数据结构与算法

第一章　数据结构与算法

C语言简要回顾

数据结构基本概念

算法分析

第一节 C语言回顾

- C语言的成份
- C语言语句
- C语言变量
- C语言的数据类型
- 函数
- 程序结构
- 结构和联合
- 指针

C语言成份

- C语言的成份：
  - 语句
  - 定义
    - 变量定义
    - 函数定义
  - 申明
    - 变量申明
    - 数据类型申明
    - 函数原型申明
    - 编译指示申明
  - 注释

C语言语句

- C语言的语句用“；”标记。分为以下几种：
  - 赋值语句/表达式语句
  - 函数调用(子程序)
  - 流程控制语句
  - 空语句
  - 复合语句
    (用一对{}括起来)

C语言变量的定义与申明

- 变量的定义

  存储类型 数据类型 变量名 = 初始值;

  int x;

  int x = 3;

  static int x = 3;

  - 变量的作用域：变量只在定义它的(函数)范围内起作用
  - 在所有函数体之外定义的变量称全局变量

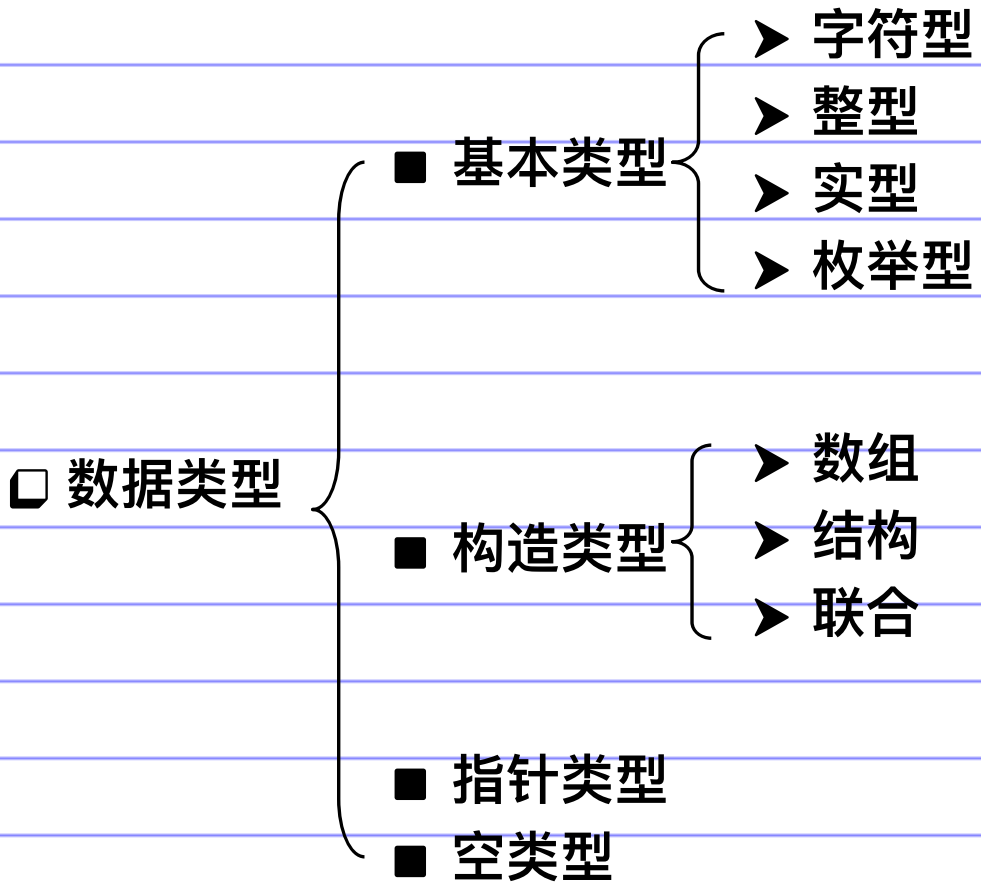
- 变量的申明

  extern int Conter;

  void strcpy(const char *Tar, char *Src);

  - 变量的申明通常是给编译器以指示，并不实际分配内存。因此通常放在头文件中。

C语言的数据类型



C语言基本数据类型

- C语言的基本数据类型包含：字符型、整型、实型、枚举型
- 不同的C编译器可能对一些类型的在内存中的字节数规定不同，可以用sizeof()检测各种类型所需的存储字节数。

