《数据结构与算法》

实践教程

参编：李华玲，马巧梅，

李玉蓉，孔祥艳，

贾美丽，董晓丽，

杨顺民，潘广贞

特色：学习与竞赛融合

拓展与产业结合

思路：三阶梯：层层递进 环环相扣

**基础篇(教材)—提高篇(竞赛)—拓展篇(产业....)**

**建议：1、以图示的方式 从内存的角度理解程序**

1. **拓展篇：科研项目（研究型 侧重点）**
2. **课程设计：工程应用结合（应用背景 模型 对比分析结构 文档 开发）**
3. **知识类型的复习和小案例，有没有必要写？放在哪?**
4. **第一章：预备知识（）**

# 绪论

## 本章实践目标

* 巩固C语言基础，能够熟练使用函数、结构体、指针等知识解决实际问题
* 能够运用类C语言的规范并结合数据结构的基本概念，正确表达和实现抽象数据类型
* 能够根据算法的复杂度对解决实际问题的方案进行比较与选择

## **本章实践体系概略**

**1 基础篇**

**三元组**

1.1 C语言核心及C++引用

* 函数
* 指针与动态分配内存
* 结构体与typedef
* C++引用

1.2在实际的应用场景中体会数据结构的基本概念

* 数据结构的基本概念
* 应用举例

1.3基础实践：抽象数据类型

* 抽象数据类型的基本概念
* 应用举例

1.4开发高效算法

* 算法的基本概念
* 确定各种类型算法的复杂度
* 使用动态编程设计寻找斐波那契数的高效算法

**2 提高篇**

2.1 求二进制数中1的个数

2.2 调控CPU的占有率

2.3 围栏修复

2.4 斐波那契数列

**3 拓展篇**

3.1 图形中的三元组

3.2 ...

## **1.1 C语言核心及C++引用**

C语言是典型的面向过程的程序设计语言，面向过程的程序设计=函数＋调用。也就是说，一个C语言可由一个主函数和若干个其它函数构成，由主函数调用其他函数，其他函数也可以互相调用，同一个函数可以被一个或多个函数调用任意多次。

在实际应用中，一个程序主要包括以下两方面的信息，对数据的描述和对操作的描述。数据是操作的对象，对数据的描述要求在程序中要指定用到哪些数据，以及这些数据的类型和数据的组织形式，这就是**数据结构**。操作的目的是对数据进行加工处理，以得到期望的结果。对操作的描述其实就是写出计算机进行操作的步骤，也就是**算法**。

著名的计算机科学家沃思（Nikiklaus Wirth）提出一个公式：**算法+数据结构=程序**。其中算法是灵魂，数据结构是加工对象，语言是工具。本课程采用类C语言描述算法，采用C语言实现算法。其中类C语言是从C语言选出一个核心子集，并添加了C++的引用作为函数参数，构成了类C语言。所以要求大家首先必须巩固C语言中的重要知识并熟悉C++中的引用。

### 1.1.1函数

函数，从本质上讲就是用来完成特定功能的。在程序设计中要善于利用函数，以实现代码复用。实际应用中，所有的算法均是用函数进行封装。

**1、函数的定义**

C语言要求，在程序中用到的函数，必须“先定义，后使用”。

定义函数的一般形式为：

类型名 函数名（形式参数表列）

{

函数体

}

对于以上的定义，说明几点：

1. 函数名应反映其代表的功能，见名知义。
2. 形式参数表列应指明函数的参数的名字和类型，以便在调用函数时向它们传递数据，对无参函数不需要这项。
3. 类型名，即函数返回值的类型。函数的返回值是通过函数体中的return语句获得的。
4. 函数体指定函数应当完成什么操作，即函数的功能，函数体包括声明部分和语句部分。

**【示例1\_1】**定义一个函数，实现两个整数之和。

解答思路：（1）函数名，因实现和的功能，见名知义，命名为 add。

（2）形式参数表列，因完成两个整数之和，所以两个参数的类型均为int。

（3）类型名，两个整数之和，结果还是整数，所以函数返回值的类型也为int。

（4）函数体，完成加法，并返回结果。

正确答案：

int add(int x, int y)

{

int sum;

sum = x+y;

return sum

}

或者

int add(int x, int y)

{

return x+y;

}

**2、函数的调用**

定义函数的目的是为了调用此函数，以得到预期的结果。调用函数的一般形式为：

**函数名（实参表列）**

对于以上的调用形式，说明两点：

1. 若在函数的定义中形式参数表列不为空，则实参表列中的实参，类型必须与形式参数表列中的参数类型一致或者赋值兼容，实际参数可以是变量、常量或表达式。
2. 若在函数的定义中形式参数表列为空，则实参表列也没有，但括号不能省略。

**【示例1\_2】** 从键盘接收两个整数，要求用例1\_1中定义的函数计算两个整数的和并输出。

**正确答案：**

# include<stdio.h>

// 例1\_1中函数的定义

int add(int x, int y)

{

return x+y;

}

// 主函数

int main()

{

int a, b, sum;

printf("请从键盘接收两个整数：\n");

scanf("%d%d", &a, &b);

sum = add(a, b); //调用函数add

printf("两个整数的和为：%d", sum);

return 0;

}

函数调用过程分析：

主函数中包含了一个函数调用语句add(a, b)，add后面括号内的a和b 是实参，a和b是在main函数中定义的变量，通过函数调用，实参a，b的值传给了函数add 的形式参数x和y，之后在函数add 中完成了x和y的相加，并把结果返回main函数，赋给在main函数中定义的变量sum。

**3、函数的声明**

若用户定义的函数在主调函数的后面，则应该对被调用的函数作声明。函数声明的一般形式为：

**函数类型 函数名（形式参数表列）；**

对于以上的函数声明形式，说明两点：

（1）此处的形式参数表列对无参函数不需要这项，对有参函数，可以只写每个参数的参数类型，或者参数类型和参数名都写。

（2）函数的声明，可以在主调函数内部进行声明，也可以在文件的开头进行声明。

**【示例1\_3】** 对于例1\_2中的程序，如果将函数add和main函数换一下位置，则必须在主函数的内部或者主函数之前对 add 函数进行声明。请写出函数声明的语句。

**正确答案：**

int add(int x, int y)；

或者

int add(int , int )；

### **1.1.2 指针和动态分配内存**

指针是C语言的一个重要概念，也是C语言的精华，指针极大地丰富了C语言的功能。准确而灵活的应用指针可以编出精练而高效的程序。学习指针是学习C语言中最重要的一环，能否正确理解和使用指针是我们是否掌握C语言的一个标志。

