2．1 线性表的类型定义

2．2 线性表的顺序表示与实现 (顺序储存结构)

2．3 线性表的链式表示与实现 (链式储存结构)

2．3．1 线性链表(单链表)

2．3．2 静态链表

2．3．3 循环链表

拉丁方阵问题

魔术师发牌问题

约瑟夫问题

2．3．4 双向链表

双向循环链表之凯撒密码与维吉尼亚密码问题

## 【线性表的类型定义】

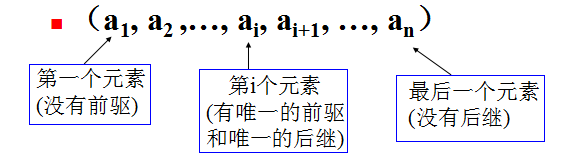
线性表举例:

字母表 (A,B,C,…,X,Y,Z)

数据序列 (6,17,28,50,92,188)

n个元素的线性表:

（a1, a2 ,…, ai, ai+1, …, an）



【抽象数据类型(ADT)】

ADT List{

数据对象：D = {ai | ai属于Elemset, (i=1,2,…,n, n≥0)}

数据关系：R1 = {＜ai-1,ai＞|ai-1,ai属于D,(i=2,3,…,n)}

基本操作：

InitList(&L); DestroyList(&L);

ListInsert(&L,i,e); ListDelete(&L,i,&e);

等等

} ADT List

1. 线性表ADT的基本操作

InitList(&L); DestroyList(&L);

ClearList(&L); ListEmpty(L);

ListLength(L); GetElem(L, i, &e);

LocateElem(L, e, compare());

PriorElem(L, cur\_e, &pre\_e);

NextElem(L, cur\_e, &next\_e);

ListInsert(&L, i, e); ListDelete(&L, i, &e);

ListTraverse(&L, visited())

InitList(&L)

操作结果:构造一个空的线性表L。

DestroyList(&L)

初始条件: 线性表L已经存在。

操作结果: 销毁线性表L。

ClearList(&L)

初始条件: 线性表L已经存在。

操作结果: 将线性表L重置为空表。

ListEmpty(L)

初始条件: 线性表L已经存在。

操作结果: 若线性表L为空表，则返回TURE;否则返回FALSE。

ListLength(L)

初始条件: 线性表L已经存在。

操作结果: 返回线性表L中的数据元素个数。

GetElem(L,i,&e)； 初始条件: 线性表L已经存在，1<=i<= ListLength(L)。

操作结果: 用e返回线性表L中第i个数据元素的值。

LocateElem(L,e,compare())

初始条件: 线性表L已经存在，compare()是数据元素判定函数。

操作结果: 返回L中第1个与e满足compare()的数据元素的位序。若这样的数据元素不存在则返回值为0。

PriorElem(L, cur\_e, &pre\_e)

初始条件: 线性表L已经存在。

操作结果: 若cur\_e是L的数据元素,且不是第一个,则用pre\_e返回它的前驱,否则操作失败; pre\_e无意义。

NextElem(L, cur\_e, &next\_e)

初始条件: 线性表L已经存在。

操作结果:若cur\_e是L的数据元素, 且不是第最后个,则用next\_e返回它的后继,否则操作失 败, next\_e无意义。

ListInsert(&L, i, e)

初始条件: 线性表L已经存在，1<=i<= ListLength(L)+1。

操作结果: 在L的第i个位置之前插入新的数据元素e, L的长度加一。

插入前（长度为n） :

（a1,a2,…, ai-1 , ai,…,an）

插入后（长度为n+1） :

（a1,a2,…, ai-1,e,ai, …,an）

ListDelete(&L,i,&e)

初始条件: 线性表L已经存在，1<=i<= ListLength(L)。

操作结果: 删除L的第i个数据元素,并用e返回其值, L的长度减一。

删除前（长度为n ） :

（ a1,a2,…, ai-1, ai, ai+1, …,an）

删除后（长度为n -1 ） :

（a1,a2,…, ai-1, ai+1, …,an）

ListTraverse(&L, visited())

## 【线性表的顺序表示与实现(物理结构)(顺序储存结构)】

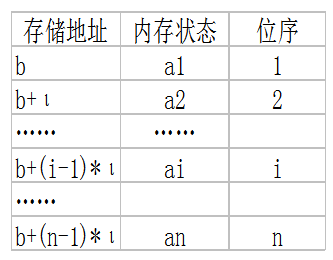
线性表（a1, a2 ,…, ai, ai+1, …, an）的顺序表示----

用一组地址连续的存贮单元依次存贮线性表的数据元素.

