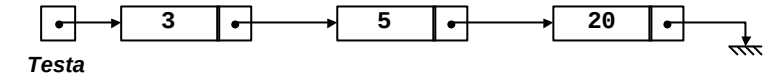
Stack Dinamici

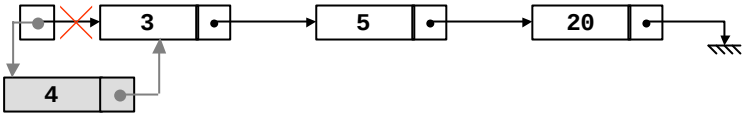
Consideriamo una lista semplicemente collegata…



{3, 5, 20}

Testa della lista

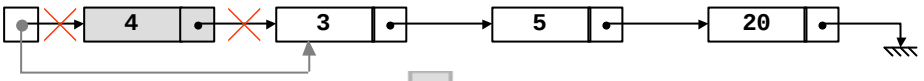
Definiamo: ***PUSH → inserimento in testa***



{4, 3, 5, 20}

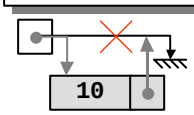
Definiamo: ***POP → estrazione dalla testa***

{3, 5, 20}

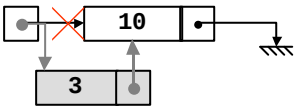


Otteniamo uno STACK DINAMICO

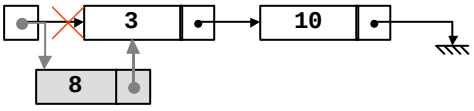
STACK DINAMICI



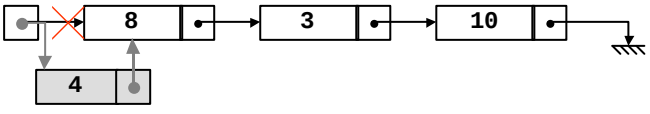
PUSH (10);



PUSH (3);

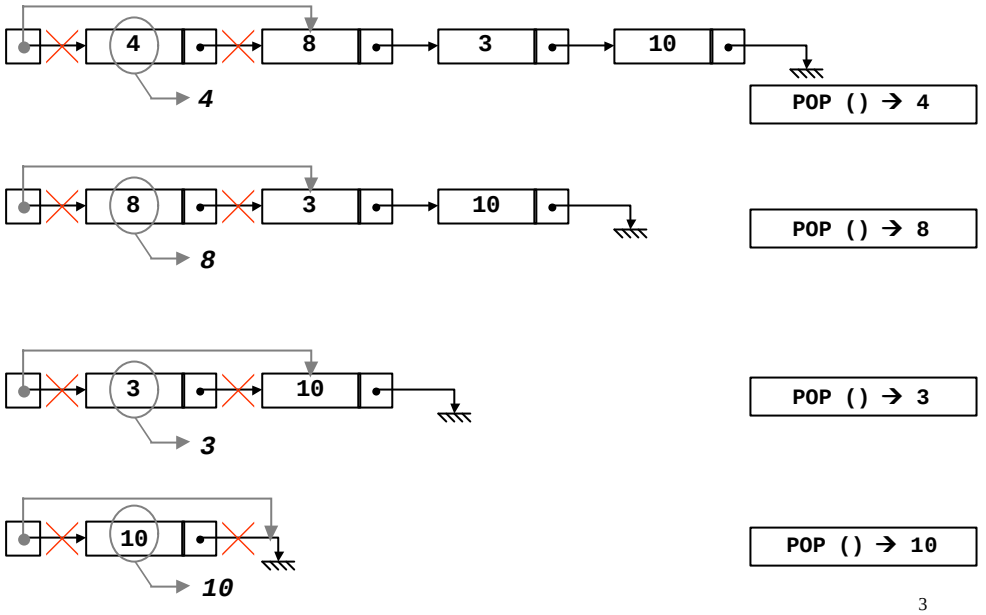


PUSH (8);



PUSH (4);

STACK DINAMICI



3

Stack dinamici: implementazione

```
typedef struct t_stack_element {
  int data;                          // stack di interi
  struct t_stack_element * next;
} t_stack_element;

typedef t_stack_element * t_stack;


t_stack init_stack (void)
{
  return NULL;
}

...

t_stack mystack;
mystack = init_stack ();

...
```

