数据结构

Data Structures

E-mail: wuyl@njau.edu.cn

开设本课程的背景

数据结构是*计算机及相关专业重要的专业基础课*，它的前期课程主要有程序设计语言，学好这门课，可以加深对程序设计的理解，有助于进一步提高程序设计能力及解决问题的能力，并为计算机专业的后续课程（如操作系统、编译原理、数据库原理等）奠定良好的基础。

学习本课程的基本方法

◆首先*要认识到该课程的重要性*，有动力学好，同时相信自己能学好；

◆其次认真听讲，积极互动，独立完成作业，注意深入理解教材与多做练习、动手编程相结合；

◆最后*多交流*，不清楚不明白的*多问*。只要努力，相信大家都能学好并喜欢这门课程的。

教材

- 严蔚敏，吴伟民，《数据结构(C语言版)》，清华大学出版社　2018年

参考书：

- 严蔚敏，李冬梅，吴伟民 《数据结构 （C语言版）人民邮电出版社　2017年
- 李春葆，尹为民，蒋晶珏，喻丹丹，蒋林 《数据结构教程》，清华大学出版社　2017年

本书的结构

1. 绪论
- 2. 线性表
- 3. 栈和队列
4. 串
5. 数组和广义表
- 6. 树和二叉树
- 7. 图
8. ~~动态存储管理~~
- 9. 查找
- 10. 内部排序
11. ~~外部排序~~
12. ~~文件~~

第一章　绪 论

1. 数据结构的研究内容
2. 基本概念和术语
3. 抽象数据类型的表示和实现
4. 算法及其描述
5. 算法分析

1.1 数据结构的研究内容

用计算机解决一个具体的问题的一般步骤：

寻求数学模型的实质是分析问题，从中提取操作的对象，并找出这些操作对象之间的关系，然后用数学语言加以描述。有些问题的数学模型可以用具体的数学方程来表示，但更多的实际问题是无法用数学方程来表示的。

数据结构主要研究非数值计算问题，非数值计算问题无法用数学方程建立数学模型。

例1. 学生成绩管理系统

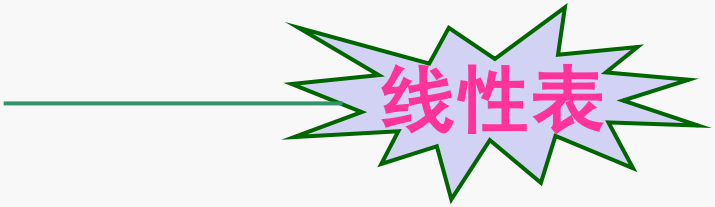
学校教务处使用计算机对学校的学生成绩情况做统一管理，学生的成绩信息如下表所示。每个学生的成绩按照不同的顺序号，依次存放在“学生成绩表”中，根据需要对这张表进行查找。每个学生的成绩记录按顺序号排列，形成了学生成绩信息记录的线性序列，呈一种线性关系。

表1 学生成绩表

学号	姓名	高等数学	英语	C语言
1601001	李林	88	87	80
1601002	王东	76	90	66
1601003	张阳	86	75	83
1601004	赵海	69	80	78

