

**实 验 报 告**

**（2019 / 2020 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 数据结构 | | | | | |
| 实验名称 | 线性表及多项式的运算 | | | | | |
| 实验时间 | 2019 | 年 | 10 | 月 | 11 | 日 |
| 指导单位 | 计算机学院 | | | | | |
| 指导教师 | 邹志强 | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 | 张颖 | 班级学号 | B18030406 |
| 学院(系) | 计算机学院 | 专 业 | 计算机科学与技术 |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | **线性表及多项式的运算** | | | **指导教师** | **邹志强** |
| **实验类型** | **验证** | **实验学时** | **2** | **实验时间** | **2019-8-31** |
| 1. **实验目的和要求**   1.掌握线性表的两种基本存储结构及其应用场合：顺序存储和链接存储。  2.掌握顺序表和链表的各种基本操作算法。  3.理解线性表应用于多项式的实现算法。 | | | | | |
| 二、**实验环境(实验设备)**  硬件：微型计算机  软件：Windows 操作系统、Dev-C++ 5.11 | | | | | |
| **三、实验原理及内容**  1、算法设计  ADT List{  数据：  零个或多个数据元素构成的线性序列()。数据元素之间的关系是一对一关系。  运算：  Init(L)：始化运算。构造一个空的线性表L，若初始化成功，则返回OK，否则返回ERROR。  Find(L, i, x)：查找运算。若线性表L已存在且0≤i≤n-1，则查找线性表L中元素的值并通过x返回；否则，返回ERROR。  Insert(L, i, x)：插入运算。若线性表L已存在且-1≤i≤n-1，则在元素之后插入新元素x，插入成功后返回OK，否则返回ERROR。  Delete(L, i)：删除运算。若线性表L非空且0≤i≤n-1，则删除元素，删除成功后返回OK，否则返回ERROR。  Output(L)：输出运算。若线性表L已存在，则输出线性表L中所有数据元素，否则返回ERROR。  Destroy(L)：撤销运算。判断线性表L是否存在，若已存在，则撤销线性表L；否则，返回ERROR。  }  2、算法实现与分析  1）参照程序2.1~程序2.7，编写程序，完成顺序表的初始化、查找、插入、删除、输出、撤销等操作。  #include <stdio.h>  #include <stdlib.h>  #define ERROR 0  #define SUCCESS 1  typedef int ElemType;  typedef struct {  int n;  ElemType \*ele;  int maxLength;  }SeqList;  int Init( SeqList \*L, int maxSize);  int Find( SeqList L, int i, ElemType \*x );  int Insert( SeqList \*L, int i, ElemType x );  int Delete( SeqList \*L, int i );  int Output( SeqList L );  int Destory( SeqList \*L);  int main(){  SeqList list;  int maxLength, element, delIndex, elementLength, findIndex, findElement;  printf("请输入你要创建的顺序表的最大长度：");  scanf("%d", &maxLength);  Init(&list, maxLength);  printf("请输入你要输入的元素个数：");  scanf("%d",&elementLength);  printf("请依次输入你要输入的元素：");  for(int i = 0;i < elementLength; i++){  scanf("%d", &element);  Insert( &list, i - 1, element );  }  printf("\n顺序表创建成功！现在的顺序表为：");  Output(list);  printf("\n你要查找的是第几个元素？请输入它的下标：");  scanf("%d", &findIndex);  Find(list,findIndex, &findElement);  printf("\n查找成功！你查找的元素为：%d", findElement);  printf("\n你要删除的是第几个元素？请输入它的下标：");  scanf("%d", &delIndex);  Delete(&list,delIndex);  printf("\n删除成功！现在的顺序表为：");  Output(list);  Destory(&list);  return SUCCESS;  }  int Init( SeqList \*L, int maxSize){  L -> maxLength = maxSize;  L -> n = 0;  L -> ele = ( ElemType\* ) malloc ( sizeof(ElemType\*) \* maxSize );  if(!L -> ele)  return ERROR;  return SUCCESS;  }  int Find( SeqList L, int i, ElemType \*x ){  if( i > L.n -1 || i < 0 )  return ERROR;  \*x = L.ele[i];  return SUCCESS;  }  int Insert( SeqList \*L, int i, ElemType x ){  if( i > L -> n -1 || i < -1 || L -> n == L -> maxLength)  return ERROR;  for( int j = L -> n - 1; j > i; j--)  L -> ele[j + 1] = L -> ele[j];  L -> ele[i + 1] = x;  L -> n = L -> n + 1;  return SUCCESS;  }  int Delete( SeqList \*L, int i ){  if( i > L -> n - 1 || i < 0)  return ERROR;  for( int j = i + 1; j < L -> n ; j++)  L -> ele[j - 1] = L -> ele[j];  L -> n --;  return SUCCESS;  }  int Output( SeqList L ){  if(!L.n)  return ERROR;  for( int i = 0; i < L.n ; i++)  printf("%d ", L.ele[i]);  return SUCCESS;  }  int Destory( SeqList \*L){  (\*L).n = 0;  (\*L).maxLength = 0;  free( (\*L).ele );  }  时间复杂度为O(n)。  2）参照程序2.8~程序2.14，编写程序，完成带表头节点单链表的初始化、查找、插入、删除、输出、撤销等操作。  #include <stdio.h>  #include <stdlib.h>  #define ERROR 0  #define SUCCESS 1  typedef int ElemType;  typedef struct node{  ElemType element;  struct node \*link;  }node;  typedef struct{  struct node \*head;  int n;  }headerList;  int Init(headerList \*L);  int Find(headerList L, int i, ElemType \*x);  int Insert(headerList \*L, int i, ElemType x);  int Delete(headerList \*L, int i);  int Output(headerList L);  int Destory(headerList \*L);  int main(){  headerList list;  int element, delIndex, elementLength, findIndex, findElement;  Init(&list);  printf("请输入你要输入的元素个数：");  scanf("%d",&elementLength);  printf("请依次输入你要输入的元素：\n");  for(int i = 0;i < elementLength; i++){  scanf("%d", &element);  Insert( &list, i - 1, element );  }  printf("\n 带有表头的单链表创建成功！现在的有表头的单链表为：");  Output(list);  printf("\n 你要查找的是第几个元素？请输入它的下标：");  scanf("%d", &findIndex);  Find(list,findIndex, &findElement);  printf("\n 查找成功！你查找的元素为：%d", findElement);  printf("\n 你要删除的是第几个元素？请输入它的下标：");  scanf("%d", &delIndex);  Delete(&list,delIndex);  printf("\n 删除成功！现在的有表头的单链表为：");  Output(list);  Destory(&list);  return SUCCESS;  }  int Init(headerList \*L){  L -> head = (node\*)malloc(sizeof(node));  L -> head -> link = NULL;  L -> n = 0;  return SUCCESS;  }  int Find(headerList L, int i, ElemType \*x){  node \*p = L.head -> link;  if( i > L.n -1 || i < 0 )  return ERROR;  for(int j = 0; j < i; j++)  p = p -> link;  \*x = p -> element;  return \*x;  }  int Insert(headerList \*L, int i, ElemType x){  node \*p = L -> head, \*q = (node\*)malloc(sizeof(node));  if( i > L -> n -1 || i < -1 )  return ERROR;  for(int j = 0; j <= i; j++)  p = p -> link;  q -> link = p -> link;  p -> link = q;  q -> element = x;  L -> n++;  return SUCCESS;  }  int Delete(headerList \*L, int i){  node \*p = L -> head, \*q;  if( i > L -> n -1 || i < 0 )  return ERROR;  for(int j = 0; j < i; j++)  p = p -> link;  q = p -> link;  p -> link = q -> link;  free(q);  L -> n--;  return SUCCESS;  }  int Output(headerList L){  node \*p = L.head -> link;  if(!L.n)  return ERROR;  while(p){  printf("%d ",p -> element);  p = p -> link;  }  return SUCCESS;  }  int Destory(headerList \*L){  node \*p = L -> head;  while(L -> head){  p = L -> head -> link;  free(L -> head);  L -> head = p;  }  }  时间复杂度为O(n)。  