Loc(ai+1) = Loc(ai)+ι

Loc(ai) = Loc(a1)+(i-1)\*ι

设ι为每个数据元素所需的存储大小



#define MaxSize 10

typedef int ElemType;

typedef int Status;

struct Sqlist{

ElemType elem[MaxSize];

int length;

};

typedef struct Sqlist \*PtrNode;

typedef PtrNode List;

typedef int status ;

//Initial the list

List InitList()

{

List L;

L = (List)malloc(sizeof(struct Sqlist));

L->length = 0;

printf("Initial List Successfully!\n");

return L;

}

#define OK 1;

typedef int Status;

Status GetElem(Sqlist L,int i,ElemType \*e)

{

if(L.length==0||i<1||i>L.length)

return ERROR;

\*e = L.data[i-1]

return OK;

}

//Insert the list

Status Insert\_Sqlist(List L,int i,ElemType x)

{

int j;

if(i<1||i>L->length+1)

{

printf("OVERFLOW!\n");

return ERROR;

}

for(j=L->length;j>i;j--)

{

L->elem[j+1] = L->elem[j];

}

L->elem[i-1] = x;

L->length++;

return OK;

}

//Print the list

void PrintList(List L)

{

int i=0;

for(i;i<L->length;i++)

printf("%d\n",L->elem[i]);

}

//DestroyList

void DestroyList(List L)

{

if(L->elem)

{

free(L->elem);

printf("DestroyList Successfully\n");

}

}

## 【线性链表（单链表）】

线性表的链式表示------是可以用一组任意的存储单元存储线性表的数据元素。

例： 线性表

（赵，钱，孙，李，周，伍，张，王）

的链式表示

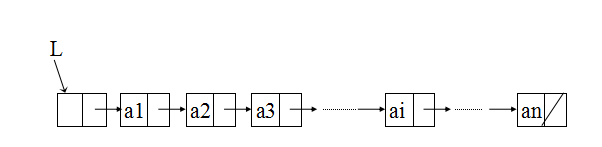
1.链表相关的名称

数据域、指针域

结点、头结点

指针(链)

链表、单链表(线性链表)



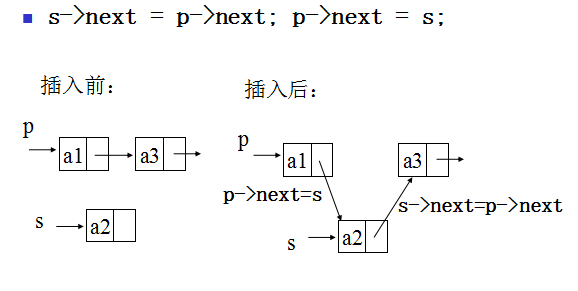
typedef struct Lnode{

ElemType data;

struct Lnode \*next;

}Lnode, \*LinkList;

1. 插入结点



插入结点程序

Status ListInsert\_L(LinkList &L, int i, ElemType e)

{ LinkList s,p;

int j;

p = L; j = 0;

while(p&&j<i-1){p=p->next;++j}

if(!p||j>i-1) return ERROR;

s = (Lnode \*)malloc(sizeof(Lnode));

if(!s) return OVERFLOW;

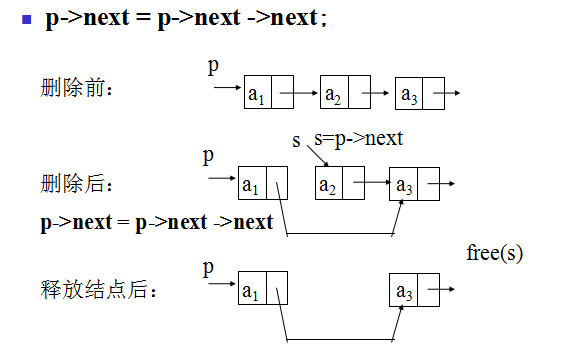
s->data = e;

s->next = p->next; p->next = s;

return OK;

