**线性表设计与实现专题讲座**

written by王保明

# 线性表基本概念

## 线性表定义

线性表(List)是零个或多个数据元素的集合

线性表中的数据元素之间是有顺序的

线性表中的数据元素个数是有限的

线性表中的数据元素的类型必须相同



## 数学定义

线性表是具有相同类型的 n（ ≥ 0）个数据元素的有限序列

（a1, a2, …, an）

ai是表项，n 是表长度。

## 性质

a0为线性表的第一个元素，只有一个后继

an为线性表的最后一个元素，只有一个前驱

除a0和an外的其它元素ai，既有前驱，又有后继  
线性表能够逐项访问和顺序存取

## 练习

下面的关系中可以用线性表描述的是

A.班级中同学的友谊关系

B.公司中的上下级关系

C.冬天图书馆排队占座关系

D.花名册上名字之间的关系

## 线性表的操作

创建线性表

销毁线性表

清空线性表

将元素插入线性表

将元素从线性表中删除

获取线性表中某个位置的元素

获取线性表的长度

线性表在程序中表现为一种特殊的数据类型

线性表的操作在程序中的表现为一组函数

|  |
| --- |
| C语言描述=====》线性表的设计与实现  人生财富库积累 |
| #ifndef \_WBM\_LIST\_H\_  #define \_WBM\_LIST\_H\_  typedef void List;  typedef void ListNode;  //创建并且返回一个空的线性表  List\* LinkList\_Create();  //销毁一个线性表list  void List\_Destroy(List\* list);  //将一个线性表list中的所有元素清空, 线性表回到创建时的初始状态  void List\_Clear(List\* list);  //返回一个线性表list中的所有元素个数  int List\_Length(List\* list);  //向一个线性表list的pos位置处插入新元素node  int List\_Insert(List\* list, ListNode\* node, int pos);  //获取一个线性表list的pos位置处的元素  ListNode\* List\_Get(List\* list, int pos);  //删除一个线性表list的pos位置处的元素 返回值为被删除的元素，NULL表示删除失败  ListNode\* List\_Delete(List\* list, int pos);  #endif |
|  |

# 线性表的顺序存储结构

## 1、基本概念

|  |
| --- |
|  |

## 2、设计与实现

|  |
| --- |
| 插入元素算法  判断线性表是否合法  判断插入位置是否合法  把最后一个元素到插入位置的元素后移一个位置  将新元素插入  线性表长度加1 |
| 获取元素操作  判断线性表是否合法  判断位置是否合法  直接通过数组下标的方式获取元素 |
| 删除元素算法  判断线性表是否合法  判断删除位置是否合法  将元素取出  将删除位置后的元素分别向前移动一个位置  线性表长度减1 |

