天勤计算机考研高分笔记系列

计算机考研大纲起草者

殷人昆

鼎力推荐

2017BAN SHUJU JIEGOU GAOFEN BIJI

2017版

数据结构高分笔记

率辉 主编

第5版

重要更新

- ▲ 根据读者反馈，增加或优化了一些知识点讲解，如广义表、败者树等
- ▲ 为本书建立互动更新计划
- ▲ 为书中相关知识点配备了对应的编程题库，供考生在线练习
- ▲ 进一步优化了知识点的讲解方式，使其更容易理解
- ▲ 修正了书中代码出现的一些bug
- ▲ 上机试题的网址http://sjjggfbj.codeup.cn/

本书互动更新平台：

wechat ID：shuaihui_ds

机械工业出版社
CHINA MACHINE PRESS

天勤计算机考研高分笔记系列

2017 版数据结构高分笔记

第 5 版

率 辉 主编

机 械 工 业 出 版 社

本书针对近几年全国计算机学科专业综合考试大纲的“数据结构”部分进行了深入解读，以一种独创的方式对考试大纲知识点进行了讲解，即从考生的视角剖析知识难点；以通俗易懂的语言取代晦涩难懂的专业术语；以成功考生的亲身经历指引复习方向；以风趣幽默的笔触缓解考研压力。读者对书中的知识点讲解有任何疑问都可与作者进行在线互动，为考生解决复习中的疑难点，提高考生的复习效率。

根据计算机专业研究生入学考试形势的变化（逐渐实行非统考），书中对大量非统考知识点进行了讲解，使本书所包含的知识点除覆盖统考大纲的所有内容外，还包括了各自主命题高校所要求的知识点。

本书可作为参加计算机专业研究生入学考试的复习指导用书（包括统考和非统考），也可作为全国各大高校计算机专业或非计算机专业的学生学习“数据结构”课程的辅导用书。

（编辑邮箱：jinacmp@163.com）

图书在版编目（CIP）数据

2017版数据结构高分笔记 / 率辉主编. —5版. —北京：机械工业出版社，2016.2

（天勤计算机考研高分笔记系列）

ISBN 978-7-111-53031-2

Ⅰ. ①2… Ⅱ. ①率… Ⅲ. ①数据结构－研究生－入学考试－自学参考资料 Ⅳ. ①TP311.12

中国版本图书馆CIP数据核字（2016）第035183号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）

策划编辑：吉 玲 责任编辑：吉 玲 刘丽敏 责任校对：薛 娜

封面设计：鞠 杨 责任印制：李 洋

北京振兴源印务有限公司印刷

2016年3月第5版 • 第1次印刷

184mm×260mm • 20.5印张 • 644千字

标准书号：ISBN 978-7-111-53031-2

定价：49.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务 网络服务

服务咨询热线：010-88361066 机 工 官 网：www.cmpbook.com

读者购书热线：010-68326294 机 工 官 博：weibo.com/cmp1952

010-88379203 金 书 网：www.golden-book.com

封面无防伪标均为盗版 教育服务网：www.cmpedu.com

前　言

高分笔记系列书籍简介

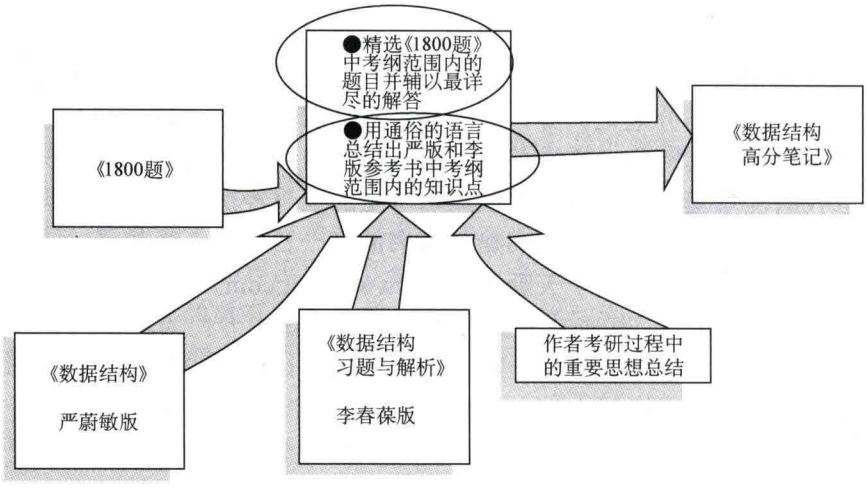
高分笔记系列书籍包括《数据结构高分笔记》《组成原理高分笔记》《操作系统高分笔记》以及《计算机网络高分笔记》等，是一套针对计算机考研的辅导书。它 2010 年夏天诞生于一群考生之手，其写作风格特色突出表现为：以学生的视角剖析知识难点；以通俗易懂的语言取代晦涩难懂的专业术语；以成功考生的亲身经历指引复习方向；以风趣幽默的笔触缓解考研压力。高分笔记系列书籍从成书的那一日起就不断接受读者的反馈意见，为了更好地与读者沟通，遂成立了天勤论坛（www.csbiji.com）。论坛名取自古训“天道酬勤”，以明示考研之路艰辛，其成功非勤而无以致。论坛中专门为高分笔记系列书籍开设了答疑专区，以弥补书中讲解的百密一疏；勘误专区，让读者成为作者的一部分，实时发现书中的不足以纠正；读者回馈专区，保留最真实的留言，用读者自己的声音向新人展示高分笔记的特色。相信高分笔记系列书籍带给考生的将是更高效、更明确、更轻松、更愉快的复习过程。

