链表

[TOC]

由于顺序表的\*\***插入**\*\*和\*\***删除**\*\*操作需要移动大量元素，影响了效率。\*\***链式存储**\*\*不要求逻辑上相邻的两个元素在物理位置上也相邻，而是通过“链”建立起数据元素的逻辑关系。因此，在链表的插入和删除不需要移动元素，只需修改指针。

链式存储的线性表称为\*\***链表**\*\*。其中每个结点（Node）只包含一个数据域和一个指针域的链表称为\*\***单链表**\*\*，首尾相连的单链表称为\*\***循环单链表**\*\*。每个结点只包含一个数据域和两个指针域的链表称为\*\***双链表**\*\*，首尾相连的双链表称为\*\***循环双链表**\*\*。还有一种链表称为\*\***静态链表**\*\*，该链表也有数据域和指针域，这里的指针是结点相对地址（数组下标），又称为\*\***游标**（cur）\*\*，静态链表和顺序表一样要预先分配一块连续的内存空间。

# #1 单链表

## ##1.1 结点定义

单链表结点定义为：

```

typedef struct LNode{ //定义单链表结点类型

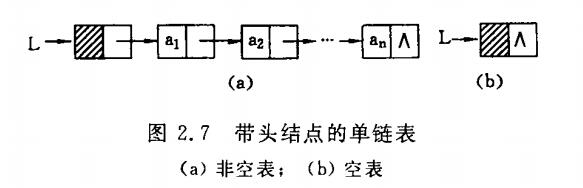
ElemType data; //数据域

struct LNode \*next; //指针域

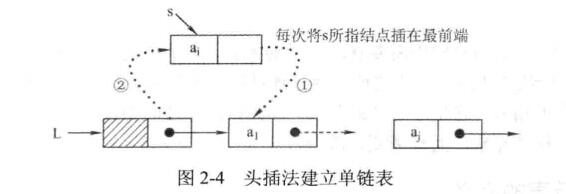
}LNode,\*LinkList; //LinkList相当于LNode\*，即：struct LNode\*

```

通常用头指针来标识一个单链表，如\*\***单链表L**\*\*（`LinkList L;`），L=NULL时表示一个空表。因此，为了操作方便就在单链表的首元结点前附加一个结点，即\*\***头结点**\*\*（`LinkList L=(LinkList)malloc(sizeof(LNode));`）。头结点数据域可以不带今后任何信息，但可记录链表长度，头结点指针域则指向首元结点。此时判断带头结点为空的条件为：`L->next==NULL`。



## ##1.2 头插法建立单链表



```

#include<stdio.h> //NULL

#include<malloc.h> //malloc

#define ElemType int

typedef struct LNode{ //定义单链表结点类型

ElemType data; //数据域

struct LNode \*next; //指针域

}LNode,\*LinkList; //LinkList相当于LNode\*，即：struct LNode\*

//头插法建立单链表

LinkList ListCreat\_L\_FromHead(){

LinkList L=(LinkList)malloc(sizeof(LNode));

L->next=NULL;

int e,length=0;

printf("请输入插法建立单链表元素（以-1结束）\n");

scanf("%d",&e);

while(e!=-1){

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

s->data=e;

s->next=L->next;

L->next=s;

length++;

scanf("%d",&e);

}

L->data=length; //令头结点记录链表长度

return L;

}

int main(){

LinkList L= ListCreat \_L\_FromHead();

printf("单链表元素个数：%d 分别是：",L->data);

for(LNode\* N=L->next;N;N=N->next){

printf("%d ",N->data);

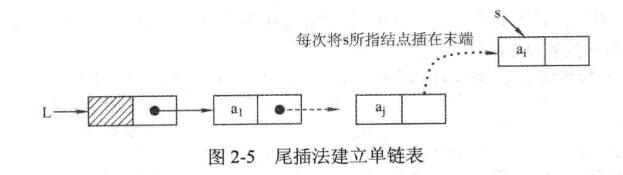
}

return 0;

}

```

## ##1.3 尾插法建立单链表



```

#include<stdio.h> //NULL

#include<malloc.h> //malloc

#define ElemType int

typedef struct LNode{ //定义单链表结点类型

ElemType data; //数据域

struct LNode \*next; //指针域

}LNode,\*LinkList; //LinkList相当于LNode\*，即：struct LNode\*

//尾插法建立单链表

void ListCreat \_L\_FromTail(LinkList &L){

L=(LinkList)malloc(sizeof(LNode));

L->data=0;L->next=NULL; //L->data为链表长度

LNode \*r=L; //r为表尾指针

printf("请输入插法建立单链表元素（以-1结束）\n");

int e;scanf("%d",&e);

while(e!=-1){

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

s->data=e;s->next=NULL;

r->next=s;

r=r->next;

L->data++; //链表长度+1

scanf("%d",&e);

}

}

int main(){

LinkList L;

ListCreat \_L\_FromTail(L);

printf("单链表元素个数：%d 分别是：",L->data);

for(LNode\* N=L->next;N;N=N->next){

printf("%d ",N->data);

}

return 0;

}

