第1章 绪论

**一、复习要点**

本章主要讨论贯穿和应用于整个《数据结构》课程始终的基本概念和性能分析方法。学习本章的内容，将为后续章节的学习打下良好的基础。

本章复习的要点：

**1、**基本知识点

要求理解的概念包括：数据，数据对象，数据元素或数据成员，数据结构，数据类型，数据抽象，抽象数据类型，数据结构的抽象层次，面向对象，对象与类的关系，类的继承关系，对象间的消息通信等。需要对各个概念进行区分与比较。例如，按照面向对象建模技术的要求，把建立对象类作为一个层次，把建立对象间的关系（即建立结构）作为另外的层次。因此，在软件开发中做数据结构设计时，不但要设计对象–类，类的属性，类的操作，还要建立类的实例之间的关系。从这个角度考虑，把数据结构定义为数据对象及对象中各数据成员之间关系的集合是合理的。又例如，类**class**或**struct**与C中的结构类型**struct**的区别在于前者不但有对象的状态描述（数据成员），还加入了操作（成员函数），描述对象的行为，这样可以体现一个完整的实体概念，而后者不行。再例如，传统的数据结构概念从数据结构的逻辑结构、物理结构和相关操作等3个方面进行讨论。它反映了数据结构设计的不同层次：逻辑结构属于问题解决范畴，物理结构是逻辑结构在计算机中的存储方式。但在面向对象开发模式中，本课程中涉及的数据结构都属于基本数据结构，但有的属于应用级的数据结构，如稀疏矩阵，字符串，栈与队列，优先级队列，图等；有的属于实现级的数据结构，如数组，链表，堆，索引，散列表等。

**2、**有关算法的概念和简单的算法性能分析方法

算法的5个特性表明算法的实现属于面向过程的开发模式，即传统的“输入-计算-输出”模式。算法的应用要求明确算法的时间和空间代价。因此，必须理解算法的定义和算法的5个特性，掌握简单的时间复杂度估计和空间复杂度估计方法（不讨论程序复杂性）。

**3、**描述语言

要求数据结构的描述既要体现算法的逻辑，又要体现面向对象的概念，需要一种能够兼有面向对象和面向过程双重特性的描述语言。传统的Pascal语言和C语言只是面向过程的语言，不能适应面向对象的开发模式。因此，采用C++ 语言描述。要求学员基本掌握C++ 语言的基本概念和用C++ 语言编写应用程序的基本技术。例如，用类定义抽象数据类型的方式，定义模板类、抽象类的方法，函数与参数的定义，建立类（公有、私有）继承的方法，例外与异常的处理等等，都需要掌握。

**二、难点与重点**

1、基本概念：理解什么是数据、数据对象、数据元素、数据结构、数据的逻辑结构与物理结构、数据结构的抽象层次。

2、面向对象概念：理解什么是数据类型、抽象数据类型、数据抽象和信息隐蔽原则。了解什么是面向对象。

⮚ 抽象数据类型的封装性

⮚ Coad与Yourdon定义：面向对象 = 对象 + 类 + 继承 + 通信。

⮚ 面向对象系统结构的稳定性

⮚ 面向对象方法着眼点在于应用问题所涉及的对象

3、算法与算法分析：理解算法的定义、算法的特性、算法的时间代价、算法的空间代价。

⮚ 算法与程序的不同之处需要从算法的特性来解释

⮚ 算法的正确性是最主要的要求

⮚ 算法的可读性是必须考虑的

⮚ 程序的程序步数的计算与算法的事前估计

* 程序的时间代价是指算法的渐进时间复杂性度量

**三、教材中习题的解析**

1-1什么是数据? 它与信息是什么关系?

【解答】

什么是信息？广义地讲，信息就是消息。宇宙三要素（物质、能量、信息）之一。它是现实世界各种事物在人们头脑中的反映。此外，人们通过科学仪器能够认识到的也是信息。信息的特征为：可识别、可存储、可变换、可处理、可传递、可再生、可压缩、可利用、可共享。

什么是数据？因为信息的表现形式十分广泛，许多信息在计算机中不方便存储和处理，例如，一个大楼中4部电梯在软件控制下调度和运行的状态、一个商店中商品的在库明细表等，必须将它们转换成数据才能很方便地在计算机中存储、处理、变换。因此，数据(data)是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。在计算机中，信息必须以数据的形式出现。

1-2什么是数据结构? 有关数据结构的讨论涉及哪三个方面?