**1、指针变量的定义和引用** 用C语言中基本数据类型定义变量，变量里所装的是数据元素的内容，而指针型变量里装的是变量的地址，通过指针可以找出这个变量在内存中的位置。

定义指针变量的一般形式为：

**类型名 \*指针变量名；**

对于以上的定义形式，说明几点：

1. 类型名用来指定此指针变量可以指向的变量的类型。
2. “\*”号表示该变量为指针型变量。

引用指针变量可能有三种情况，分别是给指针变量赋值、引用指针变量指向的变量和引用指针变量的值。

**【示例1\_4】**请说明下面每条语句的含义。

int a, \*p;

p = &a;

\*p = 10;

printf(“%d”, \*p);

printf(“%x”, p);

**正确答案：**

int a, \*p; // 此语句进行变量的定义，表示定义了一个整型变量a，定义了一个指**向整型数据的指针变量p**。**注意：**在说明指针变量时不能只是说“定义了一个指针变量p”，而应完整地说“定义了一个指向整型数据的指针变量p”**。**

p = &a; // 此语句给指针变量赋值，表示把a的地址赋给指针变量p，让p指向a

\*p = 10; // 此语句表示把整数10赋给指针变量p所指向的变量，等价于a = 10

printf(“%d”, \*p); // 此语句表示以整数形式输出指针变量p所指向的变量的值，即输出变量a的值。

printf(“%x”, p); // 此语句表示以十六进制形式输出指针变量的值，即输出变量a的地址。

**2、指针变量作为函数参数**

函数的参数不仅可以是普通的数据类型，还可以是指针类型。

**【示例1\_5】 完成a和b两个存储单元内容的交换。**

**解决方案１：**试图使用普通变量做函数的参数，传值进行函数调用，完成数据的交换

# include<stdio.h>

// 试图定义一个函数，使用普通变量做参数，完成数据的交换

void swap(int m, int n)

{

int temp;

temp = m;

m = n;

n = temp;

}

// 主函数

int main()

{

int a = 5, b = 10;

printf("a = %d, b = %d\n",a ,b);

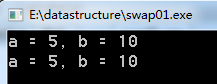
swap(a, b); // 值传递

　　printf("a = %d, b = %d\n",a ,b);

return 0;

}

输出：



**结论：**使用普通变量做参数，值传递没有完成数据的交换。为什么呢？我们来分析一下，在主函数中，执行函数调用语句swap(a, b) 时，在内存中，实际参数a和b的值确实传给了函数的形式参数m和 n，在函数执行的过程中，m和n的值确实交换了，但主函数中a和b的值没有完成交换。

**解决方案２：**使用指针变量做函数的参数，传地址进行函数调用，完成数据的交换

# include<stdio.h>

// 定义一个函数，使用指针变量做参数，完成数据的交换

void swap(int \*m, int \*n)

{

int temp;

temp = \*m;

\*m = \*n;

\*n = temp;

}

// 主函数

int main()

{

int a = 5, b = 10;

printf("a = %d, b = %d\n",a ,b);

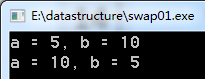
swap(&a, &b); // 地址传递

printf("a = %d, b = %d\n",a ,b);

return 0;

}

输出：



**结论：**使用指针做参数，通过地址传递完成了数据的交换。我们来分析一下，在主函数中，执行函数调用语句swap(&a, &b) 时，实际参数a的地址和b的地址传给了函数的形式参数m和 n，意味着m指向了a，n指向了b，在函数执行的过程中，m所指向的变量的内容和n所指向的变量的内容完成了交换，意味着主函数中a和b的值完成了交换。

**3、动态分配内存与指向它的指针变量**

C语言允许建立内存动态分配区域来存放数据，这些数据不必在程序的声明部分定义，由于未在声明部分定义它们为变量或数组，因此不能通过变量名或数组名去引用这些数据，只能通过指针来引用。

对内存的动态分配与释放是通过系统的库函数来实现的，主要有malloc，calloc，realloc和free这四个函数。这四个函数的声明位于stdlib.h头文件中，在用到这些函数时，应当用# include<stdlib.h> 把stdlib.h头文件包含到程序文件中。下面介绍一下实验中最常用到的三个函数的函数原型：

1. void \*malloc(unsigned int size)，其作用是在内存的动态存储区中分配一个长度为size的连续空间。此函数的返回值为所分配区域的第一个字节的地址，只提供一个纯地址，不指向任何类型的数据，在实际使用时，必须强制转换成有指向的地址，才能进行数据存取。
2. void \*realloc(void \*p, unsigned int size)，如果已经用malloc函数获得了动态空间，想改变其大小，可以用此函数重新分配，表示将p所指向的动态空间的大小改变为size。
3. void free(void \*p)，其作用是释放指针p所指向的动态空间。

**【示例1\_6】 建立动态数组，输入一个班级学生的C语言成绩，计算均分并输出。**

**解决思路：**

（1）从键盘接收班级人数；

（2）根据班级的人数为存放成绩的数组动态分配内存空间；

（3）从键盘接收学生的成绩；

（4）计算均分并输出；

（5）释放内存；

**正确答案：**

# include<stdio.h>

# include<stdlib.h>

int main()

{

int count, i, sum;

float aver;

int \*p;

// 输入班级的人数

printf("请输入班级的人数：");

scanf("%d", &count);

// 根据人数动态分配内存

p = (int \*)malloc(sizeof(int) \* count);

// 从键盘接收成绩

printf("请输入C语言的成绩：");

for (i=0; i<count; i++)

{

scanf("%d",p+i);

sum += p[i]; // 或者写成sum += \*(p+i)

}

// 计算并输出均分

aver = (float)sum / count;

printf("这个班级的均分为：%.2f", aver);

// 释放内存

free(p);

return 0;

}

**1.1.3 结构体和typedef**

结构体可以理解为用户用已有数据类型（int,char,float...）为原料制作的数据类型，是系统提供给程序员制作新的数据类型的一种机制，即用户可以用系统已经有的基本数据类型或者用户定义的结构体类型组合成复杂的数据类型。是C语言中非常重要的知识。

**1、结构体类型的定义**

声明一个结构体类型的一般形式为：

**struct 结构体名**

**{成员表列}；**

其中，大括号内是该结构体所包含的子项，称为结构体的**成员**。对每一个成员都要进行类型声明，即

**类型名 成员名；**

**2、定义结构体类型的变量**

使用结构体类型定义变量的形式常用的有三种：

（1）声明的同时进行定义

**struct 结构体名**

**{成员表列} 变量名；**

（2）先申明，后定义

struct 结构体名

{成员表列}；

**struct 结构体名 变量名；**

（3）使用无名结构体进行变量的定义

**struct**

**{成员表列} 变量名；**

**3、使用typedef给已有的或者自己定义的类型起别名**

typedef的作用是给现有的数据类型起个别名来代替已有的类型名。对于普通类型、数组类型、指针类型、结构体类型等所有类型，定义方法都是一样的。简单地说，就是按照定义变量的方式，把变量名换成新类型名，并且在最前面加上typedef后即可完成给已有类型起别名的任务，之后就可以直接使用这个别名定义变量了。

