Laboratorio di Algoritmi e Strutture Dati I

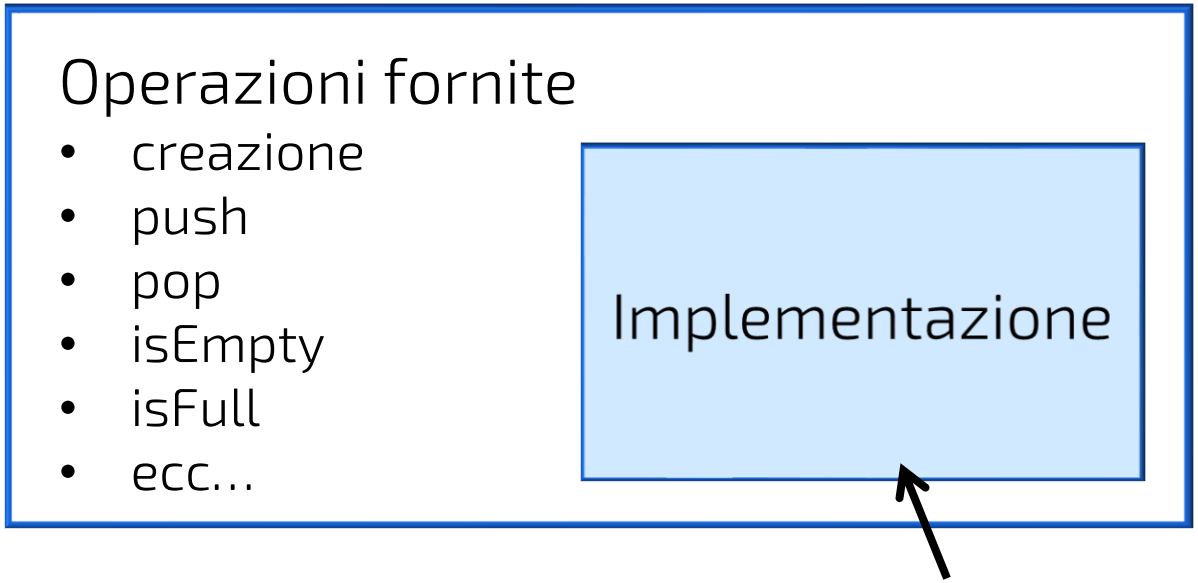
Esercitazione 5

martedì 14 aprile

Lezione 5: Strutture Dati Astratte

- L'implementazione della Struttura dati deve essere trasparente per l'utente.

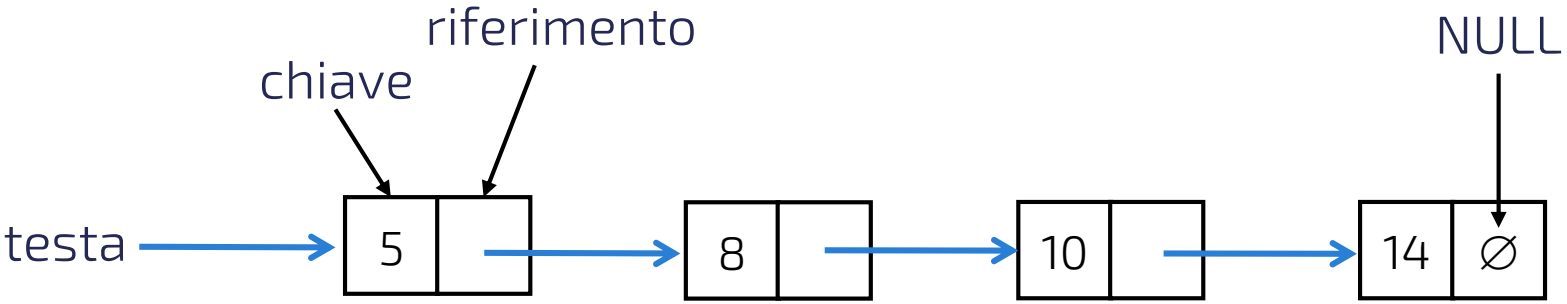
Struttura Dati generica



Possiamo modificare l'implementazione (con una più efficiente) senza modificare le operazioni fornite. L'utilizzatore finale della struttura dati non deve conoscere i dettagli implementativi.

