安徽大学 20__17__—20__18__学年第__2__学期

《 数据结构 》期中考试试题参考答案及评分标准

一、算法阅读题（本大题共 4 小题，每小题 5 分，共 20 分）

1．该算法中的基本运算是 x++和 y=3*x+2 语句。对于最外层的 for 循环，其执行频度为 n+1，但对于里层的 for 循环，只在 3i≤n（即 i≤n/3）时才执行，故基本运算的执行频度为：

$$\sum_{i=1}^{n/3}\sum_{j=3i}^{n}1=\sum_{i=1}^{n/3}(n-3i+1)=\frac{n(n-1)}{6}=O(n^2)$$

故本算法的时间复杂度为 $O(n^2)$

2．当 L 指向尾结点时 while 条件成立，再执行 L=L->next 时，则 L=NULL，此时 if 语句出现错误。修改后的算法如下：

```
int Count(LinkList L,ElemType x)
{   LNode *p=L->next;
    int n=0;
    while (p != NULL)
    {    if (p->data == x) n++;
         p = p->next;
    }
    return n;
}
```

3．(1) Fun(L,i,j)算法的功能是删除单链表 L 中从第 i 个结点到第 j 个结点的所有结点。

(2) 当 L=(1,2,3,4,5,6,7,8)时，执行 Fun(L,2,5)后，L=(1,6,7,8)。

4．(1) Fun(d)算法的功能是采用辗转相除法将十进制数 d 转换成 16 进制数，并用数组 b 存放 16 进制数串。

(2) 本程序的功能是采用辗转相除法将十进制数 1000 转换成 16 进制数并输出，其输出结果为 3E8。

二、简答题（每小题 5 分，共 20 分）

5．(1) 应选用链式存储结构。链式存储结构可以充分利用存储空间来存储线性表中的各数据元素，且其存储空间可以是连续的，也可以不连续；此外，在这种存储结构下，对元素进入插入和删除操作时无须移动元素，而仅修改指针即可，所以很适用于线性表容量变化的情况。

(2) 应选用顺序存储结构。由于顺序存储结构一旦确定了起始位置，线性表中的任何一个元素都可以进行随机存取，即存取速度较高。

6．栈的特点是后进先出，所以在解决的实际问题中涉及后进先出的情况时可考虑使用栈。如求解表达式括号匹配问题时通常使用一个栈，将读到的左括号进栈，每读入一个右括号时，判断栈顶是否为左括号，若是，则出栈；否则，表示不匹配。

队列的特点是先进先出。如求解如求解操作系统中的作用排队问题时，通常使用队列。因为在允许多道程序运行的计算机系统中同时有几个作业运行时，如果运行的结果都需要通过通道输出，那就要按请求输出的先后次序排队。每当通道传输完毕并可接受新的输出任务时，队头的作业先从队列中退出做输出操作（出队）。凡是申请输出的作业都从队尾进入队列（入队）。

7．(1) 三维数组 A[0..4][0..6][0..8]中元素的个数为：5×7×9=315

(2) A[3][5][7] 的存储地址为：100+[3×(7×9)+5×9+7] ×4=1064

8．在广义表 L 中取出原子的运算为：Head(Tail(Head(Tail()))），具体过程如下：

Tail(L)=((u,t,w));

Head(Tail(L))=((u,t,w)

Tail(Head(Tail(L)))= (t,w)

Head(Tail(Head(Tail(L))))=t

三、应用题（每小题 10 分，共 20 分）

9．每个字符对应的 next 和 nextval 函数值如下表所示：

j	1	2	3	4	5	6	7	8	9	10	11	12
模式串 t	a	b	c	a	a	b	b	a	b	c	a	b
next[j]	0	1	1	1	2	2	3	1	2	3	4	5
nextval[j]	0	1	1	0	2	1	3	0	1	1	0	5