如：定义一个字符指针类型的过程可以这样理解：

1. 定义一个指向字符型数据的指针变量p

char \*p;

1. 用新的类型名String取代上面的变量p

char \*String;

1. 在前面加上typedef

**typedef char \*String;**

**// 相当于给指向char型数据的指针类型（简称字符指针类型）起了个别名叫String**

1. 使用新类型String定义变量s

String s; // 等价于char \*s;

**4、使用指针访问结构体中的成员**

如果p指向一个结构体变量s，则访问变量s中的成员有三种方式：

1. s.成员名；
2. (\*p).成员名；
3. p->成员名；

**【示例1\_7】根据下面的要求完成任务：**

**（1）请使用结构体类型描述学生的信息，每个学生的信息包括学号、姓名、性别和年龄。**

**（2）请在（1）的基础上，使用typedef给结构体类型起个别名Student，给指向结构体类型数据的指针类型起个别名StuPoint。**

**（3）编写主函数，定义并直接初始化一个包含3个学生信息的结构体数组，使用三种方式输出三个学生的信息。**

**解题思路：**

按照上文中相关的定义逐一完成即可。

**正确答案：**

（1）描述每个学生信息的结构体定义如下：

struct Student

{ int num;

char name[20];

char sex;

int age;

}；

（2）使用typedef给结构体类型和结构体指针类型起别名

typedef struct Student

{ int num;

char name[20];

char sex;

int age;

} Student, \*StuPoint;

（3）在主函数中完成结构体数组的定义，初始化并使用三种方式完成信息的输出。

int main()

{

// 定义并初始化结构体数组

Student stu[3] = {{10101,"Li Lin",'M',18},

{10102,"Ma Jun",'M',19},

{10104,"Lv Min",'F',20}};

// 输出数组中三个学生的信息

// 方式1：使用数组下标访问

int i;

for (i=0; i<3; i++)

{

printf("%d\t%s\t%c\t%d\n", stu[i].num, stu[i].name, stu[i].sex, stu[i].age);

}

// 方式2：指针指向结构体数组，用“\*”运算符访问

StuPoint p; // 等价于Student \*p;

for (p=stu; p<stu+3; p++)

{

printf("%d\t%s\t%c\t%d\n", (\*p).num, (\*p).name, (\*p).sex, (\*p).age);

}

// 方式3：指针指向结构体数组，用“->”运算符访问

for (p=stu; p<stu+3; p++)

{

printf("%d\t%s\t%c\t%d\n", p->num, p->name, p->sex, p->age);

}

return 0;

}

在本课程中，无论是线性结构、树结构还是图结构，要构造一个节点，必须先定义结点的结构类型。如，线性结构中链表的结点有两个域，一个是数据域，用来存放数据；另一个是指针域，用来存放下一个结点的位置。如图1.1所示：

|  |  |
| --- | --- |
| data | next |

|  |  |
| --- | --- |
| data | next |

|  |  |
| --- | --- |
| ... |  |

图1.1 线性结构中链表的结构

链表结点的结构型定义如下：

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LNode, \*LinkList;

关于这个定义说明几点：

（1）这个结构体类型的名字为LNode;

（2）指向这个结构体类型的指针类型名字为LinkList;

（3）组成这个结构体的成员之一data，根据实际的业务需要，可以是任意类型，如int，float等基本数据类型或者是用户定义的结构体类型，如Student等。

（4）组成这个结构体的另一成员next，是指向和自己类型相同的变量的指针，所以要用自己来定义这个指针，所以写成struct LNode \*next。

关于各种结构对应的结构型数据类型的介绍，在后续章节中将陆续学习。

### 1.1.4 C++中的引用

在c++中，建立引用的作用是为变量另起一个名字。声明一个引用的格式如下:

**类型 &引用名 = 已定义的变量名;**

例如：

int i=5；

int &j = i；

这里，声明了一个整数类型的引用j ，用整型变量i对它进行初始化，这时j就可看做是变量i的引用,即是变量i的别名。也就是说，变量i和引用j占用内存的同一位置。当i变化时，j也随之变化，反之亦然。

**【示例1\_8】 完成a和b两个存储单元内容的交换。**

在【示例1\_5】中，函数定义时用普通变量做参数，没能完成数据的交换。用指针做参数，完成了数据的交换，但相对比较麻烦。在本例中，用引用做参数可以非常方便并非常简洁的解决这个问题。

**解决方案：**使用引用做函数的参数，完成数据的交换

# include<stdio.h>

// 定义一个函数，使用引用做参数，完成数据的交换

void swap(int &m, int &n)

{

int temp;

temp = m;

m = n;

n = temp;

}

// 主函数

int main()

{

int a = 5, b = 10;

printf("a = %d, b = %d\n",a ,b);

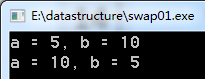
swap(a, b); // 值传递

　　printf("a = %d, b = %d\n",a ,b);

return 0;

}

输出：



**结论：**使用引用做参数，很方便地完成了数据的交换。为什么呢？我们来分析一下，在主函数中，执行函数调用语句swap(a, b) 时，相当于完成了int &m=a和int &n=b，也就是说在内存中，实际参数a、b和形式参数m、 n占用同样的内存区域，在函数执行的过程中，m和n的值完成交换了，所以a和b的值也就完成交换了。

## **1.2 在实际的应用场景中体会数据结构的基本概念**

对于数据结构的基本概念，在以后的章节中会经常出现。不需要刻意的去背，联系生活实际去理解即可。

### 1.2.1 数据结构中的基本概念

**1、数据**

描述客观事物的数字、字符以及所有能输入到计算机中并被计算机程序处理的符号的集合。可以是数字、字符、声音、图形、图像等。

**2、数据元素**

数据的基本单位，在计算机程序中常常作为一个整体进行考虑和处理，可以小到一个字符，也可以大到一本书或一张地图。对于较大的数据元素，还可以进一步分成若干个“数据项”。

**3、数据项**

是数据结构中讨论的最小单位，是数据元素中最基本的，不可分割的数据单位。如：学生成绩管理系统中，描述一个学生的信息，通常包含学号、姓名、成绩1、成绩2等，其中，学号、姓名、成绩1、成绩2就是不可分割的数据项。

**4、数据对象**

性质相同的数据元素的集合，是数据的一个子集。如：全体整数的集合Z={0，+1，+2…}。

**5、数据结构**

1. 相互之间存在一种或多种特定关系的数据元素的集合。其中数据元素之间的关系，称为“结构”，所以数据结构可以简称为带结构的数据元素的集合。
2. 形式定义为：二元组 (D,S) 其中D是数据元素的有限集，S是D上关系的有限集。
3. 数据结构的研究内容为：

