第2章练习题

**一. 单选题（共 30 分）**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | 线性表采用链接存储时，其地址（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | 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. | 不一定是连续的 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2. | 以下不是链表的特点的是（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 可随机访问任一元素 | B. | 插入、删除不需要移动元素 | | C. | 不必事先估计存储空间 | D. | 所需空间与线性表长度成正比 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3. | 若某线性表中最常用的操作取第 i 个元素和指第 i 个元素的前趋，则采用\_\_\_\_\_存储方法最节省时间（i 为任意的数据元素序号，从 1 开始）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 顺序表 | B. | 单链表 | | C. | 双链表 | D. | 单循环链表 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4. | 如下图所示的链表中，若要在指针 p 所指针结点之后插入指针 s 所指向的结点，正确的语句顺序应为（ ）。（1 分）     |  |  |  |  | | --- | --- | --- | --- | | A. | s->next=p->next; p->next = s; | B. | s=p->next; p->next=s->next; | | C. | p->next=s->next; s=p->next; | D. | p->next=s->next; p = s->next; | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5. | 可以用（ ）定义一个完整的数据结构。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 数据元素 | B. | 数据对象 | | C. | 数据关系 | D. | 抽象数据类型 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6. | 线性表的顺序存储结构是一种（ ）的存储结构。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 随机存取 | B. | 顺序存取 | | C. | 索引存取 | D. | 散列存取 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7. | 线性表的链接存储结构是一种（ ）的存储结构。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 随机存取 | B. | 顺序存取 | | C. | 索引存取 | D. | 散列存取 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8. | 以下是单链表的销毁算法，请选择填空。 （1 分）  template<typename DT> LinkList<DT>::~LinkList() {  Node \*p = first;  while (p)  {  Node \*q = p->next;  delete p;  \_\_\_\_\_\_\_\_\_\_;  } }   |  |  |  |  | | --- | --- | --- | --- | | A. | q = p | B. | p = q | | C. | p = p->next | D. | q = q->next | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9. | 设线性表有 n 个元素，以下操作中，（ ）在顺序表上实现比在链表上实现效率更高。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 输出第 i (1 ≤ i ≤ n) 个元素的值。 | B. | 交换第 1 个和第 2 个元素的值 | | C. | 顺序输出所有 n 个元素 | D. | 查找与给定值 x 相等的元素在线性表中的序号。 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10. | 以下哪一个是顺序表的特点（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 可随机访问任一元素 | B. | 插入、删除不需要移动元素 | | C. | 不必事先估计存储空间 | D. | 所需空间与线性表长度成正比 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11. | 顺序存储结构中数据元素之间的逻辑关系是由（ ）表示的。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 线性结构 | B. | 非线性结构 | | C. | 存储位置 | D. | 指针 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12. | 链式存储结构中数据元素之间的逻辑关系是由（ ）表示的。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 线性结构 | B. | 非线性结构 | | C. | 存储位置 | D. | 指针 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13. | 在等概率的情况下，执行插入和删除操作时间复杂度为O(n)的线性表是（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 顺序表 | B. | 单链表 | | C. | 顺序栈 | D. | 链栈 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14. | 单链表设置头结点的目的是（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 为了运算方便 | B. | 为了保存链表长度 | | C. | 为了按位置查找操作更快 | D. | 为了使用链表尾部的空标志起作用 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15. | 单循环链表如图所示，若结点指针 p 指向该链表中某结点，可以通过 p 判断 p 所指向的结点是不是尾结点？