数 据 结 构

p=p->link;

第二章 线性表 [3] 循环链表、双向链表

任课老师：郭艳

数据结构课程组

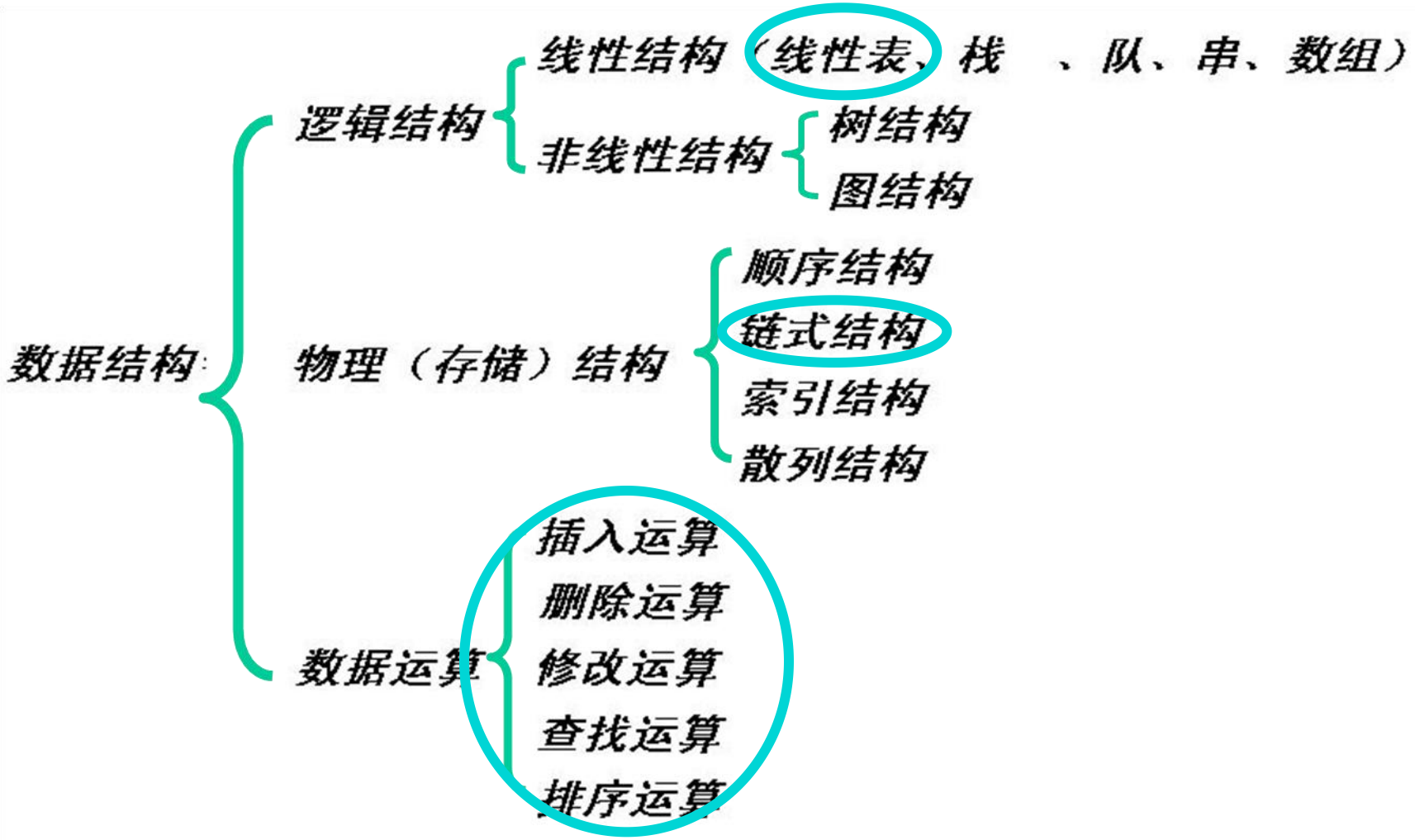
计算机学院

中国地质大学（武汉）2020年秋

上堂课要点回顾

- **ADT LinearList的实现方案2——单链表**
  - 链式存储结构的特点和单链表模型
  - 头指针、首元结点、头结点
  - 单链表（带头结点）类定义和实现（“LinkedList.h”）
    - **struct LinkNode<T>类的定义及实现**
      - 数据成员
      - 函数成员
    - **class List<T>/LinkedList<T>类的定义及实现**
      - 数据成员
      - 函数成员
        - void List()、int length()、void makeEmpty()/~List()、LinkNode * Locate(int)、bool Insert(int,T&)
        - 性能（等概率情况下）
    - 应用
      - 有序表的归并、有序表的去冗余、用有序链表实现集合ADT
- **链式存储结构的优缺点**

本堂数据结构课内容


数据结构
逻辑结构
线性结构（线性表、栈 、队、串、数组）
非线性结构
树结构
图结构
物理（存储）结构
顺序结构
链式结构
索引结构
散列结构
数据运算
插入运算
删除运算
修改运算
查找运算
排序运算

第 三 次 课

阅读：

殷人昆，第66-83页

练习：

Project1

例 有序单链表的插入

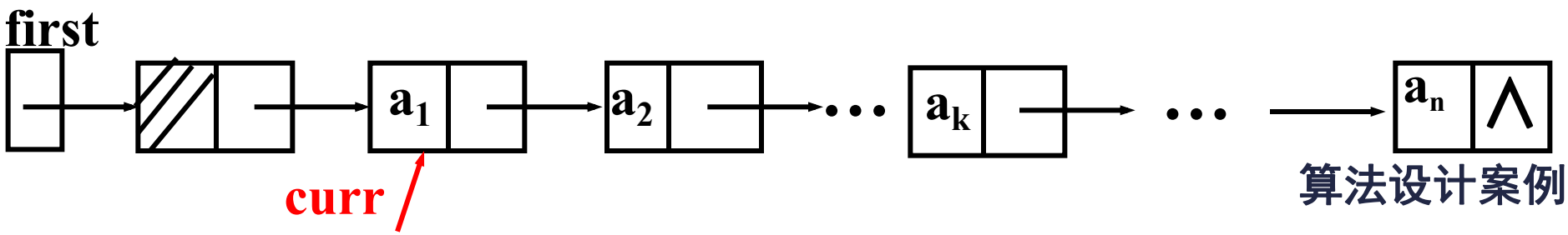
- 问题描述：

设OrderedLinkedList是有序单链表，编写算法将数据元素x插入，要求插入x后的单链表元素按值从小到大排列。

例：OrderedLinkedList=<4,8,10>

插入7后，则

OrderedLinkedList=<4,7,8,10>



- 输入：x,first(有序)
- 输出：first(有序)
- 思想：从头指针开始顺着指针链向后查找插入位置
- 具体：设搜索指针curr，curr初始指向首元结点。