```

## ##1.4按序号查找表结点

```

//按序号查找第i个结点

LNode\* ListGetElem\_L (LinkList L,int i){

LNode\* p=L->next; //p为首元结点

if(i==0)return L; //返回头结点

if(i<0)return NULL; //i无效返回NULL

for(int j=1;j<i&&p;j++){

p=p->next;

}

return p; //返回第i个结点指针；若i>表长则返回的是NULL

}

```

## ##1.5按值查找表结点

```

//按值e查找表结点

LNode\* ListLocate\_L (LinkList L,int e){

LNode\* p=L->next; //p为首元结点

while(p->data!=e&&p){

p=p->next;

}

return p; //返回第i个结点指针；若i>表长则返回的是NULL

}

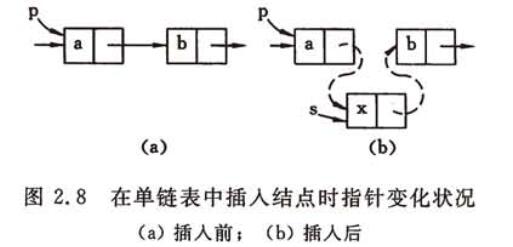
```

## ##1.6插入结点操作

在第i个位置插入结点要先查找插入位置的\*\***前驱结点**\*\*，单链表插入要执行两步必要操作：

- 把新结点s挂接后继结点并赋值；

- 把新结点挂在前驱结点p。



```

//在第i个位置插入结点操作

bool ListInsert\_L(LinkList &L,int i,int e){

LNode\* p= ListGetElem\_L(L,i-1); //查找插入位置的前驱结点

if(p==NULL)return false; //i定位无效，插入失败返回false

LNode\* s=(LNode\*)malloc(sizeof(LNode)); //为新值分配结点空间

s->next=p->next; //1.把新结点s挂接后继结点p->next

s->data=e; // 新结点赋值

p->next=s; //2.把新结点s挂在前驱结点p

return true; //插入成功返回true

}

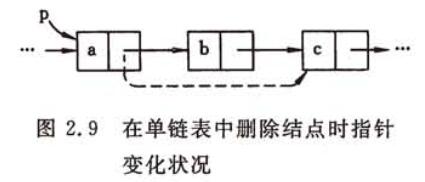
```

## ##1.7删除结点操作

删除第i个结点要先查找删除位置的\*\***前驱结点**\*\*，单链表删除要执行也两步必要操作：

- 将前驱结点p指向预删除结点的后继结点；

- 释放预删除结点空间。



```

//删除第i个结点操作

bool ListDelete\_L(LinkList &L,int i,int &e){

LNode\* p=ListGetElem\_L(L,i-1); //查找删除位置的前驱结点

if(p||p->next)return false; //i定位无效，删除失败返回false

LNode\* s=p->next; //将s指向预删除结点p->next

e=s->data; //将预删除结点s的值赋给e引用传回

p->next=p->next->next; //将前驱结点p指向预删除结点的后继结点

free(s); //释放预删除结点空间

return true; //删除成功返回true

}

```

## ##1.8合并有序链表

将两个有序单链表La和Lb合并为一个有序单链表Lc。

```

//合并有序链表

void ListMerge\_L(LinkList &La,LinkList &Lb,LinkList &Lc){

LNode\* pa=La->next,\*pb=Lb->next;

LNode\* pc=Lc=La; //用La头结点作为Lc的头结点

Lc->data=La->data+Lb->data; //Lc的长度为La长度与Lb长度之和

while(pa&&pb){

if(pa->data<pb->data){ //按非递减归并

pc->next=pa;pc=pa;pa=pa->next;

}else{

pc->next=pb;pc=pb;pb=pb->next;

}

}

pc->next=pa?pa:pb; //插入剩余段

free(Lb); //释放Lb的头结点

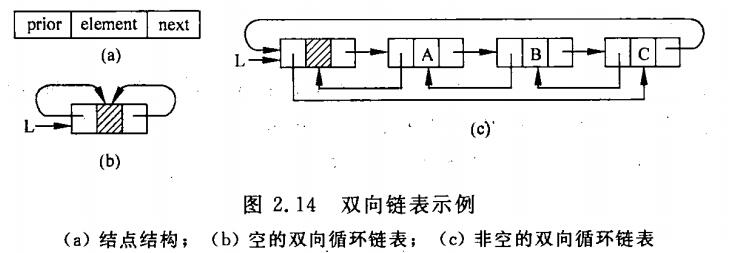
}