【解答】

数据结构是指数据以及相互之间的关系。记为：数据结构 = **{** D, R **}**。其中，D是某一数据对象，R是该对象中所有数据成员之间的关系的有限集合。

有关数据结构的讨论一般涉及以下三方面的内容：

1. 数据成员以及它们相互之间的逻辑关系，也称为数据的逻辑结构，简称为数据结构；

② 数据成员极其关系在计算机存储器内的存储表示，也称为数据的物理结构，简称为存储结构；

1. 施加于该数据结构上的操作。

数据的逻辑结构是从逻辑关系上描述数据，它与数据的存储不是一码事，是与计算机存储无关的。因此，数据的逻辑结构可以看作是从具体问题中抽象出来的数据模型，是数据的应用视图。数据的存储结构是逻辑数据结构在计算机存储器中的实现（亦称为映像），它是依赖于计算机的，是数据的物理视图。数据的操作是定义于数据逻辑结构上的一组运算，每种数据结构都有一个运算的集合。例如搜索、插入、删除、更新、排序等。

1-3数据的逻辑结构分为线性结构和非线性结构两大类。线性结构包括数组、链表、 栈、队列、优先级队列等; 非线性结构包括树、图等、这两类结构各自的特点是什么？

【解答】

线性结构的特点是：在结构中所有数据成员都处于一个序列中，有且仅有一个开始成员和一个终端成员，并且所有数据成员都最多有一个直接前驱和一个直接后继。例如，一维数组、线性表等就是典型的线性结构

非线性结构的特点是：一个数据成员可能有零个、一个或多个直接前驱和直接后继。例如，树、图或网络等都是典型的非线性结构。

1-4．什么是抽象数据类型？试用C++的类声明定义“复数”的抽象数据类型。要求

(1) 在复数内部用浮点数定义它的实部和虚部。

(2) 实现3个构造函数：缺省的构造函数没有参数；第二个构造函数将双精度浮点数赋给复数的实部，虚部置为0；第三个构造函数将两个双精度浮点数分别赋给复数的实部和虚部。

(3) 定义获取和修改复数的实部和虚部，以及+、-、\*、/等运算的成员函数。

(4) 定义重载的流函数来输出一个复数。

【解答】

抽象数据类型通常是指由用户定义，用以表示应用问题的数据模型。抽象数据类型由基本的数据类型构成，并包括一组相关的服务。

//在头文件complex.h中定义的复数类

#**ifndef** \_complex\_h\_

#**define** \_complex\_h\_

#**include** <iostream.h>

**class** comlex **{**

**public:**

complex ( )**{** Re = Im = 0**; }** //不带参数的构造函数

complex ( **double** r ) **{** Re = r**;** Im = 0**; }** //只置实部的构造函数

complex ( **double** r, **double** i ) **{** Re = r**;** Im = i**; }** //分别置实部、虚部的构造函数

**double** getReal ( ) **{ return** Re**; }** //取复数实部

**double** getImag ( ) **{ return** Im**; }** //取复数虚部

**void** setReal ( **double** r ) **{** Re = r**; }** //修改复数实部

**void** setImag ( **double** i ) **{** Im = i**; }**  //修改复数虚部

complex**& operator** = ( complex**&** ob) **{** Re = ob.Re**;** Im = ob.Im**; }** //复数赋值

complex**& operator** + ( complex**&** ob )**;** //重载函数：复数四则运算

complex**& operator** – ( complex**&** ob )**;**

complex**& operator** \* ( complex**&** ob )**;**

complex**& operator** / ( complex**&** ob )**;**

**friend ostream& operator** << ( **ostream&** os, complex**&** c )**;** //友元函数：重载<<

**private:**

**double** Re, Im**;**  //复数的实部与虚部

**};**

#**endif**

//复数类complex的相关服务的实现放在C++源文件complex.cpp中

#**include** <iostream.h>

#**include** <math.h>

#**include** “complex.h”

complex**&** complex **:: operator** + ( complex **&** ob ) **{**

//重载函数：复数加法运算。

complex result**;**

result.Re = Re + ob.Re**;** result.Im = Im + ob.Im**;**

**return** result**;**

**}**

complex**&** complex **:: operator** – ( complex**&** ob ) **{**

//重载函数：复数减法运算

complex result**;**

result.Re = Re – ob.Re**;**  result.Im = Im – ob.Im**;**

**return** result**;**

**}**

complex**&** complex **:: operator** \* ( complex**&** ob ) **{**

//重载函数：复数乘法运算

complex result**;**

result.Re = Re \* ob.Re – Im \* ob.Im**;** result.Im = Im \* ob.Re + Re \* ob.Im**;**

