

栈

一、栈 List In First Out二、顺序栈的表示和实现三、链栈的表示和实现3.1链栈的表示3.2链栈的实现四、链栈和顺序栈的总结

一、栈 List In First Out

栈（Stack）是只允许在一端进行插入或删除操作的线性表。

LIFO 特点：后进先出

1. 只能从 $n+1$ 位置插入

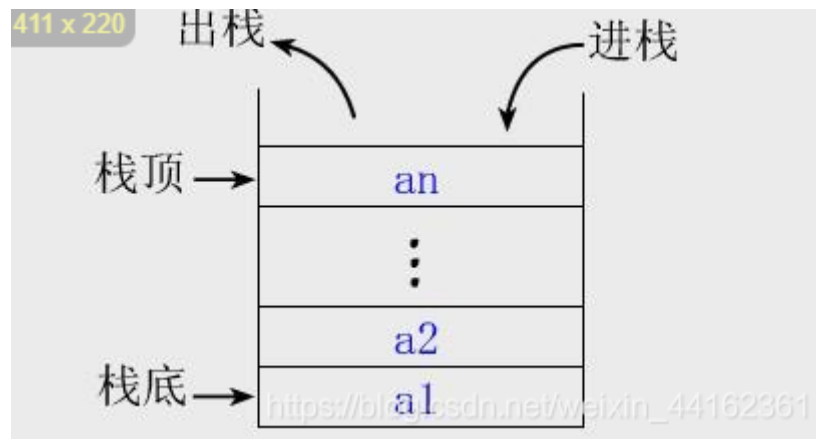
2. 只能从 n 位置删除

- 栈顶Top：线性表允许插入和删除的那一端。
- 栈底Bottom：固定的，不允许进行插入和删除的另一端。

定义：特殊的线性表，限定仅在一端进行插入和删除操作的线性表

针对表尾进行插入和删除的线性表

表尾 a_n 为栈顶top， a_1 为栈底base



基本概括

- 插入元素到top为入栈 push 压入
- 从栈顶删除一个元素为出栈 pop 弹出
- 插入和删除都在栈顶执行 也就是表尾

性质：

限定只能在表的一端进行插入和删除运算的线性表

- 逻辑结构： 与线性表相同
- 存储结构： 用顺序栈或链栈存储均可，但以顺序栈更常见
- 运算规则： 只能在栈顶运算，且访问结点时依照后进先出LIFO的原则
- 实现方式： 关键就是入栈和出栈函数

与线性表区别： 仅在于运算规则不同

线性表： 随机存储

栈： 后进先出

案例： 进制 八皇后 迷宫问题 递归

基本操作

- 初始化InitStack(&S);
 - 判空Empty(S);
 - 进栈Push(&S, x);
 - 出栈Pop(&S, &x);
 - 读栈顶元素GetTop(S);
 - 遍历栈PrintStack(&S);
 - 销毁栈DestroyStack(&S);
 - 栈置空操作clear（栈本身还在，但是里面没有元素）
- 入栈 push 出栈 pop

存储结构： 栈就是操作受限的线性表

1. 按顺序结构存储的就是顺序栈
2. 按链表存储的就是链栈

二、顺序栈的表示和实现

一组地址连续的存储单元一次存放栈底到栈顶的数据元素 栈底一般在低地址端

描述:

- **top**指针 指向真正的栈顶元素之上的下标地址
- **base**指针 指向栈底元素的顺序栈的位置
- **stacksize** 栈的大小 可使用的最大容量

标志:

- 空栈: **base == top** 是栈空标志
- 栈满: **top - base == stacksize**
- 栈溢出 **top - base > stacksize**
- 解决: 1.报错 2.额外分配更大的空间,作为栈顶存储空间,将原栈的内容移入新栈
- 栈下溢 **top - base == stacksize** 没有元素溢出
- 数组作为顺序栈容易溢出
- 1.overflow 栈已经满
- 2.underflow 栈已经空
- 上溢是一种错误,下溢是一种结束条件

代码实现:

1. 结构体

```
1 typedef struct stack
2 {
3     selemtype* base; //栈底指针
4     selemtype* top;  //栈顶指针
5     int stacksize;   //栈容量
6 }st;
```

1. 初始化

```

1 //初始化 内存中开辟一段空间来使用 top == base 都指向栈底
2 void initstack(st *s)
3 {
4     s->top = (int*)malloc( MAXSIZE*sizeof(int)); //分配空间 大小为100 指向这个空间的首元素
5     if (s->top == NULL)
6     {
7         printf("开辟失败");
8         exit(-1);
9     }
10    s->base = s->top;
11    s->stacksize = MAXSIZE;
12    printf("栈已经开辟成功!\n");
13 }