诸如此类的线性表结构还有图书馆的数目管理系统、库存管理系统等，在这类管理系统问题中，计算机处理的对象是各种表，表中元素之间存在着一对一的线性关系。

**管理系统



数学模型：线性表。

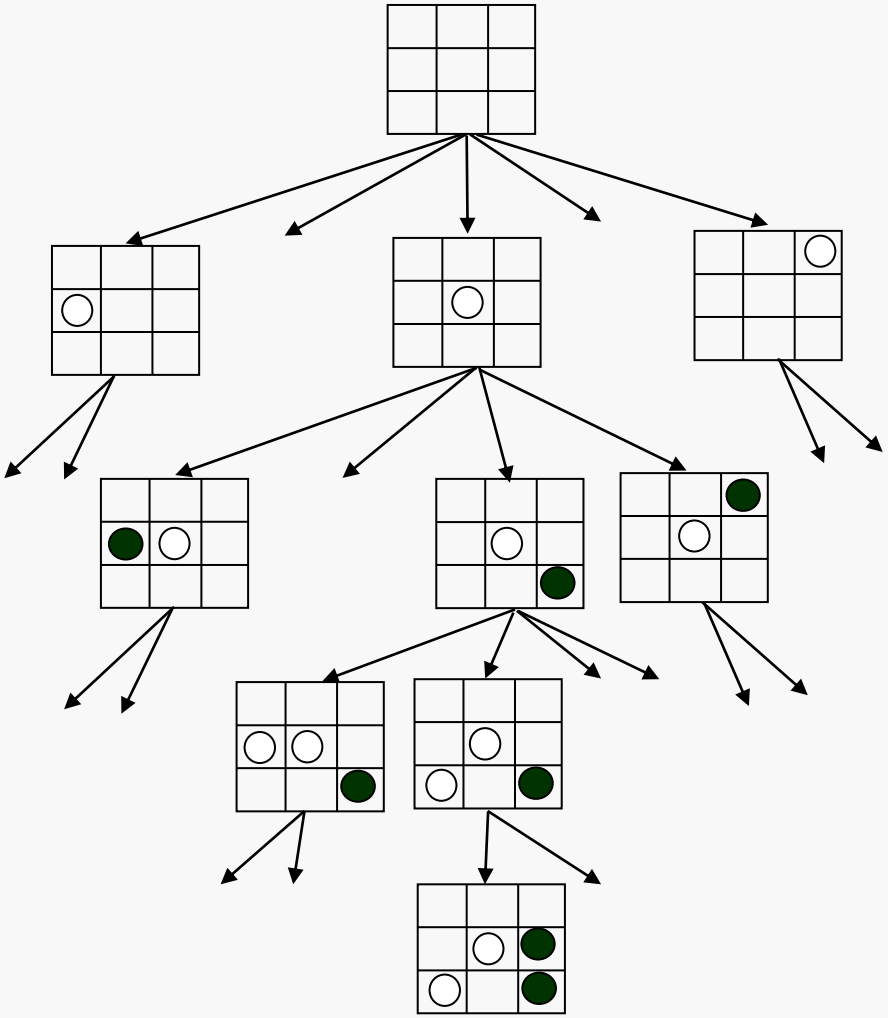
例2. 人机对弈问题

计算机之所以能和人对弈是因为已经将对弈的策略在计算机中存储好。

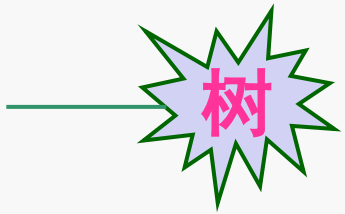
由于对弈的过程是在一定规则下随机进行的，所以，为使计算机能灵活对弈，就必须把对弈过程中所有可能发生的情况及相应的对策都加以考虑。

例如：井字棋

初始状态是一个空的棋盘格局。对弈开始后，每下一步棋，则构成一个新的棋盘格局，相对于上一个棋盘格局的可能选择可以有多种形式，因而整个对弈过程就如同“一棵倒长的树”。



人机对弈问题



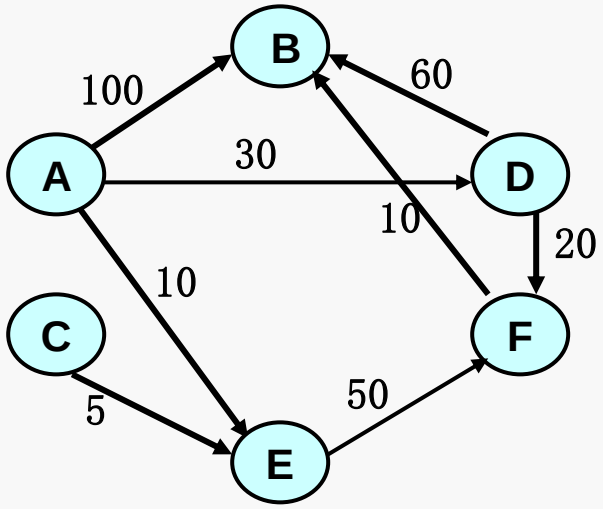
数学模型：如何用**树结构**表示棋盘和棋子等。

算法：是博弈的规则和策略。

例3. 最短路径问题

从城市A到城市B有多条线路，但每条线路的交通费不同，那么，如何选择一条线路，使得从城市A到城市B的交通费用最少？

解决方法：把这类问题抽象为图的最短路径问题。



图中的顶点：代表城市，

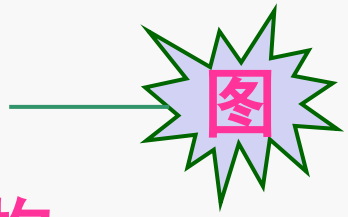
有向边：

代表两个城市之间的通路；

边上的权值：

代表两个城市之间的交通费。

最短路径问题



数学模型：图结构。

算法：是求解两点之间的最短路径。

从前面三个实例可以看出，非数值计算问题的数学模型不再是数学方程，而是诸如线性表、树和图的数据结构。

- **数据结构**是一门研究**非数值计算**程序设计中的操作对象以及这些对象之间的关系和操作的学科。

1.2 基本概念和术语

1.2.1 数据结构

一. 数据结构的定义

❖ 数据(Data)

指所有能输入到计算机中并被计算机处理的符号的集合。如整数、实数、字符串、图像、声音等。

❖ 数据元素(Data Element)

数据元素是数据的基本单位，它也可以再由不可分割的数据项组成。用于完整地描述一个对象，如前一节实例中的一名学生成绩记录，树中棋盘的一个格局(状态)，以及图中的一个顶点等。

❖ 数据项(Data Item)

数据项是组成数据的、有独立含义的、不可分割的最小单位。

表1 学生成绩表

学号	姓名	高等数学	英语	C语言
1601001	李林	88	87	80
1601002	王东	76	90	66
1601003	张阳	86	75	83
1601004	赵海	69	80	78

一个数据项

每个学生的成绩是一个数据元素

整个表记录的是学生成绩数据

❖ 结构(Structure)