（1 分）     |  |  |  |  | | --- | --- | --- | --- | | A. | p->next == NULL | B. | p->next == first | | C. | p == NULL | D. | first == NULL | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16. | 单循环链表如图所示，设工作指针（指向有效结点） p 初始位置为 first->next 若要循环遍历该链表，则循环的条件应写为（ ）（1 分）     |  |  |  |  | | --- | --- | --- | --- | | A. | p->next != NULL | B. | p->next != first | | C. | p != first | D. | first == NULL | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17. | 顺序表中第一个元素的存储地址为100，每个元素长度为2，则第5个元素的地址是（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 110 | B. | 108 | | C. | 100 | D. | 120 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 18. | 链式存储的存储结构所占存储空间（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 分两部分，一部分存放结点值，一部分存放表示结点之间关系的指针 | B. | 一个部分，存放结点值 | | C. | 一个部分，存放表示结点之间关系的指针 | D. | 分两部分，一部分存放结点值，一部分存放结点所占单元数 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 19. | 向一个有127个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动的元素个数为（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 8 | B. | 63.5 | | C. | 63 | D. | 7 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20. | 线性表L在（ ）情况下适用于使用链式结构实现。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 需经常修改L中的结点 | B. | 需要不断对L进行删除插入操作 | | C. | L中含有大量的结点 | D. | L中结点结构复杂 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 21. | 线性表L，下列说法正确的是（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 每个元素都有一个直接前驱和一个直接后继 | B. | 线性表中至少有一个元素 | | C. | 表中诸元素的排列必须是由小到大或由大到小 | D. | 除第一个和最后一个元素外，其余每一个元素都有一个且仅有一个直接前驱和直接后继 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 22. | 在线性表的下列操作中，不改变数据元素之间结构关系的操作是（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 插入 | B. | 删除 | | C. | 排序 | D. | 定位查找 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 23. | 已知表头元素为c的单链表在内存中的存储状态如图所示，先将f存放于1014H并处插入到单链表中，若f在逻辑上位于a和e之间，则a，e，f的“链接地址”一次是（ ）。（1 分）     |  |  |  |  | | --- | --- | --- | --- | | A. | 1010H， 1014H，1004H | B. | 1010H，1004H， 1014H | | C. | 1014H，1010H，1004H | D. | 1014H，1004H，1010H | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24. | 如下图所示的链表中，若要在指针 p 所指针结点之后删除指针q 所指向的结点，正确的语句应为（ ）。（1 分）     |  |  |  |  | | --- | --- | --- | --- | | A. | p->next=q->next; | B. | q->next=p->next; | | C. | p=q->next; | D. | q=p->next; | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 25. | 若某线性表中最常见的操作时取第i个元素和找第i个元素的前驱，则采用（ ）存储方法最节省时间。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 顺序表 | B. | 单链表 | | C. | 双链表 | D. | 单循环链表 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 26. | 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之间插入结点s，则执行（ ）操作。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | s->next=p-next;p-next=s; | B. | q->next=s; s->next=p; | | C. | p->next=s->next;s-next=p; | D. | p->next=s;s->next=q; | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 27. | 在一个长度为n的顺序表中第i个元素（1<=i<=n）之前插入一个元素时，需向后移动（ ）个元素。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | n-1 | B. | n-i+1 | | C. | n-i | D. | n-i-1 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 28. | 若某线性表中最常见的操作时取第i个元素和找第i个元素的前驱，则采用（ ）存储方法最节省时间。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 顺序表 | B. | 单链表 | | C. | 双链表 | D. | 单循环链表 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 29. | 链式存储的存储结构所占存储空间（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 分两部分，一部分存放结点值，一部分存放表示结点之间关系的指针 | B. | 一个部分，存放结点值 | | C. | 一个部分，存放表示结点之间关系的指针 | D. | 分两部分，一部分存放结点值，一部分存放结点所占单元数 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 30. | 在双向链表存储结构中，删除p所指的结点时须修改指针（ ）。（1 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | p->next->prior=p->prior;p->prior->next=p->next; | B. | p->next=p->next->nextr;p->next->prior=p; | | C. | p->prio->next=p;p->prior=p->prior->prior; | D. | p->prior=p->next-next;p->next=p->prior->prior; | |