Lezione 5: Liste singolarmente concatenate

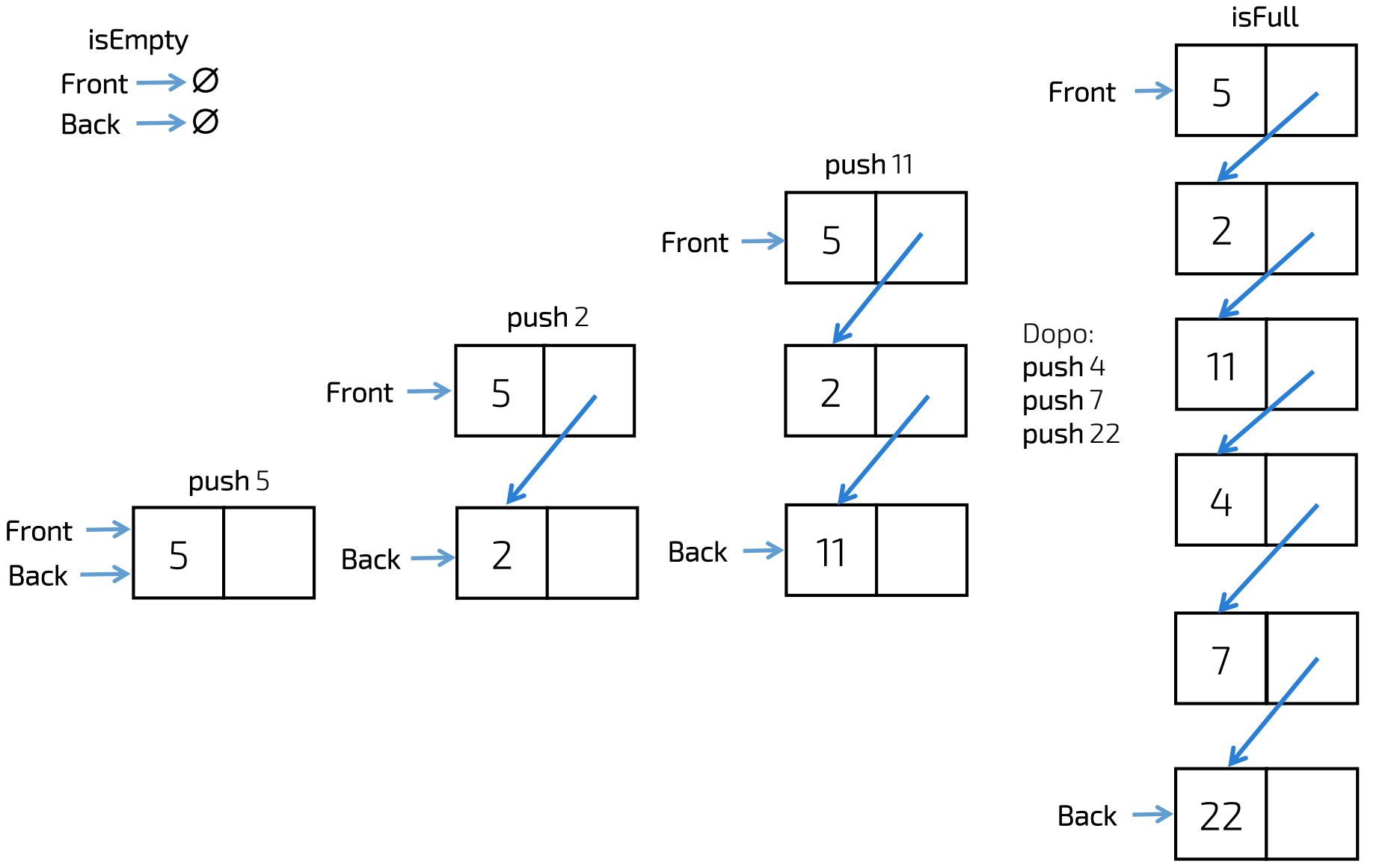
- Insieme di elementi omogenei connessi tra loro (ad esempio tramite puntatori).
- È una struttura dati astratta lineare ad accesso sequenziale ed, eventualmente, diretto (in quale caso?).
- Ogni elemento della lista ha un **chiave** e un **riferimento** (puntatore) all'elemento successivo della lista.
- NB: il campo **chiave** può essere di qualsiasi tipo (anche una struttura).