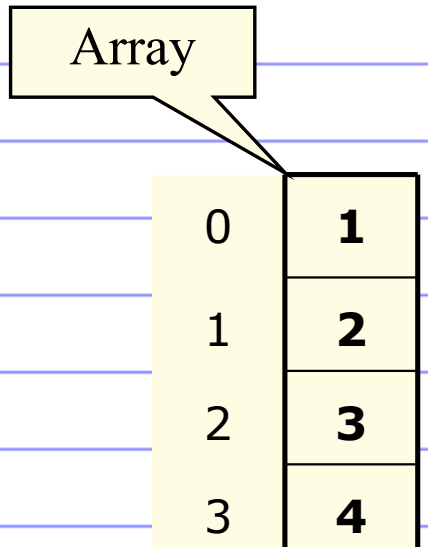
C语言基本数据类型	字节数
unsigned/signed char	1
unsigned/signed short (int)	2
unsigned/signed int	4
unsigned/signed long (int) unsigned/signed float	4
enum	4
unsigned/signed int64 unsigned/signed double	8

Windows API类型	存储字节数
BYTE	1
WORD	2
DWORD	4

数组

- 数组的定义

  存储类型 数据类型 数组名[元素个数]= {初始值表};
  - 数组的初值
  - 数组在内存中的存储：顺序存储
  - 数组的引用：以元素下标引用
    - 下标总是从0开始，到元素个数-1
- 数组引用的实质：地址+偏移
  - 数组等同于一个指针
- 数组的申明




```
static int Array[3];
```

字符数组与字符串

- 字符数组
  - 定义：跟上面的数组一样，数据类型为char
  - 初值：跟上面一样，但增加了一种
- 字符串：字符数组的最后一种形式，自动加0的字符数组

```
char *str = “Hello”;
```

函数

- 函数定义

  存储类型 返回类型 函数名(形参列表) {函数体语句}

  返回类型：void型相当于子程序

  形式参数：传值型、传址型。注意申明形式

  函数调用：非void返回值可赋给变量;可直接作语句用。

- 函数申明

  extern 返回类型 函数名(形参[类型]列表);

  注意：有；符号。形参可只给类型。

  通常函数申明放在头文件中。有时也叫函数原型。

```
int max(x, y)
extern int max( int, int);


;
```

程序结构

- 顺序结构
  - 按照语句的顺序执行
- 流程控制
  - 选择结构
    - if – else
    - switch
  - 循环结构
    - for
    - while { }
    - do { } while
    - do { } until
  - 跳转 goto

- 1966年，Bohra和Jacopini证明了可以只用三种结构表达以上各种结构

  - 顺序结构
  - 选择结构 if – else
  - 循环结构 for

结构

- 结构定义

```
struct 结构名 {
    结构成员类型        结构成员名;
    ……
};
```

- 结构使用

```
struct 结构名 变量名;
```

```
struct StudentInfo aStudent;
struct StudentInfo Students[35];
struct StudentInfo *lpStuentsList;
int x = aStudent.StID;
int y =  Students[10].StID;
```

联合

- 联合定义

union联合名 {

联合成员类型	联合成员名;

```
struct Resident {       |  struct Student {
  int      CertID;      |       int    StID;
  char     Name[16];    |       char   Name[12];
};                      |       };
union Person {
struct Resident         Res;
struct Student          St;
Char                    InfoData[20];
}
```

