

**实 验 报 告**

**（2022 / 2023 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 数据结构 | | | | | |
| 实验名称 | 线性表的基本运算及多项式的算术运算 | | | | | |
| 实验时间 | 2022 | 年 | 9 | 月 | 29 | 日 |
| 指导单位 | 计算机学院 | | | | | |
| 指导教师 | 陈小柏 | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 |  | 班级学号 |  |
| 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** | 线性表的基本运算及多项式的算术运算 | | | **指导教师** | 陈小柏 |
| **实验类型** | **设计** | **实验学时** | **2** | **实验时间** | 2022年9月29日 |
| 1. **实验目的和要求**   描述本次实验的目的和要求，可参考教材第12章相关描述。   * + - 1. 掌握线性表的顺序存储和链式存储这两种基本储存结构及其应用场合。       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、Microsoft Visual C++6.0 | | | | | |
| **三、实验原理及内容**  中文五号宋体，英文五号Times new roman字体，1.25倍行距  1、算法设计  typedef struct seqList  {  int n; // 顺序表长度  int maxLength; // 最大长度  int\* element; // 指向顺序表存储空间的首地址  }seqList;  typedef struct node  {  int element; // 数据域  struct node\* link; // 指针域  }Node;  typedef struct hearderList  {  Node\* head; // 头指针  int n; // 元素个数  }hearderList;  2、算法实现与分析  顺序表： #include<stdio.h>  #include<stdlib.h>  #define ERROR 0  #define OK 1  typedef struct seqList  {  int n; // 顺序表长度  int maxLength; // 最大长度  int\* element; // 指向顺序表存储空间的首地址  }seqList;  int Init(seqList\* L, int mSize) // 初始化  {  L->maxLength = mSize;  L->n = 0;  L->element = (int\*)malloc(sizeof(int) \* mSize); // 动态生成一维数组空间  if (!L->element)  return ERROR;  return OK;  }  int Find(seqList\* L, int i, int\* x) // 查找  {  if (i<0 || i>L->n - 1) // 判断下标越界  {  return ERROR;  }  \*x = L->element[i]; // x返回查找的值  return OK;  }  int Insert(seqList\* L, int i, int x) // 插入  {  int j;  if (i<-1 || i>L->n - 1) // 判断是否为有效索引  {  return ERROR;  }  if (L->n == L->maxLength) // 判断是否存满  {  return ERROR;  }  for ( j = L->n-1; j > i; j--) // 按顺序向后移  {  L->element[j + 1] = L->element[j];  }  L->element[i + 1] = x;  L->n += 1;  return OK;  }  int Delete(seqList\* L, int i) // 元素的删除  {  if (i<0 || i>L->n - 1) // 判断是否为有效索引  {  return ERROR;  }  if (!L->n) // 判断是否为空  return ERROR;  for (int j = i + 1; j < L->n; j++)  {  L->element[j - 1] = L->element[j]; // 从前往后逐个前移元素  }  L->n--;  return OK;  }  int Output(seqList\* L) // 链表的输出  {  int i;  if (L->n == 0) // 判断是否为空  return ERROR;  for ( i = 0; i < L->n; i++)  {  printf("%d ", L->element[i]); // 从前往后逐个输出元素  }  printf("\n");  return OK;  }  void Destory(seqList\* L) // 链表的撤销  {  L->n = 0;  L->maxLength = 0;  free(L->element);  }  // 测试用主函数  void main()  {  int i;  seqList list;  Init(&list, 10);  for (i = 0; i < 10; i++)  Insert(&list, i - 1, i);  Output(&list);  Delete(&list, 0);  Output(&list);  Destory(&list);  }  链表:  #include<stdio.h>  #include<stdlib.h>  #define ERROR 0;  #define OK 1;  typedef struct node  {  int element; // 数据域  struct node\* link; // 指针域  }Node;  typedef struct hearderList  {  Node\* head; // 头指针  int n; // 元素个数  }hearderList;  int Init(hearderList\* L) // 初始化  {  L->head = (Node\*)malloc(sizeof(Node)); // 生成表头结点  if (!L->head)  return ERROR;  L->head->link= NULL;  L->n = 0;  