1. 如果进栈的数据元素序列为1、2、3、4、5、6，能否得到4、3、5、6、1、2和1、3、5、4、2、6的出栈序列？并说明为什么不能得到或如何得到。

答：第一个不能，因为1要先进且在2之后出栈；第二个可以，进1，出1，进2，

进3，出3，进4，进5，出5，出4，出2，进6，出6。

1. 线性表可用顺序表和单链表作为存储结构。试问
2. 两种存储表示各有哪些主要优缺点？

答：顺序表：

优点：以数组形式存储，有固定的存储空间，存取速度高效，可通过下标索引直接存储； 缺点：插入和删除速度比较慢，不可以增加长度。

单链表：

优点：链表存储具有动态存储空间，不会发生存储溢出问题。插入和删除速度快，保留原有的物理顺序。

1. 如果有ｎ个表同时并存，且处理过程中各表的长度会动态发生变化，表的总数也可能自动改变，在此情况下应选用哪种存储表示？为什么？

答：应选用单链表。顺序表的长度运行中不能改变。

（３）若表的总数基本稳定，且很少进行插入和删除，但要求以最快速度存取表中元素，这时应采用哪种存储表示？为什么？

答：应选用顺序表。顺序表存取速度更高效。

1. 试设计一个算法，它在带表头结点的单链表中寻找第ｉ个结点，若找到则返回第ｉ个结点的地址，否则返回空地址（ＮＵＬＬ）。

#include"stdio.h"

typedef int elemtype;

typedef struct node

{

elemtype data;

　 struct node \*next;

}linklist;

linklist \*creatlinklist( )　 //建立带表头结点的单链表

{

int x;

　linklist \*head,\*r,\*p;

　head= \_\_\_\_\_\_\_\_\_\_\_(linklist \*)malloc(sizeof(linklist))\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　head->next= \_\_\_\_\_\_\_\_\_NULL\_\_\_\_\_\_\_\_\_\_;

　r=\_\_\_\_\_\_\_head\_\_\_\_\_\_\_\_\_;

　scanf("%d",&x);

　while(x!= -1)

　{

p=\_\_\_\_\_\_\_\_\_\_(linklist \*)malloc(sizeof(linklist))\_\_\_\_\_\_\_\_\_;

　　 p->data= \_\_\_\_\_\_\_\_\_x\_\_\_\_\_\_\_\_\_\_;

p->next= \_\_\_\_\_\_\_\_NULL\_\_\_\_\_\_\_\_\_\_\_;

　　 r->next= \_\_\_\_\_\_\_\_\_\_p\_\_\_\_\_\_\_\_\_;

　　 r= \_\_\_\_\_\_\_\_\_\_p\_\_\_\_\_\_\_\_\_;

　　 scanf("%d",&x);}

　　 return head;

　} //creatlinklist 结束

//查找第i个结点，返回该结点的存储位置或者NULL

Linklist \*Get(linklist \*h, int i)

{

int j;

　 linklist \*p;

　 p=h;j= 0; //从头结点起扫描，计数器j的初值为0

　 while((p->next!= \_\_\_\_\_\_\_\_\_NULL\_\_\_\_\_\_\_\_\_\_)&&(i>j))

　　{

p= \_\_\_\_\_\_\_\_\_h->next\_\_\_\_\_\_\_\_\_\_; //指针右移，扫描下一个结点

　　 \_\_\_\_\_\_\_\_\_\_j++\_\_\_\_\_\_\_\_; //统计扫描过的结点总数

　　 }

　if(j==i)

　　　return p; //返回找到的第i个结点的地址

　else

　　　return NULL; //未找到，返回 NULL

}

main( )

{

linklist \*head,\*p;

　int i;

　printf("\nPlease input the data:");

　head= \_\_\_\_\_\_\_creatlinklist()\_\_\_\_\_\_\_\_\_\_\_\_;

　printf("\nPlease input the i:");

　scanf("%d",&i);

　p=\_\_\_\_\_\_\_\_Get(\*h)\_\_\_\_\_\_\_\_\_\_\_;

　if(p!=NULL)

　　 printf("The %dth data is %d.",i, p->data);

　else

　　 printf("No find");

}

1. 设计一个算法，它通过一趟遍历在单链表中确定值最大的结点.

解题思路: 以ｐ遍历单链表，在遍历时用ｑ指向ｄａｔａ域值最大的结点，最后返回ｑ。

#include "stdio.h"

typedef int elemtype;

typedef struct node

{

elemtype data;

　 struct node \*next;

}linklist;

linklist \*creatlinklist() //建立带表头的链表

{

int x;

　linklist \*head,\*r,\*p;

　head= \_\_\_\_\_\_\_\_\_\_\_\_\_(linklist \*)malloc(sizeof(linklist))\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　head->next= \_\_\_\_\_\_\_\_\_\_\_\_\_NULL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　r= \_\_\_\_\_\_\_\_\_\_\_head\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　scanf("%d",&x);

　while(x! =-1)　 /-1 为结束标志/

　{

p= \_\_\_\_\_\_\_\_\_\_\_\_\_\_(linklist \*)malloc(sizeof(linklist))\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

p->data= \_\_\_\_\_\_\_\_\_\_\_\_x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

p->next= \_\_\_\_\_\_\_\_\_\_\_\_NULL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　　r->next= \_\_\_\_\_\_\_\_\_\_\_\_\_p\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　　r= \_\_\_\_\_\_\_\_\_\_\_\_\_p\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　　scanf("% d",&x);

　}

　return head;

}

linklist \*maxnode(linklist \*h) //遍历在单链表中确定值最大的结点

{

linklist \*q,\*p;

　 p= \_\_\_\_\_\_\_\_\_\_\_\_h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　 q= \_\_\_\_\_\_\_\_\_\_\_0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　 while(p!= \_\_\_\_\_\_\_\_\_\_\_NULL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

　 {

if(\_\_\_\_\_\_\_\_\_\_q->data<p->data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

　　　　 q= p;

　　　\_\_\_\_\_\_\_\_\_\_\_\_\_p=p->next\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　　}

　 return q;

}

main()

{

linklist \*head,\*h;

　printf("\nPlease input the data:");

　head= \_\_\_\_\_\_\_\_\_\_\_creatlinklist ()\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

　h= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_maxnode (\*h)\_\_\_\_\_\_\_\_\_\_\_\_\_;

　printf("Max is %d",h-> data);

}