自定义数据类型

- 使用typedef定义自定义的数据类型

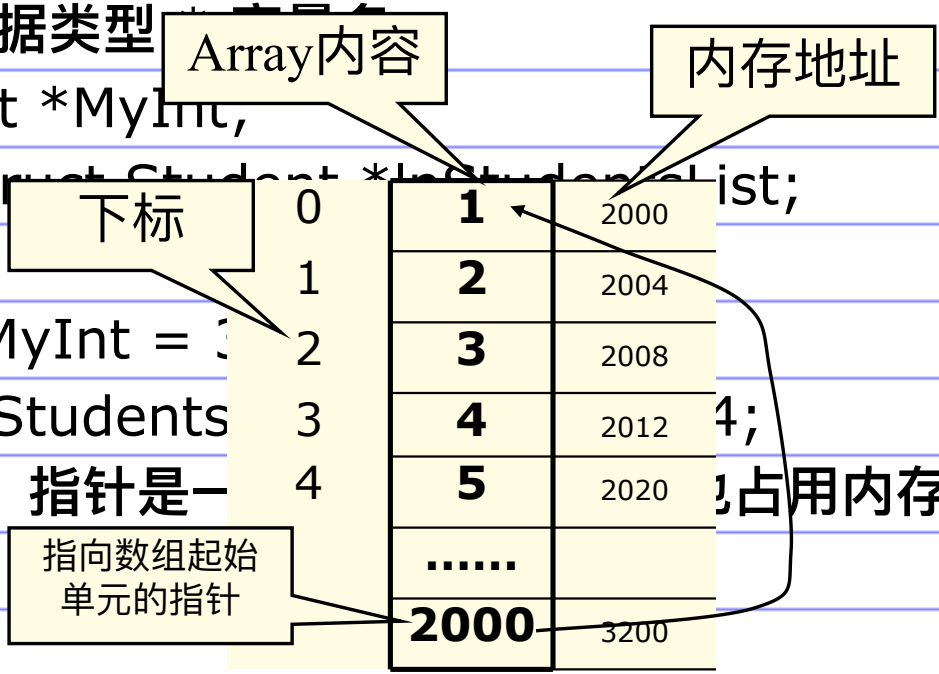
  typedef 原数据类型　　用户数据类型

- 使用：等同于系统预定义的数据类型
- 自定义数据类型与宏定义不同，宏定义只是展开，因此数据类型不变，而自定义类型则不同，是一种新的数据类型。很多编译器在编译时进行类型检查，以确定是否违例

```
BYTE        aByte, *lpByte;
Student     aStudent;
LpStudent   lpStudentList;
```

地址指针

- 程序在内存中才能得以执行，数据在内存中才能被直接访问。内存单元是以地址标记的。
- 定义：
  - 数据类型
  - Int *MyInt;
  - Struct Student *lpStudentList;
- 使用：
  - *MyInt = 1
  - lpStudents 4;
- 注意：指针是一 占用内存。



函数指针

- 每一个函数被编译成代码后，都会装入内存以执行，它们在内存中都会有一个入口地址。可以用一个变量表示这个入口地址，称为指向这个函数的指针。
- 在编写程序时，程序还没编译，如何知道函数指针呢?
  - 首先申明一个与该函数匹配的函数指针类型；
  - 定义一个该类型的变量；
  - 把一个函数的指针赋给这个变量。
- 使用

```
int MaxVal = MaxFunPtr(3, 4);
```

使用指针时的注意事项

- 可以定义指针类型的变量，但使用该指针变量前，必须初始化，否则不知道该指针变量指向何处，造成结果混乱，甚于造成访问违例(Exception)。
- 空指针用NULL表示。指针变量的值为空(NULL)，实际上是0。
- 指针变量的值存放的是地址，可以进行+,-,++,--运算。
- 指针分为短指针(32位)和长指针(64位)。
- 指针变量可以作为函数的参数进行传递。Windows API大量使用回调函数，就是将函数作为参数的。
- 一个指针变量可指向另一个指针变量的地址,即指针的指针
- void指针类型：首先表明是一个指针，但具体未指明是那一种类型的指针。

含有指针成员的结构

- 既然指针是一种数据类型，当然在定义结构时可以使用指针类型的成员，包括函数指针。
  - 在学习C语言时，曾在讲结构和指针之后，讲了如何用指针处理链表？
  - 在C++语言中，是如何实现类的？

第二节 数据结构的基本概念

- 典型数据结构的例子
- 数据结构的基本概念
- 数据结构的表示
- 数据操作

数据结构的例 – 之1

- 书目自动检索系统

线性表

书目卡片

登 录 号:
分 类 号:
书　　名:
作　　者:
出版单位:
出版时间:

书目文件

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
……	……	……	……

按书名

高等数学	001,003
理论力学	002
线性代数	004
……	……

按作者

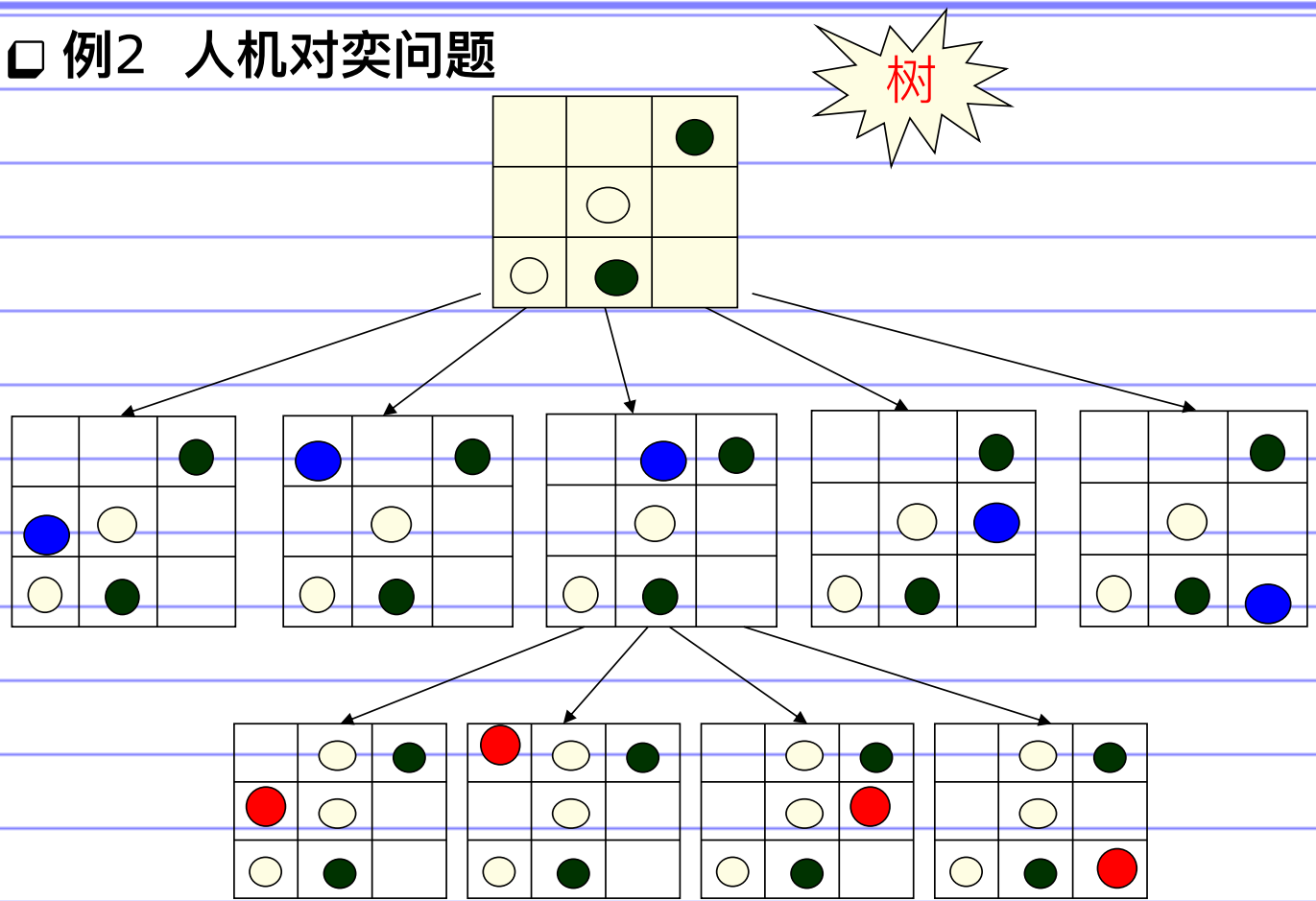
樊映川	001
罗远祥	002
华罗庚	003
栾汝书	004
……	……

按分类号

S	S01	001,003
	S02	004
L	L01	002
…	……	……

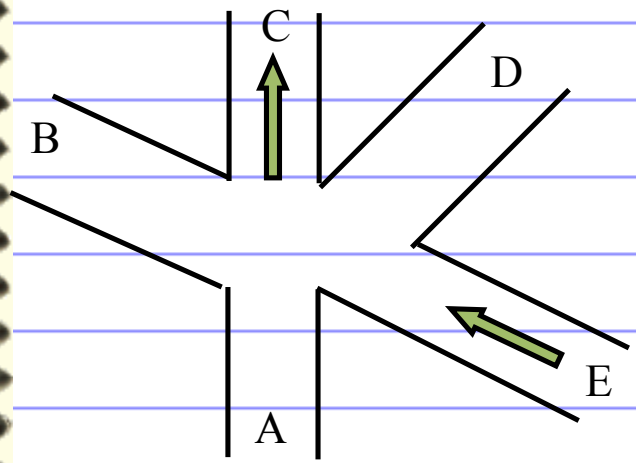
数据结构的例 – 之2

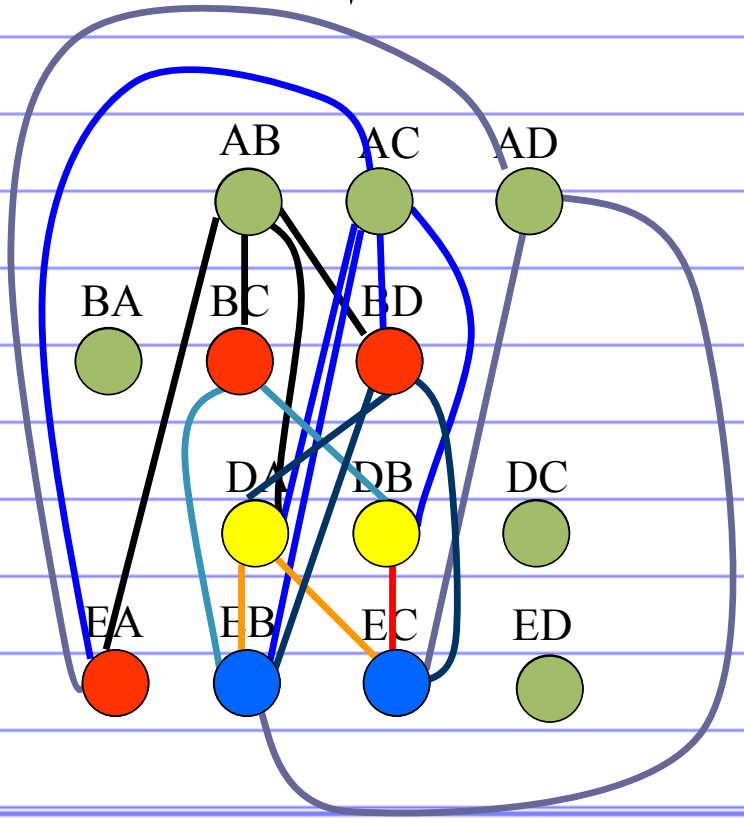
- 例2　人机对奕问题



数据结构的例 – 之3

- 多叉路口交通灯管理问题





数据结构的例－小结

- 这些例子中，描述这类问题的数学模型不再是数学方程
- 数据结构：研究程序设计中计算机通用的操作对象、以及它们之间的关系和操作的科学
- 数据结构与算法

基本概念

- 数据(data)—所有能输入到计算机中并被计算机处理的事物的符号的总和
- 数据元素(data element)—数据的基本单位，也称节点(node)或记录(record)
- 数据项(data item)—有独立含义的数据最小单位
- 数据对象(data object)—性质相同的数据元素集合，是数据的一个子集。
- 数据结构(data structure)—数据元素和数据元素关系的集合
- 数据结构的形式定义

  Data Structure = (D, S)

