第二章 链表

链表通常由一连串节点组成，每个节点包含任意的实例数据（data fields）和一个用来指向下一个节点地址的指针(next指针)。

使用链表结构可以克服数组需要预先知道数据大小的缺点，链表结构可以充分利用计算机内存空间，实现灵活的内存动态管理。但是链表失去了数组随机读取的优点，同时链表由于增加了结点的指针域，空间开销比较大

链表在插入的时候可以达到O⑴的复杂度，但是查找一个节点或者访问特定编号的节点则需要O(n）的时间。

2.1单向链表

单向链表是指存放链表的结点中只有一个指针，指向它的下一个结点，而链表中最后一个结点的next指针为NULL。链表中的第一个结点，我们把它叫做头结点。只要知道了链表的头结点，就可以通过next指针，遍历整个链表。因此，一般在函数中，我们使用头结点来代表整个链表，将头结点传参给函数。

单向链表的存储结构：

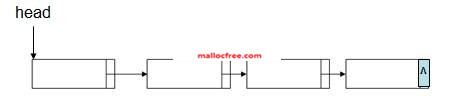
typedef struct \_node

{

       int val;//值域，可以增加更多的值域，这里仅以int类型为代表

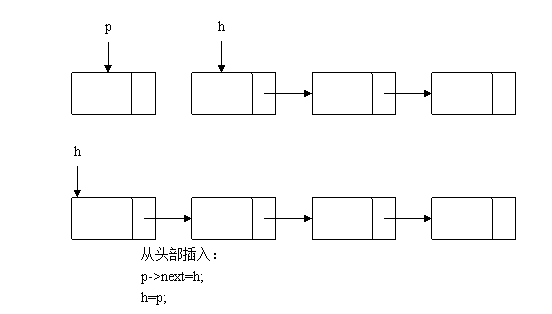
       struct \_node \*next;//指针域，指向下一个结点，最后一个结点指向NULL

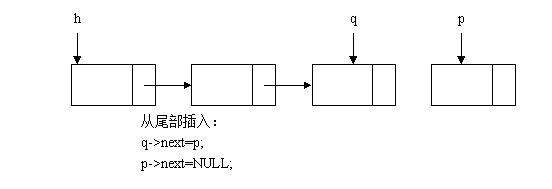
}Node, \*PNode;



2.1.1创建

如下图所示，在创建链表过程中，将新分配的一个链表结点插入到已知的链表中，可以分为两种情况：从头部插入和从尾部插入。





//从头部插入创建链表算法

node \*create\_linklist\_head()

{

       node \*h = NULL;

       while(1)

       {

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

              if(p==NULL)

              {

                     return h;

              }

              memset(p,0,sizeof(node));

              printf("Please input a data\n");

              scanf\_s("%d",&p->value);

              p->next = NULL;

              if(h==NULL) //如果头结点指针为空，表明这是创建的第一个结点

              {

                     h=p;

              }

              else

              {

                     p->next = h;

                     h=p;

              }

              if(p->value==0)

              {

                     break;

              }

       }

       return h;

}

//从尾部插入链表算法

node \*create\_linklist\_tail()

{

       node \*h=NULL;

       while(1)

       {

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

              if(p==NULL)

              {

                     return h;

              }

              memset(p,0,sizeof(node));

              printf("Please input a data\n");

              scanf\_s("%d",&p->value);

              p->next = NULL;

              if(h==NULL) //如果头结点指针为空，表明这是创建的第一个结点

              {

                     h=p;

              }

              else

              {

                     node \*q=h;//需要从头部遍历到尾部

                     while(q->next)

                     {

                            q=q->next;

                     }

                     q->next =p;//q为尾部，q->next=p，将新建结点设置为尾部

                     p->next = NULL;

              }

              if(p->value==0)//输入为零，则停止创建链表

              {

                     break;

              }

       }

       return h;