循环：curr=curr->link;
直至curr->data>x 或 curr==NULL(即curr移至表尾)

循环退出后curr所指结点之前即为新结点的插入位置，但是问题是因为不能获得curr结点的直接前驱结点而不能插入。

解决方案：为实现插入，增设缓存指针pre。pre始终指向curr的直接前驱结点。

具体步骤变化如下：

- 设搜索指针curr和pre指针，初始curr指向首元结点，pre初始指向头结点。
- 循环：pre=curr; curr=curr->link;
  直至curr->data>x 或 curr==NULL(即curr移至表尾)
- 循环退出后pre结点之后（即curr结点之前）即为新结点的插入位置。
- 新建结点q，其数据域置为x。
- 将q结点插入在pre结点之后：
  q->link=curr;
  pre->link=q;

- 时间复杂度：O(n)
- 空间复杂度：O(1)

【有序单链表的有序插入算法】

经过几次扫描完成功能？
其它高效的方法？

```
template <class T>
void List<T>::OrderInsert(const T& x)   //数据元素x有序插入
{  LinkNode<T> *curr, *pre;
   //初始化
        curr = first->link;              //curr指向首元结点
        pre = first;                     //pre指向头结点
   //定位插入位置
        while(curr != NULL && curr->data <= x)
        {       pre = curr;
                curr = curr->link;
        }
  //申请一个结点并赋值
        LinkNode<T> *newNode = new LinkNode<T>(x, pre->link);
        pre->link = newNode;      //把新结点插入pre所指结点后
}    //时间复杂度O(n)
```

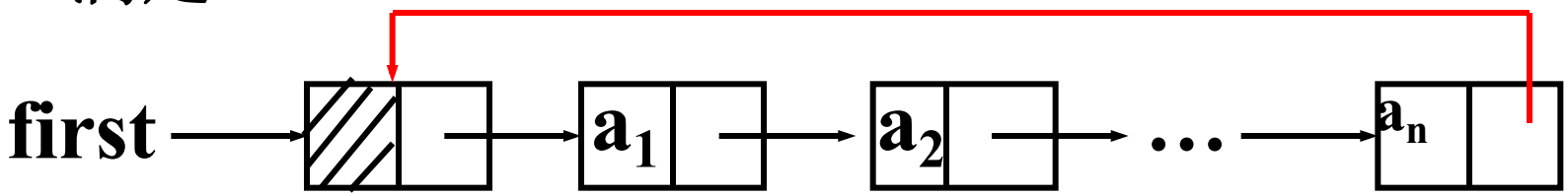
2.4.1 循环单链表（简称循环链表）

- 定义：在单链表的尾结点的链域中放入指向头结点的指针，使整个链表形成一个环形。

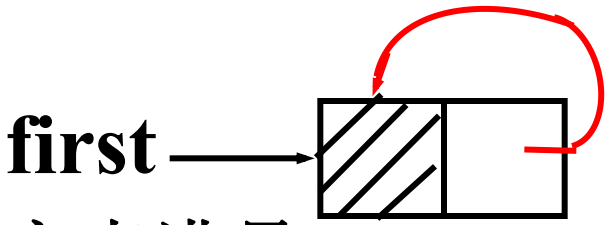
- 循环单链表（带头结点）模型：

循环单链表中有空指针吗？为什么？

非空表：满足first!=first->link



空表：



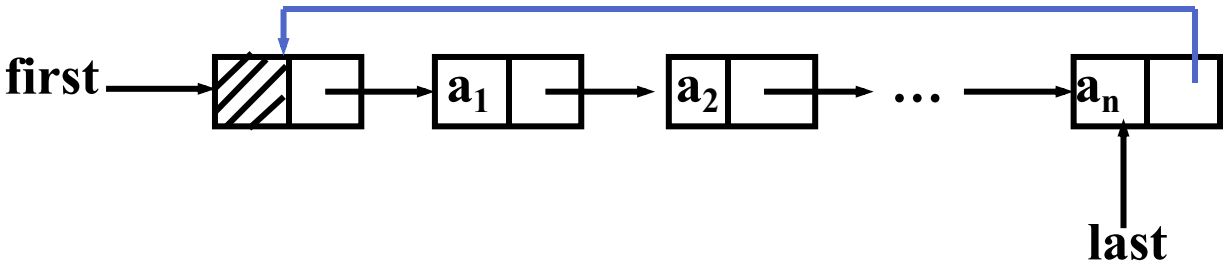
空表满足first==first->link

◆ 优点：

不增加额外开销，却给不少操作带来方便，从循环表中任一结点出发都能访问到表中其它结点。

能否优化循环单链表模型？如何将两个已知链表快速合并在一起？时间复杂度是O（？）

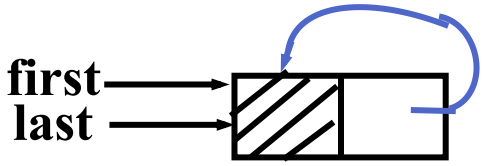
循环链表的运算与单链表的基本一致，差别仅在于算法中的循环条件，不是current或current->link是否为空，而是它们是否等于*first*。



