栈的实现方式

- **顺序栈 (Array-based Stack)**
  - 使用向量实现，本质上是顺序表的简化版
    - 栈的大小
  - 关键是确定**哪一端**作为栈顶
  - 上溢，下溢问题

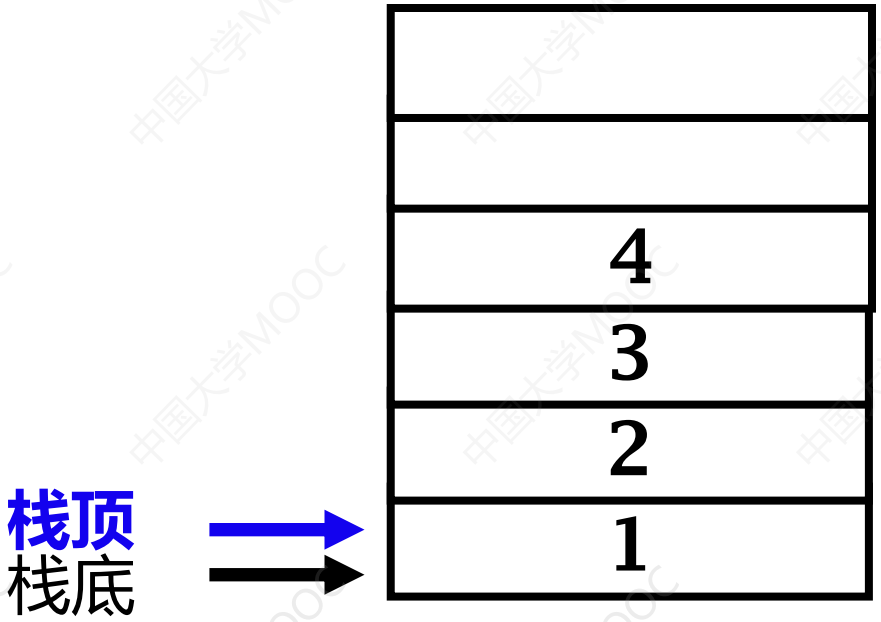
- **链式栈 (Linked Stack)**
  - 用单链表方式存储，其中指针的方向是从栈顶向下链接

顺序栈的类定义

```
template <class T> class arrStack : public Stack <T> {
private:                                  // 栈的顺序存储
    int mSize;                            // 栈中最多可存放的元素个数
    int top;                              // 栈顶位置，应小于mSize
    T *st;                                // 存放栈元素的数组
public:                                   // 栈的运算的顺序实现
    arrStack(int size) {                  // 创建一个给定长度的顺序栈实例
        mSize = size; top = -1; st = new T[mSize];
    }
    arrStack() {                          // 创建一个顺序栈的实例
        top = -1;
    }
    ~arrStack() { delete [] st; }
    void clear() { top = -1; }  // 清空栈
}
```

顺序栈

- 按压入先后次序，最后压入的元素编号为4，然后依次为3,2,1



顺序栈的溢出

- **上溢 (Overflow)**
  - 当栈中已经有maxsize个元素时，如果再做进栈运算，所产生的现象
- **下溢 (Underflow)**
  - 对空栈进行出栈运算时所产生的现象

压入栈顶

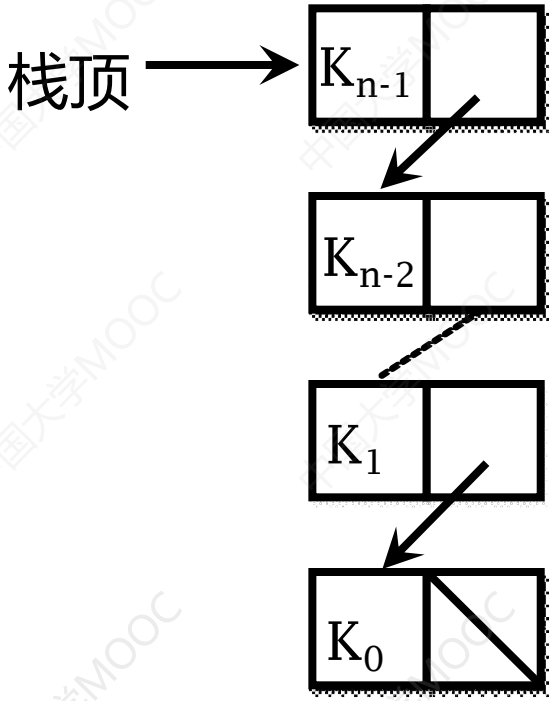
```
bool arrStack<T>::push(const T item) {
    if (top == mSize-1) {        // 栈已满
        cout << "栈满溢出" << endl;
        return false;
    } else {                     // 新元素入栈并修改栈顶指针
        st[++top] = item;
        return true;
    }
}
```

从栈顶弹出

```
bool arrStack<T>::pop(T & item) {   // 出栈
    if (top == -1) {                // 栈为空
        cout << "栈为空，不能执行出栈操作"<< endl;
        return false;
    } else {
        item = st[top--];  // 返回栈顶，并缩减1
        return true;
    }
}
```

链式栈的定义

- 用单链表方式存储
- 指针的方向从**栈顶向下**链接



链式栈的创建

```
template <class T> class lnkStack : public Stack <T> {
private:                                    // 栈的链式存储
    Link<T>* top;                           // 指向栈顶的指针
    int size;                               // 存放元素的个数
public:                                     // 栈运算的链式实现
    lnkStack(int defSize) {                 // 构造函数
        top = NULL; size = 0;
    }
    ~lnkStack() {                           // 析构函数
        clear();
    }
}
```

压入栈顶

```
// 入栈操作的链式实现
bool lnksStack<T>:: push(const T item) {
    Link<T>* tmp = new Link<T>(item, top);
    top = tmp;
    size++;
    return true;
}

Link(const T info, Link* nextValue)  {// 具有两个参数的Link构造函数
        data = info;
        next = nextValue;
}
```

从单链栈弹出元素

```
// 出栈操作的链式实现
bool lnkStack<T>:: pop(T& item) {
    Link <T> *tmp;
    if (size == 0) {
        cout << "栈为空，不能执行出栈操作"<< endl;
        return false;
    }
    item = top->data;
    tmp = top->next;
    delete top;
    top = tmp;
    size--;
    return true;
}
```

顺序栈和链式栈的比较

- **时间效率**
  - 所有操作都只需常数时间
  - 顺序栈和链式栈在时间效率上难分伯仲
- **空间效率**
  - 顺序栈须说明一个固定的长度
  - 链式栈的长度可变，但增加结构性开销
- **实际应用中，顺序栈比链式栈用得更广泛**