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；

③ 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 = new complex ( Re + ob.Re,    Im + ob.Im );
      return *result;
}
complex& complex :: operator – ( complex& ob ) {
//重载函数：复数减法运算
      complex * result = new complex ( Re – ob.Re,    Im – ob.Im );
      return * result;
}
complex& complex :: operator * ( complex& ob ) {
//重载函数：复数乘法运算
```

```
        complex * result =
            new complex ( Re * ob.Re – Im * ob.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 = new complex ( ( Re * ob.Re + Im * ob.Im ) / d,
            ( 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) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad n \geq 1$

(2) $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad n \geq 1$

(3) $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}, \quad x \neq 1, \quad n \geq 0$

【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++)
  for (int j = 1; j <= n; j++) {
    c[i][j] = 0.0;
    for (int k = 1; k <= n; k++)
      c[i][j] = c[i][j] + a[i][k] * b[k][j];
  }
```

(2)
```
x = 0;   y = 0;
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= i; j++)
    for (int k = 1; k <= j; k++)
      x = x + y;
```

(3)
```
int i = 1,  j = 1;
while (i<=n && j<=n) {
  i = i + 1;   j = j + i;
}
```

(4)
```
int i =1;
do {
  for (int j = 1; j <= n; j++)
    i = i + j;
} while ( i < 100 + n );
```

【解答】

(1) $\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}1 = n^3$

(2) $$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1 = \sum_{i=1}^{n}\sum_{j=1}^{i} j = \sum_{i=1}^{n}\left(\frac{i(i+1)}{2}\right) = \frac{1}{2}\sum_{i=1}^{n} i^2 + \frac{1}{2}\sum_{i=1}^{n} i =$$
$$= \frac{1}{2}\frac{n(n+1)(2n+1)}{6} + \frac{1}{2}\frac{n(n+1)}{2} = \frac{n(n+1)(n+2)}{6}$$

(3) 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)，

$$\because j = (k+1) + \sum_{i=1}^{k} i \le n$$

$$\therefore (k+1) + \frac{k(k+1)}{2} = \frac{k^2+3k+3}{2} \le n$$

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) $i = 1$ 时，$i = 1 + \sum_{j=1}^{n} j = 1 + \frac{n(n+1)}{2}$

$i = 2$ 时，$i = \left(1 + \frac{n(n+1)}{2}\right) + \sum_{j=1}^{n} j = \left(1 + \frac{n(n+1)}{2}\right) + \frac{n(n+1)}{2} = 1 + 2\left(\frac{n(n+1)}{2}\right)$

$i = 3$ 时，$i = \left(1 + 2\left(\frac{n(n+1)}{2}\right)\right) + \sum_{j=1}^{n} j = 1 + 3\left(\frac{n(n+1)}{2}\right)$

一般地，

$i = k$ 时，$i = 1 + k\left(\frac{n(n+1)}{2}\right) < 100 + n$

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!*2^n 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!*2^k > 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, value = 1;
    if ( n != 0 ) {
        int edge = MaxInt / n / 2;
        for ( i = 1; i < n; i++ ) {
            value *= i*2;
            if ( value > edge ) return 0;
        }
        value *= n * 2;
    }
    T[n] = value;
    cout << "A[" << 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	**void** d (ArrayElement x [ ], **int** n) **{**	0	1	0
2	**int** i = 1**;**	1	1	1
3	**do {**	0	⌊(n+1)/2⌋	0
4	x[i] += 2**;**	1	⌊(n+1)/2⌋	⌊(n+1)/2⌋
5	i += 2**;**	1	⌊(n+1)/2⌋	⌊(n+1)/2⌋
6	**} while** (i <= n)**;**	1	⌊(n+1)/2⌋	⌊(n+1)/2⌋
7	i = 1**;**	1	1	1
8	**while** (i <= (n / 2)) **{**	1	⌊n/2+1⌋	⌊n/2+1⌋
9	x[i] += x[i+1]**;**	1	⌊n/2⌋	⌊n/2⌋
10	i ++**;**	1	⌊n/2⌋	⌊n/2⌋
11	**}**	0	⌊n/2⌋	0
12	**}**	0	1	0
			(n ≠ 0)	3n + 3

1-10 设有 3 个值大小不同的整数 a、b 和 c，试求

(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];
}
```

(2) 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];
}
```

(3) 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 mid ( 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];
}
```