}

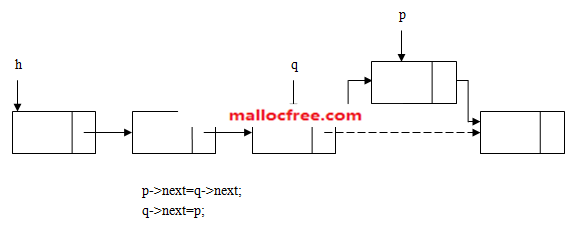
2.1.2插入

创建链表的算法中已经演示了如何从链表的头部和尾部插入链表结点。那么如何将链表结点插入到某个指定的结点之后呢？其实很简单，如下图所示：

让新建结点的next指针指向q的下一个结点，然后再让q的next指针指向p即可。

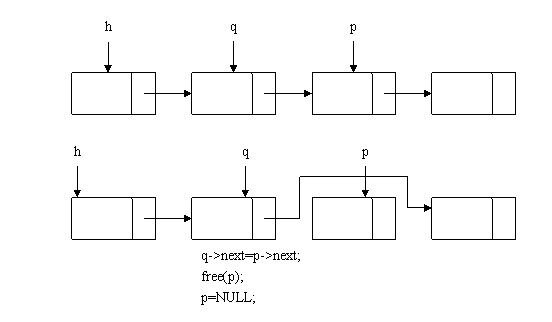
p->next=q->next;

q->next=p;



2.1.3删除

如下图所示，在单向链表中要删除一个结点p，首先要通过遍历，找到它的上一个结点q，然后将q的next指针指向p->next，然后即可将p删掉。



void del\_list\_node(node \*\*h,int value)

{

       if(h==NULL)

       {

              return;

       }

       node \*p = \*h;

       node \*q=NULL;

       while(p)

       {

              if(p->value==value)

              {

                     if(p==\*h)

                     {

                            \*h=(\*h)->next;

                            free(p);

                            return;

                     }

                     else

                     {

                            q->next=p->next;

                            free(p);

                            return;

                     }

              }

              q=p;

              p = p->next;

       }

}

2.1.4遍历

所谓对一种数据结构的遍历，是指对其中每个元素访问一次且仅访问一次。由于在链表里，有next指针指向下一个结点，而最后一个结点的next指针为NULL，因此可以通过头结点利用next指针遍历整个链表：

void traverse\_list(node \*h)

{

       node \*p = h;

       while(p)

       {

              printf("%d ",p->value);

              p=p->next;

       }

       printf("\n");

}

2.1.5销毁

销毁链表就是将链表中每个结点都删除掉。这个过程其实也是在遍历链表的时候可以完成的。参考下面的代码：

void destroy\_list(node \*h)

{

       node \*p =h;

       while(p)

       {

              node \*q=p;//先用q保存这个待删除的结点

              p=p->next;//p指向下一个结点

              free(q);//现在可以删除q了

       }

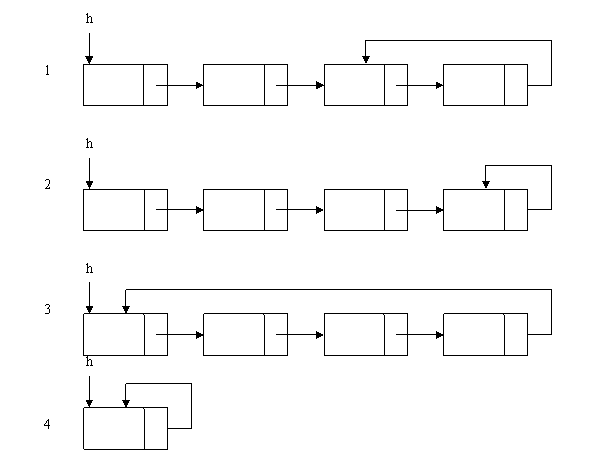
}

2.2单向循环链表

循环链表的最后一个结点的指针是指向该循环链表的第一个结点或者表头结点，从而构成一个环形的链。单链表的最后一个结点的指针是置为NULL。

单向循环链表的遍历：判断该结点链域的值是否是表头结点，当链域值等于表头指针时，说明已到表尾。而非循环单链表判断链域值是否为NULL。

单向循环链表的4种不同的形式：



//循环链表的创建：

node \*create\_list\_loop()

{

       node \*h=NULL;

       while(1)

       {

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

              if(p==NULL)

              {

                     return h;

