重庆大学本科学生实验项目任务书

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实验题目 | | 双向链表及其应用 | | | | | |
| 学院 | 计算机学院 | | 专业 | | 计算机科学与技术 | 年级 | 2020 |
| 实验目的   * 1. 理解双向链表的逻辑特征以及双向链表的优点。 * 2. 理解头结点、尾结点以及设置头结点、尾结点的优点。 * 3. 熟练掌握带头结点、尾结点的双向链表的基本操作。 | | | | | | | |
| 实验内容：   * 1. 编写使用freelist 的带头、尾结点的双向链表类的定义，实现双向链表的基本操作。 * 2. 利用双向链表实现2个一元多项式的加法和乘法运算，运算结果得到的链表要求按照指数降序排列的多项式。   输入格式:  3 2 //第一行，两个正整数分别表示多项式的项数  5 4 //输入第一个多项式各项的系数和指数，按指数降序输入  -3 2  1 0 //第一个多项式：5x4-3x2+1  6 2 //输入第二个多项式各项的系数和指数，按指数降序输入  -3 1 //第二个多项式：6x2-3x  输出格式：  4 //相加得到的多项式的项数  5 4 //每一项的系数与指数，按指数降序排列输出  3 2  -3 1  1 0 //和：5x4+3x2-3x+1  6 //相乘得到的多项式的项数  30 6  -15 5  -13 4  9 3  6 2  -3 1 //乘积：30x6-15x5-13x4+9x3+6x2-3x     * 3. 最后提交完整的实验报告和源程序。 | | | | | | | |
| 参考资料：   * 1. Data Structures and Algorithm Analysis (C++ Version) Clifford A. Shaffer   + 2. Data Structure and Algorithm Analysis in C++ (Third Edition)，Mark Allen Weiss， Pearson Education, 2006.   + 3. Data Structures, Algorithms, and Applications in C++，Sartaj Sahni， McGraw-Hill, 1998.   + 4.《数据结构（ C 语言版）》，严蔚敏，吴伟民编著，清华大学出版社，2007年第1版 | | | | | | | |
| 任务下达日期 2020 年 10月 19 日 | | | | 完成日期 2021 年 10 月21 日 | | | |

