[C/C++基础知识] 那些被遗忘的链表知识

最近快毕业了,复试又复习了一些知识.其中就包括那些被遗忘的链表知识,而它又是C语言中非常重要一个知识点.同时发现很多同学都会忘记该知识,所以通过这篇文章一方面帮助大家回忆链表知识,同时对刚接触C语言的同学也有帮助.我采用问答的方式回顾那些知识,希望能接受!

提示:该文章引用李凤霞(北理)的《C语言程序设计教程》及课件和谭浩强(清华)的《C程序设计》.

一.链表基本概念

1.什么是链表?

链表是一种常见的动态进行存储分配的数据结构.

2.为什么会出现链表这种结构呢?

(1).C语言中使用数组存放数据时,须先定义固定数组长度,确定元素个数.如果数据超过其容量就会发生数组溢出;为防止该溢出,往往会定义很大的数组,但这样又造成资源空间浪费.如果程序采用动态数组方法复制增长的数据,方法可行但效率太低;

(2).如果在数组中需要删除一个数据或插入一个数据时,此时需要将删除或插入点数组后面的数据依次移动,这样的移动也会导致程序效率非常低.

3.此时,链表这种动态存储数据的结构油然而生.你是否看到了数组与链表两者一些简单区别呢?那么链表的基本单位又是什么呢?

结点是链表的基本存储单位,在链表中所有元素都存储在一个具有相同数据结构的结点中.一个结点对应一组数据元素,每个结点在内存中使用一块连续的存储空间(一个结点可由多种数据域组成),每个结点之间使用不连续的存储空间,结点之间通过指针链接.结点由数据域和指针域/链组成.常用定义如下:

```
struct node
{
        dadatype data;          //数据域
        struct node *next;      //指针域:指向node结点指针
};
```

4.知道了链表的基本存储单位后,那链表的基本组成部分是什么呢?

链表一般由三部分组成:

(1).表头指针:指向链表头结点的指针,头指针是链表的标志,通常用head定义头指针;

(2).表头结点:链表的第一个结点,一般不保存数据信息.链表中可没有表头结点(后面讲述),它是为方便引入结点.

(3).数据结点:实际保存数据信息的结点.示意图如下:

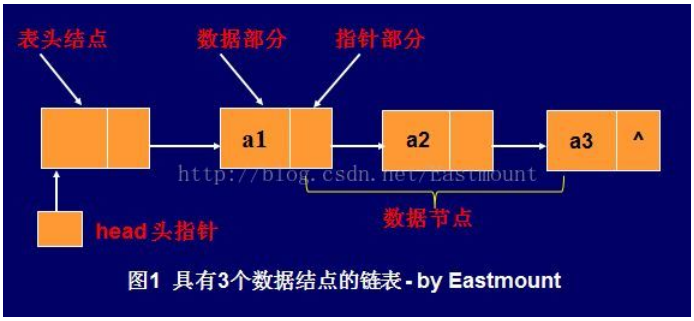


图1 具有3个数据结点的链表 - by Eastmount

5.前面讲到可能链表中没有表头结点,那么链表常见形式有哪些呢?

常见的形式包括:有表头结点的单向链表、无表头结点的单向链表、有表头的单向循环表、无表头的单向循环表.其中有表头与无表头的差别在于是否有表头结点,插入删除操作对应不同的判断;单向链表与单向循环链表的区别在于最后一个数据结点指针是NULL还是指向表头结点.双向链表即两个指针分别指向前一个位置和后一个位置的链表.

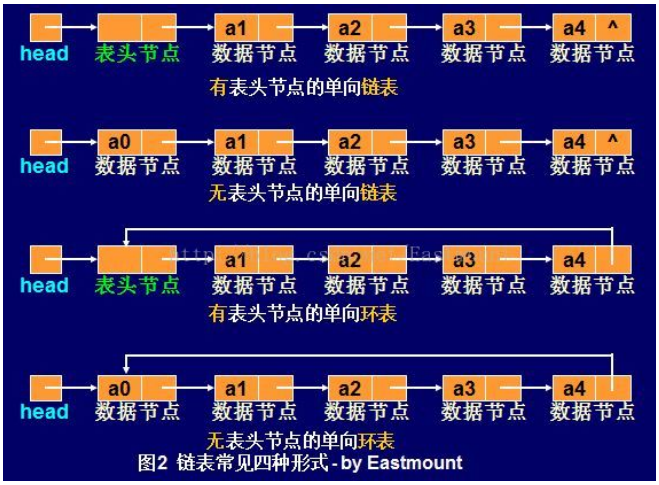


图2 链表常见四种形式 - by Eastmount

6.那么链表中的常见操作包括哪些呢?