              }

              memset(p,0,sizeof(node));

              printf("Please input a data\n");

              scanf\_s("%d",&p->value);

              p->next=NULL;

              if(h==NULL) //如果头结点指针为空，表明这是创建的第一个结点

              {

                     h=p;

                     p->next=h;

              }

              else

              {

                     node \*q=h;

                     while(q->next!=h)//q->next如果等于h，那么q就是最后一个结点

                     {

                            q=q->next;

                     }

                     q->next = p;

                     p->next =h;//将p的next指针指向h，构成一个循环

              }

              if(p->value==0)

              {

                     break;

              }

       }

       return h;

}

//循环链表的遍历方法，注意循环条件的判断：

void traverse\_loop\_list(node \*h)

{

       node \*p=h;

       do

       {

              printf("%d ",p->value);

              p=p->next;

       }while(p!=h);

       printf("\n");

}

//循环链表的销毁方法

void destroy\_loop\_list(node \*h)

{

       node \*p=h;

       do

       {

              node \*q=p;

              p=p->next;

              free(q);

       } while (p!=h);

}

//步长法判断链表是否含有循环

//思路是：定义2个指针遍历该链表，1个指针跑一步，1个指针跑两步；

//那么如果两个指针相遇，则表示有循环，否则无循环。

bool is\_a\_loop\_list(node \*h)

{

       node \*p=h;

       node \*q=h->next;

       while(p&&q&&q!=p&&q->next)

       {

              p=p->next;

              q=q->next->next;

       }

       if(p==q)//循环退出，p和q相等了，表示链表中存在循环

       {

              return true;

       }

       return false;

}

2.3双向链表

在双向链表中，结点除含有数据域外，还有两个链域，一个存储直接后继结点地址，一般称之为右链域；一个存储直接前驱结点地址，一般称之为左链域。

双向链表的存储结构定义如下：

typedef struct \_dnode

{

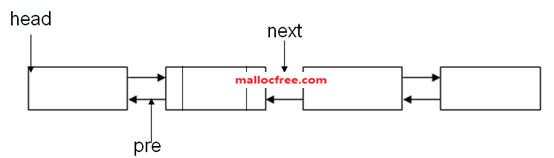
       int data;

       struct \_dnode \*pre;//前向指针，指向结点左边的结点

       struct \_dnode \*next;//后继指针，指向结点右边的结点

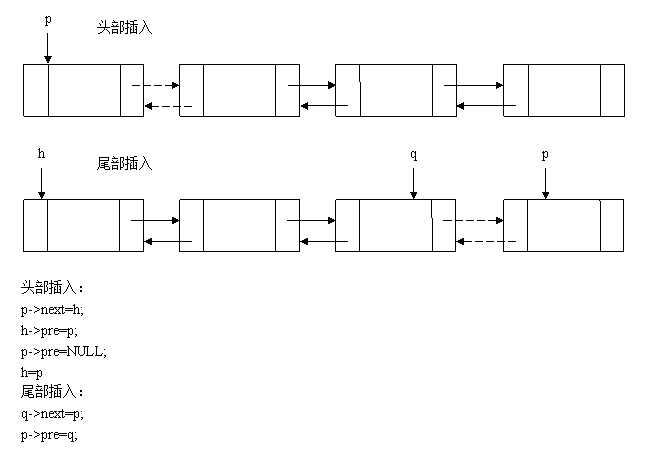
}dnode, \*pdnode;

双向链表的结构如下图所示：



2.3.1创建

在创建双向链表的时候，将一个新建的结点p，插入已知的双向链表中，有2种插入方法，如下图所示，一是直接插入头部，二是直接插入尾部。



//从头部插入方法创建双向链表：

dnode \*create\_dobulelist\_head()

{

       dnode \*h = NULL;

       while(1)

       {

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

              if(p==NULL)

              {

                     return h;

              }

              memset(p,0,sizeof(p));

              printf("Please input your data\n");

              scanf\_s("%d",&p->value);

              p->next=p->pre=NULL;

              if(h==NULL) //如果头结点指针为空，表明这是创建的第一个结点

              {

                     h=p;

              }

              else

              {

                     p->next = h;

                     h->pre=p;

                     h=p;

              }

              if(p->value==0)

              {

                     break;

              }

       }

       return h;

}

//从尾部插入创建双向链表：

dnode \*create\_dobulelist\_tail()