**6、数据的逻辑结构**

数据的逻辑结构是对数据元素之间关系的描述。根据数据元素之间关系的不同特性，通常有下列四种逻辑结构：

（1）集合：数据元素之间除了“同属于一个集合”的关系外，别无其他关系。

（2）线性结构：数据元素之间存在一对一的关系，除了第一个元素和最后一个元素，其它元素有且仅有一个直接的前驱和一个直接的后继。

（3）树型结构：数据元素之间存在一对多的关系。具有分支、层次特性，形态像自然界中的树。

（4）图状结构或网状结构：数据元素之间存在多对多的关系。其中的各个结点按逻辑关系互相缠绕，任何两个结点都可以邻接。

**7、数据的存储结构**

是逻辑结构到存储器的一个映射，即数据在计算机内的表示。数据元素之间的关系在计算机中主要有两种不同的表示方法，

（1）顺序存储结构：每个存储结点只含一个数据元素，所有存储结点存放在一块连续的存储区里，用存储结点的位置关系表示数据元素之间的逻辑关系。

（2）链式存储结构：每个存储结点不仅含有一个数据元素，还包含一个指针，每个指针指向一个与本结点有逻辑关系的结点，即用附加的指针表示逻辑关系。

### 1.2.2 应用举例

**【示例1\_9】** 2020年注定是让人难忘的一年，这一年开头中国就备受新冠病毒的侵扰，让全国人民待在家里过了一个与众不同的春节，关注疫情数据也成了大家每天的“必修课”。作为重灾区的湖北，在2020年3月18日新增确诊首次清零。表1\_1是2020年3月19日在疫情实时更新系统中查到的部分省份的数据，请描述其中的数据元素，数据项，数据对象和数据。

表1\_1 2020年3月19日部分省份数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地区** | **新增确诊** | **累计确诊** | **治愈** | **死亡** |
| 湖北 | 0 | 67800 | 57681 | 3130 |
| 广东 | 9 | 1378 | 1318 | 8 |
| 河南 | 0 | 1273 | 1250 | 22 |
| 浙江 | 1 | 1233 | 1217 | 1 |
| 湖南 | 0 | 1018 | 1014 | 4 |
| 安徽 | 0 | 990 | 984 | 6 |
| 山西 | 0 | 133 | 133 | 0 |

**正确答案：**

数据元素：一个地区的疫情记录（湖北，0，67800，57681，3130）。

数据项：湖北，0，67800，57681，3130均是该数据元素的数据项。

数据对象：国内各地区的疫情记录。

数据：全球各国家各地区的疫情记录。

**【示例1\_10】**在现实生活中，有各种各样的数据，它们呈现出不同的逻辑结构特性，归纳起来有四种，请根据你的理解，每一种各举一例。

**参考答案：**

## **1.3 抽象数据类型的表示和实现**

抽象数据类型泛指除基本数据类型以外的数据类型，是由若干基本数据类型归并之后形成的一种新的数据类型，这种类型由用户定义，是描述数据结构的一种理论工具，其目的是使人们能够独立于程序的实现细节来理解数据结构的特性。

### 1.3.1抽象数据类型的相关知识

**1、抽象数据类型的定义**

是指基于一个逻辑类型的数据模型以及定义在该模型上的一组操作。每一个操作由它的输入和输出定义。抽象数据类型的定义取决于它的一组逻辑特性，而与计算机内部如何表示无关。“抽象”的意义在于[数据类型](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B/10997964" \t "https://baike.baidu.com/item/ADT/_blank)的数学特性，其数学特性和具体的[计算机](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA/140338" \t "https://baike.baidu.com/item/ADT/_blank)或语言无关。

抽象数据类型可以用三元组 (D,S,P)表示，其中D是数据元素的有限集，S是D上关系的有限集,P是对D的基本操作集。抽象数据类型的形式定义模板如下：

ADT抽象数据类型名{ 

[数据对象](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AF%B9%E8%B1%A1/3227125" \t "https://baike.baidu.com/item/ADT/_blank)：<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](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B/10997964" \t "https://baike.baidu.com/item/ADT/_blank)名

其中，数据对象、数据关系和基本操作的定义均用伪码描述，每一个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](https://baike.baidu.com/item/%E5%88%9D%E5%A7%8B%E6%9D%A1%E4%BB%B6/7218840" \t "https://baike.baidu.com/item/ADT/_blank)描述>

操作结果：<操作结果描述>

**2、抽象数据类型的表示和实现**

本课程选用的理论教材是采用类C语言进行抽象数据类型的描述，本实践教程在完成抽象数据类型的表示和实现时直接使用C语言和C++的引用来完成，所有的代码可以直接上机运行通过。

### 1.3.2应用举例

**【示例1\_11】**设计实现抽象数据类型“N元组（N\_Tuple）”。每个N元组由任意三个实数的序列构成，基本操作包括：创建一个N元组、取N元组的任意一个分量、置N元组的任意一个分量、求N元组的最大分量、最小分量、显示N元组、销毁N元组、两个N元组的对应分量相加、N元组的每个分量同乘一个比例系数等。要求如下：

1. 写出“N元组”抽象数据类型的定义。
2. 练习抽象数据类型的C语言表示，完成所有基本操作的C语言实现。要求用指针描述“N元组”，动态分配内存。
3. 编写主函数，在主函数中调用相关函数完成“N元组”的基本操作。

**正确答案：**

（1）按照抽象数据类型的形式定义模板可以写出“N元组（N\_Tuple）”抽象数据类型的定义如下：

ADT N\_Tuple{

数据对象：D = {ei|其中1≤i≤N，ei属于ElemSet(定义了关系的某个集合)}

数据关系：R = {<ei，ei+1>|其中1≤i≤N-1}

initTuple(&T,v1,v2,...,vN)

初始条件：

操作结果：构造三元组T，元素e1，e2到en分别被赋予参数v1,v2到vN的值。

destroyTuple(&T)

初始条件：三元组T已经存在。

操作结果：销毁三元组T。

 getElem(T,i,&e)

初始条件：三元组T已经存在，1<=i<=N。

操作结果：用e返回三元组T的第i个元素。

 putElem(&T,i,e)

初始条件：三元组T已经存在，1<=i<=N。

操作结果：用e值取代三元组T的第i个元素。

isAscending(T)

初始条件：三元组T已经存在。

操作结果：如果三元组T的三个元素按升序排列，则返回TRUE；否则返回FALSE。

 isDescending(T)

初始条件：三元组T已经存在。

操作结果：如果三元组T的三个元素按降序排列，则返回TRUE，否则返回FALSE。

getMax(T,&e)

初始条件：三元组T已经存在。

操作结果：用e返回T的3个元素中的最大值。

 getMin(T,&e)