链表的常见操作包括:建立链表、遍历链表、求链表表长、插入数据、删除结点.下面将详细解决.

二.链表基本操作

1.建立链表

建立链表前先定义一个包含数据域与指针域的结构类型,然后建立指向表头结点的头指针head,通过malloc函数动态申请内存作为表头结点.其中void *malloc(int size)的头文件为"stdlib.h".动态分配长度size字节存储区.

```
//定义结构类型
typedef struct node
{
        char name[20];              //数据域
        struct node *next;          //指针域
}NODE;
NODE *head,*p;                  //说明指针
//建立空链表(仅表头结点)
p=(NODE *)malloc(sizeof(NODE));
p->next=NULL;
head=p;
//插入一个数据结点
p=(NODE *)malloc(sizeof(NODE));
gets(p->name);                   //输入姓名
p->next=head->next;              //p指向下一个结点=head指向下一个及诶单
head->next=p;                    //p结点插入表头结点head后
```

上面代码的执行过程如下图所示:

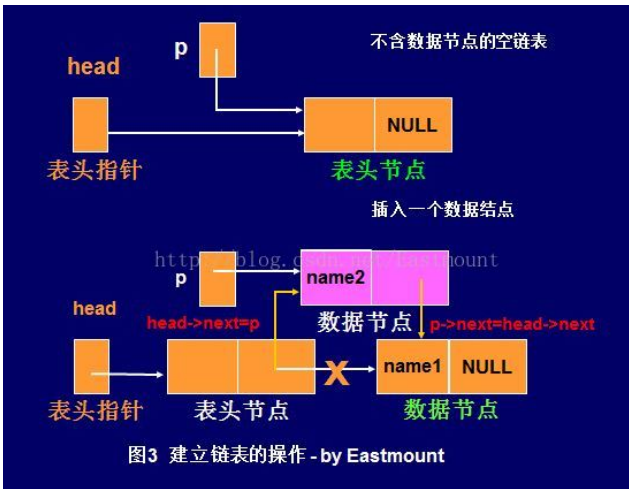


图3 建立链表的操作 - by Eastmount

如果想通过函数实现建立链表的代码如下:

```
//建立n个结点的链表
void create(NODE *head,int n)
{
        NODE *p;
        for(;n>0;n--)
        {
                p=(NODE *)malloc(sizeof(NODE));
                gets(p->name);
                p->next=head->next;
                head->next=p;
        }
}
```

但是需要注意:通过此种方法建立时,总是在head后插入一个新的结点,这就导致最终插入的顺序为输入顺序的逆序存储该n个结点的信息.如果想顺序插入,只需要让head结点指向第一个插入结点p,第一个指向第二个,依次最后一个结点指向NULL即可.在约瑟夫循环中我将讲述.

2.遍历链表

遍历链表中某个结点,即从链表第一个结点开始依次进行查找通过output函数可以实现,如果想具体增加一些遍历条件可以在函数中添加,下面output函数依次输出学生姓名.

```
//遍历输出结果
void output(NODE *head)
{
        NODE *p;
        p=head->next;     //含表头结点
        while(p!=NULL)
        {
                puts(p->name);
                p=p->next;
        }
}
```

如果想计算链表的长度,如果含表头结点时,从第一个结点开始依次遍历,没找到一个结点其长度加1,直到链表尾.如果链表为空时,表头结点head->next==null.此时返回的为0即可.

```
//计算链表长度
int count(NODE *head)
{
        int number=0;
        NODE *p;
        p=head->next;
        while(p!=NULL)
        {
                number++;
                p=p->next;
        }
        return number;
}
```

