

# 顺序表的基本操作及C语言实现 (详解版)

我们学习了[顺序表](#)及初始化的过程，本节学习有关顺序表的一些基本操作，以及如何使用 C 语言实现它们。

## 顺序表插入元素

向已有顺序表中插入数据元素，根据插入位置的不同，可分为以下 3 种情况：

- 1. 插入到顺序表的表头；
- 2. 在表的中间位置插入元素；
- 3. 尾随顺序表中已有元素，作为顺序表中的最后一个元素；

虽然数据元素插入顺序表中的位置有所不同，但是都使用的是同一种方式去解决，即：通过遍历，找到数据元素要插入的位置，然后做如下两步工作：

- 将要插入位置元素以及后续的元素整体向后移动一个位置；
- 将元素放到腾出来的位置上；

例如，在 {1,2,3,4,5} 的第 3 个位置上插入元素 6，实现过程如下：

- 遍历至顺序表存储第 3 个数据元素的位置，如[图 1](#) 所示：



图 1 找到目标元素位置

- 将元素 3 以及后续元素 4 和 5 整体向后移动一个位置，如图 2 所示：

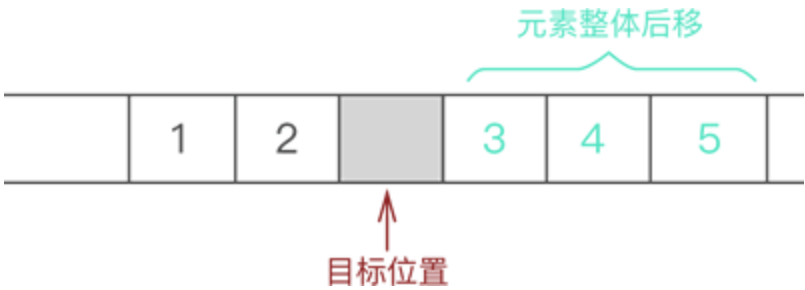


图 2 将插入位置腾出

- 将新元素 6 放入腾出的位置，如图 3 所示：



图 3 插入目标元素

因此，顺序表插入数据元素的 C 语言实现代码如下：

```
01. //插入函数，其中，elem为插入的元素，add为插入到顺序表的位置
02. table addTable(table t, int elem, int add)
03. {
04.     int i;
05.     //判断插入本身是否存在问题（如果插入元素位置比整张表的长度+1还大（如果相等，是尾随的情况），或者插入
06.     if (add > t.length + 1 || add < 1) {
07.         printf("插入位置有问题");
08.         return t;
09.     }
10.     //做插入操作时，首先需要看顺序表是否有多余的存储空间提供给插入的元素，如果没有，需要申请
11.     if (t.length == t.size) {
12.         t.head = (int *)realloc(t.head, (t.size + 1) * sizeof(int));
13.         if (!t.head) {
14.             printf("存储分配失败");
15.             return t;
16.         }
17.         t.size += 1;
18.     }
19.     //插入操作，需要将插入位置开始的后续元素，逐个后移
20.     for (i = t.length - 1; i >= add - 1; i--) {
21.         t.head[i + 1] = t.head[i];
22.     }
23.     //后移完成后，直接将所需插入元素，添加到顺序表的相应位置
24.     t.head[add - 1] = elem;
25.     //由于添加了元素，所以长度+1
26.     t.length++;
27.     return t;
28. }
```

注意，动态数组额外申请更多物理空间使用的是 `realloc` 函数。并且，在实现后续元素整体后移的过程，目标位置其实是有数据的，还是 3，只是下一步新插入元素时会把旧元素直接覆盖。

## 顺序表删除元素

从顺序表中删除指定元素，实现起来非常简单，只需找到目标元素，并将其后续所有元素整体前移 1 个位置即可。

后续元素整体前移一个位置，会直接将目标元素删除，可间接实现删除元素的目的。

例如，从 {1,2,3,4,5} 中删除元素 3 的过程如图 4 所示：



图 4 顺序表删除元素的过程示意图

因此，顺序表删除元素的 C 语言实现代码为：

```
01. table delTable(table t, int add) {
02.     int i;
03.     if (add > t.length || add < 1) {
04.         printf("被删除元素的位置有误");
05.         exit(0);
06.     }
07.     //删除操作
08.     for (i = add; i < t.length; i++) {
09.         t.head[i - 1] = t.head[i];
10.     }
11.     t.length--;
12.     return t;
13. }
```