**return** result**;**

**}**

complex**&** complex **:: operator** / ( complex**&** ) **{**

//重载函数：复数除法运算

**double** d = ob.Re \* ob.Re + ob.Im \* ob.Im**;**

complex result**;**

result.Re = ( Re \* ob.Re + Im \* ob.Im ) / d**;** result.Im = ( Im \* ob. Re – Re \* ob.Im ) / d**;**

**return** result**;**

**}**

**friend ostream& operator** << ( **ostream&** os, complex **&** ob ) **{**

//友元函数：重载<<，将复数ob输出到输出流对象os中。

**return** os << ob.Re << ( ob.Im >= 0.0 ) **?** “+” : “-” << **fabs** ( ob.Im ) << “i”**;**

**}**

1-5 用归纳法证明：

 (1)

(2) 

(3) 

【证明】略

1-6 什么是算法? 算法的5个特性是什么? 试根据这些特性解释算法与程序的区别。

【解答】

通常，定义算法为“为解决某一特定任务而规定的一个指令序列。”一个算法应当具有以下特性：

① 有输入。一个算法必须有0个或多个输入。它们是算法开始运算前给予算法的量。这些输入取自于特定的对象的集合。它们可以使用输入语句由外部提供，也可以使用赋值语句在算法内给定。

② 有输出。一个算法应有一个或多个输出，输出的量是算法计算的结果。

③ 确定性。算法的每一步都应确切地、无歧义地定义。对于每一种情况，需要执行的动作都应严格地、清晰地规定。

④ 有穷性。一个算法无论在什么情况下都应在执行有穷步后结束。

⑤ 有效性。算法中每一条运算都必须是足够基本的。就是说，它们原则上都能精确地执行，甚至人们仅用笔和纸做有限次运算就能完成。

算法和程序不同，程序可以不满足上述的特性(4)。例如，一个操作系统在用户未使用前一直处于“等待”的循环中，直到出现新的用户事件为止。这样的系统可以无休止地运行，直到系统停工。

此外，算法是面向功能的，通常用面向过程的方式描述；程序可以用面向对象方式搭建它的框架。

1-7 设*n*为正整数, 分析下列各程序段中加下划线的语句的程序步数。

(1) **for** (**int** i = 1**;** i <= n**;** i++) (2) x = 0**;** y = 0**;**

**for** (**int** j = 1**;** j <= n**;** j++) **{ for** (**int** i = 1**;** i <= n**;** i++)

c[i][j] = 0.0**; for** (**int** j = 1**;** j <= i**;** j++)

**for** (**int** k = 1**;** k <= n**;** k++) **for** (**int** k = 1**;** k <= j**;** k++)

c[i][j] = c[i][j] + a[i][k] \* b[k][j]**;** x = x + y**;**

**}**

(3) **int** i = 1**,** j = 1**;** (4) **int** i =1**;**

**while** (i<=n **&&** j<=n) **{ do {**

i = i + 1**;** j = j + i**; for** (**int** j = 1**;** j <= n**;** j++)

**}** i = i + j**;**

**} while** ( i < 100 + n )**;**

【解答】

(1)



(2)

1. i = 1时，i = 2，j = j + i = 1 + 2 = 2 + 1，

i = 2时，i = 3，j = j + i = ( 2 + 1 ) + 3 = 3 + 1 + 2，

i = 3时，i = 4，j = j + i = ( 3 + 1 + 2 ) + 4 = 4 + 1 + 2 + 3，

i = 4时，i = 5，j = j + i = ( 4 + 1 + 2 + 3 ) + 5 = 5 + 1 + 2 + 3 + 4，

……

i = k时，i = k + 1，j = j + i = ( k + 1 ) + ( 1 + 2 + 3 + 4 + … + k )，

解出满足上述不等式的k值，即为语句i = i + 1的程序步数。



(4)



 一般地，

求出满足此不等式的k值，即为语句i = i + j的程序步数。

1-8 试编写一个函数计算n!\*2n的值，结果存放于数组A[arraySize]的第n个数组元素中，0 ≤ n ≤ arraySize。若设计算机中允许的整数的最大值为maxInt，则当n > arraySize或者对于某一个k (0 ≤ k ≤ n)，使得k!\*2k > maxInt时，应按出错处理。可有如下三种不同的出错处理方式：