说明：学院、专业、年级均填全称，如：计算机学院、计算机科学与技术、2011。

**注意：1.实验程序要提交到高级语言程序设计能力训练平台上面编译通过，系统会自动打分。平台使用方法见蓝墨云上传的文档《上机系统使用说明及注意事项》。特别要提醒同学们注意输入输出格式，经常不是因为程序而是因为输入输入不正确导致扣分。**

**2. 实验报告+源程序 一起提交到蓝墨云上面。提交的实验报告电子文档请按照规定格式正确命名。**

**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计算机科学与技术卓越一班** | | | **姓名** | **孙莹莹** |
| **实验题目** | 双向链表及其应用 | | | | | |
| **实验时间** | **2021.10.21** | | **实验地点** | **DS1404** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的   * 1. 理解双向链表的逻辑特征以及双向链表的优点。 * 2. 理解头结点、尾结点以及设置头结点、尾结点的优点。 * 3. 熟练掌握带头结点、尾结点的双向链表的基本操作。 | | | | | | |
| 二、实验项目内容   * 1. 编写使用freelist 的带头、尾结点的双向链表类的定义，实现双向链表的基本操作。 * 2. 利用双向链表实现2个一元多项式的加法和乘法运算，运算结果得到的链表要求按照指数降序排列的多项式。   输入格式:  3 2 //第一行，两个正整数分别表示多项式的项数  5 4 //输入第一个多项式各项的系数和指数，按指数降序输入  -3 2  1 0 //第一个多项式：5x4-3x2+1  6 2 //输入第二个多项式各项的系数和指数，按指数降序输入  -3 1 //第二个多项式：6x2-3x  输出格式：  4 //相加得到的多项式的项数  5 4 //每一项的系数与指数，按指数降序排列输出  3 2  -3 1  1 0 //和：5x4+3x2-3x+1  6 //相乘得到的多项式的项数  30 6  -15 5  -13 4  9 3  6 2  -3 1 //乘积：30x6-18x5-13x4+9x3+6x2-3x | | | | | | |
| 1. 实验过程或算法（源程序） 2. 利用freelist实现双链表   其中每一个结点存储了一个项的系数，该项的指数，及指向前面节点和后面节点的指针。  template <typename E>  class Link  // the node  {  private:      static Link<E>\* freelist;  public:      E element; //该项的系数      E ezhinum;  //该项的指数      Link\* next; //后结点      Link\* prev;//前结点      //constructure function      Link(const E itnum, const E itzhi, Link\* prevp, Link\* nextp)      {          element = itnum;          ezhinum  = itzhi;          prev = prevp;          next = nextp;      }      Link(Link\* prevp = NULL, Link\* nextp = NULL)      {          prev = prevp;          next = nextp;      }        //overloaded "new" operator,      void\* operator new(size\_t)      {          if (freelist == nullptr)              return ::new Link;          Link<E>\* temp = freelist;          freelist = freelist->next;          return temp;      }      //overloaded "delete" opetator      void operator delete(void\* ptr)      {          ((Link<E>\*) ptr)->next = freelist;          freelist = (Link<E>\*) ptr;      }  };  //initialize freelist  template <typename E>  Link<E>\* Link<E>::freelist = NULL;  template <typename E>  class Doublelist  // 双向链表  {  private:      Link<E>\* head;      Link<E>\* tail;  //both tail and head has no element and ezhinum      Link<E>\* curr;      int cnt;      void init()  //initialization      {          curr = head =new Link<E>;          tail =new Link<E>;          curr->next = tail;          tail->prev = curr;          cnt = 0;      }      void removeall()      {          while (head != NULL)          {              curr = head;              head = head->next;              delete curr;          }      }  public:      Doublelist()  //construction      {          init();      }      Doublelist(Doublelist<E>&& L)   // copy      {          this->head = L.head;          this->tail = L.tail;          this->curr = L.curr;          this->cnt = L.cnt;          L.head = NULL;          L.tail = NULL;          L.curr = NULL;          L.cnt = 0;      }      ~Doublelist()      {          removeall();      }      Doublelist<E>& operator = (Doublelist<E>&& L)  //override =      {          this->head = L.head;          this->tail = L.tail;          this->curr = L.curr;          this->cnt = L.cnt;          L.head = NULL;          L.tail = NULL;          L.curr = NULL;          L.cnt = 0;          return \*this;      }      void clear()      {          removeall();          init();      }      void insert(const E& itnum, const E& itzhi)  //insert a node at curr position      {          curr->next = curr->next->prev = new Link<E>(itnum, itzhi, curr, curr->next);          cnt++;      }      void append(const E& itnum, const E& itzhi)  //append a node at the end of the list(before tail)      {          auto newnode = new Link<E>(itnum, itzhi, tail->prev, tail);          tail->prev->next = newnode; // move the next pointer of the node before tail to newnode          tail->prev = newnode; //set the prev pointer of tail to newnode          cnt++;  //长度+1      }      E remove()      {          if (curr->next == tail)              return NULL;          E itnum = curr->next->element;          E itzhi = curr->next->ezhinum;          Link<E>\* temp = curr->next;          curr->next->next->prev = curr;          curr->next = curr->next->next;          delete temp;          cnt--;          return itnum;      }      E currEle() //返回当前节点的系数      {          return curr->element;      }      E currNum() //返回当前结点的指数      {          return curr->ezhinum;      }        void changecurr(const E & it) //改变当前结点的系数值      {          curr->element += it;      }      void prev()      {          if (curr != head)              curr = curr->prev;      }      void moveToStart()      {          curr = head;      }      void moveToFirst()  //move to the first actual node      {          curr = head->next;      }      void moveToEnd()      {          curr = tail;      }      void next()      {          if (curr != tail)              curr = curr->next;      }      bool istail() //判断有没有到表尾      {          return curr==tail;      }      int length() const      {          return cnt;      }      void print()  //print the node's element and elezhinum from the first one to the last one      {          moveToFirst();          cout << cnt << endl;          for(int i=0; i<cnt; i++)          {              cout << curr->element << " " << curr->ezhinum << endl;              curr = curr->next;          }      }  };   1. 