```

2. 判断为空?

```

1 //顺序栈是否为空
2 bool isempty(st *s)
3 {
4     if (s->base == s->top)
5     {
6         return true;
7         printf("栈空! ");
8     }
9     else
10    {
11        return false;
12        printf("栈回收! ");
13    }
14 }

```

3. len长

```

1 int lenstack(st *s)
2 {
3     return (s->top - s->base);
4 }

```

4. 清空

```

1 //清空顺序栈
2 void qkstack(st *s)
3 {
4     if (s->base)
5     {
6         s->top = s->base;
7         printf("栈已经清空了! ");
8     }
9 }

```

4. 销毁

```

1 void destoryst(st *s)
2 {
3     if (s->base)
4     {
5         free(s->base);
6         s->stacksize = 0;
7         s->base = s->top = NULL;
8         printf("栈已经释放了! 收回内存");
9     }
10 }

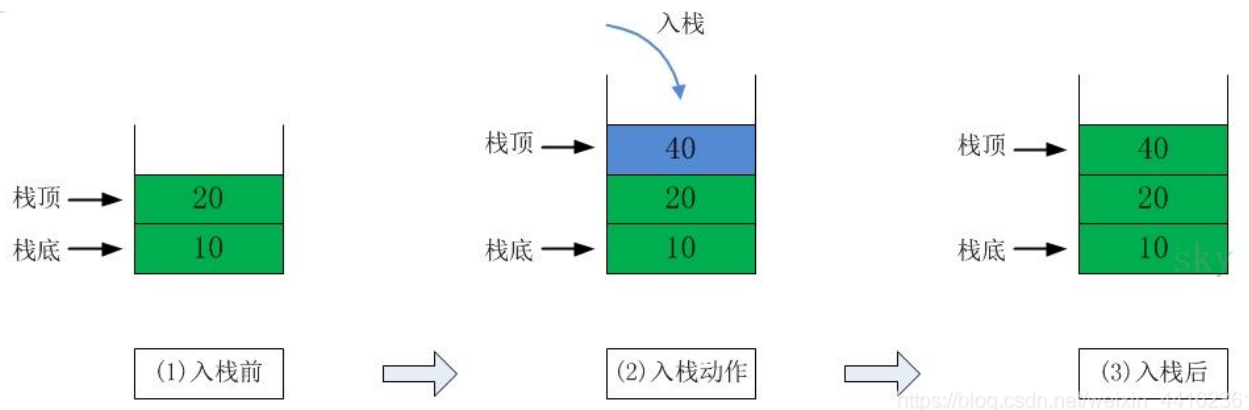
```

5. 入栈

```

1 void pushstack(st *s,selemtype x)
2 {
3     if ((lenstack(s)) == s->stacksize)
4     {
5         exit(-1);
6     }
7     else
8     {
9         *(s->top) = x; //简引用
10        s->top++;
11        /*(s->top)++ = x;
12    }
13    printf("%d ", x);
14 }

```

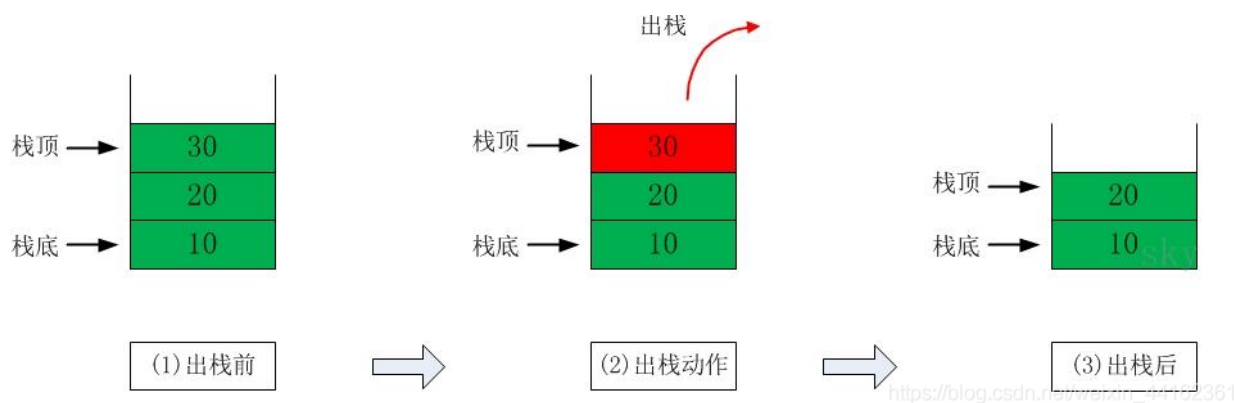


6. 出栈

```

1 void popstack(st* s)
2 {
3     int x;
4     if (isempty(s))
5     {
6         printf("栈空! ");
7         exit(-1);
8     }
9     else
10    {
11        --(s->top);
12        x = *(s->top);
13        printf("%d ", x);
14    }
15 }