数据元素相互之间存在着的某种关系。

❖ 数据结构(Data Structure)

是相互之间存在一种或多种特定关系的数据元素的集合。

即带结构的数据元素的集合。

数据结构通常包括以下三个方面：

- 数据的逻辑结构
- 数据的存储结构
- 数据的运算

二、 逻辑结构

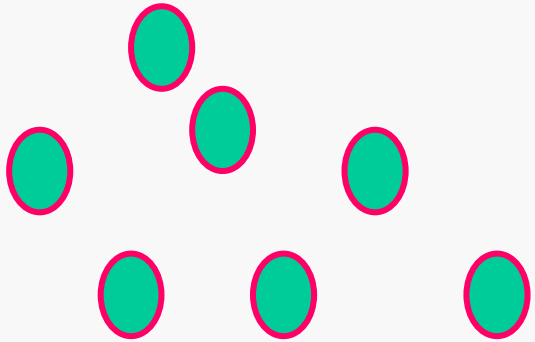
数据的逻辑结构是从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看做是从具体问题抽象出来的数学模型。

数据的逻辑结构有两个要素：
- 数据元素
- 关系

数据的逻辑结构可归结为以下四类：

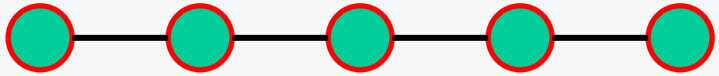
(1) 集合

结构中的数据元素之间除了“同属于一个集合”的关系外，没其它关系。



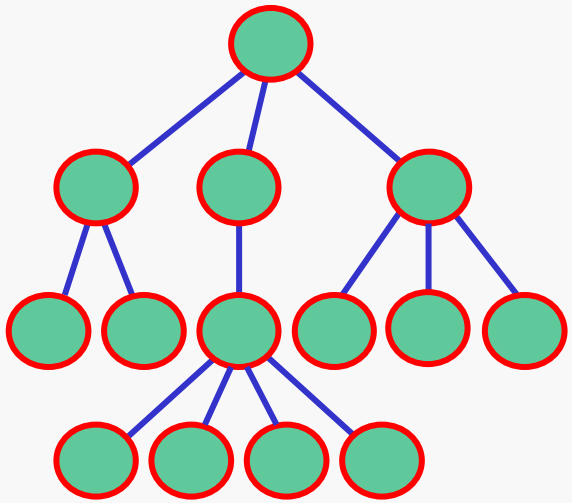
(2) 线性结构

结构中的数据元素之间存在一对一的关系。



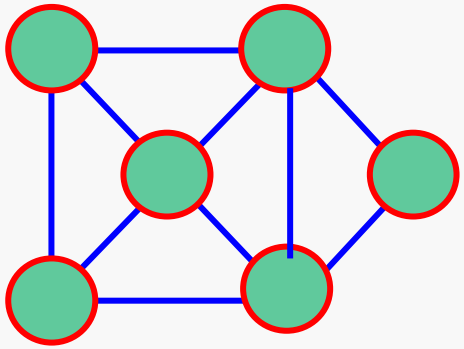
(3) 树形结构

结构中的数据元素之间存在一个对多个的关系。



(4) 图形结构（网状结构）

结构中的数据元素之间存在多个对多个的关系。



3. 存储结构

数据的逻辑结构在计算机中的存储表示称为数据的存储结构。

把数据对象存储到计算机时，通常要求既要存储每一个数据元素，又要存储数据元素之间的逻辑关系，数据元素在计算机内用一个结点来表示。

数据元素在计算机中有两种基本的存储结构:

- 顺序存储结构
- 链式存储结构

(1) 顺序存储结构

顺序存储结构是采用一组连续的存储单元存放所有的数据元素。即所有数据元素在存储器中占有一整块存储空间，而且两个逻辑上相邻的元素在存储器中的存储位置也相邻。通常借助于程序设计语言的数组类型来描述。

学生成绩表

学号	姓名	高等数学	英语	C语言
1601001	李林	88	87	80
1601002	王东	76	90	66
1601003	张阳	86	75	83
1601004	赵海	69	80	78

对于“学生成绩表”，假定每个结点(学生记录)占用50个存储单元，数据从0号单元开始由低地址向高地址方向存储，对应的顺序存储结构如下表：

地址	学号	姓名	高等数学	英语	C语言
0	1601001	李林	88	87	80
50	1601002	王东	76	90	66
100	1601003	张阳	86	75	83
150	1601004	赵海	69	80	78

