阅读：13,558　　作者：解学武

静态链表基本操作（C语言实现）详解

上一节　　下一节

上节，我们初步创建了一个静态链表，本节学习有关静态链表的一些基本操作，包括对表中数据元素的添加、删除、查找和更改。

本节是建立在已能成功创建静态链表的基础上，因此我们继续使用上节中已建立好的静态链表学习本节内容，建立好的静态链表如图 1 所示：

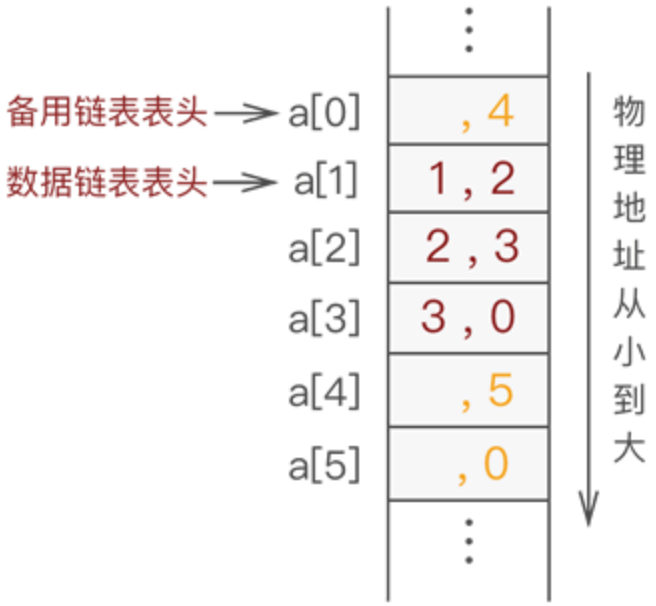

图 1 建立好的静态链表

静态链表添加元素

例如，在图 1 的基础，将元素 4 添加到静态链表中的第 3 个位置上，实现过程如下：

1. 从备用链表中摘除一个节点，用于存储元素 4；
2. 找到表中第 2 个节点（添加位置的前一个节点，这里是数据元素 2），将元素 2 的游标赋值给新元素 4；
3. 将元素 4 所在数组中的下标赋值给元素 2 的游标；

经过以上几步操作，数据元素 4 就成功地添加到了静态链表中，此时新的静态链表如图 2 所示：

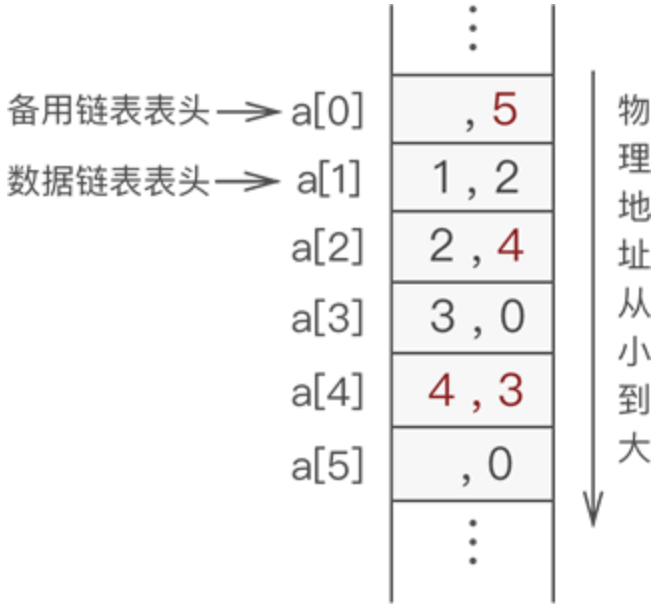

图 2 添加元素 4 的静态链表

由此，我们通过尝试编写 C 语言程序实现以上操作。读者可参考如下程序：

```
01.  //向链表中插入数据，body表示链表的头结点在数组中的位置，add表示插入元素的位置，num表示要插入的数据
02.  void insertArr(component * array, int body, int add, int num) {
03.      int tempBody = body;//tempBody做遍历结构体数组使用
04.      int i = 0, insert = 0;
05.      //找到要插入位置的上一个结点在数组中的位置
06.      for (i = 1; i < add; i++) {
07.          tempBody = array[tempBody].cur;
08.      }
09.      insert = mallocArr(array);//申请空间，准备插入
10.      array[insert].data = num;
11.
12.      array[insert].cur = array[tempBody].cur;//新插入结点的游标等于其直接前驱结点的游标
13.      array[tempBody].cur = insert;//直接前驱结点的游标等于新插入结点所在数组中的下标
14.  }
```

静态链表删除元素

静态链表中删除指定元素，只需实现以下 2 步操作：
1. 将存有目标元素的节点从数据链表中摘除；
2. 将摘除节点添加到备用链表，以便下次再用；

比较特殊的是，对于无头结点的数据链表来说，如果需要删除头结点，则势必会导致数据链表的表头不再位于数组下标为 1 的位置，换句话说，删除头结点之后，原数据链表中第二个结点将作为整个链表新的首元结点。

