数据结构概述（教材选用严蔚敏、吴伟民，该书程序是伪算法 具体的程序是高一凡，西电的，大牛，只有 程序。还有一本书，台湾的黄国瑜自己写的 只有思路，程序是另外一个合作的清华的写 的，可惜很多错的。）

学完数据结构之后会对面向过程的函数有一个更深的了解 定义

我们如何把现实中大量而复杂的问题以特定的数据类型（单 个数据怎样存储？）和特定的存储结构（个体的关系）

保存到主存储器（内存）中，以及在此基础上为实现某个功能 （比如查找某个元素，删除某个元素，对所有元素进行排序） 而执行的相应操作，这个相应的操作也叫算法。（比如班里有 15个人，其信息量也许一个数组就搞定了，但是假如10000个， 怎么办？内存也许没有这么多连续的空间，所以我们改用链表， you see这就是与存储有关系。又比如，人事管理系统的信息存储， 因为存在着上下级的关系，所以数组和链表就无能为力了，

这时候我们用树，再比如我们做的是交通图，站和站之间肯定要连通，这 时候以上的存储方式又无能为力了，所以我们又有了图。图 就是每个结点都可以和其他结点产生联系。所以当我们要解决 问题时，首先要解决的是如何把这些问题转换成数据，先保存 到我们的主存中，）

数据结构=个体的存储+个体的关系的存储 算法=对存储数据的操作 算法

解题的方法和步骤

衡量算法的标准 1、时间复杂度

大概程序要执行的次数，而非执行的时间。

2、空间复杂度

算法执行过程中大概所占用的最大内存

3、难易程度（主要是应用方面看重）

4、健壮性（不能别人给一个非法的输入就挂掉）

数据结构的地位

数据结构是软件中最核心的课程

程序=数据的存储＋数据的操作＋可以被计算机执行的语言（已经提供）

（学完数据结构，想用一种语言去实现它，必须有指针，数据结构java版，就胡扯，变味，因为我们要讲链表，就是通过指针链在一起的。比如 在java中A aa = new A();本质上，aa是个地址） 预备知识 指针

指针的重要性：（内存是可以被CPU直接访问的，硬盘不行 主要靠地址总线，数据总线，控制总线。） 指针是C语言的灵魂 定义 地址

地址就是内存单元的编号 从0开始的非负整数

范围：0--FFFFFFFF[0-4G-1]（地址线是32位，刚好控制2的32次） 指针：

指针就是地址 地址就是指针

指针变量是存放内存单元地址的变量

指针的本质是一个操作受限的非负整数（不能加乘除，只能减） 分类：

1、基本类型的指针

2、指针和数组的关系

结构体（C++中用类也能实现） 为什么会出现结构体

为了表示一些复杂的数据，而普通的基本类型变量无法满足要求

什么叫结构体

结构体是用户根据实际需要自己定义的复合数据类型

如何使用结构体 两种方式：

struct Student st = {1000, "zhangsan", 20} struct Student \* pst = &st; 1.

st.sid 2.

pst->sid

pst所指向的结构体变量中的sid这个成员

注意事项：

结构体变量不能加减乘除，但可以相互赋值

普通结构体变量和结构体指针变量作为函数参数的传递

（病毒就是靠访问正在运行的那些程序所占用的内存。Java中规定局部 变量必须初始化，因为这些变量一开始都是垃圾值，但是属性不是必须 初始化的，因为已经默认初始化为0）

动态内存分配和释放（动态分配的内存一定要手动释放，否则造成内存 泄露。）

（java中A aa = new A();其实就是A \*p = (A \*)malloc(sizeof(A))）

模块一：线性结构【把所有的结点用一根直线穿起来】 连续存储【数组】 1、什么叫做数组

元素类型相同，大小相等（数组传参，只要传进去首地址和长度就行） 2、数组的优缺点： 优点：

存取速度快 缺点：

事先必须知道数组的长度 插入删除元素很慢 空间通常是有限制的 需要大块连续的内存块 插入删除元素的效率很低

离散存储【链表】（我们搞底层的开发，类似于SUN公司的类） 定义：

n个节点离散分配 彼此通过指针相连

每个节点只有一个前驱节点，每个节点只有一个后续节点 首节点没有前驱节点，尾节点没有后续节点。

专业术语：

首节点：

第一个有效节点（有效节点就是存放有效数据的节点） 尾节点：

最后一个有效节点 头节点：

头结点的数据类型和首节点的类型一样 没有存放有效数据，放在最前面的，是在 首节点之前的，主要是为了方便对链表 的操作。 头指针：（指向头）

指向头节点的指针变量 尾指针：

指向尾节点的指针

（头结点有可能很大，占的内存可能大，假设我想造一个函数 输出所有链表的值，那你如果不用头指针类型做形参，那由于 不同链表的头节点不一样大小，这样就没办法找出形参）