## 3、优点和缺点

优点：

无需为线性表中的逻辑关系增加额外的空间

可以快速的获取表中合法位置的元素

缺点：

插入和删除操作需要移动大量元素

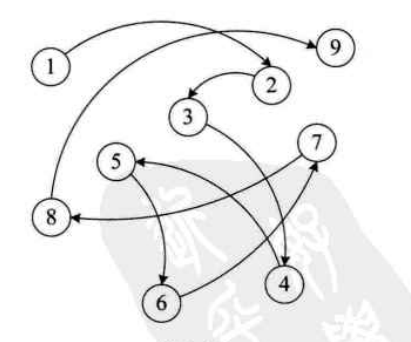
当线性表长度变化较大时难以确定存储空间的容量

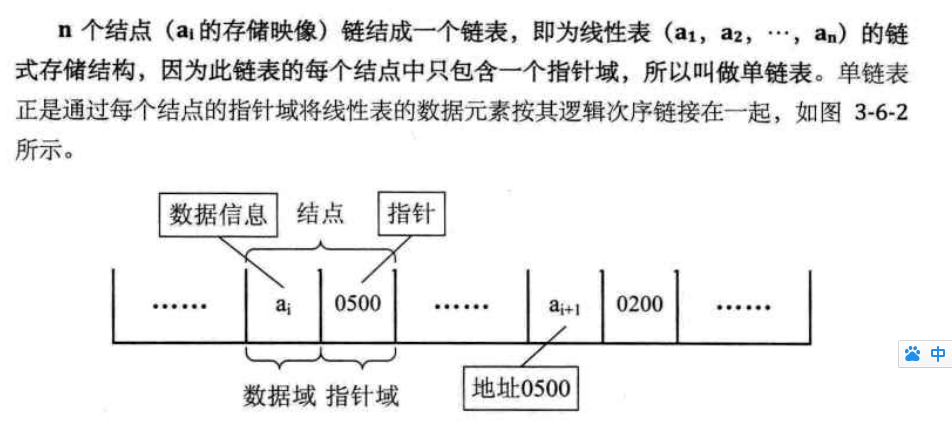
# 线性表的链式存储

## 1、基本概念

链式存储定义

为了表示每个数据元素与其直接后继元素之间的逻辑关系，每个元素除了存储本身的信息外，还需要存储指示其直接后继的信息。





表头结点

链表中的第一个结点，包含指向第一个数据元素的指针以及链表自身的一些信息

数据结点

链表中代表数据元素的结点，包含指向下一个数据元素的指针和数据元素的信息

尾结点

链表中的最后一个数据结点，其下一元素指针为空，表示无后继。

## 2、设计与实现

|  |
| --- |
| 在C语言中可以用结构体来定义链表中的指针域  链表中的表头结点也可以用结构体实现 |
|  |
|  |
| 带头结点、位置从0的单链表  返回链表中第3个位置处，元素的值  LinkListNode\* LinkList\_Get(LinkList\* list, int pos)  {  int i = 0;  TLinkList \*tList = (TLinkList \*)list;  LinkListNode \*current = NULL;  LinkListNode \*ret = NULL;  if (list==NULL ||pos<0 || pos>=tList->length)  {  return NULL;  }  current = (LinkListNode \*)tList;  for (i=0; i<pos; i++)  {  current = current->next;  }  ret = current->next;  return ret ;  }  返回第三个位置的  移动pos次以后，当前指针指向哪里？  答案：指向位置2，所以需要返回 ret = current->next;  备注：  循环遍历时， 遍历第1次，指向位置0  遍历第2次，指向位置1  遍历第3次，指向位置2  遍历第n次，指向位置n-1;  所以如果想返回位置n的元素的值，需要怎么做  ret = current->next;  此问题是：指向头结点的指针移动n次 和 第n个元素之间的关系？ |
| 删除元素 |

