# ADT  Abstract Data Type

Astratto = descritto prescindendo dalla sua implementazione

attraverso:
- la collezione di dati
- le operazioni
- la complessità

Implementazione concreta:

struttura dati + procedure

---

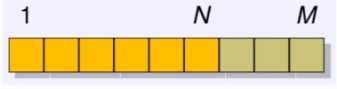
# PILE / STACK → struttura LIFO

PUSH = inserimento

POP = estrazione

TOP = restituisce l'elemento in cima

## IMPLEMENTAZIONE CON ARRAY




- gli elementi in pila occupano sempre le prime posizioni dell'array
- quando ci sono N elementi, il prossimo elemento da estrarre è in posizione N

```
PUSH(S, t)
  if S.N ≠ S.M then
      S.N ← S.N + 1
      S[N] ← t
  else
      error overflow
```

→ definito se e solo se SIZE(S) < M (numero max di elementi)

```
SIZE(S)
  return S.N
```

```
EMPTY(S)
  if S.N == 0 then
      return true
  return false
```

```
TOP(S)
  if S.N == 0 then
      error underflow
  else
      return S[S.N]
```

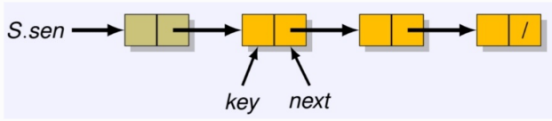
```
POP(S)
  if S.N == 0 then
      error underflow
  else
      S.N ← S.N − 1
      return S[S.N + 1]
```

Complessità temporale: O(1)

Complessità spaziale: O(M)

## IMPLEMENTAZIONE CON LISTA

- conviene lista semplice con sentinella
- teniamo conto del numero di elementi (S.N)




```
PUSH(S, t)
  S.N ← S.N + 1
  t.next ← S.sen.next
  S.sen.next ← t
```

```
SIZE(S)
  return S.N
```

```
EMPTY(S)
  if S.N == 0 then
      return true
  return false
```

```
TOP(S)
  if S.N == 0 then
      error underflow
  else
      return S.sen.next
```

```
POP(S)
  if S.N == 0 then
      error underflow
  else
      S.N ← S.N − 1
      t ← S.sen.next
      S.sen.next ← S.sen.next.next
      return t
```

Complessità temporale: O(1)

Complessità spaziale: O(N)

---

## CODE / QUEUE → struttura FIFO

ENQUEUE = inserimento

DEQUEUE = estrazione

FRONT = restituisce il primo elemento nella coda

## IMPLEMENTAZIONE CON ARRAY

- array usato in maniera circolare tenendo conto di dove stanno head e tail
- Q.head indica dove estrarre l'elemento successivo
- Q.tail indica dove inserire l'elemento successivo
- Q.head = Q.tail ⟺ coda vuota (con n celle posso gestire al massimo n-1 elementi)

```
SIZE(Q)
  if Q.tail ≥ Q.head then
      return Q.tail − Q.head
  return Q.M − (Q.head − Q.tail)
```

```
EMPTY(Q)
  if Q.tail == Q.head then
      return true
  return false
```

```
NEXTCELL(Q, c)
  if c ≠ Q.M then
      return c + 1
  return 1
```

```
FRONT(Q)
  if SIZE(Q)== 0 then
      errorunderflow
  else
      return Q[Q.head]
```

```
ENQUEUE(Q, t)
  if SIZE(Q) ≠ Q.M − 1 then
      Q[Q.tail] ← t
      Q.tail ← NEXTCELL(Q,Q.tail)
  else
      erroroverflow
```

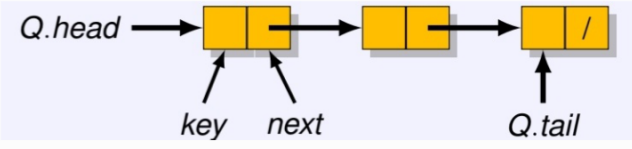
```
DEQUEUE(Q)
  if SIZE(Q)== 0 then
      errorunderflow
  else
      t ← Q[Q.head]
      Q.head ← NEXTCELL(Q,Q.head)
      return t
```

Complessità temporale: $O(1)$

Complessità spaziale: $O(n)$

## IMPLEMENTAZIONE CON LISTA

- lista semplice, aggiungendo un puntatore all'ultimo elemento




Q.head = elem. da estrarre

Q.tail = ultimo inserito

- Q.head == null $\iff$ coda vuota
- teniamo conto del numero di elementi Q.N

```
ENQUEUE(Q, t)
  if Q.N == 0 then
      Q.head ← t
      Q.tail ← t
  else
      Q.tail.next ← t
      Q.tail ← t
  Q.N ← Q.N + 1
```

```
SIZE(Q)
  return Q.N
```

```
EMPTY(Q)
  if Q.N == 0 then
      return true
  return false
```

```
FRONT(Q)
  if Q.N == 0 then
      errorunderflow
  else
      return Q.head
```

```
DEQUEUE(Q)
  if Q.N == 0 then
      errorunderflow
  else
      t ← Q.head
      Q.head ← Q.head.next
      Q.N ← Q.N − 1
      return t
```

Complessità temporale: $O(1)$

Complessità spaziale: $O(N)$