一、填空题（每空 1 分，共 15 分）

1.向量、栈和队列都是__线性__结构，可以在向量的__任何__位置插入和删除元素；对于栈只能在__栈顶__插入和删除元素；对于队列只能在__队尾__插入和__队首__删除元素。

2. 栈是一种特殊的线性表，允许插入和删除运算的一端称为__栈顶__。不允许插入和删除运算的一端称为__栈底__。

3. __队列__是被限定为只能在表的一端进行插入运算，在表的另一端进行删除运算的线性表。

4. 在一个循环队列中，队首指针指向队首元素的__前一个__位置。

5. 在具有 n 个单元的循环队列中，队满时共有__n-1__个元素。

6. 向栈中压入元素的操作是**先__移动栈顶指针__，后__存入元素__**。

7. 从循环队列中删除一个元素时，其操作是 **先__移动队首指针__，后__取出元素__**。

8.带表头结点的空循环双向链表的长度等于__0__。

解：

二、判断正误（判断下列概念的正确性，并作出简要的说明。）（每小题 1 分，共 10 分）

（ × ）1. 线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型。

错，线性表是逻辑结构概念，可以顺序存储或链式存储，与元素数据类型无关。

（ × ）2. 在表结构中最常用的是线性表，栈和队列不太常用。

错，不一定吧？调用子程序或函数常用，CPU 中也用队列。

（ √ ）3. 栈是一种对所有插入、删除操作限于在表的一端进行的线性表，是一种后进先出型结构。

（ √ ）4. 对于不同的使用者，一个表结构既可以是栈，也可以是队列，也可以是线性表。

正确，都是线性逻辑结构，栈和队列其实是特殊的线性表，对运算的定义略有不同而已。

（ × ）5. 栈和链表是两种不同的数据结构。

错，栈是逻辑结构的概念，是特殊殊线性表，而链表是存储结构概念，二者不是同类项。

（ × ）6. 栈和队列是一种非线性数据结构。

错，他们都是线性逻辑结构，栈和队列其实是特殊的线性表，对运算的定义略有不同而已。

（ √ ）7. 栈和队列的存储方式既可是顺序方式，也可是链接方式。

（ √ ）8. 两个栈共享一片连续内存空间时，为提高内存利用率，减少溢出机会，应把两个栈的栈底分别设在这片内存空间的两端。

（ × ）9. 队是一种插入与删除操作分别在表的两端进行的线性表，是一种先进后出型结构。

错，后半句不对。

（ × ）10. 一个栈的输入序列是 12345，则栈的输出序列不可能是 12345。

错，有可能。

三、单项选择题（每小题 1 分，共 20 分）

（ B ）1.栈中元素的进出原则是

A．先进先出　　B．后进先出　　C．栈空则进　　D．栈满则出

（ C ）2.若已知一个栈的入栈序列是 1，2，3，…，n，其输出序列为 p1，p2，p3，…，pn，若 p1=n，则 pi 为

A．i　　B．n=i　　C．n-i+1　　D．不确定

解释：当 p1=n，即 n 是最先出栈的，根据栈的原理，n 必定是最后入栈的（事实上题目已经表明了），那么输入顺序必定是 1，2，3，…，n，则出栈的序列是 n，…，3，2，1。

（若不要求顺序出栈，则输出序列不确定）

（ B ）3.判定一个栈 ST（最多元素为 m0）为空的条件是

A．ST->top<>0　　B．ST->top=0　　C．ST->top<>m0　　D．ST->top=m0

（ A ）4.判定一个队列 QU（最多元素为 m0）为满队列的条件是

A．QU->rear － QU->front = = m0　　B．QU->rear － QU->front －1= = m0

C．QU->front = = QU->rear　　D．QU->front = = QU->rear+1

解：队满条件是元素个数为 m0。由于约定满队时队首指针与队尾指针相差 1，所以不必再减 1 了，应当选 A。当然，更正确的答案应该取模，即：QU->front = = (QU->rear+1)% m0

