

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 数据结构 | | | | |
| 实验编号： | 实验五 | | | | |
| 实验名称： | 实系数一元多项式问题 | | | | |
| 实验人员： | 学号 | 18111303044 | | | |
| 姓名 | 邵一波 | | | |
| 班级 | 2018级计算机类一班 | | | |
| 实验日期： | 2019-10-24 | | | | |
| 实验室： | 六号楼2060302 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

|  |
| --- |
| **注：具体内容可根据专业特点和实验性质略作调整，页面不够可附页。**   1. **需求分析**   任务目的：实现实系数一元多项式的创建、打印、以及两个一元多项式的加、减、乘运算。   1. 输入的形式和输入值的范围   依次输入多项式每一项的系数和指数，其中系数为float型，指数为int形。  以 0 0作为结束符号。   1. 输出的形式   输出为多项式     1. 程序所能达到的功能   实现实系数一元多项式的创建、打印、以及两个一元多项式的加、减、乘运算。   1. 测试数据   2 3  -1 2  3 0  1 2  0 0  -2 1  -3 3  -1 -1  0 0   1. **概要设计** 2. 抽象数据类型的定义   **CommonDef.h**  typedef struct node  {  float coef; //系数  int exp; //指数  struct node \*next; //指向下一节点  }polynode;  typedef polynode \* LinkList;   1. 函数的定义   **polynDef.h**  //多项式创建  LinkList CreatPolyn();  //多项式的打印  void PrintPolyn(LinkList p);  //两个多项式的相加  LinkList AddPolyn(LinkList p1, LinkList p2);  //两个多项式的相减  LinkList SubstractPolyn(LinkList p1, LinkList p2);  //两个多项式的相乘  LinkList MultiplyPolyn(LinkList p1, LinkList p2);  //多项式的销毁  int DestroyPolyn(LinkList \* p);  //检查并销毁系数为0的节点  void check(LinkList p);   1. 主程序的流程图      1. **详细设计**   **polynTestApp.c**  #include "polynDef.h"  int main()  {  LinkList p1, p2, p3, p4, p5;  printf("输入多项式1（系数 指数）:\n");  p1=CreatPolyn();  printf("输入多项式2（系数 指数）:\n");  p2=CreatPolyn();  printf("\n多项式1:\n");  PrintPolyn(p1);  printf("\n多项式2:\n");  PrintPolyn(p2);  printf("\n多项式1 + 多项式2= \n");  p3=AddPolyn(p1, p2);  PrintPolyn(p3);  printf("\n多项式1 - 多项式2= \n");  p4=SubstractPolyn(p1, p2);  PrintPolyn(p4);  printf("\n多项式1 \* 多项式2= \n");  p5=MultiplyPolyn(p1, p2);  PrintPolyn(p5);  if(DestroyPolyn(&p1)  && DestroyPolyn(&p2)  && DestroyPolyn(&p3)  && DestroyPolyn(&p4)  && DestroyPolyn(&p5)  )  printf("销毁成功！\n");  return 0;  }  **polynApp.c**  #include "polynDef.h"  //多项式创建  LinkList CreatPolyn()  {  LinkList p,q,r;  float coef;  int exp;  //建立头结点  p = (LinkList)malloc(sizeof(struct node));  p->coef = 0;  p->exp = -1;  p->next = NULL;    //从键盘获取值  scanf("%f",&coef);  scanf("%d",&exp);  while(coef != 0 || exp !=0 )//结束标志为 0 0  {  q = p;  while(q->next && q->next->exp < exp)  q = q->next;    if(q->next && q->next->exp == exp)  q->next->coef += coef;  else  {  r = (LinkList)malloc(sizeof(struct node));  r->coef = coef;  r->exp = exp;  r->next = q->next;  q->next = r;  }  //再次取值  scanf("%f",&coef);  scanf("%d",&exp);  }  return p;  }  //多项式的打印  void PrintPolyn(LinkList p)  {  p = p->next;  if(p)  {  while(!p->coef && p)  p = p->next;  if(p->exp != 0)  {  if(p->coef != 1 && p->coef != -1)  printf("%.1f",p->coef);  else if(p->coef == -1)  printf("-");  if(p->exp == 1)  printf("x",p->exp);  else  printf("x^%d",p->exp);  }  else  printf("%.1f",p->coef);  p = p->next;  }  while(p)  {  if(p->coef > 0)  {  if(p->coef == 1 && p->exp != 0)  printf("+");  else  printf("+%.1f",p->coef);  }  if(p->coef < 0)  {  if(p->coef == -1 && p->exp != 0)  printf("-");  else  printf("%.1f",p->coef);  }  if(p->coef != 0 && p->exp != 0)  {  if(p->exp == 1)  printf("x",p->exp);  else  printf("x^%d",p->exp);  }  p = p->next;  }  printf("\n");  }  //两个多项式的相加  LinkList AddPolyn(LinkList p1, LinkList p2)  {  float coef;  LinkList ps,q,rear;  //建立头结点  ps = (LinkList)malloc(sizeof(struct node));  ps->coef = 0;  ps->exp = -1;  ps->next = NULL;  rear = ps;  p1 = p1->next;  p2 = p2->next;  