(1) 用**cerr<<**及**exit** (1)语句来终止执行并报告错误；

(2) 用返回整数函数值0, 1来实现算法，以区别是正常返回还是错误返回；

(3) 在函数的参数表设置一个引用型的整型变量来区别是正常返回还是某种错误返回。

试讨论这三种方法各自的优缺点，并以你认为是最好的方式实现它。

【解答】

#**include** "iostream.h"#**define** arraySize 100#**define** MaxInt 0x7fffffff

**int** calc ( **int** T[ ], **int** n ) **{ int** i**,** edge = MaxInt / n / 2**;**

T[0] = 1**; if** ( n != 0 ) **{**

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

T[i] = T[i-1] \*i\*2**;**

**if** ( T[i] > edge ) **return** 0**;**

**}**

T[n] = T[n-1] \* n \* 2**;**

**}**

**cout** << "T[" << n << "]=" << T[n] << **endl;**

**return** 1**;**

**}**

**void** main ( ) **{ int** A[arraySize]**; int** i**;**

**for** ( i = 0**;** i < arraySize**;** i++ )

**if** ( !calc ( A, i ) ) **{**

**cout** << "failed at " << i << " ." << **endl;**

**break;**

**}**

**}**

1-9 (1) 在下面所给函数的适当地方插入计算count的语句：

**void** d (ArrayElement x[ ]**, int** n ) **{**

**int** i = 1**;**

**do {**

x[i] += 2**;**  i +=2**;**

**} while** (i <= n )**;**

i = 1**;**

**while** ( i <= (n/2) ) **{**

x[i] += x[i+1]**;** i++**;**

**}**

**}**

(2) 将由(1)所得到的程序化简。使得化简后的程序与化简前的程序具有相同的count值。

(3) 程序执行结束时的count值是多少？

(4) 使用执行频度的方法计算这个程序的程序步数，画出程序步数统计表。

【解答】

(1) 在适当的地方插入计算count语句

**void** d ( ArrayElement x [ ], **int** n ) **{**

**int** i = 1**;**

count ++**;**

**do {**

x[i] += 2**;**  count ++**;**

i += 2**;** count ++**;**

count ++**;** //针对**while**语句

**} while** ( i <= n )**;**

i = 1**;**

count ++**;**

**while** ( i <= ( n / 2 ) ) **{**

count ++**;** //针对**while**语句

x[i] += x[i+1]**;**

count ++**;**

i ++**;**

count ++**;**

**}**

count ++**;**  //针对最后一次**while**语句

**}**

(2) 将由(1)所得到的程序化简。化简后的程序与原来的程序有相同的count值：

**void** d ( ArrayElement x [ ], **int** n ) **{**

**int** i = 1**;**

**do {**

count += 3**;** i += 2**;**

**} while** ( i <= n )**;**

i = 1**;**

**while** ( i <= ( n / 2 ) ) **{**

count += 3**;** i ++**;**

**}**

count += 3**;**

**}**

(3) 程序执行结束后的count值为 3n + 3。

当n为偶数时，count = 3 \* ( n / 2 ) + 3 \* ( n / 2 ) + 3 = 3 \* n + 3

当n为奇数时，count = 3 \* ( ( n + 1 ) / 2 ) + 3 \* ( ( n – 1 ) / 2 ) + 3 = 3 \* n + 3

(4) 使用执行频度的方法计算程序的执行步数，画出程序步数统计表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 行 号 | 程 序 语 句 | 一次执行步数 | 执行频度 | 程序步数 |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **void** d ( ArrayElement x [ ], **int** n ) **{**  **int** i = 1**;**  **do {**  x[i] += 2**;**  i += 2**;**  **} while** ( i <= n )**;**  i = 1**;**  **while** ( i <= ( n / 2 ) ) **{**  x[i] += x[i+1]**;**  i ++**;**  **}**  **}** | 0  1  0  1  1  1  1  1  1  1  0  0 | 1  1  ⎣(n+1)/2⎦  ⎣(n+1)/2⎦  ⎣(n+1)/2⎦  ⎣(n+1)/2⎦  1  ⎣n/2+1⎦  ⎣n/2⎦  ⎣n/2⎦  ⎣n/2⎦  1 | 0  1  0  ⎣(n+1)/2⎦  ⎣(n+1)/2⎦  ⎣(n+1)/2⎦  1  ⎣n/2+1⎦  ⎣n/2⎦  ⎣n/2⎦  0  0 |
|  |  |  | ( n ≠ 0 ) | 3n + 3 |

**四、其他练习题**

1-10 填空题

（ A ）由某一数据对象和该对象中各个数据成员间的关系组成。依据所有数据成员之间关系的不同，（ A ）分为两大类：（ B ）和（ C ）。在（ B ）中的各个数据成员依次排列在一个线性序列中；（ C ）的各个数据成员不再保持在一个线性序列中，每个数据成员可能与零个或多个其他数据成员发生联系。

