《数据结构与算法》实验报告2

——顺序表、单链表、双向链表的基本操作

吉林大学数学学院2020级计算1班 姓名 陈文宇 学号10200115

日期 2023/5/28

一、实验题目

分别编写顺序表、单链表、双向链表的基本操作函数并验证结果。

二、算法简述

三、程序代码

|  |
| --- |
| Table\_Basic\_Opration.cpp |
| //陈文宇  //10200115  //#include"stdafx .h"  #include<iostream>  using namespace std;  const int LISTINIT\_SIZE=100;  const int LISTINCREMENT=10;  const bool TRUE=1;  const bool FALSE=0;  typedef char ElemType;  typedef struct {  ElemType \*elem;  int length;  int listsize;  int incrementsize;  }SqList;  void InitList\_Sq(SqList &L,int maxsize,int incresize);  int LocateElem\_Sq(SqList L,ElemType e);  void ListInsert\_Sq(SqList &L,int i, ElemType e);  void increment(SqList &L);  void ListDelete\_Sq(SqList &L,int i,ElemType &e);  void DestroyList\_Sq(SqList &L);  void invert(char \*R,int s,int t);  void ListTraverse\_Sq(SqList L);  void ClearList\_Sq(SqList &L);  bool ListEmpty(SqList L);  int ListLength\_Sq(SqList L);  ElemType GetElem\_Sq(SqList L,int i, ElemType e);  ElemType PriorElem\_Sq(SqList L,ElemType cur\_e,ElemType &pre\_e);  ElemType NextElem\_Sq(SqList L,ElemType cur\_e,ElemType &next\_e);  int main(){  SqList L;  InitList\_Sq(L,100,10);  L.elem[0]='a';  L.elem[1]='b';  L.elem[2]='c';  L.elem[3]='d';  L.length=4;    cout<<"顺序表为： ";  ListTraverse\_Sq(L);    cout<<"查找'c'的位序： ";  printf("%d\n",LocateElem\_Sq(L,'c'));    cout<<"插入'f'后，顺序表为：";  ListInsert\_Sq(L,5,'f');    ListTraverse\_Sq(L);    cout<<"逆置后，顺序表为: ";  invert(L.elem,0,4);  ListTraverse\_Sq(L);    }  //顺序表的基本操作  //初始化操作  void InitList\_Sq(SqList &L,int maxsize,int incresize){  //构造一个最大容量为maxsize 的顺序表L    L.elem = new ElemType[maxsize];//为顺序表分配一个最大容量嗯我imaxsize 的数组空间    L.length= 0; //顺序表中当前所含元素个数为0    L.listsize= maxsize; //该顺序表可以容纳的maxsize 个数据元素    L.incrementsize= incresize; //需要时可扩容 incresize 个元素空间    }  //查找元素操作  int LocateElem\_Sq(SqList L,ElemType e){  //在顺序表L中查找第一个与e 相等的数据元素  //若找到，则返回其在L中的位序，否则返回0    int i; //i的初值为第一个元素的位序  i= 1;    ElemType \*p; //p 的初值为第一个元素的存储位置  p= L.elem;    while(i<=L.length && \*p++!=e) ++i;  if(i<=L.length) return i;  else return 0;    }  //插入元素操作  void ListInsert\_Sq(SqList &L,int i, ElemType e){  //在顺序表L的第i 个元素之前插入新的元素 e,  //i的合法值为 1<= i <= L.length+1,  //若表中容量不足，则按照该顺序表的预定义来增量扩容、  if(i<1 || i>L.length+1) printf("i值不合法");    else {  ElemType \*q,\*p;  if(L.length>=L.listsize) increment(L); //当前存储空间已满，为L增加分配L.incrementsize 个元素空间    q=&(L.elem[i-1]); //q 为插入位置  for(p=&(L.elem[L.length-1]);p>=q;--p) \*(p+1)=\*p; //插入位置及以后的元素右移    \*q=e; //插入e    ++L.length;  }    }  //扩容  void increment(SqList &L){  char \*a;  int i;    a=new char[L.listsize+L.incrementsize]; //a 为临时过渡的辅助数组    for(i=0;i<L.length;i++)a[i]=L.elem[i]; //腾挪原空间数据    delete[] L.elem; //释放数据元素所占的原空间 L.elem    L.elem = a; //移交空间首地址    L.listsize+=L.incrementsize; //扩容后的顺序表最大空间    }    //删除元素操作  void ListDelete\_Sq(SqList &L,int i,char &e){  ElemType \*p;  ElemType \*q;    if( i< 1 || i>L.length) printf("i值不合法");    p= &(L.elem[i-1]);  q= &(L.elem[L.length-1]);    for(++p; p<= q; ++p) \*(p-1) = \*p;    --L.length;    }  //销毁结构操作  void DestroyList\_Sq(SqList &L){    delete[] L.elem;  L.listsize=0;  L.length=0;    }  //逆置  void invert(ElemType \*R,int s,int t){    //本算法将数组 R 中下标 自 t 到 s 的元素逆置  int k;  ElemType w;    for(k= s; k<= (s+ t)/2.0; k++ ){  w=R[k];  R[k]= R[t+s-k];  R[t+s-k]=w;  }    }  //输出  void ListTraverse\_Sq(SqList L){  int i;  for(i=0;i<L.length;i++){    printf("%c ",L.elem[i]);  }  printf("\n");    }  //重置为空表  void ClearList\_Sq(SqList &L){  int i;  for(i=0; i<L.length; i++)  L.elem[i]='\0';    }  //判断是否为空表  bool ListEmpty(SqList L){  if(L.elem[0]!='\0')  return TRUE;  else  return FALSE;  }  //获取表长  int ListLength\_Sq(SqList L){    int i=0;  while(L.elem[i++]!='\0');  L.length=i;    }  //获取第 i 个元素  ElemType GetElem\_Sq(SqList L,int i, ElemType e){  if(i<1 || i>L.length) printf("i值不合法");  else {  e = L.elem[i-1];  return e;  }  }    //获取前驱  ElemType PriorElem\_Sq(SqList L,ElemType cur\_e,ElemType &pre\_e){    int i=0;    while(L.elem[i++]!=cur\_e && i<=L.length);  if(i==1 || i==L.length+1) printf("cer\_e没有前驱");  else {  pre\_e=L.elem[i-2];  return pre\_e;  }  }  //获取后继  ElemType NextElem\_Sq(SqList L,ElemType cur\_e,ElemType &next\_e){    int i=0;    while(L.elem[i++]!=cur\_e && i<=L.length-1);  if(i==L.length) printf("cer\_e没有后继");  else {  next\_e=L.elem[i];  return next\_e;  }  } |
| LinkList\_Basic\_Opration.cpp |
| //陈文宇  //10200115  //#include"stdafx .h"  #include<iostream>  using namespace std;  const int LISTINIT\_SIZE=100;  const int LISTINCREMENT=10;  const bool TRUE=1;  const bool FALSE=0;  typedef int ElemType;  //单链表定义  typedef struct LNode{ //typedef是用来设置结构体别名，使用过后 定义结构体可以 LNode p ;LNode \*p;有简化的效果  ElemType date;  struct LNode \*next; //结构体自引用-next作为指针 指向 结构体;  }LNode,\*LinkList; // LinkList 是通过结构体声明的指针变量（结构体指针），相当于LNode \*LinkList; LinkList是指向该种结构体的指针  //LNode本身就是一个结构体，但是它的命名是一个指针类型。  //单链表基本操作  int ListLength\_L(LinkList L);  LNode\* LocateElem\_L(LinkList L,ElemType e);  void LinkInsert\_L(LinkList &L,LNode \*p,LNode \*s);  void ListDelete\_L(LinkList L,LNode \*p);  void CreateList\_L(LinkList &L,ElemType \*A,int n);  void ListTraverse\_L(LinkList L);  int main(){  ElemType A[3]={1,2,3};  LinkList L,p,s,q;  L=NULL;  //printf("1");    CreateList\_L(L,A,3);  //printf("2");  cout<<"单链表输出:";  ListTraverse\_L(L);    cout<<"长度："<<ListLength\_L(L)<<endl;    p=LocateElem\_L(L,1);  cout<<"获取数据为1的结点，它的数据为:"<<p->date<<endl;  //printf("1");      s=new LNode;//给一个新结点分配内存，并返回其地址给 s ,s是一个储存了结构体地址的指针变量,而这个结构体名本身就是一个指针  //举例，s房间簿, \*s 指向是房间名，房间名本身也是一个指针，它指向房间内部的东西  //new命令 建造了一间新房子，并把这个的地址返回房间簿，  (\*s).date=0;//等价于s->date=0;  LinkInsert\_L(L,p,s);  cout<<"插入结点后，单链表输出:";  ListTraverse\_L(L);  cout<<"长度："<<ListLength\_L(L)<<endl;      q=LocateElem\_L(L,2);  //printf("陈文宇");  cout<<"获取数据为2的结点，它的数据为:"<<q->date<<endl;  //printf("陈文宇");  ListDelete\_L(L,q);  cout<<"删除结点后，单链表输出:";  ListTraverse\_L(L);  cout<<"长度："<<ListLength\_L(L)<<endl;  delete s;  }  //线性表基本操作  //求线性表的长度  int ListLength\_L(LinkList L){  //L为链表的头指针，本函数返回L 所指链表的长度  LinkList p;  int i=0;  p=L;  while(p){  i++;  p=p->next;  }  return i;  }  //查找元素  LinkList LocateElem\_L(LinkList L,ElemType e){  LinkList p;  p=L;  while(p && p->date!=e) p=p->next;    return p;  }  //插入结点操作  void LinkInsert\_L(LinkList &L,LNode \*p,LNode \*s){  //将 s 插入到 p前  LNode \*q;    if(p==L){  s->next=p;  L=s;  }  else {  q=L;  while(q->next != p) q=q->next;  s->next=p;  q->next=s;  }  }  //删除结点操作  void ListDelete\_L(LinkList L,LNode \*p){  LNode \*q;  if(p==L){  L=p->next;  }  else{  q=L;  while(q->next!=p) q=q->next;  q->next=p->next;  }  delete p;  }  //创建单链表  void CreateList\_L(LinkList &L,ElemType \*A,int n){  int i;  LNode \*s;    L = NULL;  /\*  s->date=A[n-1];  s->next=L;  L=s;  \*/      for(i=n-1;i>=0;--i){  s=new LNode;  s->date=A[i];  //printf("3");  s->next=L;  L=s;  }  }  //遍历输出  void ListTraverse\_L(LinkList L){  LNode \*p;  p=L;  while(p){  cout<<p->date<<" ";  p=p->next;  }  cout<<endl;  } |
| DuLinkList\_Basic\_Opration.cpp |
| //陈文宇  //10200115  //#include"stdafx .h"  #include<iostream>  using namespace std;  const int LISTINIT\_SIZE=100;  const int LISTINCREMENT=10;  const bool TRUE=1;  const bool FALSE=0;  typedef int ElemType;  //双向链表定义  typedef struct DuLNode{ //typedef是用来设置结构体别名，使用过后 定义结构体可以 LNode p ;LNode \*p;有简化的效果  ElemType date;  struct DuLNode \*prior;  struct DuLNode \*next; //结构体自引用-next作为指针 指向 结构体;  }DuLNode,\*DuLinkList; // LinkList 是通过结构体声明的指针变量（结构体指针），相当于LNode \*LinkList; LinkList是指向该种结构体的指针  //LNode本身就是一个结构体，但是它的命名是一个指针类型。  //函数声明  void CreateList\_DuL(DuLinkList &L,ElemType \*A,int n);  int ListLength\_DuL(DuLinkList L);  DuLinkList LocateElem\_DuL(DuLinkList L,ElemType e);  void ListInsert\_DuL(DuLinkList &L,DuLNode \*p,DuLNode \*s);  void ListDelete\_DuL(DuLinkList &L,DuLNode \*p,ElemType &e);  void ListTraverse\_DuL(DuLinkList L);  int main(){  ElemType A[4]={1,2,3,4};  ElemType e;  DuLinkList L,p,s,q;  //printf("1");  CreateList\_DuL(L,A,4);  //printf("2");  cout<<"双向循环链表输出:";  ListTraverse\_DuL(L);    cout<<"长度："<<ListLength\_DuL(L)<<endl;    p=LocateElem\_DuL(L,1);  cout<<"获取数据为1的结点，它的数据为:"<<p->date<<endl;  //printf("1");      s=new DuLNode;//给一个新结点分配内存，并返回其地址给 s ,s是一个储存了结构体地址的指针变量,而这个结构体名本身就是一个指针  //举例，s房间簿, \*s 指向是房间名，房间名本身也是一个指针，它指向房间内部的东西  //new命令 建造了一间新房子，并把这个的地址返回房间簿，  s->date=0; //等价于(\*s).date=0;  ListInsert\_DuL(L,p,s);  cout<<"插入结点后，双向循环链表输出:";  ListTraverse\_DuL(L);  cout<<"长度："<<ListLength\_DuL(L)<<endl;      q=LocateElem\_DuL(L,2);  //printf("陈文宇");  cout<<"获取数据为2的结点，它的数据为:"<<q->date<<endl;  //printf("陈文宇");  ListDelete\_DuL(L,q,e);  cout<<"删除结点后，双向循环链表输出:";  ListTraverse\_DuL(L);  cout<<"长度："<<ListLength\_DuL(L)<<endl;    delete s;  }  //创建双向循环链表  void CreateList\_DuL(DuLinkList &L,ElemType \*A,int n){  DuLNode \*s;    //初始化  L=new DuLNode;  L->prior=L;  L->next=L;    //赋值  for(int i=n-1;i>=0;--i){  s=new DuLNode;  s->date=A[i];  s->prior=L->prior;  L->prior->next=s;  s->next=L;  L->prior=s;  L=s;    }  }  //求双向循环链表的长度  int ListLength\_DuL(DuLinkList L){  //L为链表的头指针，本函数返回L 所指链表的长度  DuLinkList p;  int i=0;  p=L;  while(p->next != L){  i++;  p=p->next;  }  return i;  }  //查找元素  DuLinkList LocateElem\_DuL(DuLinkList L,ElemType e){  DuLNode\* p;  p=L;  while(p->next!=L && p->date!=e) p=p->next;  if(p->next==L) cout<<"无该结点"<<endl;  else return p;  }  //插入  void ListInsert\_DuL(DuLinkList &L,DuLNode \*p,DuLNode \*s){  //在带头结点的双向循环链表L中p结点之前插入s结点  if(p!=L){  s->prior=p->prior;  p->prior->next=s;  s->next=p;  p->prior=s;  }  else{  s->prior=p->prior;  p->prior->next=s;  s->next=p;  p->prior=s;  L=s;  }    }//ListInsert\_DuL  //删除结点  void ListDelete\_DuL(DuLinkList &L,DuLNode \*p,ElemType &e){  //删除带头结点的双向循环列表L中的p结点，并以e 返回它的数据元素  e=p->date;  p->prior->next=p->next;  p->next->prior=p->prior;  delete p;  }//ListDelete\_DuL  //遍历输出  void ListTraverse\_DuL(DuLinkList L){  DuLNode \*p;  p=L;  while(p->next !=L ){  cout<<p->date<<" ";  p=p->next;  }  cout<<endl;  } |

四、实验结果





