
```

```
    printf("%d,%d\n",a,b);
```

```
    swap(p1,p2);
```

```
    printf("%d,%d\n",a,b);
```

```
}
```

```
void swap(int*x,int*y)
```

```
{
```

```
    int temp=*x;
```

```
    *x=*y;
```

```
    *y=temp;
```

```
}
```

定义头函数

定义子函数

编写主函数

编写子函数

# 记录

## 定义

```
struct record_name  
{ data1type data1;  
  data2type data2;  
  .....  
  datatype datan;  
}  
record_name record;
```

## 引用

record.data1

record.data2



# 指针

## C程序内存分配的问题

- 静态分配——即编译器在处理程序源代码时分配
- 动态分配——即程序执行时调用运行时刻库函数来分配

### 静态与动态内存分配主要区别：

- 静态对象是有名字的变量，我们直接对其进行操作。
- 动态对象是没有名字的变量，我们通过指针间接地对它进行操作。

指针的主要用处是管理和操纵动态分配的内存。

解引用dereference 操作符\*，定义指针

取地址address-of 操作符&，把它应用在一个对象上时返回的是对象的地址值。

- 当指针为空时表明它没有指向任何对象

```
pint = NULL;
```

- 指针不能持有非地址值

```
// 错误pi 被赋以int 值ival
```

```
pint = ival;
```

- 指针不能被初始化或赋值为其他类型对象的地址值

```
double *dval;
```

```
pint = dval;
```

```
// 都是编译时刻错误
```

```
// 无效的类型赋值: int* <== double*
```

指针的主要用处是管理和操纵动态分配的内存。

```
int ival;
```

```
ival=5;
```

```
int *pint;
```

解引用dereference  
操作符\*, 定义指针

```
pint= (*int) m
```

```
*pint=3;
```

函数malloc, 分配内存  
(\*变量类型) malloc (内存单元大小)

```
pint= &ival ;
```

```
free (pint) ;
```

取地址address-of 操作符&, 把它应用在一个对象上时返回的是对象的地址值。

0x.....32

0x.....30

pint

0x.....30

5

ival

函数free, 释放内存空间。

# 指向记录的指针

```
record_name *recordp;
```

引用

```
recordp -> data1
```

```
*recordp.data1
```

# 算法的概念

## 算法性质

算法是由若干条指令组成的有限序列

- **输入性**：具有零个或多个输入量，即算法开始前对算法给出的初始量；
- **输出性**：至少产生一个输出；
- **有穷性**：每条指令的执行次数必须是有限的；
- **确定性**：每条指令的含义必须明确，无二义性；
- **可行性**：每条指令都应在有限的时间内完成。

# 算法与程序的区别

## 算法与程序

- ✓ 一个程序不一定满足**有穷性**。
- ✓ **可运行性**：程序中的指令必须是机器可执行的，而算法中的指令则无此限制。
- ✓ **两者关系**：算法若用机器可执行的语言来书写，则它就是一个程序。

# 算法的要求

- (1) **正确性(Correctness)** 算法应满足具体问题的需求。
- (2) **可读性(Readability)** 算法应该好读。以有利于阅读者对程序的理解。
- (3) **健壮性(Robustness)** 算法应具有容错处理。当输入非法数据时，算法应对其作出反应，而不是产生莫名其妙的输出结果。
- (4) **效率与存储量需求** 效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般，这两者与问题的规模有关。

# 算法效率的度量

依据算法编制的程序在计算机上运行的时间

方法：

- 事后测试 收集此算法的执行时间和实际占用空间的统计资料。
- 事先分析 求出该算法的一个时间界限函数

从算法中选取一种对于研究问题来说是基本操作的原操作，该基本操作重复执行的次数作为算法的时间度量。



# 渐近时间复杂度

定义：如果存在两个正常数 $c$ 和 $n_0$ ，对于所有的  
 $n \geq n_0$ ，有  $|f(n)| \leq c |g(n)|$

则记作  $f(n)=O(g(n))$

## 渐近时间复杂度

一般情况下，算法中基本操作重复执行的次数是问题规模 $n$ 的某个函数，算法的时间量度记作

$$T(n)=O(f(n))$$

表示随问题规模 $n$ 的增大，算法执行时间的增长率和 $f(n)$ 的增长率相同，称作算法的渐近时间复杂度。

# 频度

多数情况下，基本操作的原操作是最深层循环内语句中的操作，它的执行次数和包含它语句的**频度**相同

**频度 (frequent count) :**

语句重复执行的次数

- 赋值语句、定义语句的频度为 1，时间复杂度为  $O(1)$ ；
- 循环语句中逻辑判断语句的频率为循环次数加1；

# 算法分析实例

例：求两个n阶方阵的乘积  $C = A \times B$