根据视点的不同，数据结构分为数据的（ D ）和（ E ）。（ D ）是面向问题的，（ E ）是面向计算机的。

【解答】

A：数据结构 B：线性结构 C：非线性结构

D：逻辑结构 E：存储结构

1-11 判断下列叙述的对错。如果正确，在题前打“√”，否则打“×”。

(1) 数据元素是数据的最小单位。

(2) 数据结构是数据对象与对象中数据元素之间关系的集合。

(3) 数据结构是具有结构的数据对象。

(4) 数据的逻辑结构是指各数据元素之间的逻辑关系，是用户按使用需要建立的。

(5) 算法和程序原则上没有区别，在讨论数据结构时二者是通用的。

【解答】

(1) × (2) √ (3) × (4) √ (5) ×

1-12 判断下列叙述的对错。如果正确，在题前打“√”，否则打“×”。

(1) 所谓数据的逻辑结构是指数据元素之间的逻辑关系。

(2) 同一数据逻辑结构中的所有数据元素都具有相同的特性是指数据元素所包含的数据项的个数都相等。

(3) 数据的逻辑结构与数据元素本身的内容和形式无关。

(4) 数据结构是指相互之间存在一种或多种关系的数据元素的全体。

(5) 从逻辑关系上讲，数据结构主要分为两大类：线性结构和非线性结构。

【解答】

(1) √ (2) × (3) √ (4) × (5) √

1-13 填空题

算法是一个有穷的指令集，它为解决某一特定任务规定了一个运算序列。它应当具有输入、输出、（ A ）、有穷性和可执行性等特性。

算法效率的度量分为（ B ）和（ C ）。（ B ）主要通过在算法的某些部位插装时间函数来测定算法完成某一规定功能所需的时间。而（ C ）不实际运行算法，它是分析算法中语句的执行次数来度量算法的时间复杂性。

程序所需的存储空间包含两个部分 （ D ）和（ E ）。（ D ）空间的大小与输入输出数据的个数多少，数值大小无关；（ E ）空间主要包括其大小与问题规模有关的成分变量所占空间，引用变量所占空间，以及递归栈所用的空间，还有在算法运行过程中动态分配和回收的空间。

【解答】

A：确定性 B：事后测量 C：事前估计

D：固定部分 E：可变部分

1-14 有下列几种用二元组表示的数据结构，试画出它们分别对应的图形表示（当出现多个关系时，对每个关系画出相应的结构图），并指出它们分别属于何种结构。

(1) A = (K, R)，其中

K = **{** a1, a2, a3, a4 **}**，R = **{ }**

(2) B = (K, R)，其中

K = **{** a, b, c, d, e, f, g, h **}**，R = **{** r **}**，

r = **{**<a,b>, <b,c>, <c,d>, <d,e>, <e,f>, <f,g>, <g,h> **}**

(3) C = (K, R)，其中

K = **{** a, b, c, d, e, f, g, h **}**，R = **{** r **}**，

r = **{**<d,b>, <d,g>, <b,a>, <b,c>, <g,e>, <g,h>, <e,f> **}**

(4) D = (K, R)，其中

K = **{**1, 2, 3, 4, 5, 6 **}**，R = **{** r **}**，

r = **{** (1,2), (2,3), (2,4), (3,4), (3,5), (3,6), (4,5), (4,6) **}**

【解答】

a1

a3

a4

a2

(1)

(2)

a

b

c

d

e

f

g

h

d

bd

a

c

gd

e

h

f

(3)

1

2

3

5

4

6

(4)

1-15 单选题

(1) 一个数组元素a[i] 与\_\_\_\_\_\_\_\_的表示等价。

A：\*(a+i) B：a+i C：\*a+i D：**&**a+i

(2) 对于两个函数，若函数名相同，但只是\_\_\_\_\_\_\_\_不同则不是重载函数。

A：参数类型 B：参数个数 C：函数类型

(3) 若需要利用形参直接访问实参，则应把形参变量说明为\_\_\_\_\_\_\_\_参数

A：指针 B：引用 C：值

(4) 下面程序段的时间复杂度为\_\_\_\_**\_\_**\_\_。

**for** ( **int** i = 0**;** i < m**;** i++ )

**for** ( **int** j = 0**;** j < n**;** j++ )

a[i][j] = i\*j**;**

A：O(m2) B：O(n2) C：O(m\*n) D：O(m+n)

(5) 执行下面程序段时，执行S语句的次数为\_\_\_\_\_\_\_\_。

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

**for** ( **int** j = 1**;** j <= i**;** j++ )

S**;**

A：n2 B：n2/2 C：n(n+1) D：n(n+1)/2

(6) 下面算法的时间复杂度为\_\_\_\_\_\_\_\_。

**int** f ( **unsigned int** n ) **{**

**if** ( n==0 || n==1 ) **return** 1**;**

**else return** n\*f (n-1)**;**

**}**

A：O(1) B：O(n) C：O(n2) D：O(n!)