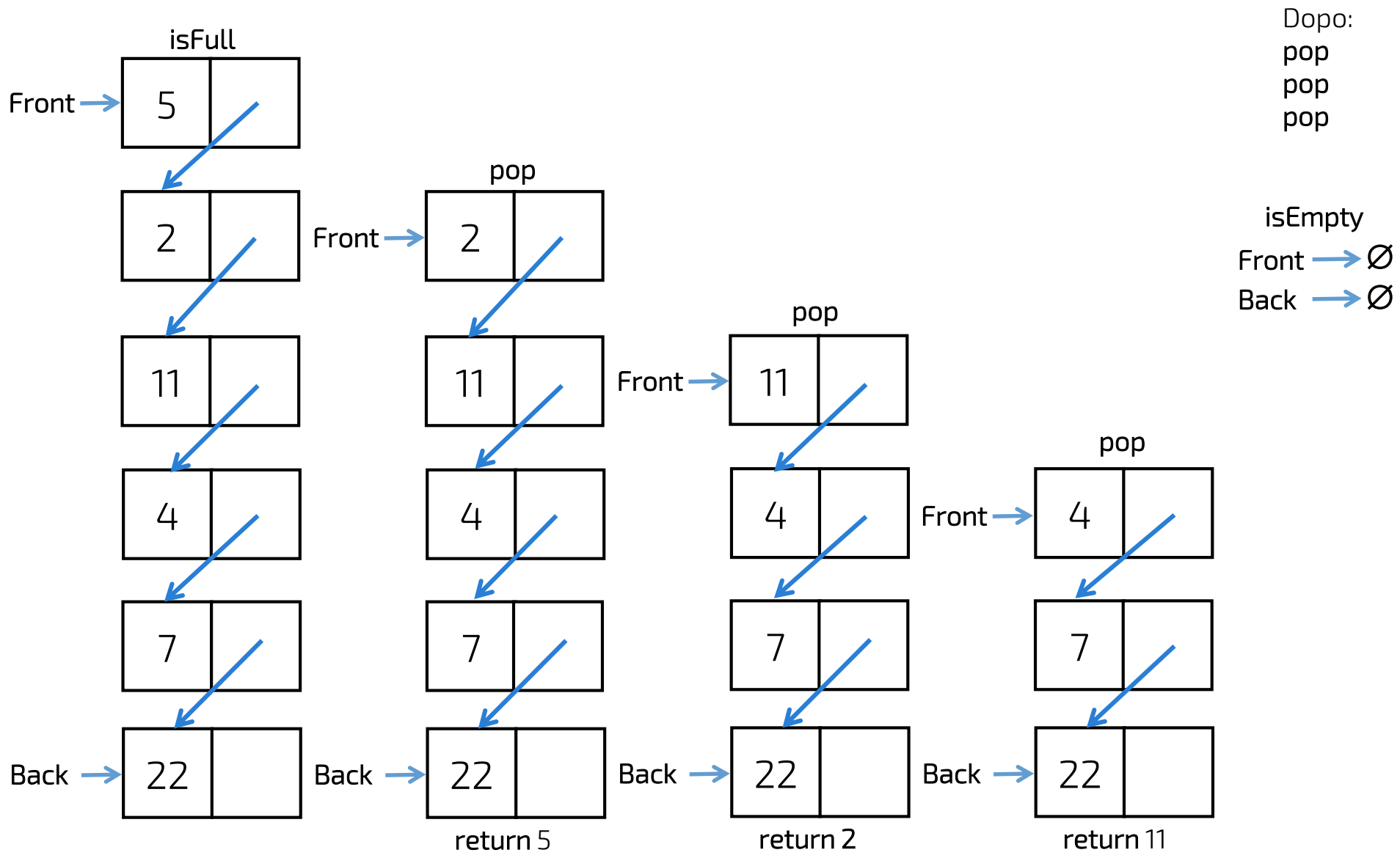
Lezione 5: Coda

- La Coda (Queue) è un esempio di struttura dati astratta *lineare*.
- È una struttura dati **FIFO** (**F**irst **I**n **F**irst **O**ut): il primo elemento inserito è anche il primo a uscire.
- Può essere implementato come **array** o **lista concatenata**.
- Deve prevedere almeno due funzioni:
  - *push*, per inserire un elemento in fondo alla coda;
  - *pop*, per eliminare un elemento dalla cima (testa) della coda e restituirlo.
- In aggiunta, sono utili le seguenti funzioni:
  - *create;*
  - *isEmpty;*
  - *isFull;*
  - *getNumElements;*
  - *getMaxSize;*
  - *...*

Lezione 5: inserimento in Coda (lista)



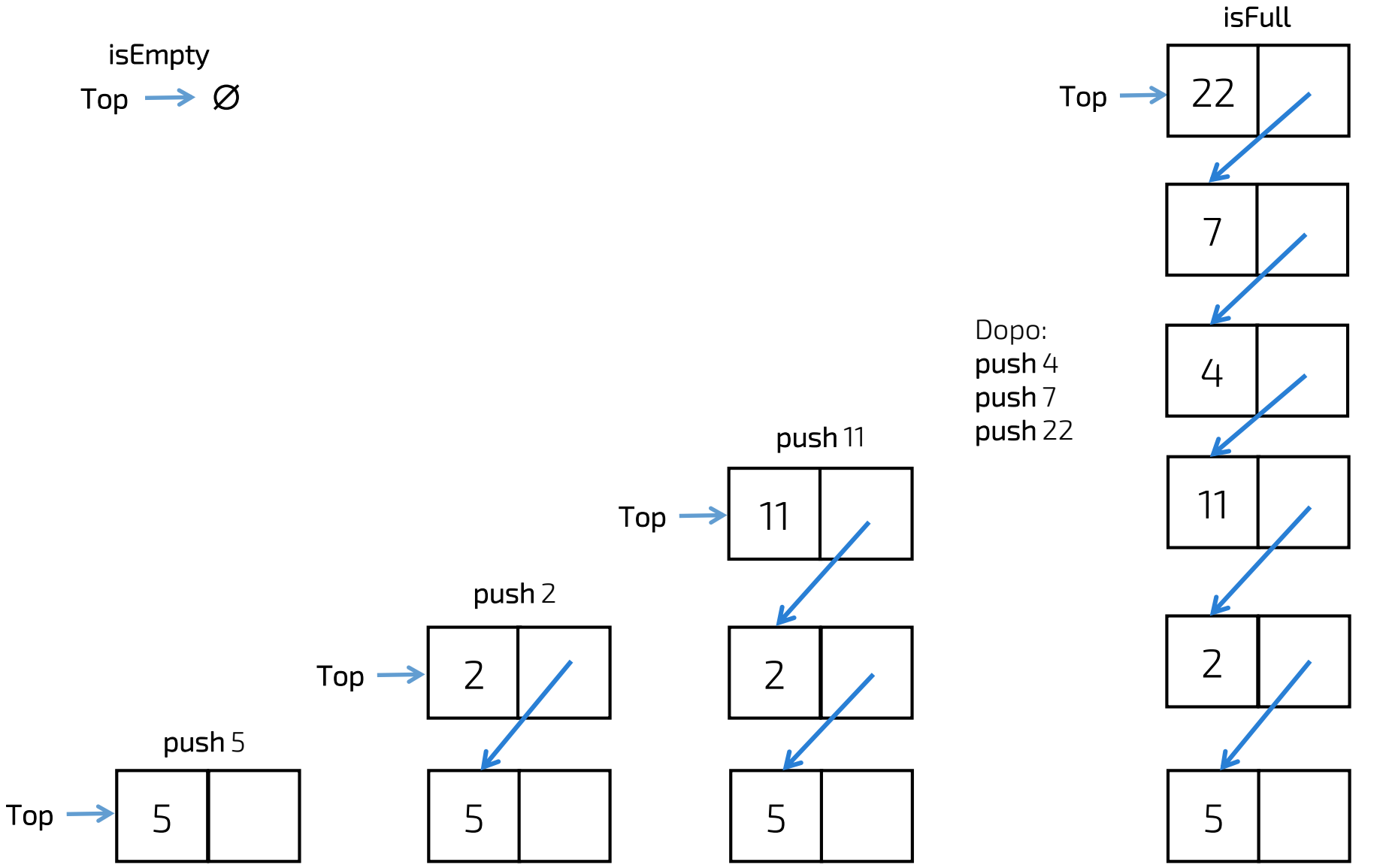
Lezione 5: eliminazione dalla Coda (lista)