初始条件：三元组T已经存在。

操作结果：用e返回T的3个元素中的最小值。

add(T1, &T2)

初始条件：T1,T2均存在。

操作结果：实现两个三元组的对应分量相加或相减，用T2带回运算结果。

mulCoef(&T1,coef)

初始条件：T1存在。

操作结果：给三元组的各分量同乘一个比例因子，用T1带回乘系数之后的结果。

printTriplet(T)

初始条件：T存在。

操作结果：输出T的每个元素的信息。

}ADT N\_Tuple

（2）“N元组”抽象数据类型的C语言表示如下：

# include<stdio.h> // 标准的输入输出函数库

# include<stdlib.h> // 使用动态分配内存必须引入的函数库

# include<stdbool.h> // 使用\_Bool型需要引入的函数库 \_Bool是在C99标准中添加的

# define N 3 // 修改此处可以变成任意元组的程序

typedef float ElemType; // 给float起个别名叫ElemType

typedef ElemType \*Tuple; // 给指向ElemType类型的指针类型起别名叫Tuple

// N元组初始化

void initTriplet(Tuple &T, ElemType v[])

{

// 动态分配内存

T = (ElemType \*)malloc(sizeof(ElemType) \* N);

if (!T) exit(-2);

int i;

for (i=0; i<N; i++)

{

T[i] = v[i];

}

}

// 销毁三元组,退出系统

void destroyTuple(Tuple &T)

{

if(!T)

free(T);

exit(0);

}

// 用e获取T的第i（1～N）个元素的值

void getElem(Tuple T, int i, ElemType &e)

{

if (i < 1 || i > N)

printf("位序应在1到%d之间", N);

else

e = T[i-1];

}

// 置T的第i元的值为e

void putElem(Tuple &T, int i, ElemType e)

{

if (i < 1||i > N)

printf("位序应在1到%d之间", N);

else

T[i-1] = e;

}

// 是否升序排列

\_Bool isAscending(Tuple T)

{

\_Bool flag = 1;

int i;

for (i=0; i<N-1; i++)

flag = flag && (T[i] < T[i+1]);

return flag;

}

// 是否降序排列

\_Bool isDescending(Tuple T)

{

\_Bool flag = 1;

int i;

for (i=0; i<N-1; i++)

flag = flag && (T[i] > T[i+1]);

return flag;

}

// 找出三元组T中最大值并将其赋值给e

ElemType getMax(Tuple &T, ElemType &e)

{

int i;

e = T[0];

for (i=1; i<N; i++)

if (T[i] > e)

e = T[i];

return e;

}

// 找出三元组T中最小值并将其赋值给e

ElemType getMin(Tuple &T, ElemType &e)

{

int i;

e = T[0];

for (i=1; i<N; i++)

if (T[i] < e)

e = T[i];

return e;

}

// 两个N元组相加

void add(Tuple T1, Tuple &T2)

{

int i;

for (i=0; i<N; i++)

T2[i] += T1[i];

}

// N元组的各分量同乘比例系数

void mulCoef(Tuple &T1, float coef)

{

int i;

for (i=0; i<N; i++)

T1[i] \*= coef;

}

// 输出N元组

void printTriplet(Tuple T)

{

int i;

for (i=0; i<N; i++)

{

printf("%.1f\t", T[i]);

}

printf("\n");

}

（3）在主函数中调用相关函数完成“N元组”的基本操作，主函数如下：

int main()

{

Tuple T1, T2;

int funNum, i;

\_Bool flag;

ElemType v[N], e, coef;

printf("----N元组操作主菜单----\n");

printf("1.N元组初始化\n");

printf("2.输出三元组\n");

printf("3.用e获取T的第i个元素的值\n");

printf("4.置t的第i元的值为e\n");

printf("5.升序排列\n");

printf("6.降序排列\n");

printf("7.取最大值\n");

printf("8.取最小值\n");

printf("9.输入另一个三元组相加\n");

printf("10.N元组元素倍增或倍减\n");

printf("11.退出系统\n");

while(1)

{

printf("请输入你想完成的功能编号:\n");

scanf("%d", &funNum);

switch (funNum)

{

case 1: printf("请输入N元组的每个元素(N=%d)\n", N);

for (i=0; i<N; i++)

scanf("%f", &v[i]);

initTriplet(T1, v);

break;

case 2: printf("初始化好的N元组为：\n");

printTriplet(T1);

break;

case 3: printf("请输入要获取第几个元素的值：\n");

scanf("%d", &i);

getElem(T1,i,e);

printf("第%d个元素的值为：%.1f\n",i,e);

break;

case 4: printf("请输入要改变的位置和更改的值：\n");

scanf("%d%f", &i, &e);

putElem(T1,i,e);

printf("新的N元组为：\n");

printTriplet(T1);

break;

case 5: flag = isAscending(T1);

if (flag == 1) printf("升序排列\n");

else printf("非升序排列\n");

break;

case 6: flag = isDescending(T1);

if (flag == 1) printf("降序排列\n");

else printf("非降序排列\n");

break;

case 7: getMax(T1,e);

printf("最大值为%.1f\n", e);

break;

case 8: getMin(T1,e);

printf("最小值为%.1f\n", e);

break;

case 9: printf("请输入另一个N元组的每个元素(N=%d)\n", N);

for (i=0; i<N; i++)

scanf("%f", &v[i]);

initTriplet(T2, v);

add(T1,T2);

printf("相加之后的N(N=%d)元组为：\n", N);

printTriplet(T2);

break;

case 10: printf("请输入所乘的系数\n");

scanf("%f", &coef);

mulCoef(T1, coef);

printf("每项乘系数后的N元组为：\n");

printTriplet(T1);

break;

case 11: destroyTuple(T1);

destroyTuple(T2);

break;

default: printf("没有你想完成的功能\n"); break;

}

}

return 0;

}

**1.4开发高效算法**

将解决实际问题的方法变成计算机可以运行的程序，中间的桥梁就是计算机的算法。 一个优秀的软件工程师与平庸的程序员的差别就在于：前者总是不断寻找并且有能力找到好的算法，而后者仅常常满足于勉强解决问题。而在所有的“好”算法中，显然存在一个最优的算法，找到它们是从事软件工程领域的人应该努力的目标。

### 1.4.1 算法的基本概念

**1、算法**

算法是对特定问题求解步骤的一种描述，是指令的有限序列。

**2、算法的特点**

一个算法具有以下5个特性：

1. 有穷性（Finiteness）：一个算法必须总是（对任何合法的输入值）保证执行有限步之后结束，且每一步都可在有穷时间内完成。
2. 确定性（Definiteness）：算法中每一条指令必须有确切的含义，读者理解或者计算机运行时不会产生二义性。
3. 可行性（Effectiveness）：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步骤，即每个计算步骤都可以在有限时间内完成（也称之为有效性）。
4. 有0个或多个输入：以刻画运算对象的初始情况，所谓0个输入是指算法本身定出了初始条件。

