哈尔滨工程大学

实 验 报 告

实 验 名 称：­­

班 级：

学 号：

姓 名：

实 验 时 间：

成 绩：

指 导 教 师：

实验室名称：

**哈尔滨工程大学实验室与资产管理处 制**

|  |
| --- |
| 实验名称： 约瑟夫环 |
| 一、问题描述  设有编号为 1,2,…,n 的 n（n＞0）个人围成一个圈，每个人持有一个密码 m。 从第一个人开始报数，报到 m 时停止报数，报 m 的人出圈，再从他的下一个人 起重新报数，报到 m 时停止报数，报 m 的出圈，……，如此下去，直到所有人 全部出圈为止。当任意给定 n 和 m 后，设计算法求 n 个人出圈的次序。  二、数据结构设计  //定义一个结构体，存储数据以及指向下一个节点的指针  struct Node {  //节点存储的数据  int data;  //该节点的指针，指向下一个节点（存储的是下一个节点的地址）  struct Node \* next;  };  //重定义，定义struct Node的别名为 Node，方便后续定义相应的函数及变量  typedef struct Node Node;   1. 算法设计 2. //定义一个结构体，存储数据以及指向下一个节点的指针   struct Node {  //节点存储的数据  int data;  //该节点的指针，指向下一个节点（存储的是下一个节点的地址）  struct Node \* next;  };  //重定义，定义struct Node的别名为 Node，方便后续定义相应的函数及变量  typedef struct Node Node;  2.定义一个函数，利用尾插法创建链表  //函数名: createCircularLinkedList  //函数返回值类型：指向结构体的指针（这个指针是指向所创建链表的头节点的指针）  //参数：n（所要创建的链表的长度）  //定义一个循环链表创建函数，下一行的 "\*"代表函数的返回值为指针类型  Node \* createCircularLinkedList(int n) {    //如果 n小于或等于0，创建失败，函数返回为空  if (n <= 0) {  return NULL;  }    //定义一个结构体指针head，并动态分配存储空间,使头节点的指针  Node\* head = (Node\*)malloc(sizeof(Node));  //初始化头结点存储数据为1  head->data = 1;  // 初始时只有一个节点指向自己  head->next = head;  //定义一个结构体（结构体指针）current，暂时指向头结点  Node\* current = head;  //定义一个整型变量i  int i;  //做一个循环，挨个儿创建结点 （从第二个结点开始创建，因为第一个结点是头结点，已经被创建了）  for (i = 2; i <= n; i++) {  //定义一个结构体（结构体指针）/（新结点），并同时分配存储空间  Node\* newNode = (Node\*)malloc(sizeof(Node));  //初始化新结点的存储的数据  newNode->data = i;  //将新结点的指针指向头结点，形成循环  newNode->next = head;  //让原本和头结点相同的current结构体指向新创建的节点  current->next = newNode;  //将current置为与newNode相等（与定义current时的操作相同）  current = newNode;  }  //返回头指针的地址（函数的返回值）  return head;  }  3.定义一个函数，根据m删除链表中的结点，并输出删除结果  //函数名: josephus  //函数返回值类型：空  //参数：n（所要创建的链表的长度），m (密码)  //定义一个约瑟夫环，并删去新的报数顺序的第m个人  void josephus(int n, int m) {    //判断m和n是否合法，不合法则返回空  if (n <= 0 || m <= 0) {  return ;  }    //创建并定义头结点  Node\* head = createCircularLinkedList(n);  //初始化current指针，指向头结点  Node\* current = head;  //做一个循环，删除所要删除的结点 （一共n次，直到把所有结点都删除掉）  while (n > 0) {    //定义一个整型变量用于循环  int i;  //做一个循环 ，移动current，直至其指向目标结点的前一个结点  for (i = 1; i < m-1; i++) {  current = current->next;  }    //定义一个temp结构体指针，指向目标结点（current的下一个结点）  Node\* temp;  //处理上面循环导致找不到第一个节点的前一个结点的情况  if(m!=1)temp = current->next;  //直接temp指向current，即指向头结点  else temp=current;    //输出目标结点的编号  printf("编号为 %d\n", temp->data);    // 下面为删除目标结点的操作：  // current代替temp，指向目标结点的下一个结点  current->next=temp->next;  //目标节点的数据同步到current中  current=temp->next;    //释放temp指针的存储空间  free(temp);    //让n进行-1，继续执行while循环  n--;  }    //所有结点删除完毕，并释放头指针的空间  free(head);    }  四、界面设计  为了使界面更加简明清楚，采用简单的printf函数：  int main() {  //定义整型变量n和m  int n, m;    printf("请输入总人数 n: ");  scanf("%d", &n);  printf("请输入密码 m: ");  scanf("%d", &m);    //处理非法数据  if (n <= 0 || m <= 0) {  printf("人数和密码必须为正整数。\n");  return 1;  }    //进行建立约瑟夫环，并根据密码删除结点的操作  josephus(n, m);    //OVER!!!  return 0;  }  五、运行测试与分析  1. 令人数为6人，密码为12 即输入 n = 6, m = 12后，界面显示如下：    2. 令人数为12人，密码为6 即输入 n = 12, m =6后，界面显示如下：    3. 令人数为9人，密码为0 即输入 n = 9, m =0后，界面显示如下：    4. 令人数为0人，密码为9 即输入 n = 0, m =9后，界面显示如下：    六、实验收获与思考  <1>思考问题：  1.采用顺序存储结构如何实现约瑟夫环问题？  ①创建顺序表  ②给顺序表赋值  ③从第一个元素开始计数，每 m 个输出对应元素值  ④该元素后方所有元素覆盖前一位元素，顺序表长度减1  ⑤从当前元素位置重新计数，遇到表尾从表头继续计数  ⑥顺序表长度为1时结束  2.如果每个人持有的密码不同，应如何实现约瑟夫环问题？  ①定义带有两个数据域的链表存储结构体：  typedef struct LNode{  int password; //数据  int position; //数据  struct LNode \*next; //指针  }  ②创建带有两个数据域的链表  ③链表赋值，进行n次循环，对结点赋值密码与位置  ④首尾链接，构成循环链表  ⑤从初始结点开始循环计数  ⑥当计数器数值与结点password值相等时，输出结点元素position值  ⑦删除结点, 从当前结点重新循环计数，进行循环  ⑧当循环链表中只剩下唯一结点，结束、  <2>实验收获：  通过这次实验，首先我对线性表有了更加深刻的理解，在编写代码的过程中进一步熟悉了链表的实现、操作和使用，对链表中节点的插入和删除等操作更加了解。除此之外，在这个过程中，我对于上学期C语言和C++的相关知识有了更好的掌握，能够熟练运用C/C++语言实现一些功能，解决生活中的问题。 |