华北电力大学

**实 验 报 告**

|

|

**实验名称** 线性表

**课程名称**  数据结构综合实践

|

|

专业班级：信息2202 学生姓名：徐梓乔

学 号：220221100327 成 绩：

|  |
| --- |
| 一、实验内容  用C#编写一个学生课程表管理系统，实现如下的功能：  可以选择课程表的存储结构：顺序表、单链表、循环链表、双向链表；  可以对给定的课程表进行插入、删除的操作；  可以按给定的课程序号、课程名称或课程性质查询课程的信息；  二、实验方法与步骤  打出顺序表，单链表，循环链表，双向链表的代码，再根据窗体操作将他们串联在一起。  具体代码如下：  顺序表:建立线性表接口,并利用顺序储存结构实现顺序表.  public interface ILinearList<T> where T : IComparable<T>  {  int Length { get; }  T this[int index] { get; set; }  bool IsEmpty();  void Insert(int index, T data);  void Remove(int index);  int Search(T data);  void Clear();  }  public class SeqList<T>: ILinearList<T> where T : IComparable<T>  {  protected readonly T[] Dataset;  public int Length { get; private set; }  public int MaxSize { get; }  public SeqList(int max)  {  if (max <= 0)  throw new ArgumentOutOfRangeException();  MaxSize = max;  Dataset = new T[MaxSize];  Length = 0;  }  public T this[int index]  {  get  {  if(index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  return Dataset[index];  }  set  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  Dataset[index] = value;  }  }  public bool IsEmpty()  {  return Length == 0;  }  public void Insert(int index, T data)  {  if (index < 0 || index > Length )  throw new IndexOutOfRangeException();  if (Length == MaxSize)  throw new Exception("达到最大值");  for(int i = Length; i > index; i--)  {  Dataset[i] = Dataset[i-1];  }  Dataset[index] = data;  Length++;  }  public void Remove(int index)  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  for(int i = index; i < Length - 1; i++)  {  Dataset[i] = Dataset[i+1];  }  Length--;  }  public int Search(T data)  {  int i;  for(i = 0; i < Length; i++)  {  if (Dataset[i].CompareTo(data) == 0)  break;  }  return i == Length ? -1 : i;  }  public void Clear()  {  Length = 0;  }  }  单链表:对单链表结点进行封装,再利用线性表接口对单链表进行封装.  public class SNode<T> where T : IComparable<T>  {  public T Data { get; set; }  public SNode<T> Next { get; set; }  public SNode(T data, SNode<T> next = null )  {  Data = data;  Next = next;  }  }  public class SLinkList<T> : ILinearList<T> where T : IComparable<T>  {  public SNode<T> PHead { get; private set; }  public int Length { get; private set; }  public SLinkList()  {  Length = 0;  PHead = null;  }  public void InsertAtFirst(T data)  {  PHead = new SNode<T>(data, PHead);  Length++;  }  private SNode<T>Locate(int index)  {  if (index < 0 || index > Length - 1 )  throw new IndexOutOfRangeException();  SNode<T> temp = PHead;  for(int i = 0; i < Length; i++)  {  temp = temp.Next;  }  return temp;  }  public void InsertAtRear(T data)  {  if(PHead == null)  {  PHead = new SNode<T>(data);  }  else  {  Locate(Length - 1).Next = new SNode<T>(data);  }  Length++;  }  public T this[int index]  {  get  {  if(index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  return Locate(index).Data;  }  set  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  Locate(index).Data = value;  }  }  public bool IsEmpty()  {  return Length == 0;  }  public void Insert(int index, T data)  {  if (index < 0 || index > Length)  throw new IndexOutOfRangeException();  if (Length == 0)  InsertAtFirst(data);  else if(index == Length)  {  InsertAtRear(data);  }  else  {  SNode<T> temp = Locate(index - 1);  temp.Next = new SNode<T>(data, temp.Next);  Length++;  }  }  public void Remove(int index)  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  if(index == 0)  {  PHead = PHead.Next;  }  else  {  SNode<T>temp = Locate(index - 1);  temp.Next = temp.Next.Next;  }  Length--;  }  public int Search(T data)  {  int i;  SNode<T> temp = PHead;  for (i = 0; i < Length; i++)  {  if (temp.Data.CompareTo(data) == 0)  break;  temp = temp.Next;  }  return i == Length ? -1 : i;  }  public void Clear()  {  PHead = null;  Length = 0;  }  }  循环链表:利用封装好的单链表节点和线性表接口对循环链表进行封装.  