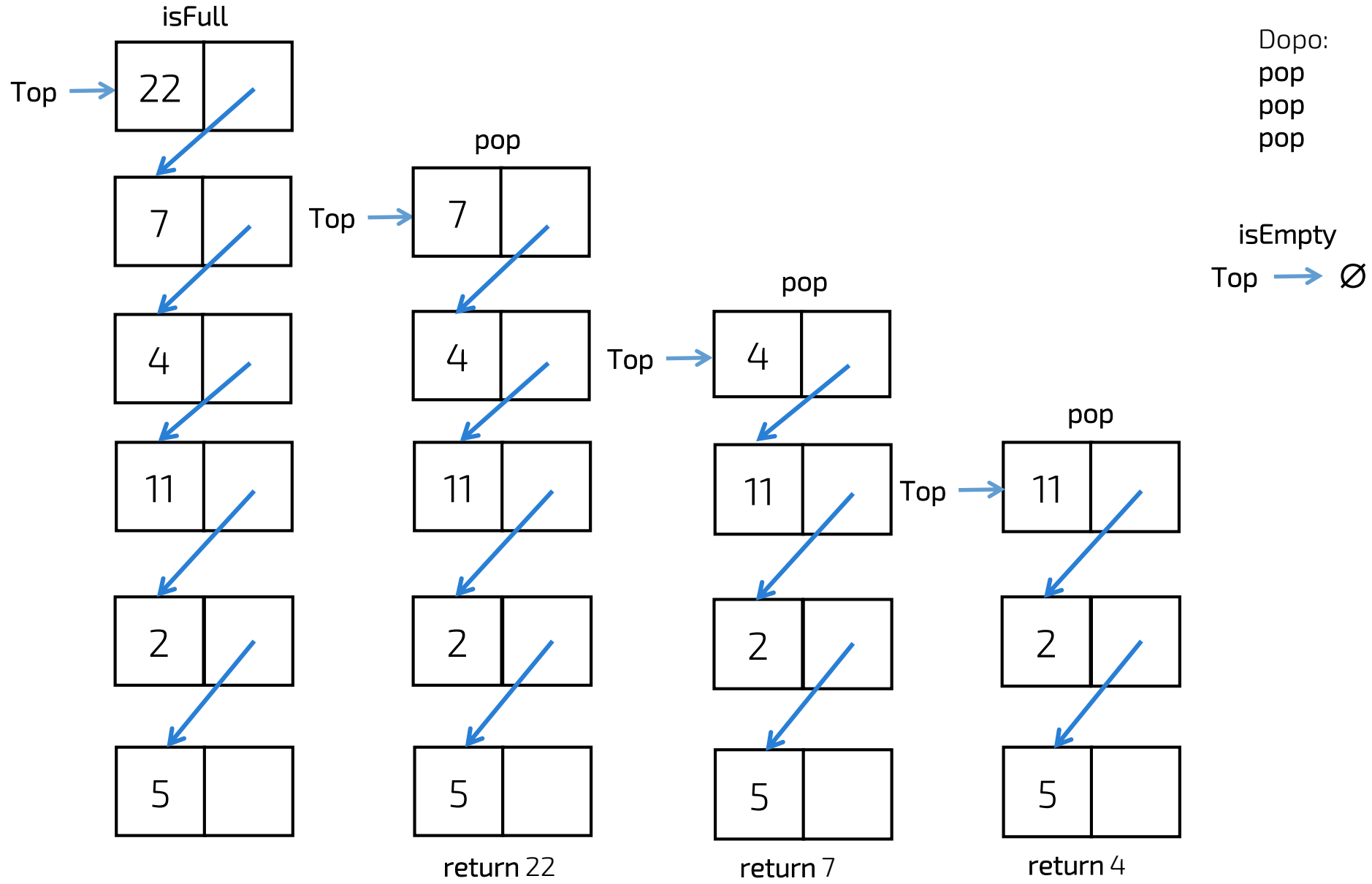
Lezione 5: Stack

- Lo Stack è un esempio di struttura dati astratta *lineare*.
- E' una struttura dati **LIFO** (**L**ast **I**n **F**irst **O**ut): l'ultimo elemento inserito sarà anche il primo a uscire.
- Può essere implementato come **array** o **lista concatenata**.
- Deve prevedere almeno due funzioni:
  - *push* per inserire un elemento in cima allo Stack;
  - *pop* per eliminare un elemento dalla cima dello Stack e restituirlo.
- In aggiunta, sono utili le seguenti funzioni:
  - *create;*
  - *isEmpty;*
  - *isFull;*
  - *getNumElements;*
  - *getMaxSize;*
  - *...*

Lezione 5: inserimento in Stack (lista)



Lezione 5: eliminazione dallo Stack (lista)



Esercizio 5: Progetto base

- Andate su eLearning e copiate il codice del progetto base per l'esercitazione odierna.
- Create un nuovo progetto su Clion e incollateci il codice.
- Si tratta di codice C che contiene la dichiarazione e l'inizializzazione di diverse strutture che utilizzeremo per la creazione di uno stack e di una coda:
  - Node
  - Queue
  - Stack
  - Ordine

- NON modificate queste strutture!
- Aggiungere solo le funzioni necessarie per la loro gestione (come spiegato nelle slide successive).

Esercizio 5: Problema

- Vogliamo simulare la gestione e la consegna di una serie di ordini su Amazon.
- Ogni ordine è rappresentato da una struttura (già presente nel progetto base) così definita:

```
typedef struct
{