自定义main函数调用其函数,程序测试结果如下图所示,是不是反序一目了然.

```
输入字符串
LiMing
YangHan
TaiMing
HuangNiu
MaYun
输出结点个数
5
输出字符串
MaYun
HuangNiu
TaiMing
YangHan
LiMing
请按任意键继续. . .
```

3.插入数据

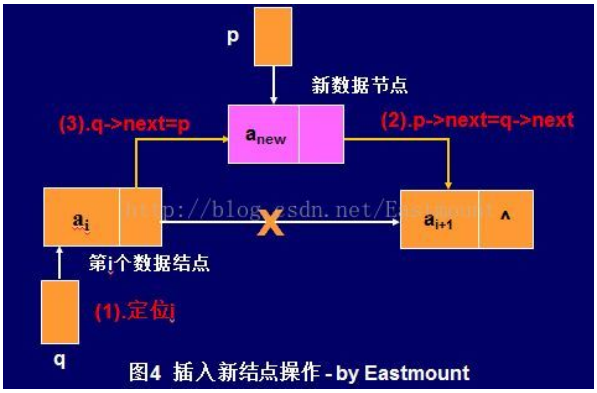
在链表中第i个结点后面插入一个新节点的算法如下:

(1).定位第i个结点.让指针q指向第i个结点,指针p指向要插入的结点.

(2).链接后面的指针:p->next=q->next.

(3).链接前面指针:q->next=p.

如下图所示过程:



具体代码如下图所示,其中采用insert函数插入新结点时,可能遇到两种特殊情况:其一是向空表中插入新节点,其二是向链表最后一个元素后面插入一个新结点.

```
//插入新结点 head头指针 p插入指针 i位置
void insert(NODE *head,NODE *p,int i)
{
        NODE *q;
        int n = 0;
        q = head;
        //第一步 寻找到第i个结点位置
        while(n<i&&q->next!=NULL)
        {
                q = q->next; n++;
        }
        //第二步 链接后面的指针
        p->next = q->next;
        //第三步 链接前面的指针
        q->next = p;
}
```

4.删除数据

在链表中可以删除任意一个数据结点,其中删除链表中第i个结点的算法如下:

(1).定位第i-1个结点位置.指针q指向第i-1个结点,指针p指向被删除结点.

(2).摘链:q->next=p->next.

(3).释放结点p:free(p).

其中void free(void *p)释放p所指向的内存空间,头文件为"stdlib.h".如下图所示:

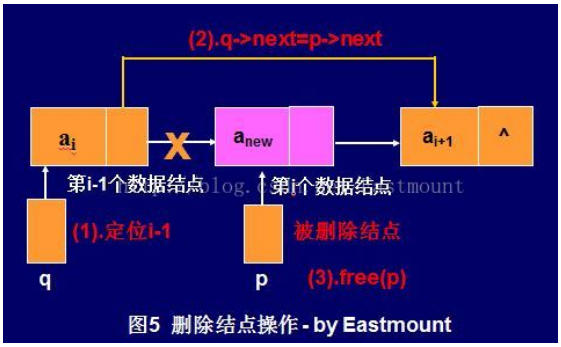


图5 删除结点操作 - by Eastmount

具体代码如下图所示,同时通常在删除结点p后,需要把q指向的下一个新的结点赋值为p,可以继续执行删除操作.

```
//删除第i个结点
void delete_node(NODE *head,int i)
{
        NODE *q,*p;
        int n = 0;
        q = head;
        //第一步 寻找到第i-1个结点位置 q指针指向
        while(n<i-1&&q->next!=NULL)
        {
                q = q->next;
                n++;
        }
        if(q->next!=NULL)
        {
                //第二步 摘链 q指向p的下一个结点
                p = q->next;
                q->next = p->next;
                //第三步 释放结点
                free(p);
        }
}
```

希望读者思考一个问题,在删除结点时,如果链指针摘链操作后没有free释放掉该结点,会导致什么结果呢?如果你学过C++或Jave,你又能回忆起它的内存管理和泄露知识吗?

