iotdog

昵称： iotdog
园龄： 4年9个月
粉丝： 3
关注： 2
+加关注

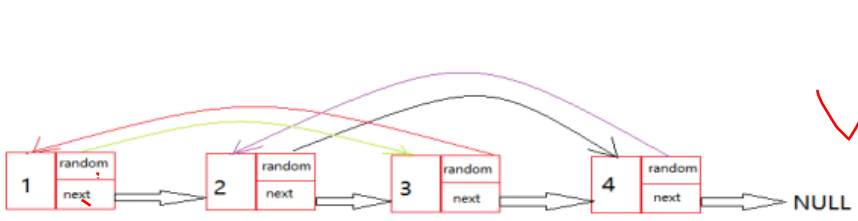
C语言之复杂链表的复制（图示详解）

什么是复杂链表?

复杂链表指的是一个链表有若干个结点，每个结点有一个数据域用于存放数据，还有两个指针域，其中一个指向下一个节点，还有一个随机指向当前复杂链表中的任意一个节点或者是一个空结点。今天我们要实现的就是对这样一个复杂链表复制产生一个新的复杂链表。

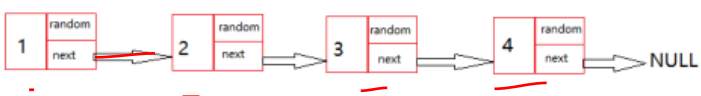
复杂链表的数据结构如下：

```
typedef int DataType;          //数据域的类型

//复杂链表的数据结构

typedef struct ComplexNode

{

DataType _data ;                        // 数据

struct ComplexNode * _next;             // 指向下个节点的指针

struct ComplexNode * _random;           // 指向随机节点（可以是链表中的任意节点 or 空）

}ComplexNode;
```



上图就是一个复杂链表的例子，那么我们应该如何实现复杂链表的复制呢?

1、首先我们应该根据已有的复杂链表创建一条新的复杂链表，但是这个新的复杂链表的所有的结点的random指针都指向空，这样是很好实现的，相当于我们创建了一条简单的单链表（newlist），我们要复制的链表不妨称之为oldlist。



< 2022年4月 >
日 一 二 三 四 五 六
27 28 29 30 31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
1 2 3 4 5 6 7

搜索
找找看
谷歌搜索

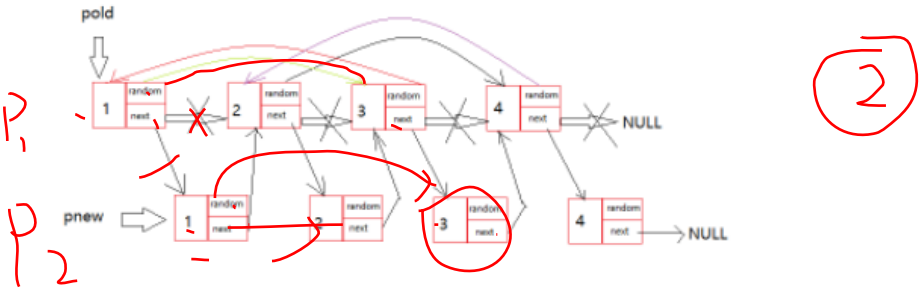
常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签
C++面试题(2)
链表面试题(1)
C语言(1)
指针与引用(1)
常量引用(1)
函数重载(1)

随笔档案
2017年7月(3)

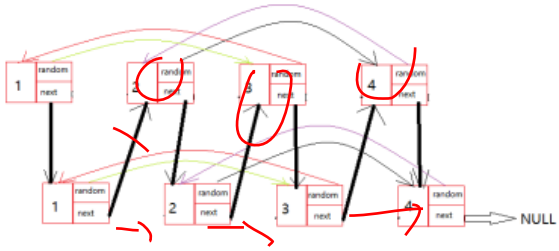
阅读排行榜
1. C语言之复杂链表的复制（图示详解）(5916)

2、接下来我们应该把新创建的这条复杂链表（newlist）与已有的复杂链表（oldlist）合并成如下的形式：

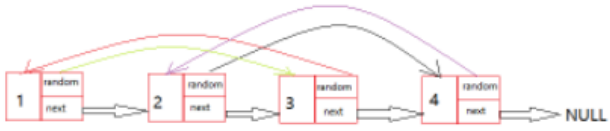


在这种情况下我们已经把两条复杂链表合并成了一条链表（称之为linklist），通过对这条链表（linklist）的观察，我们可以发现合并的链表（linklist）中属于newlist的结点pnew的上一个结点pold（属于oldlist的结点）的random指针所指向的结点的next指针就应该是pnew结点的randow指针所指向的结点。

这样我们让pold和pnew指针一直往后走最后就可以实现对所有属于新创建的复杂链表（newlist）的random指针指向相应的结点的操作。构成的复杂链表如下图



在完成以上的步骤之后我们所要做的工作就很简单了，我们只要把这一条链表linklist分开成我们的newlist链表和oldlist链表就可以了。



这样我们就完美的完成了复杂链表的复制工作下面就是具体实现的代码：

头文件complexnode.h：

```
#ifndef __COMPLEX__NODE__H__

#define __COMPLEX__NODE__H__


//包含头文件

#include <stdio.h>

#include<stdlib.h>
```

```
12
13 #include <assert.h>
14
15
16
17
18
19 typedef int DataType;         //数据域的类型
20
21
22
23 //复杂链表的数据结构
24
25 typedef struct ComplexNode
26
27 {
28
29 DataType _data ;                                  // 数据
30
31 struct ComplexNode * _next;                 // 指向下个节点的指针
32
33 struct ComplexNode * _random;         // 指向随机节点（可以是链表中的任意节点 or 空）
34
35 }ComplexNode;
36
37
38
39 //创建一个复杂链表的结点
40
41 ComplexNode * BuyComplexNode(DataType x);
42
43
44
45 //打印复杂的单链表
46
47 void Display(const ComplexNode * cplist);
48
49
50
51 //复杂链表的复制
52
53 ComplexNode * CopyComplexNode(ComplexNode * cplist);
54
55
56
57 #endif//__COMPLEX__NODE__H__
```

