**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **计算机科学与技术2020级01班** | | | **姓名** | **陈鹏宇** |
| **实验题目** | **双向链表及其应用** | | | | | |
| **实验时间** | **2021.10.21** | | **实验地点** | **DS1408** | | |
| **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。 * 3. 最后提交完整的实验报告和源程序。 | | | | | | |
| 三、实验过程或算法（源程序）  #include <iostream>  #include <cassert>  #include <cstring>  #include <cstddef>  #define max 9//这里规定指数的最大值为9,可修改  using namespace std;  template <typename E>  class Link  {  private:  static Link<E>\* freelist;  public:  E element;  Link \*next;  Link \*prev;  Link(const E& it , Link\* prevp , Link\* nextp)  {  element = it;  prev = prevp;  next = nextp;  }  Link(Link\* prevp = nullptr , Link\* nextp = nullptr)  {  next = nextp;  prev = prevp;  }  void\* operator new(size\_t){  if(freelist == nullptr) return ::new Link;  Link<E>\* temp = freelist;  freelist = freelist->next;  return temp;  }  void operator delete(void\* ptr){  ((Link<E>\*)ptr)->next = freelist;  freelist = (Link<E>\*)ptr;  }  };  template <typename E>  Link<E>\* Link<E>::freelist = nullptr;  template <typename E>  class List  {  private:  void operator = (const List&) {}  List(const List&) {}  public:  List() {};  virtual ~List() {};  virtual void clear() = 0;  virtual void insert(const E& item) = 0;  virtual E remove() = 0;  virtual void moveToStart() = 0;  virtual void moveToEnd() = 0;  virtual void prev() = 0;  virtual void next() = 0;  virtual int length() const = 0;  virtual int currPos() const = 0;  virtual void moveToPos(int pos) = 0;  virtual const E& getValue() = 0;  };  template <typename E>  class LList : public List<E>  {  private:  Link<E>\* head;  Link<E>\* tail;  Link<E>\* curr;  int cnt;  void init()  {  curr = head = new Link<E>; //头节点  tail = new Link<E>; //尾节点  head->next = tail ; head->prev = nullptr;  tail->prev = head ; tail->next = nullptr;  cnt = 0;  }  void removeall()  {  while(head != nullptr){  curr = head;  head = head->next;  delete curr;  }  }  public:  void Assert(bool val, string s)  {  if (!val) {  cout << "Assertion Failed: " << s << endl;  exit(-1);  }  }  LList(int size = 100) { init(); size++;}  ~LList() { removeall(); }  void print() const  {  Link<E>\* temp;  temp = head->next;  for(int i = 1 ; i <= cnt ; i++)  {  cout<<temp->element<<" ";  temp = temp->next;  }  }  void clear() {removeall() ; init();}  void insert(const E& it)  {  curr->next = curr->next->prev = new Link<E>(it , curr , curr->next);  cnt++;  }  void append(const E& it)  {  tail->prev = tail->prev->next = new Link<E>(it,tail->prev,tail);  cnt++;  }  E remove()  {  Assert(curr->next != tail , "No value");  E it = curr->next->element;  Link<E>\* ltemp = curr->next;  curr->next->next->prev = curr;  curr->next = curr->next->next;  delete ltemp;  cnt--;  return it;  }  void moveToStart() { curr = head; }  void moveToEnd() { curr = tail; }  void prev()  {  if(curr == head)  return;  else  curr = curr->prev;  }  void next() { curr = curr->next; }  int length() const { return cnt; }  int currPos() const  {  Link<E>\* temp = head;  int i;  for( i = 0 ; curr != temp ; i++)  temp = temp->next;  return i;  }  void moveToPos(int pos)  {  Assert(pos >= 0 && pos <= cnt , "Position out of range");  curr = head;  for(int i = 0 ; i < pos ; i++) curr = curr->next;  }  const E& getValue()  {  Assert(curr->next != tail , "No value");  return curr->next->element;  }  };  struct item{ //构造项结构储存指数和系数  int coe; //系数  int index; //指数  };  int main()  {  {/\*基本功能的实现\*/  // LList<int> a(5);  // for(int i = 0 ; i < 5 ; i++)  // {  // int temp; cin >> temp;  // a.append(temp);  // }  // a.moveToStart();  // for(int i = 0 ; i < 5 ; i++)  // {  // cout<<a.getValue();  // cout<<a.currPos()<<endl;  // a.next();  // }  // a.print();  }  int n , m ; cin >> n >> m; //两个多项式的项数n,m  LList<item> ln(n) ; LList<item> lm(m) ;  LList<item> ans1 ; //储存加法运算结果  LList<item> ans2 ; //储存乘法运算结果  int box1[max+1]{0}; //储存加法对应指数的系数  int box2[max\*2+1]{0};//储存乘法对应指数的系数  for(int i = 0 ; i < n ; i++)//ln初始化  {  item temp;  cin >> temp.coe >> temp.index;  box1[temp.index] += temp.coe;  ln.append(temp);  }  for(int i = 0 ; i < m ; i++)//lm初始化  {  item temp;  cin >> temp.coe >> temp.index;  box1[temp.index] += temp.coe;  lm.append(temp);  }  for(int i = max-1 ; i >= 0 ; i--)  {  if(box1[i])  {  item temp ; temp.index = i ; temp.coe = box1[i];  ans1.append(temp);  }  }  cout<<ans1.length()<<endl;  for(int i = 0 ; i < ans1.length() ; i++)  {  cout<<ans1.getValue().coe<<" "<<ans1.getValue().index<<endl;  ans1.next();  }  ln.moveToStart() ;  for(int i = 0 ; i < n ; i++)  {  int t1 = ln.getValue().index ; int t2 = ln.getValue().coe;  ln.next();  lm.moveToStart();  for(int j = 0 ; j < m ; j++)  {  int t3 = lm.getValue().index ; int t4 = lm.getValue().coe;  box2[t3+t1] += t2\*t4;  lm.next();  }  }  for(int i = max\*2 ; i >= 0 ; i--)  {  if(box2[i])  {  item temp ; temp.index = i ; temp.coe = box2[i];  ans2.append(temp);  }  }  cout<<ans2.length()<<endl;  ans2.moveToStart();  for(int i = 0 ; i < ans2.length() ; i++)  {  cout<<ans2.getValue().coe<<" "<<ans2.getValue().index<<endl;  ans2.next();  }  return 0;  } | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  实验分析：该代码将会设置两个个多项式中的最大指数为x，多项式加法后项数最多不会超过x+1项。由于此链表不方便修改元素值，所以以指数为索引，将系数放在数组中，通过遍历存入双向链表里面。乘法同理，多项式乘法后项数最多不超过x\*2+1项，通过二重循环得到系数和指数存入数组，再导入链表输出。  源程序调试： | | | | | | |