（5）有一个或多个输出：以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；

**3、算法的设计要求**

一个优秀的算法应该具备的设计要求主要有以下几点：

（1）正确性

算法的正确性是指算法至少具有输入，输出和加工处理无歧义，并且可以正确反映问题的需求，以及正确得到问题的答案。关于“正确”的理解一共有四个层次：

* 算法程序没有语法错误。
* 算法程序能够根据正确的输入的值得到满足要求的输出结果。
* 算法程序能够根据错误的输入的值得到满足规格说明的输出结果。
* 算法程序对于精心设计的，极其刁难的测试数据都能满足要求的输出结果。

（2）可读性

算法设计的另一个目的是为了便于阅读，理解和沟通。可读性好有助于人对算法的理解。

（3）健壮性

当输入的数据不合法的时候，算法也能给出相关的处理，而不是产生异常或者莫名起码的错误。

（4）时间效率高和空间存储量低

在满足以上几点以后，我们还可以考虑对算法进一步优化，尽量满足时间效率高和空间存储量低的需求。

**4、算法的评价**

对于计算机算法来说，虽然衡量其好坏的标准非常多，比如运算速度、对存储量的要求、是否容易理解、是否容易实现等，但是为了便于公平的比较，我们需要一个客观的标准，这个标准就是算法的复杂程度。算法的复杂程度主要从[时间复杂度](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/_blank)和[空间复杂度](https://baike.baidu.com/item/%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/_blank)来考虑。

（1）时间复杂度（Time Complexity）

算法中基本操作重复执行的次数是问题规模n的某个函数f(n)，算法的时间复杂度也因此记做T(n)=Ο(f(n))。表示随着问题规模n的增大，算法执行时间的增长率与f(n) 的增长率正相关，称作[渐进时间复杂度](https://baike.baidu.com/item/%E6%B8%90%E8%BF%9B%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/_blank)，简称时间复杂度。

（2）空间复杂度（Space Complexity）

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度，其计算和表示方法与时间[复杂度](https://baike.baidu.com/item/%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/_blank)类似，一般都用复杂度的渐近性来表示，记做S(n)=O(f(n))。

### 1.4.2 确定各种类型算法的复杂度

**1、时间复杂度**

算法的时间复杂度通常用“大O记法”来描述，即T(n)=Ο(f(n))。在实际计算时，如果输入数据集出现的概率难以确定，无法计算平均时间复杂度，这时通常会用最坏情况的算法时间复杂度来表示最终的时间复杂度。

时间复杂度的计算过程中，可以忽略函数中的倍乘常数和非主导项。如：

若c是常数，则O(c\*f(n)) = O(f(n))，是因为忽略了常系数c。

若c是常数，a>b>0，则O(cna+nb) = O(na) ，是因为忽略了常系数c和低次项nb。

常用的时间复杂度所耗费的时间从小到大依次是：

O(1)≤O(log2n)≤O(n)≤O(n2)≤O(n3)≤O(2n)≤O(n!)

其中，常数阶O(1)、对数数O(log2n)、线性阶O(n)、平方阶O(n2)和立方阶O(n3)是常用的算法，指数阶O(2n)和阶乘阶O(n!)只有当n足够小才是可使用的算法。需要注意的是：对于常数阶，只要程序中语句的执行频度是常数，无论这个常数多大，都记作O（1）。

**【示例1\_12】分析以下程序段的时间复杂度：**

for (i=1; i<n; i++)

{

y = y + 1;     // ①

for (j=0; j<=(2\*n); j++)

x++;       // ②        
    }

**分析过程：**

语句①的频度是n-1

语句②的频度是(n-1)\*(2n+1)=2n2-n-1

f(n)=n-1+2n2-n-1=2n2-2，其中常系数和常数项可以忽略  
        所以，该程序段的时间复杂度T(n)=O(n2).

**【示例1\_13】分析以下程序段的时间复杂度：**

i=1;      // ①

while (i<=n)  
       i=i\*2; // ②  
**分析过程：**

语句①的频度是1  
        设语句②的频度是x,  则：2x<=n; x<=log2n  取最大值log2n    
         f(n)=1+log2n，其中常数项可以忽略  
         所以，该程序段的时间复杂度T(n)=O(log2n )

1. **空间复杂度**

与时间复杂度类似，通常使用S(n)=Ο(f(n))来描述算法的空间复杂度。一个算法在计算机中所占用的存储空间，包括三部分：

1. 存储算法本身所占用的存储空间。本部分与算法书写的长短成正比，要压缩这方面的存储空间，就必须编写较短的算法。
2. 算法的输入输出数据所占用的存储空间。本部分是由要解决的问题决定的，是通过参数表由调用函数传递而来的，它一般不随算法的不同而改变。
3. 算法在运行过程中临时占用的存储空间。本部分会随算法的不同而异，有的算法只需要占用少量的临时工作单元，而且不随问题规模的大小而改变，我们称这种算法是“就地"进行的，空间复杂度为0(1)，是节省存储的算法。有的算法需要占用的临时工作单元数与解决问题的规模n有关，它随着n的增大而增大，当n较大时，将占用较多的存储单元，例如后续章节介绍的快速排序和[归并排序算法](https://baike.baidu.com/item/%E5%BD%92%E5%B9%B6%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95" \t "https://baike.baidu.com/item/%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6/_blank)就属于这种情况。

在实际计算时，若输入数据所占空间只取决于问题本身，和算法无关，那么只需要分析算法在运行过程中临时占用的存储空间即可。

**【示例1\_14】分析下面完成冒泡排序的算法的时间复杂度和空间复杂度。**

void bubbleSort(int array[], int n)

{

int i, j, tmp,flag;

if (n <= 1) return;

for (i = n-1; i > 0; i-- ){

flag = 0; // 变量flag用来标记本趟排序是否发生了交换

for (j = 0; j < i; j++){

if (array[j] > array[j+1]){

tmp = array[j];

array[j] = array[j+1];

array[j+1] = tmp;

flag = 1; // 没发生交换flag=0不变, 发生了交换flag变为1

}

}

if (flag == 0) return; //

}

}

**分析过程：**

1. 时间复杂度的计算

因为计算时间复杂度时，常数项可以忽略，所以可选取最内层循环中的关键字交换操作为基本操作，又因为输入数据的情况不一样，基本操作的执行频度也不一样，所以以最坏情况来计算本算法的时间复杂度。本算法的最坏情况为待排序列为逆序，此时，对于外层循环的每次执行，内层循环中if语句的条件均成立，即基本操作的执行次数为n-i，i的取值为1~n-1。因此，基本操作总的执行次数为(n-1+1)(n-1)/2=n(n-1)/2，其中常系数和低次项可以忽略。