    char articolo[DIM_STRING]
    double prezzo;
    char destinatario[DIM_STRING]
} Ordine;
```

Esercizio 5: Problema

- Ogni volta che un utente effettua un nuovo ordine l'articolo relativo viene **inserito in una coda.**
- Amazon processa gli articoli presenti in coda (in ordine di arrivo) prendendoli quindi dalla testa.

Lavatrice
450
William

Smart TV
800
Sonia

Frigorifero
1200
Paoletto

Collare
30
Lilli

Esercizio 5: Problema

- Man mano che gli articoli vengono processati vengono inseriti nel camion per la consegna. **La consegna viene gestita come uno stack.**

Esercizio 5_1: Coda

- Definiamo una **struttura nodo** ed un nuovo tipo **Nodo** per creare gli elementi della Coda:

```
struct node
{
    Type data;
    struct node* link;
};

typedef struct node Node;
```

- Definiamo una **struttura Queue** che rappresenta la nostra coda (NB: per modificare la coda, o per evitare di crearne una copia ogni volta, ne passiamo **sempre** il suo puntatore).

```
typedef struct
{
    Node* front;
    Node* back;
    int cont;
} Queue;
```

Esercizio 5_1: Coda

Implementare la struttura dati Coda mediante liste singolarmente concatenate per la gestione di un insieme di ordini su Amazon.

```
algoritmo isEmptyQueue (puntatore struttura queue) → Boolean

if (coda vuota) then
    return true
else
    return false
```

```
algoritmo isFullQueue (puntatore struttura queue) → Boolean

if (coda piena) then
    return true
else
    return false
```