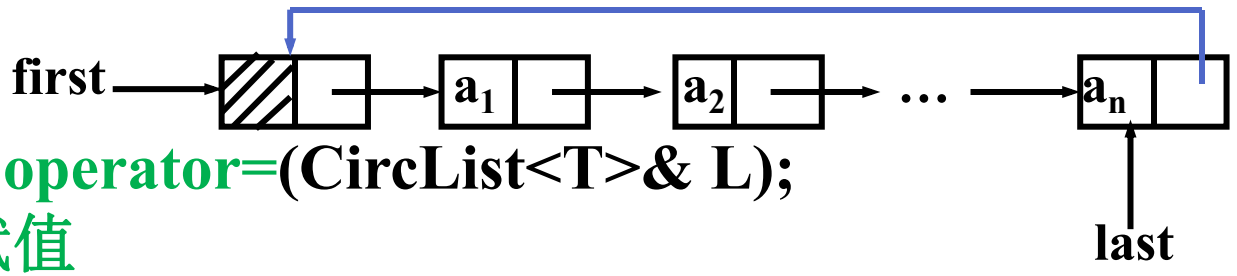
循环链表的类定义CircList（P67 程序 2.22）

struct CircLinkNode 是否可以用P60 程序2.17中的struct ListNode类替代？为什么？

```
#include"LinearList"
#include"LinkedList"
template <class T>
class CircList: public LinearList<T> {//循环链表类定义
protected:
        LinkNode<T> *first, *last;       //表头、尾指针
public:
        CircList(){}; //构造函数1
        CircList(const T& x){};       //构造函数2
        CircList( CircList<T>& L);//复制构造函数
        ~CircList();              //析构函数
        bool IsEmpty() const                          //判表空否
        { return first->link == first ? true : false; }
        bool IsFull() const {return false;}   //判表满否
```



```
int Length() const;          //计算链表的长度
void makeEmpty();            //将链表置为空表
LinkNode<T> *getHead() const {return first};
                                        //返回附加头结点地址
void setHead(LinkNode<T> *p) {first = p;}
                                        //设置附加头结点指针
LinkNode<T> *Locate(int i) const;//定位第i个元素的地址
LinkNode<T> *Search(T x);            //搜索含x元素
bool getData(int i,T& x) const;      //取出第i元素值
void setData(int i, T& x);           //更新第i元素值
bool Insert (int i, T& x);           //在第i元素后插入
bool Remove(int i, T& x);            //删除第i个元素
//void Sort();                       //排序
void input();
void output();
CircList<T>& operator=(CircList<T>& L);
//重载函数：赋值
};
```



思考：

已知链表的一个结点p，如何以O（1）效率求p结点的直接前驱结点？

2.4.2 双向链表

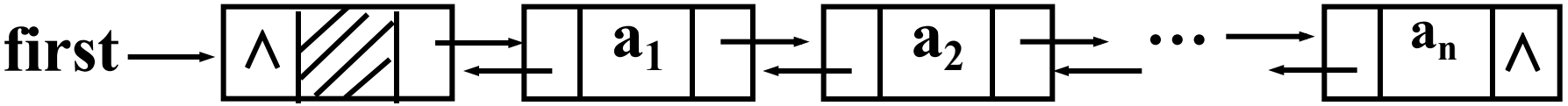
◆ 结点结构

弥补单链表link域仅指向后继结点，不能有效找到前驱的不足，增加一个指向直接前驱的指针。

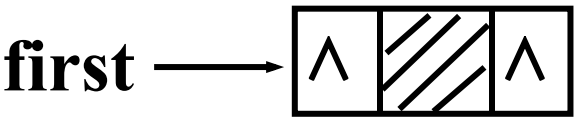
1973年Donald Ervin Knuth首创双向链表

◆ 双向链表（带头结点）模型

非空表：　满足 first->lLink==NULL &&
first->rLink!=NULL



空表：

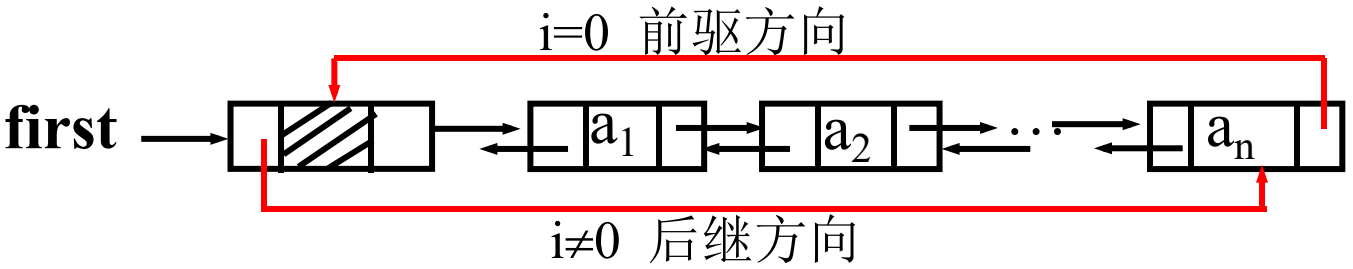


满足 first->lLink = =first->rLink = =NULL

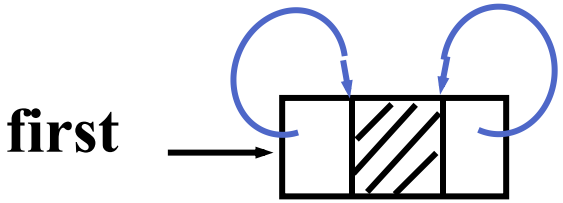
◆ 双向循环链表（带头结点）模型

非空表

满足 **first->lLink ! = first &&**
first->rLink ! = first

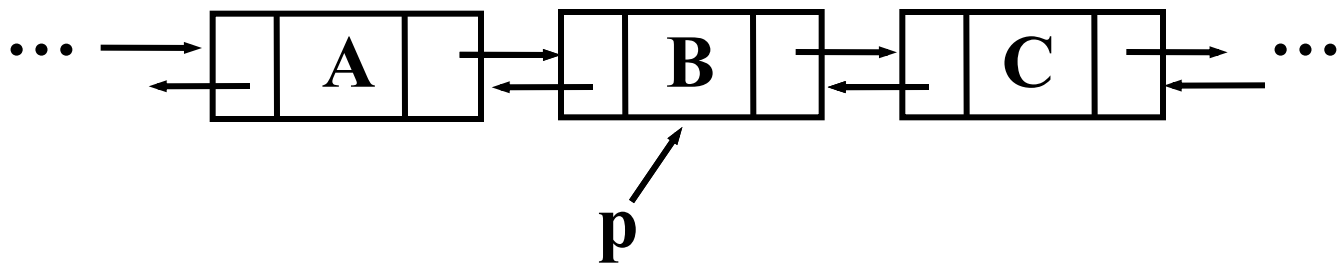


空表



满足 **first = = first->lLink = = first->rLink**

◆ 双向循环链表中结点的一个重要特征



设p指向双向循环链表的任何一个结点，显然，

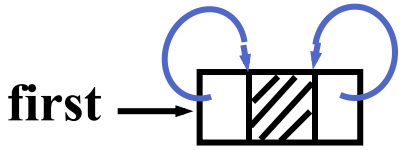
$$p\rightarrow lLink\rightarrow rlink == p\rightarrow rLink\rightarrow lLink == p$$