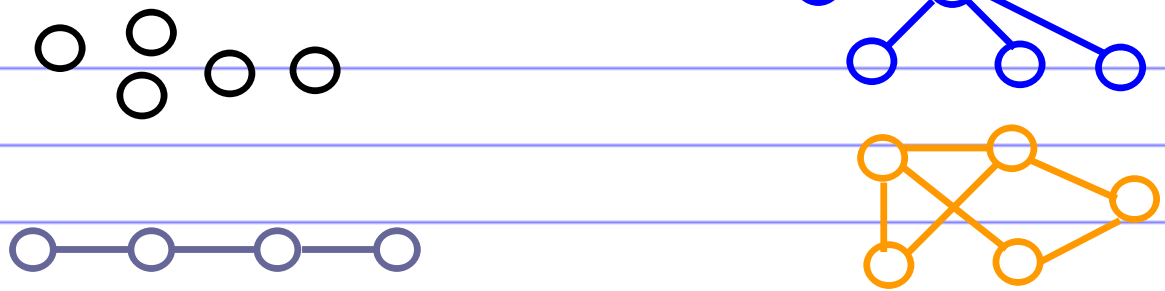
  其中D是数据元素的有限集，S是D上关系的有限集

数据结构

- 根据数据元素间关系的基本特性，有四种基本数据结构
  - 集合—也称散列结构

    数据元素间除“同属于一个集合”外，无其它关系
  - 线性结构—1对1，如线性表、栈、队列
  - 树形结构—1对多
  - 图状结构—多对多

  } 非线性结构

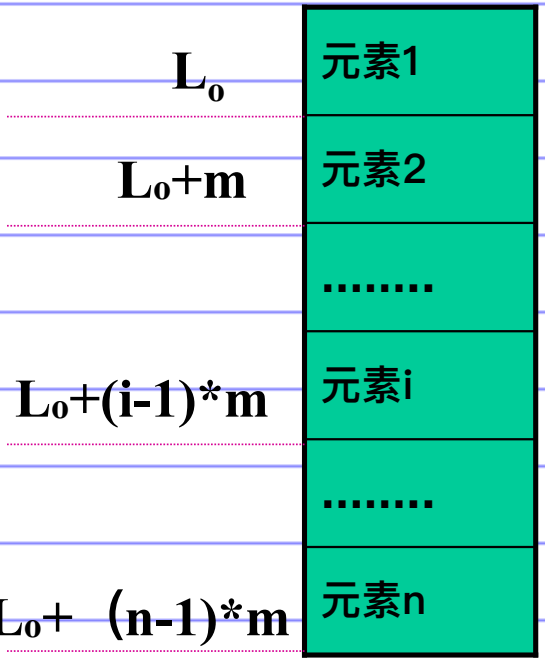


数据结构的表示

- 逻辑结构：用数学模型描述数据元素之间的逻辑关系
- 物理(存储)结构：在计算机内数据元素的存储映像
  - 顺序存储结构—借助元素在存储器中的相对位置(顺序映像)　来表示数据元素间的逻辑关系
  - 链式存储结构—借助指示元素存储地址的指针(非顺序映像)　来表示数据元素间的逻辑关系
- 数据存储结构的表示：并非直接用机器内存表示。
  - 高级语言—虚拟机—虚拟存储器
- 数据的逻辑结构与存储结构密切相关
  - 算法设计 → 逻辑结构，应该与语言无关的
  - 算法实现 → 存储结构，与语言相关

顺序存储

- 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



$$Loc(元素i)=Lo+(i-1)*m$$

链式存储结构

- 借助指示数据元素存储地址的指针表示数据元素之间的逻辑关系。

存储地址	存储内容	指针
1345	元素1	1400
1346	元素4	∧
…….	……..	…….
1400	元素2	1536
…….	……..	…….
1536	元素3	1346

数据类型

- 数据类型：Data Type — 高级语言中指数据的取值范围及其上可进行的操作的总称。

C语言提供int, char, float, double等基本数据类型；数组、结构体、共用体、枚举等构造数据类型；
还有指针、空(void)类型等。
用户也可用typedef 定义新的数据类型。

这些类型可以分为原子类型和结构类型。
原子类型是不可分的数据类型。
结构类型是可分为原子类型的数据类型。

那些是原子类型，那些是结构类型?

抽象数据类型

- 抽象数据类型Abstract Data Type — 是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义取决于它的一组逻辑特性，而与其在计算机内部的表示和实现无关。
- 一个含抽象数据类型的程序模块通常包含定义、表示和实现三个部分。
- 按抽象数据类型的数据取值的特性将其分为三种
  - 原子类型atomic data type
  - 固定聚结类型fixed-aggregate data type
  - 可变聚结类型variable-aggregate data type
- 多形数据类型polymorphic data type — 是指其值的成分不确定的数据类型。