```

# #2 循环双链表

循环双链表定义头结点要维护循环双链表规则，即：` L->next=L->prior=L; `。若`DLNode \*p=L->next;…;p=p->next;`，则判断循环双链表为空的条件是`p==L;`。

## ##2.1 结点定义



```

#include<stdio.h>

#include<stdlib.h>

#define ElemType int

typedef struct DLNode{

ElemType data;

struct DLNode \*prior,\*next;

}DLNode,\*DLinkList;

int main(int argc,char \*\* argv ){

DLinkList L=(DLinkList)malloc(sizeof(DLNode));

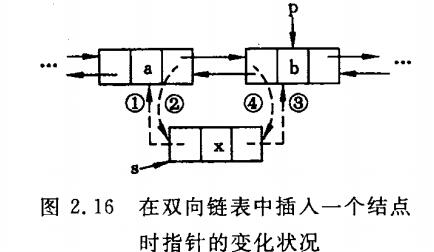
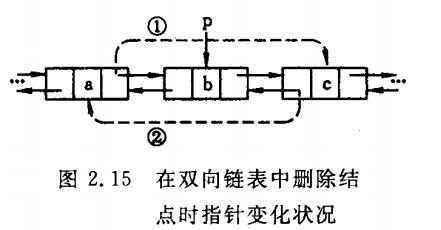
L->next=L->prior=L; //维护循环双链表规则

return 0;

}

```

## ##2.2 插入和删除操作



```

//双循环双链表第i个位置插入e

bool ListInsert\_DL(DLinkList &L,int i,ElemType e){

if(i<=0)return false;

DLNode \*p=L;

// for(int j=1;j<i;j++,p=p->next); //方式一：p是s的前驱

for(int j=0;j<i;j++,p=p->next); //方式二：p是s的后继

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

if(!s)return false;

s->data=e;

// s->next=p->next;s->prior=p->next->prior;//方式一：p是s的前驱

// p->next->prior=s;p->next=s;

s->prior=p->prior;p->prior->next=s; //方式二：p是s的后继

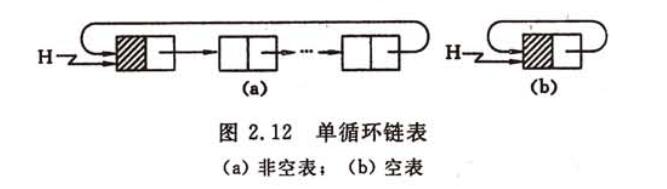
s->next=p;p->prior=s;

return true;

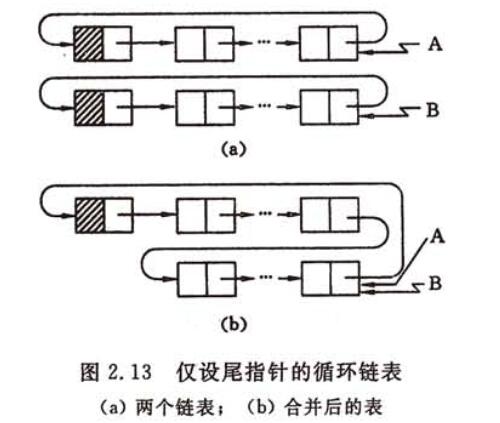
}

