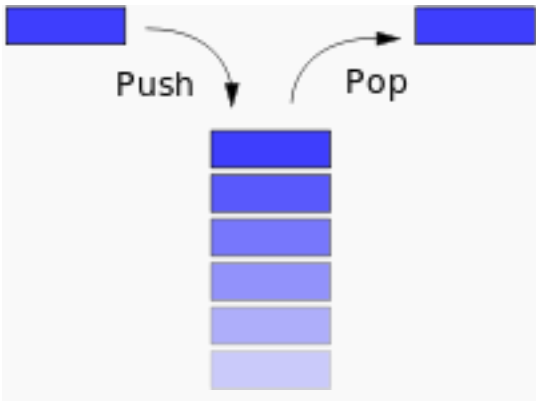
堆栈



堆叠的简单示意图

堆栈（英语：stack），也可直接称**栈**。台湾作**堆疊**，在计算机科学中，是一种特殊的串列形式的数据结构，它的特殊之处在于只能允许在链接串列或阵列的一端（称为堆叠顶端指标，英语：top）进行加入资料（英语：push）和输出资料（英语：pop）的运算。另外堆叠也可以用一维阵列或连结串列的形式来完成。堆叠的另外一个相对的操作方式称为伫列。

由于堆叠数据结构只允许在一端进行操作，因而按照后进先出（LIFO, Last In First Out）的原理运作。

堆叠数据结构使用两种基本操作：推入（push）和弹出（pop）：

- 推入：将数据放入堆叠的顶端（阵列形式或串列形式），堆叠顶端top指标加一。
- 弹出：将顶端数据资料输出（回传），堆叠顶端资料减一。

栈的基本特点：

1，先入后出，后入先出。

2，除头尾节点之外，每个元素有一个前驱，一个后继。

抽象定义

以下是堆栈的VDM（Vienna Development Method）：[1]

函数签名：

```
init: -> Stack
push: N x Stack -> Stack
top: Stack -> (N U ERROR)
pop: Stack -> Stack
isempty: Stack -> Boolean
```

此处的N代表某个元素（如自然数），而U表示集合求交。

语义：

top(init()) = ERROR
top(push(i,s)) = i
pop(init()) = init()
pop(push(i, s)) = s
isempty(init()) = true
isempty(push(i, s)) = false

软件堆栈

阵列堆叠

堆栈可以用链表和数组两种方式实现，一般为一个堆栈预先分配一个大小固定且较合适的空间并非难事，所以较流行的做法是Stack结构下含一个数组。如果空间实在紧张，也可用链表实现，且去掉表头。这里的例程是以数组实现的。

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define stack struct Stack
#define STACK_POP_ERR 42
/* 堆疊資料結構 堆栈数据结构 */
struct Stack
{
  int val[10]; // 陣列空間
  int top;     // 堆疊頂端指標（栈顶）
};
/* 檢查堆疊是否為空 */
bool empty(stack *stk) { return stk->top == 0; }
/* 推入資料 */
void push(stack *stk, int x)
{
  stk->top=stk->top+1;
  stk->val[stk->top]=x;
}
/* 彈出并返回資料 */
int pop(stack *stk)
{
  if(empty(stk))
    return STACK_POP_ERR; // 不能彈出
```

```
  else
  {
    stk->top=stk->top-1;
    return stk->val[stk->top+1];
  }
}
int main()
{
  // 宣告并初始化資料結構空間
  stack stk;
  stk.top=0;
  // 推入四个
  push(&stk, 3);
  push(&stk, 4);
  push(&stk, 1);
  push(&stk, 9);
  // 弹出三个
  printf("%d ", pop(&stk));
  printf("%d ", pop(&stk));
  printf("%d ", pop(&stk));
  return 0;
}
```

串列堆疊

```
/*链栈的结构定义*/
typedef struct {
  SLink top;        // 棧頂指針
  int length;     // 棧中元素個數
} Stack;

// 构造空栈 S
void InitStack (Stack &S)
{
  S.top = NULL;       // 設棧頂指針的初值為"空"
  S.length = 0;       // 空棧中元素個數為0
}
// 如果指针反过来从栈底到栈顶的话，删除栈顶元素时，为修改其前驱指针，需要从栈底一直找到栈顶。
```

```
// 在栈顶S 之上插入元素e为新的栈顶元素，并返回成功與否
bool Push (Stack &S, ElemType e) {
 p = new LNode;       // 建新的結點
 if(!p)
   return false;      // 存儲分配失敗
 p -> data = e;
 p -> next = S.top;// 鏈接到原來的棧頂
 S.top = p;           // 移動棧頂指針
 ++S.length;           // 棧的長度增1
}
// 在链栈的类型定义中设立“栈中元素个数”的成员是为了便于求得栈的长度。

// 删除S 棧頂且以e 返回其數值，返回成功與否
bool Pop (Stack &S, SElemType &e)
{
  if (!S.top)
    return false;
  else
  {
     e = S.top -> data;        // 返回棧頂元素
     q = S.top;
     S.top = S.top -> next; // 修改棧頂指針
     --S.length;                 // 棧的長度減1
     delete q;                  // 釋放被刪除的結點空間
     return true;
   }
}
```