双向循环链表（带头结点）类的定义

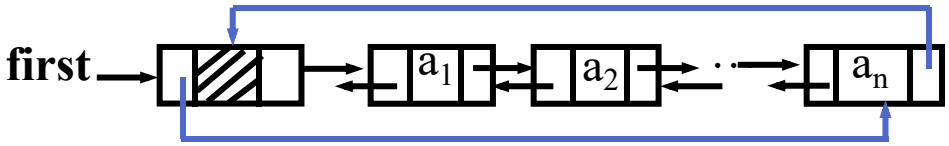
//DBLinkedList.h文件实现双向循环链表类
//P70 程序2.24

```
template <class T>
struct DblNode {              //双向链表结点类定义
 T data;                      //链表结点数据
 DblNode<T> *lLink, *rLink;   //前驱、后继指针
 DblNode(DblNode<T> *l = NULL, DblNode<T> *r = NULL)
 {    lLink = l;  rLink = r;   }        //构造函数1
 DblNode ( T value, DblNode<T> *l = NULL, DblNode<T> *r = NULL)
 {    data = value;  lLink = l;  rLink = r; }  //构造函数2
};
```

```
template <class T>
class DblList: public LinearList<T> {//双向循环链表类定义
private:
    DblNode<T> *first;                    //表头指针
public:
  DblList ( T uniqueVal ) {          //构造函数
        first = new DblNode<T> (uniqueVal);
        if (first==NULL) {cerr<< "存储分配出错!"<<endl;exit(1);}
        first->rLink = first->lLink = first;//双向循环链表构造函数
   };
   ~DblList();
   DblNode<T> *getHead() const { return first; }
   void setHead( DblNode<T> *ptr ) { first = ptr; }
   bool IsEmpty() { return first->rlink == first; }//判双链表空否
   int Length() const;                  //计算链表的长度
   void makeEmpty();                    //将链表置为空表
```



```
DblNode<T> *Locate ( int i, int d );
  /*在链表中定位序号为i(≥0)的结点， d=0按前驱方向
    ;d≠0按后继方向*/
DblNode<T> *Search (const T& x, int d);
   /*在链表中按d指示方向寻找等于给定值x的结点, d=0按前
     驱方向, d≠0按后继方向*/
bool Insert ( int i, const T& x, int d );
  /*在第i个结点后插入一个包含有值x的新结点, d=0按前驱
    方向, d≠0按后继方向*/
bool Remove ( int i, T& x, int d );   //删除第i个结点
bool getData(int i,T&x)const;
bool setData(int i,T &x)const;
void input();
void output();
void sort();
DblList<T> & operate=(DblList<T> &L);
};
```



◆ 双向循环链表求表长运算实现（自学）

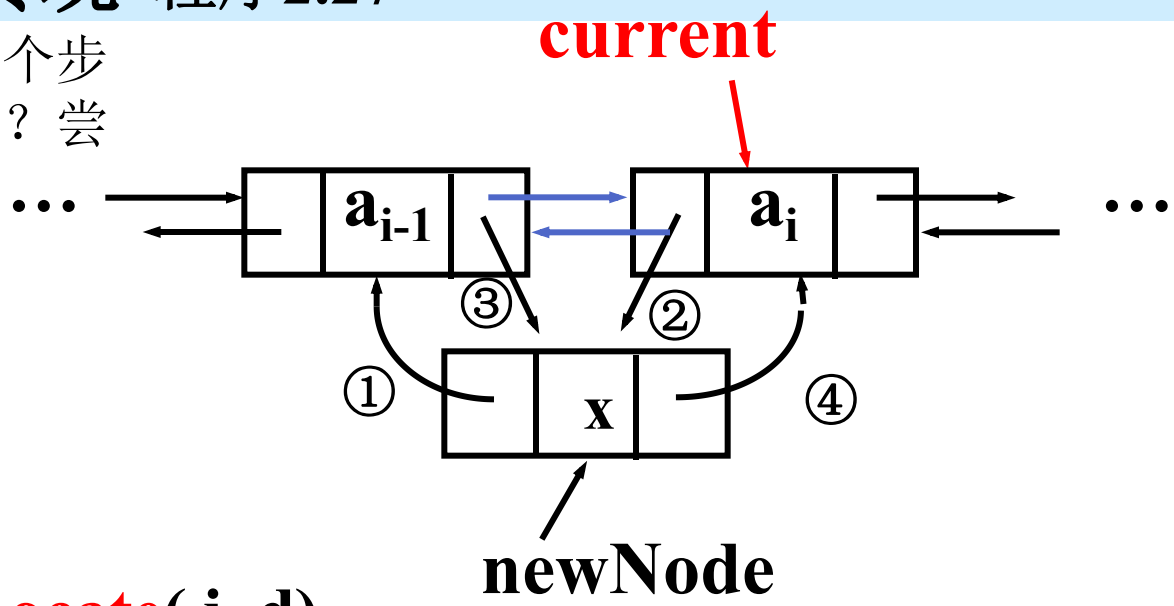
```
// P 71 程序2.24(4)
template <class T>int DblList<T>::Length()const
{//计算带附加头结点的双向循环链表长度
        DblNode<T> * cur = first ->rlink;int count = 0;
        while (cur != first)
        {       cur = cur->rlink;          count++;    }
        return count;
}//时间复杂度：O(？)
```

◆ 定位运算实现 P 71 程序2.26

```
template <class T>
DblNode<T> *DblList<T>::Locate(int i, int d)
/*定位第i个结点的地址，d=0按前驱方向，否则按后继方向，操作失败则返回NULL*/
{   if(i<0) return NULL;
        if (first->rLink==first || i==0) return first;//空双向循环链表或i为0
        DblNode<T> *current;//遍历指针
        if (d==0) current=first->lLink;//前驱方向，遍历指针初始为尾结点
        else current=first->rLink;//后继方向，遍历指针初始为首元结点
        for (int j=1;j<i;j++)
                if (current==first) break;     //遍历结束
                else if (d==0) current=current->lLink;  //按前驱方向一步
                else current=current->rLink;  //按后继方向一步
        if (current!=first) return current;
        else return NULL;                  //i值大于表长，非法
}//O(n)
```

