**实验报告**

**学院（系）名称：**计算机科学与工程学院

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 王帆 | | **学号** | | 20152180 | | | **专业** | | 计算机科学与技术 | |
| **班级** | | 15级1班 | | **实验项目** | | 实验一：线性表应用 | | | | | | |
| **课程名称** | | | | 数据结构与算法 | | | | | **课程代码** | | 0668016 | |
| **实验时间** | | | | 2017年 3月 10日 第5节 | | | | | **实验地点** | | 7-219 | |
| 考核标准 | 实验过程  25分 | | 程序运行  20分 | | 回答问题  15分 | | 实验报告  30分 | 特色  功能  5分 | | 考勤违纪情况  5分 | **成绩** |  |
| 成绩栏 |  | |  | |  | |  |  | |  | 其它批改意见:  教师签字： | |
| 考核内容 | 评价在实验课堂中的表现，包括实验态度、编写程序过程等内容等。 | | □功能完善,  □功能不全  □有小错  □无法运行 | | ○正确  ○基本正确  ○有提示  ○无法回答 | | ○完整  ○较完整  ○一般  ○内容极少  ○无报告 | ○有  ○无 | | ○有  ○无 |
|  | | | | | | | | | | | | |
| **一、实验目的**  1、 掌握用C或C++上机调试线性表应用的基本方法；  2、 掌握线性表的基本操作，如插入、删除、查找，以及线性表合并等运算在顺序存储结构和链式存储结构上的运算；并能够运用线性表基本操作解决问题，实现相应算法。  **二、实验题目与要求**  1、已知单链表L（带头节点）是一个递增有序表，试编写一算法，删除表中值大于min且小于max的节点（若表中有这样的节点），同时释放被删节点的空间，这里min和max是两个给定参数。请分析算法时间复杂度。  2、约瑟夫环问题  1）问题描述：有编号为1, 2…n 的 n 个人按顺时针方向围坐一圈，每人持有一个正整数密码。开始给定一个正整数 m，从第一个人按顺时针方向自1开始报数，报到m者出列，不再参加报数，这时将出列者的密码作为m，从出列者顺时针方向的下一人开始重新自1开始报数。如此下去，直到所有人都出列。试设计算法，输出出列者的序列。  2）实验要求：采用顺序和链式两种存储结构实现   1. 一元稀疏多项式简单的计算器   1）问题描述：用线性表表示一元稀疏多项式，设计一个一元多项式运算器  2）实验要求: 采用单链表存储结构一元稀疏多项式，输入并建立多项式，输出多项式，实现多项式加、减运算。   1. **实验过程与实验结果**   **实验1-01：单链表基本操作**  **算法思路简介与描述：**  本实验需要实现以下操作：  1.利用尾插法建立单链表（输入从小到大）  2.给定最大值与最小值，删除单链表内符合要求的节点  3.输出整个单链表  对于链表的操作主要依赖于指针的变换。   1. 尾插法建立单链表在课内已有阐述，即动态分配内存空间->存储数值->建立与前一节点的联系。 2. 输出单链表：利用next指针域对单链表进行遍历的思想，将每次结果进行输出。 3. 本题比较关键的删除最大值最小值之间的节点，则需要运用判断与循环条件，先判断给定的两个值与链表内各节点值的关系，如果超过范围，链表空等情况则以输出“-1”结束，否则进行删除操作。删除操作的思想是对next域内地址的操作，即将两个节点的联系进行变换，断开即将删除节点与前后的联系，然后释放。   **算法实现与测试结果（参考OJ）：**    **实验1-02：约瑟夫环问题**  **算法思路简介与描述：**  本实验需要实现以下操作：   1. 建立单循环链表，其中包括两个数据域（ID与PW）以及next指针域。   建立链表的过程与上一个实验类似，本实验中可以使用不带头节点的单循环链表，我选择了常规带头节点的单循环链表，最后的差异是将需将末尾元素的指针域指向头节点以构成闭环操作。   1. 给定初始值对链表进行初始化，给定初始密码m，删除单链表内符合要求的节点，并迭代实现直到结束。   在本实验中，除了初始化链表以外，我将删除操作分成几个部分，包括1.通过PW与上一轮的ID找到对应出列ID的节点位置，2.删除对应ID的节点，3.按顺序输出删除元素的序号。  本功能直接在删除内实现，并对其加以格式化操作。  本实验难点：格式化操作  **算法实现与测试结果（参考OJ）：**    **实验1-03：一元稀疏多项式简单的计算器实现**  **算法思路简介与描述：**  本实验需要实现以下操作：   1. 利用尾插法建立多项式单链表   与实验1相同，差别于多项式链表的数据区域是两块（指数域与系数域），因此不再赘述。   1. 对两个多项式进行加减运算，结果显示在第一个多项式中   加法的实现思想是通过对两个多项式链表同时进行遍历，存在着以下情形：   1. 指数相同，系数直接相加。 2. 指数不相同，进行比较   1.A>B,先输出B的节点，然后指针移动，再次比较。  2.A<B,先输出A的节点，然后指针移动，再次比较。  直到对两个链表都遍历完毕，此时运算结束，将新多项式的头指针返回。  这里特别注意减法的实现，因为第一个多项式减去第二个多项式存在出现负系数的情况，所以相对于加法只需要加上（-）的情形。   1. 输出两个多项式   运用链表遍历的思想，实现方法与实验1相同。  **算法实现与测试结果（参考OJ）：**     1. **收获与体会**   通过本次实验，我加深了对线性表操作的理解。对于线性表基本的增删改查操作有了更进一步的掌握。本次实验中最大的难点在于对指针的操作以及在OJ平台上调试过程中的格式化输出。本次实验充分利用了TJUT OJ平台，在线操作的同时也能更加精确地发现在编程中存在的问题。因此，在今后的学习过程中，我将通过更多的方式对这些弱点与问题进行强化和修正，在实践中成长。   1. **源代码清单** | | | | | | | | | | | | |