【解答】

(1) A (2) C (3) B (4) C (5) D (6) B

1-16 填空题

(1) 数据的逻辑结构被分为\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_\_和\_\_\_\_\_\_\_四种。

(2) 数据的存储结构被分为\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_\_和\_\_\_\_\_\_\_四种。

(3) 在线性结构、树形结构和图形结构中，直接前驱和直接后继结点之间分别存在着\_\_\_\_\_\_\_、\_\_\_\_\_\_\_和\_\_\_\_\_\_\_的联系。

(4) 一种抽象数据类型包括\_\_\_\_\_\_\_和\_\_\_\_\_\_\_两个部分。

(5) 当一个传值型形式参数所占的体积较大时，应最好说明为\_\_\_\_\_\_\_，以节省参数值的传输时间和存储参数的空间。

(6) 当需要用一个形式参数直接改变对应实际参数的值时，则该形式参数应说明为\_\_\_\_\_\_。

(7) 在函数中对引用型形式参数的修改就是对相应\_\_\_\_\_\_\_\_的修改，而对\_\_\_\_\_\_\_\_型形式参数的修改只局限在该函数的内部，不会反映到对应的实际参数上。

(8) 当需要进行标准I/O操作时，则应在程序文件中包含\_\_\_\_\_\_\_\_头文件，当需要进行文件I/O操作时，则应在程序文件中包含\_\_\_\_\_\_\_头文件。

(9) 一个记录r理论上占有的存储空间的大小等于所有域\_\_\_\_\_\_\_\_，实际上占有的存储空间的大小即记录长度为\_\_\_\_\_\_\_\_。

(10) 一个数组a所占有的存储空间的大小即数组长度为\_\_\_\_\_\_\_\_，下标为i的元素a[i]的存储地址为\_\_\_\_\_\_\_，或者为\_\_\_\_\_\_\_。

(11) 函数重载要求\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_或\_\_\_\_\_\_\_有所不同。

(12) 对于双目操作符，其重载函数（非成员函数）带有\_\_\_\_\_\_个参数，其中至少有一个为\_\_\_\_\_\_的类型。

(13) 若对象ra和rb中至少有一个是属于用户定义的类型，则执行ra *==* rb时，需要调用\_\_\_\_\_\_\_重载函数，该函数的第一个参数应与\_\_\_\_\_\_\_的类型相同，第二个参数应与\_\_\_\_\_\_\_的类型相同。

(14) 从一维数组a[n] 中顺序查找出一个最大值元素的时间复杂度为\_\_\_\_\_\_\_\_，输出一个二维数组b[m][n]中所有元素值的时间复杂度为\_\_\_\_\_\_\_\_。

(15) 在下面程序段中，s = s+p语句的执行次数为\_\_\_\_\_\_\_\_，p \*= j语句的执行次数为\_\_\_\_\_\_\_，该程序段的时间复杂度为\_\_\_\_\_\_\_\_。

**int** i = 0**,** s = 0**;**

**while** ( ++i <= n ) **{**

**int** p = 1**;**

**for** ( **int** j = 1**;** j <= i**;** j++ ) p \*= j**;**

s = s + p**;**

**}**

(16) 一个算法的时间复杂度为(3*n*2+2*n*log2*n*+4*n*-7)/(5*n*)，其数量级表示为\_\_\_\_\_\_\_\_。

【解答】

(1) 集合结构、线性结构、树形结构、图形结构（次序无先后）

(2) 顺序结构、链接结构、索引结构、散列结构（次序无先后）

(3) 1 **:** 1、1 **:** N、M **:** N（或者1对1，1对N，M对N）

(4) 数据、操作（次序无先后）

(5) 引用型

(6) 引用型

(7) 实际参数的值、传值型

(8) iostream.h, fstream.h

(9) 长度之和，sizeof (r)

(10) sizeof(a)、a+i、(char\*) a+i\*sizeof(a[i])