◆ 在前驱方向（d==0）在第i个结点之后插入一个新结点的实现 程序2.27

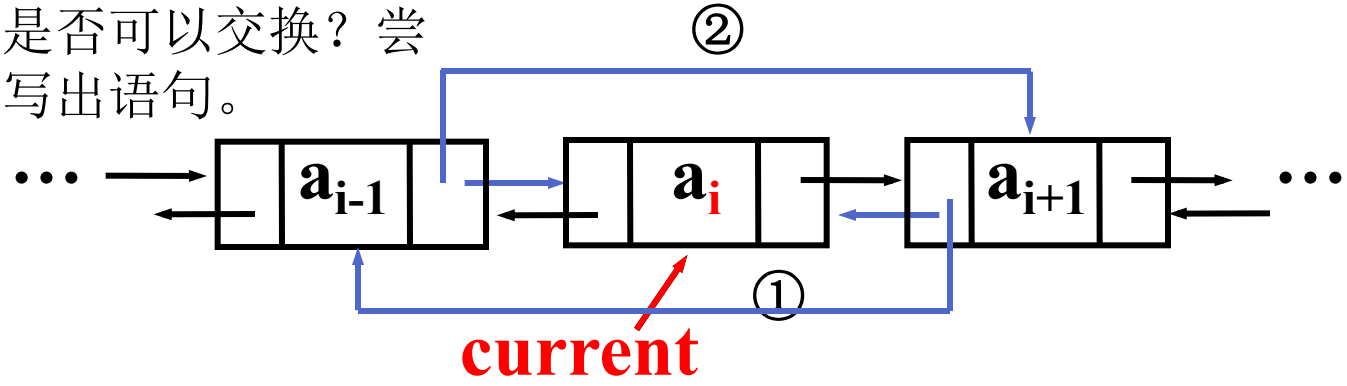
修改指针域的四个步骤是否可以交换？尝试写出语句。




```
current = Locate( i, d)
DblNode<T> *newNode=new DblNode<T>(x);
newNode->lLink=current->lLink;
current->lLink = newNode;
newNode->lLink->rlink = newNode;
newNode->rLink=current;
```

◆ 删除一个结点的实现 程序2.28

修改指针域的两个步骤是否可以交换？尝试写出语句。




```
current = Locate( i, d)
current→rLink→lLink=current→lLink;
current→lLink→rLink = current→rLink;
 x = current->data;
delete current;
```

● 链式存储结构的优缺点

链表会表满？如何判断表满？

Insert(LinkNode *p,T &x) O(？)

Remove(LinkNode *p,T &x) O(?)

◆ 优点

（1）无需事先了解线性表的长度。结点空间动态地申请和释放，允许线性表的长度有很大的变化。

（2）插入/删除操作无需大量移动元素，只要修改一、两处指针；能够适应经常插入/删除元素的情况。

（3）便于将两个表合并在一起，或者把一个表拆分成两个独立的表。此外，链表可以交叠，便于共享公共部分。

◆ 缺点

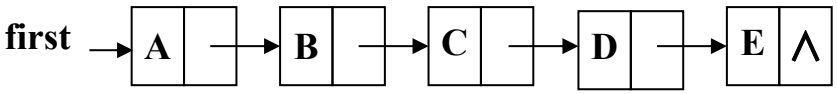
（1）链表的操作算法较复杂

（2）指针域需额外占用空间

（3）非随机存储结构，需从头指针扫描查找结点。如果大部分应用需要顺序扫描表，而不是随机访问各项，可以使用链式存储。如果需要表中间或底部的项，可以使用指针变量指向适当位置。

2.6 静态链表（自学）

在一维数组中增加指针域用来存放下一个数据元素在数组中的下标，从而构成用数组描述的链表。因为数组内存空间的申请方式是静态的，所以称为静态链表，增加的指针称做仿真指针。便于在没有指针类型的程序设计语言中使用链表。静态链表结构如下：



