

Corso “Programmazione 1”

Capitolo 12: Strutture Dati Astratte

Docente: **Marco Roveri** - `marco.roveri@unitn.it`
Esercitori: **Giovanni De Toni** - `giovanni.detoni@unitn.it`
Stefano Berlato - `stefano.berlato-1@unitn.it`
C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: <https://bit.ly/2VgfYwJ>



Ultimo aggiornamento: 24 novembre 2021

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2021-2022.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Esempio di uso di Pile: Calcolatrice RPN

- Il metodo di calcolo **Reverse Polish Notation (RPN)** funziona postponendo l'operatore ai due operandi

$34 * 3 \implies 34 \ 3 \ *$

- Permette di effettuare complicate concatenazioni di conti senza usare parentesi:

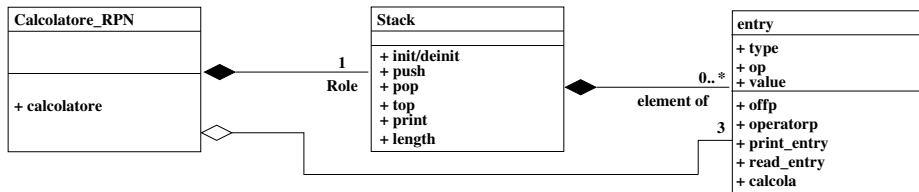
$(34 * 3) / (31 - 5) + (21+3) / (24-12)$

\implies

$34 \ 3 \ * \ 31 \ 5 \ - \ / \ 21 \ 3 \ + \ 24 \ 12 \ - \ / \ +$

- Una calcolatrice RPN funziona nel modo seguente:
 - se viene immesso un operando, si inserisce in uno stack di operandi
 - se viene immesso un operatore (binario) op :
 1. vengono prelevati dallo stack gli ultimi due operandi $op1$ e $op2$
 2. viene applicato l'operatore op a $op2$ e $op1$
 3. il risultato viene ri-immesso nello stack
 - l'ultimo elemento nello stack contiene il risultato dell'espressione

Implementazione della Calcolatrice RPN



Gestione delle entry:

{ CALC_RPN/entry.h
CALC_RPN/entry.cc }

Pila di entry:

{ CALC_RPN/stack.h
CALC_RPN/stack.cc }

Calcolatore RPN:

{ CALC_RPN/calcolatore_rpn.h
CALC_RPN/calcolatore_rpn.cc }

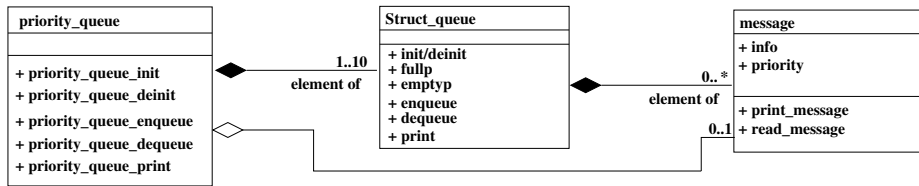
Main:

{ CALC_RPN/main.cc }

Esempio di uso di Code: Coda a priorità

- Una **coda a priorità** di messaggi è una struttura dati in cui
 - ogni messaggio arrivato ha una priorità in $[0..10]$
 - i messaggi vengono estratti in ordine di priorità, $0 \implies 1 \implies 2 \dots$
 - a parità di priorità vengono estratti in modo FIFO
- Realizzabile con un **array di code**, una per ogni livello di priorità:
 - un messaggio di priorità i viene inserito nella coda i -esima
 - l'estrazione avviene a partire dalla coda 0-sima: se vuota si passa alla successiva, ecc.
- Esempio: l'accettazione al Pronto Soccorso di un ospedale

Implementazione della Coda a Priorità



Gestione delle entità “message”:

```
{ CODA_PRIORITA/message.h }
{ CODA_PRIORITA/message.cc }
```

Coda di (puntatori a) messaggi:

```
{ CODA_PRIORITA/struct_queue.h }
{ CODA_PRIORITA/struct_queue.cc }
```

Coda a priorità di (puntatori a) messaggi:

```
{ CODA_PRIORITA/prio_queue.h }
{ CODA_PRIORITA/prio_queue.cc }
```

Main:

```
{ CODA_PRIORITA/prio_queue_main.cc }
```