**河南开封科技传媒学院实验报告**

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

**20 -20 学年第 学期**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 刘东 | | 学号： | 2136101045 | 专业： | 软件工程 | |
| 所在学院： | | 理工学院 | | | 指导老师： | | 袁帅 |
| 实验时间： | | 2023/3/8 | | | 实验地点： | | 6503 |
| 课程名称： | | 数据结构 | | | | | |
| 实验题目： | | 第三周实验 | | | | | |
| 实验目的： | | | | | | | |
| 实验内容：   1. 实现一个单链表，满足如下操作： 2. 把数组A[8]={1,2,3,4,5,100,200,300}保存至单链表 3. 在单链表第6个位置插入数值586，并显示单链表此时新的情况，比如节点数 4. 然后对当前单链表第5个节点进行删除操作，并显示删除后单链表的状态 5. 显示所有操作的头结点地址。   思考题   1. 采用循环链表的方式解决如下问题：   有n个小孩围成一圈，给他们从1开始依次标号，从编号为1的小孩开始报数，数到第m个小孩出列，然后从出列的下一个小孩重新开始报数，数到第m个小孩又出列，依次类推，直到所有的小孩全部出列为止，求整个出列序列。（如当n=6，m=5时的出列序列是5,4,6,2,3,1,）   1. 上述问题可否用顺序表（数组）解决？ 2. 上述方式可否用递归解决？ | | | | | | | |
| 实验代码：  #include<stdio.h>  #include<stdlib.h>  #include<iostream>  using namespace std;  typedef int ElemType;  //创建链表  typedef struct Node {  ElemType data;  struct Node\* next;  }Node;  typedef struct Node\* LinkList;  int printList(LinkList \*L) {  LinkList p = (LinkList)malloc(sizeof(Node));    p = (\*L)->next;  while(p != NULL) {  cout << p->data << " ";  cout << "节点地址：" << &p->next <<endl;  p = p->next;  }  cout << endl;  return 1;  }  /\* 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数 \*/  int ListLength(LinkList L)  {  int i=0;  LinkList p=L->next; /\* p指向第一个结点 \*/  while(p)  {  i++;  p=p->next;  }  return i;  }  //链表的头插法  int ListInsert(LinkList\* L,int i,ElemType e) {  int j;  LinkList p,s;  p = \*L;  j = 1;  while (p && j < i) { //寻找第i个结点  p = p->next;  ++j;  }  if (!p || j > i) {  return 0;  }  s = (LinkList)malloc(sizeof(Node));  s->data = e;  //头插法  s->next = p->next;  p->next = s;  return 1;  }  //链表的删除  int ListDelete(LinkList \*L,int i,ElemType \*e) {  int j;  LinkList p,q;  p = \*L;  j = 1;  while (p->next && j < i) {  p = p->next;  ++j;  }  if (!(p->next) || j > i) {  return 0;  }  //链表的删除  q = p->next;  p->next = q->next;  \*e = q->data;  free(q);  return 1;  }  //单链表的整表创建（尾插法）  int CreateListHead\_2(LinkList \*L,int n) {  int A[8]={1,2,3,4,5,100,200,300};  LinkList p,r; //p为新插入的结点 r为链表中最后的结点 用于尾插  int i;  \*L = (LinkList)malloc(sizeof(Node));  r = \*L;  for(i = 0; i < n; i++) {  p = (LinkList)malloc(sizeof(Node));  //尾插法  p->data = A[i];  r->next = p;  r = p;  }  r->next=NULL;  return 1;  }  void test() {  LinkList head = (LinkList)malloc(sizeof(Node));  CreateListHead\_2(&head, 8);  cout << "插入后：" << endl;  printList(&head);  int i = 6, j = 5, e = 586;  cout << "在单链表第6个位置插入数值586，并显示单链表此时新的情况，比如节点数" <<endl;  ListInsert(&head,i, e);  cout << "新的节点数变成：" << ListLength(head) <<endl;  cout << "现在的链表数据是：" << endl;  printList(&head);  ListDelete(&head, j, &e);  cout << "对当前单链表第5个节点进行删除操作：" << endl;  cout << "新的节点数变成：" << ListLength(head) <<endl;  cout << "现在的链表数据是：" << endl;  