return OK;  }  int Find(hearderList L, int i, int\* x) //查找  {  Node\* p;  int j;  if (i<0||i>L.n-1) // 是否为合法位置  {  return ERROR;  }  p = L.head;  for ( j = 0; j < i; j++) //定位查找的元素  {  p = p->link;  }  \*x = p->element; // 赋值  return OK;  }  int Insert(hearderList\* L, int i, int x) // 插入  {  Node\* p, \* q;  int j;  if (i<-1 || i>L->n - 1) //判断合法位置  return ERROR;  p = L->head;  for ( j = 0; j <= i; j++)  {  p = p->link;  }  q = (Node\*)malloc(sizeof(Node)); //申请空间  q->element = x;  q->link = p->link;  p->link = q;  L->n++;  return OK;  }  int Delete(hearderList\* L, int i) // 删除  {  int j;  Node\* p, \* q;  if (!L->n) // 判断为空  return ERROR;  if (i<0||i>L->n-1) // 判断合法  {  return ERROR;  }  q = L->head;  for ( j = 0; j < i-1; j++) // 定位删除的前一位  {  q = q->link;  }  p = q->link;  q->link = p->link;  free(p);  L->n--;  return OK;  }  int Output(hearderList\* L) // 输出链表  {  Node\* p;  if (!L->n) // 判断为空  return ERROR;  p = L->head->link;  while (p)  {  printf("%d ", p->element);  p = p->link;  }  return OK;  }  void Destory(hearderList\* L) // 撤销链表  {  Node\* p;  while (L->head) // 循环撤销链表  {  p = L->head->link; // 保存后继结点地址  free(L->head);  L->head = p;  }  }  int Revert(hearderList\* L) // 逆置  {  if (!L->n) // 判断为空  return ERROR;  Node\* pre, \*cur, \*temp; // 设置三个记录点  pre= L->head->link; // 初始化三个记录点  cur = pre->link;  temp = cur->link;  pre->link = NULL;  while (temp) // 当遍历一遍后完成逆置  {  cur->link = pre; // 逆置前两个记录点  pre = cur; // 继续后移  cur = temp;  temp = temp->link;  }  cur->link = pre;  L->head->link = cur; // 重置头节点  return OK;  }  int Sort(hearderList\* L) // 排序链表，此处采用冒泡排序  {  if (!L->n) // 判断为空  return ERROR;  int temp; // 交换用的temp  int judgeEnd = 1; // 判断是否完成排序    while (judgeEnd) // 如果在一次遍历中没有发生交换，则judgeEnd=0，结束遍历，完成排序  {  Node\* p = L->head->link;  Node\* q = p->link;  judgeEnd = 0; // 开始遍历时设为0  while (q)  {  if (p->element > q->element) // 冒泡排序  {  temp = p->element;  p->element = q->element;  q->element = temp;  judgeEnd = 1; // 如果在一次遍历中发生了交换，说明没有完成排序，judgeEnd=1继续遍历  }  p = p->link;  q = q->link;  }  }  return OK;  }  // 测试用主函数  void main()  {  int i;  int x;  hearderList list;  Init(&list);  for ( i = 0; i < 9; i++)  {  Insert(&list, i-1, i);  }  printf("the linklist is:");  Output(&list);  printf("\nthe revert linklist is:");  Revert(&list);  Output(&list);  printf("\nthe sorted linklist is:");  Sort(&list);  Output(&list);  Delete(&list, 0);  printf("\nthe linklist is:");  Output(&list);  Find(list, 1, &x);  printf("\nthe value is:");  printf("%d", x);  Destory(&list);  }  多项式：  #include<stdio.h>  #include<stdlib.h>  typedef struct pNode // 多项式项  {  int coef; // 系数  int exp; // 指数  struct pNode\* link;  }PNode;  typedef struct polynominal // 多项式  {  PNode\* head; // 指向第一项  }Polynominal;  void Create(Polynominal\* p)  {  PNode\* pn, \* pre, \* q;  p->head = (PNode\*)malloc(sizeof(PNode)); // 初始化表头  p->head->exp = -1;  p->head->link = p->head;  while(1) // 创建链表  {  pn = (PNode\*)malloc(sizeof(PNode)); // 