**《数据结构》上机报告**

**2018 年 10 月 20 日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名：** | **赵得泽** | **学号：** | **1753642** | **班级：** | **电子二班** | **得分：** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验题目** | | **有序线性表** | | |
| **问题描述** | | 有序线性表包括有序顺序表和链表，本次实验是以一元多项式为模型进行链表的运用。链表的存储方式在上次实验中已经做了描述，本次实验主要运用链表的存储方式进行实际应用。一元多项式是有序线性表的典型应用，用一个长度为m且每个元素有两个数据项（系数项和指数项）的线性表((p1,e1),(p2,e2),...,(pm,em))可以唯一地表示一个多项式。 | | |
| **基本要求** | | 1. 实现多项式的表示和相加； 2. 实现多项式的求值. | | |
| **已完成基本内容（序号）：** | 1,3 | |
| **选做要求** | | 1. 实现多项式的表示和相乘； | | |
| **已完成选做内容（序号）** | | 2 |
| **数据结构**  **设计** | | struct poly  {  int coef;  int expn;  poly \*next;  };  本次实验使用的数据结构是线性表中的链表，主要是对链表的应用，即多项式表示和各种操作。在上次实验中我们已经对链表存储及操作方式有了深刻的认识，它是在内存中用一组任意的存储单元来存储线性表的数据元素，用每个数据元素所带的指针来确定其后继元素的存储位置。在该结构体中包括指数域expn,系数域coef，以及连接结点的指针域next。有序线性表是指将输入的有序数据串联在链表中，从而形成里有序链表，这样就方便按照题目进行链表的所有操作。 | | |
| **功能(函数)说明** | | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  功能：尾插法创建单向有头结点链表  方法特点：顺序读入，顺序输出  输入参数：链表长度n  说明:由于本次题目中要求指数是顺序增加的，故采用尾插法就可以避免了排序，直接在头结点的基础上进行读取就行了.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void CreatL(poly \*h, int n)  {  poly \*p, \*q;  q = h;  int c, e;  for (int i = 0; i < n; i++)  {  cin >> c >> e;  p = new poly;  p->coef = c;  p->expn = e;  q->next = p;  q = p;  }  q->next = NULL;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  功能：实现两个多项式的相加  输入参数：三个多项式链表的头结点  输出参数：相加之后的多项式链表的头结点  说明:多项式实现相加的时候，首先分别从两个多项式的头节点进行遍历，在若第一个多项式（p）的指数域和另一个多项式（q）的指数域相同，则将它们的系数域相加作为第三个多项式（r）的系数域，指数域为两者中任意一个即可；若系数域为零，则r不创建结点，反之则创建一个结点；最后将r的头结点输出，由此得到了相加之后的的多项式链表；即为指数升序排列的多项式.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void ADD\_1\_2(poly \*h1, poly \*h2, poly \*h3)  {  poly \*p, \*q, \*r, \*s;  p = h1->next;  q = h2->next;  s = h3;  while (p&&q)  {  r = new poly;  if (p->expn == q->expn)  {  r->coef = p->coef + q->coef;  r->expn = p->expn;  p = p->next;  q = q->next;  }  else if (p->expn < q->expn)  {  r = p;  p = p->next;  }  else  {  r = q;  q = q->next;  }  s->next = r;  s = r;  }  s->next = (p) ? p : q;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  功能：创建单向有头结点链表  方法特点：头插法，逆序输出  输入参数：待插入元素，链表长度  说明：元素每次插入的时候从头结点插入，头结点始终指向新插入的元素；因为在  输入多项式的时候，两个多项式的数据域中的指针域都升序输入的，为了进行最终  的多项式升序输出和计算简便，我采取的是将一个多项式链表升序建立，另一个多  项式降序建立，就用到了尾插法、头插法；  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void CreatLH(poly \*h, int n)  {  poly \*p;  int c, e;  h->next = NULL;  for (int i = n; i > 0; i--)  {  p = new poly;  