抽象数据类型

抽象数据类型可用（D，S，P）三元组表示

其中，D是数据对象，S是D上的关系集，P是对D的基本操作集。

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

抽象数据类型

数据对象、数据关系用伪码描述；基本操作定义格式为

基本操作名（参数表）

初始条件：〈初始条件描述〉

操作结果：〈操作结果描述〉

- 基本操作有两种参数：赋值参数只为操作提供输入值；引用参数以&打头， 除可提供输入值外，还将返回操作结果。
- “初始条件”描述了操作执行之前数据结构和参数应满足的条件，若不满足，则操作失败，并返回相应出错信息。
- “操作结果”说明了操作正常完成之后，数据结构的变化状况和应返回的结果。若初始条件为空，则省略之。

数据操作

- 定义在数上的操作是没有限制的，因需而定。
- 通常的操作有，在数据结构中 -
  - 插入 — 指定位置上增加指定的数据元素
  - 删除 — 删除指定位置上或指定的数据元素
  - 更新 — 改变指定位置或指定的数据元素＝删除＋插入
  - 查找 — 寻找满足特定条件的数据元素或位置
  - 排序 — 按给定规则重建数据元素的逻辑顺序关系
- 算法 — 实现给定数据结构的操作所用的方法

第三节 算法分析

- 算法(algorithm)—解决某一特定问题的具体步骤和方法的描述。是指令的有限序列。
- 算法特性—
  - 有穷性 — 一个算法必须在执行有限步骤之后结束
  - 确定性 — 算法的每一步必须是确切定义的，不能产生二义性
  - 可行性 — 算法是能行的，通过有限次运行实现，能够得到确定的结果
  - 输　出 — 一个算法有零个或多个输出
  - 输　入 — 一个算法有零个或多个输入

算法的描述

- 算法的描述方法：
  - 用自然语言描述
  - 用流程图描述
  - 用伪代码描述
  - 用计算机高级语言描述。
- 本书采用C语言描述算法。

算法的评价

- 算法的评价 — 衡量算法优劣的标准,也是**算法设计的要求**
  - **正确性**(correctness):满足具体问题求解要求
  - **健壮性**(robustness):在非法数据输入时仍能处理
  - **可读性**(readability):易于阅读理解、调试和修改
  - **资源需求与效率**: 低需求、高效率。一个算法为完成求解特定问题，需要使用一些资源，其中最主要的CPU和内存。通常又称为**算法的复杂性**，分为时间复杂性(对CPU运算时间的需求)和空间复杂性(对内存资源的需求)。

算法复杂性

- 一个算法实现需要资源。特别是CPU和内存资源。通常又称为**算法的复杂性**，分为时间复杂性(对CPU运算时间的需求)和空间复杂性(对内存资源的需求)。
- 时间和空间是往往是一对矛盾。
  - 例如：排序问题
- 详细参考：Sartaj Sahni《数据结构、算法与应用 - C++语言描述》第二章：程序性能

算法效率

- 算法的时间复杂性即算法效率——通常以用该算法编制的程序在计算机上执行所消耗的时间来度量。
- 事后统计—利用计算机内记时功能,不同算法的程序可以用一组或多组相同的统计数据区分。
- 事前分析估计——一个高级语言程序在计算机上运行所消耗的时间取决于

①依据的算法选用何种策略
②问题的规模
③程序语言
④编译程序产生机器代码质量
⑤机器执行指令速度

算法效率

- 然而，同一个算法用不同的语言、不同的编译程序、在不同的计算机上运行，效率均不同，所以使用绝对时间单位衡量算法效率不合适
- 因此，我们通常用事先估计方法，选择算法所需的基本操作次数来衡量算法的时间复杂性。例如：比较次数、移动数据单元的次数等。
- 时间复杂度：基本操作重复执行的次数的阶数

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

算法的空间复杂性

- 空间复杂度：

  $s(n)=O(f(n))$

  n为问题的规模(或大小)。
- 空间复杂性通常考虑排除装载问题本身所必需的数据空间和算法所必需的空间外，额外所需的空间。
- 若额外空间相对于输入数据量来说是常数，则称此算法为原地工作。

算法复杂性分析的例

- 例1：NXN矩阵相乘

```
for(i=1;i<=n;i++)            //执行n次循环
  for(j=1;j<=n;j++) {        //执行n次循环
      c[i][j]=0;
      for(k=1;k<=n;k++)   //执行n次循环
        c[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
```

$$f(n)=n^3$$

$$T(n)=O(n^3)$$