**二. 多选题（共 4 分）**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | 线性表L=(a1,a2,……an)，下列说法错误的是（ ）。（2 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 每个元素都有一个直接前驱 | B. | 每个元素都有一个直接后续 | | C. | 线性表中至少有一个元素 | D. | 表中诸元素的排列必须是由小到大或由大到小 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2. | 线性表L=(a1,a2,……an)，下列说法正确的是（ ）。（2 分）   |  |  |  |  | | --- | --- | --- | --- | | A. | 除第一个元素外，每个元素都有且仅有一个直接前驱 | B. | 除最后一个元素外，每个元素都有且仅有一个直接后续 | | C. | 线性表中可以无元素 | D. | 每个元素都有一个直接前驱和直接后续 | |

**三. 判断题（共 13 分）**

|  |  |
| --- | --- |
| 1. | 线性表的元素个数可以是 0 个。（1 分） |

|  |  |
| --- | --- |
| 2. | 线性表的每个元素都有且仅有一个直接前驱和一个直接后继。（1 分） |

|  |  |
| --- | --- |
| 3. | 线性表的顺序存储结构优于链式存储结构。（1 分） |

|  |  |
| --- | --- |
| 4. | 设有定义 int \*p, \*q; 若 p == q 成立，则 \*p == \*q 也成立。（1 分） |

|  |  |
| --- | --- |
| 5. | 顺序表的缺点是：插入和删除操作需移动大量元素；表的容量难以确定；造成存储空间的“碎片”。（1 分） |

|  |  |
| --- | --- |
| 6. | 不带头结点的单链表中实现插入操作，算法不仅冗长，而且需要考虑在表头操作的特殊情况，容易出现错误。（1 分） |

|  |  |
| --- | --- |
| 7. | 以下算法可以获取带头结点的单链表的长度（元素个数）。 （1 分） template <typename DT> int LinkList<DT>::GetLength() {  p = first->next; i = 0;   while (p != NULL)   {   i++;   p++;   }   return i;  } |

|  |  |
| --- | --- |
| 8. | 线性表的链式存储结构优于顺序存储结构。（1 分） |

|  |  |
| --- | --- |
| 9. | 线性结构的基本特征是：除首尾元素外，每个元素有且仅有一个直接前趋和一个直接后继，首元素无前驱，尾元素无后继。（1 分） |

|  |  |
| --- | --- |
| 10. | 线性表是具有相同类型的数据元素的有限序列。（1 分） |

|  |  |
| --- | --- |
| 11. | 用数组存储顺序表，需要分配固定长度的数组空间，且数组的长度MaxSize需大于当前线性表的长速度length。（1 分） |

|  |  |
| --- | --- |
| 12. | 线性表的逻辑结构和存储结构总是一致的。（1 分） |

|  |  |
| --- | --- |
| 13. | 在单链表中，要取得某个元素，只要知道该元素所在结点的地址即可，因此单链表是随机存取结构。（1 分） |

**四. 填空题（共 38 分）**

|  |  |
| --- | --- |
| 1. | 【1】是用一段地址连续的存储单元依次存储线性表的数据元素，通常使用【2】来实现。（2 分） |

|  |  |
| --- | --- |
| 2. | 有一个长度为 10 顺序表，元素的序号从 1 开始，若在第 3 个元素之前插入一个元素，需向后移动【1】个元素； （2 分） 若当前长度仍然为 10，要删除第 3 个元素，需向前移动【2】个元素。 |

|  |  |
| --- | --- |
| 3. | 单链表是用一组【1】的存储单元存放线性表的元素。 （1 分） （限填两个汉字） |

|  |  |
| --- | --- |
| 4. | 程序填空题，线性表存放在数组 data[arrsize] 的前 elenum 个单元中，且递增有序（从小到大排列）。以下算法是将元素 x 插入到线性表的适当位置上，以保持线性表的有序性，请填充完善代码。 （5 分）  const int arrsize=100; class SeqList { private:  int data[arrsize];   int elenum;  public:  Insert(int x);   ...  };  void SeqList::Insert(int x) {  if (elenum==arrsize) throw "溢出";   for (i=【1】; i>=0 && 【2】; i--)   {   data[【3】]=data[【4】];   }   data[【5】]=x;  } } |