4

STACK DINAMICI: PUSH

```
typedef struct t_stack_element {
  int data;                          // stack di interi
  struct t_stack_element * next;
} t_ stack_element;

typedef t_stack_element * t_stack;

void push (t_stack * s, int data)
{
  t_stack_element * newelem;
  newelem = (t_list_element *) malloc (sizeof (t_list_element));
  if (newelem != NULL) {
    newelem->data = data;
    newelem->next = *s;
    *s = newelem;
  }
}

...
t_stack mystack;
push (&mystack, 10);
...
```


STACK DINAMICI: POP

```
typedef struct t_stack_element {
  int data;
  struct t_stack_element * next;
} t_list_element;

typedef t_stack_element * t_stack;

int pop (t_stack * s)
{
  t_stack_element * aux;
  int removed_element;
  aux = *s;
  *s = (*s)->next;
  removed_element = aux->data;
  free (aux);
  return removed_element;
}
```

```
int is_stack_empty (t_stack s)
{
  return s == NULL;
}
```

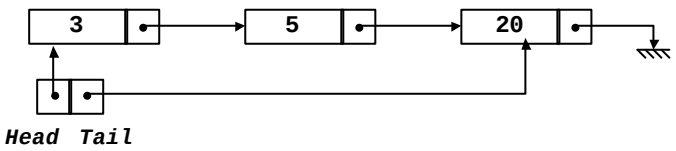
```
t_stack mystack;
if (is_stack_empty (mystack) == 0)
  printf ("Elemento estratto da POP = %d\n", pop (&mystack));
```

STACK DINAMICI: VISITA

```
typedef struct t_stack_element {
  int data;
  struct t_stack_element * next;
} t_stack_element;

typedef t_stack_element * t_stack;


void dump_stack (t_stack s)
{
  t_stack_element * aux;
  aux = s;
  while (aux != NULL) {
    printf ("%d\n", aux->data);
    aux = aux->next;
  }
}

...
t_stack mystack;
dump_stack (mystack);
...
```

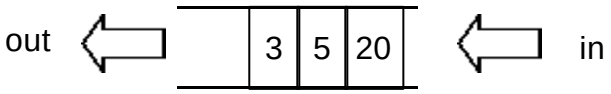

CODE DINAMICHE

Consideriamo una lista semplicemente collegata con un puntatore in testa ed uno in coda