}

1. 删除结点



删除结点程序：

Status ListDlete\_L(LinkList L, int i, ElemType &e)

{ LinkList s,p;

int j;

p = L; j = 0;

while(p->next && j<i-1){p=p->next;++j}

if(!(p->next)||j>i-1) return ERROR;

s = p->next;

p->next = s->next;

e = s->data;

free(s);

return OK;

}

1. 头插法建立链表

Void CreateListHead(LinkList \*L,int n)

{

LinkList p;

Int i;

Srand(time(0));//初始化随机种子

\*L = (LinkList)malloc(sizeof(Node));

(\*L)->next = Null;

For(i=0;i<n;i++)

{

P = (LinkList)malloc(sizeof(Node));

1. >data = rand()%100+1;
2. >next = (\*L)->next;

(\*L)->next=p;

}

}

1. 尾插法建立链表

Void CreateListTail(LinkList \*L,int n)

{

LinkList p,r;

Int i;

Srand(time(0));

\*L = (LinkList)malloc(sizeof(Node));

r = \*L;

For(i=0;i<n;i++)

{

P=(Node \*)malloc(sizeof(Node));

1. >data = rand()%100+1;

r->next=p;

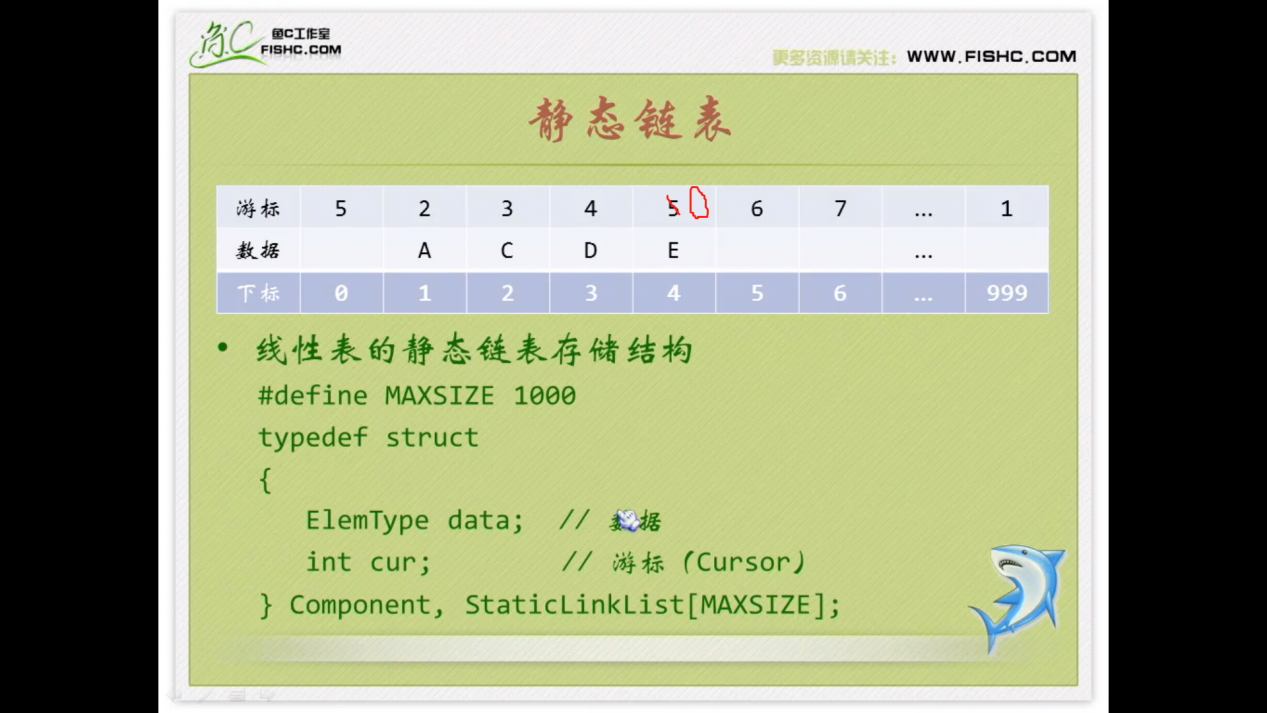
r = p;