三.链表经典问题-约瑟夫循环问题

通过上面对链表的讲述,你是否能回忆起一些它的简单知识呢?下面我想通过链表知识中最经典的题目"约瑟夫循环问题"让我们看看链表如何在实例中应用.

题目:有N个孩子围成一圈并依次编号(从1起),老师指定从第M个孩子开始报数,当报到第S个孩子时出列,然后下一个孩子从1开始继续报数,依次出列.求孩子出列的顺序或求最后一个孩子的编号.

输入:输入n(孩子个数),m(开始报数编号),s(报s出列).

输出:孩子出列顺序或最后一个孩子编号.

分析:如下图所示,当输入n=5,m=2,s=3时表示总共有5个孩子,通过单向循环链表围成一圈,m=2表示从第二个孩子开始报数,第二个孩子报数1,第三个报数2,第四个孩子报数3(s=3)出列.依次出列顺序为:4-2-1-3-5.

完成该程序需要:

(1).建立单向循环链表.注意此时建表是顺序建立,前面讲述的在head后插入新结点为逆序建表.此时需要依次插入head->a1->a2.最后在让q->next=head构建循环链表.(代码无表头结点)

(2).通过循环找到开始报数的结点,p指向开始报数的结点,q指向其前一个结点,因为删除p时需要通过前一个结点q摘链.

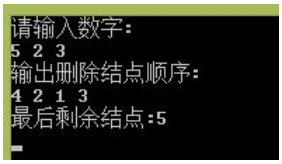
(3).循环依次报数删除结点,知道p=p->next退出循环,此时仅剩最后一个结点.

```
#include<stdio.h>
#include<stdlib.h>

//定义结构
typedef struct node
{
        int no;
        struct node * next;
}NODE;

int main()
{
        int i,j,k;
        int n,m,s;              //n个孩子 从m个开始报数 s个出列
        NODE *head,*p,*q;       //头结点 p插入结点 q插入前一个结点
        printf("请输入数字:\n");
        scanf("%d %d %d",&n,&m,&s);

        //建表 顺序插入无表头单向循环链表
        head=NULL;
        for(i=1;i<=n;i++)    //i存储序列号
        {
                p=(NODE*)malloc(sizeof(NODE));
                p->no=i;
                if(head==NULL) head=p;          //第一个结点存入head
                else q->next=p;                 //q链接新插入结点p
                q=p;                            //新插入结点构成链尾
        }
        q->next=head;                       //链尾链接链头构成循环

        //寻找输出的位置m p为开始的结点 q为其前面一个结点
        q=head;
        p=head;
        for(k=1;k<m;k++)                    //如果m=1 即第一个位置
        {
                p=p->next;
        }
        while(q->next!=p)                   //寻找q指针 q为p的前一个结点
        {
                q=q->next;
        }

        //删除结点及输出
```

```
        printf("输出删除结点顺序:\n");          while(p->next!=p)
        {
                //寻找到要删除结点位置
                for(j=1;j<s;j++)
                {
                        q=p;
                        p=p->next;
                }
                //输出结点并删除
                printf("%d ",p->no);
                q->next=p->next;
                free(p);
                p=q->next;
        }
        printf("\n最后剩余结点:%d\n",p->no);
        system("PAUSE");
        return 0;
}
```

测试用例及输出结果如下所示:

(1).输入n=5 m=2 s=3



(2).输入n=35 m=5 s=3

如果你是一位刚接触C语言的同学,希望文章能令你对链表有些认识;

如果你是考研或找工作的同学,希望对你在面试题或考研题中有所帮助;

如果你对链表有很深入的认识,希望当你阅读该文章时能对我这样的年轻人慧心一笑;

最后希望该文章对大家有所帮组,同时如果文章中有错误或不足之处,还请海涵!同时感谢母校BIT及老师,四年转瞬即逝,还有很多知识需要学习.这篇文章仅仅是自己对链表知识的一些总结及在线笔记,请尊重作者的劳动果实!

(By:Eastmount 2014-3-28 夜2点 原创CSDNhttp://blog.csdn.net/eastmount/**)**