**算法题库**

**1. 编写算法，以一维数组作为存储结构，实现线性表的就地逆置。**

Void reverse (elemtype A[ ],int n)

{ int I;

Elemtype temp;

For (i=0;i<n/2;i++)

{ temp=A[i];

A[i]=A[n-i+1];

A[n-i+1]=temp; } }

**2.假设单链表的存储结构为：**

**//单链表的存储表示**

**typedef struct Node{**

**ElemType data; //数据域**

**struct Node\* next; //指针域**

**}Node, \*LinkList;**

**设计一算法，统计带表头结点的单链表中具有给定值x的元素个数。**

Count ( LinkList L, ElemType x )

{

//在单链表中进行一趟检测，找出具有给定值的结点地址, 如果表空, 返回指针NULL

int n = 0;

Node \* p = L->next; //从第一个结点开始检测

while ( p != NULL )

{ //循环, 下一个结点存在

i f ( p->data == x ) n++; //找到一个, 计数器加1

p = p->next; } //检测下一个结点

return n; }

**3.带头结点的单链表的存储结构为：**

**typedef struct LNode**

**{ ElemType data; //数据域**

**struct LNode \*next; //指针域**

**}LNode, \*LinkList;**

**线性表操作说明：**

**LENGTH(L)**

**初始条件：线性表L已存在。**

**操作结果：返回L中数据元素个数。**

**写一算法，在带头结点的单链表结构上实现线性表操作LENGTH(L)。**

int LENGTH(LinkList L)

{ k=0;

p=L;

while(p->next)

{ p=p->next,

k++;

}

return k;

}

**4.数制转换：10进制数转换为8进制数。**

void conversion () {

InitStack(S); // 构造空栈

scanf ("%d",&N); //输入一个十进制数

while (N) {// x不等于零循环

Push(S, N % 8); // "余数"入栈

N = N/8; 　// 非零"商"继续运算  
 }

while (!StackEmpty(S)) {  
 //和"求余"所得相逆的顺序输出八进制的各位数

Pop(S,&e);

printf ( "%d", e );

}

} // conversion

1. **假设称正读和反读都相同的字符序列为“回文”，例如，‘abba’和‘abcba’是回文，‘abcde’ 和 ‘ababab’ 则不是回文。试写一个算法判别读入的一个以‘ ＠ ’ 为结束符的字符序列是否是回文。**

status isHuiWen() //判断键盘输入的是否是回文 栈  
{ InitStack(s);  
InitQueue(q);  
while((c=getchar())!=‘@‘)  
{ push(s,c)  
   EnQueue(q,c); }  
while(!StackEmpty(s))  
{  
pop(s,s\_pop);  
deQueue(q,q\_deQueue);  
if(s\_pop!=q\_deQueue)  
   return False;  
}  
return TRUE;

}

**6. 已知二叉树的存储结构为二叉链表（如图所示），编写函数PrintNode(BinTree root) ，输出以root为根的二叉树的结点。**



Void PrintNode (BiTree root) {

if (root != NULL)

{

Printf(root->data);

printNode(root->LChild);

printNode(root->RChild);

}

}

**7.假设二叉树采用二叉链表存储结构：**

**//二叉树的二叉链表存储表示**

**typedef struct BiTNode{**

**TElemType data;**

**struct BiTNode \*lchild,\*rchild; //左右孩子指针**

**}BiTNode,\*BiTree;**

**编写递归算法，对二叉树进行先序遍历。**

void PreOrderTraverse(BiTree T)

{ if(T!=NULL)

{ cout<<(T->data);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

}

}

**8.假设顺序表的存储结构为：**

**#define LIST\_SIZE 20**

**typedef int KeyType; //定义关键字类型为整数类型**

**typedef struct**

**{ KeyType key; //关键字域**

**OtherType other\_data; //其它域**

**}RecordType; //数据元素类型**

**typedef struct**

**{ RecordType r[LIST\_SIZE+1]; //r[0]为工作单元**

**int length; //表长度**

**}RecordList; //顺序表类型**

**写一个算法，在顺序表中顺序查找其关键字等于key的数据元素。若找到，返回该元素在表中的位置，否则为0。**

int SeqSearch（RecordList l, KeyType k）

/\*在顺序表l中顺序查找其关键字等于k的元素，若找到，则函数值为该元素在表中的位置，否则为0\*/

{

l.r[0].key=k; i=l.length;

while (l.r[i].key!=k) i--;

return（i）;

}

**9．假设顺序表的存储结构如下：**

**#define LIST\_SIZE 20**

**typedef int KeyType; //定义关键字类型为整数类型**

**typedef struct**

**{ KeyType key; //关键字域**

**OtherType other\_data; //其它域**

**}RecordType; //数据元素类型**

**typedef struct**

**{ RecordType r[LIST\_SIZE+1]; //r[0]为工作单元**

**int length; //表长度**

**}RecordList; //顺序表类型**

**写出在顺序有序表中进行折半查找的算法。**

int BinSrch （RecordList l, KeyType k）

{

low=1 ; high=l.length; /\*置区间初值\*/

while ( low<=high)

{

mid=(low+high) / 2;

if (k==l.r[mid]. key) return（mid）; //找到待查元素

else if (k<l.r[mid]. key) high=mid-1;

/\*未找到，则继续在前半区间进行查找\*/

else low=mid+1; /\*继续在后半区间进行查找\*/

}

return (0);

}

**10. 编写直接插入排序算法**

#define MAXSIZE 20 //一个用作示例的小顺序表的最大长度

typedef int KeyType; //定义关键字类型为整数类型

typedef struct{

KeyType key; //关键字项

InfoType otherinfo; //记录类型

}RedType;

typedef struct{

RedType r[MAXSIZE+1]; //r[0]闲置或用作哨兵单元

Int length; //顺序表长度

}SqList; //顺序表类型

void InsSort(RecordType r[],int length)

/\*对记录数组r做直接插入排序，length为数组的长度\*/

{

for ( i=2 ; i< length ; i++ )

{

r[0]=r[i]; j=i-1; /\*将待插入记录存放到r[0]中\*/

while (r[0].key< r[j].key ) /\* 寻找插入位置 \*/

{r[j+1]= r[j]; j=j-1;}

r[j+1]=r[0]; /\*将待插入记录插入到已排序的序列中\*/

}

} /\* InsSort \*/