```



详细代码：

```

1 #define _CRT_SECURE_NO_WARNINGS 1
2 /*加油! */
3
4
5 #include "test.h"
6
7 //两种存储结构 但是操作受限的线性表
8 /*

```

```

9      * 1.顺序栈
10     * 2.链栈
11     */
12    //顺序栈的表示和实现 一组地址连续的存储的一次存放栈底到栈顶的数据元素 栈底一般在低地址端
13
14    #define MAXSIZE 100
15    typedef int selemtype;
16
17    typedef struct Node
18    {
19        int data;
20        struct Node* pNext;
21    } NODE, * PNODE;
22    typedef struct stack
23    {
24        selemtype* base;//栈底指针
25        selemtype* top;//栈顶指针
26        int stacksize;//栈容量
27    }st;
28
29    //初始化 内存中开辟一段空间来使用 top == base 都指向栈底
30    void initstack(st *s)
31    {
32        s->top = (int*)malloc( MAXSIZE*sizeof(int));//分配空间 大小为100 指向这个空间的首
元素
33        if (s->top == NULL)
34        {
35            printf("开辟失败");
36            exit(-1);
37        }
38        s->base = s->top;
39        s->stacksize = MAXSIZE;
40        printf("栈已经开辟成功!\n");
41    }
42
43    //顺序栈是否为空
44    bool isempty(st *s)
45    {
46        if (s->base == s->top)
47        {
48            return true;
49            printf("栈空! ");
50        }
51        else
52        {
53            return false;
54            printf("栈回收! ");
55        }
56    }
57    //求栈len
58    int lenstack(st *s)

```

```
59 {
60     return (s->top - s->base);
61 }
62 //清空顺序栈
63 void qkstack(st *s)
64 {
65     if (s->base)
66     {
67         s->top = s->base;
68         printf("栈已经清空了! ");
69     }
70 }
71 //销毁栈
72 void destoryst(st *s)
73 {
74     if (s->base)
75     {
76         free(s->base);
77         s->stacksize = 0;
78         s->base = s->top = NULL;
79         printf("栈已经释放了! 收回内存");
80     }
81 }
82 //入栈
83 void pushstack(st *s,selemtype x)
84 {
85     if ((lenstack(s)) == s->stacksize)
86     {
87         exit(-1);
88     }
89     else
90     {
91         *(s->top) = x;//简引用
92         s->top++;
93         /*(s->top)++ = x;
94     }
95     printf("%d ", x);
96 }
97 //出栈
98 void popstack(st* s)
99 {
100     int x;
101     if (isempty(s))
102     {
103         printf("栈空! ");
104         exit(-1);
105     }
106     else
107     {
108         --(s->top);
109         x = *(s->top);
```



```
110         printf("%d ", x);
111     }
112 }
113
114 int Find(st*s,int y)
115 {
116     if (isempty(s))
117     {
118         printf("栈空! ");
119         exit(-1);
120     }
121     for (int i = 0;i<lenstack(s);i++)
122     {
123         if (*(s->top) == y)
124         {
125             return i;
126         }
127         s->top++;
128     }
129     return -1;
130 }
131 int main()
132 {
133     st s1;
134     selemtype n;
135     printf("初始化: \n");
136     initstack(&s1);
137     printf("请向栈中存入数据:\n");
138     for (int i = 0;i<10;i++)
139     {
140         scanf("%d", &n);
141         pushstack(&s1, n);
142     }
143     printf("入栈成功! \n");
144     printf("栈len:\n");
145     int len = lenstack(&s1);
146     printf("len = %d\n", len);
147     printf("出栈: \n");
148     for (int i = 0; i < 10; i++)
149     {
150         popstack(&s1);
151     }
152     destoryst(&s1);
153 }
```

```
初始化：
栈已经开辟成功！
入栈个数：
8
请向栈中存入数据：
231 423 423 4241 12 32 12 1
231 423 423 4241 12 32 12 1 入栈成功！
栈len：
len = 8
出栈：
1 12 32 12 4241 423 423 231 栈已经释放了！ 收回内存
```