在学数据结构之前，我们确定数组的信息就是数组的长度和数组名，那么学完数据结构后就有三个了，长度，有效个数，数组名

确定一个链表需要几个参数：（或者说如果期望一个函数对链表进行操作 我们至少需要接收链表的那些信息？？？） 只需要一个参数:头指针，因为通过它我们可以推出 链表的所有信息。

（链表的程序最好一定要自己敲出来） 分类： 单链表

每个节点的指针域只能指向下一个节点 双链表：

每一个节点有两个指针域

循环链表

能通过任何一个节点找到其他所有的节点 非循环链表

（java中变成垃圾内存则会自动释放，但是C和C++则不会，所以要 手动释放，否则会引起内存泄露。delete等于free） 算法： 遍历 查找 清空 销毁 求长度 排序 删除节点 插入节点

算法：狭义的算法是与数据的存储方式密切相关 广义的算法是与数据的存储方式无关 泛型：（给你一种假象，只不过牛人从内部都弄好了）

利用某种技术（比如C++里面函数重载）达到的效果就是：不同的存储方式，执行的操作是一样的

算法的真正学法：很多算法你根本解决不了！！！！！！因为很多都属于

数学上的东西，所以我们把答案找出来，如果能看懂就 行，但是大部分人又看不懂，分三步，按照流程，语句， 试数。这个过程肯定会不断地出错，所以不断出错，不断 改错，这样反复敲很多次，才能有个提高。实在看不懂 就先背会。

链表的优缺点： 优点：

空间没有限制 插入删除元素很快 缺点：

存取速度很慢。 # include <stdio.h># include <malloc.h>void f(int k){

int i;

double \* q = (double \*)malloc(200);}

int main(void){

int i = 10;

int \* p = (int \*)malloc(100);

return 0;}

//f函数和main函数里面的i,q,p都是栈里面分配的，由操作系统分配的

栈和堆表示的是一种分配数据的方式，静态的和局部变量是以压栈和出栈的方式分配内存的

//malloc(200),malloc(100)是在堆里面分配的，由程序员自己手动分配的，动态的是以一种堆排序的方式分配内存的

线性结构的两种常见应用之一 栈(存储数据的结构) 定义

一种可以实现“先进后出” 的存储结构 栈类似于箱子 分类

静态栈 （类似于用数组实现） 动态栈 （类似于用链表实现）

算法（往里放，从里取） 出栈

压栈（参看Java中线程的例子，成产消费的例子）

用栈来实现

栈

的

生产 取出（消费）

左边生成东西，右边往出取（消费）生产和取出必须要平衡，不平衡的话生产就会满了，或者取出的是空的，取空了就是不能再取了，再取就是垃圾数据了 生产往底部一直往上放，取出就从顶部一直往下取

应用

函数调用（压栈和出栈是现实的） 中断

表达式求值（用两个栈，一个存放数字，一个存放符号） 内存分配 缓冲处理 迷宫

线性结构的两种常见应用之二 队列 定义：

一种可是实现“先进先出”的存储结构 分类：

链式队列：用链表实现

静态队列：用数组实现

静态对流通常都必须是循环队列，为了减少 内存浪费。

循环队列的讲解：

1、 静态队列为什么必须是循环队列

2、 循环队列需要几个参数来确定 及其含义 需要2个参数来确定 front rear

3、 循环队列各个参数的含义

2个参数不同场合不同的含义? 建议初学者先记住，然后慢慢体会

1）队列初始化

front和rear的值都是零 2）队列非空

front代表队列的第一个元素

rear代表了最后一个有效元素的下一个元素 3）队列空

front和rear的值相等，但是不一定是零 4、 循环队列入队伪算法讲解 两步完成：

1）将值存入r所代表的位置

2）将r后移，正确写法是rear = (rear+1)%数组长度 错误写法：rear=rear+1;