数据结构高分笔记简介

众所周知，在计算机统考的四门专业课中，最难拿高分的就是数据结构。但是这门课本身的难度并不是考生最大的障碍，真正的障碍在于考生不能独自把握复习的方向和考试范围。也许有学生要问，我们不是有大纲吗？照着大纲去复习不就可以了吗？表面上看是这样，但是当你真正开始复习的时候你就会发现，其实大纲只给了考生一个大致范围，有很多地方是模糊的，这些模糊的地方可能就是你纠结的地方。比如大纲里对于栈和队列的考查中有这么一条：“栈和队列的应用”。这个知识点就说得很模糊，因为只要涉及栈和队列的地方，都是其应用的范畴，这时考生该怎么办呢？于是把所有的希望寄托于参考书，希望参考书能帮助我们理解大纲的意图。参考书分两种：一是课本，二是与课本配套的辅导书。对于课本，考生用得最多的就是严蔚敏编写的《数据结构》，这里我也推荐大家把这本书选作考研辅导教材。因为这本书的内容非常丰富，如果能把这本书中考试大纲要求的章节理解透彻，参加考研就没有任何问题，但是这个过程是漫长的，除非本科阶段就学得非常好。计算机统考后，专业课四门加上公共课三门，一共是七门，绝大多数考生复习的时间一般也就六个月，而数据结构的复习需要占用多少时间，这点大家都很清楚。要在这么短的时间内掌握严蔚敏编写的《数据结构》中考纲要求的知识点，基本上是不可能的，这就需要一本辅导书来依照大纲从课本中总结出考纲要求的知识点，才能使得考生在短时间内达到研究生考试的要求。市面上的参考书有两种：一种是四合一的辅导书，另一种是分册的。比如网上流行的《1800 题》及其第 2 版，此书中题目极多，并且有很多老式的考

研题，有些算法设计题的答案是用 Pascal 语言写的。这本书中的题目一般考生全做基本上是不可能的，挑选着做又会把时间浪费在选题上。不可否认，这本书确实是一本非常好的题库，但是考生直接拿来用作考研辅导书却不太合适。这种情况下，就需要有一本优质的完全针对新大纲的辅导书出现，这就是高分笔记产生的原因。

接下来详细介绍一下这本辅导书的写作过程，请看下图：



《数据结构高分笔记》的由来

图中所涉及的书都是大家很熟悉的。当年这些书我都买了，花了很大心思才从中找出在考研战场上真正有用的东西。比如《1800 题》，里边既有好题，又有废题，我相信很多人都希望有人能从中去掉重复的题目，选出大纲要求的题目，并能把解答写得更通俗易懂点，可是当时没有人这么做，而现在我所做的工作就是从这 1800 道题中选出了大纲要求的题目，并且修正了部分解答，使其更容易理解，我想这也是读者很想要的。其次是严蔚敏的《数据结构》，此书写得很严谨，语言表述非常专业，对于基础稍差的学生来说读起来十分费力，要很长时间才能适应这本书的写作风格。我当时就是在这本书中痛苦地挣扎了很久，看第三遍的时候才可以说适应了。如果当时有一本辅导书帮我把那些复杂程序的执行过程，拗口的专业术语，令人头大的符号翻译成容易理解的语言，我就可以节省很多时间，可惜当时没有。同样我现在所做的事情就是根据自己复习的经验，以及对这本书的理解，把其中考试不需要的内容删掉，把需要的内容改造成一般考生容易接受的形式来讲述。对于李春葆的《数据结构习题与解析》我也做了类似的处理，并且，我在本书中穿插讲解了一些考试大纲中没有明文规定但是很多算法题目中大量用到的算法设计思想，来帮助大家提高解算法设计题的能力，比如搜索（打印图中两结点之间的所有路径）、分治法（二分法排序、求树的深度等）等算法思想。因此我相信本书会给读者的考研复习带来很大的帮助。

