# 第2章 结点逻辑关系

# 为线性的结构——线性表

## 习题

#### 一、单项选择题

1．数据在计算机存储器内表示时，物理地址连续，数据间的逻辑关系依靠其物理地址间的连续性来表达，称为（ C ）。

（A）存储结构 （B）逻辑结构

（C）顺序存储结构 （D）链式存储结构

2．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个结点的直接前趋（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

3．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，平均要移动几个元素（ C ）。

（A）8 （B）63.5 （C）63 （D）7

4．链接存储的存储结构所占存储空间（ A ）。

（A）分两部分，一部分存放结点值，另一部分存放表示结点间关系的指针

（B）只有一部分，存放结点值

（C）只有一部分，存储表示结点间关系的指针

（D）分两部分，一部分存放结点值，另一部分存放结点所占单元数

5．链表是一种采用哪种存储结构存储的线性表（ B ）。

（A）顺序 （B）链式 （C）星式 （D）网状

6．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

7．线性表Ｌ在哪种情况下适用于使用链式结构实现（ B ）。

（A）需经常修改Ｌ中的结点值 （B）需不断对Ｌ进行删除插入

（C）Ｌ中含有大量的结点 （D）Ｌ中结点结构复杂

8．单链表的存储密度（ C ）。

（A）大于1 （B）等于1 （C）小于1 （D）不能确定

9．下述哪一条是顺序存储结构的优点？（ 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

10．下面关于线性表的叙述中，错误的是哪一个（ B ）？

（A）线性表采用顺序存储，必须占用一片连续的存储单元

（B）线性表采用顺序存储，便于进行插入和删除操作

（C）线性表采用链接存储，不必占用一片连续的存储单元

（D）线性表采用链接存储，便于插入和删除操作

11．若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算，则利用（ A ）存储方式最节省时间。

（A）顺序表 （B）双链表

（C）带头结点的双循环链表 （D）单循环链表

12．某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，则采用（ D ）存储方式最节省运算时间。

（A）单链表 （B）仅有头指针的单循环链表

（C）双链表 （D）仅有尾指针的单循环链表

13．链表不具有的特点是（ B ）。

（A）插入、删除不需要移动元素 （B）可随机访问任一元素

（C）不必事先估计存储空间 （D）所需空间与线性长度成正比

14．下面的叙述不正确的是（ C ）。

（A）线性表在链式存储时，查找第i个元素的时间同i的值成正比

（B）线性表在链式存储时，删除第i个元素的时间同i的值无关

（C）线性表在顺序存储时，查找第i个元素的时间同i 的值成正比

（D）线性表在顺序存储时，查找第i个元素的时间同i的值无关

15．在一个以 h 为头的单循环链中，p 指针指向链尾的条件是（ C ）。

（A）P->next=h （B）p->next=NIL

（C）p->next->next=h （D）p->data=-1

*16．完成在双循环链表结点p之后插入s的操作是（ D ）。 【选做题】*

（A）p->next=s ; s->prior=p; p->next->prior=s ; s->next=p->next;

（B）p->next->prior=s; p->next=s; s->prior=p; s->next=p->next;

（C）s->prior=p; s->next=p->next; p->next=s; p->next->prior=s ;

（D）s->prior=p; s->next=p->next; p->next->prior=s ; p->next=s;

17．在单链表指针为p的结点之后插入指针为s的结点，正确的操作是（ B ）。

（A）p->next=s;s->next=p->next;

（B）s->next=p->next;p->next=s;

（C）p->next=s;p->next=s.next;

（D）p->next=s->next;p->next=s;

18．对于一个头指针为head的带头结点的单链表，判定该表为空表的条件是（ B ）。

（A）head==NULL （B）head→next==NULL

（C）head→next==head （D）head!=NULL

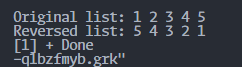
#### 二、算法设计题

3．设有一线性表E={e1,e2,…,en−1,en)，试设计一个算法，将线性表逆置，即使元素排列次序颠倒过来，成为逆线性表E′={en,en−1,…,e2,e1)，要求逆线性表占用原线性表空间，并且用顺序和单链表两种方法表示，写出不同的处理过程。

