|  |
| --- |
| 二〇一八 ～ 二〇一九 学年 第 1 学期  课程名称：**《数据结构》参考答案及评分标准**  命题教师： 试卷类型：A 试卷代号： |
| **一.选择题** (本大题共10题，每小题2分，共计20分)  1.A.2.A.3.D.4.C.5.A.6.D.7.C.8.B.9.D.10.B.  **二.填空题** (本大题共10题，每空1分，共计10分)  1.数组2.n;[log2n]+13.4;24.从任意节点出发都可以访问链表中的每一个元素5.L->next==L&&L->front==L6.1+n2+2n3+3n4+…+(m-1)nm7.（n-1)/28.顺序9.指针10.py->next=px->next; px->next=py  **三.简答题** (本大题共4题，每小题5分，共计20分)  1.q=p->next; p->next=q->next;free(q);  2.该算法的功能是：利用堆栈做辅助，将队列中的数据元素进行逆置。  3.参考答案：  // 求根结点到指定结点的路径过程中，采用了后跟遍历的思想，最终求得的路径保存在一个链栈中，其中根结点处于栈顶位置，指定结点处于栈底位置。  4.【解答】  ⑴ 边表中的结点个数之和除以2。  ⑵ 第i个边表中是否含有结点j。  ⑶ 该顶点所对应的边表中所含结点个数。  **四.编程题** (本大题共5题，每小题10分，共计50分)  1.【解答】  template &lt;class Type&gt;   　　ListNode &lt;Type&gt; \* List &lt;Type&gt; :: GetANode ( int i ) {  //取得单链表中第i个结点地址, i从0开始计数, i &lt; 0时返回指针0, i = 0时返回表头结点地址。  if ( i &lt; 1 ) return NULL;  ListNode &lt;Type&gt; \* p = first; int k = 0;  while ( p != NULL &amp;&amp; k &lt; i ) { p = p→link; k++; }  return p;  }  2.【解答】  循环队列类定义   #include <assert.h>   template <class Type> class Queue { //循环队列的类定义   public:    Queue ( int=10 );   ~Queue ( ) { delete [ ] Q; }   void EnQueue ( Type & item );   Type DeQueue ( );   Type GetFront ( );   void MakeEmpty ( ) { front = rear = tag = 0; } //置空队列   int IsEmpty ( ) const { return front == rear && tag == 0; } //判队列空否   int IsFull ( ) const { return front == rear && tag == 1; } //判队列满否   private:   int rear, front, tag; //队尾指针、队头指针和队满标志   Type \*Q; //存放队列元素的数组   int m; //队列最大可容纳元素个数   }   构造函数   template <class Type>  Queue<Type>:: Queue ( int sz ) : rear (0), front (0), tag(0), m (sz) {   //建立一个最大具有m个元素的空队列。   Q = new Type[m]; //创建队列空间   assert ( Q != 0 ); //断言: 动态存储分配成功与否   }   插入函数   template<class Type>   void Queue<Type> :: EnQueue ( Type &item ) {   assert ( ! IsFull ( ) ); //判队列是否不满，满则出错处理   rear = ( rear + 1 ) % m; //队尾位置进1, 队尾指针指示实际队尾位置   Q[rear] = item; //进队列   tag = 1; //标志改1，表示队列不空   }   删除函数   template<class Type>   Type Queue<Type> :: DeQueue ( ) {   assert ( ! IsEmpty ( ) ); //判断队列是否不空，空则出错处理   front = ( front + 1 ) % m; //队头位置进1, 队头指针指示实际队头的前一位置  tag = 0; //标志改0, 表示栈不满  return Q[front]; //返回原队头元素的值   }   读取队头元素函数   template<class Type>   Type Queue<Type> :: GetFront ( ) {   assert ( ! IsEmpty ( ) ); //判断队列是否不空，空则出错处理  return Q[(front + 1) % m]; //返回队头元素的值   }  3.参考答案：   public int countLeafNode(BiTreeNode T) {// 统计叶结点数目   int count = 0;   if (T != null) {   if (T.getLchild() == null && T.getRchild() == null) {   ++count;// 叶结点数增1   } else {   count += countLeafNode(T.getLchild()); // 加上左子树上叶结点数   count += countLeafNode(T.getRchild());// 加上右子树上的叶结点数   }   }   return count;   }  4.