| **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright 1997-2017 Duke.Wang\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **实验1-01：单链表基本操作**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  using namespace std;  **//数据结构定义**  typedef int ElemType;  typedef struct LNode{  ElemType data;  struct LNode \*next;  }LNode;  **//定义头指针**  typedef LNode \*LinkList;  **//定义单链表基本操作**  **//1.尾插法建立单链表**  LinkList CreateList\_tail(LinkList L, int num){  ElemType x;  L=(LinkList)malloc(sizeof(LNode));  LinkList s,r=L;  L->next=NULL;  for(int i=0;i<num;i++){  cin>>x;  s=(LinkList)malloc(sizeof(LNode));  s->data=x;  r->next=s;  r=s;  }  r->next=NULL;  return L;  }  **//2. 删除运算**  LinkList ListDelete\_NMM(LinkList L, int min, int max){  LinkList p,q;  p=L;  //临界条件：头节点next域非空，执行循环查询操作  while(p->next!=NULL){  //判断条件：当前指向数据域在min~max之间  if((p->next->data>min)&&(p->next->data<max)){  //执行操作：切换next域，断开当前元素连接并连接到之后元素  q=p->next;  p->next=q->next;  free(q);  }  else p=p->next;  }  return L;  }  **//3.输出链表**  void ListDisplay(LinkList L){  //判断条件：头节点next域非空，即链表非空  if(L->next!=NULL){  LinkList p=L->next;  //临界条件：数据元素next域非空，即下一节点存在  while(p->next){  cout<<p->data<<" ";  p=p->next;  }  cout<<p->data<<endl;//满足格式化要求，输出最后一个元素  }  else cout<<"-1"<<endl;  }  **//主函数实现**  int main(){  int T;  int n;  int q;  int type;  LinkList L;  cin>>T;  for(int i=0;i<T;i++){  cin>>n;  L=CreateList\_tail(L,n);  cin>>q;  for(int i=0;i<q;i++){  cin>>type;  switch(type){  case 1:{  ListDisplay(L);  break;  }  case 2:{  int min,max;  cin>>min;  cin>>max;  L=ListDelete\_NMM(L,min,max);  break;  }  default:{  break;  }  }  }  }    return 0;  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **//实验1-02：约瑟夫问题**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  using namespace std;  **//定义节点类型**  typedef int ElemType;  typedef struct LNode{  ElemType PW;  int ID;  struct LNode \*next;  }LNode;  **//定义头指针**  typedef LNode \*LinkList;  **//1.尾插法建立单循环链表：OK**  LinkList CreateList\_REC(LinkList L, int num){  ElemType x;  L=(LinkList)malloc(sizeof(LNode));  LinkList p=L,q;  L->next=L;  for(int i=1;i<=num;i++){  cin>>x;  q=(LinkList)malloc(sizeof(LNode));  q->PW=x;  q->ID=i;  p->next=q;  p=q;  p->next=L;  }  return L;  }  **//2.查找给定ID的节点，并返回其指针p：OK**  LinkList Find\_ID(LinkList L, int ID){  LinkList p=L->next;  while(p->ID!=ID){  p=p->next;  }  return p;  }  **//3.根据PW与当前ID查找下一个删除对象，并返回其ID：OK**  int Find\_LNode(LinkList L, int ID, int PW){  LinkList p=L;  int id=ID,id\_max,id\_min;  **//获取当前表最大/最小ID**  while(p->next!=L){  p=p->next;  }  id\_min=L->next->ID;  id\_max=p->ID;  **//被删节点是之前最末节点的情况：将起始id设置为当前最小id**  if(id\_min==id\_max) return id\_min;  if(id>id\_max){  id=id\_min;  }  **//找到起始节点**  p=Find\_ID(L,id);  for(int i=0;i<PW-1;i++){  if(p->next==L)  p=p->next->next;  else p=p->next;  }  id=p->ID;  return id;  }  **//4.