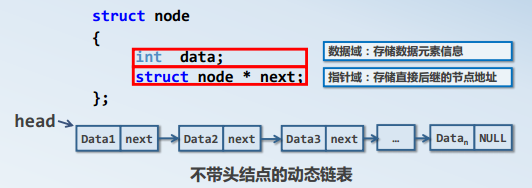
11.2 链表

1. 动态链表的构成：

 动态链表有一个或者多个结点构成，每个节点都是一个结构体对象。

 每个结点有数据域和指针域(关系)



2. ① 链表有一个头指针，存放第一个结点的地址

② 每一个结点由数据域和指针域构成

③ 最后一个结点指针域为NULL，称为“ 表尾”

④ 结点通常就是一个结构体变量

3. 链表特点：

① 链表中各元素可以不是连续存放的.

② 要找某结点必须先找到上个结点因此必须提供头指针.

③ 创建结点就是创建结构体和利用指针做成员

4. 链表的使用

（1）创建空链表

//创建节点

typedef struct node

{

int data;

struct node\*next;

}node;

//头插法

node \*f(int n)

{

int i=0;

node \*p;

node \*h=NULL;//创建head指针

while(i<n)

{

p=(node\*)malloc(sizeof(node));//分配内存

scanf("%d",&(p->data));

p->next=h;

h=p;

i++;

}

return h;

}

1. 利用malloc ( )函数向系统申请分配一个节点。  
2. 将第一个新节点的指针成员赋值为NULL，之后的使新节点的next指针指向上一个创建的节点。

（2）遍历

//遍历

void visit(node \*a)

{

while(a!=NULL)

{

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

a=a->next;//相当于栈区的i++

}

}

//计算节点个数

int shu(node \*a)

{

int i=0;

while(a!=NULL)

{

i++;

a=a->next;//相当于栈区的i++

}

return i;

}

//求和

int he(node \*l)

{

int i=0;

while(l!=NULL)

{

i=i+l->data;

l=l->next;

}

return i;

}

（3）查找

int updata(node\*l)

{

while(l!=NULL)

l=l->next;

}

（4）修改

int updata(node\*l,int a,int b)

{

while(l!=NULL&&l->data!=a)

l=l->next;

if(l!=NULL)

{

l->data=b;

return 1;

}

else

return 0;

}

或者

int updata(node \*l,int a,int b)

{

while(l!=NULL)

{

if(l->data==a);

{l->data=b;

return 1;

}

l=l->next;

}

return 0;

}

（5）增加（插入）

//在k节点之前插入

node \* insert(node \*l,int k,char x)

{

int i=1;

node \*h=l;

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

p->data=x;

if(k==1)

{

p->next=h;

return p;

}

else

{

while(l!=NULL&&i<k-1)

{

l=l->next;

i++;

}

if(l!=NULL)

{

p->next=l->next;

l->next=p;

}

return h;

}

}

//在x后插入

int insert(node \*l,char x,char y)

{

while(l!=NULL&&l->data!=x)

{

l=l->next;

}

if(l)

{

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

p->data=y;

p->next=l->next;

l->next=p;

return 1;

}

else

return 0;

}

（6）删除

node \*del(node\*l,int k)

{

node \*q=l;

node \*p;

int i=1;

if(k==1)

{

p=l;

l=l->next;

free(p);

return l;

}

else

{

while(l!=NULL&&i<k-1){

l=l->next;

i++;

}

p=l->next;

l->next=p->next;

free(p);

return q;

}

}

5.递归求节点

int f(node\*l)

{

if(l!=NULL)

return 0;

else

return 1+f(l->next);

6.释放所有节点

node \*s(node \*l)

{

node \*p;

while(l!=NULL)

{

p=l;

l=l->next;

free(p);

}

return l;

}