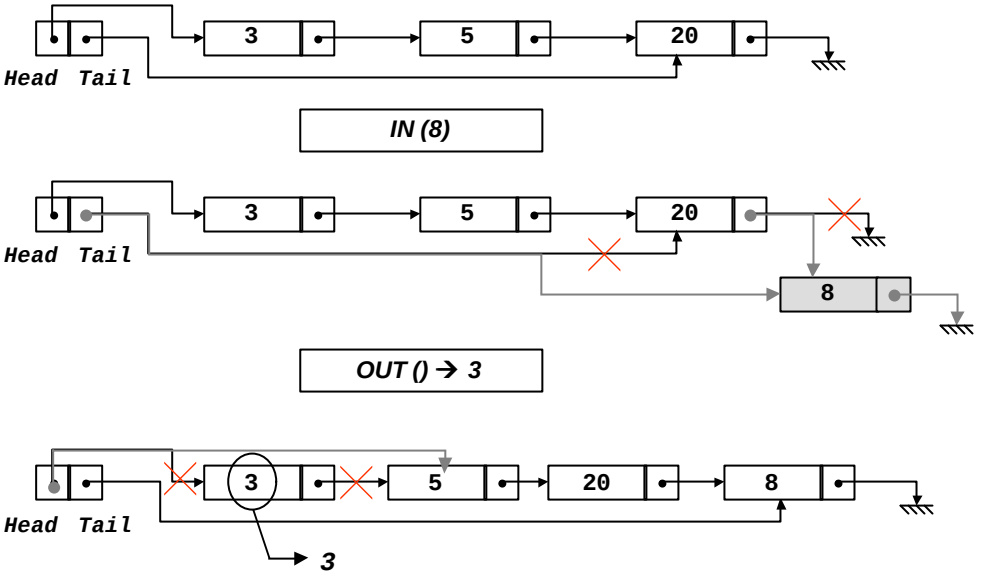


Estrazione dalla coda:
→ estrazione da `Head`

Inserimento in coda:
→ inserimento su `Tail`

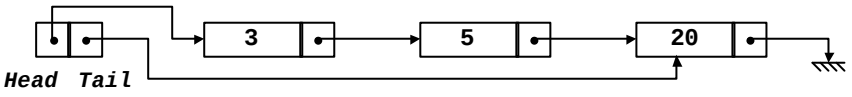


CODE DINAMICHE



9

CODE DINAMICHE: STRUTTURA DATI

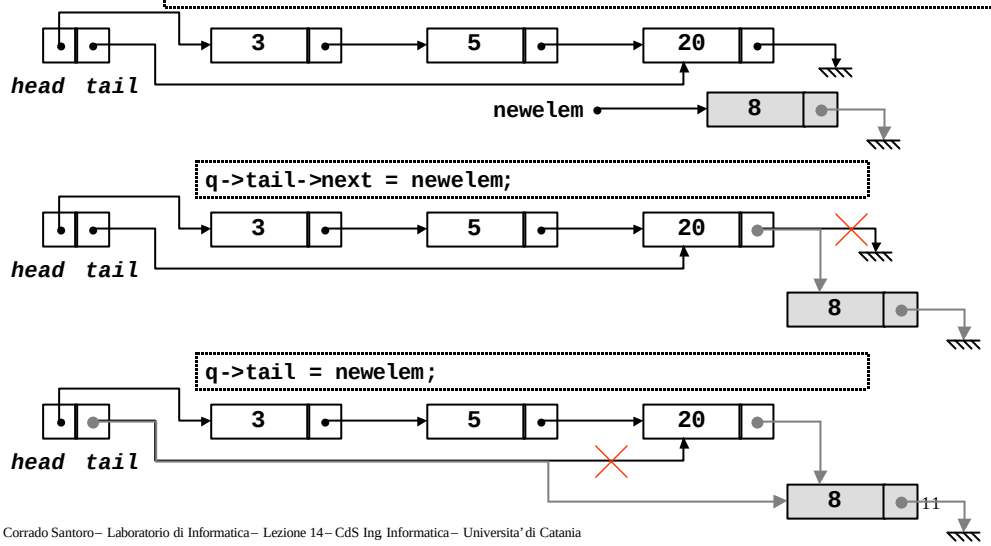



```
// l'elemento
typedef struct t_queue_element {
  int data;                          // coda di interi
  struct t_queue_element * next;
} t_queue_element;

// la coda
typedef struct {
  t_queue_element * head;
  t_queue_element * tail;
} t_queue;


t_queue init_queue ();               // inizializzazione
void in (t_queue * q, int value);    // inserimento
int out (t_queue * q);               // estrazione
int is_queue_empty (t_queue q);      // test vuota
void dump_queue (t_queue q);         // visita
```

10

void in (t_queue * q, int value);

```
t_queue_element * newelem;
newelem = (t_queue_element *)malloc (sizeof (t_queue_element));
newelem->data = value;
newelem->next = NULL;
```




```
q->tail->next = newelem;
```



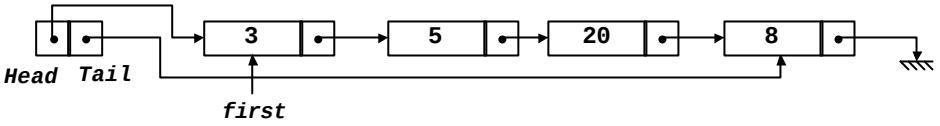
```
q->tail = newelem;
```


void in (t_queue * q, int value);