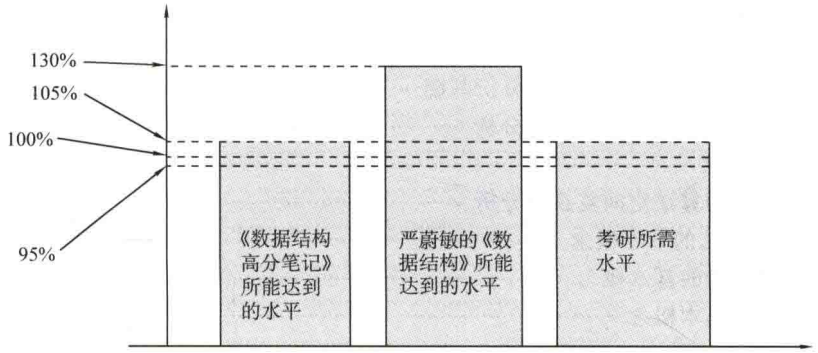
本书特点：

（1）精心挑选出适合考研的习题，并配上通俗易懂的答案，供读者自测和练习。

（2）总结出考研必备知识点，并且帮读者把其中过于专业、过于严谨的表述翻译成通俗易懂的语言。

（3）针对近年数据结构大题的出题风格（比如算法设计题目中的三段式题目：1）表述算法思想；2）写出算法描述；3）计算算法的时间和空间复杂度），设计了独特的真题仿造部分，让读者在复习的过程中逐渐适应不同类型的题目。

此时，很多学生会问：“我只看你这本书够不够？还需要自己准备其他书吗？”对于这个问题，我用下图来回答。



从图中可以看到，如果你只看本书，你能达到考研要求水平的95%～105%，为什么是这样，因为今年大纲还没有公布，所以我不敢保证我的书涵盖大纲所有内容。但是数据结构中的经典内容本书已经全部包括，再加上对这两年统考大纲范围的解读，估计今年大纲变化不会太大，毕竟数据结构是一门经典科目，因此考研对这一门科的考查范围较为稳定。从图中同样可以看出，掌握了严蔚敏的《数据结构》，读者可以至少掌握比考试范围多出30%的内容，但是这需要花很多时间，并不可行。因此在这里我建议读者先看本书，把重要知识点先拿到手，然后把严蔚敏的《数据结构》当作字典来用，等正式大纲出来之后再进行查缺补漏，这不失为一种较为高效的复习方法。这本书不仅涵盖了考试大纲绝大部分内容，更重要的是它会帮助你理解大纲，理解出题人的思路，这样读者就会明白哪一类的题目有可能考，哪一类的题目不会考，慢慢地，复习的方向感会越来越明确，效率会越来越高。

参加本书编写的人员还有：章露捷，刘建萍，施伟，刘炳瑞，刘菁，孙琪，金苍宏，蔡明婉，吴雪霞，孙建兴，张继建，胡素素，邱纪虎，率方杰，李玉兰，率秀颂，刘忠艳，赵建，张兆红，张来恩，张险峰，殷凤岭，于雪友，周桂芝，张玉奎，李亚静，周莉，李娅，刘梅，殷晓红，李艳红，王中静，张洪英，王艳红，周晓红，杨秋侠，秦凤利，叶萍。

编　者

目　录

第1章 绪 论

作者的话：

虽然本章涉及的知识点在考研大纲中没有明确要求，但它是学好数据结构的基础。本章对于数据结构科目在考研中涉及的不同参考书的繁杂表述和规定做了一定的说明和简化，对于考研大纲所要求的知识点，抽象出了一套易于接受的学习方法，因此拿到这本书的考生，务必认真阅读这一章。