(11) 参数类型、参数个数、排列次序（次序无先后）

(12) 二、用户定义

(13) **int operator** *==*、ra、rb

(14) O(n)、O(m\*n)

(15) n、n(n+1)/2、O(n2)

(16) O(n)

1-17 试计算以下程序所有语句的总执行次数。

(1) 非递归的求和程序

**float** sum ( **float** a[ ]**, int** n ) **{**

**float** s = 0.0**;**

**for** ( **int** i = 0**;** i < n**;** i++ )

s += a[i]**;**

**return** s**;**

**}**

(2) 递归的求和程序

**float** rsum ( **float** a[ ]**, int** n ) **{**

**if** ( n <= 0 ) **return** a[0]**;**

**else return** rsum ( a, n-1 ) + a[n-1]**;**

**}**

【解答】

(1) 2n+3 (2) 2n+1

1-18设有3个值大小不同的整数a、b和c，试编写一个C++函数，求

1. 其中值最大的整数；
2. 其中值最小的整数；
3. 其中位于中间值的整数。

【解答】

(1) 求3个整数中的最大整数的函数

【方案1】

**int** max ( **int** a**, int** b**, int** c ) **{**

**int** m = a**;**

**if** ( b > m ) m = b**;**

**if (** c > m ) m = c**;**

**return** m**;**

**}**

【方案2】(此程序可修改循环终止变量扩大到n个整数)

**int** max ( **int** a**, int** b**, int** c ) **{**

**int** data[3] = **{** a, b, c **};**

**int** m = 0**;** //开始时假定data[0]最大

**for** ( **int** i = 1**;** i < 3**;** i++ ) //与其他整数逐个比较

**if** ( data[i] > data[m] ) m = i**;** //m记录新的最大者

**return** data[m]**;**

**}**

1. 求3个整数中的最小整数的函数

可将上面求最大整数的函数稍做修改，“>”改为“<”，可得求最小整数函数。

【方案1】

**int** min ( **int** a**, int** b**, int** c ) **{**

**int** m = a**;**

**if** ( b < m ) m = b**;**

**if** ( c < m ) m = c**;**

**return** m**;**

**}**

【方案2】(此程序可修改循环终止变量扩大到n个整数)

**int** max ( **int** a**, int** b**, int** c ) **{**

**int** data[3] = **{** a, b, c **};**

**int** m = 0**;** //开始时假定data[0]最小

**for** ( **int** i = 1**;** i < 3**;** i++ ) //与其他整数逐个比较

**if** ( data[i] < data[m] ) m = i**;** //m记录新的最小者

**return** data[m]**;**

**}**

1. 求3个整数中具有中间值的整数

可将上面求最大整数的函数稍做修改，“>”改为“<”，可得求最小整数函数。

【方案1】

**int** min ( **int** a**, int** b**, int** c ) **{**

**int** m1 = a**,** m2**;**

**if** ( b < m1 ) **{** m2 = m1**;** m1 = b**; }**

**else** m2 = b**;**

**if** ( c < m1 ) **{** m2 = m1**;** m1 = c**; }**

**else if** ( c < m2 ) **{** m2 = c**; }**

**return** m2**;**

**}**

【方案2】(此程序可修改循环终止变量扩大到n个整数寻求次小元素)

**int** mid ( **int** a**, int** b**, int** c ) **{**

**int** data[3] = **{** a, b, c **};**

**int** m1 = 0, m2 = -1**;**  //m1指示最小整数, m2指示次小整数

**for** ( **int** i = 1**;** i < 3**;** i++ ) //与其他整数逐个比较

**if** ( data[i] < data[m1] ) **{** m2 = m1**;** m1 = i**; }** //原来最小变为次小, m1指示新的最小

**else if** ( m2 == -1 || data[i] < data[m2] ) m2 = i**;** //m2记录新的次小者

**return** data[m2]**;**

**}**

1-19 用C++函数编写一个算法，比较两个整数a和b的大小，对于a > b，a *==* b，a < b这三种不同情况应分别返回“>”，“=”和“<”字符。并求其时间复杂度。

【解答】

**char** Compare ( **int** a**, int** b ) **{**

**if** ( a > b ) **return** '>'**;**

**else if** ( a == b ) **return** '='**;**

**else return** '<'**;**

**}**

时间复杂度为O(1)

1-20 假定一维整型数组a[n]中的每个元素值均在［0, 200］区间内，用C++函数编写一个算法，分别统计出落在［0, 20)，［20, 50)，［50, 80)，［80, 130)，［130, 200］等各区间内的元素个数。