实现多项式的加法   思路：用一个新的链表C存储结果，依次比较两个链表A,B，将指数较大且唯一的该项直接加入到C中，该链表curr指针向后移一位；如果指数相同且系数相加不为零，将相加后的结点加入C中，A,B的curr指针均向后移一位。重复上述步骤，直到有一个curr指针指向了链表的tail结点。然后将没有遍历完的那一个链表后面的结点直接加入C中。  Doublelist<int> add(Doublelist<int> &A, Doublelist<int> &B)  {      Doublelist<int> newone;      A.moveToFirst();      B.moveToFirst();      int countA = A.length();      int countB = B.length();      while(true)      {          if(A.currNum() == B.currNum())          {              if(A.currEle()+B.currEle()!=0)              {                  newone.append(A.currEle()+B.currEle(),A.currNum());              }              countA--;              countB--;              A.next();              B.next();          }          else if(A.currNum() > B.currNum())          {              newone.append(A.currEle(),A.currNum());              A.next();              countA--;          }          else if(A.currNum() < B.currNum())          {              newone.append(B.currEle(),B.currNum());              B.next();              countB--;          }          if(countA==0 || countB==0)          {              break;          }      }      // find the list that hasn't move to tail and continue      if(countA==0 && countB!=0)      {          while(countB != 0)          {              newone.append(B.currEle(),B.currNum());              B.next();              countB--;          }      }      else if(countB==0 && countA!=0)      {          while(countA != 0)          {              newone.append(A.currEle(),A.currNum());              A.next();              countA--;          }      }      return newone;  };   1. 实现多项式的乘法   思路：用一个新的链表C存储结果，以A链表为基准，不断遍历B链表，将B链表结点与当前A链表结点相乘，B链表到达链表尾部后，将其curr指针指回第一个元素，A链表的curr指针向后移一位，直到A链表curr指针到达队尾。在A链表与B链表结点相乘的时候，需要比较新的结点的指数与C链表之前结点指数的大小，如果比他大就插到他前面，否则就加到他后面。  Doublelist<int> multiple(Doublelist<int> &A, Doublelist<int> &B)  {      Doublelist<int> newone;      A.moveToFirst();      B.moveToFirst();      int countA = A.length();      int countB = B.length();      while(countA!=0)      {          if(countA==A.length())          {              for(int i=0; i<countB; i++)              {                  newone.append(A.currEle()\*B.currEle(),A.currNum()+B.currNum());                  B.next();                  newone.next();              }             B.moveToFirst();          }          else          {              for(int i=0; i<countB; i++)              {                  int newele = A.currEle() \* B.currEle();                  int newnum = A.currNum()+ B.currNum();                  if(newnum > newone.currNum())                  {                      newone.prev();                      int s = 0;                      while(newone.currNum()<newnum)                      {                          newone.prev();                          s++;                      }                      if(newone.currNum()==newnum)                      {                          if(newone.currEle()+newele!=0)                          {                              newone.changecurr(newnum);                          }                          else                              newone.remove();                      }                      else                      {                          newone.insert(newele,newnum);                        }                      for(int i=0; i<s; i++)                      {                          newone.next();                      }                      if(countB!=1)                      {                          B.next();                      }                      }                  else if(newnum == newone.currNum())                  {                      if(newele + newone.currEle()!=0)                      {                          newone.changecurr(newele);                          newone.next();                      }                      else                      {                          newone.remove();                      }                        if(countB!=1)                      {                          B.next();                      }                    }                  else                  {                      newone.append(newele,newnum);                      newone.next();                      if(countB!=1)                      {                          B.next();                      }                    }              }              B.moveToFirst();            }          countA--;          A.next();      }      return newone;  } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程 2. 实验结果   ZN791_VU(DVG4BVD`A~G%7E  如图，可以得到正确的实验结果  2.调试过程  在第一次编写乘法函数的时候，没有考虑新得到的结点需要和之前多个结点的指数值进行比较，而只是和C中的当前节点进行了比较。在推理为什么只需要和当前节点比较时发现错误，于是重新修改了算法。其次也没有考虑到当指数相同，系数相加有可能为0，需要remove这个结点。在修改算法之后，得到了一个比较完备的乘法函数 | | | | | | |