}

r->next = NULL;

}

## 【静态链表】



1. 初始化静态链表

//对静态链表初始化等价于对数组的初始化

Status InitList(StaticLinkList space)

{

int i;

for(i=0;i<MAXSIZE-1;i++)

{

space[i].cur = i +1;

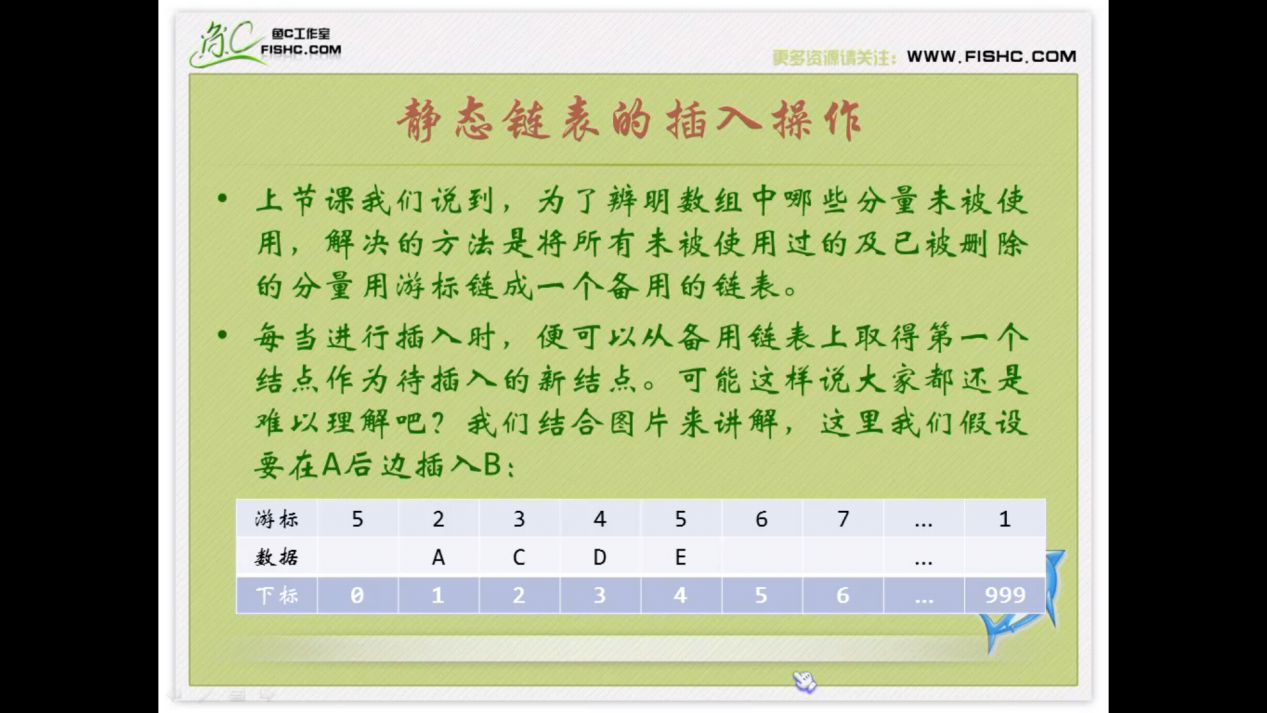
}

Space[MAXSIZE-1].cur = 0;