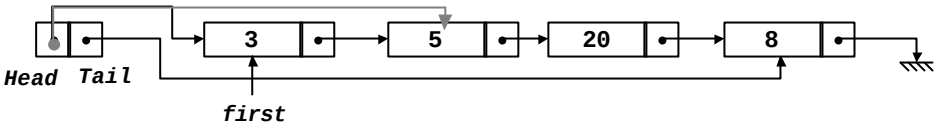
```
// l'elemento
typedef struct t_queue_element {
  int data;                          // coda di interi
  struct t_queue_element * next;
} t_queue_element;

// la coda
typedef struct {
  t_queue_element * head;
  t_queue_element * tail;
} t_queue;

void in (t_queue * q, int value)
{
  t_queue_element * newelem;
  newelem = (t_queue_element *)malloc (sizeof (t_queue_element));
  if (newelem != NULL) {
    newelem->data = value;
    newelem->next = NULL;
    q->tail->next = newelem;
    q->tail = newelem;
  }
}
```

int out (t_queue * q);

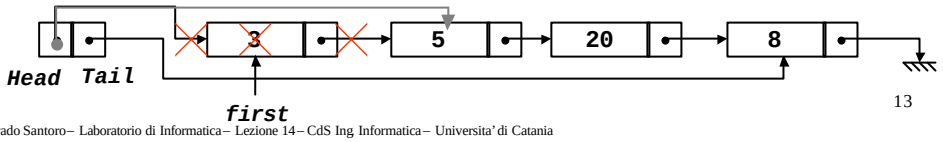
```
t_queue_element * first;
first = q->head;
```




```
q->head = q->head->next;
```



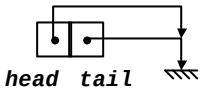

```
int value = first->data;
free (first);
return value;
```



int out (t_queue * q);

```
// l'elemento
typedef struct t_queue_element {
  int data;                          // coda di interi
  struct t_queue_element * next;
} t_queue_element;

// la coda
typedef struct {
  t_queue_element * head;
  t_queue_element * tail;
} t_queue;

int out (t_queue * q)
{
  t_queue_element * first;
  int value;
  first = q->head;
  q->head = q->head->next;
  value = first->data;
  free (first);
  return value;
}
```

Initializzazione / Condizione di coda vuota




```
t_queue init_queue (void)
{
  t_queue q;
  q.head = NULL;
  q.tail = NULL;
  return q;
}

int is_queue_empty (t_queue q)
{
  return q.head = = NULL;
}

// oppure

int is_queue_empty (t_queue q)
{
  return q.tail = = NULL;
}
```

15

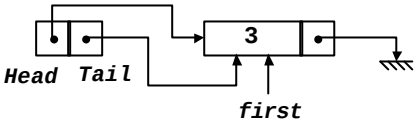
Inserimento in coda completo (con controllo del caso di coda vuota)


```
void in (t_queue * q, int value)
{
  t_queue_element * newelem;
  newelem = (t_queue_element *)malloc (sizeof (t_queue_element));
  if (newelem != NULL) {
    newelem->data = value;
    newelem->next = NULL;
    if (q->head = = NULL) {
      q->head = newelem;
    }
    else {
      q->tail->next = newelem;
    }
    q->tail = newelem;
  }
}
```

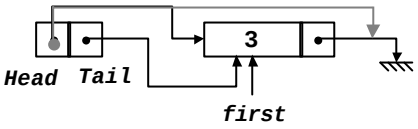
16

int out (t_queue * q); (coda con un solo elemento)

```
t_queue_element * first;
first = q->head;
```

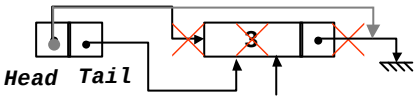



```
q->head = q->head->next;
```



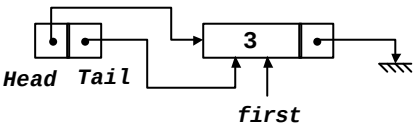
q->head == NULL

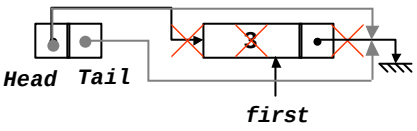
```
int value = first->data;
free (first);
return value;
```



ATTENZIONE! q->tail punta ad una zona non più valida

int out (t_queue * q);






```
int out (t_queue * q)
{
  t_queue_element * first;
  int value;
  first = q->head;
  q->head = q->head->next;
  if (q->head = = NULL)
    q->tail = NULL;
  value = first->data;
  free (first);
  return value;
}
```