栈

线性表是一个非常“开放的”数据结构，用户可以读、写、插入和删除线性表上的任何一个数据单元。但在实际的计算模型中，并不是所有的数据结构都允许这样做。通过对允许访问的数据进行限制，我们可以过滤掉多余的信息，简化模型的复杂程度，从而更快更好地解决问题。

栈（stack）、队列（queue）以及双端队列（deque）就是典型的“访问受限制”的数据结构，它们的实现都是以线性表作为基础的；如果说线性表是“容器”，那么栈和队列这些结构更接近于“接口”，在 C++ STL 中称它们为容器适配器（adapter）。另一种经典的限制访问的数据结构是优先队列（priority queue），它需要以树作为基础，因此放到较后的章节讨论。栈和队列限制了用户访问元素的范围，这使得它们能够防止用户做出对进程、系统、计算机或网络有害的“非法”行为，在《操作系统》和《网络原理》中都能看到它们的应用。本章将首先讨论栈及其应用。

栈的性质

栈的定义

栈（stack）是一种特殊的线性表。对于栈 `S[0:n]`，它的插入、访问（因为只能读一个元素，不存在“查找”）、删除操作均只能对栈顶元素（top 或 peek，即栈的最后一个元素）`S[n-1]`进行。如图 ref{fig:sta1}所示，当栈顶为 x 时，三种操作都只能在 x 处进行。红色标出了操作之后的新栈顶，蓝色标出了操作之后的不可访问区域。

1. 栈的插入就是在 `S[n-1]`后面插入新的元素（称为入栈或推入，push）。
2. 栈的访问就是取 `S[n-1]`。
3. 栈的删除就是将 `S[n-1]`从栈中删除（称为出栈或弹出，pop）。

```
template <typename T>
class AbstractStack {
public:
    virtual void push(const T& e) = 0;
    virtual void pop() = 0;
    virtual T& top() = 0;
};
```

栈的实现

由于栈实质上是对线性表的访问权限作出限定，所以很容易在线性表的基础上建立栈。以向量（顺序表）实现的栈称为顺序栈，以列表（链表）实现的栈称为链栈。

这里需要注意的是，如果以向量（常规方式）实现栈，则我们总是使用末尾元素代表栈顶，就像在上一节所定义的那样。向量在头部做插入和删除操作的效率非常低，所以向量只能以尾部作为栈顶。但如果以单向列表实现栈，末尾元素变得不可访问，所以应当反过来以起始元素代表栈顶。双向列表因为是对称的，所以在末尾或起始都可以作为栈顶。

显然无论是顺序栈还是链栈，取顶的时间复杂度是$O(1)$，入栈和出栈的时间复杂度在分摊意义下也是$O(1)$。根据上一章的介绍您可以知道，用列表实现的栈，时间效率和空间效率都不如向量，但稳定性稍好。 在绝大多数应用场景下，使用向量实现栈都比使用列表实现栈更优，很少会出现使用链栈的场景。