本章概略

▲ **针对考研数据结构的代码书写规范以及C&C++语言基础**

对于考研数据结构，需要C与C++语言作为基础，但是又不需要太多，因此此处讲解有针对性。现在你面临的是研究生考试，要在答题纸上写代码，代码的评判者是阅卷老师，而不是TC、VC6.0等编译器。如果之前你只熟悉在这些编译器下写代码，那么你要看看这一部分，这里教你怎么快速地写出能让阅卷老师满意的代码。

▲ **算法的时间复杂度分析基础**

经过这几年的计算机统考，数据结构中综合题三段式的考题模式已经成型，对于算法的时间复杂度分析已经是每年的必考内容，相对于算法的空间复杂度，时间复杂度的分析更具有统一性，因此这里抽象出时间复杂度分析的一般套路，以方便考生理解和学习。对于空间复杂度分析，则放在以后各章中具体问题具体对待。

▲ **数据结构和算法的基本概念**

这一部分介绍一些贯穿于整本书的基本概念。

1.1 针对考研数据结构的代码书写规范以及C&C++语言基础

1.1.1 考研综合应用题中算法设计部分的代码书写规范

要在答题纸上快速地写出能让阅卷老师满意的代码是有技巧的，这与写出能在编译器上编译通过的代码有所不同。为了说明这一点，首先看一个例子：

设将n（n>1）个整数存放到一维数组R中。设计一个算法，将R中的序列循环左移P（0<P<n）个位置，即将R中的数据由$\{X_0, X_1, \cdots, X_{n-1}\}$变换为$\{X_p, X_{p+1}, \cdots, X_{n-1}, X_0, X_1, \cdots, X_{p-1}\}$。要求：写出本题的算法描述。

分析：

本题不难，要实现R中序列循环左移P个位置，只需先将R中前P个元素逆置，再将剩下的元素逆置，最后将R中所有的元素再整体做一次逆置操作即可。本题算法描述如下：

```
#include<iostream>                                    //1
#define N 50                                         //2
using namespace std;                                 //3
void Reverse(int R[],int l,int r)                    //4
{                                                    //5
      int i,j;                                       //6
```

```
        int temp;                                    //7
        for(i=l,j=r;i<j;++i,--j)                     //8
        {                                            //9
          temp=R[i];                                 //10
          R[i]=R[j];                                 //11
          R[j]=temp;                                 //12
        }                                            //13
}                                                    //14
void RCR(int R[],int n,int p)                        //15
{                                                    //16
     if(p<=0||p>=n)                                  //17
       cout<<"ERROR"<<endl;                          //18
     else                                            //19
     {                                               //20
       Reverse(R,0,p-1);                             //21
       Reverse(R,p,n-1);                             //22
       Reverse(R,0,n-1);                             //23
     }                                               //24
}                                                    //25
int main()                                           //26
{                                                    //27
   int L,i;                                          //28
   int R[N],n;                                       //29
   cin>>L;                                           //30
   cin>>n;                                           //31
   for(i=0;i<=n-1;++i)                               //32
      cin>>R[i];                                     //33
   RCR(R,n,L);                                       //34
   for(i=0;i<=n-1;++i)                               //35
      cout<<R[i]<<" ";                               //36
   cout<<endl;                                       //37
   return 0;                                         //38
}                                                    //39
```

以上程序段是一段完整的可以在编译器下编译运行的程序，程序比较长，对于考试答卷，完全没有必要这么写。

第1句和第3句，在大学学习期间所写的程序中几乎都要用到，研究生考试这种选拔考试不会用这种东西来区分学生的优劣，因此答题过程中没必要写，可去掉。

第2句定义了一个常量，如果题目中要用一个常量，在用到的地方加上一句注释，说明某常量之前已经定义即可。没必要再在前面补上一句#define ××××，因为试卷是答题纸，不是编译器，插入语句不是那么方便。为了节省考试时间且使试卷整洁，第2句可去掉。

第26～39句是主函数部分，之前声明的函数（第4～25句）在这里调用。在答题中，只需要写出自己的函数说明（第4～25句），写清楚函数的接口（何为接口，下边会细致讲解）即可，阅卷老师就知道你已经做好了可以解决这个题目的工具（函数），并且说明了工具的使用方法（函数接口），因此第26～39句也可以去掉。