查找给定ID的前一个节点，并返回其指针p：OK**  LinkList Find\_PID(LinkList L, int ID){  LinkList p=L;  while(p->next->ID!=ID){  p=p->next;  }  return p;  }  **//3.删除给定ID的对象，并返回其PW**  int Delete\_LNode(LinkList L, int ID){  LinkList p=Find\_PID(L,ID);  LinkList q=p->next;  int id,pw;    id=q->ID;  pw=q->PW;  cout<<id;  p->next=q->next;  free(q);    return pw;  }  **//4.输出**  void Display\_YSF(LinkList L, ElemType start){  LinkList p;  ElemType pw=start;  int id,id\_next=0;  while(L->next!=L){  p=L->next;  if(id\_next==0) id\_next=1;    id=Find\_LNode(L,id\_next,pw);    if(Find\_ID(L,id)->next!=L)  id\_next=Find\_ID(L,id)->next->ID;  else  id\_next=Find\_ID(L,id)->next->next->ID;  if(L->next!=L){  if(p->next!=L){  pw=Delete\_LNode(L,id);  cout<<" ";  }  else pw=Delete\_LNode(L,id);  }  if(L->next==L){  cout<<endl;  break;  }  }  }  int main(){  int T;  int n,m;  cin>>T;  for(int i=0;i<T;i++){  cin>>n;  cin>>m;  LinkList L=CreateList\_REC(L,n);  Display\_YSF(L,m);  }  return 0;  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **//实验1-03：一元多项式计算器**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  using namespace std;  //定义节点类型  typedef struct PolyNode{  int coef; //系数  int exp; //指数  struct PolyNode \*next;  }PolyNode;  typedef PolyNode \*Polynomial;  //1.尾插法建立多项式单链表  Polynomial Create\_Polyn(Polynomial L,int num){  int c,e;  L=(Polynomial)malloc(sizeof(PolyNode));  Polynomial p=L,q;  L->next=L;  for(int i=1;i<=num;i++){  cin>>c>>e;  q=(Polynomial)malloc(sizeof(PolyNode));  q->coef=c;  q->exp=e;  p->next=q;  p=q;  p->next=NULL;  }  return L;  }  //2.显示多项式  void Display\_Polyn(Polynomial L){  Polynomial p=L;  if(p->next!=NULL){  while(p->next!=NULL){  p=p->next;  cout<<p->coef<<"X^"<<p->exp;  if(p->next!=NULL&&p->next->coef>0)  cout<<"+";  }  }  else cout<<"0";  cout<<endl;  }  //3.多项式相加  Polynomial Add\_Poly(Polynomial A,Polynomial B){  Polynomial C,S;  Polynomial Pa,Pb,Pc;  int x;  Pa=A->next;  Pb=B->next;  C=(Polynomial)malloc(sizeof(PolyNode));  Pc=C;  Pc->next=NULL;  while(Pa&&Pb){  if(Pa->exp==Pb->exp){  x=Pa->coef+Pb->coef;  if(x){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=x;  S->exp=Pa->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  }  Pa=Pa->next;  Pb=Pb->next;  }  else if(Pa->exp<Pb->exp){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pa->coef;  S->exp=Pa->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pa=Pa->next;  }  else{  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pb->coef;  S->exp=Pb->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pb=Pb->next;  }  while(Pa){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pa->coef;  S->exp=Pa->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pa=Pa->next;  }  while(Pb){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pb->coef;  