# Ch2 作业

| 班级 | R0112030.01 | 学号 | 2022040906023 | 姓名 | 梁书恺 | 成绩 | |
|---|---|---|---|---|---|---|---|

一、选择题

1. （ D ）是一个线性表。

   A 由 *n* 个实数组成的集合　　B 由所有实数组成的集合

   C 由所有整数组成的序列　　D 由 *n* 个字符组成的序列

2. 线性表采用顺序存储结构，设每个元素占用 4 个存储单元，第 9 个元素的地址为 144，则第 1 个元素的地址是（ D ）。

   A 108　　B 180　　C 176　　D 112

3. 顺序存储结构的优点是（ A ）。

   A 存储密度大　　B 插入运算方便

   C 删除运算方便　　D 可方便地用于各种逻辑结构的存储表示

4. 在一个长度为 n 的顺序表中，在第 i 个元素（1≤i≤n+1）之前插入一个新元素时须向后移动（ B ）个元素。

   A．n-i　　B．n-i+1　　C．n-i-1　　D．i

5. 线性表采用顺序存储结构，若元素插在第 i 个（$1 \leqslant i \leqslant n+1$）位置的概率是 $2(n-i+1)/n(n+1)$，则平均情况下插入操作移动元素的个数是（ C ）。

   A $n/2$　　B $(n+1)/2$　　C $(2n+1)/3$　　D $2n/3$

6. 在 n 个结点的顺序表中，算法的时间复杂度是 O(1)的操作是（ A ）。

   A．访问第 i 个结点（1≤i≤n）和求第 i 个结点的直接前驱（2≤i≤n）

   B．在第 i 个结点后插入一个新结点（1≤i≤n）

   C．删除第 i 个结点（1≤i≤n）

   D．将 n 个结点从小到大排序

7. 线性表若采用链式存储结构时，要求内存中可用存储单元的地址（ D ）。

   A．必须是连续的　　B．部分地址必须是连续的

   C．一定是不连续的　　D．连续或不连续都可以

8. 对于 *n* 个元素组成的线性表，建立一个单链表的时间复杂度是（ B ）。

   A $O(1)$　　B $O(n)$　　C $O(n^2)$　　D $O(n\log_2 n)$

9. 在一个单链表中，已知 q 所指结点是 p 所指结点的直接前驱，若在 q 和 p 之间插入 s 所指结点，则执行（ D ）操作。

   A p->next=s; s->next=p->next;　　B q->next=s; s->next=p;

   C p->next=s->next; s->next=p;　　D p->next=s; s->next=q;

10. 将线性表($a_1$, $a_2$, …, $a_n$)存储为带头结点的循环单链表，设 H 为链表的头指针，则链表中最后一个结点的指针域中存放的是（ B ）。

    A 变量 H 的地址　　B 变量 H 的值

    C 元素 $a_1$ 的地址　　D 空指针

11. 设指针 rear 指向带头结点的循环单链表的尾结点，若要删除链表的第一个元素结点，正确的操作是（ D ）。

    A s=rear; rear=rear->next;　　B rear=rear->next;

    C rear=rear->next->next;　　D s=rear->next->next; rear->next->next=s->next;

12. 在双链表中指针 pa 所指结点后面插入 pb 所指结点，执行的语句序列是（ D ）。

    (1) pb->next=pa->next; (2) pb->prior=pa; (3) pa->next=pb; (4) pa->next->prior=pb;

    A (1)(2)(3)(4)　　B (4)(3)(2)(1)　　C (3)(4)(1)(2)　　D (1)(4)(3)(2)

13. 在双向链表存储结构中，删除 p 所指的结点时须修改指针（ A ）。

A．p->next->prior=p->prior; p->prior->next=p->next;

B．p->next=p->next->next; p->next->prior=p;

C．p->prior->next=p; p->prior=p->prior->prior;

D．p->prior=p->next->next; p->next=p->prior->prior;