```

# #3循环单链表



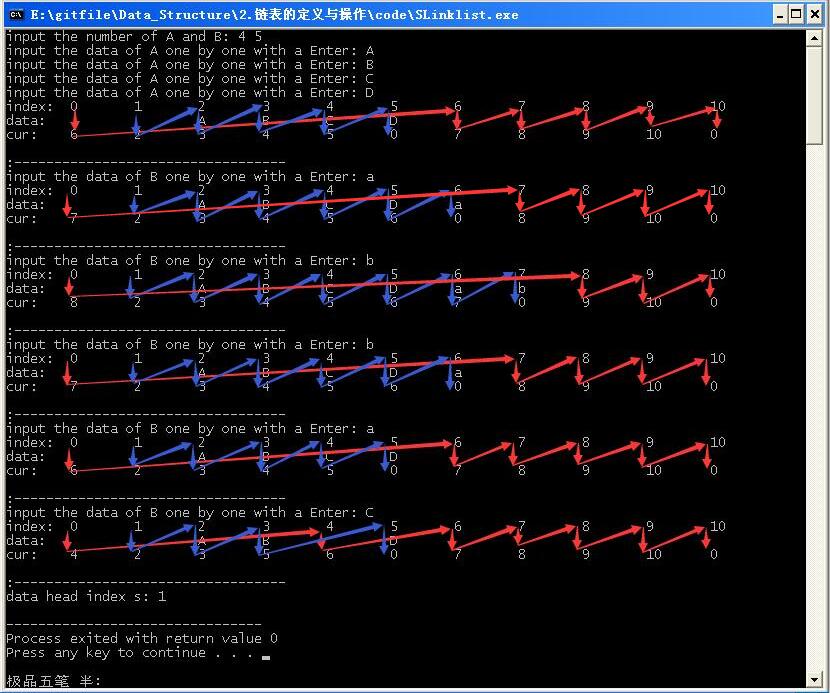
# #4带尾指针的循环单链表



# #5 静态链表

静态链表的插入、删除操作和动态链表的相同，只需修改指针（游标），而不需移动元素。但静态链表使用没有比单链表的方便，但对于一些不支持指针的高级语言（Basic语言）又是一种巧妙的设计方法。

我们用一个例子来说明静态链表的用法。现求集合(A-B)||(B-A)的元素，即遍历B中元素，如果A中没有该元素则插入A，否则删除从A中该元素。



```

#include<stdio.h>

#define MAXSIZE 11 //减去空闲链表和占用链表两个头指针，有MAXSIZE-2个空闲可以分配

#define ElemType char

typedef struct{

ElemType data; //值域

int cur; //游标

}component,SLinklist[MAXSIZE];

int s;

SLinklist SL;

void Print(); //声明 Print函数

/\*初始化一维数组space为空闲链表，space[0].cur为头指针\*/

void InitSpace\_SL(SLinklist &space){

for(int i=0;i<MAXSIZE;i++)

space[i].cur=i+1;

space[MAXSIZE-1].cur=0;

}

/\*从空闲链表分配一个空间以下标i返回,空闲链表空间不够则返回0\*/

int Malloc\_SL(SLinklist &space){

int i=space[0].cur;

if(space[0].cur)space[0].cur=space[i].cur;

return i;

}

//回收下标为k的空闲结点到空闲链表

void Free\_SL(SLinklist &space,int k){

space[k].cur=space[0].cur;

space[0].cur=k;

}

/\*求集合(A-B)||(B-A)\*/

void difference(SLinklist &space,int &S){

InitSpace\_SL(space); //初始化空闲链表

S=Malloc\_SL(space); //生成占用链表S的头结点

int m,n,r=S; //r表示占用链表S当前最后结点

printf("input the number of A and B: ");

scanf("%d %d",&m,&n);getchar(); //输入A和B的元素个数

/\*将A元素加入占用链表 \*/

for(int j=1;j<=m;j++){

int i=Malloc\_SL(space);

printf("input the data of A one by one with a Enter: ");

scanf("%c",&space[i].data);getchar();//

space[r].cur=i; r=i;

//Print();//

}

space[r].cur=0; //尾结点指针域为0

Print();//

/\*依次输入B的元素，若不在当前占用链表S中，则插入；否则删除 \*/

for(int j=1;j<=n;j++){

printf("input the data of B one by one with a Enter: ");

ElemType b;scanf("%c",&b);getchar();//

int p=S;

int k=space[S].cur;

while(k!=space[r].cur&&space[k].data!=b){

p=k;k=space[k].cur; //p表示k的前一个元素

}

if(k==space[r].cur){ //该元素不在占用链表S中，插入之

int i=Malloc\_SL(space);

space[i].data=b;

space[i].cur=space[r].cur;

space[r].cur=i;

r=i; //表尾r指向最新元素

}else{ //该元素已在占用链表S中，删除之

space[k].data=' '; //把删除的值清空

space[p].cur=space[k].cur;

Free\_SL(space,k);

if(r==k)r=p; //若删除的是r所结点，则需修改尾指针

}

Print();//

}//for

}

int main(int argc,char \*\* argv){

difference(SL,s);

printf("data head index s: %d\n",s);

return 0;

}

/\*定义 Print函数 \*/

void Print(){

printf("index:\t");

for(int i=0;i<MAXSIZE;i++){

printf("%d\t",i);

}printf("\n");

printf("data:\t");

for(int i=0;i<MAXSIZE;i++){

printf("%c\t",SL[i].data);

}printf("\n");

printf("cur:\t");

for(int i=0;i<MAXSIZE;i++){

printf("%d\t",SL[i].cur);

}printf("\n\n");

printf(":----------------------------------\n");

}

```

**\*\*Wu\_Being 博客声明\*\***：本人博客欢迎转载，请标明博客原文和原链接！谢谢！

【数据结构系列】《【数据结构1】顺序表》http://blog.csdn.net/u014134180/article/details/53907607

我的GitHub代码文件：https://github.com/1040003585/Data\_Structure