经过以上删减，就变成以下程序段了，显然简洁了很多。

```
void Reverse(int R[],int l,int r)                    //1
{                                                    //2
   int i,j;                                          //3
   int temp;                                         //4
   for(i=l,j=r;i<j;++i,--j)                          //5
   {                                                 //6
       temp=R[i];                                    //7
       R[i]=R[j];                                    //8
       R[j]=temp;                                    //9
    }                                                //10
}                                                    //11
void RCR(int R[],int n,int p)                        //12
{                                                    //13
    if(p<=0||p>=n)                                   //14
       cout<<"ERROR"<<endl;                          //15
    else                                             //16
    {                                                //17
       Reverse(R,0,p-1);                             //18
       Reverse(R,p,n-1);                             //19
       Reverse(R,0,n-1);                             //20
    }                                                //21
}                                                    //22
```

这里来说一下函数的接口。假如上述函数是一台机器，可以用原材料来加工成成品，那么接口就可以理解成原材料的入口，或成品的出口。例如，上述程序段中的第 12 句：RCR(int R[],int n,int p)就包含一个接口，它是原材料的一个入口。括号里所描述的就是原材料类型以及名称，是将来函数被调用的时候所要放进去的东西，是在告诉别人，需要 3 个原材料：第一个是一个 int 型的数组，第二个是一个 int 型的变量，第三个也是一个 int 型的变量。第 15 句：cout<<"ERROR"<<endl;也是一个接口，它会在输出设备上打印出英文单词 ERROR，用来提示用户这里出错了，这算是成品的出口，这里的成品就是一个提示。同时，第 1 句中传入 int 型数组的地方也可以理解为一个产品的出口，因为从这里传入的数组的内容，将在函数执行完后被加工成我们想要的内容。通过以上说明，可以把接口理解为**用户和函数打交道的地方，通过接口，用户输入了自己的数据，得到了自己想要的结果。**

至此，我们可以知道考研综合应用中算法设计题中的代码部分重点需要写哪些内容了，即**只需写出一个或多个可以解决问题的有着清楚接口描述的函数**即可。

1.1.2 考研中的 C&C++语言基础

本节的标题是 C 语言及 C++语言，而不是单一的一种语言，是因为本书有些程序的书写包含了这两种语法。对于考试答题来说，C++不因为它是 C 语言的升级版就能取代 C。C 和 C++是各有所长的，我们从两者中挑选出来对考研答卷有利的部分，组合起来应用。下边具体介绍针对考研数据结构的 C 和 C++语言基础。

1．数据类型

对于基本的数据类型，如整型 int、long、…（考研中涉及处理整数的题目如果没有特别要求用 int 足够了），字符型 char，浮点型 float、double、…（对于处理小数的问题，题目没有特殊要求的情况下用 float 就足够了）。这些大家都了解，就不再具体讲解了，这里主要讲解的是**结构型**和**指针型**。

（1）结构型

结构型就是用户自己制作的数据类型。其实我们常用的数组也是用户自己制作的数据类型。数组是由多个相同数据类型的变量组合起来的，例如：

int a[maxSize];//maxSize 是已经定义的常量

该语句就定义了一个数组，名字为 a，就是将 maxSize 个整型变量连续地摆在一起，其中各整型变量之间的位置关系通过数组下标来反映。如果想制作一个数组，第一个变量是整型变量，第二个变量是字符型变量，第三个变量是浮点型变量，该怎么办呢？这时就用到结构体了。**结构体就是系统提供给程序员制作新的数据类型的一种机制，即可以用系统已经有的不同的基本数据类型或用户定义的结构型，组合成用户需要的复杂数据类型。**

例如，上面提到的要制作一个由不同类型的变量组成的数组可以进行如下构造：

```
typedef struct
{
int    a;
char   b;
float  c;
}TypeA;
```

上面的语句制造了一个新的数据类型，即 TypeA 型。语句 int b[3];申请了一个数组，名字为 b，由 3 个整型分量组成。而语句 TypeA a;同样可以认为申请了一个数组，名字为 a，只不过组成 a 数组的 3 个分量是不同类型的。对于数组 b，b[0]、b[1]、b[2]分别代表数组中第一、第二、第三个元素的值。而对于结构体 a，a.a、a.b、a.c 分别对应于结构体变量 a 中第一、第二、第三个元素的值，两者十分相似。

