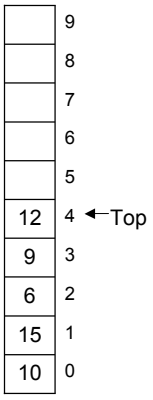
Tipi astratti pila e coda

Tipo astratto pila

- Una pila è un tipo astratto che consente di rappresentare un insieme di elementi in cui ogni eliminazione ha per oggetto l'elemento che è stato inserito per ultimo.
- Questa disciplina di gestione è spesso chiamata LIFO (Last In,First Out).
- Un tipico esempio di utilizzo di questa struttura di dati è dato dalla gestione dei parametri delle funzioni di un programma C e delle sue variabili locali:
  - all'attivazione della funzione i parametri formali e le variabili locali vengono inseriti in cima alla pila;
  - al termine dell'esecuzione vengono eliminati dalla pila in ordine inverso rispetto a quello di inserimento.



Tipo astratto pila

Per il tipo astratto pila possiamo definire le funzioni primitive:

- test_pila_vuota: verifica se una pila è vuota

  `test_pila_vuota: pila → boolean`
- top: restituisce l'elemento in cima alla pila, l'ultimo ad essere stato inserito

  `top: pila → atomo`
- push: inserisce un elemento in cima alla pila

  `push: pila×atomo → pila`
- pop: elimina dalla pila l'ultimo elemento inserito

  `pop: pila → pila`

Tipo astratto pila

- Il tipo astratto pila può essere definito come la tripla Pila=<S,F,C>
  1. S={pila, atomo, boolean} con pila dominio di interesse
  2. F={test_pila_vuota, top, push, pop}
  3. C={pila_vuota}
  - `test_pila_vuota: pila → boolean`
  - `top: pila → atomo`
  - `push: pila×atomo → pila`
  - `pop: pila → pila`
- Per rappresentare una pila possiamo usare una lista facendo in modo che l'ordine con cui si susseguono gli elementi rispecchi l'ordine di inserimento degli elementi nella pila.
- Per quanto riguarda le operazioni si può stabilire il seguente parallelo tra le operazioni sui due tipi lista e pila:

`test_pila_vuota`	⟺	`lista_vuota`
`top`	⟺	`testa`
`push`	⟺	`in_testa`
`pop`	⟺	`da_testa`

Pila


```
int test_pila_vuota( struct atomo *ptop)
{
  if(ptop==NULL) return 1;
  else return 0;
}

void push (struct atomo **ptop, int e)
{
 struct atomo *aux;

 aux=malloc(sizeof(struct atomo));
 if(aux)
  {
   aux->dato=e;
   aux->prossimo=*ptop;
   *ptop=aux;
  }
 else printf("Memoria esaurita\n");
}
```


Pila


```
struct atomo * top (struct atomo *ptop)
{
   return ptop;
}


void pop (struct atomo **ptop)
{
 struct atomo *aux;

 if(!test_pila_vuota(*ptop))
  {aux=*ptop;
   *ptop=aux->prossimo;
   free(aux);
  }
 else printf("La pila e' gia' vuota\n");
}
```

Tipo astratto coda

- Una coda è un tipo astratto che consente di rappresentare un insieme di elementi in cui ogni eliminazione ha per oggetto l'elemento che è stato inserito per primo.
- Questa disciplina di gestione è spesso chiamata FIFO (First In, First Out).
- Tale disciplina è tipica di molte situazioni della vita di tutti i giorni: agli sportelli degli uffici il primo ad essere servito è il primo della coda.
- Il tipo coda viene caratterizzato dalle seguenti primitive:
  - test_coda_vuota: verifica se la coda è vuota

    `test_coda_vuota: coda → boolean`
  - primo: restituisce il primo elemento inserito nella coda

    `primo: coda → atomo`
  - in_coda: inserisce un elemento nella coda

    `in_coda: coda × atomo → coda`
  - out_coda: elimina dalla coda il primo elemento inserito

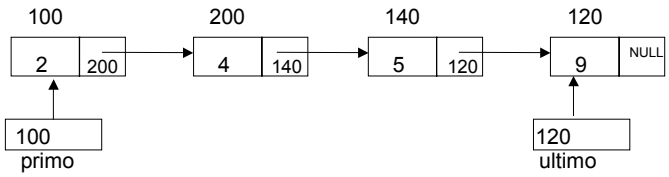
    `out_coda: coda → coda`

Tipo astratto coda

- Il tipo astratto coda può essere definito come la tripla Coda=<S,F,C>
  1. S={coda, atomo, boolean} con coda dominio di interesse
  2. F={test_coda_vuota, primo, in_coda, out_coda}
  3. C={coda_vuota}

