1.线性表的两种存储结构各有哪些优缺点？

答：线性表的顺序存储结构优点为可以直接存取数据元素，方便灵活、效率高，而缺点为插入、删除操作时将会引起元素的大量移动，降低效率。

线性表的链式存储结构中内存采用动态分配，利用率高，缺点是需要增设指示结点之间关系的指针域，存取数据元素较不方便。

2.对于线性表的两种存储结构，如果有n个线性表同时并存，并且在处理过程中各表的长度会动态发生变化，线性表的总数也会自动改变，在此情况下，应该选用哪种存储

结构，为什么？

答：应该选用链式存储结构，因为链式存储结构是用一组任意的存储单元一次存储线性表中的各元素，这里存储单元可以是连续的，也可以是不连续的；这种存储结构对于元素的删除或插入运算是不需要移动元素的，只需修改指针即可，所以很容易实现表的容量的扩充。

1. 对于线性表的两种存储结构，如果线性表的总数基本稳定，并且很少进行插入和删除操作，但是要求以最快的速度存取线性表中的元素，则应该选用哪种存储结构？试说明理由。

答：应该选用顺序存储结构，因为每个数据元素的存储位置和线性表的起始位置相差一个和数据元素在线性表中的序号成正比的常数。因此，只要确定了其起始位置，线性表中的任一个数据元素都可以随机存取。

4. 已知L是无头结点的单链表，且p结点既不是第一个结点，也不是最后一个结点，试从下列提供的语句中选出合适的语句序列。

I. 在p结点之后插入s结点： \_\_\_\_\_\_\_\_\_4\_\_\_1\_\_\_\_\_\_\_\_\_

II. 在p结点之前插入s结点： \_\_\_\_\_8\_\_12\_\_\_9\_\_\_4\_\_1\_\_\_\_

III. 在单链表L首插入s结点： \_\_\_\_\_\_\_5\_\_\_13\_\_\_\_\_\_\_\_\_\_

IV. 在单链表L后插入s结点： \_\_\_\_\_\_12\_\_\_10\_\_6\_\_\_1\_\_\_\_\_\_

提供的语句：

(1)p->next=s；

(2)p->next=p->next->next；

(3)p->next=s->next；

(4)s->next=p->next；

(5)s->next=L；

(6)s->next=p；

(7)s->next=NULL；

(8)q=p；

(9)while（p->next！=q） p=p->next；

(10)while（p->next！=NULL） p=p->next；

(11)p=q；

(12)p=L；

(13)L=s；

(14)L=p；

5. 已知p结点是某双向链表的中间结点，试从下列提供的语句中选出合适的语句序列。

I. 在p结点之后插入s结点： 。 7、12、3、6

II. 在p结点之前插入s结点： 。 13、8、5、4

III. 删除p结点的直接后继结点： 。 15、1、11、17

IV. 删除p结点的直接前驱结点： 。 16、2、10、17

提供的语句：

(1) p->next=p->next->next；

(2) p->prior=p->prior->prior；

(3) p->next=s；

(4) p->prior=s；

(5) s->next=p；

(6) s->prior=p；

(7) s->next=p->next；

(8) s->prior=p->prior；

(9) p->prior->next=p->next；

(10) p->prior->next=p；

(11) p->next->prior=p；

(12) p->next->prior=s；

(13) p->prior->next=s；

(14) p->next->prior=p->prior；

(15) q=p->next；

(16) q=p->prior；

(17) free（p）；

(18) free（q）；

6. 将下列程序补充完整

试编写一个计算头结点指针为L的单链表长度的算法。

提示：

（1）算法思想

在顺序存储结构中，线性表的长度是它的一个属性，因此很容易求得。但是当以单链表存储结构表示线性表时，线性表的长度即单链表中的结点个数，所以只能通过“遍历”链表来得到单链表的长度。

1. 设一个指针p顺链向后扫描，同时设一个整型变量k随之进行“计数”。p的初值为指 向 第一个结点，k的初值为0。
2. 如果p不为空，则k增1，令p指向其后继，如此循环直至p为空为止，此时所得到的k值即为表长。

线性表的单链表存储结构表示如下：

typedef struct LNode

{

ElemType data；

structLNode \*next；

} LNode， \*LinkList

int Length\_L (LinkListL)

{

//L为单链表的头指针，此函数返回 L所指单链表的长度

\_\_\_\_p=L->next\_\_\_； //初始时，p指向单链表的第一个结点

k=0； //初始时，计数器k为0

while（p!=NULL）

{

\_\_\_k++\_\_\_\_\_\_； //计数器k增1

\_\_p=p->next\_\_； //指针p顺链向后移动

}

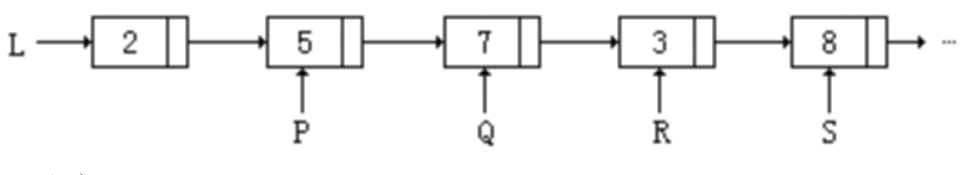
return(k)；

}

7. 对以下单链表分别执行下列各程序段，并画出结果示意图。

****

1. Q=P->next;



1. L=P->next;

L

7

Q

3

R

8

S

1. R->data=p->data;

L

2

5

P

7

Q

5

R

8

S

1. R->data=p->next->data;

L

2

5

P

7

Q

7

R

8

S

1. P->next->next->next->data=P->data;

L

2

5

P

7

Q

3

R

5

S

1. T=P;

While(T!=NULL)

{T->data=T->data\*2;T=T->next;}

L

2

10

P

14

Q

6

R

16

S

1. T=P;

While(T->next!=NULL)

{T->data=T->data\*2;T=T->next;}

L

2

10

P

14

Q

6

R

8

S

8. 编写一个在单链表中数据域值为a的结点之后，插入一个新结点的算法。若原链表中无数据域值为a的结点，则把新结点插入到表尾。设新结点数据域值为x。

typedef struct LNode

{

ElemType data；  
structLNode \*next；

}LNode，\*LinkList；

（2）算法编写  
Status Insert\_L (Linklist &L，int a，int x)

{

//在单链表中查找数据域值为 a的结点，若找到，则在其后插入一个新结点；

//若没有找到，则把新结点插入到单链表表尾。设新结点数据域值为 x

\_\_\_p=L->next\_\_\_； //初始时，p指向单链表的第一个结点  
 while(\_\_p!=NULL\_ && \_p->data!=a\_) //判断p的状态

\_\_\_p=p->next\_； //顺链向后查找  
s=\_\_\_(Linklist\*)malloc(sizeof(Linklist))\_\_\_; //给要插入的结点分配地址空间  
\_s->data=x\_\_； //给新结点的数据域赋值为x  
\_s->next=NULL\_； //将新结点插入到p结点之后  
\_p->next=s\_； //将p结点的next域指向新结点s

}