数据结构与程序设计部分（共 80 分）

一、判断下列叙述的正误。（每小题 1 分，共 10 分）

1. 线性表中所有数据元素的数据类型必须相同。
2. 线性表的顺序存储表示属于静态结构，链式存储表示属于动态结构。
3. 用两个 LIFO 栈可以模拟一个 FIFO 队列。反之，用两个 FIFO 队列也可以模拟一个 LIFO 栈。
4. 二维数组是一种非线性结构。
5. 广义表是线性表的推广，所以广义表是一种线性结构。
6. 一棵包含n个结点的完全二叉树，若其根结点属于第 0 层，则其高度为$\lfloor \log_2 n \rfloor$。
7. 当用深度优先搜索算法遍历一个连通图时，所经历的顶点和边构成一棵树。
8. 若表示一个有向图的邻接矩阵中对角线以下的元素均为 0，则该图的拓扑有序序列一定存在。
9. 从一个图的某个指定顶点出发进行广度优先搜索得到的广度优先生成树是唯一的。
10. 在一棵非空的二叉搜索树中删除一个结点后再将其插入，则所得的二叉搜索树与删除该结点前的二叉搜索树相同。

二、从语言规则和软件工程角度判断下列程序的正误并说明理由。（每小题 2 分，共 10 分）

1.
```
int getValue ( int * pValue ) {
    if ( pValue != NULL ) {
        return *pValue;
    }
}
```

2.
```
class Type_T {
public:
    Type_T ( int initValue = 0 );
    //……
};

Type_T::Type_T ( int initValue ) {
    //……
}
```

3.
```
class LinkedListNode {
protected:
    int data;
    LinkedListNode * link;
};
```

```
class LinkedList : public class LinkedListNode {
private:
    LinkedListNode * first;
    //……
}
```

4.
```
//……
for ( i = 0; i < length; i++ ) {
    if ( someCondition != 0 ) {
        i = newValue;
        //……
    }
}
```

5.
```
class MyClass_T {
    //……
    static int count ( void ) const;
}

MyClass_T * pMyobj = new MyClass_T;
//……
int i = pMyobj->count( );
```

三、填空题（每小空 1 分，共 5 分）

如果一棵有 n 个结点的满二叉树的高度为 h（树根所在的层次为 0），则其结点总数 n 可用高度 h 表示为（ A ），而高度 h 可用结点总数 n 表示为（ B ）；若对该树的结点从 0 开始按中序遍历次序进行编号，则树根结点的编号用 h 表示为（ C ），树根结点的左子女结点的编号用 h 表示为（ D ），右子女结点的编号用 h 表示为（ E ）。

四、算法设计题（每小题 5 分，共 15 分）

设中序线索化二叉树的类声明如下：

```
template <class Type> class ThreadNode {          //中序线索化二叉树的结点类
friend class ThreadTree;                          //中序线索化二叉树类
private:
    int leftThread, rightThread;                  //线索标志
    ThreadNode<Type> *leftChild, *rightChild;     //线索或子女指针
    Type data;                                    //结点中所包含的数据
public:
    int getLeftThread ( ) { return leftThread; }
    int getRightThread ( ) { return rightThread; }
    ThreadNode<Type> * getLeftChild ( ) { return LeftChild; }
    ThreadNode<Type> * getRightChild ( ) { return rightChild; }
```

```
    Type getData ( ) { return data; }
    ……
};

template <class Type> class inOrderThreadTree {      //中序线索化二叉树类
public:
    ThreadNode<Type> * getRoot ( ) { return root; }
    //其他公共成员函数
    ……
private:
    ThreadNode<Type> *root;                          //树的根指针
};
```

试依据上述类声明，分别编写下面的函数。

(1) ThreadNode<**Type**> * getPreorderFirst (ThreadNode<**Type**> *p);
//寻找以 p 为根指针的中序线索化二叉树在前序下的第一个结点。