S->exp=Pb->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pb=Pb->next;  }  }  return C;  }  Polynomial add\_Poly(Polynomial A, Polynomial B) //多项式相加  {  Polynomial C,S;  Polynomial pa,pb,pc;  int x;  pa = A->next;  pb = B->next;  C = (Polynomial )malloc(sizeof(PolyNode ));  pc = C;  pc->next = NULL;  while (pa && pb)  {  if(pa->exp == pb->exp) // 指数相等时  {  x = pa->coef + pb->coef;  if (x) // 相加完的系数不为0时  {  S = (Polynomial )malloc(sizeof(PolyNode ));  S->coef = x;  S->exp = pa->exp;  S->next = NULL;  pc->next = S;  pc = S;  }  pa = pa->next;  pb = pb->next;  }  else  if(pa->exp < pb->exp)  {  S = (Polynomial )malloc(sizeof(PolyNode ));  S->coef = pa->coef;  S->exp = pa->exp;  S->next = NULL;  pc->next = S;  pc = S;  pa = pa->next;  }  else  {  S = (Polynomial )malloc(sizeof(PolyNode ));  S->coef = pb->coef;  S->exp = pb->exp;  S->next = NULL;  pc->next = S;  pc = S;  pb = pb->next;  }  }  while (pa){  S = (Polynomial )malloc(sizeof(PolyNode ));  S->coef = pa->coef;  S->exp = pa->exp;  S->next = NULL;  pc->next = S;  pc = S;  pa = pa->next;  }  while (pb)  {  S = (Polynomial )malloc(sizeof(PolyNode ));  S->coef = pb->coef;  S->exp = pb->exp;  S->next = NULL;  pc->next = S;  pc = S;  pb = pb->next;  }  return C;  }  //3.多项式相减  Polynomial Sub\_Poly(Polynomial A,Polynomial B){  Polynomial C,S;  Polynomial Pa,Pb,Pc;  int x;  Pa=A->next;  Pb=B->next;  C=(Polynomial)malloc(sizeof(PolyNode));  Pc=C;  Pc->next=NULL;  while(Pa&&Pb){  if(Pa->exp==Pb->exp){  x=Pa->coef-Pb->coef;  if(x!=0){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=x;  S->exp=Pa->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  }  Pa=Pa->next;  Pb=Pb->next;  }  else if(Pa->exp<Pb->exp){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pa->coef;  S->exp=Pa->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pa=Pa->next;  }  else{  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pb->coef;  S->exp=Pb->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pb=Pb->next;  }  while(Pa){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=Pa->coef;  S->exp=Pa->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pa=Pa->next;  }  while(Pb){  S=(Polynomial)malloc(sizeof(PolyNode));  S->coef=-(Pb->coef);  S->exp=Pb->exp;  S->next=NULL;  Pc->next=S;  Pc=S;  Pb=Pb->next;  }  }  return C;  }  int main(){  int T;  cin>>T;  for(int i=0;i<T;i++){  int num\_A,num\_B;  cin>>num\_A;  cin>>num\_B;  Polynomial A=Create\_Polyn(A,num\_A);  Polynomial B=Create\_Polyn(B,num\_B);  int q;  cin>>q;  for(int i=0;i<q;i++){  int type;  cin>>type;  switch(type){  case 1:{  Display\_Polyn(A);  Display\_Polyn(B);  break;  }  case 2:{  A=add\_Poly(A,B);  break;  }  case 3:{  A=Sub\_Poly(A,B);  break;  }  default:{  break;  }  }  }  }  return 0;  } | | | | | | | | | | | | |