具体功能实现complexnode.c

```
 1 #include "complexnode.h"
 2
 3
 4
 5 //创建一个复杂链表的结点
 6
 7 ComplexNode * BuyComplexNode(DataType x)
 8
 9 {
10
11 ComplexNode *cnode = (ComplexNode *)malloc(sizeof(ComplexNode));
12
13 if(cnode == NULL)//创建失败
14
15 {
16
17 perror("BuyComplexNode()::malloc");
18
19 return NULL;
20
21 }
22
23 //创建成功
24
```

```
cnode->_data = x;

cnode->_next = NULL;

cnode->_random = NULL;

return cnode;

}


//打印复杂的单链表

void Display(const ComplexNode * cplist)

{

ComplexNode *pnode = cplist;

while (pnode)

{

printf("%d::%d -->",pnode->_data,pnode->_random->_data);

pnode = pnode->_next;

}

printf("over\n");


}


//复杂链表的复制

ComplexNode * CopyComplexNode(ComplexNode * cplist)

{


ComplexNode * pold = NULL;

ComplexNode * pnew = NULL;

ComplexNode * newlist = NULL;//指向新的复杂链表的头结点的指针

pold = cplist;

//创建一条新的复杂链表

while(pold != NULL)

{

ComplexNode * new_node = BuyComplexNode(pold->_data);

if(newlist == NULL)//当新的复杂链表中没有结点时

{

newlist = new_node;

}

else//当新的复杂链表有结点时

{

ComplexNode * node = newlist;

while(node->_next != NULL)//找到最后一个结点
```

```
102
103 {
104
105 node = node->_next;
106
107 }
108
109 node->_next = new_node;//插入新的结点
110
111 }
112
113 pold = pold->_next;
114
115
116
117 }//创建新的复杂链表结束
118
119
120
121 //合并两条复杂链表
122
123 pold = cplist;
124
125 pnew = newlist;
126
127 while (pold)
128
129 {
130
131 ComplexNode * curold = NULL;
132
133 ComplexNode * curnew = NULL;
134
135 curold = pold->_next;
136
137 curnew = pnew->_next;
138
139 if(pold->_next == NULL)
140
141 {
142
143 pold->_next = pnew;
144
145 pold = curold;
146
147 pnew = curnew;
148
149 break;
150
151 }
152
153 pold->_next = pnew;
154
155 pnew->_next = curold;
156
157 pold = curold;
158
159 pnew = curnew;
160
161 }//合并两条复杂链表结束
162
163
164
165 //让新创建的那条复杂链表上的所有结点的random指针指向相应的结点
166
167 pold = cplist;
168
169 pnew = newlist;
170
171 while (pnew)
172
173 {
174
175 pnew->_random = pold->_random->_next;
176
177 pold = pnew->_next;
178
```

```
179 if(pold == NULL)//这是pnew的_next指针已经指向空
180
181 {
182
183 break;
184
185 }
186
187 pnew = pold->_next;
188
189 }//结束
190
191
192
193 //分离合并后的复杂链表
194
195 pold = cplist;
196
197 pnew = newlist;
198
199 while (pold)
200
201 {
202
203 ComplexNode * curold = NULL;
204
205 ComplexNode * curnew = NULL;
206
207 if(pnew->_next == NULL)//已经分离完成
208
209 {
210
211 pold->_next = NULL;
212
213 pnew->_next = NULL;
214
215 break;
216
217
218
219 }
220
221 curold = pold->_next->_next;
222
223 curnew = pnew->_next->_next;
224
225
226
227 pold->_next = curold;
228
229 pnew->_next = curnew;
230
231 pold = curold;
232
233 pnew = curnew;
234
235 }//分离合并的复杂链表结束
236
237
238
239 return newlist;
240
241 }
```

测试代码test.c：

```
1 #include "complexnode.h"
2
3 //
4
5 //复杂链表的复制。?个链表的每个节点，有?个指向next指针指向下?个节
6
```

```
//点，还有?个random指针指向这个链表中的?个随机节点或者NULL，现在要

//求实现复制这个链表，返回复制后的新链表。

//ps: 复杂链表的结构



void test()

{

ComplexNode * cplist;

ComplexNode * copylist;

ComplexNode * node1;

ComplexNode * node2;

ComplexNode * node3;

ComplexNode * node4;

cplist = BuyComplexNode(1);

node1 = BuyComplexNode(2);

node2 = BuyComplexNode(3);

node3 = BuyComplexNode(4);

node4 = BuyComplexNode(5);

cplist->_next = node1;

node1->_next = node2;

node2->_next = node3;

node3->_next = node4;

cplist->_random = node3;

node1->_random = node4;

node2->_random = cplist;

node3->_random = node1;

node4->_random = node2;

Display(cplist);

copylist = CopyComplexNode(cplist);

Display(copylist);



}

int main()

{

test();

return 0;

}
```

程序的运行结果如下图：

```
1::4 -->2::5 -->3::1 -->4::2 -->5::3 -->over
1::4 -->2::5 -->3::1 -->4::2 -->5::3 -->over
```

标签：C语言，链表面试题

iotdog
关注 - 2
粉丝 - 3
+加关注

0 0

« 上一篇： C++基础之引用与指针的区别与联系、常引用使用时应注意的问题

posted @ 2017-07-03 23:36 iotdog 阅读(5917) 评论(1) 编辑 收藏 举报