武汉大学电子信息学院

2016 － 2017 学年第 2学期 期中考试

《数据结构与算法》试卷

专业 学号 姓名 成绩

**一、填空题（40空\*1分）**

1．编译C#源程序test.cs的命令是 csc test.cs ，生成的结果文件名是 test.exe 。C#的数据类型分为值类型和引用类型，值类型的变量本身包含他们的数据，而引用类型的变量包含的是数据的地址(引用) 。值类型包括 简单数据类型 ， 结构类型 ， 枚举类型 等，引用类型包括 类 ， 数组 等。C#的int类型的位宽是 32位 ，char类型的位宽是 16位 。为C#中定义的一维数组int[ ] x分配50个单元的语句为： x= new int[50] 。

2．数据结构包括数据的 逻辑结构 、数据的 存储结构 和数据的 操作 这三个方面的内容。

3．在线性结构中，第一个结点 没有 前驱结点，其余每个结点有且只有 一 个前驱结点；最后一个结点  
 没有 后续结点，其余每个结点有且只有­ 一 个后续结点。

4．一个算法的效率包含两方面的内容： 时间 效率和 空间 效率。

5．在n个结点的单向链表中要删除已知结点q，需找到它的 前驱结点 p，其时间复杂度为O( n )，关键语句为：p.Next = q.Next。

6．线性表、栈和队列都是 线性 结构，它们的英文名称分别是List 、Stack 和 Queue 。可以在线性表的 任意位置插入和删除元素；栈只能在栈顶\_插入和删除元素，其中插入操作称作 入栈Push , 删除操作称作 出栈Pop ；对于队列只能在 队尾 插入和 队首 删除元素，其中插入操作称作 入队Enqueue , 删除操作称作 出队Dequeue 。

7．定义string s=“420100199503301234”表示某人身份证号，则用表达式 s.Length 可以返回串中字符的个数，其结果等于 18 ， 用IndexOf方法定位第一个‘1’的下标的表达式是 s.IndexOf(‘1’) ，其结果等于 3 。表达式s.Substring(6, 4) 返回值的类型为 string ，其内容为 1995 。

**二、 综合题（65分）**

1.（3分）分析下列二重循环的时间复杂度。

int n = 9;

for(int i=1;i<=n;i\*=3)

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

Console.Write(i\*j);

**解答**：时间复杂度为O（n(log2n)）

2.（10分）写出下列程序段的输出结果，并结合泛型类的特性解释程序中表达式q.Enqueue('h')和q.Dequeue()的输入参数和返回值的类型。

public static void Main() {

Queue<char> q = new Queue<char>(); char x = 'e', y = 'c';

q.Enqueue('h'); q.Enqueue('r'); q.Enqueue(y);

x = q.Dequeue(); q.Enqueue(x);

x = q.Dequeue();q.Enqueue('a'); q.Enqueue(x);

foreach(char c in q) Console.Write(c);

Console.WriteLine();

}

**解答**：



1) 输出结果为："char"

2) 因为q 定义为Queue<char>，故q.Enqueue(‘h’)的输入参数类型为char，返回值类型为void。q.Dequeue()没有输入参数，返回值类型为char。

如果q定义为非泛型队列（System.Collections.Queue），则Enqueue的输入参数（形参）类型为object，q.Enqueue(‘h’) 将值类型实参（字符’h’） 装箱（boxing）成object型，入队，装箱是隐式的。Dequeue的返回值类型为object，x=(char)q.Dequeue()将队头的数据出队，拆箱（unboxing）赋值给x，拆箱必须是显式的。

3. （8分）编程实现在顺序表查找具有特定值的元素的操作，方法声明为：public int IndexOf(T k)。查找成功时返回k值首次出现位置，不成功则返回-1。

**解答**：

public int IndexOf(T k) {

int j = 0;

while( j<count && !k.Equals(items[j]) )

j++;

if(j>=0 && j<count)

return j;

else

return -1;

}

4 （8分）用**三元组表**表示稀疏矩阵（1），用**三元组的行单链表**存储稀疏矩阵（2）。写出表达式或画出示意图。

(1)  (2) 

**解答**：

**三元组表**表示：{(1, 0, 3), (1, 4, 8), (2, 3, 6), (4, 5, 5), (5, 0, 2)}

|  |  |  |
| --- | --- | --- |
| 行X | 列Y | 值item |
| 1 | 0 | 3 |
| 1 | 4 | 8 |
| 2 | 3 | 6 |
| 4 | 5 | 5 |
| 5 | 0 | 2 |

**三元组的行单链表**表示：



5. （9分）用递归方法实现阶乘运算**f**(*n*)=*n*!。试解释递归算法的特点。

**解答**：

public class Factorial {

//递归方法

public static int f(int n){

if(n==0) return 1;

else {

Console.WriteLine(n+"! = " + n + "\*" + (n-1) + "! ");

return n \* f(n-1);

}

}

public static void Main(string[] args){

int i=5;

Console.WriteLine(i + "! = " + f(i) );

} }

存在自调用的算法称为递归算法。递归算法就是为了得到问题的解，将问题推到比原问题更简单的解，然后再回到原问题上来。例如，求*f*(*n*) = *n*!，为计算*f*(*n*)，将它推到*f*(*n-*1)，即

*f*(*n*) = *n* × *f*(*n*-1)

而计算*f*(*n-*1)的算法与*f*(*n*)是一样的。

由此可见，对于较为复杂的问题，用递归算法求解具有下列特点：

• 如果原问题能够分解成几个相对简单且解法相同或类似的子问题时，只要解决了子问题，那么原问题就迎刃而解，这就是递归求解。例如，5！= 5×4！。

• 当分解后的子问题可以直接解决时，就停止分解。这些可以直接求解的问题称为递归结束条件。例如，1！= 1。

递归算法思路直接清晰，但是比非递归算法空间和时间的复杂度较高。

6. （8分）编程实现将10进制数n转换为16进制并在控制台输出结果。

**解答**：

public class DecOctConversion {

public static void Main(string[] args){

int n = 2468; int d = 16; Stack<int> s = new Stack<int>(20);

Console.Write("十进制数: {0} -> {1}进制:",n,d);

while (n!=0) {

s.Push( n%d );

n = n/d;

}

int i = s.Count;

while (i>0){

Console.Write("{0:x}",s.Pop()); i--;

}

Console.WriteLine();

} }

// 输出结果： 十进制数: 2468 -> 8 进制:4644 十进制数: 2468 -> 16 进制:9a4

7. （16分）编程实现顺序循环队列SequencedQueue类，要避免出现 “假溢出”现象。写出实现初始化、返回队列元素个数、入队、检测是否包含某数据等操作的类方法，这些方法具有下列签名：  
public SequencedQueue(int n)；public int Count;public void Enqueue(T k);public bool Contains(T k);

**解答**：

public class SequencedQueue<T> {

private T[] items;

private int front, rear; //front和rear为队列头尾下标

//构造长度为n的空队列

public SequencedQueue(int n) {

items = new T[n + 1];

front = rear = 0;

}

//返回队列的元素个数

public int Count { get { return (rear - front + items.Length) % items.Length; } }

//k值入队

public void Enqueue(T k) {

if (Full) DoubleCapacity();

items[rear] = k; rear = (rear + 1) % items.Length;

}

//检测是否包含某数据

public bool Contains(T k) {

int j;

for (int i = 0; i < Count; i++) {

j = (front + i) % items.Length;

if (items[j].Equals(k)) return true;

}

return false;

} }