```
#define n 自然数
MATRIXMLT(A,B,C)
float A[][n], B[][n], C[][n];
{ int i,j,k;
  for (i=0;i<n;i++)
    for (j=0; j<n;j++){
      C[i][j]=0;
      for(k=0;k<n;k++)
        C[i][j]=C[i][j]+A[i][k]*B[k][j]; } }
```

频度

$n+1$

$n(n+1)$

$n^2$

$n^2(n+1)$

$n^3$

该算法的时间特性是： $T(n)=2n^3+3n^2+2n+1$ ，  
 $T(n)$ 也称为算法的时间复杂度，简记为： $O(n^3)$

# 常用时间复杂度的表较

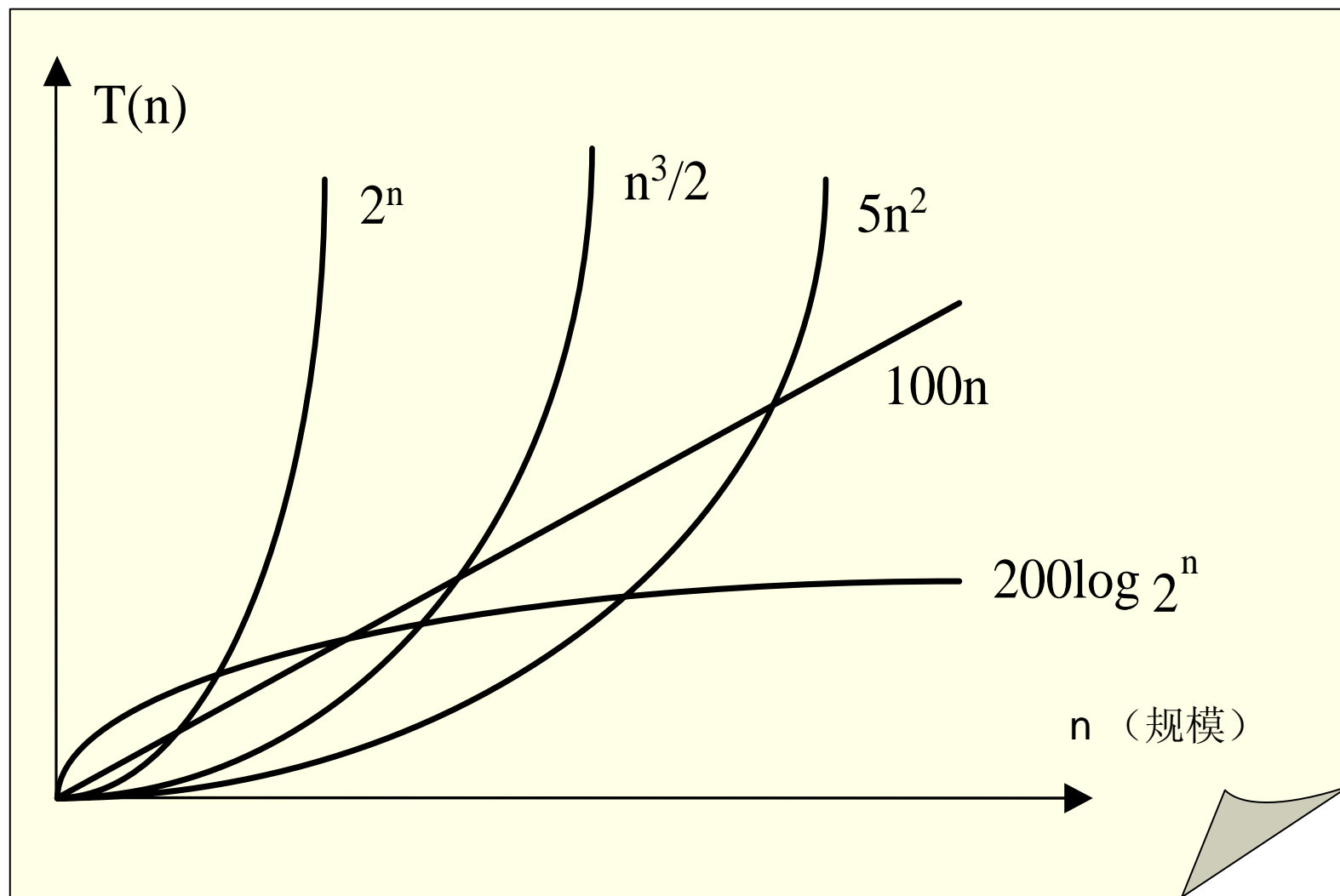
以下六种计算算法时间的多项式是最常用的。其关系为：

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$$

指数时间的关系为：

$$O(2^n) < O(n!) < O(n^n)$$

当 $n$ 取得很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊。因此，只要有人能将现有指数时间算法中的任何一个算法化简为多项式时间算法，那就取得了一个伟大的成就。



各种数量级的 $T(n)$

# 算法的存储空间要求

空间复杂度（**Spase Complexity**）：

依据算法编制的程序所需要的存储空间

记为  $S(n)=O(f(n))$   
其中  $n$  为问题的规模(或大小)

# 小结

数据结构

基本概念和术语

ADT的表示和实现

算法和算法分析

定义

要求

时间复杂度

空间复杂度

# 作业

P8

习题： 1.8 (1),(2),(6),(7)