三、链栈的表示和实现

3.1 链栈的表示

1. 关于链栈的操作，它和[链表](#)有很大的相似之处，不同的是，栈只允许在栈顶进行入栈操作，和链表的头插法相同。

2. 之所以要用链栈，是因为如果我们用[顺序栈](#)的话，需要提前申请一片内存空间，但是如果我们值存入少量的元素，那么这片内存空间难免会造成一定的浪费。如果使用链栈的话，我们只在入栈的时候进行内存的申请，然后再进行元素的存储，既可以进行动态的内存申请。根据实际入栈元素的多少申请所需的内存即可。

3. 链栈和顺序栈的基本操作都是一样的，包括：初始化，求长度，判断栈是否为空，入栈，出栈，取栈顶元素。

3.2 链栈的实现

1. 先定义一个结构体，这个结构体中，包括栈中每个节点的节点值data，和指向下一个栈节点的指针next

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  typedef int ElemType;
4  //定义一个结构体栈
5  typedef struct StackNode
6  {
7      ElemType data;
8      struct StackNode *next;
9  }StackList;
10 typedef StackList *LStack; //声明一个指向这个结构体的指针类型Lstack
```

2.进行初始化，将链栈的顶处设置一个指针，用来指向这个栈，这个指针内不存储东西，那么刚开始就把这个栈顶指针设置为NULL即可。

```
1 //初始化链栈
2 void Init(LStack *s)
3 {
4     (*s)=(LStack)malloc(sizeof(StackList));
5     //p = (snode*)malloc(sizeof(snode)*MAXSIZE);
6     (*s)->next=NULL;
7 }
```

3.判断栈是否为空，只需要判断栈顶指针的指针域是否为空即可。为空返回0，不为空返回1。

```
1 //判断栈是否为空
2 int Empty(LStack s)
3 {
4     if(s->next==NULL)
5     {
6         return 0; //为空返回0,
7     }
8     return 1;    //不为空返回1,
9 }
```

4.求栈的长度，即从栈顶的下一个，也就是第一个元素开始，往下遍历，一直遍历到指针指向空，说明遍历完了，没遍历一个，就计数器加一，临时的指针变量往后移动一位p=p->next

```
1 //求栈中的元素数量，即长度
2 int Printf(LStack s)
3 {
4     int count=0;
5     LStack p;
6     p=s->next; //指向第一个元素
7     while(p){
8         count++;
9         p=p->next;
10    }
11    return count;
12 }
```

5.入栈，将值x，压入到栈中，我们首先声明一个指针变量，来存储这个元素值x，当然，因为我们要用这个指针变量存储东西，所以我们要先为这个指针变量申请内存空间malloc，然后存入。

```

1 //入栈
2 void Push(LStack *s,ElemType x)
3 {
4     LStack p;//声明一个指针变量
5     p=(LStack)malloc(sizeof(StackList));//为这个指针变量分配空间
6     p->data=x;          //将入站的元素值存入到这个指针的数据域
7     p->next=(*s)->next; //这两个步骤和顺序链表的操作相似。
8     (*s)->next=p;
9 }

```

6.出栈，首先要判断栈是否为空，如果不为空，我们指针p指向栈中的第一个节点，然后用e记录下当前栈顶元素的值，接着将不存东西的栈顶指针的指针域指向指针p的下一个节点处，释放指针p即可

```

1 //出栈
2 ElemType Pop(LStack *s)
3 {
4     LStack p;//声明一个结构体类型的指针变量
5     ElemType e;
6     p=(*s)->next;//指向栈中的栈顶元素
7     if(p==NULL)
8     {
9         return 0;
10    }
11    (*s)->next=p->next;//不为空的话就让栈顶元素变成当前栈顶p的下一个
12    e=p->data;//存放以下删除的栈顶元素
13    free(p); //释放指针
14    return e; //返回删除的栈顶指针的值
15 }

```

7.获取栈顶元素的值，首先要判断栈是否为空，如果不为空，直接返回栈中的第一个节点的值即可

```

1 //取栈顶元素的值
2 ElemType GetTop(LStack s)
3 {
4     if(Empty(s)){//如果栈顶元素不为空的话，则打印出栈顶元素
5         return s->next->data;
6     }
7     return 0;
8 }

```

8.main