printList(&head);      }  int main() {  test();  return 0;  }  思考：  #include<stdio.h>  #include<stdlib.h>  #include<iostream>  using namespace std;  typedef int ElemType;  //创建链表  typedef struct Node {  ElemType data;  struct Node\* next;  }Node;  typedef struct Node\* LinkList;  int printList(LinkList \*L) {  LinkList p = (LinkList)malloc(sizeof(Node));  cout << "头节点地址：" << &L <<endl;  p = (\*L)->next;  while(p != NULL) {  cout << p->data << " ";  p = p->next;  }  cout << endl;  return 1;  }  /\* 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数 \*/  int ListLength(LinkList L)  {  int i=0;  LinkList p=L->next; /\* p指向第一个结点 \*/  while(p)  {  i++;  p=p->next;  }  return i;  }  //链表的头插法  int ListInsert(LinkList\* L,int i,ElemType e) {  int j;  LinkList p,s;  p = \*L;  j = 1;  while (p && j < i) { //寻找第i个结点  p = p->next;  ++j;  }  if (!p || j > i) {  return 0;  }  s = (LinkList)malloc(sizeof(Node));  s->data = e;  //头插法  s->next = p->next;  p->next = s;  return 1;  }  //链表的删除  int ListDelete(LinkList \*L,int i,ElemType \*e) {  int j;  LinkList p,q;  p = \*L;  j = 1;  while (p->next && j < i) {  p = p->next;  ++j;  }  if (!(p->next) || j > i) {  return 0;  }  //链表的删除  q = p->next;  p->next = q->next;  \*e = q->data;  free(q);  return 1;  }  //单链表的整表创建（尾插法）  int CreateListHead\_2(LinkList \*L,int n) {  LinkList p,r; //p为新插入的结点 r为链表中最后的结点 用于尾插  int i;  \*L = (LinkList)malloc(sizeof(Node));  r = \*L;  for(i = 1; i <= n; i++) {  p = (LinkList)malloc(sizeof(Node));  //尾插法  p->data = i;  r->next = p;  r = p;  }  r->next=(\*L)->next;  return 1;  }  void test01(LinkList \*L, int m) {  LinkList r, p;  p = (LinkList)malloc(sizeof(Node));  int i = 1;  p = \*L;  while(p->next != p) {  if (i == m) {  r = p->next;  cout << p->next->data << " ";  p->next = r->next;  i = 1;  continue;  }  i++;  p = p->next;  }  cout << p->data << endl; //输出最后一个数据  }  void test02(int n, int m) {  int a[n];  for (int i = 0; i < n; i++) {  a[i] = i + 1;  }  int p = n;  int t = 0;  while(p != 1) {  for (int j = 1; j <= 5; j++) {  if (j == 5) {  cout << a[t] << " ";  for (int i = t; i <= p; i++) {  a[t] = a[t + 1];  }  p--;  t--;  }  t = (t + 1) % p;  }  }  cout << a[0] << endl;      }  int test03(int sum,int value,int n)  {  if(n==1)  return (sum+value-1)%sum;  else  return (test03(sum-1,value,n-1)+value)%sum;  }  void test() {  LinkList head = (LinkList)malloc(sizeof(Node));  int n = 6, m = 5;  CreateListHead\_2(&head, n);    //思考1  cout << "1.采用循环链表的方式解决：" << endl;  test01(&head, m);    //思考2  cout << "2.用顺序表（数组）解决：" << endl;  test02(n, m);    //思考3  cout << "3.用递归解决：" << endl;  for(int i=1;i<=n;i++)  {  cout << test03(n,m,i)+1 << " ";  }      }  int main() {  test();  return 0;  } | | | | | | | |
| 实验结果截图及实验心得： | | | | | | | |
|  | | | | | | | |