5、 循环队列出队伪算法讲解

front = (front+1) %数组长度

6、 如何判断循环队列是否为空 如果front与rear的值相等， 则队列一定为空

7、 如何判断循环队列是否已满 预备知识：

front的值和rear的值没有规律， 即可以大，小，等。

两种方式：

1、多增加一个表标识的参数

2、少用一个队列中的元素（才一个，不影响的） 通常使用第二种方法

如果r和f的值紧挨着，则队列已满 用C语言伪算法表示就是：

if( (r+1)%数组长度== f ) 已满 else 不满

队列算法：

入队 出队 队列的具体应用：

所有和时间有关的操作都有队列的影子。 （例如操作系统认为先进来的先处理）

专题：递归【这对你的编码能力是个质的飞跃，如果你想成为一个很厉害的 程序员，数据结构是必须要掌握的，因为计算机专业的本科生也达不到这水 平！计算机特别适合用递归的思想来解决问题，但是我们人类用递归的思想 来考虑问题就会感到十分困扰，这也是很多学过递归的人一直都搞不明白的 地方！那是不是递归可以随便写，当然不是，有些同学一用递归程序就死翘 翘了。递归的思想是软件思想的基本思想之一，在树和图论上面，几乎全是 用递归来实现的，最简单，像求阶乘这种没有明确执行次数的问题，都是用 递归来解决】 定义：

一个函数自己直接或间接调用自己(一个函数调用另外 一个函数和他调用自己是一模一样的，都是那三步， 只不过在人看来有点诡异。) 递归用栈实现

递归满足的三个条件：

1、递归必须得有一个明确的终止条件 2、该函数处理的数据规模必须在递减 3、这个转化必须是可解的。 递归的调用

1、 当在一个函数的运行期间调用另一个函数时，在运行被调用函数之前，系统需要完成三件事： A将所有的实际参数，返回地址等信息传递给被调函数保存 B为被调函数的局部变量（也包括形参）分配存储空间 C将控制转移到被调函数的入口

2、 从被调函数返回主调函数之前，系统也要完成三件事： A保存被调函数的返回结果 B释放被调函数所占的存储空间

C依照被调函数保存的返回地址将控制转移到调用函数

3、 当有多个函数相互调用时，按照“后调用先返回”的原则，上述函数之间信息传递和控制转移必须借助“栈”

来实现，即系统将整个程序运行时所需的数据空间安排在一个栈中，每当调用一个函数时，就在栈顶分配一个存储区，进行压栈操作，每当一个函数退出时，就释放它的存储区，就进行出栈操作，当前运行的函数永远都在栈顶位置

A函数调用A函数和A函数调用B函数在计算机看来是没有任何区别的，只不过用我们日常的思维方式理解比较怪异而已！

循环和递归：

理论上循环能解决的，肯定可以转化为递归，但是这个 过程是复杂的数学转化过程，递归能解决不一定能转化 为循环，我们初学者只要把经典的递归算法看懂就行， 至于有没有能力运用看个人。

递归：

易于理解(学完汉诺塔，树，图后就会发现递归很容易理解) 速度慢 存储空间大 循环

不易于理解 速度快 存储空间小 举例： 1.求阶乘 2.1+2+3+4+。。。+100的和 3.汉诺塔

【汉诺塔】这不是线性递归，这是非线性递归！ n=1 1 n=2 3 n=3 7 ......... .........

n=64 2的64次方减1【这是个天文数字，就算世界上最快的计算机 也解决不了，汉诺塔的负责度是2的n次方减1】问题很复杂，但真正解决 问题的编码只有三句。 4.走迷宫(CS的实现)

递归的运用：

树和森林就是以递归的方式定义的 树和图的很多算法都是以递归来实现的 很多数学公式就是以递归的方式定义的 斐波拉契序列

1 2 3 5 8 13 21 34。。。用递归思想写出代码

为何数据结构难学：因为计算机内存是线性一维的，而我们要处理的数据 都是比较复杂的，那么怎么把这么多复杂的数据保存在计算机中来保存本 身就是一个难题，而计算机在保存线性结构的时候比较好理解，尤其是数 组和链表只不过是连续和离散的问题，线性结构是我们学习的重点，因为 线性算法比较成熟，无论C++还是Java中都有相关的工具例如Arraylist.Linkedlist,但是在Java中没有树和图，因为非线性结构太复杂了，他的