(2) 链式存储结构

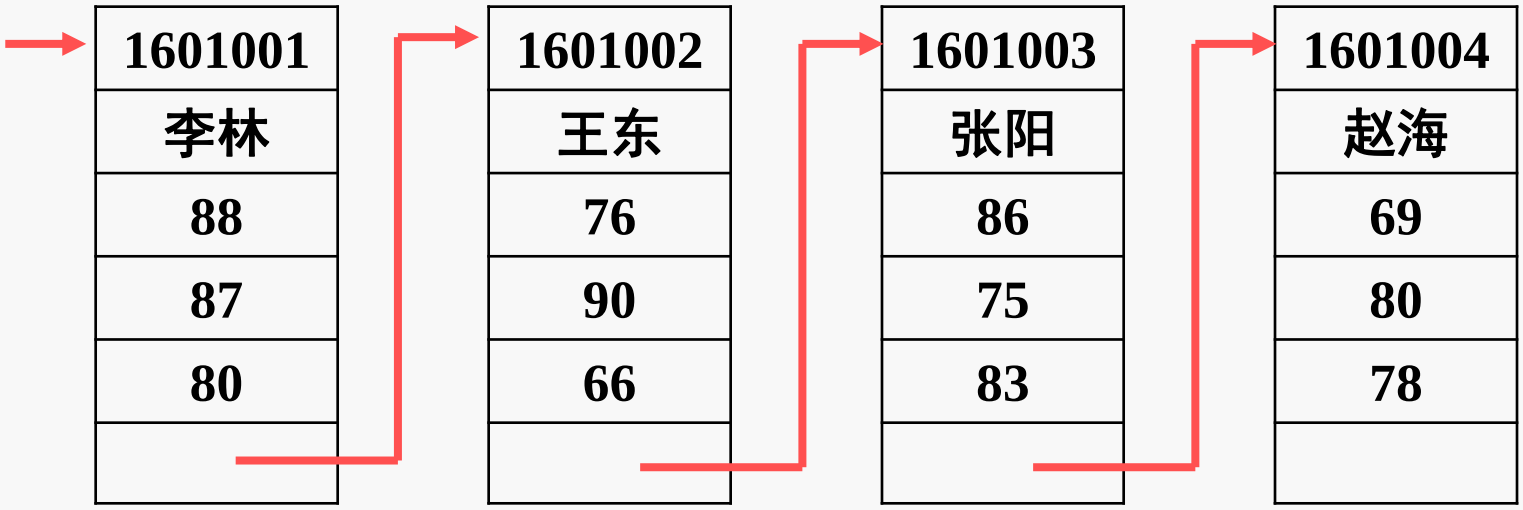
在链式存储结构中，每个逻辑元素用一个内存结点存储，每个结点是单独分配的，所有的结点地址不一定是连续的，所以无需占用一整块存储空间。但为了表示元素之间的逻辑关系，需要给每个结点附加指针域，用于存放相邻结点的存储地址。

链式存储结构通常借助于程序设计语言的指针类型来描述。

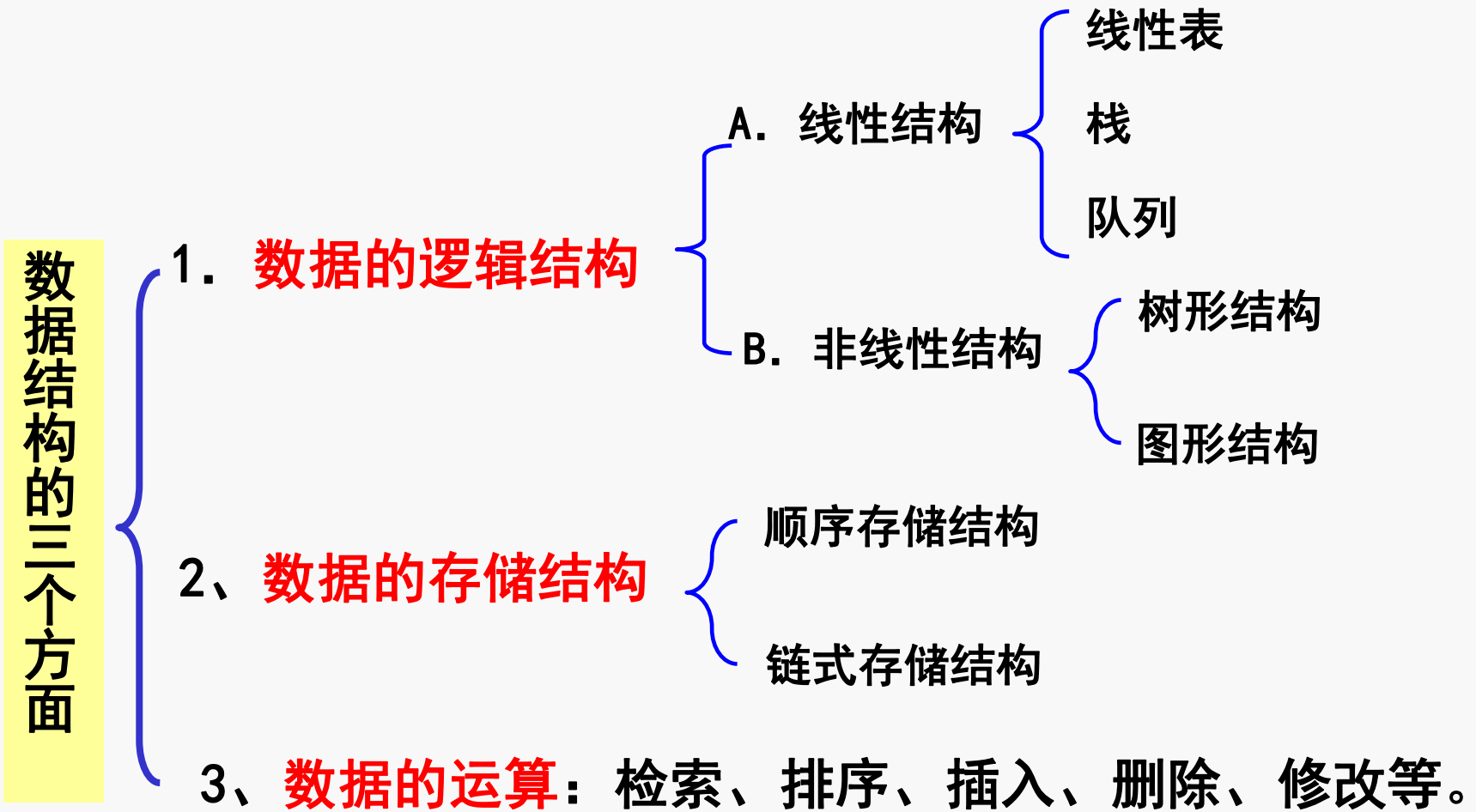
假定给“学生成绩表”的每个结点增加一个后继节点的指针字段，则可得到如下所示的链式存储结构。