14. 以下说法错误的是（ D ）。

A．求表长、定位这两种运算在采用顺序存储结构时实现的效率不比采用链式存储结构时实现的效率低

B．顺序存储的线性表可以随机存取

C．由于顺序存储要求连续的存储区域，所以在存储管理上不够灵活

D．线性表的链式存储结构优于顺序存储结构

15. 对于顺序表的优缺点，以下说法错误的是（ C ）。

A、无需为表示结点间的逻辑关系而增加额外的存储空间

B、可以方便地随机存取表中的任一结点

C、插入和删除运算较方便

D、容易造成一部分空间长期闲置而得不到充分利用

16. 链表不具有的特点是（ A ）。

A、可随机访问任一个元素

B、插入删除不需要移动元素

C、不必事先估计存储空间

D、所需空间及线性表长度成正比

17. 循环链表的主要优点是（ C ）。

A、不需要头指针

B、已知某个结点的位置后，能够容易找到它的直接前驱

C、从表中任一结点出发都能扫描到整个链表

D、在进行插入、删除运算时，能更好地保证链表不断开

二、简答题

1. 描述以下三个概念的区别：头指针，头结点，表头结点

头指针：指示链表中第一个结点（即第一个数据元素的存储映像）的存储位置。

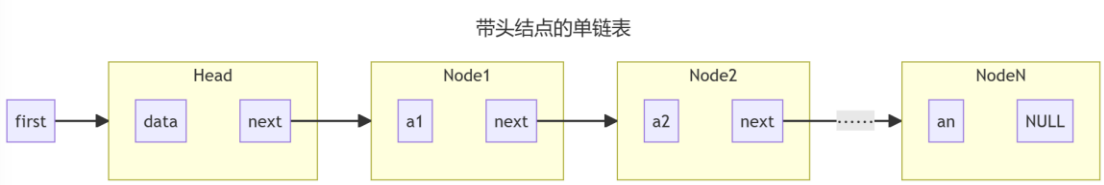
头节点：在单链表的第一个节点之前附设的一个结点。

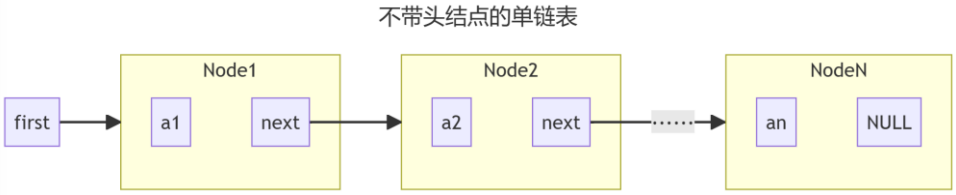
表头结点：链表中的第一个结点。

2. 带头结点的单链表和不带头结点的单链表（假设头指针是 first）为空的判断条件是什么？分别画出存储示意图。

带头结点的单链表：`first->next == NULL`

不带头结点的单链表：`first == NULL`



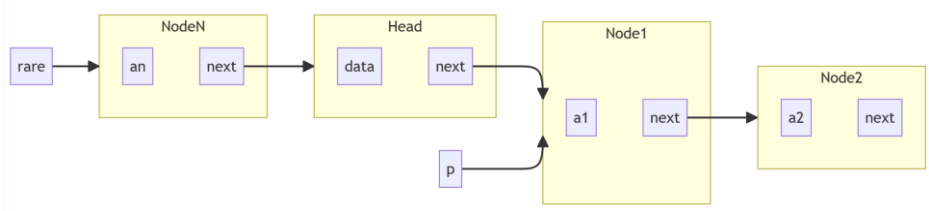

3. 在单循环链表中设置尾指针比设置头指针有哪些好处？

   1) 将两个单循环链表合并成一个表时操作更加简化。

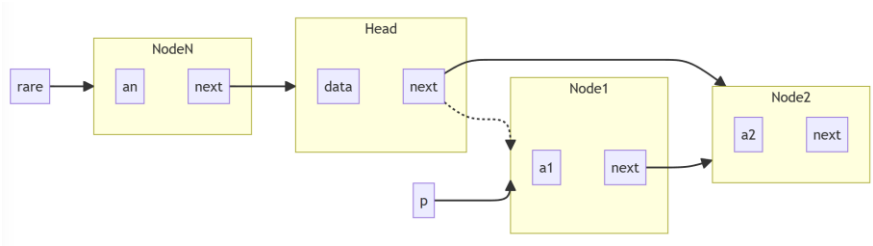
   2) 使得查找链表的开始结点和终端节点都很方便。

4. 设 rear 是指向非空的带头结点的循环单链表的尾指针，若想删除链表的第一个结点，则应执行的操作序列是什么？画出操作示意图。

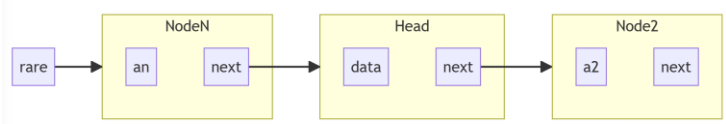
   1) `Node* p = rear->next->next;`




   2) `rear->next->next = p->next;`




   3) `free(p);`




二、算法设计（可用 C 或伪代码描述）

1. 已知数组 A[n]中的元素为整型，设计算法将其调整为左右两部分，左边所有元素为奇数，右边所有元素为偶数，并要求算法的时间复杂度为 $O(n)$。

```
void separate(int arr[], int n)
{
    int left = 0;
    int right = n - 1;

    while(left < right)
    {
        while(arr[left] % 2 == 1)
        {
            left++;
        }

        while(arr[right] % 2 == 0)
        {
            right--;
        }

        if (left < right)
        {
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
        }
    }
}
```

2. 为顺序表类 SeqList 增加一个成员函数，删除顺序表中所有值在 x 和 y 之间的所有元素，要求时间性能为 $O(n)$，空间性能为 $O(1)$。

```
void deleteRange(int arr[], int length, int x, int y)
{
    int i, j = 0;

   for(i = 0; i < length; i++)
   {
        if(arr[i] < x || arr[i] > y)
        {
        arr[j] = arr[i];
        j++;
        }
    }
}
```

3. 判断循环单链表是否递增有序。

```
_Bool isRising(Node* head)
{
    Node* p = head;

    do
    {
        if(p->data > p->next->data)
        {
            return 0;
        }
        p = p->next;
    }while(p != head);

    return 1;
}
```

4. 通过遍历一趟，将单链表中所有结点的链接方向逆转，仍利用原表的存储空间。

```
Node* reverse(Node* head)
{
    Node* prev = NULL;
    Node* p = head;
    Node* nextNode = NULL;

    while(p != NULL)
    {
        nextNode = p->next;
        p->next = prev;
        prev = p;
        p = nextNode;
    }

    return prev;
}
```