申请空间  printf("coef:\n");  scanf\_s("%d", &pn->coef); // 输入系数  printf("exp:\n");  scanf\_s("%d", &pn->exp); // 输入指数  if (pn->exp < 0) // 当指数为0时停止输入  break;  pre = p->head;  q = p->head->link;  while (q&&q->exp>pn->exp) // 按照降幂排列  {  pre = q;  q = q->link;  }  pn->link = q; // 插入结点  pre->link = pn;  }  }  void Output(Polynominal\* p) // 输出多项式  {  PNode\* temp = p->head->link;  while (temp->exp>=0)  {  printf("%dx^%d", temp->coef, temp->exp); // 循环输出  printf("+");  temp = temp->link;  }  // 控制格式  printf("\b ");  printf("\n");  }  void Add(Polynominal\* px, Polynominal\* qx) // 多项式相加  {  PNode\* q, \* q1 = qx->head, \* p, \* p1, \* temp; // q1指向表头结点  p = px->head->link; // p指向多项式px的第一个结点  q = q1->link; // q1是q的前驱  while (p->exp>=0)  {  while (p->exp < q->exp) // 跳过q->exp大的项  {  q1 = q;  q = q->link;  }  if (p->exp==q->exp) // 当指数相等时，系数相加  {  q->coef = q->coef + p->coef;  if (q->coef == 0) // 若相加后系数为0  {  q1->link = q->link; // 删除q  free(p); // 释放空间  q = q1->link; // 重置指针  p = p->link;  }  else // 若相加后不为0  {  q1 = q; // q1后移  q = q->link;  p = p->link;  }  }  else // p->epx > q->exp 的情况  {  temp = (PNode\*)malloc(sizeof(PNode)); // 生成新结点  temp->coef = p->coef;  temp->exp = p->exp;  temp->link = q1->link;  q1->link = temp;  q1 = q1->link;  p = p->link;  }  }  }  Polynominal\* Time(Polynominal\* px, Polynominal\* qx) // 多项式乘法  {  PNode\* p = px->head->link, \* q = qx->head->link; // 初始化参数指向两个多项式  PNode\* pn, \* pre, \* cur;  Polynominal\* result = (Polynominal\*)malloc(sizeof(Polynominal)); // 新建一个多项式链表来储存结果  result->head= (PNode\*)malloc(sizeof(PNode)); // 初始化新链表  result->head->exp = -1;  result->head->link = result->head;  while (p->exp>=0) // 遍历第一个多项式  {  while (q->exp>=0) // 遍历第二个多项式  {  pn = (PNode\*)malloc(sizeof(PNode)); // 申请新的结点  pn->coef = p->coef \* q->coef; // 新结点的系数为遍历中的两个多项式的项的系数之和  pn->exp = p->exp + q->exp; // 同理计算指数  // 按降幂插入链表  pre = result->head;  cur = result->head->link;  while (cur && cur->exp > pn->exp)  {  pre = cur;  cur = cur->link;  }  pn->link = cur;  pre->link = pn;  // 继续遍历  q = q->link;  }  q = qx->head->link;  p = p->link;  }  return result; // 返回结果  }  void main()  {  Polynominal pn\_1;  Polynominal pn\_2;  Create(&pn\_1);  Output(&pn\_1);  Create(&pn\_2);  Output(&pn\_2);  Output(Time(&pn\_1, &pn\_2));  }  3、实验结果与结论  顺序表：    链表：    多项式： | | | | | |

**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）  中文五号宋体，英文五号Times new roman字体，1.25倍行距  **说明：这部分内容主要包括：在编程、调试或测试过程中遇到的问题及解决方法、本次实验的心得体会、进一步改进的设想等。**  (一)实验中遇到的主要问题及解决方法  在实验过程我遇到了出现死循环的问题，经过排查发现，发现问题在于没有及时将指针移到下一个结点。  （二）实验心得  本次实验我掌握了线性表的顺序存储和链式存储这两种基本存储结构及其应用场合，掌握了顺序表和链表的各种基本操作算法，理解了线性表应用于多项式的实现算法。  （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-L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。  4.1-L能基于专业理论，针对计算机及应用领域复杂工程问题设计可行的实验方案。   |  |  |  |  | | --- | --- | --- | --- | | 实验内容 | 支撑点1.2 | 支撑点3.2 | 支撑点4.1 | | 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** |  | **批阅人** |  | **日 期** |  |