(a) 常规链表

0	A	1
1	B	2
2	C	3
3	D	4
4	E	-1
⋮		
maxSize–1		

(b) 静态链表1

0	A	2
1	E	-1
2	B	4
3	D	1
4	C	3
⋮		
maxSize–1		

(C) 静态链表2

返回

静态链表模型

空闲单元
数据单元

maxSize=10

0		-1
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		9
9		-1

空表

avil==1

0		1
1	A	2
2	B	3
3	C	4
4	D	5
5	E	-1
6		7
7		8
8		9
9		-1

非空非满表

avil==6

0		1
1	A	2
2	B	3
3	C	4
4	D	5
5	E	6
6	F	7
7	G	8
8	H	9
9	I	-1

表满

~~avil==maxSize~~

avil==-1

0		2
1	A	-1
2	B	3
3	C	4
4	D	5
5	E	6
6	F	7
7	G	8
8	H	9
9	I	-1

删除 **A**

avil==1

0		2
1	A	-1
2	B	3
3	C	4
4	D	6
5	E	1
6	F	7
7	G	8
8	H	9
9	I	-1

删除 **E**

avil==5

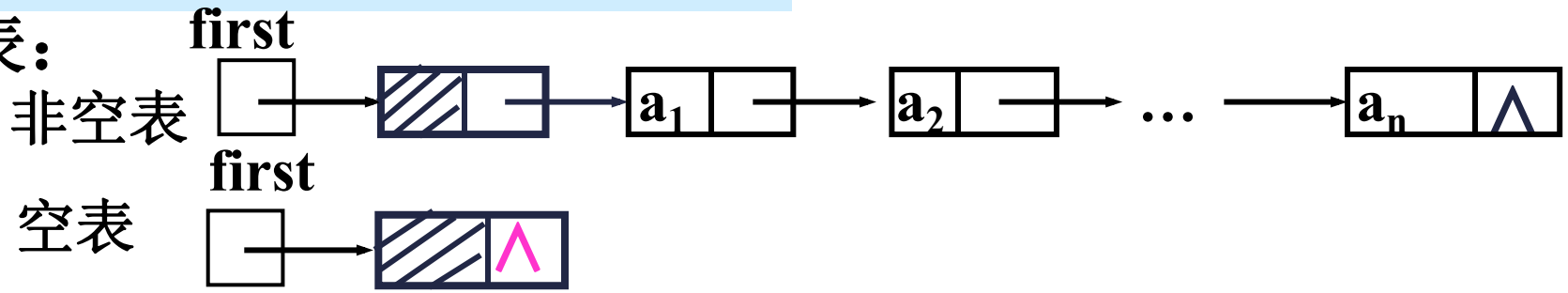
2.5 单链表的应用：

——多项式及其运算（P73-80）

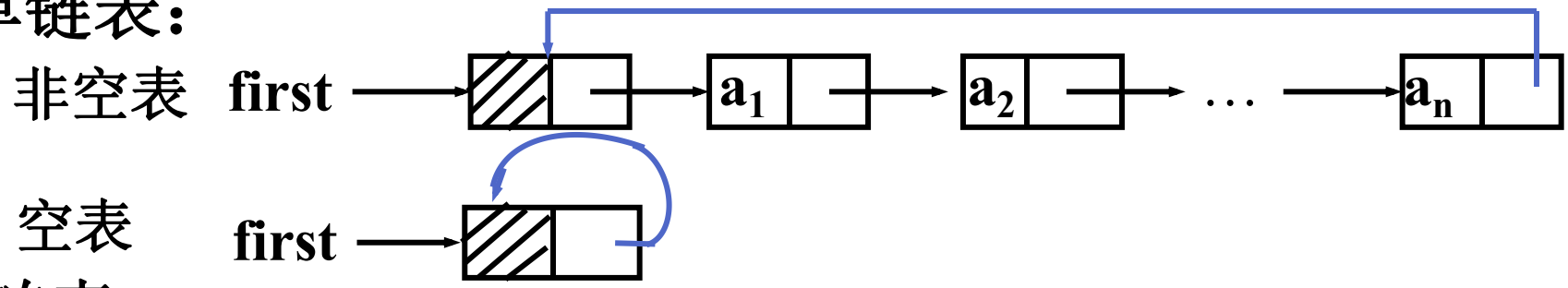
略 自学

几种主要链表比较

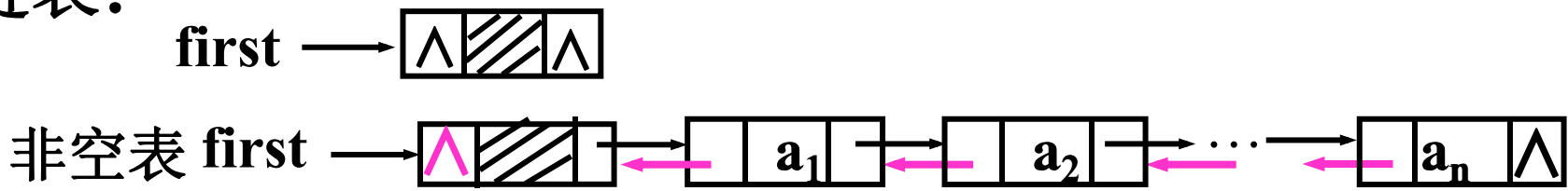
单链表：



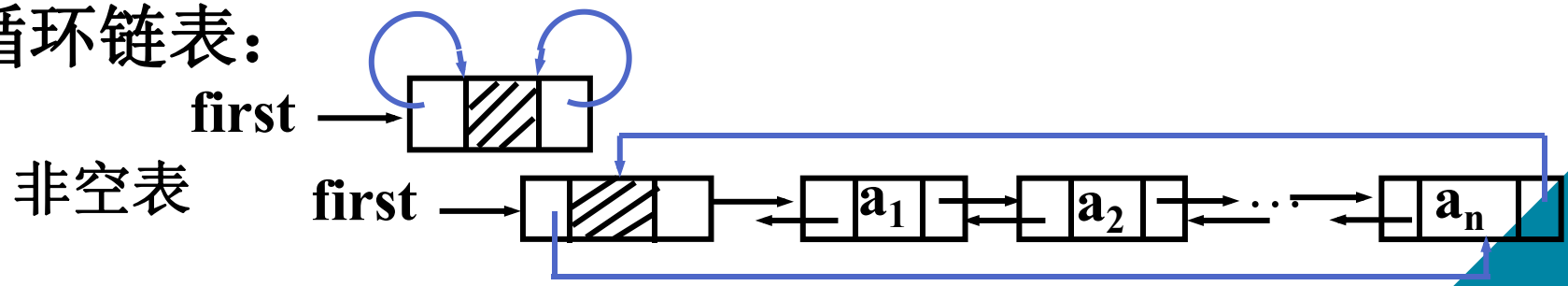
循环单链表：



双向链表：



双向循环链表：



应用场合的选择

- 不宜使用顺序表的场合
  - 线性表的最大长度不确定时
  - 经常插入/删除时
- 不宜使用链表的场合
  - 当读/写操作比插入/删除操作频率大时
  - 当指针存储开销和整个结点内容所占空间相比，其比例较大时

自 学

P68 程序2.23 Josephus()
P70 程序2.24(3,4) DblList()、Length()
P71 程序2.25 Search()
P71 程序2.26 Locate()
P72 程序2.27 Insert()
P71 程序2.28 Remove()
P80-83 2.6章节 静态链表

作业3-2

算法设计（选作）

（1）实现循环单链表（继承LinkList类）CircLinkedList.h头文件以及测试程序源文件。
（2）实现双向循环链表（继承LinkList类）DblLinkedList.h头文件以及测试程序源文件。

JesephRing问题

约瑟夫问题是个有名的问题，变化形式也很多。

1、据说著名犹太历史学家 Josephus有过以下的故事：在罗马人占领乔塔帕特后，39 个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。然而 Josephus 和他的朋友并不想遵从。首先从一个人开始，越过k-2个人（因为第一个人已经被越过），并杀掉第k个人。接着，再越过k-1个人，并杀掉第k个人。这个过程沿着圆圈一直进行，直到最终只剩下一个人留下，这个人就可以继续活着。问题是，给定了和，一开始要站在什么地方才能避免被处决？Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏。

2、17世纪的法国数学家加斯帕在《数目的游戏问题》中讲了这样一个故事：15个教徒和15 个非教徒在深海上遇险，必须将一半的人投入海中，其余的人才能幸免于难，于是想了一个办法：30个人围成一圆圈，从第一个人开始依次报数，每数到第九个人就将他扔入大海，如此循环进行直到仅余15个人为止。问怎样排法，才能使每次投入大海的都是非教徒。

3、有 n 只猴子，按顺时针方向围成一圈选大王（编号从 1 到 n），从第 1 号开始报数，一直数到 m，数到 m 的猴子退出圈外，剩下的猴子再接着从1开始报数。就这样，直到圈内只剩下一只猴子时，这个猴子就是猴王，编程求输入 n，m 后，输出最后猴王的编号。