Esercizio 5_1: Coda

```
algoritmo pushQueue (puntatore struttura queue, Type val)


if (isFullQueue(queue)) then
    stampa “errore overflow”

else
    alloca new_node
    inserisci val in data e NULL in link

    if (isEmptyQueue(queue)) then
        front punta a new_node
        back  punta a new_node

    else
        link di back punta a new_node
        back punta a new_node

    incrementa il conteggio degli elementi
```

Esercizio 5_1: Coda

```
algoritmo popQueue (puntatore struttura queue) → Type


if (isEmptyQueue(queue)) then
    stampa “errore underflow”
    val ← -1    //val è una variabile temporanea di tipo Type

else
    temp punta a front   //temp è un puntatore temporaneo
    val ← data di front

    if (coda ha un solo elemento) then
        metti a NULL front e back

    else
        front punta a link di front

    decrementa il conteggio degli elementi
    dealloca l’elemento puntato da temp

return val
```

Esercizio 5_2: Stack

- Definiamo una **struttura nodo** ed un nuovo tipo **Nodo** per creare gli elementi dello Stack (la struttura nodo è la stessa che abbiamo usato per la coda):

```
struct node
{
    Type data;
    struct node* link;
};

typedef struct node Node;
```

- Definiamo una **struttura Stack** che rappresenta in nostro stack (NB: per modificare lo stack, o per evitare di crearne una copia ogni volta, ne passiamo **sempre** il suo puntatore).

```
typedef struct
{
    Node* top;
    int cont;
} Stack;
```

Esercizio 5_2: Stack

Implementare la struttura dati Stack mediante liste singolarmente concatenate per la gestione di un insieme di ordini da consegnare.

```
algoritmo isEmptyStack (puntatore struttura stack) → Booleano

if (stack vuoto) then
    return true
else
    return false
```

```
algoritmo isFullStack (puntatore struttura stack) → Booleano

if (stack pieno) then
    return true
else
    return false
```

Esercizio 5_2: Stack

```
algoritmo pushStack (puntatore struttura stack, Type val)

if (isFullStack(stack)) then
    stampa "errore overflow"

else
    alloca new_node
    inserisci val in data e NULL in link

    if (isEmptyStack(stack)) then
        top punta a new_node

    else
        link di new_node punta a top
        top punta a new_node

    incrementa il conteggio degli elementi
```

Esercizio 5_2: Stack

```
algoritmo popStack (puntatore struttura stack) → Type

if (isEmptyStack(stack)) then
    stampa "errore underflow"
    val ← -1    //val è una variabile temporanea di tipo Type

else
    temp punta a top     //temp è un puntatore temporaneo
    val ← data di top

    top punta a link di top
    decrementa il conteggio degli elementi
    dealloca l'elemento puntato da temp

return val
```

Esercizio 5_3: Menu e visualizzazione

Prevedere un menu che consenta all'utente di effettuare le seguenti operazioni:

- Inserire un nuovo ordine in coda (**pushQueue**)
- Togliere un ordine dalla coda per processarlo (**popQueue**)
- Inserire un nuovo ordine nello stack per il trasporto (**pushStack**)
- Togliere un ordine dallo stack per consegnarlo (**popStack**)
- Processare il primo ordine in coda e automaticamente prepararlo per la consegna (**popQueue** seguito da **pushStack**)
- Stampare lo stato della coda degli ordini (**printQueue**)
- Stampare lo stato dello stack delle consegne (**printStack**)

Esercizio 5: Ricapitolando

- Es 5_1: Implementare la struttura dati Coda (per gestire degli ordini) e le relative funzioni (**isEmptyQueue**, **isFullQueue**, **pushQueue**, **popQueue**)
- Es 5_2: Implementare la struttura dati Stack (per gestire degli ordini da consegnare) e le relative funzioni (**isEmptyStack**, **isFullStack**, **pushStack**, **popStack**)
- Es 5_3: realizzare il menù con le operazioni richieste nella slide precedente.

end().