* 1. 二维数组M的每个元素是6个字符组成的串，行下标i的范围从0到8，列下标j的范围从1到10，则存放M至少需要\_\_540\_\_\_\_个字节；M的第8列和第5行共占\_\_108\_\_\_\_个字节。

解：每个元素占6个字符，一个字符一个字节，所以每个元素占6个字节；

I：共有9个，j：共有10个

第一个空：9\*10\*6=540

第八列共有9个元素，第五行共有10个元素，但是有一个元素被计算了两遍，所以，第八列和第五行共有18个元素，18\*6=108

* 1. 设有一批数据元素，为了最快地存取某元素，宜用\_\_顺序\_\_\_\_结构存储，为了方便的插入一个元素，宜用\_\_链式\_\_\_\_结构存储．

解：注意顺序结构和链式结构的各自的利弊

顺序结构：随机存储

链式结构：方便插入和删除

* 1. 设有一个空栈，现输入序列为1、2、3、4、5。经过push，push，pop，push，pop，push，pop，push后，输出序列是\_\_\_2 3 4\_\_\_\_\_\_\_\_\_。
  2. 设输入序列为1、2、3，则经过栈的作用后可以得到\_\_5\_\_\_\_种不同的输出序列。
  3. 设F和R分别表示顺序循环队列的头指针和尾指针，则判断该循环队列为空的条件为\_\_F==R\_\_\_\_\_\_\_\_\_\_。
  4. 设有一个n阶的下三角矩阵A，如果按照行的顺序将下三角矩阵中的元素（包括对角线上元素）存放在n(n+1)个连续的存储单元中，则A[i][j]与A[0][0]之间有\_\_ i(i+1)/2+j-1+1 （i-1）(i+2)/2+j-1\_\_\_\_个数据元素。

解：行下标从0开始

A[0]行：有一个A[0][0]

A[1]行：有两个

。。。

A[i-1]行:有i个

等差数列求和公式：i(i+1)/2

A[i][j],j下标从0开始，所以这个元素在A[i]行中前面有j-0个元素

所以：i(i+1)/2+j+1

第二种算法：不算A[0][0],2+3+4+5+。。。+I （i-1）(i+2)/2+j-1

* 1. 设有一个顺序共享栈S[0：n-1]，其中第一个栈项指针top1的初值为-1，第二个栈顶指针top2的初值为n，则判断共享栈满的条件是\_\_top1+1=top2\_\_\_\_\_\_\_\_\_\_。
  2. 不论是顺序存储结构的栈还是链式存储结构的栈，其入栈和出栈操作的时间复杂度均为\_\_O(1)\_\_\_\_。
  3. 链表不具备的特点是 （ ）。

1. 可随机访问任一结点
2. 插入删除不需要移动元素
3. 不必事先估计存储空间
4. 所需空间与其长度成正比
   1. 下列哪种数据结构用于执行递归调用（ ）。

A. 队列 B. 栈 C. 串 D. 链表

* 1. 二维数组A[5][6]的每个元素占5个存储单元，从首地址LOC(0,0)开始将其按行优先顺序存储在起始地址为1000的连续的内存单元中，则元素A[4][4]地址为（ ）。

A. 1140 B. 1145 C. 1120 D. 1125

解：（4\*6+4）\*5+1000

* 1. 设循环队列中数组的下标范围是1~n，其头指针为f，尾指针为r，则其当前元素个数为（ ）。

1. r-f
2. r-f+1
3. (r-f+n) mod n
4. (r-f) mod n +1
   1. 程序段s=i=0；do { i=i+1; s=s+I;}while(i<=n);的时间复杂度为（ ）。（ ）。

A. O(n) B. O(nlog2n) C. O(1) D. O(n^2)

* 1. 设指针变量p指向单链表中结点A，若删除单链表中结点A，则需要修改指针的操作序列为（ ）。

1. q=p->next；p->data=q->data；p->next=q->next；free(q)；
2. q=p->next；q->data=p->data；p->next=q->next；free(q)；
3. q=p->next；p->next=q->next；free(q)；
4. q=p->next；p->data=q->data；free(q)；
   1. 设输入序列为1、2、3、4、5、6，则通过栈的作用后可以得到的输出序列为（ ）。
5. 5，3，4，6，1，2
6. 3，2，5，6，4，1
7. 3，1，2，5，4，6
8. 1，5，4，6，2，3
   1. 设某链表中最常用的操作是在链表的尾部插入或在尾部删除元素，则下列、存储方式中最节省运算时间的是（ ）。
9. 单向链表
10. 单向循环链表
11. 双向链表
12. 双向循环链表

解：其他形式的链表还得循环的找最后一个元素，而双向循环链表不用循环的找最后一个元素

* 1. 建立一个长度为n的有序单链表的时间复杂度为（ ）

A O(n) B. O(1) C. O(n^2) D. O(log2n)

解：最坏情况下排序， 比较是基本操作，比较的次数为等差数列求和公式

* 1. 设指针变量top指向当前链式栈的栈顶，则删除栈顶元素的操作序列为（ ）。



A. top=top+1; B. top=top-1;

B. top->next=top; D. top=top->next;

|  |
| --- |
| #define TRUE 1  #define FALSE 0  #define OK 1  #define ERROR 0  #define INFEASIBLE -1  #define OVERFLOW -2  typedef int Status; |