地址	学号	姓名	高等数学	英语	C语言	后继结点的首地址
0	1601001	李林	88	87	80	150
50	1601003	张阳	86	75	83	100
100	1601004	赵海	69	80	78	^
150	1601002	王东	76	90	66	50

为了能更清楚地反映链式存储结构，可采用如下的更直观的图示来表示。



链式存储结构示意图


数据结构的三个方面
1．数据的逻辑结构
A．线性结构
线性表
栈
队列
B．非线性结构
树形结构
图形结构
2、数据的存储结构
顺序存储结构
链式存储结构
3、数据的运算：检索、排序、插入、删除、修改等。

数据结构课程学习的主要内容

1.如何对数据进行逻辑组织（逻辑结构）

- **一个具体问题的逻辑数据结构是什么?**

2.如何把数据存储到计算机中去（存储结构）

- 适宜选用什么样的存储结构?

3.相关的数据运算的定义与实现（算法）

- 采用什么样的操作实现算法效率更高?

“算法 +数据结构 =程序”　(Niklaus Wirth)

瑞士计算机科学家

1.2.2. 数据类型和抽象数据类型

❖ 数据类型(Data Type)

是一个值的集合和定义在这个值集上的一组操作的总称。

在程序设计语言中，每一个数据都属于某种数据类型。类型明显或隐含地规定了数据的取值范围、存储方式以及允许进行的运算。

例如C语言有一个int数据类型，它的取值范围为-32768~32767(16位系统），可用的运算有＋、－、＊、/和％等。

(1). C语言中的数据类型

C语言的数据类型按照取值的不同分为原子类型和结构类型。

原子类型：原子类型的值在逻辑上不可分解。如int型、float型、bool型等。

结构类型：是由若干数据类型组合而成，是可以再分解的。如结构体类型。

结构体类型是由一组称为结构体成员的数据项组成，一个结构体类型中所有成员的数据类型可以不相同。

例如，以下声明了一个Student结构体类型：

```
struct   Student    //struct是关键字，表示是结构体类型
{      int    num;
       int    age;
       char  name[8];
};
```

结构体中的各个成员

以下语句定义了结构体类型Student的一个结构体变量t并赋值：

```
struct   Student   t;

t.num=110;

t.age=19;

strcpy(t.name,“李飞”);
```

(2). 存储空间的分配

在程序设计中，定义变量就是使用内存空间，而存储空间的分配主要有两种方式。

- 静态存储空间分配方式

是指在程序编译期间分配固定的存储空间的方式。该存储分配方式通常是在变量定义时就分配存储单元并一直保持不变。

以定义一个数组为例，如下语句就采用了这种方式：

```
int  a[10];
```

一旦遇到该语句，系统就为a数组分配10个int整数空间。无论程序是否向a中放入元素，这一片空间都被占用。它也属于自动变量，当超出其作用范围时系统自动释放其内存空间。

◆ 动态存储空间分配方式

是指在程序运行期间根据需要动态地分配存储空间的方式。

C语言提供了一套机制可以在程序执行时动态分配存储空间：malloc( )/free( )函数对。即使用malloc( )函数为一个指针变量分配一片连续的空间，当不再需要时使用free( )函数释放其所指向的空间。

```
char  *p;        //定义一个字符指针变量
p = (char*) malloc(10*sizeof(char ));
……
free(p);
```

执行语句p= (char*) malloc(10*sizeof(char))时，将为其分配长度为10个字符的存储空间，并将该存储空间的首地址赋给指针变量p 。

但用malloc( )函数分配的存储空间不会被系统自动释放，所以当p所指的内存空间不再使用时，需要用free(p)语句释放p所指向的存储空间。

❖ 抽象数据类型（Abstract Data Type, ADT)