（ D ）5．数组 Q［n］用来表示一个循环队列，f 为当前队列头元素的前一位置，r 为队尾元素的位置，假定队列中元素的个数小于 n，计算队列中元素的公式为

（A）r－f;　　（B）（n＋f－r）% n;　（C）n＋r－f;　　（D）（n＋r－f）% n

6. 从供选择的答案中，选出应填入下面叙述 ？ 内的最确切的解答，把相应编号写在答卷的对应栏内。

设有 4 个数据元素 a1、a2、a3 和 a4，对他们分别进行栈操作或队操作。在进栈或进队操作时，按 a1、a2、a3、a4 次序每次进入一个元素。假设栈或队的初始状态都是空。

现要进行的栈操作是进栈两次，出栈一次，再进栈两次，出栈一次；这时，第一次出栈得到的元素是 A ，第二次出栈得到的元素是 B 是；类似地，考虑对这四个数据元素进行的队操作是进队两次，出队一次，再进队两次，出队一次；这时，第一次出队得到的元素是 C ，第二次出队得到的元素是 D 。经操作后，最后在栈中或队中的元素还有 E 个。

供选择的答案：

A～D：①a1　②a2　③ a3　④a4

E：①1　②2　③ 3　④ 0

答：ABCDE＝2, 4, 1, 2, 2

7. 从供选择的答案中，选出应填入下面叙述 ？ 内的最确切的解答，把相应编号写在答卷的对应栏内。

栈是一种线性表，它的特点是 A 。设用一维数组 A[1,…,n]来表示一个栈，A[n]为栈底，用整型变量 T 指示当前栈顶位置，A[T]为栈顶元素。往栈中推入（PUSH）一个新元素时，变量 T 的值 B ；从栈中弹出（POP）一个元素时，变量 T 的值 C 。设栈空时，有输入序列 a，b，c，经过 PUSH，POP，PUSH，PUSH，POP 操作后，从栈中弹出的元素的序列是 D ，变量 T 的值是 E 。

供选择的答案：

A：① 先进先出　②后进先出　③进优于出　④出优于进　⑤ 随机进出

B，C：① 加 1　②减 1　③不变　④清 0　⑤ 加 2　⑥减 2

D：① a,b　②b,c　③c,a　④b,a　⑤ c,b　⑥ a,c

E：① n+1　②n+2　③ n　④ n-1　⑤ n-2

答案：ABCDE=2, 2, 1, 6, 4

注意，向地址的高端生长，称为向上生成堆栈；向地址低端生长叫向下生成堆栈，本题中底部为 n，向地址的低端递减生成，称为向下生成堆栈。

8. 从供选择的答案中，选出应填入下面叙述 ？ 内的最确切的解答，把相应编号写在答卷的对应栏内。

在做进栈运算时，应先判别栈是否 A ；在做退栈运算时，应先判别栈是否 B 。当栈中元素为 n 个，做进栈运算时发生上溢，则说明该栈的最大容量为 C 。

为了增加内存空间的利用率和减少溢出的可能性，由两个栈共享一片连续的内存空间时，应将两栈的__D__分别设在这片内存空间的两端，这样，只有当__E__时，才产生上溢。

供选择的答案：

A，B：①空　　②　满　　③ 上溢　　④ 下溢

C：　①n-1　　② n　　③ n+1　　④ n/2

D：　① 长度　　②深度　　③ 栈顶　　④ 栈底

E：①两个栈的栈顶同时到达栈空间的中心点　　②其中一个栈的栈顶到达栈空间的中心点

③两个栈的栈顶在达栈空间的某一位置相遇　　④两个栈均不空，且一个栈的栈顶到达另一个栈的栈底

答案：ABCDE＝2，　1，　2，　4，　3

四、简答题（每小题 4 分，共 20 分）

1.说明线性表、栈与队的异同点。

答：**相同点：**都是线性结构，都是逻辑结构的概念。都可以用顺序存储或链表存储；栈和队列是两种特殊的线性表，即受限的线性表，只是对插入、删除运算加以限制。

不同点：①运算规则不同，线性表为随机存取，而栈是只允许在一端进行插入、删除运算，因而是后进先出表 LIFO；队列是只允许在一端进行插入、另一端进行删除运算，因而是先进先出表 FIFO。