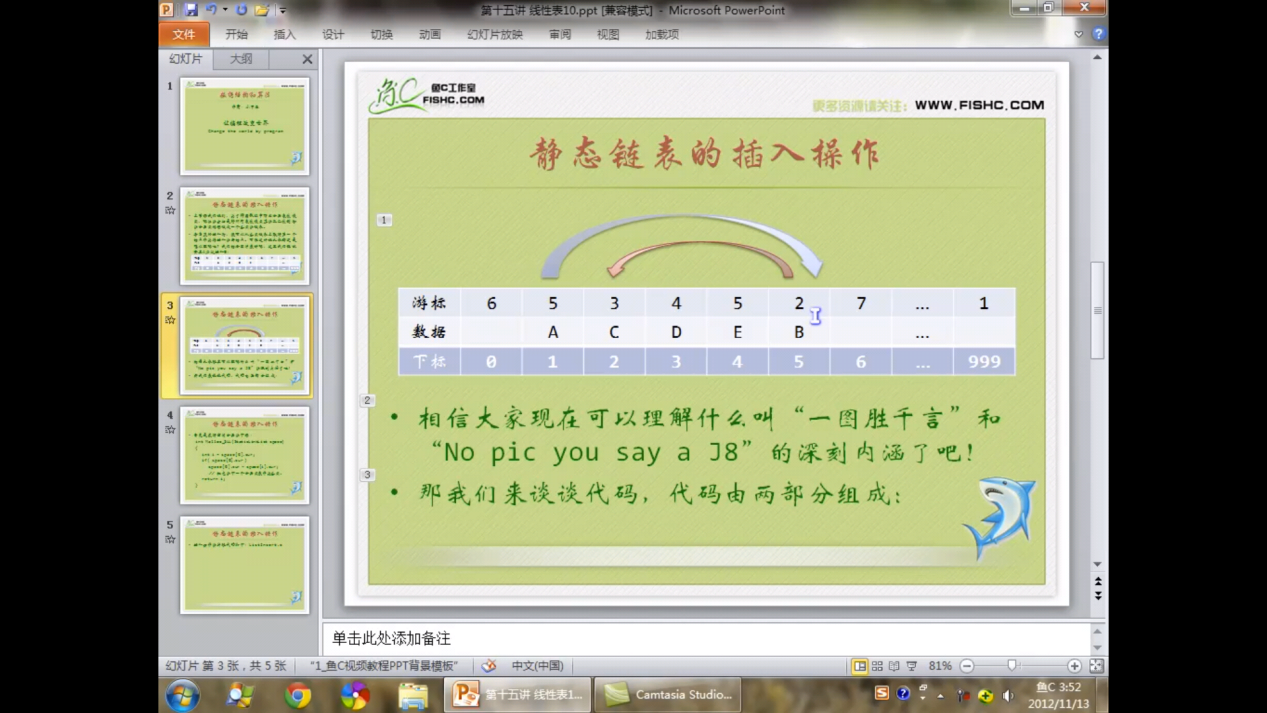
return OK;