抽象数据类型实际上是数据类型的进一步抽象，即把数据类型和数据类型上的运算捆在一起，进行封装。

一个具体问题的抽象数据类型的定义一般包括数据对象、数据关系和基本操作三方面的内容。

抽象数据类型的定义格式如下：

```
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
```

其中，数据对象和数据关系的定义采用数学符号和自然语言描述。

基本操作的定义格式为：

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，若初始条件为空，则省略。

“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。

1.3 算法及其描述

1. 算法的定义及特性

❖ 算法

算法(Algorithm)是对特定问题求解方法和步骤的一种描述，它是指令的有限序列。

问题：按从小到大的顺序重新排列x，y，z三个数值的内容。

算法：

（1）输入x，y，z三个数值；

（2）从三个数值中挑选出最小者并换到x中；

（3）从y，z中挑选出较小者并换到y中；

（4）输出排序后的结果。

❖ **一个算法具有以下五个重要特性：**

（1） **有穷性** 一个算法对任意合法输入值，在执行有穷步后一定能结束，且每一步都必须在有穷时间内完成。

（2） **确定性** **对于每种情况下所应执行的操作，在算法中都有确切的规定，不会产生二义性，使算法的执行者或阅读者都能明确其含义及如何执行。**

（3）可行性　　算法中的所有操作都可以通过已经实现的基本操作运算执行有限次来实现。

（4）　输入性　　一个算法有零个或多个输入。

（5）　输出性　　一个算法必须有一个或多个输出。

```
int
void getsum(int num)
{
    int i,sum=0;
    for(i=1;i<=num;i++)
    sum+=i;   return sum;

}
```

添加这样一条语句即可

不难看出：这个函数的功能是求和，但是结果没有输出，没有什么实际意义。

常用的算法设计方法：

- **穷举法**

　　将问题空间中的所有求解对象一一列举出来，逐一分析、处理，并验证结果是否满足给定的条件。

- **回溯法**

　　将问题的候选解按某种顺序逐一枚举和检验，来寻找一个满足预定条件的解。当发现当前的候选解不可能是解时，就退回到上一步重新选择下一个候选解（回溯）。（如：八皇后问题、迷宫、深度优先遍历）

- **分治法和递归法**

遇到一个难以解决的大问题时，将其分割成一些规模较小的子问题，以便各个击破，分而治之，然后把各个子问题的解合并起来，得出整个问题的解。（如：快速排序、归并排序、二分查找等）

- **贪心法和动态规划法**

贪心法的基本思想是从问题的初始状态出发，依据某种贪心标准，通过若干次的贪心选择而得出局部最优解，寄希望于局部的最优解构建最终的全局最优解)。（如：Prim和Kruskal算法、Dijkstra算法）

动态规划是一种将问题实例分解为更小的、相似的子问题，并存储子问题的解而避免计算重复的子问题，以解决最优化问题的算法策略。（例：Floyd算法）

二. 算法设计的目标

算法设计应满足以下目标：

（1） **正确性**

在合理的数据输入下，能够在有限的运行时间内得到正确的结果。

（2） **可读性**

一个好的算法，首先应便于人们理解和相互交流，其次才是为计算机执行。可读性强的算法有助于人们对算法的理解，而难懂的算法易于隐藏错误，且难以调试和修改。

（3） **健壮性(Robustness)**

当输入的数据非法时，算法应当恰当地作出反应或进行相应处理，而不会产生莫名奇妙的输出结果。

（4） **高效率与低存储量需求**

省时又节省空间。

三. 算法描述

本课程采用介于伪码和C语言之间的类C语言作为描述工具。

类C语言精选了C语言的一个核心子集，同时作了若干扩充修改，增强了语言的描述功能。以下对其做简要说明。

类C语言和C语言或C++语言的主要区别如下：

- 可以省略变量的定义
- 某些操作可以用自然语言描述
- 允许采用C语言的控制结构，包括if语句、switch语句、for语句、 while语句和do… while语句。
- 允许采用C语言的所有运算和表达式。

(1) 预定义常量和类型

```
//函数结果状态代码
#define   TRUE    1
#define   FALSE   0
#define   OK    1
#define    ERROR    0
#define    INFEASIBLE   -1
#define   OVERFLOW   -2

 typedef   int   Status;
    //Status是函数的类型，其值是函数结果状态代码
```

(2). 数据的存储结构用C的类型定义（typedef）描述。

数据元素类型约定为ElemType，由用户在使用该数据类型时自行定义。

(3).本课程中的算法都用以下格式的函数描述：

```
函数类型    函数名（函数参数表）
  {
   //算法说明
        语句序列
  }//函数名
```

- 除了函数的参数需要说明类型外，算法中使用的辅助变量可以不作变量说明，必要时对其作用给予注释。
- 当函数返回值为函数结果状态代码时，函数的类型定义为Status；

- 为了便于算法描述，除了值调用方式外，增添了C++语言的引用调用的参数传递方式。在函数的形参表中，以&打头的参数即为引用参数。传递引用给函数与传递指针的效果是一样的，形参变化实参也变化，但引用使用起来比指针更加方便、高效。

