**数据结构基本概念专题讲座**

written by王保明

# 数据结构实用概念

## 疑惑

1、我学完了C语言，可是现在感觉还是写不出代码。

2、为什么会有各种各样的程序存在？

3、程序的本质是什么？

程序是为了具体问题而存在的

程序需要围绕问题的解决进行设计

同一个问题可以有多种解决方案

如何追求程序的“性价比”？

是否有可量化的方法判别程序的好坏？

## 数据结构起源

计算机从解决数值计算问题到解决生活中的问题

现实生活中的问题涉及不同个体间的复杂联系

需要在计算机程序中描述生活中个体间的联系

数据结构主要研究非数值计算程序问题中的操作对象以及它们之间的关系

不是研究复杂的算法

## 数据结构中的基本概念

数据 – 程序的操作对象，用于描述客观事物

数据的特点：

可以输入到计算机

可以被计算机程序处理

数据是一个抽象的概念，将其进行分类后得到程序设计语言中的类型。如：int，float，char等等

数据元素：组成数据的基本单位

数据项：一个数据元素由若干数据项组成

数据对象 – 性质相同的数据元素的集合

|  |
| --- |
| //王保明，提醒你，来自结构体课堂代码  //声明一个结构体类型  struct \_MyTeacher //一种数据类型  {  char name[32];  char tile[32];  int age;  char addr[128];  };  int main21()  {  struct \_MyTeacher t1; //数据元素  struct \_MyTeacher tArray[30]; //数据对象  memset(&t1, 0, sizeof(t1));  strcpy(t1.name, "name"); //数据项  strcpy(t1.addr, "addr"); //数据项  strcpy(t1.tile, "addr"); //数据项  t1.age = 1;  } |

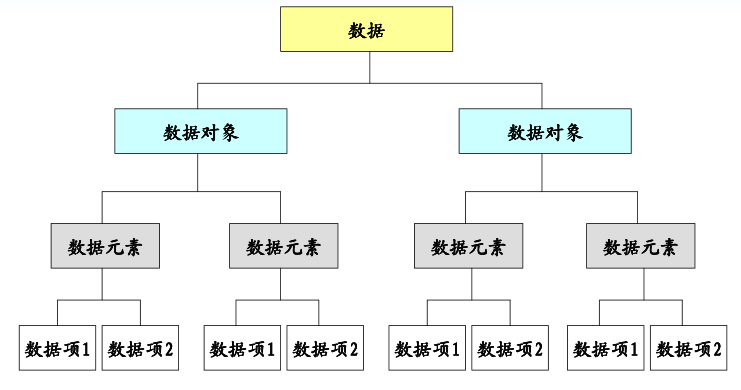
数据元素之间不是独立的，存在特定的关系，这些关系即结构

数据结构指数据对象中数据元素之间的关系

如：数组中各个元素之间存在固定的线性关系

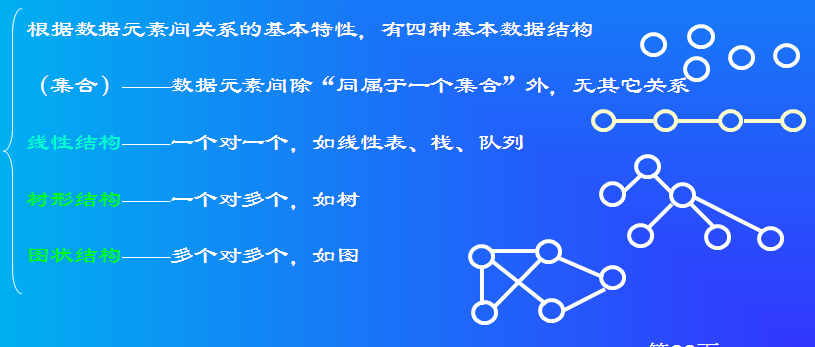
编写一个“好”的程序之前，必须分析待处理问题中各个对象的特性，以及对象之间的关系。