② 用途不同，堆栈用于子程调用和保护现场，队列用于多道作业处理、指令寄存及其他运算等等。

2.设有编号为 1，2，3，4 的四辆列车，顺序进入一个栈式结构的车站，具体写出这四辆列车开出车站的所有可能的顺序。

答：至少有 14 种。

① 全进之后再出情况，只有 1 种：4，3，2，1

② 进 3 个之后再出的情况，有 3 种，3,4,2,1　3,2,4,1　3,2,1,4

③ 进 2 个之后再出的情况，有 5 种，2,4,3,1　2,3,4,1　2,1, 3,4　2,1,4,3　2,1,3,4

④ 进 1 个之后再出的情况，有 5 种，1,4,3,2　1,3,2,4　1,3,4,2　1, 2,3,4　1,2,4,3

3.假设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。假设一字符序列已存入计算机，请分析用线性表、堆栈和队列等方式正确输出其回文的可能性？

答：线性表是随机存储，可以实现，靠循环变量（j--）从表尾开始打印输出；

堆栈是后进先出，也可以实现，靠正序入栈、逆序出栈即可；

队列是先进先出，不易实现。

哪种方式最好，要具体情况具体分析。若正文在机内已是顺序存储，则直接用线性表从后往前读取即可，或将堆栈栈顶开到数组末尾，然后直接用 POP 动作实现。（但堆栈是先减后压还是……）

若正文是单链表形式存储，则等同于队列，需开辅助空间，可以从链首开始入栈，全部压入后再依次输出。

4.顺序队的“假溢出”是怎样产生的？如何知道循环队列是空还是满？

答：一般的一维数组队列的尾指针已经到了数组的上界，不能再有入队操作，但其实数组中还有空位置，这就叫“假溢出”。采用循环队列是解决假溢出的途径。

另外，解决队满队空的办法有三：

① 设置一个布尔变量以区别队满还是队空；

② 浪费一个元素的空间，用于区别队满还是队空。

③ 使用一个计数器记录队列中元素个数（即队列长度）。

我们常采用法②，即队头指针、队尾指针中有一个指向实元素，而另一个指向空闲元素。

判断循环队列队空标志是： f=rear　　队满标志是：f=(r+1)%N

5.设循环队列的容量为 40（序号从 0 到 39），现经过一系列的入队和出队运算后，有

① front=11，rear=19;　② front=19，rear=11；问在这两种情况下，循环队列中各有元素多少个？

答：用队列长度计算公式：　(N＋r－f)% N

① L=（40＋19－11）% 40=8　　② L=（40＋11－19）% 40=32

五、阅读理解（每小题 5 分，共 20 分。至少要写出思路）

1. 按照四则运算加、减、乘、除和幂运算（↑）优先关系的惯例，并仿照教材**例 3-2** 的格式，画出对下列算术表达式求值时操作数栈和运算符栈的变化过程：

A－B×C/D+E↑F

答：

序号	OPTR 栈	OPND 栈	当前字符											备注（操作）
			A	－	B	*	C	/	D	＋	E	^	F	#	
1	#		•												push(OPND,'A')
2	#	A		•											push(OPTR,'－')
3	#－	A			•										push(OPND,'B')
4	#－	AB				•									push(OPTR,'*')
5	#－*	AB					•								push(OPND,'C')
6	#－*	ABC						•							归约，令 T_1=B*C
7	#－	AT_1						•							push(OPTR,'/')
8	#－	AT_1							•						push(OPND,'D')
9	#－/	AT_1D								•					归约，令 T_2=T_1/D
10	#－/	AT_2								•					归约，令 T_3=A－T_2
11	#	T_2								•					push(OPTR,'＋')
12	#＋	T_3									•				push(OPND,'E')
13	#＋	T_3E										•			push(OPTR,'↑')
14	#＋↑	T_3E											•		push(OPND,'F')
15	#＋↑	T_3EF												•	归约，令 T_4=E↑F
16	#＋	T_3T_4												•	归约，令 T_5=T_3+T_4
17	#	T_5												•	return(T_5)