所以，该冒泡排序算法的时间复杂度为T(n)=O(n2)。

1. 空间复杂度的计算

由算法代码可以看出，算法在运行过程中临时占用的存储空间只有一个temp，与问题规模n没有关系，所以该冒泡排序算法的空间复杂度为S(n)=O(1)。

### 1.4.3使用动态编程设计寻找斐波那契数的高效算法

动态规划程序设计是求解最优化问题的一种途径、一种方法，而不是一种特殊算法。其主要思想就是将一个复杂的问题分解成多个子问题，将子问题的解结合在一起就构成了原问题的解。由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划方法，可以解决各类最优化问题，必须[具体问题具体分析](https://baike.baidu.com/item/%E5%85%B7%E4%BD%93%E9%97%AE%E9%A2%98%E5%85%B7%E4%BD%93%E5%88%86%E6%9E%90" \t "https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92/_blank)处理，以丰富的想象力去建立模型，用创造性的技巧去求解。本节我们将使用动态编程方法寻找斐波那契数。

**【示例1\_15】这是一个有趣的古典数学问题，著名意大利数学家Fibonacci曾提出一个问题：有一对小兔子，从出生后第3个月起每个月都生一对兔子。小兔子长到第3个月后每个月又生一对兔子。按此规律，假设没有兔子死亡，第一个月有一对刚出生的小兔子，问第n个月有多少对兔子？(假设每对兔子一雌一雄且不病不死)**

Input

输入月数n（1<=n<=44）

Output

输出第n个月有多少对兔子

**解题方案一：**

在本题中，共谈到了两种类型的兔子，小兔子和大兔子，我们不妨逐月来分析一下：第一个月只有一对初生的小兔子，兔子总对数是1；第二个月小兔子长成了大兔子，兔子总对数是1；第三个月大兔子生下了小兔子，大兔子还在，兔子总对数是2；第四个月大兔子生下了小兔子，原来的小兔子长成了大兔子，大兔子还在，兔子总对数是3；第五个月，也可以同样分析得到兔子总对数是5；继续这样分析下去，可以得到每个月小兔子对数、大兔子对数和兔子总对数如表1\_1所示。

表1\_1 每个月小兔子对数、大兔子对数和兔子总对数情况表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 经过月数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
| 小兔子对数 | 1 | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | ... |
| 大兔子对数 | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | ... |
| 总对数 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | ... |

从表中可以看出，兔子的总对数构成了斐波那契数列，马上我们可以想到用斐波那契数列的递推公式fib1(n)=fib1(n-1) + fib2(n-2)(n≥3)来完成算法。完整算法如下：

int fib1(int n)

{

if (n < = 2) return n;

else return fib1(n-1) + fib2(n-2);

}

本算法最大的问题是在运算的过程中各递归实例均被多次调用。计算数列第5项的过程如图1.1所示。

fib(2)

fib(5)

fib(4)

fib(3)

fib(3)

fib(2)

fib(1)

fib(2)

fib(1)

图1.1计算斐波那契数列第5项的过程

接下来，我们来分析一下本算法的时间复杂度，用c代表计算第一项和第二项的频度之和。

T(n) = T(n-1) + T(n-2) + c

≤ 2T(n-1) + c

≤ 2(2T(n-2) + c) + c

= 22T(n-2) + 2c + c

≤ ...

= 2n-1T(1) + 2n-2c + ... +2c + c

≤ (2n-1 + 2n-2 + ... +2 + 1)c

= (2n-1)c

忽略掉常系数和常数项，所以本算法的时间复杂度为：T(n) = O(2n)

**解题方案二：**

方案1中，使用递归方式从后往前计算，从编译的角度，这种尾递归方式均可以转换为循环来处理，接下来我们从前往后来看问题的解决思路，假设小兔子对数是f1，大兔子对数是f2，总兔子对数是fn，每个月小兔子对数、大兔子对数和兔子总对数如表1\_2所示。

表1\_2 每个月小兔子对数、大兔子对数和兔子总对数情况表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 经过月数n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
| 小兔子对数f1 | 1 | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | ... |
| 大兔子对数f2 | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | ... |
| 总对数fn | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | ... |

从表中可发现，每个月的兔子总对数fn = f1 + f2，而f1是上个月的f2，f2是上个月fn。根据这个发现，我们马上可以写出完整的算法如下：

int fib2(int n)

{

int f1 = 1, f2 = 1, fn = 1, i;

for(i=3; i<=n; i++)

{

fn = f1 + f2;

f1 = f2;

f2 = fn;

}

return fn;

}

本算法中只涉及到一个单重循环，算法的时间复杂度很显然为：T(n) = O(n)，比方案1的算法好很多。

## **1.5习题及答案解析**

### 1.5.1习题

一.单选题

1、数据结构的逻辑结构被形式化地定义为一个二元组（D,S），其中D是（ ① ）的有限集合，S是D上（ ② ）的有限集合。

[A、数据 映射](https://mooc1-2.chaoxing.com/work/javascript:void(0))

B、数据元素 关系

C、算法 操作

数据操作 存储

考核知识点：数据结构的定义和形式化定义

支撑课程目标：CO1(CO1.1)

正确答案：B

答案解析：举个例子便于大家理解：如包含6个元素（a1,a2,a3,a4,a5,a6）的一维数组可以用这样的二元组形式化的描述为：D={a1,a2,a3,a4,a5,a6}，S={<ai,ai+1>|i=1,2,3,4,5}。

2、在数据结构中，从逻辑上可以把数据结构分成（）。

A、动态结构和静态结构

B、紧凑结构和非紧凑结构

C、线性结构和非线性结构

D、内部结构和外部结构

考核知识点：数据的逻辑结构（重点）

支撑课程目标：CO1(CO1.1)

正确[答案：C](https://mooc1-2.chaoxing.com/work/javascript:void(0))

[答案解析：线性表属于线性结构，集合、树和图属于非线性结构。](https://mooc1-2.chaoxing.com/work/javascript:void(0))

3、以下说法正确的是（   ）。

A、数据元素是数据的最小单位

B、数据项是数据的基本单位

C、数据结构是带有结构的各数据项的集合

D、一些表面上很不相同的数据可以有相同的逻辑结构

考核知识点：数据结构的基本概念

支撑课程目标：CO1(CO1.1)

正确[答案：D](https://mooc1-2.chaoxing.com/work/javascript:void(0))

[答案解析：](https://mooc1-2.chaoxing.com/work/javascript:void(0))A、B、C不对，正确答案是：数据项是数据的最小单位；数据元素是数据的基本单位；数据结构是带有结构的各数据元素的集合；D正确，如：windows系统中文件的组织架构图和一个单位的组织架构图，表面上是不相同的数据，但其实都是树形结构。