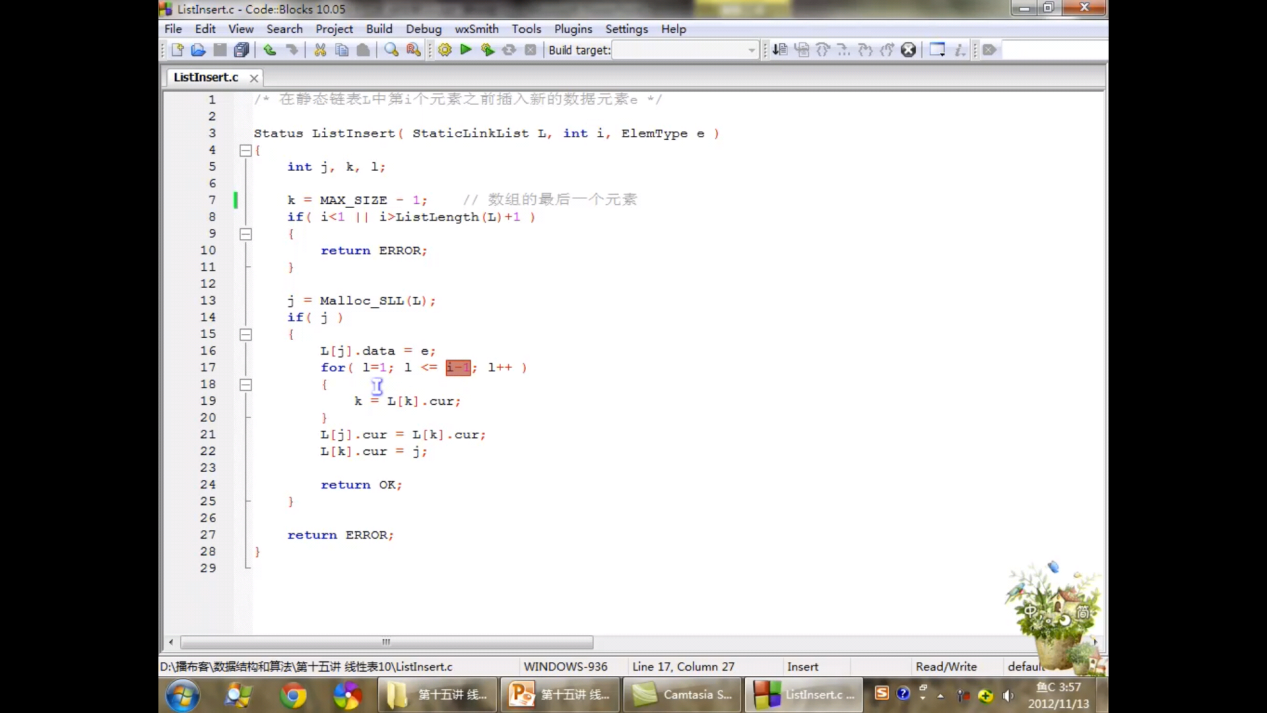
}

1. 静态链表的插入操作





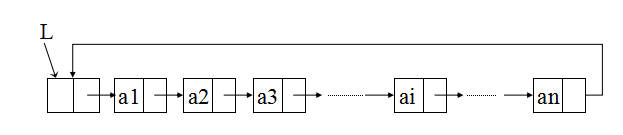




【循环链表】

循环链表的特点 ---- 表中的最后一个结点的指针域指向头结点, 整个链表形成一个环。

从表的任意结点出发可以找到表中其它结点。



1. 约瑟夫问题

//n个人围圈报数，报m出列，最后剩下的是几号？

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*next;

}node;

node \*create (int n)

{

node \*p = NULL,\*head;

head = (node\*)malloc(sizeof(node));

p = head;

node \*s;

int i = 1;

if(0!=n)

{

while (i<=n)

{

s = (node\*)malloc(sizeof(node));

s->data = i++;

p->next = s;

p = s;

}

s->next = head->next;//不要头结点了，就能形成闭环

}

free(head);

return s->next;

}

int main()

{

int n =41;

int m = 3;

int i;

node \*p = create(n);

node \*temp;

m%=n;

while(p!=p->next)

{

for(i=1;i<m-1;i++)

{

p = p->next;

}

printf(“%d->”,p>next->data);

temp = p->next;

p->next = temp->next;

free(temp);

p = p->next;

}

printf(“%d\n”,p->data);

}

【双向循环列表】

typedef char ElemType;

typedef int Status;

typedef struct DualNode

{

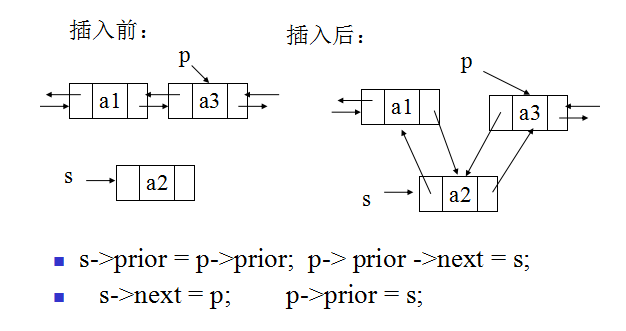
ElemType data;

struct DualNode \*prior;

struct DualNode \*next;

}DualNode,\*DuLinkList;

1. 插入结点



Status InitLink(DuLinkList \*L)

{

DualNode \*p,\*q;

\*L = (DuLinkList)malloc(sizeof(DualNode));

if(!(\*L))

{

return ERROR;

}

(\*L)->next = (\*L)->prior = NULL;

p = (\*L);

int i;

for(i=0;i<26;i++)

{

q = (DualNode\*)malloc(sizeof(DualNode));

if(!q)

{

return ERROR;

}

q->data = 'A'+i;

q->prior = p;

q->next = p->next;

p->next = q;

p = q;

}

p->next = (\*L)->next;

(\*L)->prior = p;

return OK;

}