2. 写出下列程序段的输出结果（栈的元素类型 SElem Type 为 char）。

```
void main( ){
Stack S;
Char x,y;
InitStack(S);
X='c';y='k';
Push(S,x); Push(S,'a');  Push(S,y);
Pop(S,x); Push(S,'t'); Push(S,x);
```

```
Pop(S,x); Push(S,'s');
while(!StackEmpty(S)){ Pop(S,y);printf(y); };
Printf(x);
}
```
答：输出为“stack”。

3. 写出下列程序段的输出结果（队列中的元素类型 QElem Type 为 char）。

```
void main( ){
Queue Q;   Init Queue (Q);
Char x='e'; y='c';
EnQueue (Q,'h'); EnQueue (Q,'r');   EnQueue (Q, y);
DeQueue (Q,x); EnQueue (Q,x);
DeQueue (Q,x); EnQueue (Q,'a');
while(!QueueEmpty(Q)){ DeQueue (Q,y);printf(y); };
Printf(x);
}
```
答：输出为“char”。

4. 简述以下算法的功能（栈和队列的元素类型均为 int）。

```
void algo3(Queue &Q){
Stack S; int d;
InitStack(S);
while(!QueueEmpty(Q)){
DeQueue (Q,d);   Push(S,d);
};
while(!StackEmpty(S)){
Pop(S,d); EnQueue (Q,d);
   }
}
```
答：该算法的功能是：利用堆栈做辅助，将队列中的数据元素进行逆置。

六、算法设计（每小题 5 分，共 15 分。至少要写出思路）

1. 假设一个算术表达式中包含圆括弧、方括弧和花括弧三种类型的括弧，编写一个判别表达式中括弧是否正确配对的函数 correct(exp,tag)；其中：exp 为字符串类型的变量（可理解为每个字符占用一个数组元素），表示被判别的表达式，tag 为布尔型变量。

答：用堆栈 st 进行判定，将 (、 [或 { 入栈，当遇到 } 、] 或) 时，检查当前栈顶元素是否是对应的(、 [或 {，若是则退栈，否则返回表示不配对。当整个算术表达式检查完毕时，若栈为空表示括号正确配对，否则不配对。

编程后的整个函数如下（李书 P31—32）

```
#define m0 100        /*m0 为算术表达式中最多字符个数*/
correct(exp,tag)
char exp[m0];
int tag;
{char st[m0];
```

```
   int top=0, i=1;
   tag=1;
   while (i<=m0 && tag)
   {if (exp[i]= = ‘(‘||exp[i]= =’[‘||exp[i]= =’{‘)      /*遇到‘(‘、’[‘或’{‘，则将其入栈*/
   {top++;
   st[top]=exp[i];
   }
   if (exp[i]= =’)’)    /*遇到’)’，若栈顶是‘(‘，则继续处理，否则以不配对返回*/
      if(st[top]= =‘(‘ ) top--;
      else tag=0;
   if (exp[i]= =’ )’)    /*遇到’ ]’，若栈顶是‘[‘，则继续处理，否则以不配对返回*/
      if(st[top]= =‘[ ‘]   top--;
      else tag=0;
   if (exp[i]= =’)’)    /*遇到’ }’，若栈顶是‘{‘，则继续处理，否则以不配对返回*/
      if(st[top]= =‘{‘ top--;
      else tag=0;
   i++;
}
if(top>0)tag=0;   /*若栈不空，则不配对*/
}
```