(2) ThreadNode<**Type**> * getPreorderNext (ThreadNode<**Type**> *p)
//寻找结点*p 的在中序线索化二叉树中前序下的后继结点。

(3) **void** preorder (inOrderThreadTree<**Type**>& T);
//应用以上两个操作，在中序线索化二叉树上做前序遍历。

五、算法分析题（每小题 5 分，共 15 分）

下面给出一个算法，其中 n 是数组 A[ ]中元素总数。

```
template<class Type> void unknown ( Type a[ ], int n ) {
    int d = 1, j;
    while ( d < n /3 ) d = 3*d+1;
    while ( d > 0 ) {
      for ( int i = d; i < n; i++ ) {
          Type temp = a[i];
          j = i;
          while ( j >= d && a[j-d] > temp ) { a[j] = a[j-d];   j -= d; }
          a[j] = temp;
        }
        d /= 3;
    }
}
```

(1) 阅读此算法，说明它的功能。

(2) 对于下面给出的整数数组，追踪第一趟 **while** (d > 0) 内的每次 **for** 循环结束时数组中数据的变化。(为清楚起见，本次循环未涉及的不移动的数据可以不写出，每行仅写出一个 **for** 循环的变化)

(3) 以上各次循环的数据移动次数分别是多少。

步	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	移动次数
	77	44	99	66	33	55	88	22	44	11	
1											
2											
3											
4											
5											
6											

六、计算题（每小题 5 分，共 10 分）

如果有一个时间复杂性为$O(n^2)$的算法（如冒泡排序、选择排序或插入排序等），在有 200 个元素的数组上运行需要耗时 3.1 毫秒，试问在下列类似的数组上运行大约需要多长时间？

(1) 具有 400 个元素；

(2) 具有 40000 个元素。

七、算法设计题（每小题 5 分，共 15 分）

下面是队列类和栈类的声明：

```
template <class Type> class queue {
public:
    queue ( );                                  //队列的构造函数
    queue ( const queue& );                     //队列的复制构造函数
    queue& operator= ( const queue& );          //赋值操作
    bool isEmpty ( );                           //判断队列空否，=1 为空, =0 不空
    Type& getFront ( );                         //返回队头元素的值
    void push ( const Type& item);              //将新元素插入到队列的队尾
    void pop ( );                               //从队列的队头删除元素
    //……                                       //其他成员函数
}

template <class Type> class stack {
public:
    stack ( );                                  //栈的构造函数
    bool isEmpty ( );                           //判断栈空否。=1 栈空，=0 不空
    void push ( const stack& item );            //将新元素进栈
    void pop ( );                               //栈顶元素退栈
    Type& getTop ( );                           //返回栈顶元素的值
}
```

试利用栈和队列的成员函数，编写以下针对队列的函数的实现代码（**要求非递归实现**）。

(1) “逆转”函数 **template <class** Type> **void** reverse (queue<Type>**& Q);** （5 分）

(2) “判等”函数 bool queue**::operator**== (**const** queue**& Q);**（5 分）

(3) “清空”函数 **void** queue**::**clear (**);** （5 分）

数据结构与程序设计部分（共 80 分）

一、解答：（每小题 1 分，共 10 分）

1. 错误。线性表中所有数据元素的类型可以不同，但必须用链表存储。

2. 错误。线性表的顺序存储可以是一维静态数组，也可以是动态数组。

3. 正确。用两个 LIFO 栈可以模拟一个 FIFO 队列，而用两个 FIFO 队列也可以模拟一个 LIFO 栈。

4. 正确。二位数组中每个数组元素最多有两个直接前驱和两个直接后继。

5. 错误。广义表中的表元素可以是子表，所以它是一种非线性结构。

6. 错误。当n = 0 时，$\log_2 n$无意义。

7. 正确。在深度优先遍历连通图时访问了 n 个顶点以及连通它们的 n-1 条边，构成了一棵深度优先生成树。

8. 正确。若邻接矩阵对角线以下的元素均为 0，表明<v_i, v_j>存在时（i < j），<v_j, v_i>不存在，而且从v_j绕过其他顶点v_{p1}, v_{p2}, …, v_{pm}到达v_i的路径也不存在，因为即使从v_j能够到达v_{pm}（j < pm），但由于i < j < pm，所以<v_{pm}, v_i>不存在，因此有向图中没有回路，故按拓扑排序方法必定能够得到其拓扑有序序列。