约瑟夫和朋友站在什么位置才保住了性命呢，这就是我们今天要讲的约瑟夫环问题，又称“丢手绢问题”。

约瑟夫问题并不难，求解的方法很多：（1）模拟法（循环遍历法）；（2)递归法（公式法）。

P68 JesephRing问题模拟

问题的分析与数据结构的设计**案例**

【问题描述】

已知n个人（编号分别为1,2,...,n）按顺时针方向围坐一圈，每人持有一个正整数密码。开始时任选一个报数上限值m，从编号为1的人按顺时针方向自1开始报数，报到m时停止报数，报m的那个人出圈；将他的密码作为新的m值，从他在顺时针方向的下一个人开始重新从1报数，数到m的那个人又出列；依此规律重复下去，直到所有人全部出列。输入正整数n、m和n个正整数密码值，设计程序来输出满足游戏规则的出圈序列。

【例如】n = 7, 7个人的密码分别为3,1,7,2,4,8,4。初始报数上限值m=20。

【解答】出圈人的顺序为6, 1, 4, 7, 2, 3, 5

数据特性与操作特性的分析

- 数据分析
  - n代表总人数，m是初始密码
  - 数据元素：$a_i=(n_i, m_i)$，其中i=1,…,n (n≥1)，n_i、m_i代表第i个人的编号和持有的密码，其中n_i不能省略
  - 数据元素间的关系：对a_i (1≤i≤n-1)存在后继次序关系$<a_i, a_{i+1}>$；a_n的直接后继是a_1
- 操作分析
  - 建空约瑟夫环
  - 通过循环在约瑟夫环的尾部插入新的数据元素，从而建立约瑟夫环（频繁）
  - 在约瑟夫环中从某数据元素开始遍历若干步
  - 删除约瑟夫环中某数据元素的下一个数据元素（频繁）

问题的抽象数据类型定义

ADT JesephRing

数据元素：$a_i=(n_i, m_i)$，i=1,2,…,n (n≥1)，n_i，m_i代表第i个人的编号和持有的密码，其中n_i不能省略。

结构：是线性结构，对数据元素a_i (1≤i≤n-1)存在次序关系 $<a_i, a_{i+1}>$, a_n的直接后继是a_1。

逻辑操作：设J为 JesephRing型

JesephRingInitiate(J) //构造一个空的约瑟夫环J。

JesephRingInsertEnd(J, x) /*在约瑟夫环J尾部插入新元素x，成功返回1，失败返回0。*/

JesephRing(J,m) /*给定初始报数上限m，从1开始循环报数、删除、输出直至J为空，得到出列序列。*/

存储结构分析

目的：有效地组织和处理数据

- 分析逻辑操作特点
  - n值是个变量
  - 频繁插入和删除
  - 指针存储开销和整个结点内容所占空间相比，其比例较小

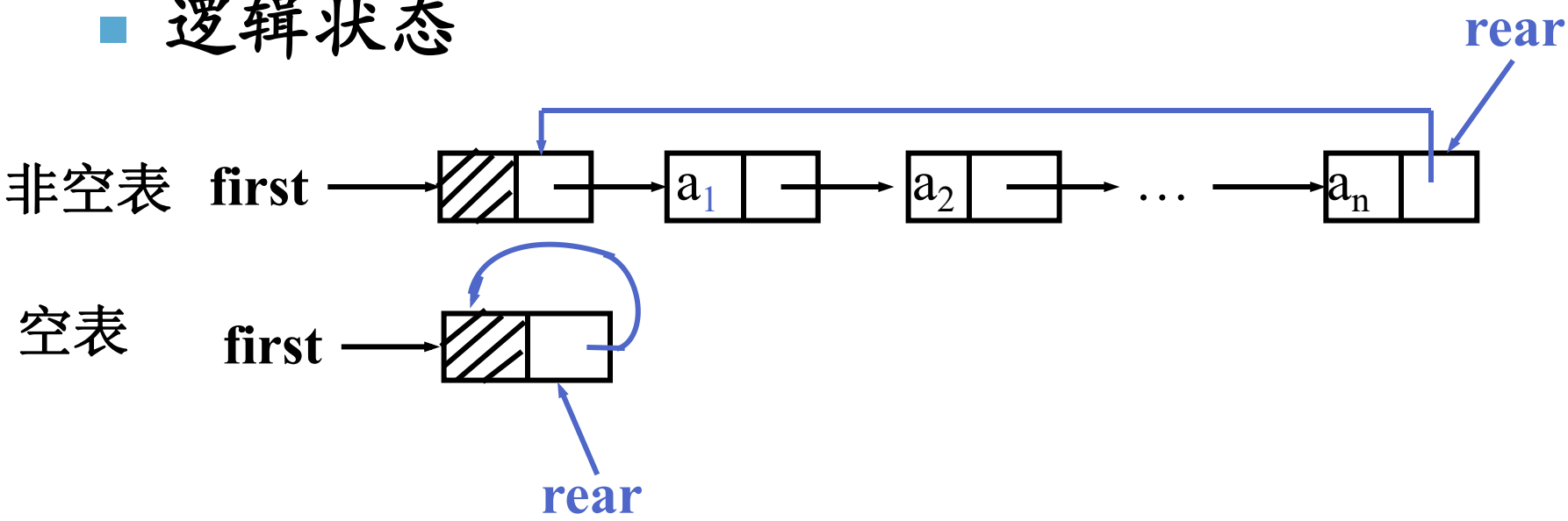
综上所述，不采用顺序表，而采用链表

  - 经常在尾部插入，因此增设尾指针，始终指向尾部结点
  - a_n的直继是a_1，因此采用循环单链表

综上所述，采用带尾指针的循环单链表存储结构

带尾指针的循环单链表 LinListCWithRear类

- 逻辑状态



- 分析逻辑操作
  - 初始化LinListCWithRear()
  - 在表尾插入新结点InsertEnd(x)
  - 删除p结点的下一个结点DeleteAfter(p)
  - 判断是否是空表IsEmpty()
  - 撤销~ LinListCWithRear()