参考答案：  package ch05Exercise;  import ch05.BiTreeNode;//教材第5章中有此类的描述    public class Exercise5\_4\_1 {     public boolean isEqual(BiTreeNode T1, BiTreeNode T2) {  //判断两棵树是否相等，若相等则返回true,否则返回false   if (T1 == null && T2 == null)// 同时为空   return true;   if (T1 != null && T2 != null) // 同时非空进行比较   if (T1.getData().equals(T2.getData()))// 根结点数据元素是否相等   if (isEqual(T1.getLchild(), T2.getLchild())) // 左子树是否相等   if (isEqual(T1.getRchild(), T2.getRchild()))// 右子树是否相等   return true;   return false;   }   //测试主方法   public static void main(String[] args) {   // 创建根结点为T1的二叉树   BiTreeNode D1 = new BiTreeNode('D');   BiTreeNode G1 = new BiTreeNode('G');   BiTreeNode H1 = new BiTreeNode('H');   BiTreeNode E1 = new BiTreeNode('E', G1, null);   BiTreeNode B1 = new BiTreeNode('B', D1, E1);   BiTreeNode F1 = new BiTreeNode('F', null, H1);   BiTreeNode C1 = new BiTreeNode('C', F1, null);   BiTreeNode T1 = new BiTreeNode('A', B1, C1);   // 创建根结点为T2的二叉树   BiTreeNode D2 = new BiTreeNode('D');   BiTreeNode G2 = new BiTreeNode('G');   BiTreeNode H2= new BiTreeNode('H');   BiTreeNode E2 = new BiTreeNode('E', G2, null);   BiTreeNode B2 = new BiTreeNode('B', D2, E2);   BiTreeNode F2 = new BiTreeNode('F', null, H2);   BiTreeNode C2 = new BiTreeNode('C', F2, null);   BiTreeNode T2 = new BiTreeNode('A', B2, C2);   // 创建根结点为T3的二叉树   BiTreeNode E3= new BiTreeNode('E');   BiTreeNode F3 = new BiTreeNode('F');   BiTreeNode D3= new BiTreeNode('D',F3,null);   BiTreeNode B3 = new BiTreeNode('B', null, D3);   BiTreeNode C3 = new BiTreeNode('C', null, E3);   BiTreeNode T3 = new BiTreeNode('A', B3, C3);   Exercise5\_4\_1 e = new Exercise5\_4\_1();   if (e.isEqual(T1, T2))   System.out.println("T1、T2两棵二叉树相等！");   else   System.out.println("T1、T2两棵二叉树不相等！");   if (e.isEqual(T1, T3))   System.out.println("T1、T3两棵二叉树相等！");   else   System.out.println("T1、T3两棵二叉树不相等！");   }  }  5.#include <stdio.h>  typedef struct node {char data; struct node \*link; }node;  node \*select(node \*head)  {node \*p,\*q,\*r,\*s;   p=(node \*)malloc(sizeof(node));   p->link=head; head=p;  while(p->link!=null)   {q=p->link; r=p;   while ((1)\_\_\_\_\_\_\_\_\_\_\_\_)   { if (q->link->data<r->link->data) r=q;   q=q->link;   }  if ((2)\_\_\_\_\_\_\_\_\_\_\_) {s=r->link; r->link=s->link;   s->link= ((3)\_\_\_\_\_\_\_\_\_); ((4)\_\_\_\_\_\_\_\_\_);}   ((5)\_\_\_\_\_\_\_\_) ;   }  p=head; head=head->link; free(p); return(head);  } |