再看语句 TypeA a[3];，它定义了一个数组，由 3 个 TypeA 型的元素组成。前边已经定义 TypeA 为结构型，它含有 3 个分量（其实应该叫作结构体的成员，这里为了类比，将它叫作分量），因此 a 数组中的每个元素都是结构型且每个元素都有 3 个分量，可以把它类比成一个二维数组。例如，int b[3][3];定义了一个名字为 b 的二维数组。二维数组可以看成其数组元素是一维数组的一维数组，如果把 b 看成一个一维数组，其中的每个数组元素都有 3 个分量，与 a 数组不同的地方在于，b 中每个元素的 3 个分量是相同类型的，而 a 数组中每个元素的 3 个分量是不同数据类型的。b 数组取第一个元素的第一个分量的值的写法为 b[0][0]，对应到 a 数组则为 a[0].a。

结构体与数组类比关系可以通过图 1-1 来形象地说明。

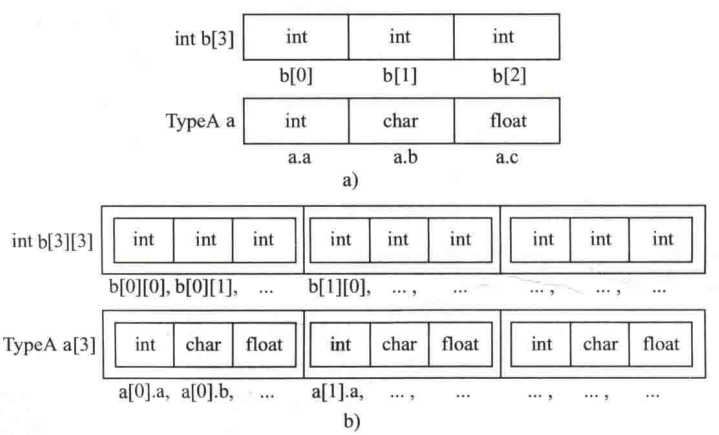


图 1-1 结构体与数组的类比

a）结构体和一维数组的类比 b）数组和结构体数组的类比

（2）指针型

指针型和结构型一样，是比较难理解的部分。对于其他类型的变量，变量里所装的是数据元素的内

容，而指针型变量内部装的是变量的地址，通过它可以找出这个变量在内存中的位置，就像一个指示方向的指针，指出了某个变量的位置，因此叫作指针型。

指针型的定义方法对每种数据类型都有特定的写法，有专门指向 int 型变量的指针，有专门指向 char 型变量的指针等。对于每种变量，指针的定义方法有相似的规则，例如以下语句：

```
int *a;     //对比一下定义 int 型变量的语句：int a;
char *b;    //对比一下定义 char 型变量的语句：char b;
float *c;   //对比一下定义 float 型变量的语句：float c;
TypeA *d;   //对比一下定义 TypeA 型变量的语句：TypeA d;
```

上面 4 句分别定义了指向整型变量的指针 a、指向字符型变量的指针 b、指向浮点型变量的指针 c 和指向 TypeA 型变量的指针 d。与之前所讲述的其他变量的定义相对比，指针型变量的定义只是在变量名之前多出一个“*”而已。

如果 a 是个指针型变量，且它已经指向一个变量 b，则 a 中存放变量 b 所在的地址。*a 就是取变量 b 的内容（x=*a;等价于 x=b;），&b 就是取变量 b 的地址，语句 a=&b;就是将变量 b 的地址存于 a 中，即大家常说的指针 a 指向 b。

指针型在考研中用得最多的就是和结构型结合起来构造结点（如链表的结点、二叉树的结点等）。下面具体讲解常用结点的构造，这里的“构造”可以理解成先定义一个结点的结构类型，然后用这个结构型制作一个结点。这样说虽不太严谨，但便于理解。

（3）结点的构造

要构造一种结点，必须先定义结点的结构类型。下面介绍链表结点和二叉树结点结构型的定义方法。

1）链表结点的定义。

链表的结点有两个域：一个是值域（data），用来存放数据；另一个是指针域（next），用来存放下一个结点的位置，如图 1-2 所示。

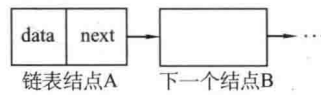


图 1-2 链表结点

因此，链表的结构型定义如下：

```
typedef struct Node
{
    int data;                //这里默认的是 int 型，如需其他类型可直接修改
    struct Node *next;       //指向 Node 型变量的指针
}Node;
```