|  |  |
| --- | --- |
| 5. | 设单链表中指针 p 指向结点 ai，若要删除 ai 的后继结点（假设 ai 存在后继结点），则需修改指针的操作为【1】 （1 分）  注意：只使用指针 p 表示，只写修改指针这一条语句。 |

|  |  |
| --- | --- |
| 6. | 以下是顺序表的删除算法，请填充（不要填写多余的分号）。 （4 分）  template <class DT> DT SeqList<DT>::Delete(int i) {  if (i < 1 || i > length) throw "删除位置不合法";  DT x = data[【1】];  【2】;  for (int j = i; j < length; j++) 【3】;  【4】; } |

|  |  |
| --- | --- |
| 7. | 以下是遍历打印单链表数据元素的算法，请填充（不要填写多余的分号） （3 分） template <class DataType> void LinkList<DataType> :: PrintList( ) {  Node \*p = 【1】;   while (【2】)   {   cout << p->data << " ";   【3】;   }   cout << endl;  } |

|  |  |
| --- | --- |
| 8. | 以下是顺序表的插入算法，请填充（不要填写多余的分号）。 （2 分）  template <class DT> DT SeqList<DT>::Insert(int i，DataType x) {  if (length>=MaxSize) throw "上溢";  if (i < 1 || i > length+1) throw "插入位置不合法";   for(j=length;j>=i;j- -)  data[j] = data[j-1];   【2】;   【3】;    } |

|  |  |
| --- | --- |
| 9. | 一个具有n个结点的单链表，在指针p所指结点后插入一个新结点的时间复杂度为【1】，在给定值x的结点后插入一个新结点的时间复杂度是【2】。（2 分） |

|  |  |
| --- | --- |
| 10. | 设单链表中指针p指向结点A，若要删除A的后继结点（假设A存在后继节点），则需要修改指针的操作是【1】.（1 分） |

|  |  |
| --- | --- |
| 11. | 若长度为n的非空线性表采用顺序存储结构，删除表的第i个数据元素，i的合法值应该是【1】。 （1 分）  备注：请填写不等式表达 |

|  |  |
| --- | --- |
| 12. | 若长度为n的非空线性表采用顺序存储结构，插入表的第i个数据元素，i的合法值应该是【1】。 （1 分）  备注：请填写不等式表达 |

|  |  |
| --- | --- |
| 13. | 线性表的两种存储结构分别是【1】存储结构和【2】存储结构。（2 分） |

|  |  |
| --- | --- |
| 14. | 链表中指针next表示的是当前结点【1】结点的地址。 （1 分）  备注：填写仅限两个字 |

|  |  |
| --- | --- |
| 15. | 非空顺序表中，数据元素之间的逻辑关系由元素的【1】位置来表示。 （1 分）  备注：填写仅限两个字 |

|  |  |
| --- | --- |
| 16. | 非空链表中，数据元素之间的逻辑关系由【1】来表示。 （1 分）  备注：填写仅限两个字 |

|  |  |
| --- | --- |
| 17. | 一个具有n个结点的单链表，在指针p所指结点后插入一个新结点的时间复杂度为【1】，在给定值x的结点后插入一个新结点的时间复杂度是【2】。（2 分） |

|  |  |
| --- | --- |
| 18. | 以下是单链表的插入算法，请填充（不要填写多余的分号）。 （2 分）  template <class DT> DT LinkList<DT>::Insert(int i，DataType x) {  p=first;int count=0  while (p!=NULL && count<i-1)   {  【1】;   【2】;   }   if(p==NULL)throw"位置";   else   {   s=new Node;   s->data=x;   s->next=p->next;   p->next=s;   }    } |

|  |  |
| --- | --- |
| 19. | 顺序表中第一个元素的存储地址是100，每个元素的长度是2，则第7个元素的存储地址是【1】。（1 分） |

|  |  |
| --- | --- |
| 20. | 顺序表中第10个元素的存储地址是100，每个元素的长度是2，则第5个元素的存储地址是【1】。（1 分） |

|  |  |
| --- | --- |
| 21. | 单链表中结点结构由【1】和【2】两部分组成了数据元素的存储映象。（2 分） |

**练习题答案**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**一. 单选题（共 30 分）**

1.D 2.A 3.A 4.A 5.D 6.A 7.B 8.B 9.A 10.A 11.C 12.D 13.A 14.A 15.B  
16.C 17.B 18.A 19.B 20.B 21.D 22.D 23.C 24.A 25.A 26.B 27.B 28.A 29.A 30.A