## 3、优点和缺点

优点：

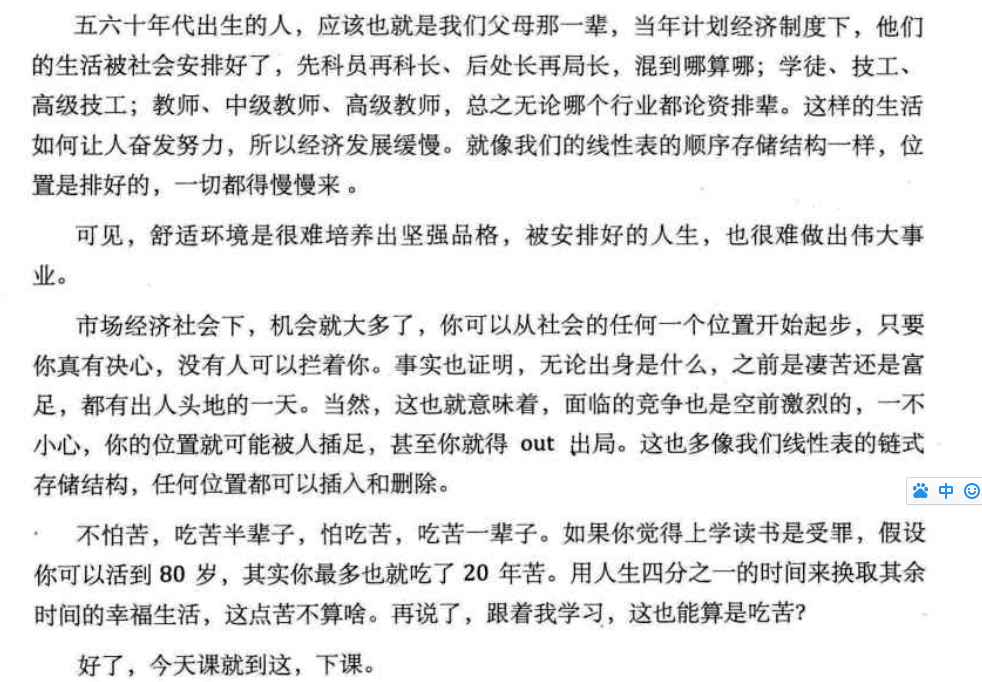
无需一次性定制链表的容量

插入和删除操作无需移动数据元素

缺点：

数据元素必须保存后继元素的位置信息

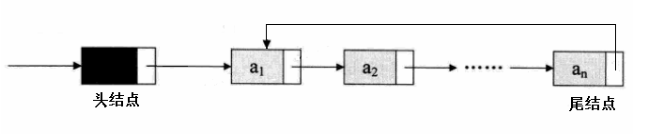
获取指定数据的元素操作需要顺序访问之前的元素



# 循环链表

## 1、基本概念

循环链表的定义：将单链表中最后一个数据元素的next指针指向第一个元素



|  |
| --- |
| 循环链表拥有单链表的所有操作 |
| 创建链表  销毁链表  获取链表长度  清空链表  获取第pos个元素操作  插入元素到位置pos  删除位置pos处的元素 |
| 新增功能：游标的定义 |
| 在循环链表中可以定义一个“当前”指针，这个指针通常称为游标，可以通过这个游标来遍历链表中的所有元素。 |
|  |
| 循环链表新操作 |
| 获取当前游标指向的数据元素  将游标重置指向链表中的第一个数据元素  将游标移动指向到链表中的下一个数据元素  直接指定删除链表中的某个数据元素 |
| CircleListNode\* CircleList\_DeleteNode(CircleList\* list, CircleListNode\* node);  CircleListNode\* CircleList\_Reset(CircleList\* list);  CircleListNode\* CircleList\_Current(CircleList\* list);  CircleListNode\* CircleList\_Next(CircleList\* list); |
|  |

## 2、设计与实现

|  |
| --- |
| 插入元素的分析   1. 普通位置插入元素 2. 添加第一个元素（第一次插入元素） 3. 最后一个位置插入元素 4. 第一个位置插入元素 |
| 在第一个位置插入 |
|  |
|  |
| 删除节点 |
|  |

## 3、优点和缺点

|  |
| --- |
| 优点：功能强了。  循环链表只是在单链表的基础上做了一个加强 |
| 循环链表可以完全取代单链表的使用  循环链表的Next和Current操作可以高效的遍历链表中的所有元素 |
| 缺点：  代码复杂度提高了 |
| 约瑟夫问题-循环链表典型应用  n 个人围成一个圆圈，首先第 1 个人从 1 开始一个人一个人顺时针报数，报到第 m 个人，令其出列。然后再从下一 个人开始从 1 顺时针报数，报到第 m 个人，再令其出列，…，如此下去，求出列顺序。 |
|  |

# 双向链表

## 1、基本概念

|  |
| --- |
| 单链表的结点都只有一个指向下一个结点的指针  单链表的数据元素无法直接访问其前驱元素  逆序访问单链表中的元素是极其耗时的操作！ |
| len = LinkList\_Length(list);  for (i=len-1; len>=0; i++) //O(n)  {  LinkListNode \*p = LinkList\_Get(list, i); //O(n)  //访问数据元素p中的元素  //  } |
| 双向链表的定义  在单链表的结点中增加一个指向其前驱的pre指针 |
|  |
| 双向链表拥有单链表的所有操作  创建链表  销毁链表  获取链表长度  清空链表  获取第pos个元素操作  插入元素到位置pos  删除位置pos处的元素 |

## 2、设计与实现

|  |
| --- |
| 插入操作 |
|  |
| 删除操作 |
|  |
| 双向链表的新操作 |
| 获取当前游标指向的数据元素  将游标重置指向链表中的第一个数据元素  将游标移动指向到链表中的下一个数据元素  将游标移动指向到链表中的上一个数据元素  直接指定删除链表中的某个数据元素 |
| DLinkListNode\* DLinkList\_DeleteNode(DLinkList\* list, DLinkListNode\* node);  DLinkListNode\* DLinkList\_Reset(DLinkList\* list);  DLinkListNode\* DLinkList\_Current(DLinkList\* list);  DLinkListNode\* DLinkList\_Next(DLinkList\* list);  DLinkListNode\* DLinkList\_Pre(DLinkList\* list); |
| //大家一定要注意：教科书不会告诉你 项目上如何用；哪些点是项目的重点；  做一个企业级的财富库，完成你人生开发经验的积累，是我们的学习重点，要注意！ |

## 3、优点和缺点

|  |
| --- |
| 优点：双向链表在单链表的基础上增加了指向前驱的指针  功能上双向链表可以完全取代单链表的使用  循环链表的Next，Pre和Current操作可以高效的遍历链表中的所有元素 |
| 缺点：代码复杂 |