cin >> c >> e;  p->coef = c;  p->expn = e;  p->next = h->next;  h->next = p;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  功能：实现两个多项式相乘  方法特点：一升一降查找  输入参数：三个多项式链表的头结点  说明：首先找出两个相乘多项式的最大指数域相加，（由于分别利用头尾插法建立  的链表故不需要特地去查找最大指数域结点，直接分别两个链表的头和尾指数域相  加即可）然后从最大指数k逐一循环递减，在每一个k下都要对每个多项式进行查找，  对第一个多项式p（降序建立的）找到第一个使p->expn<=k的结点p，然后在另一个  多项式（升序建立的）中循环查找到第一个使p->expn + q->expn >= k的结点q;然  后再次在循环中对刚才查找到的p和q进行操作，如果两者的指数域相加等于k就将系  数域相乘加到coe上，在分别进行循环查找；如果两者的指数域相加小于k,就将升序  建立的链表p，继续向后遍历；直到两者相加等于k；如果两者相加大于k,就将将于  建立的链表q向后遍历，直到两者相加等于k;这样就找到了所有指数域相加等于k的  p,q；最后对得到的coe是否为0进行结点是否建立的判断，若建立，则用头插法，  （因为输出要升序），coe清零再回到最初的循环，直到k减小到0，就完成了链表相乘。  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void Mul\_1\_2(poly \*h1, poly \*h2, poly \*h3)  {  poly \*p, \*q, \*r;  h3->next = NULL;  p = h1->next;  q = h2;  while (q->next) q = q->next;  int max\_expn = p->expn + q->expn;  for (int k = max\_expn; k >= 0; k--)  {  int coe = 0;  p = h1->next;  while (p&&p->expn > k)  p = p->next;  q = h2->next;  while (q&&p&&p->expn + q->expn < k)  q = q->next;  while (p&&q)  {  if (p->expn + q->expn == k)  {  coe += p->coef\*q->coef;  p = p->next;  q = q->next;  }  else if (p->expn + q->expn < k)  q = q->next;  else  p = p->next;  }  if (coe != 0)  {  r = new poly;  r->coef = coe;  r->expn = k;  r->next = h3->next;  h3->next = r;  }  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  功能:多项式求值  输入参数：x  说明：顺序遍历多项式链表，然后再对每个结点利用pow(x,expn)函数进行幂的求值  （x已知）；最后将每次得到的值相加即可。  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void value\_poly(poly \*h, int x)  {  poly \*p;  p = h->next;  double value = 0;  while (p)  {  value += p->coef\* pow(x, p->expn);  p = p->next;  }  cout << setiosflags(ios::fixed) << setprecision(1) << value << endl;  } | | |
| **开发环境** | | Win10,VS2017,C++高级程序语言设计 | | |
| **调试分析** | | **多项式相加：**    **多项式相乘：**    **多项式求值：** | | |
| **心得体会** | 本次实验主要是对链表进行实际应用——**多项式的表示、相加、相乘及求值**，通过这几项操作，加深对链表使用的理解和掌握。通过这次实验，我对链表建立的方法有了更深层次的认识，要学会用不同的方法对同一个问题中的步骤进行处理。  比如**多项式相加**这个问题中，由于题目要求是升序输入指数，那么采用尾插法比较省力；  在**多项式相乘**这个问题中，通过对题目进行准确的分析之后，发现两种方法可以解决这个问题，**第一种就是直接利用常规思路**，每个多项式的每一项分别相乘，然后最终将幂相同的单项式相加，形成一个无序多项式，再进行幂的大小排序即可，这种方法将问题转化成了多项式相加及排序的问题，比较费时费力；然而在**另一种思路**中直接是找到**最大的指数和**，从最大到零依次遍历，找到每个指数对应的系数累加，那就相当于**直接找出了指数最大**的一项，再在循环中依次查找第二大、第三大……，再结合**头插法**，那么所有的项就都出来了，避免了繁琐的中间过程，一步到位。而且在这种方法中，对**头插法和尾插法建立链表的交替使用**，则将算法的简洁度提升了一个层次，比只运用一种方法建立省了好多事儿（因为要求是都要升序输入）。所以在这个算法中，其思路和链表建立的方法是比较重要的。  **多项式的求值**方法比较简单，只涉及了简单的建立和运算。 | | | |