* 1. （10分）算法改错,下面是实现循环队列的算法，存在5处严重错误，请找出错误并改正（找出1处给1分，改正1处给1分）。

|  |  |
| --- | --- |
| 1. #define MAXQSIZE 100 // 最大队列长度 2. typedef int QElemType; 3. typedef struct { 4. QElemType \* base; // 初始化的动态分配存储空间 5. int front; // 头指针，若队列不空，指向队列头元素 6. int rear; // 尾指针，若队列不空，指向队列尾元素的下一个位置 7. } SqQueue; | |
| 1. // 初始化队列 2. Status InitQueue(SqQueue &Q) { 3. Q.base = (QElemType \*)malloc(sizeof(QElemType)); 4. Q.base = (QElemType \*)malloc(MAXQSIZE\*sizeof(QElemType)); 5. if(!Q.base) exit(OVERFLOW); 6. Q.front = Q.rear = 0; 7. return OK; 8. } 9. // 返回队列长度 10. int QueueLength(SqQueue Q) { 11. return Q.rear - Q.front; 12. (Q.rear - Q.front+MAXQSIZE)%MAXQSIZE; 13. } 14. // 队尾插入元素 15. Status EnQueue(SqQueue &Q, QElemType e) { 16. if(Q.rear+1 == Q.front) return ERROR; 17. (Q.rear+1)%MAXSIZE == Q.front 18. Q.base[Q.rear] = e; 19. Q.rear = Q.rear+1; 20. Q.rear = (Q.rear+1)%MAXQSIZE; 21. return OK; 22. } 23. // 删除并返回队头元素 24. Status DeQueue(SqQueue &Q, QElemType &e) { 25. if(Q.front == Q.rear+1) return ERROR; 26. (Q.rear+1)%MAXSIZE == Q.front 27. e = Q.base[Q.front]; 28. Q.front = (Q.front+1) % MAXQSIZE; 29. return OK; 30. } | |

* 1. 已知串的堆分配存储结构HString，使用C语言编写函数StrInsert()实现在第pos个字符之前的串插入操作，pos合法取值为[1, S.length+1]，相关声明如下。

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #defineoverflow-1  #defineerror0  #defineok1  typedefintstatus;  typedefstruct{  char\*ch;  intlength;  }HString;  statusStrInsert(HString\*S,intpos,HStringT)  {  inti,j;  if(pos<1||pos>S->length+1)returnerror;  if(T.length)  {  if(!(S->ch=(char\*)realloc(S->ch,(S->length+T.length)\*sizeof(char))))  exit(overflow);  for(i=S->length-1;i>=pos-1;--i)  {  S->ch[i+T.length]=S->ch[i];  }  //S.ch[pos-1…pos+T.lengh-2]=T.ch[0…T.length-1];  for(j=0;j<=T.length-1;++j)  {  S->ch[pos-1+j]=T.ch[j];  }  S->length+=T.length;  }  returnok;  }  voidmain()  {  } |
| * 1. 现有5\*5矩阵，要求根据提示信息使用C语言编写函数isSymmetricMatrix()来判断矩阵是否为对称矩阵，如果是对称矩阵返回TRUE，否则返回FALSE。  |  | | --- | | #include<stdio.h>  #include<stdlib.h>  #defineTRUE1  #defineFALSE0  intIsSymmetricMatrix(intM[5][5])  {  inti,j;  for(i=0;i<=4;++i)  {  for(j=0;j<=4;j++)  {  if(M[i][j]==M[j][i])  {  printf("此矩阵不是方阵");  returnFALSE;  break;  }  }  }  printf("此矩阵是方阵");  returnTRUE;  } | |  |  * 1. 共享栈是两个栈在一个顺序的存储空间中，两个栈的栈底分别是存储空间的首尾地址。根据下面给出的声明语句，使用C语言完成共享栈initStack()、pushStack()、popStack()三个函数的定义。  |  | | --- | | #define MaxSize 60 // 共享栈存储的最大元素个数  typedef int ElemType;  typedef int Status;  typedef struct {  ElemType data[MaxSize];  int top1;  int top2;  } Stack;  // 共享栈的初始化  Status initStack(Stack &S);  // 在共享栈中的n栈里压入元素，n取值为1、2  Status pushStack(Stack &S, ElemType e, int n);  // 在共享栈中的n栈里弹出元素，并用e返回弹出元素  Status popStack(Stack &S, ElemType &e, int n); | |

* 1. 设单链表中有且仅有三类字符的数据元素(字母、数字和其它字符)，要求使用C语言根据给出的提示信息编写split()函数，实现将指定原单链表根据元素种类拆分成三个新单链表的算法，拆分后原单链表变为空链表，每个新单链表中只存储一类数据元素，程序中不允许新建任何节点，三个新单链表中的结点均使用原单链表中的节点。

|  |
| --- |
| typedef struct Node {  char data;  struct Node \* next;  } \* LinkList;  // head 原单链表  void split(LinkList &head, LinkList & ha, LinkList & hb, LinkList & hc); |

|  |
| --- |
|  |

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

char data;

struct Node \* next;

} \* LinkList;

//根据asc码值表，数字0-9对应的asc码为48~57，大写字母A~Z对应的asc码为65~90

//小写字母a~z对应的asc码为97~122

// head 原单链表

void split(LinkList \*head, LinkList \*ha, LinkList \*hb, LinkList \* hc)

{

LinkList p,q,s;

p=(\*head)->next;

q=(\*head);

while(p!=NULL)

{

(\*head)->next=p->next;

if(p->data>='0' &&p->data<='9')

{

//插入链表ha，头插法或尾插法

p->next=(\*ha)->next;

(\*ha)->next=p;

p=(\*head)->next;

}

else if((p->data>='A' && p->data<='Z') || (p->data>='a' &&p->data<='z'))

{

//插入链表hb，头插法或尾插法

p->next=(\*ha)->next;

(\*ha)->next=p;

p=(\*head)->next;

}

else

{

//插入链表hc，头插法或者尾插法

p->next=(\*ha)->next;

(\*ha)->next=p;

p=(\*head)->next;

}

}