例如：设计一个交换两个整数的算法。

编写相应的函数swap1(n, m)如下：

```
void swap1 ( int n, int m)
{   int temp;
   temp=n; n=m; m=temp;
 }
```

在该函数中的确实现了两个形参n和m的交换，但执行语句 swap1(a, b)时实参a和b的值并不会发生交换。

改正方法1：

采用指针的方式来回传形参的值，将上述函数改为如下：

```
void swap2 ( int *n, int *m)
 { int temp;
   temp=*n; *n=*m; *m=temp;
 }
```

调用该函数的语句为： `swap2( &a, &b);`

改正方法2：采用引用型形参。

在C++语言中提供了一种引用运算符“&”。引用常用于函数形参中，当采用引用型形参时，在函数调用时会将形参的改变回传给实参。利用引用运算符将swap1()修改如下：

```
swap3( int &n, int &m)  //形参前面的“&”符号是引用运算符
 {  int  temp;
    temp=n; n=m; m=temp;
 }
```

调用该函数的语句为： swap3(a, b);

(4) 内存的动态分配和释放

使用malloc函数和free函数动态分配和释放内存空间。

(5) 输入和输出语句

输入语句　　scanf ([格式串], 变量1, …, 变量n);

输出语句　　printf([格式串], 表达式1, …, 表达式n);

通常省略格式串。

1.4 算法分析

一、算法的时间复杂度

1. 问题规模和语句频度

不考虑计算机的软硬件等环境因素，影响算法时间代价的最主要因素是问题规模。问题规模是算法求解问题输入量的多少，是问题大小的本质表示，一般用整数n表示。

一条语句的重复执行次数称作语句频度。

设每条语句执行一次所需的时间均是单位时间，则一个算法的执行时间可用该算法中所有语句频度之和来度量。

例 求两个n阶矩阵的乘积算法

```
for ( i = 1; i<=n; ++i )              //频度为n+1
    for (j = 1; j<=n; ++j )           //频度为n*(n+1)
    {
        c[ i ][ j ] = 0 ;             //频度为n^2
        for (k = 1; k<= n; ++k ) //频度为n^2 *(n+1)
            c[ i ][ j ] += a[ i ][ k ] * b[ k ] [ j ]; //频度为n^3
     }
```

该算法中所有语句频度之和，是矩阵阶数n的函数，用f(n)表示，即上例算法的执行时间与f(n)成正比。

$f(n)=2n^3+3n^2+2n+1$

2. 算法的时间复杂度定义

对于稍复杂的算法，直接求出算法中所有语句的频度，通常比较困难。因此，为了客观地反映一个算法的执行时间，可以只用算法中的“基本语句” 的执行次数来度量算法的工作量。

所谓“基本语句”指的是算法中重复执行次数和算法的执行时间成正比的语句。

例如在求两个n阶矩阵的乘积算法中

语句c[i][j] += a[i][k] * b[k] [j]就是基本语句。

一般来说，算法中基本语句重复执行的次数是问题规模n的某个函数f(n)，算法的时间复杂度记作：

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

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

3. 算法的时间复杂度分析举例

分析算法的时间复杂度的基本方法为：找出所有语句中语句频度最大的那条语句作为基本语句，计算基本语句的频度得到问题规模n的某个函数f(n)，取其数量级用“O”表示即可。

例1. 常量阶示例

```
{x++；s=0;}
```

两条语句的频度均为1，算法的执行时间是一个与问题规模n无关的常数，所以算法的时间复杂度为T(n)=O(1)，称为常量阶。

例2. 线性阶示例

```
for(i=0;i<n;i++)  {x++;  s=0; }
```

循环体内两条基本语句的频度均为n，所以算法的时间复杂度为T(n)=O(n)，称为线性阶。

例3. 平方阶示例

```
(1)   x=0;y=0;
(2)   for (k=1;k<=n;i++)
(3)       x++;
(4)   for(i=0;i<n;i++)
(5)       for(j=1;j<=n;j++)
(6)           y++;
```

对循环语句只需考虑循环体中语句的执行次数，以上程序段中频度最大的语句是6，其频度为$f(n)=n^2$，所以该算法的时间复杂度为$T(n)=O(n^2)$，称为平方阶。

例4. 对数阶示例

```
for(i=0; i<n; i=i*2)  {x++;  s=0; }
```

设循环体内两条基本语句的频度均为f(n)，则有$2^{f(n)} \leq n$，$f(n) \leq \log_2 n$，所以算法的时间复杂度为$T(n)=O(\log_2 n)$，称为对数阶。

常见的时间复杂度按数量级递增排列依次为：

- O(1)----常数阶；
- $O(\log_2 n)$----对数阶；
- O(n)----线性阶；
- $O(n\log_2 n)$ ----线性对数阶；
- $O(n^2)$---- 平方阶；
- $O(n^3)$----立方阶；
- $O(2^n)$----指数阶。