【解答】

**int** Count ( **int** a[ ], **int** n, **int** c[ ] ) **{**

//用数组c[5]保存统计结果

**int** d[5] = {20, 50, 80, 130, 201}**;** //用来保存各统计区间的上限

**int** i, j**;**

**for** ( i = 0**;** i < 5**;** i++ ) c[i] = 0**;** //给数组c[5]中的每个元素赋初值0

**for** ( i = 0**;** i < n**;** i++ ) **{**

**if** ( a[i] < 0 **||** a[i] > 200 )

**return** 0**;** //返回数值0表示数组中数据有错，统计失败。

**for** ( j = 0**;** j < 5**;** j++ ) //查找a[i]所在的区间

**if** ( a[i] < d[j] ) **break;**

c[j]++**;**  //使统计相应区间的元素增1

**}**

**return** 1**;**  //返回数值1表示统计成功

**}**

1-21 指出下列各算法的功能并求出其时间复杂度。

(1) **int** Prime ( **int** n ) **{**

**int** i = 2**,** x = (**int**) sqrt ( n )**;**

**while** ( i <= x ) **{**

**if** ( n % i == 0 ) **break;**

i++**;**

**}**

**if** ( i > x ) **return** 1**;**

**else return** 0**;**

**}**

(2) **int** sum1 ( **int** n ) **{**

**int** p = 1**,** s = 0**;**

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

**{** p \*= i**;**  s += p**; }**

**return** s**;**

**}**

(3) **int** sum2 ( **int** n ) **{**

**int** s = 0**;**

**for** ( **int** i = 1**;** i <= n**;** i++ ) **{**

**int** p = 1;

**for** ( **int** j = 1**;** j <= i**;** j++ ) p \*= j**;**

s += p**;**

**}**

**return** s**;**

**}**

(4) **int** fun ( **int** n ) **{**

**int** i = 1, s = 1**;**

**while** ( s < n ) s += ++i**;**

**return** i**;**

**}**

(5) **void** UseFile ( **ifstream&** inp**, int** c[ ] ) **{**

//假定inp所对应的文件中保存有n个整数。

**for** ( **int** i = 0**;** i < 10; i++ ) c[i] = 0**;**

**int** x**;**

**while** ( inp >> x )

**{** i = x%10**;**  c[i]++**; }**

**}**

(6) **void** mtable ( **int** n ) **{**

**for** ( **int** i = 1**;** i <= n**;** i++ ) **{**

**for** ( **int** j = i**;** j <= n**;** j++ )

**cout** << i << "\*" << j << "=" << setw(2) << i \* j << " "**;**

**cout** << **endl;**

**}**

**}**

(7) **void** cmatrix ( **int** a[ ][ ]**, int** M**, int** N**, int** d ) **{**

//M和N为全局整型常量

**for** ( **int** i = 0**;** i < M**;** i++ )

**for** ( **int** j = 0**;** j < N**;** j++ )

a[i][j] \*= d**;**

**}**

(8) **void** matrimult ( **int** a[ ][ ]**, int** b[ ][ ]**, int** c[ ][ ]**, int** M**, int** N**, int** L ) **{**

//数组a[M][N]、b[N][L]、c[M][L]均为整型数组

**int** i, j, k**;**

**for** ( i = 0**;** i < M**;** i++ )

**for** ( j = 0**;** j < L**;** j++ )

c[i][j] = 0**;**

**for** ( i = 0**;** i < M**;** i++ )

**for** ( j = 0**;** j < L**;** j++ )

**for** ( k = 0**;** k < N**;** k++ )

c[i][j] += a[i][k] \* b[k][j]**;**

**}**

【解答】

(1) 判断n是否是一个素数，若是则返回数值1，否则返回0。该算法的时间复杂性为O()。

(2) 计算的值。时间复杂性为O(n)。

(3) 计算的值。时间复杂性为O(*n2*)。

(4) 求出满足不等式1+2+3++i ≥ n的最小i值。时间复杂性为O()。

(5) 利用数组c[10] 中的每个元素c[i] 对应统计出inp所联系的整数文件中个位值同为i的整数个数。时间复杂性为O(n)。

(6) 打印出一个具有n行的乘法表，第i行（1≤i≤n）中有n-i+1个乘法项，每个乘法项为i与j（i≤j≤n）的乘积。时间复杂性为O(n2)。

(7) 使数组a[M][N]中的每一个元素均乘以d的值。时间复杂性为O(M×N)。

(8) 矩阵相乘，即a[M][N]×b[N][L]→c[M][L]。时间复杂性为O(M×N×L)。