4、在存储数据时，通常不仅要存储各数据元素的值，而且还要存储（    ）。

A、数据的处理方法

B、数据的存储方法

C、数据元素的类型

D、数据元素之间的关系

考核知识点：数据的存储结构（重点）

支撑课程目标：CO1（CO1.1）

[答案：D](https://mooc1-2.chaoxing.com/work/javascript:void(0))

[答案解析：](https://mooc1-2.chaoxing.com/work/javascript:void(0))存储数据时，或者用顺序方式来描述数据元素之间的关系，或者用链式来描述数据元素之间的关系。

5、在数据结构中，与所使用的计算机无关的是数据的（）结构。

A、逻辑

B、存储

C、逻辑和存储

D、物理

考核知识点：数据的逻辑结构（重点）

支撑课程目标：CO1（CO1.1）

正确答案：A

答案解析：数据的逻辑结构反映的是数据元素之间的逻辑关系，与使用的计算机无关：数据的物理结构，又称存储结构，是指数据在计算机中的表示，它包括数据元素的值和元素的关系，其中数据元素之间的关系在计算机中主要有顺序存储结构和链式存储结构两种。

二.程序填空题

有一段程序如下，p指针指向的结构体数组中存储了一个班级的学生信息，请将程序补充完整。

typedef struct Student

{

int no;

char name[20];

float score1;

float score2;

} Student;

void input(Student \*);

void print(Student \*);

int count; //班级人数

int main()

{

Student \*p;

input(p);

print(p);

return 0;

}

// 数据输入

void input(Student \*q)

{

//完成了班级学生信息的录入，此处省略，不需大家写！

}

void print(Student \*q)

{

printf("班级全体信息如下：\n");

printf("学号 姓名 成绩1 成绩2:\n");

int i;

for(i=0; i<count; i++)

{

    //此处省略了一句话，请写出。完成将学生信息输出！；

}

}

考核知识点：抽象数据类型的定义、表示和实现

支撑课程目标：CO1.1

正确答案：

printf("%d %s %f %f\n", q[i].no, q[i].name,q[i].score1, q[i].score2);

或者

printf("%d %s %f %f\n", (\*(q+i)).no, (\*(q+i)).name, (\*(q+i)).score1, (\*(q+i)).score2);

或者

printf("%d %s %f %f\n", (q+i)->no, (q+i)->name, (q+i)->score2,(q+i)->score2);

答案解析：

此语句位于循环中，所以输出语句中必须要出现循环变量i, 如访问第一个学生的学号，访问方式可以有三种：q[0].no，(\*q).no，q->no, 那么访问第i个学生的访问方式也同样有三种，即q[i].no,(\*(q+i)).no,(q+i)->no,所以上面三种答案均可，希望大家理解，在以后的章节中经常会用到！

1.5.2习题答案解析

一.单选题

1、数据结构的逻辑结构被形式化地定义为一个二元组（D,S），其中D是（ ① ）的有限集合，S是D上（ ② ）的有限集合。

[A、数据 映射](https://mooc1-2.chaoxing.com/work/javascript:void(0))

B、数据元素 关系

C、算法 操作

数据操作 存储

考核知识点：数据结构的定义和形式化定义

支撑课程目标：CO1(CO1.1)

正确答案：B

答案解析：举个例子便于大家理解：如包含6个元素（a1,a2,a3,a4,a5,a6）的一维数组可以用这样的二元组形式化的描述为：D={a1,a2,a3,a4,a5,a6}，S={<ai,ai+1>|i=1,2,3,4,5}。

2、在数据结构中，从逻辑上可以把数据结构分成（）。

A、动态结构和静态结构

B、紧凑结构和非紧凑结构

C、线性结构和非线性结构

D、内部结构和外部结构

考核知识点：数据的逻辑结构（重点）

支撑课程目标：CO1(CO1.1)

正确[答案：C](https://mooc1-2.chaoxing.com/work/javascript:void(0))

[答案解析：线性表属于线性结构，集合、树和图属于非线性结构。](https://mooc1-2.chaoxing.com/work/javascript:void(0))

3、以下说法正确的是（   ）。

A、数据元素是数据的最小单位

B、数据项是数据的基本单位

C、数据结构是带有结构的各数据项的集合

D、一些表面上很不相同的数据可以有相同的逻辑结构

考核知识点：数据结构的基本概念

支撑课程目标：CO1(CO1.1)

正确[答案：D](https://mooc1-2.chaoxing.com/work/javascript:void(0))

[答案解析：](https://mooc1-2.chaoxing.com/work/javascript:void(0))A、B、C不对，正确答案是：数据项是数据的最小单位；数据元素是数据的基本单位；数据结构是带有结构的各数据元素的集合；D正确，如：windows系统中文件的组织架构图和一个单位的组织架构图，表面上是不相同的数据，但其实都是树形结构。

4、在存储数据时，通常不仅要存储各数据元素的值，而且还要存储（    ）。

A、数据的处理方法

B、数据的存储方法

C、数据元素的类型

D、数据元素之间的关系

考核知识点：数据的存储结构（重点）

支撑课程目标：CO1（CO1.1）

[答案：D](https://mooc1-2.chaoxing.com/work/javascript:void(0))

[答案解析：](https://mooc1-2.chaoxing.com/work/javascript:void(0))存储数据时，或者用顺序方式来描述数据元素之间的关系，或者用链式来描述数据元素之间的关系。

5、在数据结构中，与所使用的计算机无关的是数据的（）结构。

A、逻辑

B、存储

C、逻辑和存储

D、物理

考核知识点：数据的逻辑结构（重点）

支撑课程目标：CO1（CO1.1）

正确答案：A

答案解析：数据的逻辑结构反映的是数据元素之间的逻辑关系，与使用的计算机无关：数据的物理结构，又称存储结构，是指数据在计算机中的表示，它包括数据元素的值和元素的关系，其中数据元素之间的关系在计算机中主要有顺序存储结构和链式存储结构两种。

# 第 七 章 图

## 本章实践目标

* 巩固图的基本知识及表示方式；
* 能够运用图的存储结构表示实际应用问题；
* 能够根据算法的复杂度对解决实际问题的方案进行比较与选择

## **本章实践体系概略**

**1 基础篇**

1.1 图的创建

* 邻接矩阵表示的图的创建
* 邻接表的表示的图的创建

1.2 图的深度优先遍历

* 邻接矩阵表示的图的遍历
* 邻接表的表示的图的遍历

1.3 图的广度优先遍历

* 邻接矩阵表示的图的遍历
* 邻接表的表示的图的遍历

**2 提高篇**

2.1 最小生成树算法

2.2 最短路径算法

2.3 拓扑排序算法

**3 拓展篇**

3.1 导航系统中的应用

3.2 工程预算中的应用