上面这个结构型的名字为 Node，因为组成此结构体的成员中有一个是指向和自己类型相同的变量的指针，内部要用自己来定义这个指针，所以写成 struct Node *next;。这里指出，**凡是结构型（假设名为 a）内部有这样的指针型（假设名为 b），即 b 是用来存放和 a 类型相同的结构体变量地址的指针型（如图 1-2 中结点 A 的指针 next，next 所指的结点 B 与结点 A 是属于同一结构型的），则在定义 a 的 typedef struct 语句之后都要加上 a 这个结构型的名字**，如上述结构体定义中黑体的 **Node**。与之前定义的结构型 TypeA 相比较，会发现这里的结构型 Node 在定义方法上的不同。

有的参考书中把上述链表结点结构定义写成如下形式：

```
typedef struct node
{
    ...
    ...
}Node;
```

可以发现，有一个“node”和一个“Node”，即结构体定义中的上下两个名称不同。其实对于考研来说，这样写除了增加记忆负担之外，没有别的好处，所以希望考生在定义结点结构型的时候，将上下名称写成一致，并且如果结构体内没有指向自己类型的指针，你也可以把黑体的 Node 加上，这样更方便记忆。

2）二叉树结点的定义。

在链表结点结构型的基础上，再加上一个指向自己同一类型变量的指针域，即二叉树结点结构型，例如：

```
typedef struct BTNode
{
    int   data;
    struct BTNode *lchild;
    struct BTNode *rchild;
}BTNode;
```

在考研的数据结构中，只需要熟练掌握以上两种结点（链表、二叉树）的定义方法，其他的结点都是由这两种衍生而来的（其实二叉树结点的定义也是由链表结点的定义衍生而来的，二叉树结点只不过比链表结点多了一个指针而已），无需特意地去记忆。

说明：对于结构型，用来实现构造结点的语法有很多不同的表述，没必要全部掌握。上面讲到的那些语法用来构造结点已经足够用了，建议大家熟练掌握以上两种构造结点的结构体定义的写法，其他的写法可不予理睬。有些语法对考试来说既复杂又没有意义，例如，上面二叉树结点的定义有些参考书中写成：

```
typedef struct BTNode
{
    int   data;
    struct BTNode *lchild;
    struct BTNode *rchild;
}BTNode,*btnode;
```

可以看到在最后又多了个*btnode，其实在定义一个结点指针 p 的时候，BTNode *p;等价于 btnode p;。对于定义结点指针，BTNode *p;这种写法是顺理成章的，因为它继承了之前 int *a;、char *b;、char *c;和 TypeA *d;这些指针定义的一般规律，使我们记忆起来非常方便，不必再加个 btnode p;来增加记忆负担。因此在考研中我们不采取这种方法，对于上面的结构体定义，删去*btnode，统一一个 BTNode 就可以解决所有问题。

通过以上的讲解，知道了链表结点和二叉树结点的定义方法。结构型定义好之后，就要用它来制作新结点了。

以二叉树结点的制作为例，有以下两种写法：

```
① BTNode BT;
② BTNode *BT;
BT=(BTNode*)malloc(sizeof(BTNode)); //此句要熟练掌握
```

①中只用一句就制作了一个结点，而②中需要两句，比①要繁琐，但是考研中用得最多的是②。②的执行过程：先定义一个结点的指针 BT，然后用 malloc()函数来申请一个结点的内存空间，最后让指针 BT 指向这片内存空间，这样就完成了一个结点的制作。②中的第二句就是用系统已有的 malloc() 函数申请新结点所需内存空间的方法。考研数据结构中所有类型结点的内存分配都可用 malloc()函数来完成，模式固定，容易记忆。

图 1-3 所示为利用空间申请函数申请一个结点空间，并用一个指针（图中为 p）指向这个空间的标准模板。考生需要将这个模板背下来，以后制作一个新结点的时候，只要把结点结构型的名称填入图 1-3 括号中的空白处即可。

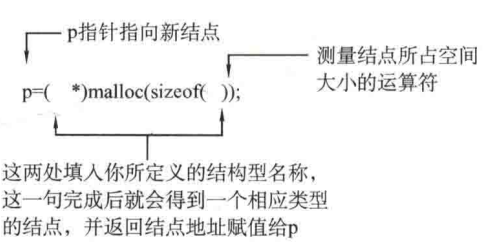


图 1-3 结点空间申请函数