**二. 多选题（共 4 分）**

1.ABCD 2.ABC

**三. 判断题（共 13 分）**

1.√ 2.× 3.× 4.√ 5.√ 6.√ 7.× 8.× 9.√ 10.√ 11.√ 12.×  
13.×

**四. 填空题（共 38 分）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. | |  |  | | --- | --- | | 【1】 | 顺序表 |  |  |  |  | | --- | --- | --- | | 【2】 | 一维数组 | 数组 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2. | |  |  | | --- | --- | | 【1】 | 8 |  |  |  | | --- | --- | | 【2】 | 7 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 3. | |  |  | | --- | --- | | 【1】 | 任意 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4. | |  |  | | --- | --- | | 【1】 | elenum-1 |  |  |  |  | | --- | --- | --- | | 【2】 | x<data[i] | data[i]>x |  |  |  |  | | --- | --- | --- | | 【3】 | i+1 | 1+i |  |  |  | | --- | --- | | 【4】 | i |  |  |  |  | | --- | --- | --- | | 【5】 | i+1 | 1+i | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5. | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 【1】 | p->next=p->next->next | p->next=(p->next)->next | p->next=p->next->next; | p->next=(p->next)->next; | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6. | |  |  | | --- | --- | | 【1】 | i-1 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 【2】 | length-- | --length | length=length-1 | length-=1 |  |  |  | | --- | --- | | 【3】 | data[j-1] = data[j] |  |  |  | | --- | --- | | 【4】 | return x | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7. | |  |  | | --- | --- | | 【1】 | first->next |  |  |  |  |  | | --- | --- | --- | --- | | 【2】 | p!=NULL | NULL!=p | p |  |  |  | | --- | --- | | 【3】 | p=p->next | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8. | |  |  | | --- | --- | | 【1】 | data[i-1]=x |  |  |  |  | | --- | --- | --- | | 【2】 | length++ | length=length+1 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 9. | |  |  | | --- | --- | | 【1】 | O(1) |  |  |  | | --- | --- | | 【2】 | O(n) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10. | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 【1】 | p->next=(p->next)->next; | p->next=p->next->next; | p->next=(p->next)->next | p->next=p->next->next | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 11. | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 【1】 | 1<=i<=n | i>=1,i<=n | i>=1且i<=n | i<=n且i>=1 | i<=n，i>=1 | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 12. | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 【1】 | 1<=i<=n+1 | i>=1,i<=n+1 | i>=1且i<=n+1 | i<=n+1且i>=1 | i<=n+1，i>=1 | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 13. | |  |  | | --- | --- | | 【1】 | 顺序 |  |  |  |  |  | | --- | --- | --- | --- | | 【2】 | 链式 | 链接 | 连接式 | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14. | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 【1】 | 后继 | 后面 | 后头 | 后跟 | 后邻 | 后接 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 15. | |  |  | | --- | --- | | 【1】 | 存储 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 16. | |  |  | | --- | --- | | 【1】 | 指针 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 17. | |  |  | | --- | --- | | 【1】 | O(1) |  |  |  | | --- | --- | | 【2】 | O(n) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 18. | |  |  | | --- | --- | | 【1】 | p=p->next |  |  |  |  | | --- | --- | --- | | 【2】 | count++ | count=count+1 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 19. | |  |  | | --- | --- | | 【1】 | 112 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 20. | |  |  | | --- | --- | | 【1】 | 90 | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 21. | |  |  |  | | --- | --- | --- | | 【1】 | 数据域 | 指针域 |  |  |  |  | | --- | --- | --- | | 【2】 | 指针域 | 数据域 | |