while(p1 && p2)  {  if (p1->exp == p2->exp)  {  if(coef = p1->coef + p2->coef)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = coef;  q->exp = p1->exp;  q->next = NULL;  rear->next = q;  rear = q;  }  p1 = p1->next;  p2 = p2->next;  }  else if (p1->exp > p2->exp)  {  if(p2->coef)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p2->coef;  q->exp = p2->exp;  q->next = NULL;  rear->next = q;  rear = q;  }  p2 = p2->next;  }  else  {  if(p1->coef)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p1->coef;  q->exp = p1->exp;  q->next = NULL;  rear->next = q;  rear = q;  }  p1 = p1->next;  }  }  while (p1)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p1->coef;  q->exp = p1->exp;  q->next = NULL;  rear->next = q;  rear = q;  p1 = p1->next;  }  while (p2)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p2->coef;  q->exp = p2->exp;  q->next = NULL;  rear->next = q;  rear = q;  p2 = p2->next;  }  check(ps);  return ps;  }  //两个多项式的相减  LinkList SubstractPolyn(LinkList p1, LinkList p2)  {  float coef;  LinkList ps,q,rear;  //建立头结点  ps = (LinkList)malloc(sizeof(struct node));  ps->coef = 0;  ps->exp = -1;  ps->next = NULL;  rear = ps;  p1 = p1->next;  p2 = p2->next;  while(p1 && p2)  {  if (p1->exp == p2->exp)  {  if(coef = p1->coef - p2->coef)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = coef;  q->exp = p1->exp;  q->next = NULL;  rear->next = q;  rear = q;  }  p1 = p1->next;  p2 = p2->next;  }  else if (p1->exp > p2->exp)  {  if(p2->coef)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p2->coef\*-1;  q->exp = p2->exp;  q->next = NULL;  rear->next = q;  rear = q;  }  p2 = p2->next;  }  else  {  if(p1->coef)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p1->coef;  q->exp = p1->exp;  q->next = NULL;  rear->next = q;  rear = q;  }  p1 = p1->next;  }  }  while (p1)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p1->coef;  q->exp = p1->exp;  q->next = NULL;  rear->next = q;  rear = q;  p1 = p1->next;  }  while (p2)  {  q = (LinkList)malloc(sizeof(struct node));  q->coef = p2->coef\*-1;  q->exp = p2->exp;  q->next = NULL;  rear->next = q;  rear = q;  p2 = p2->next;  }  check(ps);  return ps;  }  //两个多项式的相乘  LinkList MultiplyPolyn(LinkList p1, LinkList p2)  {  float coef;  int exp;  LinkList pp,q1,q2,q,r;  //建立头结点  pp = (LinkList)malloc(sizeof(struct node));  pp->coef = 0;  pp->exp = -1;  pp->next = NULL;  for(q2 = p2->next; q2 ;q2 = q2->next)  {  for(q1 = p1->next; q1 ;q1 = q1->next)  {  coef = q1->coef \* q2->coef;  exp = q1->exp + q2->exp;  q = pp;  while(q->next && q->next->exp < exp)  q = q->next;    if(q->next && q->next->exp == exp)  q->next->coef += coef;  else  {  r = (LinkList)malloc(sizeof(struct node));  r->coef = coef;  r->exp = exp;  r->next = q->next;  q->next = r;  }  }  }  check(pp);  return pp;  }  //多项式的销毁  int DestroyPolyn(LinkList \* p)  {  //连同头结点一并销毁  LinkList pre = (\*p);  LinkList q = (\*p)->next;  for(;q;q = q->next)  {  free(pre);  pre = q;  }  free(pre);  \*p = NULL;  return 1;  }  //检查并销毁系数为0的节点  void check(LinkList p)  {  LinkList q;  while(p->next!=NULL)  {  //当没有到达链表尾部的时候，继续循环  if(p->next->coef == 0)  {  q=p->next;//保存L->next的指针位置  p->next = p->next->next;//改变链表的指针  free(q);//释放节点空间q  }  else  p= p->next;//往下循环  }  }   1. **调试分析** 2. 调试过程中遇到的问题是如何解决的以及对设计与实现的回顾讨论和分析   多项式读入时要注意排序以及指数相同的情况   1. 算法的时空分析，改进设想   多项式乘法的时间复杂度达到了O(n3),需要改进。可以考虑在多项式计算完成后，再使用排序   1. **测试数据与结果** |