返回

带尾指针的循环单链表实现

" LinListCWithRear.h"

```
//单链表类头文件LinkedList.h
template<class T>struct LinkNode{
   T data;   LinkNode<T> * link;
   LinkNode(LinkNode<T>*ptr= NULL)...;
   LinkNode(const T& item, LinkNode<T> * ptr = NULL)...;
};
template<class T>class List
{public: List(); List(const T& x);
  List(List<T>& L);   ~List();
  void makeEmpty();
  int Length()const;
  ......
protected:
  LinkNode<T> * first;
}
```

```
template <class T>
class LinListCWithRear:public
   List<T>
{private:
   LinkNode<T> * rear;      //尾指针
 public:
    LinListCWithRear();
   ~ LinListCWithRear();
   int IsEmpty(void);
   void InsertEnd(const T& x);
   T DeleteAfter(LinkNode<T>
   *p);
   void JesephRing(void);
}; /*带尾指针的循环单链表类定义*/
```

```
template <class T>
void LinListCWithRear :: LinListCWithRear ()
//初始化带尾指针的循环单链表
{
    first->link=first;          //循环结构
    rear=first;                 //尾指针处理
}
```

```
template <class T>
LinListCWithRear<T>::~LinListCWithRear(void)
//析构函数，释放链表所有结点（包括头结点）
{  LinkNode<T> *p, *q;/*q指向被删结点，缓冲指针p指向q结点的直接后继结点*/
   p = first;  //p指向头结点
   while(first->link != first)//循环释放结点空间直至没有结点
   {    q = p;          p = p->link;          delete q;     }
   first= NULL;              //头指针值为NULL
   rear=NULL;
}
```

```
template <class T>
int LinListCWithRear :: IsEmpty(void)
{  if(first->link==first)
        return 1;
   else
        return 0;
}
```

```
template <class T>
void LinListCWithRear :: InsertEnd (const T& x)
/*在带尾指针循环单链表的表尾结点后插入一个新元素x。*/
{  LinkNode<T> *q=new LinkNode<T>(x); /*新结点*/
   q->link=rear->link; /*插入*/
   rear->link=q;
   rear=q;   //尾指针处理
}
//时间复杂度O(1)
```

```
template <class T>
T LinListCWithRear :: DeleteAfter(LinkNode<T>  *p)
/*删除并释放带尾指针的循环单链表L中p结点的下一个结点*/
{  LinkNode<T> *s; T t;
   if(IsEmpty()) exit(0);
   s=p->link;
   p->link=p->link->link;          //删除p结点的下一个结点
   if(s->link==first)
        rear=p;        //尾指针处理
   t=s->data;
   delete s;
   return t;
}   //时间复杂度O(1)
```

JesephRing( )基本算法：

- 1、根据初始报数上限m值，寻找第m个结点（应该找到第m-1个结点的地址才便于删除，因此为便于删除，定义curr指针找第m个结点，pre指针始终指向curr结点的直接前驱结点）
- 2、输出第m结点的number值
- 3、把该结点的cipher值赋给m，作为下一次循环的报数上限m
- 4、删除第m个结点

如此循环n次或直至表为空时结束。（实际只需循环n-1次，最后只剩一个结点时，直接输出即可）

```
void JesephRing( )
{//给定初始报数上限m，从1开始循环报数、删出、输出直至J为空
    LinkNode<T>  *pre,*curr; int i;
    pre=first;   curr=first->link;
    while(IsEmpty( )==0)
    {     for (i=1;i<m;i++)
          {        pre=curr;      curr=curr->link;
                   if(curr==first)   //跳过头结点
                   {   pre=curr; curr=curr->link;      }
          } //end of for
          cout<<curr->data.number<<" ";
          m=curr->data.cipher;          curr=curr->link;
          if(curr==first)    curr=curr->link;  //跳过头结点
          DeleteAfter(pre);
    }
}//时间复杂度O( $\sum_{i=0}^{n} m_i \; 0 \le i \le n$ )，其中n为人数，$m_i$是第i个人的密码值
```

JesephRing.cpp

```
#include <iostream>
#include <stdio.h>
typedef struct
{  int number;
   int cipher;
}DataType;
#include "LinListCWithRear.h”
```

```
void main()
{
    int n=7,m=20, i;
    DataType test[7]={{1,3},{2,1},{3,7},{4,2},{5,4},
    {6,8},{7,4}};
    LinListCWithRear<DataType> JR(m);

    for(i=0;i<n;i++)
        JR. InsertEnd(test[i]);
    JesephRing( );
}
```

本堂课要点总结

- 单链表
  - 应用：有序线性表的有序插入
  - 案例：算法设计与描述（思想、步骤、效率分析）
- 循环单链表的定义和实现
- 双向循环链表的定义和实现
- 静态链表
- 应用场合的选择
- 线性表的应用
  - 应用：约瑟夫环问题的模拟
    - 问题需求分析→抽象出数据元素、数据元素间的关系和操作
    - 存储结构的设计
    - 算法的设计
    - 实现
  - 案例：问题的分析与数据结构的设计

Project1：第一次上机实习题目
——实现任意长度整数抽象数据类型

【问题描述】设计一个程序实现两个任意长的整数求和运算。

【基本要求】

（1）输入和输出格式要求：十进制数；要求用户输入数据时，给出清晰明确的提示信息，包括输入的数据内容、格式及结束方式等；输入支持使用或者不使用千位分隔符（每三位一组，组间用逗号隔开）；输出格式要求使用千位分隔符。

（2）设计方案尽可能高效：运算快、所耗内存小。

【实现提示】

（1）题目的难点是存储任意长度的整数。题目要求能够以100%的精度表示任意长度的整数，但在C++语言的类型中int类型所用的存储量是4个字节，因此int类型数据最多只能精确表示二十几位的整数。因此处理任意长度的整数需要设计合适的存储结构。

（2）存储结构决定了任意长度整数所采用的表示形式。由于数的输入/出格式是十进制数，因此需要频繁地在十进制数与所采用的表示形式之间转换。设计存储结构时需要考虑能够高效地进行这种转换。

任意长度的整数： $x = \pm a_{n-1}.....a_i.....a_2 a_1 a_0 = \pm \sum_{i=0}^{n-1} a_i \times 10^i$

ADT arbitrary_length_integer

数据元素：a_i=？

结构：？

逻辑操作：

1、构造一个空的任意长度整数

2、在任意长度整数中插入一个新的数据元素

3、在任意长度整数中删除 一个数据元素

4、任意长度整数的 加法1：z=x+y（非就地实现）

5、任意长度整数 加法2：x=x+y（就地实现）

6、输出任意长度整数

数据结构实习报告模版

数据结构
上机实习报告

实验题目：约瑟夫环问题求解

班级：_____

姓名：_____

学号：_____

指导老师：_____

完成日期：_____

实习报告内容

1. 问题描述和规格说明/需求分析
2. 设计（目标：有效组织与处理数据）
   - 2.1 抽象数据类型
   - 2.2 存储结构设计
   - 2.3 类图（划分系统中的类、每个类的组成、类间的关系）
   - 2.4 模块结构图及调用关系
   - 2.5 算法设计
3. 软件使用说明
4. 调试分析
5. 测试数据
6. 心得体会