若问题中涉及大量删除元素的操作，建议读者在建立静态链表之初创建一个带有头节点的静态链表，方便实现删除链表中第一个数据元素的操作。

如下是针对无头结点的数据链表，实现删除操作的 C 语言代码：

```
//删除结点函数，num表示被删除结点中数据域存放的数据，函数返回新数据链表的表头位置
int deletArr(component * array, int body, int num) {
    int tempBody = body;
    int del = 0;
    int newbody = 0;
    //找到被删除结点的位置
    while (array[tempBody].data != num) {
        tempBody = array[tempBody].cur;
        //当tempBody为0时，表示链表遍历结束，说明链表中没有存储该数据的结点
        if (tempBody == 0) {
            printf("链表中没有此数据");
            return;
        }
    }
    //运行到此，证明有该结点
    del = tempBody;
    tempBody = body;
    //删除首元结点，需要特殊考虑
    if (del == body) {
        newbody = array[del].cur;
        freeArr(array, del);
        return newbody;
    }
    else
    {
        //找到该结点的上一个结点，做删除操作
        while (array[tempBody].cur != del) {
            tempBody = array[tempBody].cur;
        }
        //将被删除结点的游标直接给被删除结点的上一个结点
        array[tempBody].cur = array[del].cur;
        //回收被摘除节点的空间
        freeArr(array, del);
        return body;
    }
}
```

静态链表查找元素

静态链表查找指定元素，由于我们只知道静态链表第一个元素所在数组中的位置，因此只能通过逐个遍历静态链表的方式，查找存有指定数据元素的节点。

静态链表查找指定数据元素的 C 语言实现代码如下：

```
//在以body作为头结点的链表中查找数据域为elem的结点在数组中的位置
int selectNum(component * array, int body, int num) {
    //当游标值为0时，表示链表结束
    while (array[body].cur != 0) {
        if (array[body].data == num) {
            return body;
        }
        body = array[body].cur;
    }
    //判断最后一个结点是否符合要求
    if (array[body].data == num) {
        return body;
    }
    return -1;//返回-1，表示在链表中没有找到该元素
}
```

静态链表中更改数据

更改静态链表中的数据，只需找到目标元素所在的节点，直接更改节点中的数据域即可。

实现此操作的 C 语言代码如下：

```
//在以body作为头结点的链表中将数据域为oldElem的结点，数据域改为newElem
void amendElem(component * array, int body, int oldElem, int newElem) {
    int add = selectNum(array, body, oldElem);
    if (add == -1) {
        printf("无更改元素");
        return;
    }
    array[add].data = newElem;
}
```

总结

这里给出以上对静态链表做 "增删查改" 操作的完整实现代码：

```
#include <stdio.h>
#define maxSize 7
typedef struct {
    int data;
    int cur;
}component;
//将结构体数组中所有分量链接到备用链表中
void reserveArr(component *array);
//初始化静态链表
int initArr(component *array);
```

```c
//向链表中插入数据，body表示链表的头结点在数组中的位置，add表示插入元素的位置，num表示要插入的数据
void insertArr(component * array, int body, int add, int num);
//删除链表中存有num的结点,返回新数据链表中第一个节点所在的位置
int deletArr(component * array, int body, int num);
//查找存储有num的结点在数组的位置
int selectNum(component * array, int body, int num);
//将链表中的字符oldElem改为newElem
void amendElem(component * array, int body, int oldElem, int newElem);
//输出函数
void displayArr(component * array, int body);
//从备用链表中摘除空闲节点的实现函数
int mallocArr(component * array);
//将摘除下来的节点链接到备用链表上
void freeArr(component * array, int k);

int main() {
    component array[maxSize];
    int body = initArr(array);
    int selectAdd;
    printf("静态链表为: \n");
    displayArr(array, body);

    printf("在第3的位置上插入元素4:\n");
    insertArr(array, body, 3, 4);
    displayArr(array, body);

    printf("删除数据域为1的结点:\n");
    body = deletArr(array, body, 1);
    displayArr(array, body);

    printf("查找数据域为4的结点的位置:\n");
    selectAdd = selectNum(array, body, 4);
    printf("%d\n", selectAdd);
    printf("将结点数据域为4改为5:\n");
    amendElem(array, body, 4, 5);
    displayArr(array, body);
    return 0;
}
//创建备用链表
void reserveArr(component *array) {
    int i = 0;
    for (i = 0; i < maxSize; i++) {
        array[i].cur = i + 1;//将每个数组分量链接到一起
    }
    array[maxSize - 1].cur = 0;//链表最后一个结点的游标值为0
}