3）以第2题所示带表头结点单链表为存储结构，编写程序实现单链表的逆置操作。（原单链表为（），逆置后为（），要求不引入新的存储空间。）  int Reverse(headerList \*L)  {  node \*pre, \*cur, \*rear;  pre = L -> head -> link;  cur = L -> head -> link -> link;  while (cur)  {  rear = cur->link;  cur->link = pre;  pre = cur;  cur = rear;  }  L -> head -> link -> link = NULL;  L -> head -> link = pre;  }  时间复杂度为O()。  4）以第2题所示带表头结点链表为存储结构，编写程序实现将单链表排序成为有序单链表的操作。  void Sort(headerList \*h){  node \*s1,\*s2,\*s3,\*s4,\*p,\*q;  for (p=h->head;p!=NULL && p->link!=NULL;p=p->link) {  for (q=p->link;q!=NULL && q->link!=NULL;q=q->link) {  if (p->link->element > q->link->element) {  s1=p->link;  s2=p->link->link;  s3=q->link;  s4=q->link->link;    if (s2!=s3) {  p->link=s3;  s3->link=s2;  q->link=s1;  s1->link=s4;  }  else {  p->link=s3;  s3->link=s1;  q=s3;  s1->link=s4;  }  }  }  }  }  时间复杂度为O()。  5）编写程序实现一元多项式的创建、输出、撤销以及两个一元多项式相加和相乘的操作。  #include<stdio.h>  #include<stdlib.h>  typedef struct PNode{  int coef; //系数  int exp; //指数  struct PNode\* link;  }PNode;  typedef struct{  struct PNode \*head;  }polynominal;  void Create(polynominal \*p){  PNode \*pn,\*pre,\*q;  p->head = (PNode\*)malloc(sizeof(PNode));  p->head->exp = -1;  p->head->link = p->head;  for(;;){  pn=(PNode\*)malloc(sizeof(PNode));  printf("coef:\n");  scanf("%d",&pn->coef);  printf("exp:\n");  scanf("%d",&pn->exp);  if(pn->exp<0) break;  pre = p->head;  q = p->head->link;  while(q && q->exp > pn->exp){  pre = q;  q = q->link;  }  pn->link = q;  pre->link = pn;  }  }  void Output(polynominal p){  PNode \*q;  int flag = 1;  q = p.head->link;  if (!q){  return;  }  while(q != p.head){  if (!flag && (q->coef > 0)) printf("+");  flag = 0;  if(q->coef == 0){  return;  }  printf("%d",q->coef);  switch(q->exp){  case 0:break;  case 1:printf("X");break;  default:printf("X^%d",q->exp);break;  }  q = q->link;  }  }  void Add(polynominal \*px,polynominal \*qx){  PNode \*q,\*q1 = qx->head,\*p, \*p1,\*temp;  p = px->head->link;  p1 = px->head;  q = q1->link;  while(p->exp >= 0){  while(p->exp < q->exp){  q1 = q;  q = q->link;  }  if(p->exp == q->exp){  q->coef = q->coef + p->coef;  if(q->coef == 0){  q1->link = q->link;  free(q);  q = q1->link;  p = p->link;  }  else{  q1 = q;  q = q->link;  p = p->link;  }  }  else{  temp = (PNode\*)malloc(sizeof(PNode));  temp->coef = p->coef;  temp->exp = p->exp;  temp->link = q1->link;  q1->link = temp;  q1 = q1->link;  p = p->link;  }  }  }  void Multiply(polynominal \*px,polynominal \*qx){  polynominal qx1,qx2;  PNode \*q1,\*q2,\*q3,\*q4,\*pre,\*q;  qx1.head = (PNode\*)malloc(sizeof(PNode));  qx1.head->exp = -1;  qx1.head->link = qx1.head;  q1 = px->head->link;  q2 = qx->head->link;  while(q2->exp != -1){  q3 = (PNode\*)malloc(sizeof(PNode));  q3->coef = q1->coef \* q2->coef;  q3->exp = q1->exp + q2->exp;  if(qx1.head->link->exp == -1){  q3->link = qx1.head->link;  qx1.head->link = q3;  pre = qx1.head->link;  }  else{  q3->link = qx1.head;  pre->link = q3;  pre = pre->link;  }  q2 = q2->link;  }  q1 = q1->link;  while(q1->exp != -1){  q2 = q2->link;  qx2.head = (PNode\*)malloc(sizeof(PNode));  qx2.head->exp = -1;  qx2.head->link = qx2.head;  while(q2->exp != -1){  q4 = (PNode\*)malloc(sizeof(PNode));  q4->coef = q1->coef \* q2->coef;  q4->exp = q1->exp + q2->exp;  if(qx2.head->link->exp == -1){  q4->link = qx2.head->link;  qx2.head->link = q4;  pre = qx2.head->link;  }  else{  q4->link = qx2.head;  pre->link = q4;  pre = pre->link;  }  q2 = q2->link;  }  Add(&qx2,&qx1);  q1 = q1->link;  }  Output(qx1);  }  void Destroy(polynominal \*p){  PNode \*pn;  while(p->head->link)  {  pn = p->head->link;  free(p->head);  p->head = pn;  }  }  int main(){  polynominal p,q;  int x;  printf("请输入第一个多项式为：\n");  Create(&p);  printf("\n\n请输入第二个多项式为\n");  Create(&q);    printf("\n现在第一个一元多项式为：");  Output(p);  printf("\n现在第二个一元多项式为：");  Output(q);    printf("\n\n请选择你要进行的操作：（0为加法，1为乘法）\n");  scanf("%d",&x);  switch(x){  case 0:printf("Add:\n");  Add(&p,&q);  Output(q);  break;  case 1:printf("Multiply:\n");  Multiply(&p,&q);  default:break;  }  return 0;  }  加法的时间复杂度为O(m+n)，乘法的时间复杂度为O(mn)。  3、实验结果与结论  1）    2）    3）    4）    5） | | | | | |

**实 验 报 告**

|  |
| --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  (一)实验中遇到的主要问题及解决方法  问题：在题目五中加乘法都将结果存在了第二个单链表中，导致计算加乘法只能二选一  解决办法：想到再建一个链表去存，可老是有问题，最后采取运行两次程序的方法解决了  （二）实验心得  多利用线性表进行对数据的增删改查操作，降低时间复杂度，对较大且乱序的数据集合是很好的方法。  （三）意见与建议（没有可省略） |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. **支撑毕业要求指标点**   《数据结构》课程支撑毕业要求的指标点为：  1.2-M掌握计算机软硬件相关工程基础知识，能将其用于分析计算机及应用领域的相关工程问题。  3.2-H能够根据用户需求，选取适当的研究方法和技术手段，确定复杂工程问题的解决方案。  4.2-H能够根据实验方案，配置实验环境、开展实验，使用定性或定量分析方法进行数据分析与处理，综合实验结果以获得合理有效的结论。   |  |  |  |  | | --- | --- | --- | --- | | 实验内容 | 支撑点1.2 | 支撑点3.2 | 支撑点4.2 | | 线性表及多项式的运算 | √ |  |  | | 二叉树的基本操作及哈夫曼编码译码系统的实现 |  | √ | √ | | 图的基本运算及智能交通中的最佳路径选择问题 |  | √ | √ | | 各种内排序算法的实现及性能比较 | √ |  | √ | | | | | | |
| **六、指导教师评语 (含学生能力达成度的评价)**  如评分细则所示 | | | | | |
| **成 绩** | XX | **批阅人** | **XX** | **日 期** | **XXX** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **评 分 细 则** | **评分项** | **优秀** | **良好** | **中等** | **合格** | **不合格** |
| **遵守实验室规章制度** |  |  |  |  |  |
| **学习态度** |  |  |  |  |  |
| **算法思想准备情况** |  |  |  |  |  |
| **程序设计能力** |  |  |  |  |  |
| **解决问题能力** |  |  |  |  |  |
| **课题功能实现情况** |  |  |  |  |  |
| **算法设计合理性** |  |  |  |  |  |
| **算法效能评价** |  |  |  |  |  |
| **回答问题准确度** |  |  |  |  |  |
| **报告书写认真程度** |  |  |  |  |  |
| **内容详实程度** |  |  |  |  |  |
| **文字表达熟练程度** |  |  |  |  |  |
| **其它评价意见** |  | | | | |
| **本次实验能力达成评价（总成绩）** |  |  |  |  |  |