严题集对应答案：

3.19

```
Status AllBrackets_Test(char *str)//判别表达式中三种括号是否匹配
{
   InitStack(s);
   for(p=str;*p;p++)
   {
      if(*p=='('||*p=='['||*p=='{') push(s,*p);
      else if(*p==')'||*p==']'||*p=='}')
      {
         if(StackEmpty(s)) return ERROR;
         pop(s,c);
         if(*p==')'&&c!='(') return ERROR;
         if(*p==']'&&c!='[') return ERROR;
         if(*p=='}'&&c!='{') return ERROR; //必须与当前栈顶括号匹配
      }
   }//for
   if(!StackEmpty(s)) return ERROR;
   return OK;
}//AllBrackets_Test
```

答案（已上机通过）

```
#include<stdio.h>
#include<stdlib.h>
```

```
void push(char x);
void pop();
void correct(enum Boolean &tag);
//原来的定义是 void correct(struct Stack* head,enum Boolean &tag);

typedef struct Stack
{
	char data;
	struct Stack *next;
};

struct Stack *head,*p;
enum Boolean{FALSE,TRUE}tag;


void main()
{
	head=(struct Stack*)malloc(sizeof(struct Stack));
	head->data='S';
	head->next=NULL;
	// head's data has not been initialized!!
	correct(tag);
	if(tag)
		printf("Right!");
	else
		printf("Wrong!");
}

void push(char x)
{
	p=(struct Stack*)malloc(sizeof(struct Stack));
	if(!p)
		printf("There's no space.\n");
	else
	{
		p->data=x;
		p->next=head;
		head=p;
	}

}
// if you define the "Correct" function like that
//Debug will show that the Push action doesn't take effection

void pop()
{
```

```
    if(head->next==NULL)
        printf("The stack is empty.\n");
    else
    {
        p=head;
        head=head->next;
        free(p);
    }
}

//void correct(struct Stack* head,enum Boolean &tag)
void correct(enum Boolean &tag)
{
    int i;
    char y;

    printf("Please enter a bds:");
    for(i=0;y!='\n';i++)
    {
        scanf("%c",&y);
        if((y==')'&&head->data=='(')||(y==']'&&head->data=='[')||(y=='}'&&head->data=='{'))
            pop();
        else if((y=='(')||(y=='[')||(y=='{'))
            push(y);
/*调试程序显示，y 并没有被推入堆栈中。即 head->data 的值在 Push 中显示为 y 的值，但是出 Push 函数。马上变成 Null。*/
        else
            continue;
    }

    if(head->next==NULL)          //原来的程序是 if(head ==NULL)   tag=TRUE;
        tag=TRUE;
    else
        tag=FALSE;
}

/*总结：  由于 head 为全局变量，所以不应该将其再次作为函数的变量。因为 C 语言的函数变量是传值机制，所以在函数中对参数进行了拷贝复本，所以不能改变 head 的数值。*/
```

2.假设一个数组 squ[m]存放循环队列的元素。若要使这 m 个分量都得到利用，则需另一个标志 tag，以 tag 为 0 或 1 来区分尾指针和头指针值相同时队列的状态是“空”还是“满”。试编写相应的入队和出队的算法。

解：这就是解决队满队空的三种办法之① 设置一个布尔变量以区别队满还是队空（其他两种见简答题）；

思路：一开始队空，设 tag=0，若从 rear 一端加到与 front 指针相同时，表示入队已满，则令 tag=1；

若从 front 一端加到与 rear 指针相同时，则令 tag=0，表示出队已空。

3.**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”。**

答：编程如下：

```
int Palindrome_Test()//判别输入的字符串是否回文序列,是则返回 1,否则返回 0
{
 InitStack(S);InitQueue(Q);
 while((c=getchar())!='@')
 {
  Push(S,c);EnQueue(Q,c); //同时使用栈和队列两种结构
 }
 while(!StackEmpty(S))
 {
  Pop(S,a);DeQueue(Q,b));
  if(a!=b) return ERROR;
 }
 return OK;
}//Palindrome_Test
```