```

```
58. //初始化静态链表
59. int initArr(component *array) {
60.     int tempBody = 0, body = 0;
61.     int i = 0;
62.     reserveArr(array);
63.     body = mallocArr(array);
64.     //建立首元结点
65.     array[body].data = 1;
66.     array[body].cur = 0;
67.     //声明一个变量，把它当指针使，指向链表的最后的一个结点，当前和首元结点重合
68.     tempBody = body;
69.     for (i = 2; i < 4; i++) {
70.         int j = mallocArr(array); //从备用链表中拿出空闲的分量
71.         array[j].data = i;       //初始化新得到的空间结点
72.         array[tempBody].cur = j; //将新得到的结点链接到数据链表的尾部
73.         tempBody = j;              //将指向链表最后一个结点的指针后移
74.     }
75.     array[tempBody].cur = 0;//新的链表最后一个结点的指针设置为0
76.     return body;
77. }
78.
79. //向链表中插入数据，body表示链表的头结点在数组中的位置，add表示插入元素的位置，num表示要插入的数据
80. void insertArr(component * array, int body, int add, int num) {
81.     int tempBody = body;//tempBody做遍历结构体数组使用
82.     int i = 0, insert = 0;
83.     //找到要插入位置的上一个结点在数组中的位置
84.     for (i = 1; i < add; i++) {
85.         tempBody = array[tempBody].cur;
86.     }
87.     insert = mallocArr(array);//申请空间，准备插入
88.     array[insert].data = num;
89.
90.     array[insert].cur = array[tempBody].cur;//新插入结点的游标等于其直接前驱结点的游标
91.     array[tempBody].cur = insert;//直接前驱结点的游标等于新插入结点所在数组中的下标
92. }
93.
94. //删除结点函数，num表示被删除结点中数据域存放的数据
95. int deletArr(component * array, int body, int num) {
96.     int tempBody = body;
97.     int del = 0;
98.     int newbody = 0;
99.     //找到被删除结点的位置
100.    while (array[tempBody].data != num) {
101.        tempBody = array[tempBody].cur;
102.        //当tempBody为0时，表示链表遍历结束，说明链表中没有存储该数据的结点
103.        if (tempBody == 0) {
104.            printf("链表中没有此数据");
```

```
105.                 return;
106.             }
107.         }
108.         //运行到此，证明有该结点
109.         del = tempBody;
110.         tempBody = body;
111.         //删除首元结点，需要特殊考虑
112.         if (del == body) {
113.             newbody = array[del].cur;
114.             freeArr(array, del);
115.             return newbody;
116.         }
117.         else
118.         {
119.             //找到该结点的上一个结点，做删除操作
120.             while (array[tempBody].cur != del) {
121.                 tempBody = array[tempBody].cur;
122.             }
123.             //将被删除结点的游标直接给被删除结点的上一个结点
124.             array[tempBody].cur = array[del].cur;
125.             //回收被摘除节点的空间
126.             freeArr(array, del);
127.             return body;
128.         }
129.     }
130.
131.     //在以body作为头结点的链表中查找数据域为elem的结点在数组中的位置
132.     int selectNum(component * array, int body, int num) {
133.         //当游标值为0时，表示链表结束
134.         while (array[body].cur != 0) {
135.             if (array[body].data == num) {
136.                 return body;
137.             }
138.             body = array[body].cur;
139.         }
140.         //判断最后一个结点是否符合要求
141.         if (array[body].data == num) {
142.             return body;
143.         }
144.         return -1;//返回-1，表示在链表中没有找到该元素
145.     }
146.
147.     //在以body作为头结点的链表中将数据域为oldElem的结点，数据域改为newElem
148.     void amendElem(component * array, int body, int oldElem, int newElem) {
149.         int add = selectNum(array, body, oldElem);
150.         if (add == -1) {
151.             printf("无更改元素");
```

```
152.          return;
153.      }
154.      array[add].data = newElem;
155.  }
156.
157.  void displayArr(component * array, int body) {
158.      int tempBody = body;//tempBody准备做遍历使用
159.      while (array[tempBody].cur) {
160.          printf("%d,%d ", array[tempBody].data, array[tempBody].cur);
161.          tempBody = array[tempBody].cur;
162.      }
163.      printf("%d,%d\n", array[tempBody].data, array[tempBody].cur);
164.
165.  }
166.
167.  //提取分配空间
168.  int mallocArr(component * array) {
169.      //若备用链表非空，则返回分配的结点下标，否则返回0（当分配最后一个结点时，该结点的游标值为0）
170.      int i = array[0].cur;
171.      if (array[0].cur) {
172.          array[0].cur = array[i].cur;
173.      }
174.      return i;
175.  }
176.
177.  //备用链表回收空间的函数，其中array为存储数据的数组，k表示未使用节点所在数组的下标
178.  void freeArr(component * array, int k) {
179.      array[k].cur = array[0].cur;
180.      array[0].cur = k;
181.  }
```

程序运行结果为：

```
静态链表为：
1,2 2,3 3,0
在第3的位置上插入元素4:
1,2 2,3 3,4 4,0
删除数据域为1的结点:
2,3 3,4 4,0
查找数据域为4的结点的位置:
4
将结点数据域为4改为5:
2,3 3,4 5,0
```

联系方式　购买教程（带答疑）