## 顺序表查找元素

顺序表中查找目标元素，可以使用多种查找算法实现，比如说[二分查找](#)算法、插值查找算法等。

这里，我们选择[顺序查找](#)算法，具体实现代码为：

```
01. //查找函数，其中，elem表示要查找的数据元素的值
02. int selectTable(table t, int elem) {
03.     int i;
04.     for (i = 0; i < t.length; i++) {
05.         if (t.head[i] == elem) {
06.             return i + 1;
        }
```

```

07.         }
08.     }
09.     return -1; //如果查找失败, 返回-1
10. }

```

## 顺序表更改元素

顺序表更改元素的实现过程是：

1. 找到目标元素；
2. 直接修改该元素的值；

顺序表更改元素的 C 语言实现代码为：

```

01. //更改函数, 其中, elem为要更改的元素, newElem为新的数据元素
02. table amendTable(table t, int elem, int newElem) {
03.     int add = selectTable(t, elem);
04.     t.head[add - 1] = newElem; //由于返回的是元素在顺序表中的位置, 所以-1就是该元素在数组中的下标
05.     return t;
06. }

```

以上是顺序表使用过程中最常用的基本操作，这里给出本节完整的实现代码：

```

01. #include <stdio.h>
02. #include <stdlib.h>
03. #define Size 5
04. typedef struct Table {
05.     int * head;
06.     int length;
07.     int size;
08. }table;
09. table initTable() {
10.     table t;
11.     t.head = (int*)malloc(Size * sizeof(int));
12.     if (!t.head)
13.     {
14.         printf("初始化失败");
15.         exit(0);
16.     }
17.     t.length = 0;
18.     t.size = Size;
19.     return t;
20. }
21. table addTable(table t, int elem, int add)
22. {
23.     int i;

```

```
24.     if (add > t.length + 1 || add < 1) {
25.         printf("插入位置有问题");
26.         return t;
27.     }
28.     if (t.length >= t.size) {
29.         t.head = (int *)realloc(t.head, (t.size + 1) * sizeof(int));
30.         if (!t.head) {
31.             printf("存储分配失败");
32.         }
33.         t.size += 1;
34.     }
35.     for (i = t.length - 1; i >= add - 1; i--) {
36.         t.head[i + 1] = t.head[i];
37.     }
38.     t.head[add - 1] = elem;
39.     t.length++;
40.     return t;
41. }
42. table delTable(table t, int add) {
43.     int i;
44.     if (add > t.length || add < 1) {
45.         printf("被删除元素的位置有误");
46.         exit(0);
47.     }
48.     for (i = add; i < t.length; i++) {
49.         t.head[i - 1] = t.head[i];
50.     }
51.     t.length--;
52.     return t;
53. }
54. int selectTable(table t, int elem) {
55.     int i;
56.     for (i = 0; i < t.length; i++) {
57.         if (t.head[i] == elem) {
58.             return i + 1;
59.         }
60.     }
61.     return -1;
62. }
63. table amendTable(table t, int elem, int newElem) {
64.     int add = selectTable(t, elem);
65.     t.head[add - 1] = newElem;
66.     return t;
67. }
68. void displayTable(table t) {
69.     int i;
70.     for (i = 0; i < t.length; i++) {
```

```

71.         printf("%d", t.head[i]);
72.     }
73.     printf("\n");
74. }
75. int main() {
76.     int i, add;
77.     table t1 = initTable();
78.     for (i = 1; i <= Size; i++) {
79.         t1.head[i - 1] = i;
80.         t1.length++;
81.     }
82.     printf("原顺序表: \n");
83.     displayTable(t1);
84.
85.     printf("删除元素1:\n");
86.     t1 = delTable(t1, 1);
87.     displayTable(t1);
88.
89.     printf("在第2的位置插入元素5:\n");
90.     t1 = addTable(t1, 5, 2);
91.     displayTable(t1);
92.
93.     printf("查找元素3的位置:\n");
94.     add = selectTable(t1, 3);
95.     printf("%d\n", add);
96.
97.     printf("将元素3改为6:\n");
98.     t1 = amendTable(t1, 3, 6);
99.     displayTable(t1);
100.    return 0;
101. }

```

程序运行结果为：

原顺序表：

1 2 3 4 5

删除元素1：

2 3 4 5

在第2的位置插入元素5：

2 5 3 4 5

查找元素3的位置：

3

将元素3改为6：

2 5 6 4 5

[联系方式](#)   [购买教程（带答疑）](#)