```

1 int main()
2 {
3     LStack s;

```

```

4     int x;
5     ElemType e;
6     //初始化栈
7     Init(&s);
8     //判断栈是否为空
9     x=Empty(s);
10    if(x)
11    {
12        printf("栈空\n");
13    }
14    //扫描要入站的元素
15    Push(&s,5);
16    Push(&s,4);
17    Push(&s,6);
18    Push(&s,8);
19    Push(&s,2);
20    //获取栈的长度
21    x=Printf(s);
22    printf("栈的长度为%d\n",x);
23    //获取当前栈的栈顶元素
24    e=GetTop(s);
25    if(e){//如果栈不为空的话输出
26        printf("当前栈顶元素为: %d\n",e);
27    }
28    //删除当前栈的栈顶元素
29    e=Pop(&s);
30    if(x)//如果栈不为空的话进行出栈操作
31    {
32        printf("删除的元素为: %d\n",e);
33    }
34    //获取当前的栈顶元素
35    e=GetTop(s);
36    if(e){//如果栈不为空的话输出
37        printf("当前栈顶元素为: %d\n",e);
38    }
39    x=Printf(s);
40    printf("栈的长度为%d\n",x);
41    return 0;
42 }

```

```

1  //define _CRT_SECURE_NO_WARNINGS 1
2  /**加油! */
3  #include "test.h"
4  #define MAXSIZE 100
5  typedef int selemtype;
6
7  //通过链表表示栈 -- 链栈  栈的结点类型
8  /*

```

```

9      * 链栈的指针方向是前驱元素
10     *
11     * 1. 链表的头指针就是栈顶
12     * 2. 链栈不需要头节点
13     * 3. 基本不存在栈满的情况
14     * 4. 空栈相当于头指针指向空
15     * 5. 插入和删除仅在栈顶处执行
16     *
17     */
18     typedef struct snode
19     {
20         selemtype data;
21         struct snode* next;
22     }snode,*ls;
23
24     //初始化 就是开辟空间往里面存数据
25     void init(snode *s)
26     {
27         s = NULL;
28     }
29
30     bool isempty(snode *s)
31     {
32         if (s==NULL)
33         {
34             return false;
35         }
36         else
37         {
38             return true;
39         }
40     }
41     void push(snode *s,int x)
42     {
43         //栈里插入元素 栈顶 先找出一个空间 指针p指向给它 给数据域赋值 就成为了栈顶元素 在指向s
44         //的next域 然后在修改栈顶的指针p 然后s的链栈就创建完成
45         snode* p;
46         p = (snode*)malloc(sizeof(snode)*MAXSIZE);
47         if (p==NULL)
48         {
49             printf("结点开辟失败! ");
50             exit(-1);
51         }
52         //将数据给p指向的data域
53         p->data = x;
54         //将p的next指向s的栈顶
55         p->next = s;
56         //修改栈顶指针
57         s = p;
58     }

```

```
59
60 void pop(snode* s)
61 {
62     snode* p;
63     int x;
64     p = (snode*)malloc(sizeof(snode));
65     if (s == NULL)
66     {
67         printf("栈空! ");
68     }
69     x = p->data;
70     p = s;
71     s = s->next;
72     free(p);
73
74 }
75 //取栈顶元素
76 int gettop(snode*s)
77 {
78     if (s!= NULL)
79     {
80         return s->data;
81     }
82     return 1;
83 }
84 int main()
85 {
86
87     snode s;
88     printf("链栈初始化\n");
89     init(&s);
90     printf("链栈初始化成功\n");
91     printf("进行入栈! \n");
92     push(&s,1);
93     push(&s, 21);
94     push(&s, 31);
95     push(&s, 51);
96     printf("入栈成功! \n");
97     printf("进行出栈: \n");
98     pop(&s);
99     pop(&s);
100    pop(&s);
101    printf("出栈结束: \n");
102    //slen(&s);
103
104
105 }
```

四、链栈和顺序栈的总结

1. 顺序栈:

结构体构造:

1. 栈顶指针 `top`
2. 栈底指针 `base`
3. 数据个数 `size`

特性:

~栈空: `top = base`

~栈满: `top - base = maxsize`

栈溢出:

~上溢报错: `top - base > maxsize`

~下溢结束: `top - base = maxsize`

2. 链栈:

结构体构造:

1. 数据域
2. 结构体指针域

特性:

链栈的指针方向是前驱元素

1. 链表的头指针就是栈顶
2. 链栈不需要头节点
3. 基本不存在栈满的情况
4. 空栈相当于头指针指向空
5. 插入和删除仅在栈顶处执行

首先从栈顶插入元素，开辟一个空间，造一个指针**p**指向改空间的的首地址

给数据域进行赋值，再指向链栈的**next**的域，然后修改栈顶指针**p**为**s**,链栈创建完成

都是通过动态分配函数开辟内存空间