{

       dnode \*h = NULL;

       while(1)

       {

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

              if(p==NULL)

              {

                     return h;

              }

              memset(p,0,sizeof(p));

              printf("Please input your data\n");

              scanf\_s("%d",&p->value);

              p->next=p->pre=NULL;

              if(h==NULL)//如果头结点指针为空，表明这是创建的第一个结点

              {

                     h=p;

              }

              else

              {

                     dnode \*q=h;

                     while(q->next)//需要先从头结点开始遍历到尾结点

                     {

                            q=q->next;

                     }

                     q->next = p;

                     p->pre=q;

                     p->next = NULL;

              }

              if(p->value==0)//创建循环退出

              {

                     break;

              }

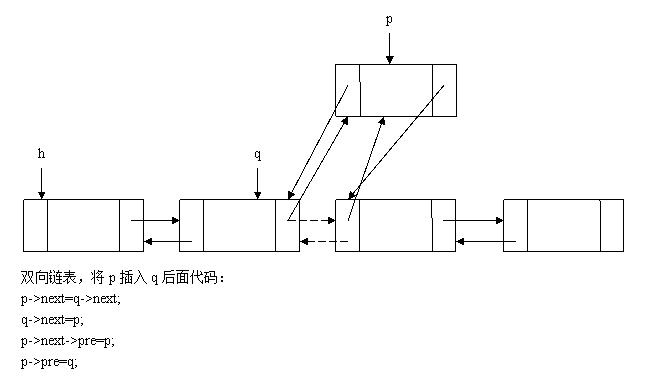
       }

       return h;

}

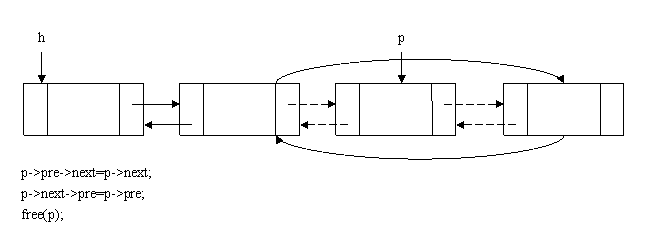
2.3.2插入

如图所示，将一个结点p插入到双向链表q的后面（注意，照着图分析最有效，不要死记硬背插入代码）：



2.3.3删除

双向链表中，将一个结点p从双向链表中删除方法：



2.3.4遍历

双向链表的遍历，和单向链表的遍历方法差不多：

void traverse\_dlist\_next(dnode \*h)

{

       dnode \*p=h;

       while(p)

       {

              printf("%d ",p->value);

              p=p->next;

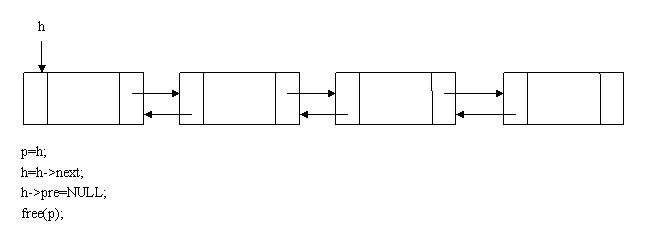
       }

       printf("\n");

}

2.3.5销毁

双向链表的销毁，与单向链表的销毁算法类似：



void destroy\_dlist(dnode \*h)

{

       dnode \*p =h;

       while(p)

       {

              dnode \*q=p;

              p=p->next;

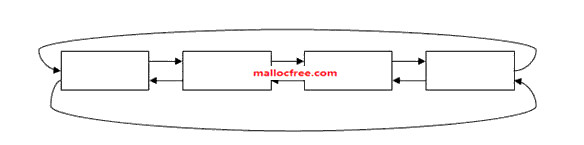
              free(q);

       }

}

2.4双向循环链表

双向循环链表既是双向链表，又是循环链表。头结点的左链域指向最后一个结点，最后一个结点的右链域指向头结点。



我们可以通过pre或next指针遍历双向循环链表，如下面的代码，双向循环链表的遍历方法：

p=h;

do {

    p=p->next;

} while(p!=h)

课后练习：

实现双向循环链表的创建，遍历，删除，销毁，插入