10．OPTR 和 OPND 栈的具体变化过程如下表所示：

步骤	OPTR 栈	OPND 栈	读入字符	主要操作
1	#		3*(6+4)#	Push(OPND,’3’)
2	#	3	*(6+4)#	Push(OPTR,’*’)
3	#*	3	(6+4)#	Push(OPTR,’(’)
4	#*(	3	6+4)#	Push(OPND,’6’)
5	#*(	3 6	+4)#	Push(OPTR,’+’)
6	#*(-	3 6	4)#	Push(OPND,’4’)
7	#*(-	3 6 4	)#	Push(OPND,Operate(’6’, ’+’, ’4’))
8	#*(	3 10	)#	Pop(OPTR){消去一对括号}
9	#*	3 10	#	Push(OPND,Operate(’3’, ’*’, ’10’))
10	#	30	#	Return(GetTop(OPND))

四、算法设计题（共 40 分）

11． 该算法设计如下：

```
void Insert( LinkList &L, ElemType e)
{
    LNode *pre=L,*p,*s;
    if(!pre->next)   //若 L 为空链表，将 e 直接插入在头结点后面
    {    s=new LNode;   s->data = e;   s->next = NULL;    pre->next=s;
    }
    else      //若 L 为非空的有序单链表
    {    p=pre->next;              //p 为工作指针
         while(p != NULL)        //向后查找插入的位置
         {   if(p->data > e)        //找到，将 e 插入，使 s 后继为 p，s 前驱为 pre
              {    s=new LNode;     s->data = e;      pre->next = s;
                    s->nxet = p;         break;
              }
              else{       //继续向后查找
                   pre = p;       p=p->nxet;
              }
         }  //end while
    }
}
```

12． 该算法设计思想如下：定义两个指针变量 p 和 q，初始时均指向头节点的下一个节点。p 指针沿链表移动；当 p 指针移动到第 k 个节点时，q 指针开始与 p 指针同步移动；当 p 指针移动到链表最后一个节点时，q 指针所指元素为倒数第 k 个节点。

```
int Searchk(LinkList list,   int k)
{   LNode *p,*q;
    int count=0;
    p=q=list->next;
    while (p!=NULL)
    {   if (count<k)
            count++;
        else
            q=q->next;
        p=p->next;
    }
    if (count<k)   { cout << “链表 list 中无倒数第 k 个数据元素。”<< endl ;     return(0);}
    else
    {    cout << “链表 list 中倒数第 k 个数据元素：”<< q->data << endl ;
         return(1);
    }
}
```

13. 栈的特点是后进先出，队列的特点是先进先出。入队和出队算法设计如下：

（1）入队操作如下：

```
void EnQueue(Stack &s1,Stack &s2,ElemType e)
{    ElemType x;
     if(s1.top < maxsize )       //s1 未满，元素 e 入 s1 栈，表示队列的入队
          Push(s1,e);
     else        //s1 满
     {
          if(!StackEmpty(s2))     //s1 满,且 s2 非空时，不能入队
               cout<<"队列满，不能入队!"<<endl;
          else       //s1 满,且 s2 为空时
          {    while(!StackEmpty(s1))
               {
                    Pop(s1,x);     Push(s2,x);     //将 s1 全部退栈，再依次入栈 s2
               }
               Push(s1,e);  //再将元素 e 入栈到 s1，表示队列的入队
          }
     }
}
```

（2）出队操作如下：

```
void DeQueue(Stack &s1,Stack &s2,ElemType &e)
{
     if(!StackEmpty(s2))  //s2 不空，退栈，即为队列出队
     {
          Pop(s2,e);     cout<<"出队元素为："<<e<<endl;
     }
     else    //s2 为空时，检查 s1 是否为空
     {
          if(StackEmpty(s1))          // s2 和 s1 均为空时，则表示队列空，不能出队
               cout<<"队列空，不能出队!"<<endl;
          else    // /s2 为空，且 s1 非空时，表示队列非空，可以出队
          {
               while(!StackEmpty(s1))  //s1 全部元素依次退栈并压入到 s2 中
               {
                    Pop(s1,e);     Push(s2,e);
               }
               Pop(s2,e);                  // s2 栈顶退栈，即队列的队头出队
               cout<<"出队元素为："<<e<<endl;
          }
     }
}
```