1. 顺序表

|  |
| --- |
| */\*\**  *\* @project        : docs*  *\* @file           : sqlist.cpp*  *\* @dir            : ~/projects/docs/Homework/Algorithms/oj*  *\* @date           : 2024/03/09*  *\* @author         : Kaikai*  *\* @brief          : 线性表逆置算法*  *\* \*/*  #include <iostream>  using namespace std;  #define MAX\_SIZE 100  */// @brief 线性表数据结构*  typedef int Elemtype;  typedef struct  {      Elemtype elem[MAX\_SIZE];      int length;  } Sqlist;  */// @brief 线性表逆置算法*  */// @param l 线性表*  */// @details 中间为界，前后交换*  void ListReverse(Sqlist &*l*)  {      for (int i = 0; i < *l*.length / 2; i++)      {          Elemtype temp = *l*.elem[i];  *l*.elem[i] = *l*.elem[*l*.length - i - 1];  *l*.elem[*l*.length - i - 1] = temp;      }  }  */// @brief 入口*  */// @return 0*  int main()  {  *// 构造线性表*      Sqlist list;      list.length = 5;      for (int i = 0; i < list.length; i++)      {          list.elem[i] = i + 1;      }  *// 打印原线性表*      cout << "Original list: ";      for (int i = 0; i < list.length; i++)      {          cout << list.elem[i] << " ";      }      cout << endl;  *// 线性表逆置*      ListReverse(list);  *// 打印逆置后线性表*      cout << "Reversed list: ";      for (int i = 0; i < list.length; i++)      {          cout << list.elem[i] << " ";      }      cout << endl;      return 0;  } |

程序运行结果：



4．已知带头结点的动态单链表L中的结点是按整数值递增排列的，试写一算法将值为x的结点插入表L中，使L仍然有序。

|  |
| --- |
| */\*\**  *\* @project        : docs*  *\* @file           : inlink.cpp*  *\* @dir            : ~/projects/docs/Homework/Algorithms/oj*  *\* @date           : 2024/03/09*  *\* @author         : Kaikai*  *\* @brief          : 单向链表（递增排列）插入元素 翻转链表*  *\* \*/*  #include <iostream>  using namespace std;  #define MAX\_SIZE 100  */// @brief 单向链表*  struct ListNode  {      int val;      ListNode \*next;      ListNode(int *x*) : val(*x*), next(nullptr) {}  };  */// @brief 创建单链表*  */// @param 指定长度*  ListNode \*CreateList(int *length*)  {      ListNode \*head = new ListNode(-1); *// 头结点的值可以设为任意值，这里设为-1*      ListNode \*tail = head;      for (int i = 1; i <= *length*; ++i)      {          ListNode \*newNode = new ListNode(i \* 10);          tail->next = newNode;          tail = newNode;      }      return head;  }  */// @brief 在带头结点的单链表中插入值为 val 的结点*  */// @param head 头节点*  */// @param val 值*  */// @param inc 递增 True or 递减 False*  void InsertNode(ListNode \**head*, int *val*, bool *inc* = true)  {      ListNode \*newNode = new ListNode(*val*);      ListNode \*p = *head*;      while (p->next != nullptr && *inc* ? p->next->val < *val* : p->next->val > *val*)      {          p = p->next;      }      newNode->next = p->next;      p->next = newNode;  }  ListNode \*Reverse(ListNode \**node*)  {      if (*node*->next == NULL) *// 尾节点*      {          return *node*;      }      ListNode \*tail = Reverse(*node*->next);  *node*->next->next = *node*;      return tail;  }  */// @brief 反转链表*  */// @param head 头节点，反转前后头节点不变*  void ReverseList(ListNode \**head*)  {      ListNode \*tail = Reverse(*head*);  *head*->next->next = NULL;  *head*->next = tail;  }  void ReverseListWhile(ListNode \**head*)  {      ListNode \*prior = *head*;      ListNode \*curr = *head*->next;      ListNode \*next = *head*->next->next;      while (next != nullptr)      {          curr->next = prior;          prior = curr;          curr = next;          next = curr->next;      }      curr->next = prior;  *head*->next->next = NULL;  *head*->next = curr;  }  */// @brief 打印带头结点的单链表*  */// @param head 头节点（值不参与打印）*  void PrintList(ListNode \**head*)  {      ListNode \*p = *head*->next;      while (p)      {          cout << p->val << " ";          p = p->next;      }      cout << endl;  }  int main()  {      int length = 10;      ListNode \*head = CreateList(length); *// 创建带头结点的单链表*      int insert\_data;      cout << "Before Insert: ";      PrintList(head); *// 打印链表*      cout << "Data: ";      cin >> insert\_data; *// 输入插入位置和被插入元素值*      cout << "After Insert: ";      InsertNode(head, insert\_data); *// 插入结点*      PrintList(head); *// 打印链表*      cout << "After Reverse(Inter): ";      ReverseList(head);      PrintList(head);      cout << "After Reverse(While): ";      ReverseListWhile(head);      PrintList(head);      return 0;  } |

程序运行结果：