操作远远大于线性结构的操作。即使SUN公司也没造出来。 小复习一下：^\_^

逻辑结构：（就是在你大脑里面能产生的，不考虑在计算机中存储） 线性（用一根直线穿） 在计算机中存储用： 数组 链表

栈和队列是一种特殊的线性结构，是具体应用。

（操作受限的线性结构，不受限的应该是在任何地方可以增删改查 可以用数组和链表实现。只要把链表学会，栈和队列都能搞定，数 组稍微复杂一些。） 非线性： 树 图 物理结构：

数组

链表

模块二：非线性结构(现在人类还没有造出一个容器，能把树和图 都装进去的，因为他们确实是太复杂了)（都要靠链表去实现） 树

树定义

专业定义：

1、有且只有一个称为根的节点

2、有若干个互不相交的子树，这些子树本身也是一棵树

通俗定义：

1、树是由节点和边组成

2、每个节点只有一个父节点但可以有多个子节点

3、但有一个节点例外，该节点没有根节点，此节点称为根节点

专业术语

节点 父节点 子节点 子孙 堂兄弟 深度：

从根节点到最底层节点的层数称之为深度 根节点是第一层 叶子节点；（叶子就不能劈叉了） 没有子节点的节点 非终端节点：

实际就是非叶子节点。

根节点既可以是叶子也可以是非叶子节点 度：

子节点的个数称为度。（一棵树看最大的） 树分类： 一般树

任意一个节点的子节点的个数都不受限制 二叉树（有序树）

任意一个节点的子节点的个数最多两个，且子节点 的位置不可更改。

分类：

一般二叉树 满二叉树

在不增加树的层数的前提下。无法再多 添加一个节点的二叉树就是满二叉树。 完全二叉树

如果只是删除了满二叉树最底层最右边的 连续若干个节点，这样形成的二叉树就是 完全二叉树。 森林

n个互不相交的树的集合

一般的二叉树要以数组的方式存储，要先转化成完全二叉树，因为如果你 只存有效节点（无论先序，中序，后序），则无法知道这个树的组成方式 是什么样子的。

树的存储（都是转化成二叉树来存储） 二叉树的存储

连续存储【完全二叉树】 优点：

查找某个节点的父节点和子节点（也包括判断有咩有）速度很快 缺点：

耗用内存空间过大

链式存储

一般树的存储 双亲表示法

求父节点方便 孩子表示法

求子节点方便 双亲孩子表示法

求父节点和子节点都很方便 二叉树表示法

把一个普通树转化成二叉树来存储 具体转换方法：

设法保证任意一个节点的

左指针域指向它的第一个孩子 有指针域指向它的下一个兄弟

只要能满足此条件，就可以把一个普通树转化成二叉树 一个普通树转化成的二叉树一定没有右子树

森林的存储

先把森林转化为二叉树，再存储二叉树，具体方式为：根节点 之间可以当成是兄弟来看待

二叉树操作 遍历

先序遍历【先访问根节点】 先访问根节点 再先序访问左子树 再先序访问右子树

中序遍历【中间访问根节点】 中序遍历左子树 再访问根节点 再中序遍历右子树

后序遍历【最后访问根节点】 先后序遍历左子树 再后序遍历右子树 再访问根节点

已知两种遍历序列求原始二叉树

通过先序和中序 或者 中序和后续我们可以 还原出原始的二叉树

但是通过先序和后续是无法还原出原始的二叉树的

换种说法：

只有通过先序和中序， 或通过中序和后序 我们才可以唯一的确定一个二叉树 应用

树是数据库中数据组织的一种重要形式（例如图书馆 的图书分类一层一层往下分。）

操作系统子父进程的关系本身就是一棵树 面向对象语言中类的继承关系本身就是一棵树 赫夫曼树（树的一个特例） 图

模块三：查找和排序 折半查找

排序：

冒泡 插入 选择 快速排序 归并排序

排序和查找的关系 排序是查找的前提 排序是重点

Java中容器和数据结构相关知识 Iterator接口 Map

哈希表（与Java关系比较大）

再次讨论什么是数据结构：

数据结构研究是数据结构的存储和数据的操作的一门学问 数据的存储分为两部分： 个体的存储 个体关系的存储

从某个角度而言，数据的存储最核心的就是个体关系 的存储，个体的存储可以忽略不计。

再次讨论到底什么是泛型：

同一种逻辑结构，无论该逻辑结构物理存储是什么样子的 我们都可以对它执行相同的操作（例如都是线性结构或者 用数组实现的树和用链表实现的树。利用重载技术。）