一般情况下，随n的增大，T(n)增长较慢的算法，为较优的算法。

4. 平均时间复杂度

对于某些问题的算法，其基本语句的频度不仅仅与问题的规模相关，还依赖于其他因素。

如：在一维数组a中顺序查找某个值等于e的元素，并返回其所在的位置。

```
(1)   for(i=0; i<n; i++)
(2)         if(a[i]==e)  return  i+1;
(3)    return 0;
```

可以看出，此算法中语句2的频度不仅与问题的规模n有关，还与e的值有关。

最好情况，语句2的频度f(n)=1;

最坏情况，语句2的频度f(n)=n。

如果算法基本语句的频度与问题的输入数据有关，这时可考虑:

- 算法的平均时间复杂度
- 算法在最坏情况下的时间复杂度

然而在很多情况下，算法的平均时间复杂度难以确定。因此，通常只讨论算法在最坏情况下的时间复杂度。

二. 算法的空间复杂度

在本课程中，用执行算法所需的辅助空间的大小作为算法所需空间的度量。

设执行算法所需的辅助空间是问题规模n的某个函数f(n)，则算法空间复杂度记作：　S（n）= O(f(n))

对于一个算法，其时间复杂度和空间复杂度往往是相互影响的，当追求一个较好的时间复杂度时，可能会导致占用较多的存储空间，反之亦然。通常情况下，鉴于运算空间较为充足，我们一般以算法的时间复杂度作为算法优劣的衡量指标。

补充

1. 结构体概念

有时需要将不同类型的数据组合成一个有机的整体

如: 一个学生的信息

学号(num), 姓名(name), 性别(sex),

年龄(age), 成绩(score), 家庭地址(addr)等

num	name	sex	age	score	addr
10010	Li Ming	M	18	87.5	Beijing

结构体:

若干个数据类型不同（也可相同）的数据项的一个组合。

结构体是一种数据结构，它需要用户根据自己的需要、按某种规则定义，即定义结构体类型。

声明结构体类型的一般形式：

```
struct 结构体类型名
{
        成员列表;
};
```

```
struct student
{   int num;
    char name[20];
    int age;
    float score;
};
```

struct 是关键字，表示是结构体类型。
student 是结构体类型名。

结构体中的各个成员，
形式：类型符 成员名

2.定义结构体类型变量的方法

一．先定义结构体类型再定义变量名

```
struct student
    { int num;
      char name[20];
      char sex;
      int age;
      float score;
      char addr[30];
    } ;
 struct student student1，student2;
```

一般形式：

```
struct 结构体类型名
{
   成员表列
};
struct 结构体类型名 变量名表列;
```

二．在定义类型的同时定义变量

```
struct student
    {int num;
     char name[20];
     char sex;
     int age;
     float score;
     char addr[30];
     } student1，student2 ；
```

一般形式

```
struct 结构体类型名
{
   成员表列

} 变量名表列；
```

三．直接定义结构类型变量

```
struct
    { int num;
      char name[20];
      char sex;
      int age;
      float score;
      char addr[30];
    } student1，student2 ;
```

一般形式：

```
struct
{
  成员表列

} 变量名表列;
```

3. 结构体类型变量的引用

结构体成员运算符

引用形式：　结构体变量名.成员名

规则：

1. 不能将结构体变量作为一个整体进行赋值、输出，只能对结构体中的各个成员分别进行；但允许将一个结构体变量直接赋值给另一个具有相同结构的结构体变量。

如：**student1.num=10000;**

用 typedef 定义新类型名

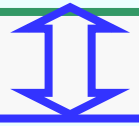
C 语言中除了系统定义的标准类型（如**int**、**char**、**long**、**double**等）和用户自己定义的结构和联合等类型之外，还可以用类型说明语句 **typedef** 定义新的类型名来代替已有的类型。**typedef** 语句的一般形式是：

typedef 已定义的类型 新的类型名;

例 typedef int INTEGER;

例 typedef float REAL;

例 INTEGER a,b,c;
REAL f1,f2;



int a,b,c;
float f1,f2;

例 一本书可以用有2个数据成员（数据域）的结构体变量存储。

```
typedef struct {
        int no;
        char title[40];
}BookType;
BookType book1;
```

结构体类型名

结构体变量名

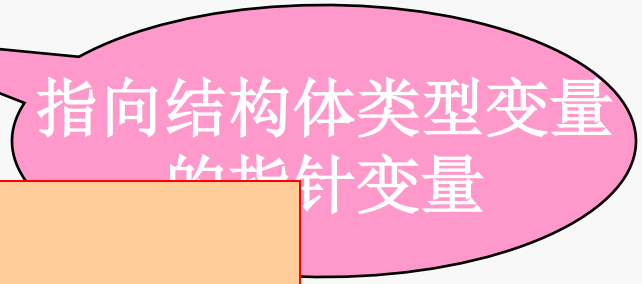
结构体变量的引用

结构体变量名.成员名

例 ：

```
    ...
    book1.no=1;
    scanf(“%s”,&book1.title);
    ...
```

例
```
typedef struct {
    int  no;
    char title;
}*BookPtType;
BookPtType  pbook;
```



指针所指变量的引用

指针变量->结构变量成员名

```
pbook->no=1;
scanf("%s",&pbook->title);
```