9. 错误。可能得到不同的广度优先生成树。

10. 错误。二叉搜索树的新结点都是作为叶结点插入的。除非删除的是叶结点，否则删除后又插入，二叉搜索树的形态是不同的。

二、解答：（每小题 2 分，共 10 分）

1. 错，else 情况没有返回值

2. 对，默认值应当在类声明内成员函数原型定义中给出，不在成员函数实现时给出。

3. 错，LinkedList 公共继承 LinkedListNode，表明 LinkedList is a LinkedListNode。

4. 错，在 for 循环的循环体内部修改了循环控制变量的值。

5. 错，静态成员应当用 i = MyClass_T**::**count( )，不要用 i = pMyobj−>count( ) 方式。

三、解答：（每小空 1 分，共 5 分）

A. $n = 2^{h+1} - 1$

B. $h = \log_2(n+1) - 1$ 或$\log_2((n+1)/2)$

C. $2^h - 1$

D. $2^{h-1} - 1$

E. $3*2^{h-1} - 1$

四、解答：（每小题 5 分，共 15 分）

(1) ThreadNode<**Type**> * getPreorderFirst (ThreadNode<**Type**> *p) **{**

return p;

}

(2) ThreadNode<**Type**> * getPreorderNext (ThreadNode<**Type**> *p) **{**

if (p->getLeftThread( ) == 0)

return p->getLeftChild( );

else if (p->getRightThread( ) == 0)

return p->getRightThread( );

else {

ThreadNode<**Type**> * q = p->getRightChild( );

while (q != NULL **&&** q->getRightThread( ) == 1)

q = q->getRightChild( );

if (q == NULL) **return** NULL;

else return q->getRightChild( );

}

}

(3) **void** preorder (inOrderThreadTree<**Type**>**&** T) **{**

ThreadNode<**Type**> *t = T.getRoot( ), *p;

p = getPreorderFirst (t);

while (p != NULL) {

cout << p->getData( ) << **endl;**

p = getPreorderNext (p);

}

}

五、解答：（每小题 5 分，共 15 分）

此算法为 shell 排序。

步	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	移动次数
	77	44	99	66	33	55	88	22	44	11	
1	33				77						3
2		44				55					2
3			88				99				3
4				22				66			3
5	33				44				77		3
6		11				44				55	4

六、解答：（每小题 5 分，共 10 分）

(1) 3.1*4 = 12.4 毫秒

(2) 3.1*40000 = 124000 毫秒 = 124 秒

七、解答：（每小题 5 分，共 15 分）

(1)
```
template <class Type> void reverse ( queue<Type>& Q ) {
        stack<Type> S;
        while ( !Q.isEmpty ( ) )
            { S.push ( Q.getFront ( ) );   Q.pop ( ); }
        while ( !S.isEmpty ( ) )
            { Q.push( S.getTop ( ) );   S.pop ( ); }
    };
```

(2)
```
template <class Type> bool queue<Type>::operator== (const queue<Type>& Q) {
        queue<Type> q1 = Q, q2 = *this;
        while ( !q1.isEmpty ( ) )
            if ( q1.getFront ( ) != q2.getFront ( ) ) return false;
            else { q1.pop ( );   q2.pop ( ); }
        return ture;
    };
```

(3)
```
template <class Type> void queue<Type>::clear ( ) {
        while ( !isEmpty ( ) ) pop ( );
    };
```