  - `test_coda_vuota: coda → boolean`
  - `primo: coda → atomo`
  - `in_coda: codaxatomo → coda`
  - `out_coda: coda → coda`

Rappresentazione collegata di una coda



- Gli elementi di una coda possiamo rappresentarli mediante delle struct contenenti i suo valore e il puntatore all'elemento successivo.
- Inoltre, si usano due puntatori, primo e ultimo, che puntano al primo e all'ultimo elemento inserito nella coda.

```
struct atomo
{ int dato;
  struct atomo *prossimo;
};

struct coda {
   struct atomo *primo;
   struct atomo *ultimo;
};
```


Primitive del tipo coda

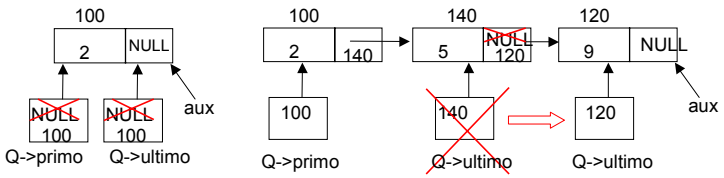
- La condizione di coda vuota è che sia il campo primo sia il campo ultimo della coda abbiano valore NULL.
- Poiché nell'operazione di out_coda, in caso di eliminazione dell'unico elemento della coda entrambi sono posti a NULL, nella funzione test_coda_vuota è sufficiente controllare il valore solo di Q.primo.

```
int test_coda_vuota( struct coda Q)
{
  if(Q.primo==NULL) return 1;
  else return 0;
}
```

- Alla funzione primo viene fatto restituire il puntatore al primo elemento inserito nella coda.

```
struct atomo * primo (struct coda Q)
{
   return Q.primo;
}
```

in_coda



Dopo avere allocato il nuovo elemento il cui indirizzo è assegnato ad **aux**

```
aux=malloc(sizeof(struct atomo));
if(aux)
{aux->dato=e;
 aux->prossimo=NULL;
```

Se la coda è vuota, a **Q->primo** e a **Q->ultimo** viene assegnato **aux**;

```
if(test_coda_vuota(*Q))
  {Q->primo=aux;
   Q->ultimo=aux;}
```

Altrimenti viene asseganto al campo prossimo dell'ultimo elemento **aux**, in modo da collegare l'ultimo elemento con il nuovo elemento, e viene posto **Q->ultimo=aux** in modo da aggiornare il puntatore al nuovo ultimo elemento.

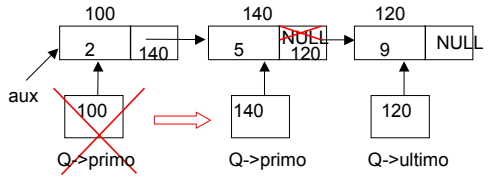
```
else {  Q->ultimo->prossimo=aux;
        Q->ultimo=aux;       }
```


in_coda


```
void in_coda (struct coda *Q, int e)
{ struct atomo *aux;

 aux=malloc(sizeof(struct atomo));
 if(aux)
  {aux->dato=e;
   aux->prossimo=NULL;
   if(test_coda_vuota(*Q))
       Q->primo=aux;
   else Q->ultimo->prossimo=aux;
   Q->ultimo=aux;
  }
 else printf("Memoria esaurita\n");
}
```

out_coda



- Nell'eliminazione del primo elemento si procede in modo analogo all'eliminazione dalla testa della lista, viene eliminato l'elemento puntato da Q->primo.

```
if(!test_coda_vuota(*Q))
 {aux=Q->primo;
  Q->primo=aux->prossimo;
  free(aux);
```

- Nel caso in cui dopo l'eliminazione del primo elemento la coda diventa vuota (Q->primo==NULL) , è necessario garantire che anche Q->ultimo abbia stesso valore NULL.

```
if(Q->primo==NULL)
   Q->ultimo=NULL;
```


out_coda


```
void out_coda (struct coda *Q)
{
 struct atomo *aux;

 if(!test_coda_vuota(*Q))
  {aux=Q->primo;
   Q->primo=aux->prossimo;
   free(aux);
   if(Q->primo==NULL)
      Q->ultimo=NULL;
  }
 else printf("La coda e' gia' vuota\n");
}
```