public interface ILinearList<T> where T : IComparable<T>  {  int Length { get; }  T this[int index] { get; set; }  bool IsEmpty();  void Insert(int index, T data);  void Remove(int index);  int Search(T data);  void Clear();  }  public class CLinkList<T> : ILinearList<T> where T : IComparable<T>  {  public SNode<T> PRear { get;private set; }  public int Length { get; private set; }  public CLinkList()  {  Length = 0;  PRear = null;  }  public void InsertAtFirst(T data)  {  if (IsEmpty())  {  PRear = new SNode<T>(data);  PRear.Next = PRear;  }  else  {  SNode<T> temp = new SNode<T>(data, PRear.Next);  PRear.Next = temp;  }  Length++;  }  public void InsertAtRear(T data)  {  if (IsEmpty())  {  PRear = new SNode<T>(data);  PRear.Next = PRear;  }  else  {  SNode<T> temp = new SNode<T>(data, PRear.Next);  PRear.Next = temp;  PRear =temp;  }  Length++;  }  private SNode<T>Locate(int index)  {  if (index < 0 || index >= Length - 1)  throw new IndexOutOfRangeException();  SNode<T> temp = PRear.Next;  for(int i = 0; i < index; i++)  {  temp = temp.Next;  }  return temp;  }  public T this[int index]  {  get  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  return Locate(index).Data;  }  set  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  Locate(index).Data = value;  }  }  public bool IsEmpty()  {  return Length == 0;  }  public void Insert(int index, T data)  {  if(index < 0 || index > Length)  throw new IndexOutOfRangeException();  if(index == 0)  {  InsertAtFirst(data);  }  else if(index == Length)  {  InsertAtRear(data);  }  else  {  SNode<T> temp = Locate(index - 1);  temp.Next = new SNode<T>(data, temp.Next);  Length++;  }  }  public void Remove(int index)  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  if(PRear == PRear.Next)  {  PRear = null;  }  else  {  if(index == Length - 1)  {  SNode<T>temp = Locate(Length - 2);  temp.Next = PRear.Next;  PRear = temp;  }  else if(index == 0)  {  PRear.Next = PRear.Next.Next;  }  else  {  SNode<T> temp = Locate(index - 1);  temp.Next = temp.Next.Next;  }  }  Length--;  }  public int Search(T data)  {  int i;  SNode<T> temp = PRear;  for(i = 0; i < Length; i++)  {  if (temp.Next.Data.CompareTo(data) == 0)  break;  temp = temp.Next;  }  return (i == Length) ? -1 : i;  }  public void Clear()  {  Length = 0;  PRear = null;  }  }  双链表:先对双链表结点进行封装,在利用线性表接口进行对双链表的封装.  public class DNode<T> where T : IComparable<T>  {  public DNode<T> Prior { get; set; }  public DNode<T> Next { get; set; }  public T Data { get; set; }  public DNode(T data, DNode<T> prior = null, DNode<T>next = null)  {  Prior = prior;  Data = data;  Next = next;  }  }  public class DLinkList<T> : ILinearList<T> where T : IComparable<T>  {  public DNode<T> PHead { get; private set; }  public DNode<T> PRear { get; private set; }  public int Length { get; private set; }  public T this[int index]  {  get  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  return Locate(index).Data;  }  set  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  Locate(index).Data = value;  }  }  public DLinkList()  {  PHead = null;  PRear = null;  Length = 0;  }  public bool IsEmpty()  {  return Length == 0;  }  public void InsertAtFirst(T data)  {  if(IsEmpty())  {  DNode<T> temp = new DNode<T>(data);  PHead = temp;  PRear = temp;  }  else  {  DNode<T>temp = new DNode<T>(data, null, PHead);  PHead.Prior = temp;  PHead = temp;  }  Length++;  }  public void InsertAtRear(T data)  {  if (IsEmpty())  {  DNode<T> temp = new DNode<T>(data);  PHead = temp;  PRear = temp;  }  else  {  DNode<T> temp = new DNode<T>(data, PRear, null);  PRear.Prior = temp;  PRear = temp;  }  Length++;  }  private DNode<T> Locate(int index)  {  if (index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  DNode<T> temp = PHead;  for(int i = 0; i < index; i++)  {  temp = temp.Next;  }  return temp;  }  public void Insert(int index, T data)  {  if(index < 0 || index > Length)  throw new IndexOutOfRangeException();  if (index == 0)  InsertAtFirst(data);  else if (index == Length)  InsertAtRear(data);  else  {  DNode<T>temp1 = Locate(index);  DNode<T> temp2 = new DNode<T>(data, temp1.Prior, temp1);  temp2.Prior.Next = temp2;  temp2.Next.Prior = temp2;  Length++;  }  }  public void Remove(int index)  {  if(index < 0 || index > Length - 1)  throw new IndexOutOfRangeException();  if (Length == 1)  {  PHead = null;  PRear = null;  }  else  {  if(index == 0)  {  PHead =PHead.Next;  PHead.Prior = null;  }  else if(index == Length - 1)  {  PRear = PRear.Prior;  PRear.Next = null;  }  else  {  DNode<T>temp = Locate(index);  temp.Prior.Next = temp.Next;  temp.Next.Prior = temp.Prior;  }  }  Length--;  }  public int Search(T data)  {  int i;  DNode<T> temp = PHead;  for( i = 0; i < Length; i++)  {  if (temp.Data.CompareTo(data) == 0)  break;  temp = temp.Next;  }  return i == Length ? -1 : i;  }  public void Clear()  {  Length = 0;  PHead = null;  PRear = null;  }  }  最后在窗体设计中绑定这4种列表并显示.  三、实验结果    四、结论与总结  结论:线性表是由同一类型的数据元素构成的有序序列的线性结构  总结:可以利用线性表对成绩进行收集并整理. |
|  |