基本概念总结：

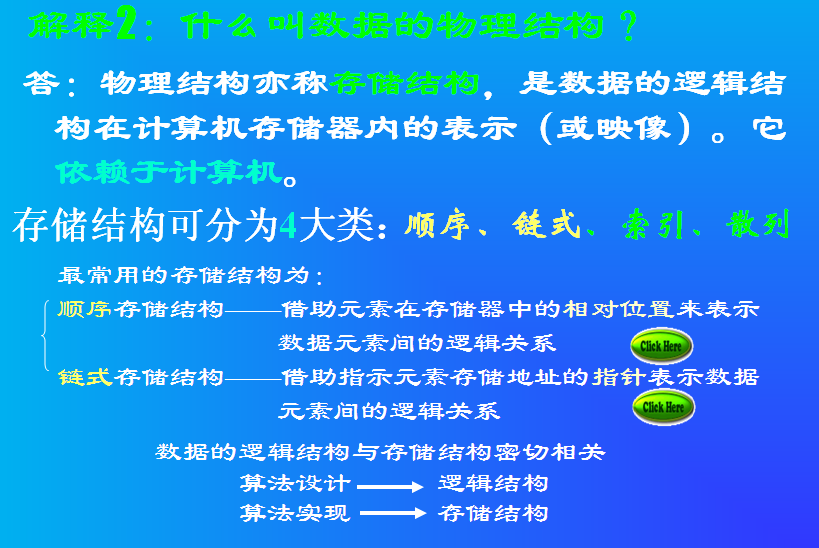


## 数据的逻辑结构

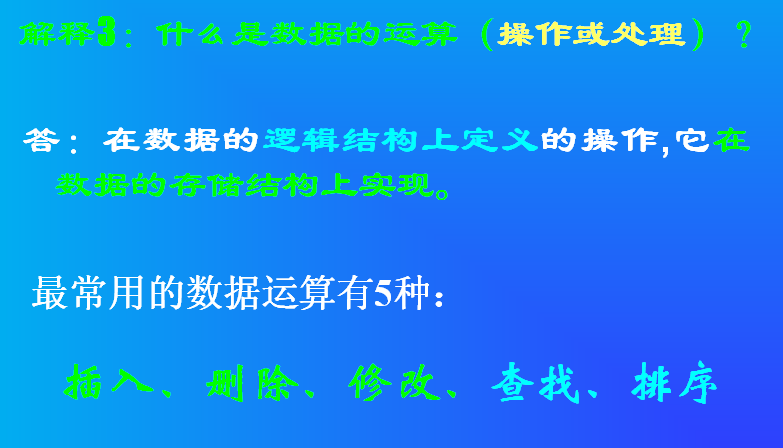
指数据元素之间的逻辑关系。即从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。逻辑结构可细分为4类：



## 数据的物理结构



## 数据的运算



# 算法

## 算法概念

算法是特定问题求解步骤的描述

在计算机中表现为指令的有限序列

算法是独立存在的一种解决问题的方法和思想。

对于算法而言，语言并不重要，重要的是思想。



## 算法和数据结构区别

数据结构只是静态的描述了数据元素之间的关系

高效的程序需要在数据结构的基础上设计和选择算法

===🡺程序=数据结构+算法

总结：

算法是为了解决实际问题而设计的

数据结构是算法需要处理的问题载体

数据结构与算法相辅相成

## 算法特性

输入

算法具有0个或多个输入

输出

算法至少有1个或多个输出

有穷性

算法在有限的步骤之后会自动结束而不会无限循环

确定性

算法中的每一步都有确定的含义，不会出现二义性

可行性

算法的每一步都是可行的

## 算法效率的度量

|  |
| --- |
| 1、事后统计法  比较不同算法对同一组输入数据的运行处理时间  缺陷  为了获得不同算法的运行时间必须编写相应程序  运行时间严重依赖硬件以及运行时的环境因素  算法的测试数据的选取相当困难  事后统计法虽然直观，但是实施困难且缺陷多 |
| 算法效率的度量  事前分析估算  依据统计的方法对算法效率进行估算  影响算法效率的主要因素  算法采用的策略和方法  问题的输入规模  编译器所产生的代码  计算机执行速度 |
| long sum1(int n)  {  long ret = 0;  int\* array = (int\*)malloc(n \* sizeof(int));  int i = 0;    for(i=0; i<n; i++)  {  array[i] = i + 1;  }    for(i=0; i<n; i++)  {  ret += array[i];  }    free(array);    return ret;  }  long sum2(int n)  {  long ret = 0;  int i = 0;    for(i=1; i<=n; i++)  {  ret += i;  }    return ret;  }  long sum3(int n)  {  long ret = 0;    if( n > 0 )  {  ret = (1 + n) \* n / 2;  }    return ret;  }  int main()  {  printf("%d\n", sum1(100));  printf("%d\n", sum2(100));  printf("%d\n", sum3(100));    return 0;  } |
| int func(int a[], int len)  {  int i = 0;  int j = 0;  int s = 0;    for(i=0; i<len; i++) n  {  for(j=0; j<len; j++) n  {  s += i\*j; //n\*n  }  }  return s;  }  //n\*n |
| 注意1：判断一个算法的效率时，往往只需要关注操作数量的最高次项，其它次要项和常数项可以忽略。  注意2：在没有特殊说明时，我们所分析的算法的时间复杂度都是指最坏时间复杂度。 |

2、大O表示法

|  |
| --- |
| 算法效率严重依赖于操作(Operation)数量  在判断时首先关注操作数量的最高次项  操作数量的估算可以作为时间复杂度的估算 |
| O(5) = O(1)  O(2n + 1) = O(2n) = O(n)  O(n2+ n + 1) = O(n2)  O(3n3+1) = O(3n3) = O(n3) |
| 常见时间复杂度    关系 |
|  |

3、算法的空间复杂度

|  |
| --- |
| 算法的空间复杂度通过计算算法的存储空间实现  S(n) = O(f(n))  其中，n为问题规模，f(n))为在问题规模为nn时所占用存储空间的函数  大O表示法同样适用于算法的空间复杂度  当算法执行时所需要的空间是常数时，空间复杂度为O(1) |
| 空间与时间的策略  多数情况下，算法执行时所用的时间更令人关注  如果有必要，可以通过增加空间复杂度来降低时间复杂度  同理，也可以通过增加时间复杂度来降低空间复杂度 |
| 练习1：分析sum1 sum2 sum3函数的空间复杂度  O(4n+12) O(8)=O(1) O(4)=O(1)  总结：实现算法时，需要分析具体问题，对执行时间和空间的要求。 |
|  |
| 练习2：时间换空间  /\*  问题：  在一个由自然数1-1000中某些数字所组成的数组中，每个数字可能出现零次或者多次。  设计一个算法，找出出现次数最多的数字。  \*/  方法1：  排序，然后找出出现次数最多的数字  方法2：  void search(int a[], int len)  {  int sp[1000] = {0};  int i = 0;  int max = 0;    for(i=0; i<len; i++)  {  int index = a[i] - 1;    sp[index]++;  }    for(i=0; i<1000; i++)  {  if( max < sp[i] )  {  max = sp[i];  }  }    for(i=0; i<1000; i++)  {  if( max == sp[i] )  {  printf("%d\n", i+1);  }  }  }  int main()  {  int array[] = {1, 1, 3, 4, 5, 6, 6, 6, 2, 3};    search(array, sizeof(array)/sizeof(\*array));    return 0;  } |