

链栈基本操作 (入栈和出栈) C语言详解

链栈，即用链表实现栈存储结构。

链栈的实现思路同顺序栈类似，顺序栈是将数顺序表（数组）的一端作为栈底，另一端为栈顶；链栈也如此，通常我们将链表的头部作为栈顶，尾部作为栈底，如图 1 所示：

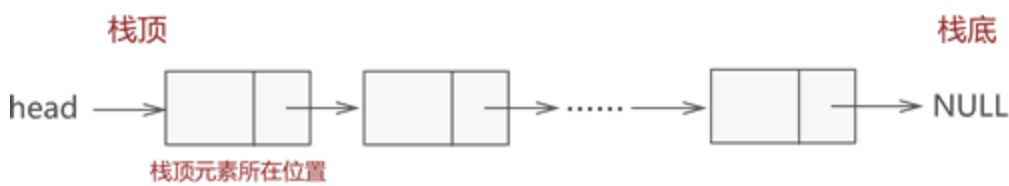


图 1 链栈示意图

将链表头部作为栈顶的一端，可以避免在实现数据 "入栈" 和 "出栈" 操作时做大量遍历链表的耗时操作。

链表的头部作为栈顶，意味着：

- 在实现数据"入栈"操作时，需要将数据从链表的头部插入；
- 在实现数据"出栈"操作时，需要删除链表头部的首元节点；

因此，链栈实际上就是一个只能采用头插法插入或删除数据的链表。

链栈元素入栈

例如，将元素 1、2、3、4 依次入栈，等价于将各元素采用头插法依次添加到链表中，每个数据元素的添加过程如图 2 所示：



图 2 链栈元素依次入栈过程示意图

C语言实现代码为：

```

01. //链表中的节点结构
02. typedef struct lineStack{
03.     int data;
04.     struct lineStack * next;
05. }lineStack;
06. //stack为当前的链栈, a表示入栈元素
07. lineStack* push(lineStack * stack,int a){
08.     //创建存储新元素的节点
09.     lineStack * line=(lineStack*)malloc(sizeof(lineStack));
10.     line->data=a;
11.     //新节点与头节点建立逻辑关系
12.     line->next=stack;
13.     //更新头指针的指向
14.     stack=line;
15.     return stack;
16. }

```

链栈元素出栈

例如, 图 2e) 所示的链栈中, 若要将元素 3 出栈, 根据"先进后出"的原则, 要先将元素 4 出栈, 也就是从链表中摘除, 然后元素 3 才能出栈, 整个操作过程如图 3 所示:

a) 初始链栈: head → 4 → 3 → 2 → 1 → NULL
 b) 元素 4 出栈: head → 3 → 2 → 1 → NULL
 c) 元素 3 出栈: head → 2 → 1 → NULL

图 3 链栈元素出栈示意图

因此, 实现栈顶元素出链栈的 C 语言实现代码为:

```

01. //栈顶元素出链栈的实现函数
02. lineStack * pop(lineStack * stack){
03.     if (stack) {
04.         //声明一个新指针指向栈顶节点
05.         lineStack * p=stack;
06.         //更新头指针
07.         stack=stack->next;
08.         printf("出栈元素: %d ",p->data);
09.         if (stack) {
10.             printf("新栈顶元素: %d\n",stack->data);
11.         }else{
12.             printf("栈已空\n");
13.         }
14.         free(p);
15.     }else{

```

```

16.         printf("栈内没有元素");
17.         return stack;
18.     }
19.     return stack;
20. }

```

代码中通过使用 if 判断语句，避免了用户执行"栈已空却还要数据出栈"错误操作。

总结

本节，通过采用头插法操作数据的单链表实现了链栈结构，这里给出链栈及基本操作的C语言完整代码：

```

01.  #include <stdio.h>
02.  #include <stdlib.h>
03.  typedef struct lineStack{
04.      int data;
05.      struct lineStack * next;
06.  }lineStack;
07.  lineStack* push(lineStack * stack,int a){
08.      lineStack * line=(lineStack*)malloc(sizeof(lineStack));
09.      line->data=a;
10.      line->next=stack;
11.      stack=line;
12.      return stack;
13.  }
14.  lineStack * pop(lineStack * stack){
15.      if (stack) {
16.          lineStack * p=stack;
17.          stack=stack->next;
18.          printf("弹栈元素: %d ",p->data);
19.          if (stack) {
20.              printf("栈顶元素: %d\n",stack->data);
21.          }else{
22.              printf("栈已空\n");
23.          }
24.          free(p);
25.      }else{
26.          printf("栈内没有元素");
27.          return stack;
28.      }
29.      return stack;
30.  }
31.  int main() {
32.      lineStack * stack=NULL;
33.      stack=push(stack, 1);
34.      stack=push(stack, 2);
35.      stack=push(stack, 3);

```

```
36.     stack=push(stack, 4);  
37.     stack=pop(stack);  
38.     stack=pop(stack);  
39.     stack=pop(stack);  
40.     stack=pop(stack);  
41.     stack=pop(stack);  
42.     return 0;  
43. }
```

程序运行结果为：

弹栈元素：4 栈顶元素：3

弹栈元素：3 栈顶元素：2

弹栈元素：2 栈顶元素：1

弹栈元素：1 栈已空

栈内没有元素

[< 上一节](#)

[下一节 >](#)

[联系方式](#) [购买教程（带答疑）](#)